Java SE 6 类库查询手册

龙马工作室搜集整理制作

索引

java.applet

java.lang

java.io

java.awt

java.applet

提供创建 applet 所必需的类和 applet 用来与其 applet 上下文通信的类。

接口

AppletContext 接口

此接口对应于 applet 的环境：包含 applet 的文档以及同一文档中的其他 applet。applet 可以使用此接口中的方法获取有关其环境的信息。

方法详细信息

getAudioClip

```
AudioClip getAudioClip(URL url)
```

创建音频剪辑。

参数：

url - 给出音频剪辑位置的绝对 URL。

返回：

指定 URL 处的音频剪辑。

---

getImage

Image getImage(URL url)

返回能被绘制到屏幕上的 Image 对象。作为参数传递的 url 必须指定绝对 URL。

不管图像存在与否，此方法总是立即返回。当此 applet 试图在屏幕上绘制图像时，数据将被加载。绘制图像的图形图元将逐渐绘制到屏幕上。

参数：

url - 给出图像位置的绝对 URL。

返回：

指定 URL 处的图像。

另请参见：

Image

---

getApplet

Applet getApplet(String name)

通过给定名称找到并返回此 applet 上下文表示的文档中的 applet。名称可以在 HTML 标记中通过设置 name 属性来设置。

参数：

name - applet 名称。

返回：

具有给定名称的 applet；如果未找到，则返回 null。

---

getApplets

Enumeration<Applet> getApplets()

找到此 applet 上下文表示的文档中的所有 applet。

返回：

此 applet 上下文表示的文档中所有 applet 的枚举。

---

showDocument

void showDocument(URL url)

请求浏览器或 applet viewer 显示 url 参数指示的 Web 页面。浏览器或 applet viewer 确定使用哪个窗口或窗体来显示 Web 页面。非浏览器的 applet 上下文可以忽略此方法。

参数：

url - 给出文档位置的绝对 URL。

showDocument

void showDocument(URL url,
String target)

请求浏览器或 applet viewer 显示由 url 参数所指示的 Web 页。target 参数指示在哪个 HTML 窗体中显示文档。target 参数的解释如下:

Target 参数	描述
"_self"	在包含 applet 的窗口和窗体中显示。
"_parent"	在 applet 的父窗体中显示。如果 applet 的窗体没有父窗体，执行与 "_self" 相同的动作。
"_top"	在 applet 窗口的顶层窗体中显示。如果 applet 窗体是顶层窗体，执行与 "_self" 相同的动作。
"_blank"	在一个新的、未命名的顶层窗口中显示。
name	在名为 *name* 的窗体或窗口中显示。如果名为 *name* 的 target 尚未存在，将创建一个具有指定名称的新顶层窗口，文档将在该窗口中显示。

applet viewer 或浏览器可以随意地忽略 showDocument。

参数：

url - 给定文档位置的绝对 URL。

target - 指示在哪里显示页面的 String。

showStatus

void showStatus(String status)

请求参数字符串显示在“状态窗口”中。很多浏览器和 applet viewer 提供这种窗口，应用程序可以通过该窗口将其当前状态通知给用户。

参数：

status - 在状态窗口中显示的字符串。

---

setStream

```
void setStream(String key,
               InputStream stream)
               throws IOException
```

在此 applet 上下文中用指定的键关联指定的流。如果 applet 上下文以前包含了一个此键的映射关系，那么将替换旧值。

出于安全性考虑，每个代码基都存在流和键的映射。换句话说，一个代码基中的 applet 不能访问不同代码基中的 applet 创建的流。

参数：

key - 指定值要关联的键。

stream - 指定键要关联的流。如果此参数为 null，则在此 applet 上下文中将移除指定键。

抛出：

IOException - 如果流大小超过了一定大小限制。大小限制由此接口的实现者确定。

IOException

---

getStream

```
InputStream getStream(String key)
```

返回此 applet 上下文中指定键所关联的流。如果 applet 上下文中不包含此键对应的流，则返回 null。

由于安全性考虑，每个代码基都存在流和键的映射。换句话说，一个代码基中的 applet 不能访问不同代码基中的 applet 创建的流。

参数：

key - 要返回其关联流的键。

返回：

此 applet 上下文将键映射到的流

getStreamKeys

```
Iterator<String> getStreamKeys()
```

找到此 applet 上下文中所有流对应的键。

由于安全性考虑，每个代码基都存在流和键的映射。换句话说，一个代码基中的 applet 不能访问不同代码基中的 applet 创建的流。

返回：

返回此 applet 上下文中所有流名称的迭代器。

AppletStub 接口

当 applet 第一次被创建时，使用 applet 的 setStub 方法把 applet stub 连接到它。此 stub 充当 applet 和浏览器环境或 applet viewer 环境之间的接口，应用程序在此环境中运行。

方法详细信息

isActive

```
boolean isActive()
```

确定 applet 是否处于激活状态。仅在 applet 的 `start` 方法被调用前，applet 才处于激活状态。仅在 applet 的 `stop` 方法被调用前，applet 才变成非激活状态。

返回：

如果 applet 处于激活状态，则返回 `true`；否则返回 `false`。

getDocumentBase

URL getDocumentBase()

获取嵌入 applet 的文档的 URL。例如，假定 applet 包含在下面的文档中：

http://java.sun.com/products/jdk/1.2/index.html

则该文档基于：

http://java.sun.com/products/jdk/1.2/index.html

返回：

包含 applet 的文档的 URL。

另请参见：

getCodeBase()

---

getCodeBase

URL getCodeBase()

获取基 URL。这是包含 applet 的目录的 URL。

返回：

包含 applet 的目录的基 URL。

另请参见：

getDocumentBase()

---

getParameter

String getParameter(String name)

返回 HTML 标记中命名参数的值。例如，如果 applet 被指定为：

```
<applet code="Clock" width=50 height=50>
<param name=Color value="blue">
</applet>
```

则对 getParameter("Color") 的调用将返回值 "blue"。

参数：

name - 参数名。

返回：

命名参数的值，如果没有设置，则为 null。

getAppletContext

```
AppletContext getAppletContext()
```

返回 applet 的上下文。

返回：

applet 上下文。

appletResize

```
void appletResize(int width,
                  int height)
```

当 applet 想要重新调整大小时调用。

参数：

width - 为 applet 新请求的宽度。

height - 为 applet 新请求的高度。

AudioClip 接口

AudioClip 接口是用于播放音频剪辑的简单抽象。多个 AudioClip 项能够同时播放，得到的声音混合在一起可产生合成声音。

方法详细信息

Play

```
void play()
```

开始播放此音频剪辑。每次调用此方法时，剪辑都从头开始重新播放。

loop

```
void loop()
```

以循环方式开始播放此音频剪辑。

stop

```
void stop()
```

停止播放此音频剪辑。

类

Applet 类

applet 是一种不能单独运行但可嵌入在其他应用程序中的小程序。

Applet 类必须是任何嵌入 Web 页或可用 Java Applet Viewer 查看的 applet 的超类。Applet 类提供了 applet 及其运行环境之间的标准接口。

构造方法详细信息

Applet

```
public Applet()
        throws HeadlessException
```

构造一个新 Applet。

注：java.applet.Applet 中的许多方法只有在完全构造了 applet 之后才能由该 applet 调用；在构造方法中，applet 应该避免调用 java.applet.Applet 中的方法。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

方法详细信息

setStub

```
public final void setStub(AppletStub stub)
```

设置此 applet 的 stub。此操作可以通过系统自动完成。

如果存在安全管理器并且设置了 stub，则通过 AWTPermission("setAppletStub") 权限调用它的 checkPermission 方法。

参数：

stub - 新 stub。

抛出：

SecurityException - 如果调用者不能设置 stub

---

isActive

```
public boolean isActive()
```

确定 applet 是否处于活动状态。在调用 applet 的 start 方法之前，applet 被标记为活动状态。在调用 applet 的 stop 方法之前，applet 被标记为非活动状态。

返回：

如果 applet 处于活动状态，则返回 true；否则返回 false。

另请参见：

start(), stop()

---

getDocumentBase

```
public URL getDocumentBase()
```

获取嵌入此 applet 的文档的 URL。例如，假定 applet 包含在以下文档中:

```
http://java.sun.com/products/jdk/1.2/index.html
```

则文档基为:

```
http://java.sun.com/products/jdk/1.2/index.html
```

返回:

包含此 applet 的文档的 URL。

另请参见:

getCodeBase()

---

getCodeBase

```
public URL getCodeBase()
```

获得基 URL。这是包含此 applet 的目录的 URL。

返回:

包含此 applet 的目录的基 URL。

另请参见:

getDocumentBase()

---

getParameter

```
public String getParameter(String name)
返回 HTML 标记中指定参数的值。例如，如果此 applet 被指定为:
 <applet code="Clock" width=50 height=50>
 <param name=Color value="blue">
 </applet>
```

那么调用 getParameter("Color") 将返回值 "blue"。

name 参数不区分大小写。

参数:

name - 参数名称。
返回：
指定参数的值；如果未设置，则返回 null。

getAppletContext

```
public AppletContext getAppletContext()
```

确定此 applet 的上下文，上下文允许 applet 查询和影响它所运行的环境。

applet 的环境指包含该 applet 的文档。

返回：
applet 的上下文。

resize

```
public void resize(int width,
                   int height)
```

请求调整此 applet 的大小。
覆盖：
类 Component 中的 resize
参数：
width - 为 applet 请求的新宽度。
height - 为 applet 请求的新高度。

resize

```
public void resize(Dimension d)
```

请求调整此 applet 的大小。
覆盖：
类 Component 中的 resize
参数：

d - 给出新宽度和高度的对象。

showStatus

```
public void showStatus(String msg)
```

请求将参数字符串显示在“状态窗口”中。很多浏览器和 applet viewer 提供这种窗口，应用程序可以通过该窗口将其当前状态通知给用户。

参数：

msg - 在状态窗口中显示的字符串。

getImage

```
public Image getImage(URL url)
```

返回能被绘制到屏幕上的 Image 对象。作为参数传递的 url 必须指定绝对 URL。

不管图像存在与否，此方法总是立刻返回。当此 applet 试图在屏幕上绘制图像时，数据将被加载。绘制图像的图形图元将逐渐绘制到屏幕上。

参数：

url - 给出图像位置的绝对 URL。

返回：

指定 URL 处的图像。

另请参见：

Image

getImage

```
public Image getImage(URL url,
                      String name)
```

返回能被绘制到屏幕上的 Image 对象。url 参数必须指定绝对 URL。name 参数是相对于 url 参数的说明符。

不管图像存在与否，此方法总是立刻返回。当此 applet 试图在屏幕上绘制图像时，数据将被加载。绘制图像的图形图元将逐渐绘制到屏幕上。

参数：
url - 给出图像基本位置的绝对 URL。
name - 相对于 url 参数的图象位置。
返回：
指定 URL 处的图像。
另请参见：
Image

---

newAudioClip

```
public static final AudioClip newAudioClip(URL url)
```

从给定 URL 处获取音频剪辑。
参数：
url - 指向音频剪辑
返回：
指定 URL 处的音频剪辑。

---

getAudioClip

```
public AudioClip getAudioClip(URL url)
```

返回 URL 参数指定的 AudioClip 对象。

不管音频剪辑存在与否，此方法总是立即返回。当此 applet 试图播放音频剪辑时，数据将被加载。

参数：
url - 给出音频剪辑位置的绝对 URL。
返回：
指定 URL 处的音频剪辑。
另请参见：
AudioClip

---

getAudioClip

```
public AudioClip getAudioClip(URL url,
                              String name)
```

返回 URL 和 name 参数指定的 AudioClip 对象。

不管音频剪辑存在与否，此方法总是立即返回。当此 applet 试图播放音频剪辑时，数据将被加载。

参数：
url - 给定音频剪辑基本位置的绝对 URL。
name - 相对于 url 参数的音频剪辑位置。
返回：
指定 URL 处的音频剪辑。
另请参见：
AudioClip

---

getAppletInfo

```
public String getAppletInfo()
```

返回有关此 applet 的信息。applet 应该重写此方法，返回包含有关 applet 的作者、版本和版权信息的 String。

Applet 类提供的此方法实现返回 null。

返回：
包含有关 applet 的作者、版本和版权信息的字符串。

---

getLocale

```
public Locale getLocale()
```

获取 applet 的语言环境。该方法允许 applet 维护自己的语言环境，该环境与浏览器或 appletviewer 的语言环境是分离的。
覆盖：

类 Component 中的 getLocale

返回：

applet 的语言环境；如果尚未设置该语言环境，则返回默认语言环境。

从以下版本开始：

JDK1.1

另请参见：

```
Component.setLocale(java.util.Locale)
```

getParameterInfo

```
public String[][] getParameterInfo()
```

返回此 applet 理解的关于参数的信息。applet 应该重写此方法，返回描述这些参数的 String 数组。

数组的每个元素应该是三个 String 的集合，包括名称、类型和描述。例如：

```
String pinfo[][] = {
        {"fps",      "1-10",      "frames per second"},
        {"repeat", "boolean", "repeat image loop"},
        {"imgs",     "url",        "images directory"}
};
```

Applet 类提供的此方法实现返回 null。

返回：

描述此 applet 所寻找的参数的数组。

play

```
public void play(URL url)
```

播放指定绝对 URL 处的音频剪辑。如果未找到音频剪辑，则没有任何效果。

参数：

url - 给出音频剪辑位置的绝对 URL。

play

```
public void play(URL url,
                 String name)
```

播放音频剪辑，给定了 URL 及与之相对的说明符。如果未找到音频剪辑，则没有任何效果。

参数：

url - 给定音频剪辑基位置的绝对 URL。

name - 相对于 url 参数的音频剪辑位置。

---

init

```
public void init()
```

由浏览器或 applet viewer 调用，通知此 applet 它已经被加载到系统中。它经常在第一次调用 start 方法前被调用。

如果 Applet 的子类要执行初始化，则应该重写此方法。例如，使用线程的 applet 将用 init 方法创建线程，用 destroy 方法销毁它们。

Applet 类提供的此方法实现不执行任何操作。

另请参见：

destroy(), start(), stop()

---

start

```
public void start()
```

由浏览器或 applet viewer 调用，通知此 applet 它应该开始执行。它在 init 方法调用后以及在 Web 页中每次重新访问 applet 时调用。

如果 Applet 子类在包含它的 Web 页被访问时有想要执行的操作，则它应该重写此方法。例如，带有动画的 applet 可能想使用 start 方法再次播放动画，使用 stop 方法挂起动画。

注：某些方法（如 getLocationOnScreen）只有在如果 applet 正在显示时，才能提供有意义的结果。因为当 applet 的 start 方法第一次被调用时，isShowing 返回 false，所以需要 isShowing 返回 true 的方法应该从 ComponentListener 进行调用。

Applet 类提供的此方法实现不执行任何操作。

另请参见：

destroy(), init(), stop(), Component.isShowing(), ComponentListener.componentShown(java.awt.event.ComponentEvent)

stop

```
public void stop()
```

由浏览器或 applet viewer 调用，通知此 applet 它应该终止执行。当包含此 applet 的 Web 页已经被其他页替换时，在 applet 被销毁前调用此方法。

如果 Applet 子类在包含它的 Web 页每次不可见时有想要执行的操作，则它应该重写此方法。例如，带有动画的 applet 可能想使用 start 方法再次播放动画，使用 stop 方法挂起动画。

Applet 类提供的此方法实现不执行任何操作。

另请参见：

destroy(), init()

destroy

```
public void destroy()
```

由浏览器或 applet viewer 调用，通知此 applet 它正在被回收，它应该销毁分配给它的任何资源。stop 方法总是在 destroy 之前被调用。

如果 Applet 子类在被销毁前有想要执行的操作，则它应该重写此方法。例如，使用线程的 applet 将用 init 方法来创建线程，用 destroy 方法销毁它们。

Applet 类提供的此方法实现不执行任何操作。

另请参见：
init(), start(), stop()

getAccessibleContext

public AccessibleContext getAccessibleContext()

获取与此 Applet 关联的 AccessibleContext。对于 applet，AccessibleContext 采用 AccessibleApplet 的形式。如有必要，可创建一个新的 AccessibleApplet 实例。

指定者：
接口 Accessible 中的 getAccessibleContext
覆盖：
类 Panel 中的 getAccessibleContext
返回：
一个 AccessibleApplet，它充当此 Applet 的 AccessibleContext
……

java.lang

提供利用 Java 编程语言进行程序设计的基础类。

接口

Appendable 接口

能够被添加 char 序列和值的对象。如果某个类的实例打算接收取自 Formatter 的格式化输出，那么该类必须实现 Appendable 接口。

要添加的字符应该是有效的 Unicode 字符，正如 Unicode Character Representation 中描述的那样。注意，增补字符可能由多个 16 位 char 值组成。

Appendable 对于多线程访问而言没必要是安全的。线程安全由扩展和实现此接口的类负责。

由于此接口可能由具有不同的错误处理风格的现有类实现，所以无法保证错误不会传播给调用者。

方法详细信息

append

```
Appendable append(CharSequence csq)
                  throws IOException
```

向此 Appendable 添加指定的字符序列。

有时可能没有添加整个序列，这取决于使用哪个类来实现字符序列 csq。例如，如果 csq 是 CharBuffer 的一个实例，则通过缓冲区的位置和限制来定义要添加的子序列。

参数：

csq - 要添加的字符串序列。如果 csq 为 null，则向该 Appendable 添加四个字符 "null"。

返回：

此 Appendable 的引用

抛出：

IOException - 如果发生 I/O 错误

append

```
Appendable append(CharSequence csq,
                  int start,
                  int end)
                  throws IOException
```

向此 Appendable 添加指定字符序列的子序列。

当 csq 不为 null 时，用 out.append(csq, start, end) 的形式调用此方法与用以下形式调用此方法的行为完全相同：

```
out.append(csq.subSequence(start, end))
```

参数：

csq - 子序列将被添加的字符序列。如果 csq 为 null，则添加四个字符 "null"，就好像 csq 包含这些字符一样。

start - 子序列中第一个字符的索引

end - 紧随子序列中最后一个字符的字符的索引

返回：

此 Appendable 的引用

抛出：

IndexOutOfBoundsException - 如果 start 或 end 为负，以及 start 大于 end 或者 end 大于 csq.length()

IOException - 如果发生 I/O 错误

append

Appendable append(char c)
throws IOException

向此 Appendable 添加指定字符。

参数：

c - 要添加的字符

返回：

此 Appendable 的引用

抛出：

IOException - 如果发生 I/O 错误

runnable 接口

Runnable 接口应该由那些打算通过某一线程执行其实例的类来实现。类必须定义一个称为 run 的无参数方法。

设计该接口的目的是为希望在活动时执行代码的对象提供一个公共协议。例如，Thread 类实现了 Runnable。激活的意思是说某个线程已启动并且尚未停止。

此外，Runnable 为非 Thread 子类的类提供了一种激活方式。通过实例化某个 Thread 实例并将自身作为运行目标，就可以运行实现 Runnable 的类而无需创建 Thread 的子类。大多数情况下，如果只想重写 run() 方法，而不重写其他 Thread 方法，那么应使用 Runnable 接口。这很重要，因为除非程序员打算修改或增强类的基本行为，否则不应为该类创建子类。

方法详细信息

run

void run()

使用实现接口 Runnable 的对象创建一个线程时，启动该线程将导致在独立执行的线程中调用对象的 run 方法。

方法 run 的常规协定是，它可能执行任何所需的动作。

另请参见：
Thread.run()

Cloneable 接口

此类实现了 Cloneable 接口，以指示 Object.clone() 方法可以合法地对该类实例进行按字段复制。

如果在没有实现 Cloneable 接口的实例上调用 Object 的 clone 方法，则会导致抛出 CloneNotSupportedException 异常。

按照惯例，实现此接口的类应该使用公共方法重写 Object.clone（它是受保护的）。请参阅 Object.clone()，以获得有关重写此方法的详细信息。

注意，此接口不 包含 clone 方法。因此，因为某个对象实现了此接口就克隆它是不可能的。即使 clone 方法是反射性调用的，也无法保证它将获得成功。

类

Boolean 类

Boolean 类将基本类型为 boolean 的值包装在一个对象中。一个 Boolean 类型的对象只包含一个类型为 boolean 的字段。

此外，此类还为 boolean 和 String 的相互转换提供了许多方法，并提供了处理 boolean 时非常有用的其他一些常量和方法。

方法详细信息

parseBoolean

```
public static boolean parseBoolean(String s)
```

将字符串参数解析为 boolean 值。如果 String 参数不是 null 且在忽略大小写时等

于 "true"，则返回的 boolean 表示 true 值。

示例：Boolean.parseBoolean("True") 返回 true。
示例：Boolean.parseBoolean("yes") 返回 false。

参数：

s - 包含要解析的布尔表达式的 String

返回：

String 参数所表示的布尔值

Byte 类

Byte 类将基本类型 byte 的值包装在一个对象中。一个 Byte 类型的对象只包含一个类型为 byte 的字段。

此外，该类还为 byte 和 String 的相互转换提供了几种方法，并提供了处理 byte 时非常有用的其他一些常量和方法。

构造方法详细信息

Byte

```
public Byte(byte value)
```

构造一个新分配的 Byte 对象，以表示指定的 byte 值。

参数：

value - Byte 对象所表示的值。

Byte

```
public Byte(String s)
        throws NumberFormatException
```

构造一个新分配的 Byte 对象，以表示 String 参数所指示的 byte 值。该字符串以使用基数 10 的 parseByte 方法所使用的方式被转换成一个 byte 值。

参数：

s - 要转换成 Byte 的 String

抛出：

NumberFormatException - 如果 String 不包含一个可解析的 byte。

另请参见：

parseByte(java.lang.String, int)

方法详细信息

toString

public static String toString(byte b)

返回表示指定 byte 的一个新 String 对象。假定基数为 10。

参数：

b - 要转换的 byte

返回：

指定 byte 的字符串表示形式

另请参见：

Integer.toString(int)

---

valueOf

public static Byte valueOf(byte b)

返回表示指定 byte 值的一个 Byte 实例。如果不需要新的 Byte 实例，则通常应优先使用此方法，而不是构造方法 Byte(byte)，因为该方法有可能通过缓存经常请求的值来显著提高空间和时间性能。

参数：

b - 一个 byte 值。

返回：

表示 b 的 Byte 实例。

---

parseByte

public static byte parseByte(String s)

```
throws NumberFormatException
```

将 string 参数解析为有符号的十进制 byte。除了第一个字符可以是表示负值的 ASCII 负号 '-' ('\u002D') 之外，该字符串中的字符必须都是十进制数字。返回得到的 byte 值与以该 string 参数和基数 10 为参数的 parseByte(java.lang.String, int) 方法所返回的值一样。

参数：

s - 要解析的包含 byte 表示形式的 String

返回：

以十进制的参数表示的 byte 值

抛出：

NumberFormatException - 如果该 string 不包含一个可解析的 byte。

parseByte

```
public static byte parseByte(String s,
                             int radix)
throws NumberFormatException
```

将 string 参数解析为一个有符号的 byte，其基数由第二个参数指定。除了第一个字符可以是表示负值的 ASCII 负号 '-' ('\u002D') 之外（这取决于 Character.digit(char, int) 是否返回非负值），该 string 中的字符必须都是指定基数的数字。返回得到的 byte 值。

如果出现下列任何一种情况，则抛出一个 NumberFormatException 类型的异常：

- 第一个参数为 null 或是一个长度为零的字符串。
- 基数小于 Character.MIN_RADIX 或者大于 Character.MAX_RADIX。
- 字符串的任一字符不是指定基数的数字，第一个字符是负号 '-' ('\u002D') 的情况除外（但此时字符串的长度应超过 1）。
- 字符串所表示的值不是 byte 类型的值。

参数：

s - 要解析的包含 byte 表示形式的 String

radix - 在解析 s 时使用的基数

返回：

以指定基数表示的 string 参数所表示的 byte 值

抛出：

NumberFormatException - 如果该 string 不包含一个可解析的 byte。

valueOf

```
public static Byte valueOf(String s,
                           int radix)
                    throws NumberFormatException
```

返回一个 Byte 对象，该对象保持从指定的 String 中提取的值，该值是在用第二个参数所给定的基数对指定字符串进行解析时提取的。第一个参数被解释为用第二个参数所指定的基数表示一个有符号的 byte，正如将该参数指定给 parseByte(java.lang.String, int) 方法一样。结果是一个表示该 string 所指定的 byte 值的 Byte 对象。

换句话说，该方法返回一个等于以下代码的值的 Byte 对象：

```
new Byte(Byte.parseByte(s, radix))
```

参数：

s - 要解析的字符串

radix - 在解释 s 时使用的基数

返回：

保持用指定基数表示的字符串参数所表示的值的 Byte 对象。

抛出：

NumberFormatException - 如果 String 不包含一个可解析的 byte。

valueOf

```
public static Byte valueOf(String s)
                    throws NumberFormatException
```

返回一个保持指定 String 所给出的值的 Byte 对象。参数被解释为表示一个有符号的十进制的 byte，正如将该参数指定给 parseByte(java.lang.String) 方法一样。结果是一个表示该 string 所指定的 byte 值的 Byte 对象。

换句话说，该方法返回一个等于以下代码的值的 Byte 对象：

```
new Byte(Byte.parseByte(s))
```

参数：
s - 要解析的字符串
返回：
保持 string 参数所表示的值的 Byte 对象
抛出：
NumberFormatException - 如果该 String 不包含一个可解析的 byte。

decode

```
public static Byte decode(String nm)
                          throws NumberFormatException
```

将 String 解码为 Byte。接受按下列语法给出的十进制、十六进制和八进制数：

```
DecodableString:
Signopt DecimalNumeral
Signopt 0x HexDigits
Signopt 0X HexDigits
Signopt # HexDigits
Signopt 0 OctalDigits
Sign:
-
```

Java Language Specification 的 §3.10.1 中给出了 *DecimalNumeral*、*HexDigits* 和 *OctalDigits* 的定义。

对（可选）负号和/或基数说明符（“0x”、“0X”、“#” 或前导零）后面的字符序列进行解析就如同使用带指定基数（10、16 或 8）的 Byte.parseByte 方法一样。该字符序列必须表示一个正值，否则将抛出 NumberFormatException。如果指定 String 的第一个字符是负号，则结果将被求反。该 String 中不允许出现空白字符。

参数：
nm - 要解码的 String。
返回：
保持由 nm 表示的 byte 值的 Byte 对象
抛出：
NumberFormatException - 如果 String 不包含一个可解析的 byte。
另请参见：

parseByte(java.lang.String, int)

byteValue

public byte byteValue()

作为一个 byte 返回此 Byte 的值。

覆盖：

类 Number 中的 byteValue

返回：

转换为 byte 类型后该对象表示的数值。

shortValue

public short shortValue()

作为一个 short 返回此 Byte 的值。

覆盖：

类 Number 中的 shortValue

返回：

转换为 short 类型后该对象表示的数值。

intValue

public int intValue()

作为一个 int 返回此 Byte 的值。

指定者：

类 Number 中的 intValue

返回：

转换为 int 类型后该对象表示的数值。

longValue

public long longValue()

作为一个 long 返回此 Byte 的值。

指定者：

类 Number 中的 longValue

返回：

转换为 long 类型后该对象表示的数值。

floatValue

public float floatValue()

作为一个 float 返回此 Byte 的值。

指定者：

类 Number 中的 floatValue

返回：

转换为 float 类型后该对象表示的数值。

doubleValue

public double doubleValue()

作为一个 double 返回此 Byte 的值。

指定者：

类 Number 中的 doubleValue

返回：

转换为 double 类型后该对象表示的数值。

toString

public String toString()

返回表示此 Byte 的值的 String 对象。该值被转换成有符号的十进制表示形式，并作为一个 string 返回，正如将 byte 值作为一个参数指定给 toString(byte) 方法所返回的一样。

覆盖：

类 Object 中的 toString

返回：

以基数 10 表示的此对象值的字符串表示形式。

hashCode

```
public int hashCode()
```

返回此 Byte 的哈希码。

覆盖：

类 Object 中的 hashCode

返回：

此对象的一个哈希码值。

另请参见：

Object.equals(java.lang.Object),Hashtable

equals

```
public boolean equals(Object obj)
```

将此对象与指定对象比较。当且仅当参数不为 null，而是一个与此对象一样包含相同 Byte 值的 byte 对象时，结果才为 true。

覆盖：

类 Object 中的 equals

参数：

obj - 要进行比较的对象

返回：

如果这些对象相同，则为 true；否则为 false。

另请参见：

Object.hashCode(),Hashtable

compareTo

```
public int compareTo(Byte anotherByte)
```

在数字上比较两个 Byte 对象。

指定者：

接口 Comparable<Byte> 中的 compareTo

参数：

anotherByte - 要比较的 Byte。

返回：

如果此 Byte 等于参数 Byte，则返回 0；如果此 Byte 在数字上小于参数 Byte，则返回小于 0 的值；如果此 Byte 在数字上大于参数 Byte，则返回大于 0 的值（有符号比较）。

Character 类

Character 类在对象中包装一个基本类型 char 的值。Character 类型的对象包含类型为 char 的单个字段。

此外，该类提供了几种方法，以确定字符的类别（小写字母，数字，等等），并将字符从大写转换成小写，反之亦然。

字符信息基于 Unicode 标准，版本 4.0。

Character 类的方法和数据是通过 UnicodeData 文件中的信息定义的，该文件是 Unicode Consortium 维护的 Unicode Character Database 的一部分。此文件指定了各种属性，其中包括每个已定义 Unicode 代码点或字符范围的名称和常规类别。

此文件及其描述可从 Unicode Consortium 获得，网址如下：

http://www.unicode.org

Unicode 字符表示形式char 数据类型(和 Character 对象封装的值)基于原始的 Unicode 规范，将字符定义为固定宽度的 16 位实体。Unicode 标准曾做过修改，以允许那些其表示形式需要超过 16 位的字符。合法代码点 的范围现在是从 U+0000 到 U+10FFFF，即通常所说的 Unicode 标量值。(请参阅 Unicode 标准中 U+n 表示法的定义。)

从 U+0000 到 U+FFFF 的字符集有时也称为 Basic Multilingual Plane (BMP)。代码点大于 U+FFFF 的字符称为增补字符。Java 2 平台在 char 数组以及 String 和 StringBuffer 类中使用 UTF-16 表示形式。在这种表现形式中，增补字符表示为一对 char 值，第一个值取自高代理项 范围，即 (\uD800-\uDBFF)，第二个值取自低代理项 范围，即 (\uDC00-\uDFFF)。

所以，char 值表示 Basic Multilingual Plane (BMP) 代码点，其中包括代理项代码点，或 UTF-16 编码的代码单元。int 值表示所有 Unicode 代码点，包括增补代码点。int 的 21 个低位（最低有效位）用于表示 Unicode 代码点，并且 11 个高位（最高有效位）必须为零。除非另有指定，否则与增补字符和代理项 char 值有关的行为如下：

只接受一个 char 值的方法无法支持增补字符。它们将代理项字符范围内的 char 值视为未定义字符。例如，Character.isLetter('\uD840') 返回 false，即使是特定值，如果在字符串的

后面跟着任何低代理项值，那么它将表示一个字母。

接受一个 int 值的方法支持所有 Unicode 字符，其中包括增补字符。例如，Character.isLetter(0x2F81A) 返回 true，因为代码点值表示一个字母（一个 CJK 象形文字）。

在 Java SE API 文档中，Unicode 代码点 用于范围在 U+0000 与 U+10FFFF 之间的字符值，而 Unicode 代码点 用于作为 UTF-16 编码的代码单元的 16 位 char 值。有关 Unicode 技术的详细信息，请参阅 Unicode Glossary。

构造方法详细信息

Character

```
public Character(char value)
```

构造一个新分配的 Character 对象，用以表示指定的 char 值。

参数：

value - Character 对象表示的值。

方法详细信息

valueOf

```
public static Character valueOf(char c)
```

返回一个表示指定 char 值的 Character 实例。如果不需要新的 Character 实例，则通常应该优先采用此方法，而不是构造方法 Character(char)，因为该方法很可能通过缓存经常请求的值来显著提高空间和时间性能。

参数：

c - 一个 char 值。

返回：

表示 c 的 Character 实例。

---

charValue

public char charValue()

返回此 Character 对象的值。

返回：

此对象表示的基本 char 值。

hashCode

public int hashCode()

返回此 Character 的哈希码。

覆盖：

类 Object 中的 hashCode

返回：

此对象的哈希码值。

另请参见：

Object.equals(java.lang.Object),Hashtable

equals

public boolean equals(Object obj)

将此对象与指定对象比较。当且仅当参数不是 null，而是一个与此对象包含相同 char 值的 Character 对象时，结果才是 true。

覆盖：

类 Object 中的 equals

参数：

obj - 比较的对象。

返回：

如果对象相同，则返回 true；否则返回 false。

另请参见：

Object.hashCode(),Hashtable

toString

public String toString()

返回表示此 Character 值的 String 对象。结果是一个长度为 1 的字符串，其唯一组件是此 Character 对象表示的基本 char 值。

覆盖：

类 Object 中的 toString

返回：

此对象的字符串表示形式。

toString

```
public static String toString(char c)
```

返回一个表示指定 char 值的 String 对象。结果是长度为 1 的字符串，仅由指定的 char 组成。

参数：

c - 要转换的 char 值

返回：

指定 char 值的字符串表示形式

isValidCodePoint

```
public static boolean isValidCodePoint(int codePoint)
```

确定指定的代码点是否为从 0x0000 到 0x10FFFF 范围之内的有效 Unicode 代码点值。该方法等效于以下表达式：

```
codePoint >= 0x0000 && codePoint <= 0x10FFFF
```

参数：

codePoint - 要测试的 Unicode 代码点

返回：

如果指定的代码点值是一个有效的代码点值，则返回 true；否则返回 false。

isSupplementaryCodePoint

```
public static boolean isSupplementaryCodePoint(int codePoint)
```

确定指定字符（Unicode 代码点）是否在增补字符范围内。该方法调用以下表达式：

```
codePoint >= 0x10000 && codePoint <= 0x10FFFF
```

参数：
codePoint - 要测试的字符（Unicode 代码点）
返回：
如果指定字符在 Unicode 增补字符范围内，则返回 true；否则返回 false。

isHighSurrogate

```
public static boolean isHighSurrogate(char ch)
```

确定给出的 char 值是否为一个高代理项代码单元（也称为*前导代理项代码单元*）。这类值并不表示它们本身的字符，而被用来表示 UTF-16 编码中的增补字符。

该方法返回 true 的条件是当且仅当

```
ch >= '\uD800' && ch <= '\uDBFF'
```

为 true。

参数：
ch - 要测试的 char 值。
返回：
如果 char 值在 '\uD800' 与 '\uDBFF' 所包含的范围之间，则返回 true；否则返回 false。
另请参见：
isLowSurrogate(char),Character.UnicodeBlock.of(int)

isLowSurrogate

```
public static boolean isLowSurrogate(char ch)
```

确定给定 char 值是否一个低代理项代码单元（也称为*尾部代理项代码单元*）。这类值并不表示它们本身的字符，而被用来表示 UTF-16 编码中的增补字符。

该方法返回 true 的条件是当且仅当

```
ch >= '\uDC00' && ch <= '\uDFFF'
```

为 true。

参数：

ch - 要测试的 char 值。

返回：

如果 char 值在 '\uDC00' 与 '\uDFFF' 所包含的范围之间，则返回 true；否则返回 false。

另请参见：

isHighSurrogate(char)

---

isSurrogatePair

```
public static boolean isSurrogatePair(char high,
                                       char low)
```

确定指定的 char 值对是否为有效的代理项对。该方法等效于以下表达式：

```
isHighSurrogate(high) && isLowSurrogate(low)
```

参数：

high - 要测试的高代理项代码值

low - 要测试的低代理项代码值

返回：

如果指定的高代理项和低代理项代码值表示的是一个有效的代理项对，则返回 true；否则返回 false。

---

charCount

```
public static int charCount(int codePoint)
```

确定表示指定字符（Unicode 代码点）所需的 char 值的数量。如果指定字符等于或大于 0x10000，则该方法返回的值为 2。否则，该方法返回的值为 1。

该方法没有验证指定的字符是否为一个有效的 Unicode 代码点。如有必要，调用者必须使用 isValidCodePoint 验证字符值。

参数：

codePoint - 要测试的字符（Unicode 代码点）。

返回：

如果字符是一个有效的增补字符，则返回 2；否则返回 1。

另请参见：

isSupplementaryCodePoint(int)

---

toCodePoint

```
public static int toCodePoint(char hight, char low)
```

将指定的代理项对转换为其增补代码点值。该方法没有验证指定的代理项对。如有必要，调用者必须使用 isSurrogatePair 验证它。

参数：

high - 高代理项代码单元

low - 低代理项代码单元

返回：

用指定代理项对组成的增补代码点

---

codePointAt

```
public static int codePointAt(CharSequence seq,  int index)
```

返回 CharSequence 的给定索引上的代码点。如果 CharSequence 中的给定索引上的 char 值在高代理项范围内，则下列索引的长度小于 CharSequence 的长度，如果下列索引上的 char 值在低代理项范围内，则返回对应于此代理项对的增补代码点。否则，返回给定索引上的 char 值。

参数：

seq - char 值（Unicode 代码单元）的序列

index - 要转换的 seq 中的 char 值（Unicode 代码单元）的索引

返回：

给定索引上的 Unicode 代码点

抛出：

NullPointerException - 如果 seq 为 null。

IndexOutOfBoundsException - 如果 index 值为负或不小于 seq.length()。

---

codePointAt

```
public static int codePointAt(char[] a,
                              int index)
```

返回 char 数组的给定索引上的代码点。如果 char 数组中的给定索引上的 char 值在高代理项范围内，则下一个索引的长度小于 char 数组的长度，如果下一个索引上的 char 值在低代理项范围内，则返回对应于此代理项对的增补代码点。否则，返回给定索引上的 char 值。

参数：

a - char 数组

index - 要转换的 char 数组中的 char 值（Unicode 代码单元）的索引

返回：

给定索引上的 Unicode 代码点

抛出：

NullPointerException - 如果 a 为 null。

IndexOutOfBoundsException - 如果 index 值为负或不小于 char 数组的长度。

codePointAt

```
public static int codePointAt(char[] a,
                              int index,
                              int limit)
```

返回 char 数组的给定索引上的代码点，该数组中只有那些具有小于 limit 的 index 值的数组元素可以使用。如果 char 数组中的给定索引上的 char 值在高代理项范围内，则下一个索引小于 limit，如果下一个索引上的 char 值在低代理项范围内，则返回对应于此代理项对的增补代码点。否则，返回给定索引上的 char 值。

参数：

a - char 数组

index - 要转换的 char 数组中的 char 值（Unicode 代码点）的索引

limit - 可以在 char 数组中使用的最后一个数组元素后面的索引

返回：

给定索引上的 Unicode 代码点

抛出：

NullPointerException - 如果 a 为 null。

IndexOutOfBoundsException - 如果 index 参数为负或不小于 limit 参数，或者 limit 参数为负或大于 char 数组的长度。

codePointBefore

```
public static int codePointBefore(CharSequence seq,
                                  int index)
```

返回 CharSequence 的给定索引前面的代码点。如果 CharSequence 中的 (index - 1) 上的 char 值在低代理项范围内，则 (index - 2) 为非负，如果 CharSequence 中的 (index - 2) 上的 char 值在高代理项范围内，则返回对应于此代理项对的增补代码点。否则，返回 (index - 1) 上的 char 值。

参数：

seq - CharSequence 实例

index - 应该返回的代码点后面的索引

返回：

给定索引前面的 Unicode 代码点值。

抛出：

NullPointerException - 如果 seq 为 null。

IndexOutOfBoundsException - 如果 index 参数小于 1 或者大于 seq.length()。

---

codePointBefore

```
public static int codePointBefore(char[] a,
                                  int index)
```

返回 char 数组的给定索引前面的代码点。如果 char 数组中的 (index - 1) 上的 char 值在低代理项范围内，则 (index - 2) 为非负，如果 char 数组中的 (index - 2) 上的 char 值在高代理项范围内，则返回对应于此代理项对的增补代码点。否则，返回 (index - 1) 上的 char 值。

参数：

a - char 数组

index - 应该返回的代码点后面的索引

返回：

给定索引前面的 Unicode 代码点值。

抛出：

NullPointerException - 如果 a 为 null。

IndexOutOfBoundsException - 如果 index 参数小于 1 或大于 char 数组的长度

---

codePointBefore

```
public static int codePointBefore(char[] a,
                                  int index,
                                  int start)
```

返回 char 数组的给定索引前面的代码点，该数组中只有那些具有大于等于 start 的 index 值的数组元素可以使用。如果 char 数组中的 (index－1) 上的 char 值在低代理项范围内，则 (index － 2) 不小于 start，如果 char 数组中的 (index － 2) 上的 char 值在高代理项范围内，则返回对应于此代理项对的增补代码点。否则，返回 (index － 1) 上的 char 值。

参数：

a - char 数组

index - 应该返回的代码点后面的索引

start - char 数组中第一个数组元素的索引

返回：

给定索引前面的 Unicode 代码点。

抛出：

NullPointerException - 如果 a 为 null。

IndexOutOfBoundsException - 如果 index 参数不大于 start 参数或大于 char 数组的长度，或者 start 参数为负或小于 char 数组的长度。

toChars

```
public static int toChars(int codePoint,
                          char[] dst,
                          int dstIndex)
```

将指定字符（Unicode 代码点）转换为其 UTF-16 表示形式。如果指定代码点是一个 BMP（Basic Multilingual Plane 或 Plane 0）值，则在 dst[dstIndex] 中存储相同的值，并返回 1。如果指定代码点是一个增补字符，则将其代理项值存储在 dst[dstIndex]（高代理项）和 dst[dstIndex+1]（低代理项）中，并返回 2。

参数：

codePoint - 要转换的字符（Unicode 代码点）。

dst - char 的一个数组，codePoint 的 UTF-16 值存储在其中。

dstIndex - 进入存储已转换值的 dst 数组中的起始索引。

返回：

如果代码点是一个 BMP 代码点，则返回 1，如果代码点是一个增补代码点，则返回 2。

抛出：

IllegalArgumentException - 如果指定的 codePoint 不是一个有效的 Unicode 代码点。

NullPointerException - 如果指定的 dst 为 null。

IndexOutOfBoundsException - 如果 dstIndex 为负或不小于 dst.length，或者 dstIndex 上的 dst 没有足够多的数组元素来存储得到的 char 值。（如果 dstIndex 等于 dst.length-1 并且指定的 codePoint 是一个增补字符，则不在 dst[dstIndex] 中存储高代理项值。）

toChars

```
public static char[] toChars(int codePoint)
```

将指定的字符（Unicode 代码点）转换成其存储在 char 数组中的 UTF-16 表示形式。如果指定的代码点是一个 BMP（Basic Multilingual Plane 或 Plane 0）值，则得到的 char 数组具有与 codePoint 相同的值。如果指定的代码点是一个增补代码点，则得到的 char 数组具有相应的代理项对。

参数：

codePoint - 一个 Unicode 代码点

返回：

一个具有 codePoint 的 UTF-16 表示形式的 char 数组。

抛出：

IllegalArgumentException - 如果指定的 codePoint 不是一个有效的 Unicode 代码点。

codePointCount

```
public static int codePointCount(CharSequence seq,
                                 int beginIndex,
                                 int endIndex)
```

返回指定字符序列的文本范围内的 Unicode 代码点数量。文本范围始于指定的

beginIndex，并扩展到索引 endIndex - 1 上的 char。因此文本范围的长度（char 形式）为 endIndex-beginIndex。文本范围内的不成对代理项是按一个代码点算作一个项进行计数的。

参数：

seq - 字符序列

beginIndex - 文本范围的第一个 char 的索引。

endIndex - 文本范围的最后一个 char 后面的索引。

返回：

指定文本范围内的 Unicode 代码点的数量

抛出：

NullPointerException - 如果 seq 为 null。

IndexOutOfBoundsException - 如果 beginIndex 为负，或者 endIndex 大于给定序列的长度，或者 beginIndex 大于 endIndex。

---

codePointCount

```
public static int codePointCount(char[] a,
                                 int offset,
                                 int count)
```

返回 char 数组参数的子数组中 Unicode 代码点的数量。offset 参数是子数组的第一个 char 的索引，count 参数指定了 char 中的子数组的长度。子数组中不成对的代理项是按一个代码点算作一个项进行计数的。

参数：

a - char 数组

offset - 给定 char 数组中第一个 char 的索引

count - char 中的子数组的长度

返回：

指定子数组中 Unicode 代码点的数量

抛出：

NullPointerException - 如果 a 为 null。

IndexOutOfBoundsException - 如果 offset 或 count 为负，或者 offset + count 大于给定数组的长度。

---

offsetByCodePoints

```
public static int offsetByCodePoints(CharSequence seq,
                                      int index,
                                      int codePointOffset)
```

返回给定字符序列中的索引，它是从给定 index 到 codePointOffset 代码点的偏移量。index 和 codePointOffset 给出的文本范围内的不成对代理项是按一个代码点算作一个项进行计数的。

参数：

seq - 字符序列

index - 要偏移的索引

codePointOffset - 代码点中的偏移量

返回：

字符序列内的索引

抛出：

NullPointerException - 如果 seq 为 null。

IndexOutOfBoundsException - 如果 index 为负或大于字符序列的长度，或者 codePointOffset 为负并且起始于 index 的子序列拥有的偏移量少于 codePointOffset 代码点，或者 codePointOffset 为负并且 index 前面的子序列的偏移量少于 codePointOffset 代码点的绝对值。

offsetByCodePoints

```
public static int offsetByCodePoints(char[] a,
                                      int start,
                                      int count,
                                      int index,
                                      int codePointOffset)
```

返回给定 char 子数组中的索引，它是从给定 index 到 codePointOffset 代码点的偏移量。start 和 count 参数指定了 char 数组的一个子数组。index 和 codePointOffset 给出的文本范围内的不成对代理项是按一个代码点算作一个项进行计数的。

参数：

a - char 数组

start - 子数组的第一个 char 的索引

count - char 中的子数组的长度

index - 要偏移的索引

codePointOffset - 代码点中的偏移量

返回：
子数组内的索引
抛出：
NullPointerException - 如果 a 为 null。
IndexOutOfBoundsException - 如果存在以下情况：start 或 count 为负；start + count 大于给定数组的长度；index 小于 start 或大于 start + count；codePointOffset 为负并且起始于 index、终止于 start + count - 1 的文本字段的偏移量少于 codePointOffset 代码点；codePointOffset 为负并且起始于 start、终止于 index－1 的文本字段的偏移量少于 codePointOffset 代码点的绝对值。

isLowerCase

public static boolean isLowerCase(char ch)

确定指定字符是否为小写字母。

如果通过 Character.getType(ch) 提供的字符的常规类别类型为 LOWERCASE_LETTER，则字符为小写字母。

以下是小写字母的示例：

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
'\u00DF' '\u00E0' '\u00E1' '\u00E2' '\u00E3' '\u00E4'
'\u00E5' '\u00E6'
'\u00E7' '\u00E8' '\u00E9' '\u00EA' '\u00EB' '\u00EC'
'\u00ED' '\u00EE'
'\u00EF' '\u00F0' '\u00F1' '\u00F2' '\u00F3' '\u00F4'
'\u00F5' '\u00F6'
'\u00F8' '\u00F9' '\u00FA' '\u00FB' '\u00FC' '\u00FD'
'\u00FE' '\u00FF'
```

其他许多 Unicode 字符也是小写的。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isLowerCase(int) 方法。

参数：
ch - 要测试的字符。
返回：
如果字符为小写，则返回 true；否则返回 false。
另请参见：
isLowerCase(char), isTitleCase(char), toLowerCase(char), getType(char)

isLowerCase

public static boolean isLowerCase(int codePoint)

确定指定字符（Unicode 代码点）是否为小写字母。

如果通过 getType(codePoint) 提供的字符的常规类别的类型为 LOWERCASE_LETTER，则字符为小写字母。

以下是小写字母的示例：

```
 a b c d e f g h i j k l m n o p q r s t u v w x y z
 '\u00DF' '\u00E0' '\u00E1' '\u00E2' '\u00E3' '\u00E4'
'\u00E5' '\u00E6'
 '\u00E7' '\u00E8' '\u00E9' '\u00EA' '\u00EB' '\u00EC'
'\u00ED' '\u00EE'
 '\u00EF' '\u00F0' '\u00F1' '\u00F2' '\u00F3' '\u00F4'
'\u00F5' '\u00F6'
 '\u00F8' '\u00F9' '\u00FA' '\u00FB' '\u00FC' '\u00FD'
'\u00FE' '\u00FF'
```

其他许多 Unicode 字符也是小写的。

参数：
codePoint - 要测试的字符（Unicode 代码点）。
返回：
如果字符为小写，则返回 true；否则返回 false。
另请参见：
isLowerCase(int),isTitleCase(int),toLowerCase(int),getType(int)

isUpperCase

public static boolean isUpperCase(char ch)

确定指定字符是否为大写字母。

如果通过 Character.getType(ch) 提供的字符的常规类别类型为 UPPERCASE_LETTER，则字符为大写字母。

以下是大写字母的示例：

```
 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 '\u00C0' '\u00C1' '\u00C2' '\u00C3' '\u00C4' '\u00C5'
'\u00C6' '\u00C7'
 '\u00C8' '\u00C9' '\u00CA' '\u00CB' '\u00CC' '\u00CD'
'\u00CE' '\u00CF'
 '\u00D0' '\u00D1' '\u00D2' '\u00D3' '\u00D4' '\u00D5'
'\u00D6' '\u00D8'
 '\u00D9' '\u00DA' '\u00DB' '\u00DC' '\u00DD' '\u00DE'
```

其他许多 Unicode 字符也是大写的。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isUpperCase(int) 方法。

参数：

ch - 要测试的字符。

返回：

如果字符为大写，则返回 true；否则返回 false。

另请参见：

isLowerCase(char), isTitleCase(char), toUpperCase(char), getType(char)

isUpperCase

public static boolean isUpperCase(int codePoint)

确定指定字符（Unicode 代码点）是否为大写字母。

如果通过 getType(codePoint) 提供的字符的常规类别类型为 UPPERCASE_LETTER，则字符为大写字母。

以下是大写字母的示例：

```
 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 '\u00C0' '\u00C1' '\u00C2' '\u00C3' '\u00C4' '\u00C5'
'\u00C6' '\u00C7'
 '\u00C8' '\u00C9' '\u00CA' '\u00CB' '\u00CC' '\u00CD'
'\u00CE' '\u00CF'
 '\u00D0' '\u00D1' '\u00D2' '\u00D3' '\u00D4' '\u00D5'
'\u00D6' '\u00D8'
 '\u00D9' '\u00DA' '\u00DB' '\u00DC' '\u00DD' '\u00DE'
```

其他许多 Unicode 字符也是大写的。

参数：

codePoint - 要测试的字符（Unicode 代码点）。

返回：

如果字符为大写，则返回 true；否则返回 false。

另请参见：

isLowerCase(int),isTitleCase(int),toUpperCase(int),getType(int)

isTitleCase

public static boolean isTitleCase(char ch)

确定指定字符是否为首字母大写字符。

如果通过 Character.getType(ch) 提供的字符的常规类别类型为 TITLECASE_LETTER，则字符为首字母大写字符。

一些字符看似成对的 Latin 字母。例如，有一个看起来像“LJ”的大写字母和一个看起来像“lj”的对应小写字母。第三种形式看起来像

“Lj”，这是呈现首字母大写的小写单词时使用的适当形式，比如用于书籍的标题。

下面是一些可以让该方法返回 true 的 Unicode 字符：

- LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON
- LATIN CAPITAL LETTER L WITH SMALL LETTER J
- LATIN CAPITAL LETTER N WITH SMALL LETTER J
- LATIN CAPITAL LETTER D WITH SMALL LETTER Z

其他许多 Unicode 字符也是首字母大写的。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isTitleCase(int) 方法。

参数：
ch - 要测试的字符。
返回：
如果字符为首字母大写，则返回 true；否则返回 false。
另请参见：
isLowerCase(char), isUpperCase(char), toTitleCase(char), getType(char)

isTitleCase

public static boolean isTitleCase(int codePoint)

确定指定字符（Unicode 代码点）是否为首字母大写字符。

如果通过 getType(codePoint) 提供的字符的常规类别类型为 TITLECASE_LETTER，则字符为首字母大写字符。

一些字符看似成对的 Latin 字母。例如，有一个看起来像“LJ”的大写字母和一个看起来像“lj”的对应小写字母。第三种形式看起来像“Lj”，这是呈现首字母大写的小写单词时使用的适当形式，比如用于书籍的标题。

下面是一些可以让该方法返回 true 的 Unicode 字符：

- LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON
- LATIN CAPITAL LETTER L WITH SMALL LETTER J
- LATIN CAPITAL LETTER N WITH SMALL LETTER J
- LATIN CAPITAL LETTER D WITH SMALL LETTER Z

其他许多 Unicode 字符也是首字母大写的。

参数：
codePoint - 要测试的字符（Unicode 代码点）。
返回：
如果字符为首字母大写，则返回 true；否则返回 false。
另请参见：
isLowerCase(int),isUpperCase(int),toTitleCase(int),getType(int)

isDigit

```
public static boolean isDigit(char ch)
```

确定指定字符是否为数字。

如果通过 Character.getType(ch) 提供的字符的常规类别类型为 DECIMAL_DIGIT_NUMBER，则字符为数字。

包含数字的 Unicode 字符范围：

- '\u0030' 到 '\u0039'，ISO-LATIN-1 数字（'0' 到 '9'）
- '\u0660' 到 '\u0669'，Arabic-Indic 数字
- '\u06F0' 到 '\u06F9'，扩展了的 Arabic-Indic 数字
- '\u0966' 到 '\u096F'，梵文数字
- '\uFF10' 到 '\uFF19'，全形数字

其他许多字符范围也包含数字。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isDigit(int) 方法。

参数：
ch - 要测试的字符。

返回：
如果字符为数字，则返回 true；否则返回 false。
另请参见：
digit(char, int),forDigit(int, int),getType(char)

---

isDigit

```
public static boolean isDigit(int codePoint)
```

确定指定字符（Unicode 代码点）是否为数字。

如果通过 getType(codePoint) 提供的字符的常规类别类型为 DECIMAL_DIGIT_NUMBER，则字符为数字。

包含数字的 Unicode 字符范围：

- '\u0030' 到 '\u0039'，ISO-LATIN-1 数字（'0' 到 '9'）
- '\u0660' 到 '\u0669'，Arabic-Indic 数字
- '\u06F0' 到 '\u06F9'，扩展了的 Arabic-Indic 数字
- '\u0966' 到 '\u096F'，梵文数字
- '\uFF10' 到 '\uFF19'，全形数字

其他许多字符范围也包含数字。
参数：
codePoint - 要测试的字符（Unicode 代码点）。
返回：
如果字符为数字，则返回 true；否则返回 false。
另请参见：
forDigit(int, int),getType(int)

---

isDefined

```
public static boolean isDefined(char ch)
```

确定字符是否被定义为 Unicode 中的字符。

如果以下条件中至少有一个为真，则字符被定义为 Unicode 中的字符：

- 它具有 UnicodeData 文件中的条目。
- 它具有 UnicodeData 文件定义的范围中的值。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isDefined(int) 方法。

参数：
ch - 要测试的字符
返回：
如果字符具有为 Unicode 中字符定义的含义，则返回 true；否则返回 false。
另请参见：
isDigit(char), isLetter(char), isLetterOrDigit(char), isLowerCase(char),isTitleCase(char),isUpperCase(char)

---

isDefined

```
public static boolean isDefined(int codePoint)
```

确定字符（Unicode 代码点）是否被定义为 Unicode 中的字符。

如果以下条件中至少有一个为真，则字符被定义为 Unicode 中的字符：

- 它具有 UnicodeData 文件中的条目。
- 它具有 UnicodeData 文件定义的范围中的值。

参数：
codePoint - 要测试的字符（Unicode 代码点）。
返回：
如果字符具有为 Unicode 中字符定义的含义，则返回 true；否则返回 false。
另请参见：
isDigit(int),isLetter(int),isLetterOrDigit(int),isLowerCase(int), isTitleCase(int),isUpperCase(int)

---

isLetter

```
public static boolean isLetter(char ch)
```

确定指定字符是否为字母。

如果通过 Character.getType(ch) 为字符提供的常规类别的类型为以下类型中的任意一种，则认为该字符为字母：

- UPPERCASE_LETTER
- LOWERCASE_LETTER
- TITLECASE_LETTER
- MODIFIER_LETTER
- OTHER_LETTER

并非所有的字母都有大小写。许多字符都是字母，但它们既不是大写的，也不是小写的，并且也不是首字母大写的。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isLetter(int) 方法。

参数：
ch - 要测试的字符。
返回：
如果字符为字母，则返回 true；否则返回 false。
另请参见：
isDigit(char), isJavaIdentifierStart(char), isJavaLetter(char), isJavaLetterOrDigit(char), isLetterOrDigit(char), isLowerCase(char), isTitleCase(char), isUnicodeIdentifierStart(char),isUpperCase(char)

---

isLetter

public static boolean isLetter(int codePoint)

确定指定字符（Unicode 代码点）是否为字母。

如果通过 getType(codePoint) 为字符提供的常规类别的类型为以下类型中的任意一种，则认为该字符为字母：

- UPPERCASE_LETTER
- LOWERCASE_LETTER

- TITLECASE_LETTER
- MODIFIER_LETTER
- OTHER_LETTER

并非所有的字母都有大小写。许多字符都是字母，但它们既不是大写的，也不是小写的，并且也不是首字母大写的。

参数：

codePoint - 要测试的字符（Unicode 代码点）。

返回：

如果字符为字母，则返回 true；否则返回 false。

另请参见：

isDigit(int), isJavaIdentifierStart(int), isLetterOrDigit(int), isLowerCase(int), isTitleCase(int), isUnicodeIdentifierStart(int), isUpperCase(int)

isLetterOrDigit

```
public static boolean isLetterOrDigit(char ch)
```

确定指定字符是否为字母或数字。

如果 Character.isLetter(char ch) 或 Character.isDigit(char ch) 对字符返回的是 true，则认为字符是一个字母或数字。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isLetterOrDigit(int) 方法。

参数：

ch - 要测试的字符。

返回：

如果字符为字符或数字，则返回 true；否则返回 false。

另请参见：

isDigit(char), isJavaIdentifierPart(char), isJavaLetter(char), isJavaLetterOrDigit(char), isLetter(char), isUnicodeIdentifierPart(char)

isLetterOrDigit

public static boolean isLetterOrDigit(int codePoint)

确定指定字符（Unicode 代码点）是否为字母或数字。

如果 isLetter(codePoint) 或 isDigit(codePoint) 对字符返回的是 true，则认为字符是一个字母或数字。

参数：

codePoint - 要测试的字符（Unicode 代码点）。

返回：

如果字符为字符或数字，则返回 true；否则返回 false。

从以下版本开始：

1.5

另请参见：

isDigit(int), isJavaIdentifierPart(int), isLetter(int), isUnicodeIdentifierPart(int)

---

isJavaLetter

@Deprecated

public static boolean isJavaLetter(char ch)

已过时。 *由 isJavaIdentifierStart(char) 取代。*

确定是否允许将指定字符作为 Java 标识符中的首字符。

当且仅当以下条件之一为真时，字符才可以作为 Java 标识符的首字母：

- isLetter(ch) 返回 true
- getType(ch) 返回 LETTER_NUMBER
- ch 是一个货币符号（如“$”）
- ch 是一个连字符（如“_”）。

参数：

ch - 要测试的字符。

返回：

如果字符为 Java 标识符的首字符，则返回 true；否则返回 false。

另请参见：
isJavaLetterOrDigit(char), isJavaIdentifierStart(char), isJavaIdentifierPart(char), isLetter(char), isLetterOrDigit(char), isUnicodeIdentifierStart(char)

---

isJavaLetterOrDigit

```
@Deprecated
    public static boolean isJavaLetterOrDigit(char ch)
```

已过时。 *由 isJavaIdentifierPart(char) 取代。*
确定指定字符是否可以是 Java 标识符中首字符以外的部分。

当且仅当以下任何条件为真时，字符才可能是 Java 标识符的一部分：

- 是一个字母
- 是一个货币符号（如 '$'）
- 是一个连字符（如 '_'）。
- 是一个数字
- 是一个数字字母（如罗马数字字符）
- 是个合成标记
- 是一个非空格标记
- isIdentifierIgnorable 对字符返回的是 true。

参数：
ch - 要测试的字符。
返回：
如果字符可以为 Java 标识符的一部分，则返回 true；否则返回 false。
另请参见：
isJavaLetter(char), isJavaIdentifierStart(char), isJavaIdentifierPart(char), isLetter(char), isLetterOrDigit(char), isUnicodeIdentifierPart(char), isIdentifierIgnorable(char)

---

isJavaIdentifierStart

```
    public static boolean isJavaIdentifierStart(char ch)
```

确定是否允许将指定字符作为 Java 标识符中的首字符。

当且仅当以下条件之一为真时，字符才可以作为 Java 标识符的首字符：

- isLetter(ch) 返回 true
- getType(ch) 返回 LETTER_NUMBER
- ch 是一个货币符号（如“$”）
- ch 是一个连字符（如“_”）。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isJavaIdentifierStart(int) 方法。

参数：
ch - 要测试的字符。
返回：
如果字符为 Java 标识符的首字符，则返回 true；否则返回 false。
另请参见：
isJavaIdentifierPart(char), isLetter(char), isUnicodeIdentifierStart(char), SourceVersion.isIdentifier(CharSequence)

isJavaIdentifierStart

```
public static boolean isJavaIdentifierStart(int codePoint)
```

确定是否允许将字符（Unicode 代码点）作为 Java 标识符中的首字符。

当且仅当以下条件之一为真时，字符才可以作为 Java 标识符的首字符：

- isLetter(codePoint) 返回 true
- getType(codePoint) 返回 LETTER_NUMBER
- 引用的字符是一个货币符号（如“$”）
- 引用的字符是一个连字符（如“_”）。

参数：
codePoint - 要测试的字符（Unicode 代码点）。
返回：
如果字符为 Java 标识符的首字符，则返回 true；否则返回 false。

另请参见：
isJavaIdentifierPart(int), isLetter(int), isUnicodeIdentifierStart(int), SourceVersion.isIdentifier(CharSequence)

isJavaIdentifierPart

public static boolean isJavaIdentifierPart(char ch)

确定指定字符是否可以是 Java 标识符中首字符以外的部分。

如果以下任何条件为真，那么字符可能是 Java 标识符的一部分：

- 是一个字母
- 是一个货币符号（如 '$'）
- 是一个连字符（如 '_'）。
- 是一个数字
- 是一个数字字母（如罗马数字字符）
- 是个合成标记
- 是一个非空格标记
- isIdentifierIgnorable 对字符返回的是 true

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isJavaIdentifierPart(int) 方法。

参数：
ch - 要测试的字符。
返回：
如果字符可以为 Java 标识符的一部分，则返回 true；否则返回 false。
另请参见：
isIdentifierIgnorable(char), isJavaIdentifierStart(char), isLetterOrDigit(char), isUnicodeIdentifierPart(char), SourceVersion.isIdentifier(CharSequence)

isJavaIdentifierPart

public static boolean isJavaIdentifierPart(int codePoint)

确定字符（Unicode 代码点）是否可以是 Java 标识符中首字符以外的部分。

如果以下任何条件为真，那么字符可能是 Java 标识符的一部分：

- 是一个字母
- 是一个货币符号（如 '$'）
- 是一个连字符（如 '_'）。
- 是一个数字
- 是一个数字字母（如罗马数字字符）
- 是个合成标记
- 是一个非空格标记
- isIdentifierIgnorable(codePoint) 对字符返回的是 true

参数：
codePoint - 要测试的字符（Unicode 代码点）。
返回：
如果字符可以为 Java 标识符的一部分，则返回 true；否则返回 false。
另请参见：
isIdentifierIgnorable(int), isJavaIdentifierStart(int), isLetterOrDigit(int), isUnicodeIdentifierPart(int), SourceVersion.isIdentifier(CharSequence)

isUnicodeIdentifierStart

public static boolean isUnicodeIdentifierStart(char ch)

确定是否允许将指定字符作为 Unicode 标识符中的首字符。

当且仅当以下条件之一为真时，字符才可以作为 Unicode 标识符的首字符：

- isLetter(ch) 返回 true
- getType(ch) 返回 LETTER_NUMBER。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isUnicodeIdentifierStart(int) 方法。

参数：
ch - 要测试的字符。
返回：
如果字符可以作为 Unicode 标识符的首字符，则返回 true；否则返回 false。
另请参见：
isJavaIdentifierStart(char), isLetter(char), isUnicodeIdentifierPart(char)

isUnicodeIdentifierStart

```
public static boolean isUnicodeIdentifierStart(int codePoint)
```

确定是否允许将指定字符（Unicode 代码点）作为 Unicode 标识符中的首字符。

当且仅当以下条件之一为真时，字符才可以作为 Unicode 标识符的首字符：

- isLetter(codePoint) 返回 true
- getType(codePoint) 返回 LETTER_NUMBER。

参数：
codePoint - 要测试的字符（Unicode 代码点）。
返回：
如果字符可以作为 Unicode 标识符的首字符，则返回 true；否则返回 false。
另请参见：
isJavaIdentifierStart(int), isLetter(int), isUnicodeIdentifierPart(int)

isUnicodeIdentifierPart

```
public static boolean isUnicodeIdentifierPart(char ch)
```

确定指定字符是否可以是 Unicode 标识符中首字符以外的部分。

当且仅当以下语句之一为真时，字符才可能是 Unicode 标识符的一部分：

- 是一个字母

- 是一个连字符（如 '_'）。
- 是一个数字
- 是一个数字字母（如罗马数字字符）
- 是个合成标记
- 是一个非空格标记
- isIdentifierIgnorable 对该字符返回的是 true。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isUnicodeIdentifierPart(int) 方法。

参数：
ch - 要测试的字符。
返回：
如果字符可以为 Unicode 标识符的一部分，则返回 true；否则返回 false。
另请参见：
isIdentifierIgnorable(char), isJavaIdentifierPart(char), isLetterOrDigit(char),isUnicodeIdentifierStart(char)

---

isUnicodeIdentifierPart

public static boolean isUnicodeIdentifierPart(int codePoint)

确定指定字符（Unicode 代码点）是否可以是 Unicode 标识符中首字符以外的部分。

当且仅当以下语句之一为真时，字符才可能是 Unicode 标识符的一部分：

- 是一个字母
- 是一个连字符（如 '_'）。
- 是一个数字
- 是一个数字字母（如罗马数字字符）
- 是个合成标记
- 是一个非空格标记
- isIdentifierIgnorable 对该字符返回的是 true。

参数：
codePoint - 要测试的字符（Unicode 代码点）。
返回：
如果字符可以为 Unicode 标识符的一部分，则返回 true；否则返回 false。

另请参见：
isIdentifierIgnorable(int), isJavaIdentifierPart(int), isLetterOrDigit(int),isUnicodeIdentifierStart(int)

isIdentifierIgnorable

```
public static boolean isIdentifierIgnorable(char ch)
```

确定是否应该认为指定字符是 Java 标识符或 Unicode 标识符中可忽略的一个字符。

以下 Unicode 字符是 Java 标识符或 Unicode 标识符中可忽略的字符：

```
非空白的 ISO 控制字符
'\u0000' 到 '\u0008'
'\u000E' 到 '\u001B'
'\u007F' 到 '\u009F'
拥有 FORMAT 常规类别值的所有字符
```

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isIdentifierIgnorable(int) 方法。

参数：
ch - 要测试的字符。
返回：
如果字符是可以成为 Java 或 Unicode 标识符的一部分的可忽略控制字符，则返回 true；否则返回 false。
另请参见：
isJavaIdentifierPart(char),isUnicodeIdentifierPart(char)

isIdentifierIgnorable

```
public static boolean isIdentifierIgnorable(int codePoint)
```

确定是否应该认为指定字符（Unicode 代码点）是 Java 标识符或 Unicode 标识符中可忽略的一个字符。

以下 Unicode 字符是 Java 标识符或 Unicode 标识符中可忽略的字符：

非空白的 ISO 控制字符
'\u0000' 到 '\u0008'
'\u000E' 到 '\u001B'
'\u007F' 到 '\u009F'

- 拥有 FORMAT 常规类别值的所有字符

参数：
codePoint - 要测试的字符（Unicode 代码点）。
返回：
如果字符是可以成为 Java 或 Unicode 标识符的一部分的可忽略控制字符，则返回 true；否则返回 false。
另请参见：
isJavaIdentifierPart(int),isUnicodeIdentifierPart(int)

---

toLowerCase

public static char toLowerCase(char ch)

使用取自 UnicodeData 文件的大小写映射信息将字符参数转换为小写。

注意，对于某些范围内的字符，特别是那些是符号或表意符号的字符，Character.isLowerCase(Character.toLowerCase(ch)) 并不总是返回 true。

通常，应该使用 String.toLowerCase() 将字符映射为小写。String 大小写映射方法有几个胜过 Character 大小写映射方法的优点。String 大小写映射方法可以执行语言环境敏感的映射、上下文相关的映射和 1:M 字符映射，而 Character 大小写映射方法却不能。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 toLowerCase(int) 方法。

参数：
ch - 要转换的字符。
返回：
等效于字符的小写形式，如果有的话；否则返回字符本身。
另请参见：

isLowerCase(char),String.toLowerCase()

toLowerCase

public static int toLowerCase(int codePoint)

使用取自 UnicodeData 文件的大小写映射信息将字符（Unicode 代码点）参数转换为小写。

注意，对于某些范围内的字符，特别是那些是符号或表意符号的字符，Character.isLowerCase(Character.toLowerCase(codePoint)) 并不总是返回 true。

通常，应该使用 String.toLowerCase() 将字符映射为小写。String 大小写映射方法有几个胜过 Character 大小写映射方法的优点。String 大小写映射方法可以执行语言环境敏感的映射、上下文相关的映射和 1:M 字符映射，而 Character 大小写映射方法却不能。

参数：

codePoint - 要转换的字符（Unicode 代码点）。

返回：

等效于字符（Unicode 代码点）的小写形式，如果有的话；否则返回字符本身。

另请参见：

isLowerCase(int),String.toLowerCase()

toUpperCase

public static char toUpperCase(char ch)

使用取自 UnicodeData 文件的大小写映射信息将字符参数转换为大写。

注意，对于某些范围内的字符，特别是那些是符号或表意符号的字符，Character.isUpperCase(Character.toUpperCase(ch)) 并不总是返回 true。

通常，应该使用 String.toUpperCase() 将字符映射为大写。String 大小写映射方法有几个胜过 Character 大小写映射方法的优点。String 大小写映射方法可以执行语言环境敏感的映射、上下文相关的映射和 1:M 字符映射，而 Character 大小写映射方法却不能。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 toUpperCase(int) 方法。

参数：

ch - 要转换的字符。

返回：

等效于字符的大写形式，如果有的话；否则返回字符本身。

另请参见：

isUpperCase(char),String.toUpperCase()

---

toUpperCase

public static int toUpperCase(int codePoint)

使用取自 UnicodeData 文件的大小写映射信息将字符（Unicode 代码点）参数转换为大写。

注意，对于某些范围内的字符，特别是那些是符号或表意符号的字符，Character.isUpperCase(Character.toUpperCase(codePoint)) 并不总是返回 true。

通常，应该使用 String.toUpperCase() 将字符映射为大写。String 大小写映射方法有几个胜过 Character 大小写映射方法的优点。String 大小写映射方法可以执行语言环境敏感的映射、上下文相关的映射和 1:M 字符映射，而 Character 大小写映射方法却不能。

参数：

codePoint - 要转换的字符（Unicode 代码点）。

返回：

等效于字符的大写形式，如果有的话；否则返回字符本身。

另请参见：

isUpperCase(int),String.toUpperCase()

---

toTitleCase

public static char toTitleCase(char ch)

使用取自 UnicodeData 文件的大小写映射信息将字符参数转换为首字母大写。如果

字符没有明确的首字母大写映射，并且根据 UnicodeData，它本身并不是一个首字母大写的 char，则返回大写映射作为等效的首字母大写映射。如果 char 参数总是一个首字母大写的 char，则返回相同的 char 值。

注意，对于某些范围内的字符，Character.isTitleCase(Character.toTitleCase(ch)) 并不总是返回 true。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 toTitleCase(int) 方法。

参数：
ch - 要转换的字符。
返回：
如果有的话，则返回等效于字符的首字母大写形式；否则返回字符本身。
另请参见：
isTitleCase(char), toLowerCase(char), toUpperCase(char)

---

toTitleCase

public static int toTitleCase(int codePoint)

使用取自 UnicodeData 文件的大小写映射信息将字符（Unicode 代码点）参数转换为首字母大写。如果字符没有明确的首字母大写映射，并且根据 UnicodeData，它本身并不是一个首字母大写的 char，则返回大写映射作为等效的首字母大写映射。如果字符参数总是一个首字母大写的字符，则返回相同的字符值。

注意，对于某些范围内的字符，
Character.isTitleCase(Character.toTitleCase(codePoint)) 并不总是返回 true。

参数：
codePoint - 要转换的字符（Unicode 代码点）。
返回：
如果有的话，则返回等效于字符的首字母大写形式；否则返回字符本身。
另请参见：
isTitleCase(int), toLowerCase(int), toUpperCase(int)

---

digit

```
public static int digit(char ch,
                        int radix)
```

返回使用指定基数的字符 ch 的数值。

如果基数不在 MIN_RADIX <= radix <= MAX_RADIX 范围之内，或者 ch 的值是一个使用指定基数的无效数字，则返回 -1。如果以下条件中至少有一个为真，则字符是一个有效数字：

- 方法 isDigit 为 true，且字符（或分解的单字符）的 Unicode 十进制数值小于指定的基数。在这种情况下，返回十进制数值。
- 字符为 'A' 到 'Z' 范围内的大写拉丁字母之一，且它的代码小于 radix + 'A' - 10。在这种情况下，返回 ch - 'A' + 10。
- 字符为 'a' 到 'z' 范围内的小写拉丁字母之一，且它的代码小于 radix + 'a' - 10。在这种情况下，返回 ch - 'a' + 10。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 digit(int, int) 方法。

参数：

ch - 要转换的字符。

radix - 基数。

返回：

使用指定基数的字符所表示的数值。

另请参见：

forDigit(int, int),isDigit(char)

digit

```
public static int digit(int codePoint,
                        int radix)
```

返回使用指定基数的指定字符（Unicode 代码点）的数值。

如果基数不在 MIN_RADIX <= radix <= MAX_RADIX 范围之内，或者字符是一个使用指定基数的无效数字，则返回 -1。如果以下条件中至少有一个为真，则字符是一个有效数字：

- 方法 isDigit(codePoint) 为 true，且字符（或分解的单字符）的 Unicode 十进制数值小于指定的基数。在这种情况下，返回十进制数值。
- 字符为 'A' 到 'Z' 范围内的大写拉丁字母之一，且它的代码小于 radix + 'A' - 10。在这种情况下，返回 ch - 'A' + 10。
- 字符为 'a' 到 'z' 范围内的小写拉丁字母之一，且它的代码小于 radix + 'a' - 10。在这种情况下，返回 ch - 'a' + 10。

参数：

codePoint - 要转换的字符（Unicode 代码点）。

radix - 基数。

返回：

使用指定基数的字符所表示的数值。

另请参见：

forDigit(int, int),isDigit(int)

getNumericValue

```
public static int getNumericValue(char ch)
```

返回指定的 Unicode 字符表示的 int 值。例如，字符 '\u216C'（罗马数字 50）将返回一个值为 50 的整数。

字母 A-Z 的大写（'\u0041' 到 '\u005A'）、小写（'\u0061' 到 '\u007A'）和全形参数（'\uFF21' 到 '\uFF3A' 和 '\uFF41' 到 '\uFF5A'）形式拥有从 10 到 35 的数值。这独立于 Unicode 规范，该规范没有为这些 char 值分配数值。

如果字符中没有数值，则返回 -1。如果字符中有一个数值，但无法将它表示为非负整数（例如，小数值），则返回 -2。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 getNumericValue(int) 方法。

参数：

ch - 要转换的字符。

返回：

以非负 int 值形式返回字符的数值；如果字符中有一个不是非负整数的数值，则返回 -2；如果字符中没有数值，则返回 -1。

另请参见：

forDigit(int, int),isDigit(char)

getNumericValue

public static int getNumericValue(int codePoint)

返回指定字符（Unicode 代码点）表示的 int 值。例如，字符 '\u216C'（罗马数字 50）将返回一个值为 50 的 int 值。

字母 A-Z 的大写（'\u0041' 到 '\u005A'）、小写（'\u0061' 到 '\u007A'）和全形参数（'\uFF21' 到 '\uFF3A' 和 '\uFF41' 到 '\uFF5A'）形式拥有从 10 到 35 的数值。这独立于 Unicode 规范，该规范没有为这些 char 值分配数值。

如果字符中没有数字值，则返回 -1。如果字符中有一个数值，但无法将它表示为非负整数（例如，小数值），则返回 -2。

参数：

codePoint - 要转换的字符（Unicode 代码点）。

返回：

以非负 int 值形式返回字符的数值；如果字符中有一个不是非负整数的数值，则返回 -2；如果字符中没有数值，则返回 -1。

另请参见：

forDigit(int, int),isDigit(int)

isSpace

@Deprecated

public static boolean isSpace(char ch)

已过时。 *由 isWhitespace(char) 取代。*

确定指定字符是否为 ISO-LATIN-1 空白。该方法只对以下五个字符返回 true：

'\t'	'\u0009'	HORIZONTAL TABULATION
'\n'	'\u000A'	NEW LINE
'\f'	'\u000C'	FORM FEED
'\r'	'\u000D'	CARRIAGE RETURN

' '　　'\u0020'　　SPACE

参数：

ch - 要测试的字符。

返回：

如果字符为 ISO-LATIN-1 空白，则返回 true；否则返回 false。

另请参见：

isSpaceChar(char),isWhitespace(char)

---

isSpaceChar

public static boolean isSpaceChar(char ch)

确定指定字符是否为 Unicode 空白字符。当且仅当根据 Unicode 标准将字符指定为空白字符时，才认为字符是一个空白字符。如果字符的常规类别的类型为以下类型中的任意一种，则该方法返回 true：

- SPACE_SEPARATOR
- LINE_SEPARATOR
- PARAGRAPH_SEPARATOR

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isSpaceChar(int) 方法。

参数：

ch - 要测试的字符。

返回：

如果字符为空白字符，则返回 true；否则返回 false。

另请参见：

isWhitespace(char)

---

isSpaceChar

public static boolean isSpaceChar(int codePoint)

确定指定字符（Unicode 代码点）是否为 Unicode 空白字符。当且仅当根据 Unicode 标准将字符指定为空白字符时，才认为字符是一个空白字符。如果字符的常规类别的

类型为以下类型中的任意一种，则该方法返回 true：

- SPACE_SEPARATOR
- LINE_SEPARATOR
- PARAGRAPH_SEPARATOR

参数：

codePoint - 要测试的字符（Unicode 代码点）。

返回：

如果字符为空白字符，则返回 true；否则返回 false。

另请参见：

isWhitespace(int)

---

isWhitespace

```
public static boolean isWhitespace(char ch)
```

确定指定字符依据 Java 标准是否为空白字符。当且仅当字符满足以下标准时，该字符才是一个 Java 空白字符：

它是 Unicode 空格字符（SPACE_SEPARATOR、LINE_SEPARATOR 或 PARAGRAPH_SEPARATOR），但不是非中断空格（'\u00A0'、'\u2007'、'\u202F'）

它是 '\u0009'，HORIZONTAL TABULATION

它是 '\u000A'，LINE FEED

它是 '\u000B'，VERTICAL TABULATION

它是 '\u000C'，FORM FEED

它是 '\u000D'，CARRIAGE RETURN

它是 '\u001C'，FILE SEPARATOR

它是 '\u001D'，GROUP SEPARATOR

它是 '\u001E'，RECORD SEPARATOR

它是 '\u001F'，UNIT SEPARATOR

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isWhitespace(int) 方法。

参数：

ch - 要测试的字符。

返回：
如果字符为 Java 空白字符，则返回 true；否则返回 false。
另请参见：
isSpaceChar(char)

isWhitespace

public static boolean isWhitespace(int codePoint)

确定指定字符（Unicode 代码点）依据 Java 标准是否为空白字符。当且仅当字符满足以下标准时，该字符才是一个 Java 空白字符：

- 它是一个 Unicode 空白字符（SPACE_SEPARATOR、LINE_SEPARATOR 或 PARAGRAPH_SEPARATOR），但不是一个不间断空格（'\u00A0'、'\u2007' 和 '\u202F'）。
- '\u0009'，水平制表符。
- '\u000A'，换行。
- '\u000B'，纵向制表符。
- '\u000C'，换页。
- '\u000D'，回车。
- '\u001C'，文件分隔符。
- '\u001D'，组分隔符。
- '\u001E'，记录分隔符。
- '\u001F'，单元分隔符。

参数：
codePoint - 要测试的字符（Unicode 代码点）。
返回：
如果字符为 Java 空白字符，则返回 true；否则返回 false。
另请参见：
isSpaceChar(int)

isISOControl

public static boolean isISOControl(char ch)

确定指定字符是否为 ISO 控制字符。如果字符的代码在从 '\u0000' 到 '\u001F' 或从 '\u007F' 到 '\u009F' 的范围内，则认为该字符是一个 ISO 控制字符。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isISOControl(int) 方法。

参数：

ch - 要测试的字符。

返回：

如果字符为 ISO 控制字符，则返回 true；否则返回 false。

另请参见：

isSpaceChar(char),isWhitespace(char)

---

isISOControl

public static boolean isISOControl(int codePoint)

确定引用的字符（Unicode 代码点）是否为 ISO 控制字符。如果字符的代码在从 '\u0000' 到 '\u001F' 或从 '\u007F' 到 '\u009F' 的范围内，则认为该字符是一个 ISO 控制字符。

参数：

codePoint - 要测试的字符（Unicode 代码点）。

返回：

如果字符为 ISO 控制字符，则返回 true；否则返回 false。

另请参见：

isSpaceChar(int),isWhitespace(int)

---

getType

public static int getType(char ch)

返回一个指示字符的常规类别的值。

注： 此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 getType(int) 方法。

参数：

ch - 要测试的字符。

返回：

表示字符的常规类别的 int 类型的值。

另请参见：

COMBINING_SPACING_MARK, CONNECTOR_PUNCTUATION, CONTROL, CURRENCY_SYMBOL, DASH_PUNCTUATION, DECIMAL_DIGIT_NUMBER, ENCLOSING_MARK, END_PUNCTUATION, FINAL_QUOTE_PUNCTUATION, FORMAT, INITIAL_QUOTE_PUNCTUATION, LETTER_NUMBER, LINE_SEPARATOR, LOWERCASE_LETTER, MATH_SYMBOL, MODIFIER_LETTER, MODIFIER_SYMBOL, NON_SPACING_MARK, OTHER_LETTER, OTHER_NUMBER, OTHER_PUNCTUATION, OTHER_SYMBOL, PARAGRAPH_SEPARATOR, PRIVATE_USE, SPACE_SEPARATOR, START_PUNCTUATION, SURROGATE, TITLECASE_LETTER, UNASSIGNED, UPPERCASE_LETTER

---

getType

```
public static int getType(int codePoint)
```

返回一个指示字符的常规类别的值。

参数：

codePoint - 要测试的字符（Unicode 代码点）。

返回：

表示字符的常规类别的 int 类型的值。

另请参见：

COMBINING_SPACING_MARK, CONNECTOR_PUNCTUATION, CONTROL, CURRENCY_SYMBOL, DASH_PUNCTUATION, DECIMAL_DIGIT_NUMBER, ENCLOSING_MARK, END_PUNCTUATION, FINAL_QUOTE_PUNCTUATION, FORMAT, INITIAL_QUOTE_PUNCTUATION, LETTER_NUMBER, LINE_SEPARATOR, LOWERCASE_LETTER, MATH_SYMBOL, MODIFIER_LETTER, MODIFIER_SYMBOL, NON_SPACING_MARK, OTHER_LETTER, OTHER_NUMBER, OTHER_PUNCTUATION, OTHER_SYMBOL, PARAGRAPH_SEPARATOR, PRIVATE_USE, SPACE_SEPARATOR, START_PUNCTUATION, SURROGATE, TITLECASE_LETTER, UNASSIGNED, UPPERCASE_LETTER

---

forDigit

public static char forDigit(int digit,
int radix)

确定使用指定基数的特定数字的字符表示形式。如果 radix 的值不是一个有效基数，或者 digit 的值不是一个使用指定基数的有效数字，则返回 null 字符（'\u0000'）。

如果 radix 参数大于等于 MIN_RADIX 并且小于等于 MAX_RADIX，则该参数是有效的。如果 0 <=digit < radix，则 digit 参数是有效的。

如果数字小于 10，则返回 '0' + digit。否则，返回的值为 'a' + digit - 10。

参数：
digit - 转换为字符的数字。
radix - 基数。
返回：
使用指定基数的指定数字的 char 表示形式。
另请参见：
MIN_RADIX,MAX_RADIX,digit(char, int)

getDirectionality

public static byte getDirectionality(char ch)

返回给定字符的 Unicode 方向属性。利用字符方向性来计算文本的视觉顺序。未定义的 char 值的方向值是 DIRECTIONALITY_UNDEFINED。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 getDirectionality(int) 方法。

参数：
ch - 为其请求方向属性的 char。
返回：
char 值的方向属性。
另请参见：
DIRECTIONALITY_UNDEFINED, DIRECTIONALITY_LEFT_TO_RIGHT, DIRECTIONALITY_RIGHT_TO_LEFT,DIRECTIONALITY_RIGHT_TO_LEFT_ARABIC, DIRECTIONALITY_EUROPEAN_NUMBER,

DIRECTIONALITY_EUROPEAN_NUMBER_SEPARATOR,
DIRECTIONALITY_EUROPEAN_NUMBER_TERMINATOR,
DIRECTIONALITY_ARABIC_NUMBER,
DIRECTIONALITY_COMMON_NUMBER_SEPARATOR,
DIRECTIONALITY_NONSPACING_MARK, DIRECTIONALITY_BOUNDARY_NEUTRAL,
DIRECTIONALITY_PARAGRAPH_SEPARATOR,
DIRECTIONALITY_SEGMENT_SEPARATOR, DIRECTIONALITY_WHITESPACE,
DIRECTIONALITY_OTHER_NEUTRALS,
DIRECTIONALITY_LEFT_TO_RIGHT_EMBEDDING,
DIRECTIONALITY_LEFT_TO_RIGHT_OVERRIDE,
DIRECTIONALITY_RIGHT_TO_LEFT_EMBEDDING,
DIRECTIONALITY_RIGHT_TO_LEFT_OVERRIDE,
DIRECTIONALITY_POP_DIRECTIONAL_FORMAT

getDirectionality

public static byte getDirectionality(int codePoint)

返回给定字符（Unicode 代码点）的 Unicode 方向属性。利用字符方向性来计算文本的视觉顺序。未定义字符的方向值是 DIRECTIONALITY_UNDEFINED。

参数：

codePoint - 为其请求方向属性的字符（Unicode 代码点）。

返回：

字符的方向属性。

另请参见：

DIRECTIONALITY_UNDEFINED, DIRECTIONALITY_LEFT_TO_RIGHT,
DIRECTIONALITY_RIGHT_TO_LEFT,DIRECTIONALITY_RIGHT_TO_LEFT_ARABIC,
DIRECTIONALITY_EUROPEAN_NUMBER,
DIRECTIONALITY_EUROPEAN_NUMBER_SEPARATOR,
DIRECTIONALITY_EUROPEAN_NUMBER_TERMINATOR,
DIRECTIONALITY_ARABIC_NUMBER,
DIRECTIONALITY_COMMON_NUMBER_SEPARATOR,
DIRECTIONALITY_NONSPACING_MARK, DIRECTIONALITY_BOUNDARY_NEUTRAL,
DIRECTIONALITY_PARAGRAPH_SEPARATOR,
DIRECTIONALITY_SEGMENT_SEPARATOR, DIRECTIONALITY_WHITESPACE,
DIRECTIONALITY_OTHER_NEUTRALS,
DIRECTIONALITY_LEFT_TO_RIGHT_EMBEDDING,

DIRECTIONALITY_LEFT_TO_RIGHT_OVERRIDE,
DIRECTIONALITY_RIGHT_TO_LEFT_EMBEDDING,
DIRECTIONALITY_RIGHT_TO_LEFT_OVERRIDE,
DIRECTIONALITY_POP_DIRECTIONAL_FORMAT

isMirrored

public static boolean isMirrored(char ch)

确定指定字符依据 Unicode 规范是否对称。当显示在以从右至左的方式显示的文本中时，对称字符的字形在水平方向上应该是对称的。例如，'\u0028' LEFT PARENTHESIS 在语义上被定义为是*开括号*。在从左至右显示的文本中，它将显示为"("，但在以从右至左的方式显示的文本中，它显示为")"。

注：此方法无法处理增补字符。若要支持所有 Unicode 字符，包括增补字符，请使用 isMirrored(int) 方法。

参数：
ch - 为其请求对称属性的 char
返回：
如果字符是对称的，则返回 true，如果 char 不是对称的或者尚未定义，则返回 false。

isMirrored

public static boolean isMirrored(int codePoint)

确定指定字符（Unicode 代码点）依据 Unicode 规范是否对称。当显示在以从右至左的方式显示的文本中时，对称字符的字形在水平方向上应该是对称的。例如，'\u0028' LEFT PARENTHESIS 在语义上被定义为是*开括号*。在从左至右显示的文本中，它将显示为"("，但在以从右至左的方式显示的文本中，它显示为")"。
参数：
codePoint - 要测试的字符（Unicode 代码点）。
返回：
如果字符是对称的，则返回 true，如果字符是不对称的或者尚未定义，则返回 false。

compareTo

public int compareTo(Character anotherCharacter)

根据数字比较两个 Character 对象。

指定者：

接口 Comparable<Character> 中的 compareTo

参数：

anotherCharacter - 要比较的 Character。

返回：

如果该 Character 等于此 Character，则返回 0；如果该 Character 的数值小于参数 Character，则返回小于 0 的值；如果该 Character 的数值大于参数 Character，则返回大于 0 的值（有符号比较）。注意，这是一次严格的数字比较；它并不依赖于区域。

---

reverseBytes

public static char reverseBytes(char ch)

返回通过反转指定 char 值中的字节顺序而获得的值。

返回：

通过反转（或者等效于交换）指定 char 值中的字节而获得的值。

Double 类

Double 类在对象中包装一个基本类型 double 的值。每个 Double 类型的对象都包含一个 double 类型的字段。

此外，该类还提供多个方法，可以将 double 转换为 String，将 String 转换为 double，也提供了其他一些处理 double 时有用的常量和方法。

构造方法详细信息

Double

public Double(double value)

构造一个新分配的 Double 对象，它表示基本的 double 参数。

参数：

value - 用 Double 表示的值。

Double

public Double(String s)
throws NumberFormatException

构造一个新分配的 Double 对象，表示用字符串表示的 double 类型的浮点值。该方法与 valueOf 方法一样，可将字符串转换为 double 值。

参数：

s - 要转换为 Double 的字符串。

抛出：

NumberFormatException - 如果字符串不包含可解析的数字。

另请参见：

valueOf(java.lang.String)

方法详细信息 toString

public static String toString(double d)

返回 double 参数的字符串表示形式。下面提到的所有字符都是 ASCII 字符。

- 如果参数为 NaN，那么结果为字符串 "NaN"。
- 否则，结果是表示参数符号和数值（绝对值）的字符串。如果符号为负，那么结果的第一个字符是 '-'('\u002D')；如果符号为正，那么结果中不显示符号字符。对于数值 *m*：
  - 如果 *m* 为无穷大，则用字符 "Infinity" 表示；因此，正无穷大生成结果 "Infinity"，负无穷大生成结果 "-Infinity"。
  - 如果 *m* 为 0，则用字符 "0.0" 表示；因此，负 0 生成结果 "-0.0"，正 0 生成结果 "0.0"。
  - 如果 *m* 大于或者等于 10^{-3}，但小于 10^{7}，则采用不带前导 0 的十进制形式，用 *m* 的整数部分表示，后跟 '.'('\u002E')，再后面是表示 *m* 小数部分的一个或多个十进制数字。

- 如果 *m* 小于 10^{-3} 或大于等于 10^{7}，则使用所谓的“计算机科学记数法”表示。设 *n* 为满足 $10^{n} <= m < 10^{n+1}$ 的唯一整数；然后设 *a* 为 *m* 与 10^{n} 的精确算术商，从而 $1 <= a < 10$。那么，数值便表示为 *a* 的整数部分，其形式为：一个十进制数字，后跟 '.'('\u002E')，接着是表示 *a* 小数部分的十进制数字，再后面是字母 'E'('\u0045')，最后是用十进制整数形式表示的 *n*，这与方法 Integer.toString(int) 生成的结果一样。

必须为 *m* 或 *a* 的小数部分显示多少位呢？至少必须有一位数来表示小数部分，除此之外，需要更多（但只能和需要的一样多）位数来唯一地区分参数值和 double 类型的邻近值。这就是说，假设 *x* 是用十进制表示法表示的精确算术值，是通过对有限非 0 参数 *d* 调用此方法生成的。那么 *d* 一定是最接近 *x* 的 double 值；如果两个 double 值都同等地接近 *x*，那么 *d* 必须是其中之一，并且 *d* 的有效数字的最低有效位必须是 0。

要创建浮点值的本地化字符串表示形式，请使用 NumberFormat 的子类。

参数：
d - 要转换的 double 值。
返回：
参数的字符串表示形式。

---

toHexString

```
public static String toHexString(double d)
```

返回 double 参数的十六进制字符串表示形式。下面提到的所有字符都是 ASCII 字符。

- 如果参数为 NaN，那么结果为字符串 "NaN"。
- 否则，结果是表示参数符号和数值的字符串。如果符号为负，那么结果的第一个字符是 '-'('\u002D')；如果符号为正，那么结果中不显示符号字符。对于数值 *m*：
  - 如果 *m* 为无穷大，则用字符串 "Infinity" 表示；因此，正无穷大生成结果 "Infinity"，负无穷大生成结果 "-Infinity"。
  - 如果 *m* 为 0，则用字符串 "0x0.0p0" 表示；因此，负 0 生成结果 "-0x0.0p0"，正 0 生成结果 "0x0.0p0"。

- 如果 *m* 是具有标准化表示形式的 double 值，则使用子字符串表示有效数字和指数字段。有效数字用字符 "0x1." 表示，后跟该有效数字剩余小数部分的小写十六进制表示形式。除非所有位数都为 0，否则移除十六进制表示中的尾部 0，在所有位数都为零的情况下，可以用一个 0 表示。接下来用 "p" 表示指数，后跟无偏指数的十进制字符串，该值与通过对指数值调用 Integer.toString 生成的值相同。
- 如果 *m* 是非标准表示形式的 double 值，则用字符 "0x0." 表示有效数字，后跟该有效数字剩余小数部分的十六进制表示形式。移除十六进制表示中的尾部 0。然后用 "p-1022" 表示指数。注意，在非标准有效数字中，必须至少有一个非 0 数字。

浮点值	十六进制字符串
1.0	0x1.0p0
-1.0	-0x1.0p0
2.0	0x1.0p1
3.0	0x1.8p1
0.5	0x1.0p-1
0.25	0x1.0p-2
Double.MAX_VALUE	0x1.fffffffffffffp1023
Minimum Normal Value	0x1.0p-1022
Maximum Subnormal Value	0x0.fffffffffffffp-1022
Double.MIN_VALUE	0x0.0000000000001p-1022

参数：

d - 要转换的 double 值。

返回：

参数的十六进制字符串表示形式。

valueOf

```
public static Double valueOf(String s)
                             throws NumberFormatException
```

返回保存用参数字符串 s 表示的 double 值的 Double 对象。

如果 s 为 null，则抛出 NullPointerException 异常。

忽略 s 中的前导空白字符和尾部空白字符。就像调用 String.trim() 方法那样移除空白；这就是说，ASCII 空格和控制字符都要移除。s 的其余部分应该按词法语法规则所描述的那样构成一个 *FloatValue*：

```
FloatValue:
Signopt NaN
Signopt Infinity
Signopt FloatingPointLiteral
Signopt HexFloatingPointLiteral
SignedInteger
HexFloatingPointLiteral:
HexSignificand BinaryExponent FloatTypeSuffixopt
HexSignificand:
HexNumeral
HexNumeral .
0x HexDigitsopt. HexDigits
0X HexDigitsopt. HexDigits
BinaryExponent:
BinaryExponentIndicator SignedInteger
BinaryExponentIndicator:
p
P
```

其中 *Sign*、*FloatingPointLiteral*、*HexNumeral*、*HexDigits*、*SignedInteger* 和 *FloatTypeSuffix* 与 Java Language Specification 的词法结构部分中定义的相同。如果 s 不具有 *FloatValue* 的形式，则抛出 NumberFormatException。否则，可以认为 s 是一个以常用的“计算机科学记数法”表示的精确十进制值，或者是一个精确的十六进制值；在概念上，这个精确的数值将被转换为一个“无限精确的”二进制值，然后根据常用的 IEEE 754 浮点算法的“舍入为最接近的数”规则，将该值舍入为 double 类型，其中包括保留 0 值的符号。最后，返回表示此 double 值的 Double 对象。

要解释浮点值的本地化字符串表示形式，请使用 NumberFormat 的子类。

注意，尾部格式说明符、确定浮点字面值类型的说明符（1.0f 是一个 float 值；1.0d 是一个 double 值）*不会* 影响此方法的结果。换句话说，输入字符串的数值被直接转换为目标浮点类型。分两步的转换（先将字符串转换为 float，然后将 float 转换为 double）*不* 同于直接将字符串

转换为 double。例如，float 字面值 0.1f 等于 double 值 0.10000000149011612；float 字面值 0.1f 表示与 double 字面值 0.1 不同的数值。（数值 0.1 无法用二进制浮点数准确表示。）

为了避免因为对无效字符串调用此方法而导致抛出 NumberFormatException，可以使用以下正则表达式作为输入到屏幕的字符串：

```
            final String Digits        = "(\\p{Digit}+)";
    final String HexDigits    = "(\\p{XDigit}+)";
            // an exponent is 'e' or 'E' followed by an optionally
            // .
            final String Exp             = "[eE][+-]?"+Digits;
            final String fpRegex       =
                ("[\\x00-\\x20]*"+            // Optional leading "whitespace"
                 "[+-]?(" +            // Optional sign character
                 "NaN|" +              // "NaN" string
                 "Infinity|" +       // "Infinity" string

                 // A decimal floating-point string representing a finite positive
                 // number without a leading sign has at most five basic pieces:
                 // Digits . Digits ExponentPart FloatTypeSuffix
                 //
                 // Since this method allows integer-only strings as input
                 // in addition to strings of floating-point literals, the
                 // two sub-patterns below are simplifications of the grammar
                 // productions from the Java Language Specification, 2nd
                 // edition, section 3.10.2.

                 // Digits ._opt Digits_opt ExponentPart_opt FloatTypeSuffix_opt
                 "(((("+Digits+"(\\.)?("+Digits+"?)("+Exp+")?)|"+

                 // . Digits ExponentPart_opt FloatTypeSuffix_opt
                 "(\\.("+Digits+")("+Exp+")?)|"+

           // Hexadecimal strings
           "((" +
            // 0[xX] HexDigits ._opt BinaryExponent FloatTypeSuffix_opt
```

```
                "(0[xX]" + HexDigits + "(\\.)?)|" +

                // 0[xX] HexDigits_opt . HexDigits BinaryExponent FloatTypeSuffix_opt
                "(0[xX]" + HexDigits + "?(\\.)" + HexDigits + ")" +

                ")[pP][+-]?" + Digits + "))" +
                    "[fFdD]?))" +
                    "[\\x00-\\x20]*");// Optional trailing "whitespace"

        if (Pattern.matches(fpRegex, myString))
                Double.valueOf(myString);  // Will not throw NumberFormatException
        else {
            // Perform suitable alternative action
        }
```

参数：

s - 要解析的字符串。

返回：

保存用 String 参数表示的值的 Double 对象。

抛出：

NumberFormatException - 如果字符串不包含可解析的数。

---

valueOf

```
public static Double valueOf(double d)
```

返回表示指定的 double 值的 Double 实例。如果不需要新的 Double 实例，则通常应优先使用此方法，而不是构造方法 Double(double)，因为此方法可能通过缓存经常请求的值来显著提高空间和时间性能。

参数：

d - 一个 double 值。

返回：

表示 d 的 Double 实例。

---

parseDouble

```
public static double parseDouble(String s)
                                  throws NumberFormatException
```

返回一个新的 double 值，该值被初始化为用指定 String 表示的值，这与 Double 类的 valueOf 方法一样。

参数：

s - 要解析的字符串。

返回：

由字符串参数表示的 double 值。

抛出：

NumberFormatException - 如果字符串不包含可解析的 double 值。

另请参见：

valueOf(String)

isNaN

```
public static boolean isNaN(double v)
```

如果指定的数是一个 NaN 值，则返回 true；否则返回 false。

参数：

v - 要测试的值。

返回：

如果参数值为 NaN，则返回 true；否则返回 false。

isInfinite

```
public static boolean isInfinite(double v)
```

如果指定数在数值上为无穷大，则返回 true；否则返回 false。

参数：

v - 要测试的值。

返回：

如果参数值是正无穷大或负无穷大，则返回 true；否则返回 false。

isNaN

public boolean isNaN()

如果此 Double 值是非数字（NaN）值，则返回 true；否则返回 false。

返回：

如果此对象表示的值为 NaN，则返回 true；否则返回 false。

isInfinite

public boolean isInfinite()

如果此 Double 值在数值上为无穷大，则返回 true；否则返回 false。

返回：

如果此对象所表示的值是正无穷大或负无穷大，则返回 true；否则返回 false。

toString

public String toString()

返回此 Double 对象的字符串表示形式。使用此对象表示的基本 double 值被转换为一个字符串，这与带一个参数的 toString 方法完全一样。

覆盖：

类 Object 中的 toString

返回：

此对象的 String 表示形式。

另请参见：

toString(double)

byteValue

public byte byteValue()

以 byte 形式返回此 Double 的值（通过强制转换为 byte）。

覆盖：

类 Number 中的 byteValue

返回：

转换为 byte 类型的由此对象所表示的 double 值

shortValue

```
public short shortValue()
```

以 short 形式返回此 Double 的值（通过强制转换为 short）。

覆盖：

类 Number 中的 shortValue

返回：

转换为 short 类型的由此对象所表示的 double 值

intValue

```
public int intValue()
```

以 int 形式返回此 Double 的值（通过强制转换为 int 类型）。

指定者：

类 Number 中的 intValue

返回：

转换为 int 类型的由此对象所表示的 double 值

longValue

```
public long longValue()
```

以 long 形式返回此 Double 的值（通过强制转换为 long 类型）。

指定者：

类 Number 中的 longValue

返回：

转换为 long 类型的由此对象所表示的 double 值

floatValue

```
public float floatValue()
```

返回此 Double 对象的 float 值。

指定者：

类 Number 中的 floatValue

返回：

转换为 float 类型的由此对象所表示的 double 值

doubleValue

```
public double doubleValue()
```

返回此 Double 对象的 double 值。

指定者：

类 Number 中的 doubleValue

返回：

此对象所表示的 double 值

hashCode

```
public int hashCode()
```

返回此 Double 对象的哈希码。结果是此 Double 对象所表示的基本 double 值的 long 整数位表示形式(与 doubleToLongBits(double) 方法生成的结果完全一样）两部分整数之间的异或 (XOR)。也就是说，哈希码就是以下表达式的值：

```
(int)(v^(v>>>32))
```

其中 v 的定义为：

```
long v = Double.doubleToLongBits(this.doubleValue());
```

覆盖：

类 Object 中的 hashCode

返回：

此对象的 hashCode 值。

另请参见：
Object.equals(java.lang.Object),Hashtable

equals

```
public boolean equals(Object obj)
```

将此对象与指定对象比较。当且仅当参数不是 null 而是 Double 对象，且表示的 Double 值与此对象表示的 double 值相同时，结果为 true。为此，当且仅当将方法 doubleToLongBits(double) 应用于两个值所返回的 long 值相同时，才认为这两个 double 值相同。

注意，在大多数情况下，对于 Double 类的两个实例 d1 和 d2，当且仅当

```
d1.doubleValue() == d2.doubleValue()
```

为 true 时，d1.equals(d2) 的值才为 true。但是，有以下两种例外情况：

- 如果 d1 和 d2 都表示 Double.NaN，那么即使 Double.NaN==Double.NaN 值为 false，equals 方法也将返回 true。
- 如果 d1 表示 +0.0 而 d2 表示 -0.0，或者相反，那么即使 +0.0==-0.0 值为 true，equals 测试也将返回 false。

此定义使得哈希表得以正确操作。
覆盖：
类 Object 中的 equals
参数：
obj- 要与之进行比较的对象。
返回：
如果对象相同，则返回 true；否则返回 false。
另请参见：
doubleToLongBits(double)

doubleToLongBits

public static long doubleToLongBits(double value)

根据 IEEE 754 浮点双精度格式 ("double format") 位布局，返回指定浮点值的表示形式。

第 63 位（掩码 0x8000000000000000L 选定的位）表示浮点数的符号。第 62-52 位（掩码 0x7ff0000000000000L 选定的位）表示指数。第 51-0 位（掩码 0x000fffffffffffffL 选定的位）表示浮点数的有效数字（有时也称为尾数）。

如果参数是正无穷大，则结果为 0x7ff0000000000000L。

如果参数是负无穷大，则结果为 0xfff0000000000000L。

如果参数是 NaN，则结果为 0x7ff8000000000000L。

在所有情况下，结果都是一个 long 整数，将其赋予 longBitsToDouble(long) 方法将生成一个与 doubleToLongBits 的参数相同的浮点值（所有 NaN 值被压缩成一个“规范”NaN 值时除外）。

参数：

value - 双精度 (double) 浮点数。

返回：

表示浮点数的位。

doubleToRawLongBits

public static long doubleToRawLongBits(double value)

根据 IEEE 754 浮点“双精度格式”位布局，返回指定浮点值的表示形式，并保留 NaN 值。

第 63 位（掩码 0x8000000000000000L 选定的位）表示浮点数的符号。第 62-52 位（掩码 0x7ff0000000000000L 选定的位）表示指数。第 51-0 位（掩码 0x000fffffffffffffL 选定的位）表示浮点数的有效数字（有时也称为尾数）。

如果参数是正无穷大，则结果为 0x7ff0000000000000L。

如果参数是负无穷大，则结果为 0xfff0000000000000L。

如果参数是 NaN，则结果是表示实际 NaN 值的 long 整数。与 doubleToLongBits 方法不同，doubleToRawLongBits 并没有压缩那些将 NaN 编码为一个“规范的”NaN 值的所有位模式。

在所有情况下，结果都是一个 long 整数，将其赋予 longBitsToDouble(long) 方法将生成一个与 doubleToRawLongBits 的参数相同的浮点值。

参数：

value - 双精度 (double) 浮点数。

返回：

表示浮点数的位。

---

longBitsToDouble

```
public static double longBitsToDouble(long bits)
```

返回对应于给定位表示形式的 double 值。根据 IEEE 754 浮点“双精度格式”位布局，参数被视为浮点值表示形式。

如果参数是 0x7ff0000000000000L，则结果为正无穷大。

如果参数是 0xfff0000000000000L，则结果为负无穷大。

如果参数值在 0x7ff0000000000001L 到 0x7fffffffffffffffL 之间或者在 0xfff0000000000001L 到 0xffffffffffffffffL 之间，则结果为 NaN。Java 提供的任何 IEEE 754 浮点操作都不能区分具有不同位模式的两个同类型 NaN 值。不同的 NaN 值只能使用 Double.doubleToRawLongBits 方法区分。

在所有其他情况下，设 *s*、*e* 和 *m* 为可以通过以下参数计算的三个值：

```
int s = ((bits >> 63) == 0) ? 1 : -1;
int e = (int)((bits >> 52) & 0x7ffL);
```

```
long m = (e == 0) ?
                    (bits & 0xfffffffffffffL) << 1 :
                    (bits & 0xfffffffffffffL) | 0x10000000000000L;
```

那么浮点结果等于算术表达式 $s \cdot m \cdot 2^{e-1075}$ 的值。

注意，此方法不能返回与 long 参数具有完全相同位模式的 double NaN。IEEE 754 区分了两种 NaN：quiet NaN 和 *signaling NaN*。这两种 NaN 之间的差别在 Java 中通常是不可见的。对 signaling NaN 进行的算术运算将它们转换为具有不同（但通常类似）位模式的 quiet NaN。但是在某些处理器上，只复制 signaling NaN 也执行这种转换。特别是在复制 signaling NaN 以将其返回给调用方法时，可能会执行这种转换。因此，longBitsToDouble 可能无法返回具有 signaling NaN 位模式的 double 值。所以，对于某些 long 值，doubleToRawLongBits(longBitsToDouble(start)) 可能*不* 等于 start。此外，尽管所有 NaN 位模式（不管是 quiet NaN 还是 signaling NaN）都必须在上面提到的 NaN 范围内，但表示 signaling NaN 的特定位模式与平台有关。

参数：

bits - 任意 long 整数。

返回：

具有相同位模式的 double 浮点值。

compareTo

```
public int compareTo(Double anotherDouble)
```

对两个 Double 对象所表示的数值进行比较。在应用到基本 double 值时，有两种方法可以比较执行此方法生成的值与执行 Java 语言数字比较运算符（<、<=、== 和 >= >）生成的值之间的区别：

- 此方法认为 Double.NaN 等于它自身，且大于其他所有 double 值（包括 Double.POSITIVE_INFINITY）。
- 此方法认为 0.0d 大于 -0.0d。

这可以确保受此方法影响的 Double 对象的*自然顺序与 equals 一致*。

指定者：

接口 Comparable<Double> 中的 compareTo

参数：

anotherDouble - 要比较的 Double 值。

返回：

如果 anotherDouble 在数字上等于此 Double，则返回 0；如果此 Double 在数字上小于 anotherDouble，则返回小于 0 的值；如果此 Double 在数字上大于此 anotherDouble，则返回大于 0 的值。

compare

```
public static int compare(double d1,
                          double d2)
```

比较两个指定的 double 值。返回整数值的符号与以下调用返回的整数的符号相同：

```
new Double(d1).compareTo(new Double(d2))
```

参数：

d1 - 要比较的第一个 double

d2 - 要比较的第二个 double

返回：

如果 d1 在数字上等于 d2，则返回 0；如果 d1 在数字上小于 d2，则返回小于 0 的值；如果 d1 在数字上大于d2，则返回大于 0 的值。

Float 类

Float 类在对象中包装一个基本类型 float 的值。Float 类型的对象包含一个 float 类型的字段。

此外，此类提供了几种方法，可将 float 类型与 String 类型互相转换，还提供了处理 float 类型时非常有用的其他一些常量和方法。

字段详细信息

POSITIVE_INFINITY

public static final float POSITIVE_INFINITY

保存 float 类型的正无穷大值的常量。它等于 Float.intBitsToFloat(0x7f800000) 返回的值。

另请参见：

常量字段值

NEGATIVE_INFINITY

public static final float NEGATIVE_INFINITY

保存 float 类型的负无穷大值的常量。它等于 Float.intBitsToFloat(0xff800000) 返回的值。

另请参见：

常量字段值

NaN

public static final float NaN

保存 float 类型的非数字 (NaN) 值的常量。它等于 Float.intBitsToFloat(0x7fc00000) 返回的值。

另请参见：

常量字段值

MAX_VALUE

public static final float MAX_VALUE

保存 float 类型的最大正有限值的常量，即 $(2-2^{-23})\cdot 2^{127}$。它等于十六进制的浮点字

面值 0x1.fffffeP+127f，也等于 Float.intBitsToFloat(0x7f7fffff)。

另请参见：

常量字段值

MIN_NORMAL

public static final float MIN_NORMAL

保存 float 类型数据的最小正标准值的常量，即 2^{-126}。它等于十六进制的浮点字面值 0x1.0p-126f，也等于 Float.intBitsToFloat(0x00800000)。

从以下版本开始：

1.6

另请参见：

常量字段值

MIN_VALUE

public static final float MIN_VALUE

保存 float 类型数据的最小正非零值的常量，即 2^{-149}。它等于十六进制的浮点字面值 0x0.000002P-126f，也等于 Float.intBitsToFloat(0x1)。

另请参见：

常量字段值

MAX_EXPONENT

public static final int MAX_EXPONENT

有限 float 变量可能具有的最大指数。它等于 Math.getExponent(Float.MAX_VALUE) 返回的值。

从以下版本开始：

1.6

另请参见：

常量字段值

MIN_EXPONENT

public static final int MIN_EXPONENT

标准化 float 变量可能具有的最小指数。它等于 Math.getExponent(Float.MIN_NORMAL) 返回的值。

从以下版本开始：

1.6

另请参见：

常量字段值

SIZE

public static final int SIZE

表示一个 float 值所使用的位数。

从以下版本开始：

1.5

另请参见：

常量字段值

TYPE

public static final Class<Float> TYPE

表示 float 基本类型的 Class 实例。

从以下版本开始：

JDK1.1

构造方法详细信息

Float

public Float(float value)

构造一个新分配的 Float 对象，它表示基本的 float 参数。

参数：

value - 用 Float 表示的值。

Float

```
public Float(double value)
```

构造一个新分配的 Float 对象，它表示转换为 float 类型的参数。

参数：

value - 用 Float 表示的值。

Float

```
public Float(String s)
        throws NumberFormatException
```

构造一个新分配的 Float 对象，它表示用字符串表示的 float 类型的浮点值。字符串将被转换为 float 值，这与 valueOf 方法一样。

参数：

s - 将转换为 Float 值的字符串。

抛出：

NumberFormatException - 如果字符串不包含可解析的数字。

另请参见：

valueOf(java.lang.String)

方法详细信息

toString

```
public static String toString(float f)
```

返回 float 参数的字符串表示形式。下面提到的所有字符都是 ASCII 字符。

- 如果参数是 NaN，那么结果是字符串 "NaN"。

- 否则，结果是表示参数的符号和数值（绝对值）的字符串。如果符号为负，那么结果的第一个字符是 '-'('\u002D')；如果符号为正，则结果中不显示符号字符。至于数值 *m*：
  - 如果 *m* 为无穷大，则用字符串 "Infinity" 表示；因此，正无穷大生成结果 "Infinity"，负无穷大生成结果 "-Infinity"。
  - 如果 *m* 为 0，则用字符 "0.0" 表示；因此，负 0 生成结果 "-0.0"，正 0 生成结果 "0.0"。
  - 如果 *m* 大于等于 10^{-3}，但小于 10^{7}，则采用不带前导 0 的十进制形式，用 *m* 的整数部分表示，后跟 '.'('\u002E')，再后面是表示 *m* 小数部分的一个或多个十进制位数。
  - 如果 *m* 小于 10^{-3} 或大于等于 10^{7}，则用所谓的"计算机科学记数法"表示。设 *n* 为满足 $10^{n} <= m < 10^{n+1}$ 的唯一整数；然后设 *a* 为 *m* 与 10^{n} 的精确算术商数值，从而 $1 <= a < 10$。那么，数值便表示为 *a* 的整数部分，其形式为：一个十进制位数，后跟 '.'('')，接着是表示 *a* 小数部分的十进制位数，再后面是字母 'E'('\u0045')，最后是用十进制整数形式表示的 *n*，这与方法 Integer.toString(int) 生成的结果一样。

必须为 *m* 或 *a* 的小数部分打印多少位呢？至少必须有一位数来表示小数部分，除此之外，需要更多（但只能和需要的一样多）位数来唯一地区分参数值和 float 类型的邻近值。也就是说，假设 *x* 是用十进制表示法表示的精确算术值，是通过对有限非 0 参数 *f* 调用此方法生成的。那么 *f* 一定是最接近 *x* 的 float 值，如果有两个 float 值同等地接近于 *x*，那么 *f* 必须是这两个值中的一个，并且 *f* 的最低有效位必须是 0。

要创建浮点值的本地化字符串表示形式，请使用 NumberFormat 的子类。

参数：

f - 要转换的浮点型数据。

返回：

参数的字符串表示形式。

---

toHexString

```
public static String toHexString(float f)
```

返回 float 参数的十六进制字符串表示形式。下面提到的所有字符都是 ASCII 字符。

- 如果参数为 NaN，那么结果是字符串 "NaN"。
- 否则，结果是表示参数的符号和数值（绝对值）的字符串。如果符号为负，那么结果的第一个字符是 '-'('\u002D')；如果符号为正，则结果中不显示符号字符。至于数值 *m*：
  - 如果 *m* 为无穷大，则用字符串 "Infinity" 表示；因此，正无穷大生成结果 "Infinity"，负无穷大生成结果 "-Infinity"。
  - 如果 *m* 为 0，则用字符串 "0x0.0p0" 表示；因此，负 0 生成结果 "-0x0.0p0"，正 0 生成结果 "0x0.0p0"。
  - 如果 *m* 是具有标准化表示形式的 float 值，则使用子字符串表示有效位数和指数。有效位数用字符串 "0x1." 表示，后跟该有效位数小数部分的小写十六进制表示形式。除非所有位数都为 0，否则移除十六进制表示形式中的尾部 0，在所有位数为 0 的情况下，可以用一个 0 表示。然后用 "p" 表示指数，后跟无偏指数的十进制字符串，该值与对指数值调用 Integer.toString 生成的值相同。
  - 如果 *m* 是具有 subnormal 表示形式的 float 值，则用字符串 "0x0." 表示有效位数，后跟该有效位数小数部分的十六进制表示形式。移除十六进制表示形式中的尾部 0。然后用 "p-126" 表示指数。注意，在 subnormal 有效位数中，至少必须有一个非 0 位数。

浮点值	十六进制字符串
1.0	0x1.0p0
-1.0	-0x1.0p0
2.0	0x1.0p1
3.0	0x1.8p1
0.5	0x1.0p-1
0.25	0x1.0p-2
Float.MAX_VALUE	0x1.fffffep127
Minimum Normal Value	0x1.0p-126
Maximum Subnormal Value	0x0.fffffep-126
Float.MIN_VALUE	0x0.000002p-126

参数：

f - 要转换的 float 值。

返回：

参数的十六进制字符串表示形式。

从以下版本开始：

1.5

valueOf

```
public static Float valueOf(String s)
                        throws NumberFormatException
```

返回保存用参数字符串 s 表示的 float 值的 Float 对象。

如果 s 为 null，则抛出 NullPointerException 异常。

忽略 s 中的前导空白字符和尾部空白字符。就像调用 String.trim() 方法那样移除空白；也就是说，ASCII 空格和控制字符都要移除。s 的其余部分应该根据词法语法规则描述构成 *FloatValue*：

```
FloatValue:
Signopt NaN
Signopt Infinity
Signopt FloatingPointLiteral
Signopt HexFloatingPointLiteral
SignedInteger
HexFloatingPointLiteral:
HexSignificand BinaryExponent FloatTypeSuffixopt
HexSignificand:
HexNumeral
HexNumeral .
0x HexDigitsopt .HexDigits
0X HexDigitsopt .HexDigits
BinaryExponent:
BinaryExponentIndicator SignedInteger
BinaryExponentIndicator:
p
P
```

其中，*Sign*、*FloatingPointLiteral*、*HexNumeral*、*HexDigits*、*SignedInteger* 和 *FloatTypeSuffix* 与 Java Language Specification 的词法结构部分中的定义相同。如果 s 的表示形式不是 *FloatValue*，则抛出 NumberFormatException。否则，可以认为 s 表示的是常用“计算机科学记数法”表示的精确十进制值，或者是一个精确的十

六进制值；在概念上，这个精确的数值被转换一个“无限精确的”二进制值，然后根据常用 IEEE 754 浮点算法的“舍入为最接近的数”规则将该值舍入为 float 类型，其中包括保留 0 值的符号。最后，返回表示这个 float 值的 Float 对象。

要解释浮点值的本地化字符串表示形式，请使用 NumberFormat 的子类。

注意，尾部格式说明符、确定浮点字面值类型的说明符（1.0f 是一个 float 值；1.0d 是一个 double 值）*不会* 影响此方法的结果。换句话说，输入字符串的数值被直接转换为目标浮点类型。通常，分两步的转换（先将字符串转换为 double 类型，然后将 double 类型转换为 float 类型）*不* 同于直接将字符串转换为 float 类型。例如，如果首先转换为中间类型 double，然后再转换为 float 类型，则字符串
"1.00000017881393421514957253748434595763683319091796875001d"
将得到 float 值 1.0000002f；如果直接将字符串转换为 float 值，则结果将是 1.0000001f。

为了避免对无效字符串调用此方法并导致抛出 NumberFormatException，Double.valueOf 的文档中列出了一个正则表达式，可以用该表达式在屏幕上显示输入。

参数：
s - 要解析的字符串。
返回：
保存用 String 参数表示的值的 Float 对象。
抛出：
NumberFormatException - 如果字符串不包含可解析的数字。

---

valueOf

public static Float valueOf(float f)

返回表示指定的 float 值的 Float 实例。如果不需要新的 Float 实例，则通常应优先使用此方法，而不是构造方法 Float(float)，因为此方法可能通过缓存经常请求的值来显著提高空间和时间性能。

参数：
f - 一个浮点值。
返回：
一个表示 f 的 Float 实例。

从以下版本开始：
1.5

parseFloat

```
public static float parseFloat(String s)
                        throws NumberFormatException
```

返回一个新的 float 值，该值被初始化为用指定 String 表示的值，这与 Float 类的 valueOf 方法一样。
参数：
s - 要解析的字符串。
返回：
用字符串参数表示的 float 值。
抛出：
NumberFormatException - 如果字符串不包含可解析的 float 值。
从以下版本开始：
1.2
另请参见：
valueOf(String)

isNaN

```
public static boolean isNaN(float v)
```

如果指定的数是一个非数字 (NaN) 值，则返回 true，否则返回 false。
参数：
v - 要测试的值。
返回：
如果参数为 NaN，则返回 true；否则返回 false。

isInfinite

```
public static boolean isInfinite(float v)
```

如果指定数的数值是无穷大，则返回 true，否则返回 false。

参数：

v - 要测试的值。

返回：

如果参数是正无穷大或负无穷大，则返回 true；否则返回 false。

isNaN

```
public boolean isNaN()
```

如果此 Float 值是一个非数字 (NaN) 值，则返回 true，否则返回 false。

返回：

如果此对象表示的值是 NaN，则返回 true；否则返回 false。

isInfinite

```
public boolean isInfinite()
```

如果此 Float 值的大小是无穷大，则返回 true，否则返回 false。

返回：

如果此对象表示的值是正无穷大或负无穷大，则返回 true；否则返回 false。

toString

```
public String toString()
```

返回此 Float 对象的字符串表示形式。使用此对象表示的基本 float 值被转换为一个 String，此方法与带一个参数的 toString 方法完全一样。

覆盖：

类 Object 中的 toString

返回：

此对象的 String 表示。

另请参见：

toString(float)

byteValue

```
public byte byteValue()
```

将此 Float 值以 byte 形式返回（强制转换为 byte）。

覆盖：

类 Number 中的 byteValue

返回：

此对象表示的 float 值，该值被转换为 byte 类型

shortValue

```
public short shortValue()
```

将此 Float 值以 short 形式返回（强制转换为 short）。

覆盖：

类 Number 中的 shortValue

返回：

此对象表示的 float 值，该值被转换为 short 类型

从以下版本开始：

JDK1.1

intValue

```
public int intValue()
```

将此 Float 值以 int 形式返回（强制转换为 int 类型）。

指定者：

类 Number 中的 intValue

返回：

此对象表示的 float 值，该值被转换为 int 类型

longValue

public long **longValue**()

将此 Float 值以 long 形式返回（强制转换为 long 类型）。

指定者：

类 Number 中的 longValue

返回：

此对象表示的 float 值，该值被转换为 long 类型

---

floatValue

public float floatValue()

返回此 Float 对象的 float 值。

指定者：

类 Number 中的 floatValue

返回：

此对象表示的 float 值。

---

doubleValue

public double doubleValue()

返回此 Float 对象的 double 值。

指定者：

类 Number 中的 doubleValue

返回：

此对象表示的 float 值被转换为 double 类型，并返回转换的结果。

---

hashCode

public int hashCode()

返回此 Float 对象的哈希码。结果是此 Float 对象表示的基本 float 值的整数位表示形式，与 floatToIntBits(float) 方法生成的结果完全一样。

覆盖：

类 Object 中的 hashCode

返回：
此对象的哈希码值。
另请参见：
Object.equals(java.lang.Object),Hashtable

equals

public boolean equals(Object obj)

将此对象与指定对象进行比较。当且仅当参数不是 null 而是 Float 对象，且表示的 float 值与此对象表示的 float 值相同时，结果为 true。为此，当且仅当将方法 #floatToLongBits(double) 应用于两个值所返回的 int 值相同时，才认为这两个 float 值相同。

注意，在大多数情况下，对于 Float 类的两个实例 f1 和 f2，当且仅当

```
f1.floatValue() == f2.floatValue()
```

的值为 true 时，f1.equals(f2) 的值才为 true。但是，有以下两种例外情况：

- 如果 f1 和 f2 都表示 Float.NaN，那么即使 Float.NaN==Float.NaN 的值为 false，equals 方法也将返回 true。
- 如果 f1 表示 +0.0f，而 f2 表示 -0.0f，或相反，那么即使 0.0f==-0.0f 的值为 true，equal 测试也将返回 false。

该定义使得哈希表得以正确操作。
覆盖：
类 Object 中的 equals
参数：
obj- 要比较的对象
返回：
如果对象是相同的，则返回 true；否则返回 false。
另请参见：
floatToIntBits(float)

floatToIntBits

public static int floatToIntBits(float value)

根据 IEEE 754 浮点“单一格式”位布局，返回指定浮点值的表示形式。

第 31 位（掩码 0x80000000 选定的位）表示浮点数的符号。第 30-23 位（掩码 0x7f800000 选定的位）表示指数。第 22-0 位（掩码 0x007fffff 选定的位）表示浮点数的有效位数（有时也称为尾数）。

如果参数为正无穷大，则结果为 0x7f800000。

如果参数为负无穷大，则结果为 0xff800000。

如果参数为 NaN，则结果为 0x7fc00000。

在所有情况下，结果都是一个整数，将其赋予 intBitsToFloat(int) 方法将生成一个浮点值，该浮点值与 floatToIntBits 的参数相同（而所有 NaN 值则会生成一个“规范”NaN 值）。

参数：

value - 一个浮点数。

返回：

表示浮点数的位。

floatToRawIntBits

public static int floatToRawIntBits(float value)

根据 IEEE 754 浮点“单一格式”位布局，返回指定浮点值的表示形式，并保留非数字 (NaN) 值。

第 31 位（掩码 0x80000000 选定的位）表示浮点数的符号。第 30-23 位（掩码 0x7f800000 选定的位）表示指数。第 22-0 位（掩码 0x007fffff 选定的位）表示浮点数的有效位数（有时也称为尾数）。

如果参数为正无穷大，则结果为 0x7f800000。

如果参数为负无穷大，则结果为 0xff800000。

如果参数为 NaN，则结果是表示实际 NaN 值的整数。与 floatToIntBits 方法不同，floatToRawIntBits 不压缩所有将 NaN 编码为一个“规范”NaN 值的位模式。

在所有情况下，结果都是一个整数，将其赋予 intBitsToFloat(int) 方法将生成一个与 floatToRawIntBits 的参数相同的浮点值。

参数：
value - 一个浮点数字。
返回：
表示浮点数字的位。
从以下版本开始：
1.3

intBitsToFloat

public static float intBitsToFloat(int bits)

返回对应于给定位表示形式的 float 值。根据 IEEE 754 浮点“单一格式”位布局，该参数被视为浮点值表示形式。

如果参数为 0x7f800000，则结果为正无穷大。

如果参数为 0xff800000，则结果为负无穷大。

如果参数值在 0x7f800001 到 0x7fffffff 或在 0xff800001 到 0xffffffff 之间，则结果为 NaN。Java 提供的任何 IEEE 754 浮点操作都不能区分具有不同位模式的两个同类型 NaN 值。不同的 NaN 值只能使用 Float.floatToRawIntBits 方法区分。

在所有其他情况下，设 *s*、*e* 和 *m* 为可以通过以下参数进行计算的三个值：

```
int s = ((bits >> 31) == 0) ? 1 : -1;
int e = ((bits >> 23) & 0xff);
int m = (e == 0) ?
            (bits & 0x7fffff) << 1 :
```

```
(bits & 0x7fffff) | 0x800000;
```

那么浮点结果等于算术表达式 $s \cdot m \cdot 2^{e-150}$ 的值。

注意，此方法不能返回与 int 参数具有完全相同位模式的 float NaN 值。IEEE 754 区分了两种 NaN：quiet NaN 和 *signaling NaN*。这两种 NaN 之间的差别在 Java 中通常是不可见的。对 signaling NaN 进行的算术运算将它们转换为具有不同（但通常类似）位模式的 quiet NaN。但在某些只复制 signaling NaN 的处理器上也执行这种转换。特别是在复制 signaling NaN 以将其返回给调用方法时，可能会执行这种转换。因此，intBitsToFloat 可能无法返回具有 signaling NaN 位模式的 float 值。所以，对于某些 int 值，floatToRawIntBits(intBitsToFloat(start)) 可能*不* 等于 start。此外，尽管所有 NaN 位模式（不管是 quiet NaN 还是 signaling NaN）都必须在前面确定的 NaN 范围内，但表示 signaling NaN 的特定位模式是与平台有关。

参数：

bits - 一个整数。

返回：

具有相同位模式的 float 浮点值。

---

compareTo

public int compareTo(Float anotherFloat)

比较两个 Float 对象所表示的数值。在应用到基本 float 值时，有两种方法可以比较执行此方法产生的值与执行 Java 语言的数字比较运算符（<、<=、== 和 >= >）产生的那些值之间的区别：

- 此方法认为 Float.NaN 等于其自身，且大于其他所有 float 值（包括 Float.POSITIVE_INFINITY）。
- 此方法认为 0.0f 大于 -0.0f。

这可以确保受此方法影响的 Float 对象的*自然顺序与 equals 一致*。

指定者：

接口 Comparable<Float> 中的 compareTo

参数：

anotherFloat - 要比较的 Float 值。

返回：

如果 anotherFloat 在数字上等于此 Float，则返回 0；如果 anotherFloat 在数字上小于此 Float，则返回小于 0 的值；如果 anotherFloat 在数字上大于此 Float，则返回大于 0 的值。

从以下版本开始：

1.2

另请参见：

Comparable.compareTo(Object)

compare

```
public static int compare(float f1,
                          float f2)
```

比较两个指定的 float 值。返回整数值的符号与以下调用返回整数的符号相同：

```
new Float(f1).compareTo(new Float(f2))
```

参数：

f1 - 要比较的第一个 float 值。

f1 - 要比较的第二个 float 值。

返回：

如果 f1 在数字上等于此 f2，则返回值为 0；如果 f1 在数字上小于此 f2，则返回小于 0 的值；如果 f1 在数字上大于此 f2，则返回大于 0 的值。

从以下版本开始：

1.4

Integer 类

Integer 类在对象中包装了一个基本类型 int 的值。Integer 类型的对象包含一个 int 类型的字段。

此外，该类提供了多个方法，能在 int 类型和 String 类型之间互相转换，还提供了处理 int 类型时非常有用的其他一些常量和方法。

字段详细信息

MIN_VALUE

public static final int MIN_VALUE

值为 -2^{31} 的常量，它表示 int 类型能够表示的最小值。

另请参见：

常量字段值

MAX_VALUE

public static final int **MAX_VALUE**

值为 $2^{31}-1$ 的常量，它表示 int 类型能够表示的最大值。

另请参见：

常量字段值

TYPE

public static final Class<Integer> TYPE

表示基本类型 int 的 Class 实例。

从以下版本开始：

JDK1.1

SIZE

public static final int SIZE

用来以二进制补码形式表示 int 值的比特位数。

从以下版本开始：

1.5

另请参见：

常量字段值

构造方法详细信息

Integer

```
public Integer(int value)
```

构造一个新分配的 Integer 对象，它表示指定的 int 值。

参数：

value - Integer 对象表示的值。

Integer

```
public Integer(String s)
            throws NumberFormatException
```

构造一个新分配的 Integer 对象，它表示 String 参数所指示的 int 值。使用与 parseInt 方法（对基数为 10 的值）相同的方式将该字符串转换成 int 值。

参数：

s - 要转换为 Integer 的 String。

抛出：

NumberFormatException - 如果 String 不包含可解析的整数。

另请参见：

parseInt(java.lang.String, int)

方法详细信息

toString

```
public static String toString(int i,
                              int radix)
```

返回用第二个参数指定基数表示的第一个参数的字符串表示形式。

如果基数小于 Character.MIN_RADIX 或者大于 Character.MAX_RADIX，则改用基数 10。

如果第一个参数为负，则结果中的第一个元素为 ASCII 的减号 '-' ('\u002D')。如果第一个参数为非负，则没有符号字符出现在结果中。

结果中的剩余字符表示第一个参数的大小。如果大小为零，则用一个零字符 '0' ('\u0030') 表示；否则，大小的表示形式中的第一个字符将不是零字符。用以下 ASCII 字符作为数字：

```
0123456789abcdefghijklmnopqrstuvwxyz
```

其范围是从 '\u0030' 到 '\u0039' 和从 '\u0061' 到 '\u007A'。如果 radix 为 *N*, 则按照所示顺序，使用这些字符中的前 *N* 个作为其数字。因此，十六进制（基数为 16）的数字是 0123456789abcdef。如果希望得到大写字母，则可以在结果上调用 String.toUpperCase() 方法：

```
Integer.toString(n, 16).toUpperCase()
```

参数：

i - 要转换成字符串的整数。

radix - 用于字符串表示形式的基数。

返回：

使用指定基数的参数的字符串表示形式。

另请参见：

Character.MAX_RADIX,Character.MIN_RADIX

---

toHexString

```
public static String toHexString(int i)
```

以十六进制（基数 16）无符号整数形式返回一个整数参数的字符串表示形式。

如果参数为负，那么无符号整数值为参数加上 2^{32}；否则等于该参数。将该值转换为十六进制（基数 16）的无前导 0 的 ASCII 数字字符串。如果无符号数的大小值为零，则用一个零字符 '0' ('\u0030') 表示它；否则，无符号数大小的表示形式中的第一个字符将不是零字符。用以下字符作为十六进制数字：

```
0123456789abcdef
```

这些字符的范围是从 '\u0030' 到 '\u0039' 和从 '\u0061' 到 '\u0066'。如果希望得到大写字母，可以在结果上调用 String.toUpperCase() 方法：

```
Integer.toHexString(n).toUpperCase()
```

参数：

i - 要转换成字符串的整数。

返回：

参数的十六进制（基数 16）无符号整数值的字符串表示形式。

从以下版本开始：

JDK1.0.2

---

toOctalString

```
public static String toOctalString(int i)
```

以八进制（基数 8）无符号整数形式返回一个整数参数的字符串表示形式。

如果参数为负，该无符号整数值为参数加上 2^{32}；否则等于该参数。该值被转换成八进制（基数 8）ASCII 数字的字符串，且没有附加前导 0。

如果无符号数大小为零，则用一个零字符 '0' ('\u0030') 表示它；否则，无符号数大小的表示形式中的第一个字符将不是零字符。用以下字符作为八进制数字：

```
01234567
```

它们是从 '\u0030' 到 '\u0037' 的字符。

参数：

i - 要转换成字符串的整数。

返回：

用八进制参数（基数 8）表示的无符号整数值的字符串表示形式。

从以下版本开始：

JDK1.0.2

---

toBinaryString

```
public static String toBinaryString(int i)
```

以二进制（基数 2）无符号整数形式返回一个整数参数的字符串表示形式。

如果参数为负，该无符号整数值为参数加上 2^{32}；否则等于该参数。将该值转换为二进制（基数 2）形式的无前导 0 的 ASCII 数字字符串。如果无符号数的大小为零，则用一个零字符 '0'（'\u0030'）表示它；否则，无符号数大小的表示形式中的第一个字符将不是零字符。字符 '0'（'\u0030'）和 '1'（'\u0031'）被用作二进制数字。

参数：

i - 要转换为字符串的整数。

返回：

用二进制（基数 2）参数表示的无符号整数值的字符串表示形式。

从以下版本开始：

JDK1.0.2

---

toString

```
public static String toString(int i)
```

返回一个表示指定整数的 String 对象。将该参数转换为有符号的十进制表示形式，以字符串形式返回它，就好像将参数和基数 10 作为参数赋予 toString(int, int) 方法。

参数：

i - 要转换的整数。

返回：

十进制（基数 10）参数的字符串表示形式。

---

parseInt

```
public static int parseInt(String s,
                           int radix)
                    throws NumberFormatException
```

使用第二个参数指定的基数，将字符串参数解析为有符号的整数。除了第一个字符可以是用来表示负值的 ASCII 减号 '-'('\u002D')外，字符串中的字符必须都是指

定基数的数字（通过 Character.digit(char, int) 是否返回一个负值确定）。返回得到的整数值。

如果发生以下任意一种情况，则抛出一个 NumberFormatException 类型的异常：

- 第一个参数为 null 或一个长度为零的字符串。
- 基数小于 Character.MIN_RADIX 或者大于 Character.MAX_RADIX。
- 假如字符串的长度超过 1，那么除了第一个字符可以是减号 '-'('u002D') 外，字符串中存在任意不是由指定基数的数字表示的字符。
- 字符串表示的值不是 int 类型的值。

示例：

```
parseInt("0", 10) 返回 0
parseInt("473", 10) 返回 473
parseInt("-0", 10) 返回 0
parseInt("-FF", 16) 返回 -255
parseInt("1100110", 2) 返回 102
parseInt("2147483647", 10) 返回 2147483647
parseInt("-2147483648", 10) 返回 -2147483648
parseInt("2147483648", 10) 抛出 NumberFormatException
parseInt("99", 8) 抛出 NumberFormatException
parseInt("Kona", 10) 抛出 NumberFormatException
parseInt("Kona", 27) 返回 411787
```

参数：

s - 包含要解析的整数表示形式的 String

radix - 解析 s 时使用的基数。

返回：

使用指定基数的字符串参数表示的整数。

抛出：

NumberFormatException - 如果 String 不包含可解析的 int。

parseInt

```
public static int parseInt(String s)
```

```
throws NumberFormatException
```

将字符串参数作为有符号的十进制整数进行解析。除了第一个字符可以是用来表示负值的 ASCII 减号 '-' ('\u002D') 外，字符串中的字符都必须是十进制数字。返回得到的整数值，就好像将该参数和基数 10 作为参数赋予 parseInt(java.lang.String, int) 方法一样。

参数：

s - 包含要解析的 int 表示形式的 String。

返回：

用十进制参数表示的整数值。

抛出：

NumberFormatException - 如果字符串不包含可解析的整数。

valueOf

```
public static Integer valueOf(String s,
                              int radix)
                       throws NumberFormatException
```

返回一个 Integer 对象，该对象中保存了用第二个参数提供的基数进行解析时从指定的 String 中提取的值。将第一个参数解释为用第二个参数指定的基数表示的有符号整数，就好像将这些参数赋予 parseInt(java.lang.String, int) 方法一样。结果是一个表示字符串指定的整数值的 Integer 对象。

换句话说，该方法返回一个等于以下值的 Integer 对象：

```
new Integer(Integer.parseInt(s, radix))
```

参数：

s - 要解析的字符串。

radix - 解释 s 时使用的基数。

返回：

一个 Integer 对象，它含有字符串参数（以指定的基数）所表示的数值。

抛出：

NumberFormatException - 如果 String 不包含可解析的 int。

valueOf

public static Integer valueOf(String s)
throws NumberFormatException

返回保存指定的 String 的值的 Integer 对象。将该参数解释为表示一个有符号的十进制整数，就好像将该参数赋予 parseInt(java.lang.String) 方法一样。结果是一个表示字符串指定的整数值的 Integer 对象。

换句话说，该方法返回一个等于以下值的 Integer 对象：

new Integer(Integer.parseInt(s))

参数：

s - 要解析的字符串。

返回：

保存字符串参数表示的值的 Integer 对象。

抛出：

NumberFormatException - 如果字符串不能解析为一个整数。

valueOf

public static Integer valueOf(int i)

返回一个表示指定的 int 值的 Integer 实例。如果不需要新的 Integer 实例，则通常应优先使用该方法，而不是构造方法 Integer(int)，因为该方法有可能通过缓存经常请求的值而显著提高空间和时间性能。

参数：

i - 一个 int 值。

返回：

表示 i 的 Integer 实例。

从以下版本开始：

1.5

byteValue

public byte byteValue()

以 byte 类型返回该 Integer 的值。

覆盖：

类 Number 中的 byteValue

返回：

转换为 byte 类型后该对象表示的数值。

---

shortValue

```
public short shortValue()
```

以 short 类型返回该 Integer 的值。

覆盖：

类 Number 中的 shortValue

返回：

转换为 short 类型后该对象表示的数值。

---

intValue

```
public int intValue()
```

以 int 类型返回该 Integer 的值。

指定者：

类 Number 中的 intValue

返回：

转换为 int 类型后该对象表示的数值。

---

longValue

```
public long longValue()
```

以 long 类型返回该 Integer 的值。

指定者：

类 Number 中的 longValue

返回：

转换为 long 类型后该对象表示的数值。

---

floatValue

```
public float floatValue()
```

以 float 类型返回该 Integer 的值。

指定者：

类 Number 中的 floatValue

返回：

转换为 float 类型后该对象表示的数值。

doubleValue

```
public double doubleValue()
```

以 double 类型返回该 Integer 的值。

指定者：

类 Number 中的 doubleValue

返回：

转换为 double 类型后该对象表示的数值。

toString

```
public String toString()
```

返回一个表示该 Integer 值的 String 对象。将该参数转换为有符号的十进制表示形式，并以字符串的形式返回它，就好像将该整数值作为参数赋予 toString(int) 方法一样。

覆盖：

类 Object 中的 toString

返回：

该对象的值（基数 10）的字符串表示形式。

hashCode

public int **hashCode**()

返回此 Integer 的哈希码。

覆盖：

类 Object 中的 hashCode

返回：

该对象的哈希码值，它的值即为该 Integer 对象表示的基本 int 类型的数值。

另请参见：

Object.equals(java.lang.Object),Hashtable

---

equals

public boolean equals(Object obj)

比较此对象与指定对象。当且仅当参数不为 null，并且是一个与该对象包含相同 int 值的 Integer 对象时，结果为 true。

覆盖：

类 Object 中的 equals

参数：

obj- 要比较的对象。

返回：

如果对象相同，则返回 true，否则返回 false。

另请参见：

Object.hashCode(),Hashtable

---

getInteger

public static Integer getInteger(String nm)

确定具有指定名称的系统属性的整数值。

第一个参数被视为系统属性的名称。通过 System.getProperty(java.lang.String) 方法可以访问系统属性。然后，将该属性的字符串值解释为一个整数值，并返回表示该值的 Integer 对象。使用 getProperty 的定义可以找到可能出现的数字格式的详细信息。

如果没有具有指定名称的属性，或者指定名称为空或 null，或者属性的数字格式不正确，则返回 null。

换句话说，该方法返回一个等于以下值的 Integer 对象：

getInteger(nm, null)

参数：

nm - 属性名。

返回：

属性的 Integer 值。

另请参见：

System.getProperty(java.lang.String),
System.getProperty(java.lang.String, java.lang.String)

getInteger

```
public static Integer getInteger(String nm,
                                 int val)
```

确定具有指定名称的系统属性的整数值。

第一个参数被视为系统属性的名称。通过 System.getProperty(java.lang.String) 方法可以访问系统属性。然后，将该属性的字符串值解释为一个整数值，并返回表示该值的 Integer 对象。使用 getProperty 的定义可以找到可能出现的数字格式的详细信息。

第二个参数是默认值。如果未具有指定名称的属性，或者属性的数字格式不正确，或者指定名称为空或 null，则返回一个表示第二个参数的值的 Integer 对象。

换句话说，该方法返回一个等于以下值的 Integer 对象：

```
getInteger(nm, new Integer(val))
```

但在实践中可能会用以下类似方式实现它：

```
Integer result = getInteger(nm, null);
return (result == null) ? new Integer(val) : result;
```

从而避免在无需默认值时分配不必要的 Integer 对象。

参数：

nm - 属性名。

val - 默认值。

返回：

属性的 Integer 值。

另请参见：

System.getProperty(java.lang.String),
System.getProperty(java.lang.String, java.lang.String)

---

getInteger

```
public static Integer getInteger(String nm,
                                 Integer val)
```

返回具有指定名称的系统属性的整数值。第一个参数被视为系统属性的名称。通过 System.getProperty(java.lang.String) 方法可以访问系统属性。然后，根据每个 Integer.decode 方法，将该属性的字符串值解释为一个整数值，并返回一个表示该值的 Integer 对象。

- 如果属性值以两个 ASCII 字符 0x 或者 ASCII 字符 # 开始，并且后面没有减号，则将它的剩余部分解析为十六进制整数，就好像以 16 为基数调用 valueOf(java.lang.String, int) 方法一样。
- 如果属性值以 ASCII 字符 0 开始，后面还有其他字符，则将它解析为八进制整数，就好像以 8 为基数调用 valueOf(java.lang.String, int) 方法一样。
- 否则，将属性值解析为十进制整数，就好像以 10 为基数调用 valueOf(java.lang.String, int) 方法一样。

第二个参数是默认值。如果未具有指定名称的属性，或者属性的数字格式不正确，或者指定名称为空或 null，则返回默认值。

参数：

nm - 属性名。

val - 默认值。

返回：

属性的 Integer 值。

另请参见：
System.getProperty(java.lang.String),
System.getProperty(java.lang.String, java.lang.String),
decode(java.lang.String)

decode

```
public static Integer decode(String nm)
                             throws NumberFormatException
```

将 String 解码为 Integer。接受通过以下语法给出的十进制、十六进制和八进制数字：
DecodableString:
$Sign_{opt}$ *DecimalNumeral*
$Sign_{opt}$ 0x *HexDigits*
$Sign_{opt}$ 0X *HexDigits*
$Sign_{opt}$ # *HexDigits*
$Sign_{opt}$ 0 *OctalDigits*
Sign:
-

Java Language Specification 的第 §3.10.1 节中有 *DecimalNumeral*、*HexDigits* 和 *OctalDigits* 的定义。

跟在（可选）负号和/或基数说明符（“0x”、“0X”、“#”或前导零）后面的字符序列是使用指示的基数(10、16 或 8)通过 Integer.parseInt 方法解析的。字符序列必须表示一个正值，否则会抛出 NumberFormatException。如果指定的 String 的第一个字符是减号，则对结果求反。String 中不允许出现空白字符。

参数：
nm - 要解码的 String。
返回：
保存 nm 所表示的 int 值的 Integer 对象。
抛出：
NumberFormatException - 如果 String 不包含可解析整数。
从以下版本开始：
1.2

另请参见：
parseInt(java.lang.String, int)

compareTo

public int compareTo(Integer anotherInteger)

在数字上比较两个 Integer 对象。
指定者：
接口 Comparable<Integer> 中的 compareTo
参数：
anotherInteger - 要比较的 Integer。
返回：
如果该 Integer 等于 Integer 参数，则返回 0 值；如果该 Integer 在数字上小于 Integer 参数，则返回小于 0 的值；如果 Integer 在数字上大于 Integer 参数，则返回大于 0 的值（有符号的比较）。
从以下版本开始：
1.2

highestOneBit

public static int highestOneBit(int i)

返回具有至多单个 1 位的 int 值，在指定的 int 值中最高位（最左边）的 1 位的位置。如果指定的值在其二进制补码表示形式中不具有 1 位，即它等于零，则返回零。
返回：
返回具有单个 1 位的 int 值，在指定值中最高位的 1 位的位置，否则，如果指定值本身等于零，则返回零。
从以下版本开始：
1.5

lowestOneBit

public static int lowestOneBit(int i)

返回具有至多单个 1 位的 int 值，在指定的 int 值中最低位（最右边）的 1 位的位置。如果指定的值在其二进制补码表示形式中不具有 1 位，即它等于零，则返回零。

返回：

返回具有单个 1 位的 int 值，在指定值中最低位的 1 位的位置，否则，如果指定值本身等于零，则返回零。

从以下版本开始：

1.5

---

numberOfLeadingZeros

public static int numberOfLeadingZeros(int i)

在指定 int 值的二进制补码表示形式中最高位（最左边）的 1 位之前，返回零位的数量。如果指定值在其二进制补码表示形式中不存在 1 位，换句话说，如果它等于零，则返回 32。

注意，此方法与基数为 2 的对数密切相关。对于所有的正 int 值 x：

- floor($\log_2(x)$) = 31 - numberOfLeadingZeros(x)
- ceil($\log_2(x)$) = 32 - numberOfLeadingZeros(x - 1)

返回：

返回在指定 int 值的二进制补码表示形式中最高位（最左边）的 1 位之前的零位的数量；否则，如果该值等于零，则返回 32。

从以下版本开始：

1.5

---

numberOfTrailingZeros

public static int numberOfTrailingZeros(int i)

返回指定的 int 值的二进制补码表示形式中最低（“最右边”）的为 1 的位后面的零位个数。如果指定值在它的二进制补码表示形式中没有为 1 的位，即它的值为零，则返回 32。

返回：

返回在指定 int 值的二进制补码表示形式中最低位（最右边）的 1 位之后零位的数量；否则，如果该值等于零，则返回 32。

从以下版本开始：

1.5

---

bitCount

```
public static int bitCount(int i)
```

返回指定 int 值的二进制补码表示形式的 1 位的数量。此函数有时用于*人口普查*。

返回：

返回指定 int 值的二进制补码表示形式的 1 位的数量。

从以下版本开始：

1.5

---

rotateLeft

```
public static int rotateLeft(int i,
                             int distance)
```

返回根据指定的位数循环左移指定的 int 值的二进制补码表示形式而得到的值。（位是从左边（即高位）移出，从右边（即低位）再进入）

注意，使用负距离的左循环等同于右循环：rotateLeft(val, -distance) == rotateRight(val, distance)。还要注意的是，以 32 的任何倍数进行的循环都是无操作指令，因此，即使距离为负，除了最后五位外，其余所有循环距离都可以忽略：rotateLeft(val, distance) == rotateLeft(val, distance & 0x1F)。

返回：

返回根据指定的位数循环左移指定的 int 值的二进制补码表示形式而得到的值。

从以下版本开始：

1.5

---

rotateRight

```
public static int rotateRight(int i,
                              int distance)
```

返回根据指定的位数循环右移指定的 int 值的二进制补码表示形式而得到的值。（位是从右边（即低位）移出，从左边（即高位）再进入）

注意，使用负距离的右循环等同于左循环：rotateRight(val, -distance) == rotateLeft(val, distance)。还要注意的是，以 32 的任何倍数进行的循环都是无操作指令，因此，即使距离为负，除了最后五位外，其余所有循环距离都可以忽略：rotateRight(val, distance) == rotateRight(val, distance & 0x1F)。

返回：

返回根据指定的位数循环右移指定的 int 值的二进制补码表示形式而得到的值。

从以下版本开始：

1.5

---

reverse

```
public static int reverse(int i)
```

返回通过反转指定 int 值的二进制补码表示形式中位的顺序而获得的值。

返回：

返回通过反转指定 int 值中位的顺序而获得的值。

从以下版本开始：

1.5

---

signum

```
public static int signum(int i)
```

返回指定 int 值的符号函数。（如果指定值为负，则返回 －1；如果指定值为零，则返回 0；如果指定的值为正，则返回 1。）

返回：

返回指定 int 值的符号函数。

从以下版本开始：

1.5

---

reverseBytes

public static int reverseBytes(int i)

返回通过反转指定 int 值的二进制补码表示形式中字节的顺序而获得的值。

返回：

返回通过反转指定 int 值的字节而获得的值。

从以下版本开始：

1.5

Long 类

Long 类在对象中包装了基本类型 long 的值。每个 Long 类型的对象都包含一个 long 类型的字段。

此外，该类提供了多个方法，可以将 long 转换为 String，将 String 转换为 long，除此之外，还提供了其他一些处理 long 时有用的常量和方法。

字段详细信息

MIN_VALUE

public static final long MIN_VALUE

保持 long 类型的最小值的常量，该值为 -2^{63}。

另请参见：

常量字段值

MAX_VALUE

public static final long MAX_VALUE

保持 long 类型的最大值的常量，该值为 $2^{63}-1$。

另请参见：

常量字段值

TYPE

```
public static final Class<Long> TYPE
```

表示基本类型 long 的 Class 实例。

从以下版本开始：

JDK1.1

SIZE

```
public static final int SIZE
```

用来以二进制补码形式表示 long 值的位数。

构造方法详细信息

Long

```
public Long(long value)
```

构造新分配的 Long 对象，表示指定的 long 参数。

参数：

value - Long 对象表示的值。

Long

```
public Long(String s)
        throws NumberFormatException
```

构造新分配的 Long 对象，表示由 String 参数指示的 long 值。该字符串被转换为 long 值，其方式与 radix 参数为 10 的 parseLong 方法所使用的方式一致。

参数：

s - 要转换为 Long 的 String。

抛出：

NumberFormatException - 如果 String 不包含可解析的 long。

方法详细信息

toString

```
public static String toString(long i,
                              int radix)
```

返回在使用第二个参数指定的基数时第一个参数的字符串表示形式。

如果该基数小于 Character.MIN_RADIX，或大于 Character.MAX_RADIX，则使用基数 10。

如果第一个参数是负数，则结果的第一个元素是 ASCII 字符的减号 '-' ('\u002d')。如果第一个参数非负，则结果中不会出现符号字符。

结果的其余字符表示第一个参数的大小。如果大小为零，则用单个零字符 '0' 表示它 ('\u0030')；否则大小表示形式中的第一个字符将不是零字符。以下 ASCII 字符均被用作数字：

```
0123456789abcdefghijklmnopqrstuvwxyz
```

这些是从 '\u0030' 到 '\u0039' 和从 '\u0061' 到 '\u007a' 的字符。如果 radix 是 *N*，则这些字符的第一个 *N* 用作显示顺序中基数 *N* 的数字。因此，该数字的十六进制（基数 16）表示形式为 0123456789abcdef。如果需要使用大写字母，则可以在结果上调用 String.toUpperCase() 方法：

```
Long.toString(n, 16).toUpperCase()
```

参数：

i - 要转换为字符串的 long。

radix - 将在字符串表示形式中使用的基数。

返回：

指定基数中参数的字符串表示形式。

另请参见：

Character.MAX_RADIX,Character.MIN_RADIX

toHexString

public static String toHexString(long i)

以十六进制无符号整数形式返回 long 参数的字符串表示形式。

如果参数为负，则无符号 long 值为该参数加上 2^{64}；否则，它等于该参数。此值将被转换为不带附加前导 0 的十六进制（基数 16）ASCII 数字字符串。如果无符号大小为零，则该数字由单个零字符 '0' 表示 ('\u0030')；否则，无符号大小表示形式中的第一个字符将不是零字符。下列字符都被用作十六进制数字：

```
0123456789abcdef
```

这些是从 '\u0030' 到 '\u0039' 和从 '\u0061' 到 '\u0066' 的字符。如果需要使用大写字母，则可以在结果上调用 String.toUpperCase() 方法：

```
Long.toHexString(n).toUpperCase()
```

参数：

i - 要转换为字符串的 long。

返回：

十六进制（基数 16）参数表示的无符号 long 值的字符串表示形式。

从以下版本开始：

JDK 1.0.2

toOctalString

public static String toOctalString(long i)

以八进制无符号整数形式返回 long 参数的字符串表示形式。

如果参数为负，则无符号 long 值为该参数加上 2^{64}；否则，它等于该参数。此值将被转换为不带附加前导 0 的八进制（基数 8）ASCII 数字字符串。

如果无符号大小为零，则该数字用单个零字符 '0' ('\u0030') 表示，否则无符号大小表示形式中的第一个字符将不是零字符。以下字符都用作八进制数字：

```
01234567
```

这些是从 '\u0030' 到 '\u0037' 的字符。

参数：

i - 要转换为字符串的 long。

返回：

八进制（基数 8）参数表示的无符号 long 值的字符串表示形式。

从以下版本开始：

JDK 1.0.2

---

toBinaryString

```
public static String toBinaryString(long i)
```

以二进制无符号整数形式返回 long 参数的字符串表示形式。

如果参数为负数，则无符号 long 值为该参数加上 2^{64}；否则，它等于该参数。此值将被转换为不带附加前导 0 的二进制（基数 2）ASCII 数字字符串。如果无符号大小为零，则用单个零字符 '0' 表示它 ('\u0030')；否则，无符号大小表示形式中的第一个字符将不是零字符。字符 '0' ('\u0030') 和 '1' ('\u0031') 被用作二进制位。

参数：

i - 要转换为字符串的 long。

返回：

二进制 (base 2) 参数表示的无符号 long 值的字符串表示形式。

从以下版本开始：

JDK 1.0.2

---

toString

```
public static String toString(long i)
```

返回表示指定 long 的 String 对象。该参数被转换为有符号的十进制表示形式，并作为字符串返回，该字符串与用该参数和基数 10 作为参数的 toString(long, int) 方法所得到的值非常相似。

参数：
i - 要转换的 long。
返回：
十进制参数的字符串表示形式。

parseLong

```
public static long parseLong(String s,
                             int radix)
                      throws NumberFormatException
```

将 string 参数解析为有符号的 long，基数由第二个参数指定。字符串中的字符必须为指定基数中的数字（由 Character.digit(char, int) 是否返回一个非负值来确定），除非第一个字符为 ASCII 字符的减号 '-' ('\u002D')，它表示一个负值。返回得到的 long 值。

注意，不允许将字符 L ('\u004C') 和 l ('\u006C') 作为类型指示符出现在字符串的结尾处，而这一点在 Java 编程语言源代码中是允许的——除非 L 或 l 以大于 22 的基数形式出现。

如果出现以下情形之一，则抛出 NumberFormatException 类型的异常：

- 第一个参数是 null 或零长度的字符串。
- radix 小于 Character.MIN_RADIX 或者大于 Character.MAX_RADIX。
- 任何字符串的字符都不是指定基数的数字，除非第一个字符是减号 '-' ('\u002d')，假定字符串的长度大于 1。
- 字符串表示的值不是 long 类型的值。

示例：

```
parseLong("0", 10) returns 0L
parseLong("473", 10) returns 473L
parseLong("-0", 10) returns 0L
parseLong("-FF", 16) returns -255L
parseLong("1100110", 2) returns 102L
parseLong("99", 8) returns NumberFormatException
parseLong("Hazelnut", 10) returns NumberFormatException
parseLong("Hazelnut", 36) returns 1356099454469L
```

参数：

s - 包含要解析的 long 表示形式的 String。

radix - 将在解析 s 时使用的基数。

返回：

由指定基数中的字符串参数表示的 long。

抛出：

NumberFormatException - 如果字符串不包含可解析的 long。

parseLong

```
public static long parseLong(String s)
                      throws NumberFormatException
```

将 string 参数解析为有符号十进制 long。字符串中的字符必须都是十进制数字，除非第一个字符是 ASCII 字符的减号 '-' (\u002D')，它表示一个负值。返回得到的 long 值，该值与用该参数和基数 10 作为参数的 parseLong(java.lang.String, int) 方法得到的值非常相似。

注意，不允许将字符 L ('\u004C') 和 l ('\u006C') 作为类型指示符出现在字符串的结尾处，这一点在 Java 编程语言源代码中是允许的。

参数：

s - 包含要解析的 long 表示形式的 String

返回：

十进制参数表示的 long。

抛出：

NumberFormatException - 如果字符串不包含可解析的 long。

valueOf

```
public static Long valueOf(String s,
                           int radix)
                    throws NumberFormatException
```

当使用第二个参数给出的基数进行解析时，返回保持从指定 String 中提取的值的 Long 对象。第一个参数被解释为有符号的 long，基数由第二个参数指定，该值与

用该参数作为参数的 parseLong(java.lang.String, int) 方法得到的值非常类似。结果是表示字符串指定的 long 值的 Long 对象。

换句话说，此方法返回一个 Long 对象，它的值等于：

new Long(Long.parseLong(s, radix))

参数：

s - 要解析的字符串

radix - 将在解释 s 时使用的基数

返回：

保持由指定基数中的字符串参数表示的值的 Long 对象。

抛出：

NumberFormatException - 如果 String 不包含可解析的 long。

valueOf

```
public static Long valueOf(String s)
                    throws NumberFormatException
```

返回保持指定 String 的值的 Long 对象。该参数被解释为表示一个有符号的十进制 long，该值与用该参数作为参数的 parseLong(java.lang.String) 方法得到的值非常相似。结果是表示由字符串指定的整数值的 Long 对象。

换句话说，此方法返回一个 Long 对象，它的值等于：

```
new Long(Long.parseLong(s))
```

参数：

s - 要解析的字符串。

返回：

包含由字符串参数表示的值的 Long 对象。

抛出：

NumberFormatException - 如果不能将字符串解析为 long。

valueOf

public static Long valueOf(long l)

返回表示指定 long 值的 Long 实例。如果不需要新的 Long 实例，则通常优先使用此方法，而不是使用构造方法 Long(long)，因为此方法通过缓存频繁请求的值，可以显著提高时间和空间性能。

参数：

l - long 值。

返回：

表示 l 的 Long 实例。

从以下版本开始：

1.5

decode

public static Long decode(String nm)
throws NumberFormatException

将 String 解码成 Long。接受通过以下语法给出的十进制、十六进制和八进制数：

DecodableString:

$Sign_{opt}$ *DecimalNumeral*

$Sign_{opt}$ 0x *HexDigits*

$Sign_{opt}$ 0X *HexDigits*

$Sign_{opt}$ # *HexDigits*

$Sign_{opt}$ 0 *OctalDigits*

Sign:

-

DecimalNumeral、*HexDigits* 和 *OctalDigits* 在 Java Language Specification 中的 §3.10.1 中已经定义。

跟在（可选）负号和/或基数说明符（"0x"、"0X"、"#" 或前导零）后面的字符的顺序由 Long.parseLong 方法通过指示的基数（10、16 或 8）来解析。字符的顺序必须表示为一个正值，否则将抛出 NumberFormatException。如果指定 String 的第一个字符是减号，则结果无效。String 中不允许出现空白字符。

参数：

nm - 要解码的 String。

返回：

保持由 nm 表示的 long 值的 Long 对象
抛出：
NumberFormatException - 如果 String 不包含可解析的 long。
从以下版本开始：
1.2
另请参见：
parseLong(String, int)

byteValue

```
public byte byteValue()
```

以 byte 形式返回此 Long 的值。
覆盖：
类 Number 中的 byteValue
返回：
转换为 byte 类型后该对象表示的数值。

shortValue

```
public short shortValue()
```

以 short 形式返回此 Long 的值。
覆盖：
类 Number 中的 shortValue
返回：
转换为 short 类型后该对象表示的数值。

intValue

```
public int intValue()
```

以 int 形式返回此 Long 的值。
指定者：
类 Number 中的 intValue

返回：

转换为 int 类型后该对象表示的数值。

longValue

```
public long longValue()
```

以 long 值的形式返回此 Long 的值。

指定者：

类 Number 中的 longValue

返回：

转换为 long 类型后该对象表示的数值。

floatValue

```
public float floatValue()
```

以 float 形式返回此 Long 的值。

指定者：

类 Number 中的 floatValue

返回：

转换为 float 类型后该对象表示的数值。

doubleValue

```
public double doubleValue()
```

以 double 形式返回此 Long 的值。

指定者：

类 Number 中的 doubleValue

返回：

转换为 double 类型后该对象表示的数值。

toString

public String toString()

返回表示 Long 值的 String 对象。该值被转换为有符号十进制表示形式，并作为字符串返回，该字符串与用 long 值作为参数的 toString(long) 方法得到的字符串非常相似。

覆盖：

类 Object 中的 toString

返回：

十进制对象值的字符串表示形式。

hashCode

public int hashCode()

返回 Long 的哈希码。结果是此 Long 对象保持的基本 long 值的两个部分的异或(XOR)。也就是说，哈希码就是表达式的值：

```
(int)(this.longValue()^(this.longValue()>>>32))
```

覆盖：

类 Object 中的 hashCode

返回：

此对象的哈希码值。

另请参见：

Object.equals(java.lang.Object),Hashtable

equals

public boolean equals(Object obj)

将此对象与指定对象进行比较。当且仅当该参数不是 null，且 Long 对象与此对象包含相同的 long 值时，结果才为 true。

覆盖：

类 Object 中的 equals

参数：

obj - 要与之进行比较的对象。

返回：

如果对象相同，则返回 true；否则，返回 false。

另请参见：

Object.hashCode(),Hashtable

---

getLong

```
public static Long getLong(String nm)
```

确定具有指定名称的系统属性的 long 值。

第一个参数被视为系统属性的名称。通过 System.getProperty(java.lang.String) 方法可以访问该系统属性。然后，以 long 值的形式解释此属性的字符串值，并返回表示此值的 Long 对象。在 getProperty 的定义中可以找到可能的数字格式的详细信息。

如果指定名称没有属性，或者指定名称为空或 null，抑或属性不具有正确的数字格式时，则返回 null。

换句话说，此方法返回一个 Long 对象，它的值等于：

```
getLong(nm, null)
```

参数：

nm - 属性名。

返回：

属性的 Long 值。

另请参见：

System.getProperty(java.lang.String),
System.getProperty(java.lang.String, java.lang.String)

---

getLong

```
public static Long getLong(String nm,
                           long val)
```

使用指定名称确定系统属性的 long 值。

第一个参数被视为系统属性的名称。通过 System.getProperty(java.lang.String) 方法可以访问该系统属性。然后，以 long 值的形式解释此属性的字符串值，并返回表示此值的 Long 对象。在 getProperty 的定义中可以找到可能的数字格式的详细信息。

第二个参数是默认值。如果指定的名称没有属性，或者该属性不具备正确的数字格式，抑或指定名称为空或 null，则返回表示第二个参数的值的 Long 对象。

换句话说，此方法返回一个 Long 对象，它的值等于：

```
getLong(nm, new Long(val))
```

但是实际上，它可能通过以下方式实现：

```
Long result = getLong(nm, null);
return (result == null) ? new Long(val) :result;
```

这样可以避免不需要默认值时进行的不必要的 Long 对象分配。

参数：

nm - 属性名。

val - 默认值。

返回：

属性的 Long 值。

另请参见：

System.getProperty(java.lang.String),
System.getProperty(java.lang.String, java.lang.String)

getLong

```
public static Long getLong(String nm,
                           Long val)
```

使用指定名称返回系统属性的 long 值。第一个参数被视为系统属性的名称。通过 System.getProperty(java.lang.String) 方法可以访问该系统属性。然后，以 long 值的形式解释此属性的字符串值，并且按照 Long.decode 方法返回表示此值的 Long 对象。

- 如果该属性值以两个 ASCII 字符 0x 或 ASCII 字符 # 开头，后面没有跟减号，则将该属性的其余部分解析为一个十六进制整数，该值与调用参数 radix 为 16 的 valueOf(java.lang.String, int) 方法得到的值非常相似。
- 如果该属性值以 ASCII 字符 0 开头，后跟别的字符，则将它解析为一个八进制整数，该值与调用参数 radix 为 8 的 valueOf(java.lang.String, int) 方法得到的值非常相似。
- 否则，将属性值解析为一个十进制整数，该值与调用参数 radix 为 10 的 valueOf(java.lang.String, int) 方法得到的值非常相似。

注意，在所有情况下，都不允许将 L ('\u004C') 和 l ('\u006C') 作为类型指示符出现在属性值的结尾处，这一点在 Java 编程语言源代码中是允许的。

第二个参数是默认值。如果指定的名称没有属性，或者属性不具有正确的数字格式，抑或指定名称为空或 null，则返回默认值。

参数：
nm - 属性名。
val - 默认值。
返回：
属性的 Long 值。
另请参见：
System.getProperty(java.lang.String),
System.getProperty(java.lang.String, java.lang.String),
decode(java.lang.String)

compareTo

public int compareTo(Long anotherLong)

在数字上比较两个 Long 对象。
指定者：
接口 Comparable<Long> 中的 compareTo
参数：
anotherLong - 要比较的 Long。
返回：
如果 Long 等于参数 Long，则返回 0 值；如果 Long 在数字上小于参数 Long，

则返回小于 0 的值；如果 Long 在数字上大于参数 Long，则返回大于 0 的值（有符号比较）。

从以下版本开始：

1.2

---

highestOneBit

```
public static long highestOneBit(long i)
```

返回至多有一个 1 位的 long 值，在指定的 long 值中最高位（最左边）的 1 位的位置。如果指定值在其二进制补码表示形式中没有 1 位，即等于零，则返回零。

返回：

返回具有单个 1 位的 long 值，在指定值中最高位的 1 位的位置；否则，如果指定值本身等于零，则返回零。

从以下版本开始：

1.5

---

lowestOneBit

```
public static long lowestOneBit(long i)
```

返回至多有一个 1 位的 long 值，在指定的 long 值中最低位（最右边）的 1 位的位置。如果指定值在其二进制补码表示形式中没有 1 位，即等于零，则返回零。

返回：

返回具有单个 1 位的 long 值，在指定值中最低位的 1 位的位置；否则，如果指定值本身等于零，则返回零。

从以下版本开始：

1.5

---

numberOfLeadingZeros

```
public static int numberOfLeadingZeros(long i)
```

在指定 long 值的二进制补码表示形式中最高位（最左边）的 1 位之前，返回零位的数量。如果指定值在其二进制补码表示形式中不存在 1 位，换句话说，如果它等

于零，则返回 64。

注意，此方法与二进制对数密切相关。对于所有的正 long 值 x：

```
floor(log2(x)) = 63 - numberOfLeadingZeros(x)
ceil(log2(x)) = 64 - numberOfLeadingZeros(x - 1)
```

返回：

返回在指定 long 值的二进制补码表示形式中最高位（最左边）的 1 位之前的零位的数量；否则，如果该值等于零，则返回 64。

从以下版本开始：

1.5

---

numberOfTrailingZeros

```
public static int numberOfTrailingZeros(long i)
```

返回在指定 long 值的二进制补码表示形式中最低位（最右边）的 1 位之后的零位的数量。如果指定值在其二进制补码表示形式中不存在 1 位，换句话说，如果它等于零，则返回 64。

返回：

返回在指定 long 值的二进制补码表示形式中最低位（最右边）的 1 位之后零位的数量；否则，如果该值等于零，则返回 64。

从以下版本开始：

1.5

---

bitCount

```
public static int bitCount(long i)
```

返回指定 long 值的二进制补码表示形式中的 1 位的数量。此功能有时被称为*填充计算*。

返回：

返回指定 long 值的二进制补码表示形式的 1 位的数量。

从以下版本开始：

1.5

---

rotateLeft

```
public static long rotateLeft(long i,
                              int distance)
```

返回根据指定的位数循环左移指定的 long 值的二进制补码表示形式而得到的值。（位是从左边（即高位）移出，从右边（即低位）再进入）

注意，使用负距离的左循环等同于右循环：rotateLeft(val, -distance) == rotateRight(val, distance)。另请注意，使用 64 的倍数循环无效，因此除了最后六位，所有循环距离都可以忽略，即使距离是负值：

```
rotateLeft(val, distance) == rotateLeft(val, distance & 0x3F)。
```

返回：

返回根据指定的位数循环左移指定的 long 值的二进制补码表示形式而得到的值。

从以下版本开始：

1.5

rotateRight

```
public static long rotateRight(long i,
                               int distance)
```

返回根据指定的位数循环右移指定的 long 值的二进制补码表示形式而得到的值。（位是从右边（即低位）移出，从左边（即高位）再进入）

注意，使用负距离右循环等于左循环：rotateRight(val, -distance) == rotateLeft(val, distance)。另请注意，使用 64 的倍数循环无效，因此除了最后六位，所有循环距离都可以忽略，即使距离是负值：rotateRight(val, distance) == rotateRight(val, distance & 0x3F)。

返回：

返回根据指定的位数循环右移指定的 long 值的二进制补码表示形式而得到的值。

从以下版本开始：

1.5

reverse

public static long reverse(long i)

返回通过反转指定 long 值的二进制补码表示形式中位的顺序而获得的值。

返回：

返回通过反转指定 long 值中位的顺序而获得的值。

从以下版本开始：

1.5

---

signum

public static int signum(long i)

返回指定 long 值的符号函数。（如果指定值为负，则返回值 -1；如果指定值为零，则返回 0；如果指定值为正，则返回 1。）

返回：

返回指定 long 值的符号函数。

从以下版本开始：

1.5

---

reverseBytes

public static long reverseBytes(long i)

返回通过反转指定 long 值的二进制补码表示形式中字节的顺序而获得的值。

返回：

返回通过反转指定 long 值中的字节而获得的值。

Math 类

Math 类包含用于执行基本数学运算的方法，如初等指数、对数、平方根和三角函数。

与 StrictMath 类的某些数学方法不同，并非 Math 类所有等价函数的实现都定义为返回逐位相同的结果。此类在不需要严格重复的地方可以得到更好的执行。

默认情况下，很多 Math 方法仅调用 StrictMath 中的等价方法来完成它们的实现。建议代码生成器使用特定于平台的本机库或者微处理器指令（可用时）来提供 Math 方法更高性能的实现。这种更高性能的实现仍然必须遵守 Math 的规范。

实现规范的质量涉及到两种属性，即返回结果的准确性和方法的单调性。浮点 Math 方法的准确性根据 ulp（units in the last place，最后一位的进退位）来衡量。对于给定的浮点格式，特定实数值的 ulp 是包括该数值的两个浮点值的差。当作为一个整体而不是针对具体参数讨论方法的准确性时，引入的 ulp 数用于任何参数最差情况下的误差。如果一个方法的误差总是小于 0.5 ulp，那么该方法始终返回最接近准确结果的浮点数；这种方法就是正确舍入。一个正确舍入的方法通常能得到最佳的浮点近似值；然而，对于许多浮点方法，进行正确舍入有些不切实际。相反，对于 Math 类，某些方法允许误差在 1 或 2 ulp 的范围内。非正式地，对于 1 ulp 的误差范围，当准确结果是可表示的数值时，应该按照计算结果返回准确结果；否则，返回包括准确结果的两个浮点值中的一个。对于值很大的准确结果，括号的一端可以是无穷大。除了个别参数的准确性之外，维护不同参数的方法之间的正确关系也很重要。因此，大多数误差大于 0.5 ulp 的方法都要求是半单调的：只要数学函数是非递减的，浮点近似值就是非递减的；同样，只要数学函数是非递增的，浮点近似值就是非递增的。并非所有准确性为 1 ulp 的近似值都能自动满足单调性要求。

字段详细信息

E

```
public static final double E
```

比任何其他值都更接近 *e*（即自然对数的底数）的 double 值。

另请参见：

常量字段值

PI

```
public static final double PI
```

比任何其他值都更接近 *pi*（即圆的周长与直径之比）的 double 值。

方法详细信息

sin

public static double sin(double a)

返回角的三角正弦。特殊情况如下：

- 如果参数为 NaN 或无穷大，那么结果为 NaN。
- 如果参数为 0，那么结果为 0，符号与参数符号相同。

计算结果必须在准确结果的 1 ulp 范围内。结果必须具有半单调性。

参数：
a - 以弧度表示的角。
返回：
参数的正弦。

---

cos

public static double cos(double a)

返回角的三角余弦。特殊情况如下：

- 如果参数为 NaN 或无穷大，那么结果为 NaN。

计算结果必须在准确结果的 1 ulp 范围内。结果必须具有半单调性。

参数：
a - 以弧度表示的角。
返回：
参数的余弦。

---

tan

public static double tan(double a)

返回角的三角正切。特殊情况如下：

- 如果参数为 NaN 或无穷大，那么结果为 NaN。
- 如果参数为 0，那么结果为 0，符号与参数符号相同。

计算结果必须在准确结果的 1 ulp 范围内。结果必须具有半单调性。

参数：
a - 以弧度表示的角。
返回：
参数的正切。

---

asin

```
public static double asin(double a)
```

返回一个值的反正弦；返回的角度范围在 -*pi*/2 到 *pi*/2 之间。特殊情况如下：

- 如果参数为 NaN 或它的绝对值大于 1，那么结果为 NaN。
- 如果参数为 0，那么结果为 0，符号与参数符号相同。

计算结果必须在准确结果的 1 ulp 范围内。结果必须具有半单调性。

参数：
a - 要返回其反正弦的值。
返回：
参数的反正弦。

---

acos

```
public static double acos(double a)
```

返回一个值的反余弦；返回的角度范围在 0.0 到 *pi* 之间。特殊情况如下：

- 如果参数为 NaN 或它的绝对值大于 1，那么结果为 NaN。

计算结果必须在准确结果的 1 ulp 范围内。结果必须具有半单调性。

参数：
a - 要返回其反余弦的值。
返回：
参数的反余弦。

---

atan

public static double atan(double a)

返回一个值的反正切；返回的角度范围在 -*pi*/2 到 *pi*/2 之间。特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为 0，那么结果为 0，符号与参数符号相同。

计算结果必须在准确结果的 1 ulp 范围内。结果必须具有半单调性。

参数：
a - 要返回其反正切的值。
返回：
参数的反正切。

---

toRadians

public static double toRadians(double angdeg)

将用角度表示的角转换为近似相等的用弧度表示的角。从角度到弧度的转换通常是不精确的。
参数：
angdeg - 用角度表示的角
返回：
角 angrad 用弧度表示的值。
从以下版本开始：
1.2

---

toDegrees

public static double toDegrees(double angrad)

将用弧度表示的角转换为近似相等的用角度表示的角。从弧度到角度的转换通常是不精确的；用户*不*应该期望 cos(toRadians(90.0)) 与 0.0 完全相等。
参数：
angrad - 用弧度表示的角。

返回：
角 angrad 用角度表示的值。
从以下版本开始：
1.2

---

exp

```
public static double exp(double a)
```

返回欧拉数 *e* 的 double 次幂的值。特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为正无穷大，那么结果为正无穷大。
- 如果参数为负无穷大，那么结果为正 0。

计算结果必须在准确结果的 1 ulp 范围内。结果必须具有半单调性。

参数：
a - *e* 的指数。
返回：
值 e^a，其中 *e* 是自然对数的底数。

---

log

```
public static double log(double a)
```

返回 double 值的自然对数（底数是 *e*）。特殊情况如下：

- 如果参数为 NaN 或小于 0，那么结果为 NaN。
- 如果参数为正无穷大，那么结果为正无穷大。
- 如果参数为正 0 或负 0，那么结果为负无穷大。

计算结果必须在准确结果的 1 ulp 范围内。结果必须具有半单调性。

参数：
a - 一个值
返回：
ln a 的值，即 a 的自然对数。

---

log10

public static double log10(double a)

返回 double 值的底数为 10 的对数。特殊情况如下：

- 如果参数为 NaN 或小于 0，那么结果为 NaN。
- 如果参数为正无穷大，那么结果为正无穷大。
- 如果参数为正 0 或负 0，那么结果为负无穷大。
- 如果参数等于 10^n（n 为整数），那么结果为 n。

计算结果必须在准确结果的 1 ulp 范围内。结果必须具有半单调性。

参数：
a - 一个值
返回：
a 的底数为 10 的对数。
从以下版本开始：
1.5

---

sqrt

public static double sqrt(double a)

返回正确舍入的 double 值的正平方根。特殊情况如下：

- 如果参数为 NaN 或小于 0，那么结果为 NaN。
- 如果参数为正无穷大，那么结果为正无穷大。
- 如果参数为正 0 或负 0，那么结果与参数相同。

否则，结果为最接近该参数值的实际数学平方根的 double 值。
参数：
a - 一个值。
返回：
a 的正平方根。如果参数为 NaN 或小于 0，那么结果为 NaN。

---

cbrt

public static double cbrt(double a)

返回 double 值的立方根。对于正的有限值 x，cbrt(-x) == -cbrt(x)；也就是说，负值的立方根是该值数值的负立方根。特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为无穷大，那么结果为无穷大，符号与参数符号相同。
- 如果参数为 0，那么结果为 0，符号与参数符号相同。

计算结果必须在准确结果的 1 ulp 范围内。

参数：
a - 一个值。
返回：
a 的立方根。
从以下版本开始：
1.5

IEEEremainder

```
public static double IEEEremainder(double f1,
                                   double f2)
```

按照 IEEE 754 标准的规定，对两个参数进行余数运算。余数的算术值等于 f1 - f2 × *n*，其中 *n* 是最接近商 f1/f2 准确算术值的整数，如果两个整数都同样接近 f1/f2，那么 *n* 是其中的偶数。如果余数是 0，那么它的符号与第一个参数的符号相同。特殊情况如下：

- 如果两个参数都为 NaN，或者第一个参数为无穷大，或者第二个参数为正 0 或负 0，那么结果为 NaN。
- 如果第一个参数为有限值，第二个参数为无穷大，那么结果与第一个参数相同。

参数：
f1 - 被除数。
f2 - 除数。
返回：
f1 除以 f2 的余数。

---

ceil

public static double ceil(double a)

返回最小的（最接近负无穷大）double 值，该值大于等于参数，并等于某个整数。特殊情况如下：

- 如果参数值已经等于某个整数，那么结果与该参数相同。
- 如果参数为 NaN、无穷大、正 0 或负 0，那么结果与参数相同。
- 如果参数值小于 0，但是大于 -1.0，那么结果为负 0。

注意，Math.ceil(x) 的值与 -Math.floor(-x) 的值完全相同。

参数：

a - 一个值。

返回：

最小（最接近负无穷大）浮点值，该值大于等于该参数，并等于某个整数。

floor

public static double floor(double a)

返回最大的（最接近正无穷大）double 值，该值小于等于参数，并等于某个整数。特殊情况如下：

- 如果参数值已经等于某个整数，那么结果与该参数相同。
- 如果参数为 NaN、无穷大、正 0 或负 0，那么结果与参数相同。

参数：

a - 一个值。

返回：

最大（最接近正无穷大）浮点值，该值小于等于该参数，并等于某个整数。

rint

public static double rint(double a)

返回最接近参数并等于某一整数的 double 值。如果两个同为整数的 double 值都同样接近，那么结果取偶数。特殊情况如下：

- 如果参数值已经是整数，那么结果与参数相同。
- 如果参数为 NaN、无穷大、正 0 或负 0，那么结果与参数相同。

参数：
a - double 值。
返回：
最接近 a 的整数浮点值。

atan2

```
public static double atan2(double y,
                           double x)
```

将矩形坐标 (x, y) 转换成极坐标 (r, *theta*)，返回所得角 *theta*。该方法通过计算 y/x 的反正切值来计算相角 *theta*，范围为从 -*pi* 到 *pi*。特殊情况如下：

- 如果任一参数为 NaN，那么结果为 NaN。
- 如果第一个参数为正 0，第二个参数为正数；或者第一个参数为正的有限值，第二个参数为正无穷大，那么结果为正 0。
- 如果第一个参数为负 0，第二个参数为正数；或者第一个参数为负的有限值，第二个参数为正无穷大，那么结果为负 0。
- 如果第一个参数为正 0，第二个参数为负数；或者第一个参数为正的有限值，第二个参数为负无穷大，那么结果为最接近 *pi* 的 double 值。
- 如果第一个参数为负 0，第二个参数为负数；或者第一个参数为负的有限值，第二个参数为负无穷大，那么结果为最接近 *pi* 的 double 值。
- 如果第一个参数为正数，第二个参数为正 0 或负 0；或者第一个参数为正无穷大，第二个参数为有限值，那么结果为最接近 *pi*/2 的 double 值。
- 如果第一个参数为负数，第二个参数为正 0 或负 0；或者第一个参数为负无穷大，第二个参数为有限值，那么结果为最接近 -*pi*/2 的 double 值。
- 如果两个参数都为正无穷大，那么结果为最接近 *pi*/4 的 double 值。
- 如果第一个参数为正无穷大，第二个参数为负无穷大，那么结果为最接近 3*pi*/4 的 double 值。
- 如果第一个参数为负无穷大，第二个参数为正无穷大，那么结果为最接近 -*pi*/4 的 double 值。

- 如果两个参数都为负无穷大，那么结果为最接近 -3*pi/4 的 double 值。

计算结果必须在准确结果的 2 ulp 范围内。结果必须具有半单调性。

参数：
y - 纵坐标
x - 横坐标
返回：
与笛卡儿坐标中点（*x*, *y*）对应的极坐标中点（*r*, *theta*）的 *theta* 组件。

---

pow

```
public static double pow(double a,
                         double b)
```

返回第一个参数的第二个参数次幂的值。特殊情况如下：

- 如果第二个参数为正 0 或负 0，那么结果为 1.0。
- 如果第二个参数为 1.0，那么结果与第一个参数相同。
- 如果第二个参数为 NaN，那么结果为 NaN。
- 如果第一个参数为 NaN，第二个参数非 0，那么结果为 NaN。
- 如果
  - 第一个参数的绝对值大于 1，并且第二个参数为正无穷大，或者
  - 第一个参数的绝对值小于 1，并且第二个参数为负无穷大，

  那么结果为正无穷大。

- 如果
  - 第一个参数的绝对值大于 1，并且第二个参数为负无穷大，或者
  - 第一个参数的绝对值小于 1，并且第二个参数为正无穷大，

  那么结果为正 0。

- 如果第一个参数的绝对值等于 1，并且第二个参数为无穷大，那么结果为 NaN。

- 如果
  - 第一个参数为正 0，并且第二个参数大于 0，或者
  - 第一个参数为正无穷大，并且第二个参数小于 0，

  那么结果为正 0。

- 如果
  - 第一个参数为正 0，并且第二个参数小于 0，或者
  - 第一个参数为正无穷大，并且第二个参数大于 0，

  那么结果为正无穷大。

- 如果
  - 如果第一个参数为负 0，并且第二个参数大于 0 但不是有限的奇数整数，或者
  - 第一个参数为负无穷大，并且第二个参数小于 0 但不是有限的奇数整数，

  那么结果为正 0。

- 如果
  - 第一个参数为负 0，并且第二个参数为正的有限奇数整数，或者
  - 第一个参数为负无穷大，并且第二个参数为负的有限奇数整数，

  那么结果为负 0。

- 如果
  - 如果第一个参数为负 0，并且第二个参数小于 0 但不是有限的奇数整数，或者
  - 第一个参数为负无穷大，并且第二个参数大于 0 但不是有限的奇数整数，

  那么结果为正无穷大。

- 如果
  - 第一个参数为负 0，并且第二个参数为负的有限奇数整数，或者

- 第一个参数为负无穷大，并且第二个参数为正的有限奇数整数，

那么结果为负无穷大。

- 如果第一个参数为小于 0 的有限值，
  - 如果第二个参数为有限的偶数整数，那么结果等于第一个参数绝对值的第二个参数次幂的结果。
  - 如果第二个参数为有限的奇数整数，那么结果等于负的第一个参数绝对值的第二个参数次幂的结果。
  - 如果第二个参数为有限的非整数值，那么结果为 NaN。
- 如果两个参数都为整数，并且结果恰好可以表示为一个 double 值，那么该结果恰好等于第一个参数的第二个参数次幂的算术结果。

（在前面的描述中，当且仅当浮点数为有限值并且是方法 ceil 的定点数，或者是方法 floor 的定点数时，才可以认为浮点值是整数。当且仅当将某个单参数方法应用到某个值的结果等于该值时，该值才是这个方法的定点值。）

计算结果必须在准确结果的 1 ulp 范围内。结果必须具有半单调性。

参数：
a - 底数。
b - 指数。
返回：
值 a^b。

---

round

```
public static int round(float a)
```

返回最接近参数的 int。结果将舍入为整数：加上 1/2，对结果调用 floor 并将所得结果强制转换为 int 类型。换句话说，结果等于以下表达式的值：

```
(int)Math.floor(a + 0.5f)
```

特殊情况如下：

- 如果参数为 NaN，那么结果为 0。

- 如果结果为负无穷大或任何小于等于 Integer.MIN_VALUE 的值，那么结果等于 Integer.MIN_VALUE 的值。
- 如果参数为正无穷大或任何大于等于 Integer.MAX_VALUE 的值，那么结果等于 Integer.MAX_VALUE 的值。

参数：
a - 要舍入为整数的浮点值。
返回：
舍入为最接近的 int 值的参数值。
另请参见：
Integer.MAX_VALUE, Integer.MIN_VALUE

---

round

```
public static long round(double a)
```

返回最接近参数的 long。结果将舍入为整数：加上 1/2，对结果调用 floor 并将所得结果强制转换为 long 类型。换句话说，结果等于以下表达式的值：

```
(long)Math.floor(a + 0.5d)
```

特殊情况如下：

- 如果参数为 NaN，那么结果为 0。
- 如果结果为负无穷大或任何小于等于 Long.MIN_VALUE 的值，那么结果等于 Long.MIN_VALUE 的值。
- 如果参数为正无穷大或任何大于等于 Long.MAX_VALUE 的值，那么结果等于 Long.MAX_VALUE 的值。

参数：
a - 舍入为 long 的浮点值。
返回：
舍入为最接近的 long 值的参数值。
另请参见：
Long.MAX_VALUE, Long.MIN_VALUE

---

random

public static double random()

返回带正号的 double 值，该值大于等于 0.0 且小于 1.0。返回值是一个伪随机选择的数，在该范围内（近似）均匀分布。

第一次调用该方法时，它将创建一个新的伪随机数生成器，与以下表达式完全相同

new java.util.Random

之后，新的伪随机数生成器可用于此方法的所有调用，但不能用于其他地方。

此方法是完全同步的，可允许多个线程使用而不出现错误。但是，如果许多线程需要以极高的速率生成伪随机数，那么这可能会减少每个线程对拥有自己伪随机数生成器的争用。

返回：
大于等于 0.0 且小于 1.0 的伪随机 double 值。
另请参见：
Random.nextDouble()

---

abs

public static int abs(int a)

返回 int 值的绝对值。如果参数为非负数，则返回该参数。如果参数为负数，则返回该参数的相反数。

注意，如果参数等于 Integer.MIN_VALUE 的值（即能够表示的最小负 int 值），那么结果与该值相同且为负。

参数：
a - 要确定绝对值的参数。
返回：
参数的绝对值。
另请参见：
Integer.MIN_VALUE

---

abs

```
public static long abs(long a)
```

返回 long 值的绝对值。如果参数为非负数，则返回该参数。如果参数为负数，则返回该参数的相反数。

注意，如果参数等于 Long.MIN_VALUE 的值（即能够表示的最小负 long 值），则结果与该值相同且为负。

参数：
a - 要确定绝对值的参数。
返回：
参数的绝对值。
另请参见：
Long.MIN_VALUE

---

abs

```
public static float abs(float a)
```

返回 float 值的绝对值。如果参数为非负数，则返回该参数。如果参数为负数，则返回该参数的相反数。特殊情况如下：

- 如果参数为正 0 或负 0，那么结果为正 0。
- 如果参数为无穷大，那么结果为正无穷大。
- 如果参数为 NaN，那么结果为 NaN。

换句话说，结果等于以下表达式的值：

```
Float.intBitsToFloat(0x7fffffff & Float.floatToIntBits(a))
```

参数：
a - 要确定绝对值的参数。
返回：
参数的绝对值。

---

abs

```
public static double abs(double a)
```

返回 double 值的绝对值。如果参数为非负数，则返回该参数。如果参数为负数，则返回该参数的相反数。特殊情况如下：

- 如果参数为正 0 或负 0，那么结果为正 0。
- 如果参数为无穷大，那么结果为正无穷大。
- 如果参数为 NaN，那么结果为 NaN。

换句话说，结果等于以下表达式的值：

```
Double.longBitsToDouble((Double.doubleToLongBits(a)<<1)>>>1)
```

参数：

a - 要确定绝对值的参数。

返回：

参数的绝对值。

---

max

```
public static int max(int a,
                      int b)
```

返回两个 int 值中较大的一个。也就是说，结果为更接近 Integer.MAX_VALUE 值的参数。如果参数值相同，那么结果也是同一个值。

参数：

a - 参数。

b - 另一个参数。

返回：

a 和 b 中的较大者。

另请参见：

Long.MAX_VALUE

---

max

```
public static long max(long a,
                       long b)
```

返回两个 long 值中较大的一个。也就是说，结果为更接近 Long.MAX_VALUE 值的参数。如果参数值相同，那么结果也是同一个值。

参数：

a - 参数。

b - 另一个参数。

返回：

a 和 b 中的较大者。

另请参见：
Long.MAX_VALUE

max

```
public static float max(float a,
                        float b)
```

返回两个 float 值中较大的一个。也就是说，结果为更接近正无穷大的参数。如果参数值相同，那么结果也是同一个值。如果任一值为 NaN，那么结果为 NaN。与数值比较运算不同，此方法认为负 0 严格小于正 0。如果一个参数为正 0，另一个参数为负 0，那么结果为正 0。

参数：
a - 参数。
b - 另一个参数。

返回：
a 和 b 中的较大者。

max

```
public static double max(double a,
                         double b)
```

返回两个 double 值中较大的一个。也就是说，结果为更接近正无穷大的参数。如果参数值相同，那么结果也是同一个值。如果任一值为 NaN，那么结果为 NaN。与数值比较运算不同，该方法认为负 0 严格小于正 0。如果一个参数为正 0，另一个参数为负 0，那么结果为正 0。

参数：
a - 参数。
b - 另一个参数。

返回：
a 和 b 中的较大者。

min

```
public static int min(int a,
                      int b)
```

返回两个 int 值中较小的一个。也就是说，结果为更接近 Integer.MIN_VALUE 值的参数。如果参数值相同，那么结果也是同一个值。

参数：

a - 参数。

b - 另一个参数。

返回：

a 和 b 中的较小者。

另请参见：

Long.MIN_VALUE

---

min

```
public static long min(long a,
                       long b)
```

返回两个 long 值中较小的一个。也就是说，结果为更接近 Long.MIN_VALUE 值的参数。如果参数值相同，那么结果也是同一个值。

参数：

a - 参数。

b - 另一个参数。

返回：

a 和 b 中的较小者。

另请参见：

Long.MIN_VALUE

---

min

```
public static float min(float a,
                        float b)
```

返回两个 float 值中较小的一个。也就是说，结果为更接近负无穷大的值。如果参数值相同，那么结果也是同一个值。如果任一值为 NaN，那么结果为 NaN。与数值比较运算不同，该方法认为负 0 严格小于正 0。如果一个参数为正 0，另一个参数为负 0，那么结果为负 0。

参数：

a - 参数。

b - 另一个参数。

返回：

a 和 b 中的较小者。

min

```
public static double min(double a,
                         double b)
```

返回两个 double 值中较小的一个。也就是说，结果为更接近负无穷大的值。如果参数值相同，那么结果也是同一个值。如果任一值为 NaN，那么结果为 NaN。与数值比较运算不同，该方法认为负 0 严格小于正 0。如果一个参数为正 0，另一个参数为负 0，那么结果为负 0。

参数：

a - 参数。

b - 另一个参数。

返回：

a 和 b 中的较小者。

ulp

```
public static double ulp(double d)
```

返回参数的 ulp 大小。double 值的 ulp 是此浮点值与下一个数值较大的 double 值之间的正距离。注意，对于非 NaN x，ulp(-x) == ulp(x)。

特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为正的或负的无穷大，那么结果为正无穷大。
- 如果参数为正 0 或负 0，那么结果为 Double.MIN_VALUE。
- 如果参数为 ±Double.MAX_VALUE，那么结果等于 2^{971}。

参数：

d - 要返回 ulp 的浮点值

返回：

参数的 ulp 大小

从以下版本开始：

1.5

ulp

public static float ulp(float f)

返回参数的 ulp 大小。float 值的 ulp 是该浮点值与下一个数值较大的 float 值之间的正距离。注意，对于非 NaN *x*，ulp(-*x*) == ulp(*x*)。

特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为正的或负的无穷大，那么结果为正无穷大。
- 如果参数为正 0 或负 0，那么结果为 Float.MIN_VALUE。
- 如果参数为 ±Float.MAX_VALUE，那么结果等于 2^{104}。

参数：
f - 要返回 ulp 的浮点值
返回：
参数的 ulp 大小
从以下版本开始：
1.5

---

signum

public static double signum(double d)

返回参数的符号函数；如果参数为 0，则返回 0；如果参数大于 0，则返回 1.0；如果参数小于 0，则返回 -1.0。

特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为正 0 或负 0，那么结果与参数相同。

参数：
d - 要返回符号函数的浮点值
返回：
参数的符号函数
从以下版本开始：
1.5

---

signum

```
public static float signum(float f)
```

返回参数的符号函数；如果参数为 0，则返回 0；如果参数大于 0，则返回 1.0；如果参数小于 0，则返回 -1.0。

特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为正 0 或负 0，那么结果与参数相同。

参数：
f - 要返回符号函数的浮点值
返回：
参数的符号函数
从以下版本开始：
1.5

---

sinh

```
public static double sinh(double x)
```

返回 double 值的双曲线正弦。x 双曲线正弦的定义是 $(e^x - e^{-x})/2$，其中 e 是欧拉数。

特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为无穷大，那么结果为无穷大，符号与参数符号相同。
- 如果参数为 0，那么结果为 0，符号与参数符号相同。

计算结果必须在准确结果的 2.5 ulp 范围内。

参数：
x - 要返回其双曲线正弦的数字。
返回：
x 的双曲线正弦。
从以下版本开始：
1.5

cosh

public static double cosh(double x)

返回 double 值的双曲线余弦。*x* 的双曲线余弦的定义是 $(e^x + e^{-x})/2$，其中 *e* 是欧拉数。

特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为无穷大，那么结果为正无穷大。
- 如果参数为 0，那么结果为 1.0。

计算结果必须在准确结果的 2.5 ulp 范围内。

参数：
x - 要返回其双曲线余弦的数字。
返回：
x 的双曲线余弦。
从以下版本开始：
1.5

tanh

public static double tanh(double x)

返回 double 值的双曲线余弦。*x* 的双曲线正切的定义是 $(e^x - e^{-x})/(e^x + e^{-x})$，即 sinh(*x*)/cosh(*x*)。注意，准确的 tanh 绝对值始终小于 1。

特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为 0，那么结果为 0，符号与参数符号相同。
- 如果参数为正无穷大，那么结果为 +1.0。
- 如果参数为负无穷大，那么结果为 -1.0。

计算结果必须在准确结果的 2.5 ulp 范围内。任何有限输入值的 tanh 结果的绝对值必定小于等于 1。注意，一旦准确的 tanh 结果在极限值 ±1 的 1/2 ulp 内，则应该返回有正确符号的 ±1.0。

参数：
x - 要返回其双曲线正切的数字。
返回：
x 的双曲线正切。
从以下版本开始：
1.5

---

hypot

```
public static double hypot(double x,
                           double y)
```

返回 sqrt(x^2 +y^2)，没有中间溢出或下溢。

特殊情况如下：

- 如果两个参数都为无穷大，那么结果为正无穷大。
- 如果两个参数都为 NaN 且都不是无穷大，那么结果为 NaN。

计算结果必须在准确结果的 1 ulp 范围内。如果一个参数保持常量，那么在另一个参数中，结果必须具有半单调性。

参数：
x - 一个值
y - 一个值
返回：
没有中间溢出或下溢的 sqrt(x^2 +y^2)
从以下版本开始：
1.5

---

expm1

```
public static double expm1(double x)
```

返回 e^x -1。注意，对于接近 0 的 x 值，expm1(x) + 1 的准确和比 exp(x) 更接近 e^x 的真实结果。

特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为正无穷大，那么结果为正无穷大。
- 如果参数为负无穷大，那么结果为 -1.0。
- 如果参数为 0，那么结果为 0，符号与参数符号相同。

计算结果必须在准确结果的 1 ulp 范围内。结果必须具有半单调性。任何有限输入值的 expm1 的结果一定大于或等于 -1.0。注意，一旦 e^x - 1 的准确结果在极限值 -1 的 1/2 ulp 范围内，则应该返回 -1.0。

参数：
x - 在 e^x -1 的计算中 e 的指数。
返回：
值 e^x - 1。
从以下版本开始：
1.5

---

log1p

```
public static double log1p(double x)
```

返回参数与 1 之和的自然对数。注意，对于小的 x 值，log1p(x) 的结果比 log(1.0+x) 的浮点计算结果更接近 ln(1 + x) 的实际结果。

特殊情况如下：

- 如果参数为 NaN 或小于 -1，那么结果为 NaN。
- 如果参数为正无穷大，那么结果为正无穷大。
- 如果参数为负数，那么结果为负无穷大。
- 如果参数为 0，那么结果为 0，符号与参数符号相同。

计算结果必须在准确结果的 1 ulp 范围内。结果必须具有半单调性。

参数：
x - 一个值

返回：
值 ln(x + 1)，即 x + 1 的自然对数
从以下版本开始：
1.5

---

copySign

```
public static double copySign(double magnitude,
                              double sign)
```

返回带有第二个浮点参数符号的第一个浮点参数。注意，与 StrictMath.copySign 方法不同，此方法不要求将 NaN sign 参数视为正值；允许实现将某些 NaN 参数视为正，将另一些视为负，以获得更好的性能。
参数：
magnitude - 提供结果数值的参数
sign - 提供结果符号的参数
返回：
一个值，带有 magnitude 的数值，sign 的符号。
从以下版本开始：
1.6

---

copySign

```
public static float copySign(float magnitude,
                             float sign)
```

返回带有第二个浮点参数符号的第一个浮点参数。注意，与 StrictMath.copySign 方法不同，此方法不要求将 NaN sign 参数视为正值；允许实现将某些 NaN 参数视为正，将另一些视为负，以获得更好的性能。
参数：
magnitude - 提供结果数值的参数
sign - 提供结果符号的参数
返回：
一个值，带有 magnitude 的数值，sign 的符号。
从以下版本开始：
1.6

getExponent

```
public static int getExponent(float f)
```

返回 float 表示形式中使用的无偏指数。特殊情况如下：

- 如果参数为 NaN 或无穷大，那么结果为 Float.MAX_EXPONENT + 1。
- 如果参数为 0 或 subnormal，那么结果为 Float.MIN_EXPONENT -1。

参数：
f - 一个 float 值
返回：
参数的无偏指数
从以下版本开始：
1.6

getExponent

```
public static int getExponent(double d)
```

返回 double 表示形式中使用的无偏指数。特殊情况如下：

- 如果参数为 NaN 或无穷大，那么结果为 Double.MAX_EXPONENT + 1。
- 如果参数为 0 或 subnormal，那么结果为 Double.MIN_EXPONENT -1。

参数：
d - double 值
返回：
参数的无偏指数
从以下版本开始：
1.6

nextAfter

```
public static double nextAfter(double start,
                               double direction)
```

返回第一个参数和第二个参数之间与第一个参数相邻的浮点数。如果两个参数比较起来相等，则返回第二个参数。

特殊情况如下：

- 如果任一参数为 NaN，则返回 NaN。
- 如果两个参数都为有符号的 0，则不做更改地返回 direction（根据要求，如果参数比较起来相等，将返回第二个参数）。
- 如果 start 为 ±Double.MIN_VALUE，而 direction 的值要求结果为一个比 start 小的数值，那么将返回 0，并带有与 start 相同的符号。
- 如果 start 为无穷大，而 direction 的值要求结果为一个比 start 小的数值，则返回 Double.MAX_VALUE，并带有与 start 相同的符号。
- 如果 start 等于 ±Double.MAX_VALUE，而 direction 的值要求结果为一个比 start 大的数值，则返回无穷大，并带有与 start 相同的符号。

参数：
start - 起始浮点值。
direction - 一个值，指示应返回 start 的某个邻数还是 start。
返回：
start 和 direction 之间与 start 相邻的浮点数。
从以下版本开始：
1.6

---

nextAfter

```
public static float nextAfter(float start,
                              double direction)
```

返回第一个参数和第二个参数之间与第一个参数相邻的浮点数。如果两个参数比较起来相等，则返回一个与第二个参数相等的值。

特殊情况如下：

- 如果任一参数为 NaN，则返回 NaN。
- 如果两个参数都为有符号的 0，则返回等于 direction 的值。

- 如果 start 为 ±Float.MIN_VALUE，而 direction 的值要求结果为一个比 start 小的数值，那么将返回 0，并带有与 start 相同的符号。
- 如果 start 为无穷大，而 direction 的值要求结果为一个比 start 小的数值，则返回 Float.MAX_VALUE，并带有与 start 相同的符号。
- 如果 start 等于 ±Float.MAX_VALUE，而 direction 的值要求结果为一个比 start 大的数值，则返回无穷大，并带有与 start 相同的符号。

参数：
start - 起始浮点值。
direction - 一个值，指示应返回 start 的某个邻数还是 start。
返回：
start 和 direction 之间与 start 相邻的浮点数。
从以下版本开始：
1.6

---

nextUp

```
public static double nextUp(double d)
```

返回 d 和正无穷大之间与 d 相邻的浮点值。此方法在语义上等同于 nextAfter(d, Double.POSITIVE_INFINITY)；但是，nextUp 实现的返回速度可能比其等价 nextAfter 调用快。

特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为正无穷大，那么结果为正无穷大。
- 如果参数为 0，那么结果为 Double.MIN_VALUE。

参数：
d - 起始浮点值。
返回：
离正无穷大较近的相邻浮点值。
从以下版本开始：
1.6

---

nextUp

public static float nextUp(float f)

返回 f 和正无穷大之间与 f 相邻的浮点值。此方法在语义上等同于 nextAfter(f, Float.POSITIVE_INFINITY)；但是，nextUp 实现的返回速度可能比其等价 nextAfter 调用快。

特殊情况如下：

- 如果参数为 NaN，那么结果为 NaN。
- 如果参数为正无穷大，那么结果为正无穷大。
- 如果参数为 0，那么结果为 Float.MIN_VALUE

参数：
f - 起始浮点值
返回：
离正无穷大较近的相邻浮点值。
从以下版本开始：
1.6

scalb

public static double scalb(double d,
int scaleFactor)

返回 $d \times 2^{scaleFactor}$，其舍入方式如同将一个正确舍入的浮点值乘以 double 值集合中的一个值。有关浮点值集合的讨论，请参阅 Java 语言规范。如果结果的指数在 Double.MIN_EXPONENT 和 Double.MAX_EXPONENT 之间，则可以正确地计算答案；如果结果的指数大于 Double.MAX_EXPONENT，则返回无穷大。注意，如果结果为 subnormal，则可能丢失精度；也就是说，scalb(x, n) 为 subnormal 时，scalb(scalb(x, n), -n) 可能不等于 x。结果为非 NaN 时，结果的符号将与 d 相同。

特殊情况如下：

- 如果第一个参数为 NaN，则返回 NaN。
- 如果第一个参数为无穷大，则返回带有相同符号的无穷大。
- 如果第一个参数为 0，则返回带有相同符号的 0。

参数：
d - 要使用 2 的次幂缩放的数。

scaleFactor - 用来缩放 d 的 2 的次幂

返回：

d × $2^{scaleFactor}$

从以下版本开始：

1.6

---

scalb

```
public static float scalb(float f,
                          int scaleFactor)
```

返回 f × $2^{scaleFactor}$，其舍入方式如同将一个正确舍入的浮点值乘以 float 值集合中的一个值。有关浮点值集合的讨论，请参阅 Java 语言规范。如果结果的指数在 Float.MIN_EXPONENT 和 Float.MAX_EXPONENT 之间，则可以正确地计算答案；如果结果的指数大于 Float.MAX_EXPONENT，则返回无穷大。注意，如果结果为 subnormal，则可能丢失精度；也就是说，scalb(x, n) 为 subnormal 时，scalb(scalb(x, n), -n) 可能不等于 *x*。结果为非 NaN 时，结果的符号将与 f 相同。

特殊情况如下：

- 如果第一个参数为 NaN，则返回 NaN。
- 如果第一个参数为无穷大，则返回带有相同符号的无穷大。
- 如果第一个参数为 0，则返回带有相同符号的 0。

参数：

f - 要使用 2 的次幂缩放的数。

scaleFactor - 用来缩放 f 的 2 的次幂

返回：

f × $2^{scaleFactor}$

从以下版本开始：

1.6

Number 类

抽象类 Number 是 BigDecimal、BigInteger、Byte、Double、Float、Integer、Long 和 Short

类的超类。

Number 的子类必须提供将表示的数值转换为 byte、double、float、int、long 和 short 的方法。

构造方法详细信息

Number

public Number()

方法详细信息

intValue

public abstract int intValue()

以 int 形式返回指定的数值。这可能会涉及到舍入或取整。

返回：

转换为 int 类型后该对象表示的数值。

---

longValue

public abstract long longValue()

以 long 形式返回指定的数值。这可能涉及到舍入或取整。

返回：

转换为 long 类型后该对象表示的数值。

---

floatValue

public abstract float floatValue()

以 float 形式返回指定的数值。这可能会涉及到舍入。

返回：

转换为 float 类型后该对象表示的数值。

---

doubleValue

```
public abstract double doubleValue()
```

以 double 形式返回指定的数值。这可能会涉及到舍入。

返回：

转换为 double 类型后该对象表示的数值。

---

byteValue

```
public byte byteValue()
```

以 byte 形式返回指定的数值。这可能会涉及到舍入或取整。

返回：

转换为 byte 类型后该对象表示的数值。

从以下版本开始：

JDK1.1

---

shortValue

```
public short shortValue()
```

以 short 形式返回指定的数值。这可能会涉及到舍入或取整。

返回：

转换为 short 类型后该对象表示的数值。

Object 类

类 Object 是类层次结构的根类。每个类都使用 Object 作为超类。所有对象（包括数组）都实现这个类的方法。

构造方法详细信息

Object

public Object()

方法详细信息

getClass

```
public final Class<?> getClass()
```

返回此 Object 的运行时类。返回的 Class 对象是由所表示类的 static synchronized 方法锁定的对象。
实际结果类型是 Class<? extends |X|>，其中 |X| 表示清除表达式中的静态类型，该表达式调用 getClass。 例如，以下代码片段中不需要强制转换：

```
Number n = 0;
Class<? extends Number> c = n.getClass();
```

返回：
表示此对象运行时类的 Class 对象。
另请参见：
The Java Language Specification, Third Edition (15.8.2 Class Literals)

---

hashCode

```
public int hashCode()
```

返回该对象的哈希码值。支持此方法是为了提高哈希表（例如 java.util.Hashtable 提供的哈希表）的性能。

hashCode 的常规协定是：

- 在 Java 应用程序执行期间，在对同一对象多次调用 hashCode 方法时，必须一致地返回相同的整数，前提是将对象进行 equals 比较时所用的信息没有被修改。从某一应用程序的一次执行到同一应用程序的另一次执行，该整数无需保持一致。
- 如果根据 equals(Object) 方法，两个对象是相等的，那么对这两个对象中的每个对象调用 hashCode 方法都必须生成相同的整数结果。
- 如果根据 equals(java.lang.Object) 方法，两个对象不相等，那么对这两个对象中的任一对象上调用 hashCode 方法不 要求一定生

成不同的整数结果。但是，程序员应该意识到，为不相等的对象生成不同整数结果可以提高哈希表的性能。

实际上，由 Object 类定义的 hashCode 方法确实会针对不同的对象返回不同的整数。（这一般是通过将该对象的内部地址转换成一个整数来实现的，但是 Java™ 编程语言不需要这种实现技巧。）

返回：
此对象的一个哈希码值。
另请参见：
equals(java.lang.Object), Hashtable

equals

public boolean equals(Object obj)

指示其他某个对象是否与此对象“相等”。

equals 方法在非空对象引用上实现相等关系：

- *自反性*：对于任何非空引用值 x，x.equals(x) 都应返回 true。
- *对称性*：对于任何非空引用值 x 和 y，当且仅当 y.equals(x) 返回 true 时，x.equals(y) 才应返回 true。
- *传递性*：对于任何非空引用值 x、y 和 z，如果 x.equals(y) 返回 true，并且 y.equals(z) 返回 true，那么 x.equals(z) 应返回 true。
- *一致性*：对于任何非空引用值 x 和 y，多次调用 x.equals(y) 始终返回 true 或始终返回 false，前提是对象上 equals 比较中所用的信息没有被修改。
- 对于任何非空引用值 x，x.equals(null) 都应返回 false。

Object 类的 equals 方法实现对象上差别可能性最大的相等关系；即，对于任何非空引用值 x 和 y，当且仅当 x 和 y 引用同一个对象时，此方法才返回 true（x == y 具有值 true）。

注意：当此方法被重写时，通常有必要重写 hashCode 方法，以维护 hashCode 方法的常规协定，该协定声明相等对象必须具有相等的哈希码。

参数：

obj - 要与之比较的引用对象。
返回：
如果此对象与 obj 参数相同，则返回 true；否则返回 false。
另请参见：
hashCode(), Hashtable

clone

```
protected Object clone()
                    throws CloneNotSupportedException
```

创建并返回此对象的一个副本。“副本”的准确含义可能依赖于对象的类。这样做的目的是，对于任何对象 x，表达式：
x.clone() != x
为 true，表达式：
x.clone().getClass() == x.getClass()
也为 true，但这些并非必须要满足的要求。一般情况下：
x.clone().equals(x)
为 true，但这并非必须要满足的要求。

按照惯例，返回的对象应该通过调用 super.clone 获得。如果一个类及其所有的超类（Object 除外）都遵守此约定，则 x.clone().getClass() == x.getClass()。

按照惯例，此方法返回的对象应该独立于该对象（正被复制的对象）。要获得此独立性，在 super.clone 返回对象之前，有必要对该对象的一个或多个字段进行修改。这通常意味着要复制包含正在被复制对象的内部“深层结构”的所有可变对象，并使用对副本的引用替换对这些对象的引用。如果一个类只包含基本字段或对不变对象的引用，那么通常不需要修改 super.clone 返回的对象中的字段。

Object 类的 clone 方法执行特定的复制操作。首先，如果此对象的类不能实现接口 Cloneable，则会抛出 CloneNotSupportedException。注意，所有的数组都被视为实现接口 Cloneable。否则，此方法会创建此对象的类的一个新实例，并像通过分配那样，严格使用此对象相应字段的内容初始化该对象的所有字段；这些字段的内容没有被自我复制。所以，此方法执行的是该对象的“浅表复制”，而不“深层复制”操作。

Object 类本身不实现接口 Cloneable，所以在类为 Object 的对象上调用 clone 方法将会导致在运行时抛出异常。

返回：
此实例的一个副本。
抛出：
CloneNotSupportedException - 如果对象的类不支持 Cloneable 接口，则重写 clone 方法的子类也会抛出此异常，以指示无法复制某个实例。
另请参见：
Cloneable

---

toString

```
public String toString()
```

返回该对象的字符串表示。通常，toString 方法会返回一个“以文本方式表示”此对象的字符串。结果应是一个简明但易于读懂的信息表达式。建议所有子类都重写此方法。

Object 类的 toString 方法返回一个字符串，该字符串由类名（对象是该类的一个实例）、at 标记符“@”和此对象哈希码的无符号十六进制表示组成。换句话说，该方法返回一个字符串，它的值等于：

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

返回：
该对象的字符串表示形式。

---

notify

```
public final void notify()
```

唤醒在此对象监视器上等待的单个线程。如果所有线程都在此对象上等待，则会选择唤醒其中一个线程。选择是任意性的，并在对实现做出决定时发生。线程通过调用其中一个 wait 方法，在对象的监视器上等待。

直到当前线程放弃此对象上的锁定，才能继续执行被唤醒的线程。被唤醒的线程将以常规方式与在该对象上主动同步的其他所有线程进行竞争；例

如，唤醒的线程在作为锁定此对象的下一个线程方面没有可靠的特权或劣势。

此方法只应由作为此对象监视器的所有者的线程来调用。通过以下三种方法之一，线程可以成为此对象监视器的所有者：

- 通过执行此对象的同步实例方法。
- 通过执行在此对象上进行同步的 synchronized 语句的正文。
- 对于 Class 类型的对象，可以通过执行该类的同步静态方法。

一次只能有一个线程拥有对象的监视器。

抛出：
IllegalMonitorStateException - 如果当前线程不是此对象监视器的所有者。
另请参见：
notifyAll(), wait()

notifyAll

```
public final void notifyAll()
```

唤醒在此对象监视器上等待的所有线程。线程通过调用其中一个 wait 方法，在对象的监视器上等待。

直到当前线程放弃此对象上的锁定，才能继续执行被唤醒的线程。被唤醒的线程将以常规方式与在该对象上主动同步的其他所有线程进行竞争；例如，唤醒的线程在作为锁定此对象的下一个线程方面没有可靠的特权或劣势。

此方法只应由作为此对象监视器的所有者的线程来调用。有关线程能够成为监视器所有者的方法的描述，请参阅 notify 方法。

抛出：
IllegalMonitorStateException - 如果当前线程不是此对象监视器的所有者。
另请参见：
notify(), wait()

wait

```
public final void wait(long timeout)
                    throws InterruptedException
```

在其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者超过指定的时间量前，导致当前线程等待。

当前线程必须拥有此对象监视器。

此方法导致当前线程（称之为 *T*）将其自身放置在对象的等待集中，然后放弃此对象上的所有同步要求。出于线程调度目的，在发生以下四种情况之一前，线程 *T* 被禁用，且处于休眠状态：

- 其他某个线程调用此对象的 notify 方法，并且线程 *T* 碰巧被任选为被唤醒的线程。
- 其他某个线程调用此对象的 notifyAll 方法。
- 其他某个线程中断线程 *T*。
- 大约已经到达指定的实际时间。但是，如果 timeout 为零，则不考虑实际时间，在获得通知前该线程将一直等待。

然后，从对象的等待集中删除线程 *T*，并重新进行线程调度。然后，该线程以常规方式与其他线程竞争，以获得在该对象上同步的权利；一旦获得对该对象的控制权，该对象上的所有其同步声明都将被恢复到以前的状态，这就是调用 wait 方法时的情况。然后，线程 *T* 从 wait 方法的调用中返回。所以，从 wait 方法返回时，该对象和线程 T 的同步状态与调用 wait 方法时的情况完全相同。

在没有被通知、中断或超时的情况下，线程还可以唤醒一个所谓的*虚假唤醒* (spurious wakeup)。虽然这种情况在实践中很少发生，但是应用程序必须通过以下方式防止其发生，即对应该导致该线程被提醒的条件进行测试，如果不满足该条件，则继续等待。换句话说，等待应总是发生在循环中，如下面的示例：

```
synchronized (obj) {
while (<condition does not hold>)
obj.wait(timeout);
... // Perform action appropriate to condition
      }
```

（有关这一主题的更多信息，请参阅 Doug Lea 撰写的 Concurrent Programming in Java (Second Edition) (Addison-Wesley, 2000) 中的第 3.2.3 节或 Joshua Bloch 撰写的 Effective Java Programming Language Guide (Addison-Wesley, 2001) 中的第 50 项。

如果当前线程在等待之前或在等待时被任何线程中断，则会抛出 InterruptedException。在按上述形式恢复此对象的锁定状态时才会抛出此异常。

注意，由于 wait 方法将当前线程放入了对象的等待集中，所以它只能解除此对象的锁定；可以同步当前线程的任何其他对象在线程等待时仍处于锁定状态。

此方法只应由作为此对象监视器的所有者的线程来调用。有关线程能够成为监视器所有者的方法的描述，请参阅 notify 方法。

参数：
timeout - 要等待的最长时间（以毫秒为单位）。
抛出：
IllegalArgumentException - 如果超时值为负。
IllegalMonitorStateException - 如果当前线程不是此对象监视器的所有者。
InterruptedException - 如果在当前线程等待通知之前或者正在等待通知时，任何线程中断了当前线程。在抛出此异常时，当前线程的*中断状态*被清除。
另请参见：
notify(), notifyAll()

---

wait

```
public final void wait(long timeout,
                       int nanos)
                throws InterruptedException
```

在其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量前，导致当前线程等待。

此方法类似于一个参数的 wait 方法，但它允许更好地控制在放弃之前等待通知的时间量。用毫微秒度量的实际时间量可以通过以下公式计算出来：

```
1000000*timeout+nanos
```

在其他所有方面，此方法执行的操作与带有一个参数的 wait(long) 方法相同。需要特别指出的是，wait(0, 0) 与 wait(0) 相同。

当前线程必须拥有此对象监视器。该线程发布对此监视器的所有权，并等待下面两个条件之一发生：

- 其他线程通过调用 notify 方法，或 notifyAll 方法通知在此对象的监视器上等待的线程醒来。
- timeout 毫秒值与 nanos 毫微秒参数值之和指定的超时时间已用完。

然后，该线程等到重新获得对监视器的所有权后才能继续执行。

对于某一个参数的版本，实现中断和虚假唤醒是有可能的，并且此方法应始终在循环中使用：

```
synchronized (obj) {
while (<condition does not hold>)
obj.wait(timeout, nanos);
... // Perform action appropriate to condition
      }
```

此方法只应由作为此对象监视器的所有者的线程来调用。有关线程能够成为监视器所有者的方法的描述，请参阅 notify 方法。

参数：

timeout - 要等待的最长时间（以毫秒为单位）。

nanos - 额外时间（以毫微秒为单位，范围是 0-999999）。

抛出：

IllegalArgumentException - 如果超时值是负数，或者毫微秒值不在 0-999999 范围内。

IllegalMonitorStateException - 如果当前线程不是此对象监视器的所有者。

InterruptedException - 如果在当前线程等待通知之前或者正在等待通知时，任何线程中断了当前线程。在抛出此异常时，当前线程的*中断状态*被清除。

wait

```
public final void wait()
                    throws InterruptedException
```

在其他线程调用此对象的 notify() 方法或 notifyAll() 方法前，导致当前线程等待。换句话说，此方法的行为就好像它仅执行 wait(0) 调用一样。

当前线程必须拥有此对象监视器。该线程发布对此监视器的所有权并等待，直到其他线程通过调用 notify 方法，或 notifyAll 方法通知在此对象的监视器上等待的线程醒来。然后该线程将等到重新获得对监视器的所有权后才能继续执行。

对于某一个参数的版本，实现中断和虚假唤醒是可能的，而且此方法应始终在循环中使用：

```
synchronized (obj) {
while (<condition does not hold>)
obj.wait();
... // Perform action appropriate to condition
      }
```

此方法只应由作为此对象监视器的所有者的线程来调用。有关线程能够成为监视器所有者的方法的描述，请参阅 notify 方法。

抛出：

IllegalMonitorStateException - 如果当前线程不是此对象监视器的所有者。

InterruptedException - 如果在当前线程等待通知之前或者正在等待通知时，任何线程中断了当前线程。在抛出此异常时，当前线程的*中断状态*被清除。

另请参见：

notify(), notifyAll()

finalize

```
protected void finalize()
                    throws Throwable
```

当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。子类重写 finalize 方法，以配置系统资源或执行其他清除。

finalize 的常规协定是：当 Java™ 虚拟机已确定尚未终止的任何线程无法再通过任何方法访问此对象时，将调用此方法，除非由于准备终止的其他某个对象或类的终结操作执行了某个操作。finalize 方法可以采取任何操作，其中包括再次使此对象对其他线程可用；不过，finalize 的主要目的是在不可撤消地丢弃对象之前执行清除操作。例如，表示输入/输出连接的对象的 finalize 方法可执行显式 I/O 事务，以便在永久丢弃对象之前中断连接。

Object 类的 finalize 方法执行非特殊性操作；它仅执行一些常规返回。Object 的子类可以重写此定义。

Java 编程语言不保证哪个线程将调用某个给定对象的 finalize 方法。但可以保证在调用 finalize 时，调用 finalize 的线程将不会持有任何用户可见的同步锁定。如果 finalize 方法抛出未捕获的异常，那么该异常将被忽略，并且该对象的终结操作将终止。

在启用某个对象的 finalize 方法后，将不会执行进一步操作，直到 Java 虚拟机再次确定尚未终止的任何线程无法再通过任何方法访问此对象，其中包括由准备终止的其他对象或类执行的可能操作，在执行该操作时，对象可能被丢弃。

对于任何给定对象，Java 虚拟机最多只调用一次 finalize 方法。

finalize 方法抛出的任何异常都会导致此对象的终结操作停止，但可以通过其他方法忽略它。

抛出：
Throwable - 此方法抛出的 Exception

Package 类

Package 对象包含有关 Java 包的实现和规范的版本信息。通过用于加载类的 ClassLoader 实例，可以获取并获得此版本信息。通常，此信息存储在与类一起分发的清单中。

组成包的类集可以实现一个特定规范，如此一来，就可以通过规范标题、版本号和供应商字符串来标识该规范。应用程序可以询问包是否与特定版本兼容，有关详细信息，请参阅 isCompatibleWith 方法。

规范的版本号使用了一个由句点 "." 分隔的十进制正整数组成的语法，例如 "2.0" 或 "1.2.3.4.5.6.7"。这允许使用可扩展的编号来表示主版本号、次版本号、缩微版本号，等等。版本规范是由下列形式的语法描述的：

SpecificationVersion:
Digits RefinedVersionopt
RefinedVersion:
. Digits
. Digits RefinedVersion
Digits:
Digit
Digits
Digit:

Character.isDigit(char) 会对其返回 true 的任何字符，如 0、1、2 等等。

实现标题、版本和供应商字符串共同标识了一个实现，并且可以很方便地使用它们来支持出现问题时所涉及的包的准确报告。三个实现字符串内容都是特定于供应商的。实现版本字符串没有特定的语法，并且应该只在为了使它们等同于所需的版本标识符时对它们进行比较。

在每一个 ClassLoader 实例中，相同 java 包中的所有类都有相同的 Package 对象。静态方法允许通过名称或当前类加载器已知的所有包的集合来找到包。

方法详细信息

getName

```
public String getName()
```

返回此包的名称。

返回：

根据 *Java 语言规范第三版* §6.5.3 中的定义，返回此包的完全限定名，例如 java.lang。

getSpecificationTitle

public String getSpecificationTitle()

返回此包实现的规范标题。

返回：

规范的标题，如果不知道此标题，则返回 null。

getSpecificationVersion

public String getSpecificationVersion()

返回此包实现的规范的版本号。该版本字符串必须是以 "." 分隔的十进制正整数的序列，并可能有前导零。在比较版本字符串时，比较最重要的数字。

返回：

规范版本，如果不知道该版本，则返回 null。

getSpecificationVendor

public String getSpecificationVendor()

返回拥有并维护实现此包的类规范的组织、供应商或公司的名称。

返回：

规范的供应商，如果不知道此供应商，则返回 null。

getImplementationTitle

public String getImplementationTitle()

返回此包的标题。

返回：

实现的标题，如果不知道此标题，则返回 null。

getImplementationVersion

public String getImplementationVersion()

返回该实现的版本。它由该实现的供应商分配的任何字符串组成，并且没有任何由 Java 运行时指定或需要的特定语法。可以对实现的版本进行比较，使其等同于此包的供应商用于该实现的其他包版本字符串。

返回：

实现的版本，如果不知道此版本，则返回 null。

getImplementationVendor

public String getImplementationVendor()

返回提供该实现的组织、供应商或公司的名称。

返回：

实现此包的供应商。

isSealed

public boolean isSealed()

如果此包是密封的，则返回 ture。

返回：

如果包是密封的，则返回 ture，否则返回 false。

isSealed

public boolean isSealed(URL url)

如果此包对于指定的代码源 url 是密封的，则返回 ture。

参数：

url - 代码源 url

返回：

如果此包对于 url 是密封的，则返回 ture

isCompatibleWith

```
public boolean isCompatibleWith(String desired)
                                throws NumberFormatException
```

比较此包的规范版本和所需版本。如果此包的规范版本号大于或等于所需版本号，则返回 ture。

通过按顺序比较所需字符串和规范字符串的对应组件，对版本号进行比较。每个组件都被转换为一个十进制整数和比较的值。如果规范值大于所需值，则返回 true。如果小于所需值，则返回 false。如果两个值相等，则跳过这一比较周期，比较下一对组件。

参数：

desired - 所需版本的版本字符串。

返回：

如果此包的版本号大于或等于所需的版本号，则返回 ture。

抛出：

NumberFormatException - 如果所需版本或当前版本的点形式不正确。

getPackage

```
public static Package getPackage(String name)
```

通过调用方的 ClassLoader 实例中的名称找到一个包。调用方的 ClassLoader 实例用于找到对应于指定类的包实例。如果调用方的 ClassLoader 实例为 null，则搜索系统的 ClassLoader 实例加载的包集合，以找到指定包。

只有类加载器创建了具有适当属性的包实例，包中才会有规范和版本属性。通常，这些属性定义在随类一起提供的清单中。

参数：

name - 包名称，例如，java.lang。

返回：

具有所请求名称的包。如果不能从存档或基本代码中获得包信息，则可能返回 null。

getPackages

public static Package[] getPackages()

获得调用方的 ClassLoader 实例当前已知的所有包。这些包对应于通过 ClassLoader 实例中的名称加载或访问的类。如果调用方的 ClassLoader 实例是引导程序的 ClassLoader 实例（它在一些实现中可能用 null 表示），则只返回通过引导程序的 ClassLoader 实例加载的类所对应的包。

返回：

调用方的 ClassLoader 实例已知的包的新数组。如果没有已知的包，则返回零长度的数组。

---

hashCode

public int hashCode()

返回从包名称计算的哈希码。

覆盖：

类 Object 中的 hashCode

返回：

从包名称计算的哈希码。

另请参见：

Object.equals(java.lang.Object),Hashtable

---

toString

public String toString()

返回此 Package 的字符串表示形式。它的值是字符串 "package" 和包名称。如果定义了包的标题，则追加包的标题。如果定义了包的版本，则追加包的版本。

覆盖：

类 Object 中的 toString

返回：

包的字符串表示形式。

---

getAnnotation

```
public <A extends Annotation> A getAnnotation(Class<A> annotationClass)
```

从接口 AnnotatedElement 复制的描述

如果存在该元素的指定类型的注释，则返回这些注释，否则返回 null。

指定者：

接口 AnnotatedElement 中的 getAnnotation

参数：

annotationClass - 对应于注释类型的 Class 对象

返回：

如果该元素的指定注释类型的注释存在于此对象上，则返回这些注释，否则返回 null

抛出：

NullPointerException - 如果给定的注释类为 null

---

isAnnotationPresent

```
public boolean isAnnotationPresent(Class<? extends Annotation> annotationClass)
```

从接口 AnnotatedElement 复制的描述

如果指定类型的注释存在于此元素上，则返回 true，否则返回 false。此方法主要是为了便于访问标记注释而设计的。

指定者：

接口 AnnotatedElement 中的 isAnnotationPresent

参数：

annotationClass - 对应于注释类型的 Class 对象

返回：

如果指定注释类型的注释存在于此对象上，则返回 true，否则返回 false

抛出：

NullPointerException - 如果给定的注释类为 null

---

getAnnotations

```
public Annotation[] getAnnotations()
```

从接口 AnnotatedElement 复制的描述

返回此元素上存在的所有注释。（如果此元素没有注释，则返回长度为零的数组。）该方法的调用者可以随意修改返回的数组；这不会对其他调用者返回的数组产生任何影响。

指定者：

接口 AnnotatedElement 中的 getAnnotations

返回：

此元素上存在的所有注释

---

getDeclaredAnnotations

```
public Annotation[] getDeclaredAnnotations()
```

从接口 AnnotatedElement 复制的描述

返回直接存在于此元素上的所有注释。与此接口中的其他方法不同，该方法将忽略继承的注释。（如果没有注释直接存在于此元素上，则返回长度为零的一个数组。）该方法的调用者可以随意修改返回的数组；这不会对其他调用者返回的数组产生任何影响。

指定者：

接口 AnnotatedElement 中的 getDeclaredAnnotations

返回：

直接存在于此元素上的所有注释

Process 类

ProcessBuilder.start() 和 Runtime.exec 方法创建一个本机进程，并返回 Process 子类的一个实例，该实例可用来控制进程并获得相关信息。Process 类提供了执行从进程输入、执行输出到进程、等待进程完成、检查进程的退出状态以及销毁（杀掉）进程的方法。

创建进程的方法可能无法针对某些本机平台上的特定进程很好地工作，比如，本机窗口进程，守护进程，Microsoft Windows 上的 Win16/DOS 进程，或者 shell 脚本。创建的子进程没有自己的终端或控制台。它的所有标准 io（即 stdin、stdout 和 stderr）操作都将通过三个流 (getOutputStream()、getInputStream() 和 getErrorStream()) 重定向到父进程。父进程使用这些流来提供到子进程的输入和获得从子进程的输出。因为有些本机平台仅针对标准输入和输出流提供有限的缓冲区大小，如果读写子进程的输出流或输入流迅速出现失败，则可能导致子进程阻塞，甚至产生死锁。

当没有 Process 对象的更多引用时，不是删掉子进程，而是继续异步执行子进程。

对于带有 Process 对象的 Java 进程，没有必要异步或并发执行由 Process 对象表示的进程。

构造方法详细信息

Process

```
public Process()
```

方法详细信息

getOutputStream

```
public abstract OutputStream getOutputStream()
```

获取子进程的输出流。输出流被传送给由该 Process 对象表示的进程的标准输入流。

实现注意事项：对输出流进行缓冲是一个好主意。

返回：
连接到子进程正常输入的输出流。

---

getInputStream

```
public abstract InputStream getInputStream()
```

获取子进程的输入流。输入流获得由该 Process 对象表示的进程的标准输出流。

实现注意事项：对输入流进行缓冲是一个好主意。

返回：
连接到子进程正常输出的输入流。
另请参见：
ProcessBuilder.redirectErrorStream()

---

getErrorStream

```
public abstract InputStream getErrorStream()
```

获取子进程的错误流。错误流获得由该 Process 对象表示的进程的错误输出流传送的数据。

实现注意事项：对输入流进行缓冲是一个好主意。

返回：
连接到子进程错误流的输入流。
另请参见：
ProcessBuilder.redirectErrorStream()

---

waitFor

```
public abstract int waitFor()
                    throws InterruptedException
```

导致当前线程等待，如有必要，一直要等到由该 Process 对象表示的进程已经终止。如果已终止该子进程，此方法立即返回。如果没有终止该子进程，调用的线程将被阻塞，直到退出子进程。
返回：
进程的出口值。根据惯例，0 表示正常终止。
抛出：
InterruptedException - 如果当前线程在等待时被另一线程中断，则停止等待，抛出 InterruptedException。

---

exitValue

```
public abstract int exitValue()
```

返回子进程的出口值。
返回：
此 Process 对象表示的子进程的出口值。根据惯例，值 0 表示正常终止。
抛出：
IllegalThreadStateException - 如果此 Process 对象表示的子进程尚未终止。

---

destroy

public abstract void **destroy**()

杀掉子进程。强制终止此 Process 对象表示的子进程。

Runtime 类

每个 Java 应用程序都有一个 Runtime 类实例，使应用程序能够与其运行的环境相连接。可以通过 getRuntime 方法获取当前运行时。

应用程序不能创建自己的 Runtime 类实例。

方法详细信息

getRuntime

public static Runtime getRuntime()

返回与当前 Java 应用程序相关的运行时对象。Runtime 类的大多数方法是实例方法，并且必须根据当前的运行时对象对其进行调用。

返回：

与当前 Java 应用程序相关的 Runtime 对象。

---

exit

public void exit(int status)

通过启动虚拟机的关闭序列，终止当前正在运行的 Java 虚拟机。此方法从不正常返回。可以将变量作为一个状态码；根据惯例，非零的状态码表示非正常终止。

虚拟机的关闭序列包含两个阶段。在第一个阶段中，会以某种未指定的顺序启动所有已注册的关闭钩子（hook）（如果有的话），并且允许它们同时运行直至结束。在第二个阶段中，如果已启用退出终结，则运行所有未调用的终结方法。一旦完成这个阶段，虚拟机就会暂停。

如果在虚拟机已开始其关闭序列后才调用此方法，那么若正在运行关闭钩子，则将无限期地阻断此方法。如果已经运行完关闭钩子，并且已启用退出终结 (on-exit finalization)，那么此方法将利用给定的状态码（如果状态码是非零值）暂停虚拟机；否则将无限期地阻断虚拟机。

System.exit 方法是调用此方法的一种传统而便捷的方式。

参数：
status - 终止状态。按照惯例，非零的状态码表明非正常终止。
抛出：
SecurityException - 如果安全管理器存在，并且其 checkExit 方法不允许存在指定的状态
另请参见：
SecurityException, SecurityManager.checkExit(int),
addShutdownHook(java.lang.Thread),
removeShutdownHook(java.lang.Thread),
runFinalizersOnExit(boolean), halt(int)

addShutdownHook

```
public void addShutdownHook(Thread hook)
```

注册新的虚拟机来关闭钩子。

Java 虚拟机会为了响应以下两类事件而*关闭*:

- 程序正常*退出*,这发生在最后的非守护线程退出时,或者在调用 exit（等同于 System.exit）方法时。或者，
- 为响应用户中断而*终止* 虚拟机，如键入 ^C；或发生系统事件，比如用户注销或系统关闭。

关闭钩子 只是一个已初始化但尚未启动的线程。虚拟机开始启用其关闭序列时，它会以某种未指定的顺序启动所有已注册的关闭钩子，并让它们同时运行。运行完所有的钩子后，如果已启用退出终结，那么虚拟机接着会运行所有未调用的终结方法。最后，虚拟机会暂停。注意，关闭序列期间会继续运行守护线程，如果通过调用 exit 方法来发起关闭序列，那么也会继续运行非守护线程。

一旦开始了关闭序列，则只能通过调用 halt 方法来停止这个序列，此方法可强行终止虚拟机。

一旦开始了关闭序列，则不可能注册新的关闭钩子或取消注册先前已注册的钩子。尝试执行这些操作会导致抛出 IllegalStateException。

关闭钩子可在虚拟机生命周期中的特定时间运行，因此应保护性地对其进行编码。特别是应将关闭钩子编写为线程安全的，并尽可能地避免死锁。关闭钩子还应该不盲目地依靠某些服务，这些服务可能已注册了自己的关闭钩子，所以其本身可能正处于关闭进程中。例如，试图使用其他基于线程的服务（如 AWT 事件指派线程）可能导致死锁。

关闭钩子应该快速地完成其工作。当程序调用 exit 时，虚拟机应该迅速地关闭并退出。由于用户注销或系统关闭而终止虚拟机时，底层的操作系统可能只允许在固定的时间内关闭并退出。因此在关闭钩子中尝试进行任何用户交互或执行长时间的计算都是不明智的。

与其他所有线程一样，通过调用线程 ThreadGroup 对象的 uncaughtException 方法，可在关闭钩子中处理未捕获的异常。此方法的默认实现是将该异常的堆栈跟踪打印至 System.err 并终止线程；它不会导致虚拟机退出或暂停。

仅在很少的情况下，虚拟机可能会*中止*，也就是没有完全关闭就停止运行。虚拟机被外部终止时会出现这种现象，比如在 Unix 上使用 SIGKILL 信号或者在 Microsoft Windows 上调用 TerminateProcess。如果由于内部数据结构损坏或试图访问不存在的内存而导致本机方法执行错误，那么可能也会中止虚拟机。如果虚拟机中止，则无法保证是否将运行关闭钩子。

参数：
hook - 一个已初始化但尚未启动的 Thread 对象
抛出：
IllegalArgumentException - 如果指定的钩子已注册，或者可以确定钩子正在运行或者已运行完毕
IllegalStateException - 如果虚拟机已经处于关闭进程中
SecurityException - 如果安全管理器存在并且拒绝 RuntimePermission("shutdownHooks")
从以下版本开始：
1.3
另请参见：

removeShutdownHook(java.lang.Thread), halt(int), exit(int)

removeShutdownHook

public boolean removeShutdownHook(Thread hook)

取消注册某个先前已注册的虚拟机关闭钩子。

参数：

hook - 要删除的钩子

返回：

如果指定的钩子先前已注册并且成功地取消注册，则返回 true，其他情况返回 false。

抛出：

IllegalStateException - 如果虚拟机已经处于关闭进程中

SecurityException - 如果安全管理器存在并且拒绝 RuntimePermission("shutdownHooks")

从以下版本开始：

1.3

另请参见：

addShutdownHook(java.lang.Thread), exit(int)

halt

public void halt(int status)

强行终止目前正在运行的 Java 虚拟机。此方法从不正常返回。

应小心使用此方法。与 exit 方法不同，此方法不会启动关闭钩子，并且如果已启用退出终结，此方法也不会运行未调用的终结方法。如果已经发起关闭序列，那么此方法不会等待所有正在运行的关闭钩子或终结方法完成其工作。

参数：

status - 终止状态。按照惯例，非零的状态码表明非正常终止。如果已经调用了 exit（System.exit 也一样）方法，那么该状态码将重写已传递至此方法的状态码。

抛出：

SecurityException - 如果安全管理器存在，并且其 checkExit 方法不允许具有指定状态时退出

从以下版本开始：

1.3

另请参见：

exit(int), addShutdownHook(java.lang.Thread), removeShutdownHook(java.lang.Thread)

---

runFinalizersOnExit

```
@Deprecated
    public static void runFinalizersOnExit(boolean value)
```

已过时。 *此方法本身具有不安全性。它可能对正在使用的对象调用终结方法，而其他线程正在操作这些对象，从而导致不正确的行为或死锁。*

在退出时启用或禁用终结；这样做可指定拥有未被自动调用终结方法的所有对象的终结方法，并将在退出 Java 运行时前运行此终结方法。默认情况下，禁用退出终结。

如果有安全管理器，则首先使用 0 作为变量来调用其 checkExit 方法，以确保允许退出。这可能会导致 SecurityException。

参数：

value - 如果启用退出时终结，则该参数为 true，如果禁用退出时终结，则该参数为 false

抛出：

SecurityException - 如果安全管理器存在并且其 checkExit 方法不允许退出。

从以下版本开始：

JDK1.1

另请参见：

exit(int), gc(), SecurityManager.checkExit(int)

---

exec

```
    public Process exec(String command)
                throws IOException
```

在单独的进程中执行指定的字符串命令。

这是一个很有用的方法。对于 exec(command) 形式的调用而言，其行为与调用 exec(command, null, null) 完全相同。

参数：
command - 一条指定的系统命令。
返回：
一个新的 Process 对象，用于管理子进程
抛出：
SecurityException - 如果安全管理器存在，并且其 checkExec 方法不允许创建子进程
IOException - 如果发生 I/O 错误
NullPointerException - 如果 command 为 null
IllegalArgumentException - 如果 command 为空
另请参见：
exec(String[], String[], File), ProcessBuilder

---

exec

```
public Process exec(String command,
                    String[] envp)
             throws IOException
```

在指定环境的单独进程中执行指定的字符串命令。

这是一个很有用的方法。对于 exec(command, envp) 形式的调用而言，其行为与调用 exec(command, envp, null) 完全相同。

参数：
command - 一条指定的系统命令。
envp - 字符串数组，其中每个元素的环境变量的设置格式为 *name*=*value*；如果子进程应该继承当前进程的环境，或该参数为 null。
返回：
一个新的 Process 对象，用于管理子进程
抛出：
SecurityException - 如果安全管理器存在，并且其 checkExec 方法不允许创建子进程
IOException - 如果发生 I/O 错误

NullPointerException - 如果 command 为 null，或 envp 的元素之一为 null
IllegalArgumentException - 如果 command 为空
另请参见：
exec(String[], String[], File), ProcessBuilder

exec

```
public Process exec(String command,
                    String[] envp,
                    File dir)
             throws IOException
```

在有指定环境和工作目录的独立进程中执行指定的字符串命令。

这是一个很有用的方法。对于 exec(command, envp, dir) 形式的调用而言，其行为与调用 exec(cmdarray, envp, dir) 完全相同，其中 cmdarray 是 command 中所有标记的数组。

更准确地说，可以使用通过调用 new StringTokenizer(command) 创建的 StringTokenizer 将 command 字符串拆解成标记，调用时不对字符类别做进一步的修改。然后将标记生成器所生成的标记以相同的顺序放入新的字符串数组 cmdarray 中。

参数：
command - 一条指定的系统命令。
envp - 字符串数组，其中每个元素的环境变量的设置格式为 *name*=*value*；如果子进程应该继承当前进程的环境，或该参数为 null。
dir - 子进程的工作目录；如果子进程应该继承当前进程的工作目录，则该参数为 null。
返回：
一个新的 Process 对象，用于管理子进程
抛出：
SecurityException - 如果安全管理器存在，并且其 checkExec 方法不允许创建子进程
IOException - 如果发生 I/O 错误
NullPointerException - 如果 command 为 null，或者 envp 的某个元素为 null

IllegalArgumentException - 如果 command 为空
从以下版本开始：
1.3
另请参见：
ProcessBuilder

---

exec

```
public Process exec(String[] cmdarray)
                    throws IOException
```

在单独的进程中执行指定命令和变量。

这是一个很有用的方法。对于 exec(cmdarray) 形式的调用而言，其行为与调用 exec(cmdarray, null, null) 完全相同。

参数：
cmdarray - 包含所调用命令及其参数的数组。
返回：
一个新的 Process 对象，用于管理子进程
抛出：
SecurityException - 如果安全管理器存在，并且其 checkExec 方法不允许创建子进程
IOException - 如果发生 I/O 错误
NullPointerException - 如果 cmdarray 为 null，或者 cmdarray 的某个元素为 null
IndexOutOfBoundsException - 如果 cmdarray 是一个空数组（长度为 0）
另请参见：
ProcessBuilder

---

exec

```
public Process exec(String[] cmdarray,
                    String[] envp)
                    throws IOException
```

在指定环境的独立进程中执行指定命令和变量。

这是一个很有用的方法。对于 exec(cmdarray, envp) 形式的调用而言，其行为与调用 exec(cmdarray, envp, null) 完全相同。

参数：
cmdarray - 包含所调用命令及其参数的数组。
envp - 字符串数组，其中每个元素的环境变量的设置格式为 *name*=*value*；如果子进程应该继承当前进程的环境，或该参数为 null。
返回：
一个新的 Process 对象，用于管理子进程
抛出：
SecurityException - 如果安全管理器存在，并且其 checkExec 方法不允许创建子进程
IOException - 如果发生 I/O 错误
NullPointerException - 如果 cmdarray 为 null，或者 cmdarray 的某个元素为 null，或者 envp 的某个元素为 null
IndexOutOfBoundsException - 如果 cmdarray 是一个空数组（长度为 0）
另请参见：
ProcessBuilder

---

exec

```
public Process exec(String[] cmdarray,
                    String[] envp,
                    File dir)
             throws IOException
```

在指定环境和工作目录的独立进程中执行指定的命令和变量。

给定的字符串数组 cmdarray 表示一个命令行标记，字符串数组 envp 则表示“环境”变量设置，此方法会创建一个新进程，而指定的命令就在这个进程中执行。

此方法检查 cmdarray 是否是一条有效的操作系统命令。哪些命令有效取决于系统，但是该命令至少必须有一个非 null 字符串的非空列表。

如果 envp 为 null，那么子进程会继承当前进程的环境设置。

ProcessBuilder.start() 现在是启用一个具有已修改环境的进程的首选方法。

dir 指定了新子进程的工作目录。如果 dir 为 null，那么子进程会继承当前进程的当前工作目录。

如果安全管理器存在，则用数组 cmdarray 的第一个元素作为变量来调用安全管理器的 checkExec 方法。这可能导致抛出 SecurityException。

启动操作系统进程的方式完全取决于系统。其中有很多方面会导致错误：

- 未找到操作系统程序文件。
- 对程序文件的访问被拒绝。
- 工作目录不存在。

这些情况都会抛出一个异常。该异常的具体特性取决于系统，但它总是 IOException 的一个子类。

参数：
cmdarray - 包含所调用命令及其参数的数组。
envp - 字符串数组，其中每个元素的环境变量的设置格式为 *name*=*value*，如果子进程应该继承当前进程的环境，或该参数为 null。
dir - 子进程的工作目录；如果子进程应该继承当前进程的工作目录，则该参数为 null。
返回：
一个新的 Process 对象，用于管理子进程
抛出：
SecurityException - 如果安全管理器存在，并且其 checkExec 方法不允许创建子进程
IOException - 如果发生 I/O 错误
NullPointerException - 如果 cmdarray 为 null，或者 cmdarray 的某个元素为 null，抑或 envp 的某个元素为 null
IndexOutOfBoundsException - 如果 cmdarray 是一个空数组（长度为 0）
从以下版本开始：
1.3
另请参见：
ProcessBuilder

---

availableProcessors

```
public int availableProcessors()
```

向 Java 虚拟机返回可用处理器的数目。

该值在特定的虚拟机调用期间可能发生更改。因此，对可用处理器数目很敏感的应用程序应该不定期地轮询该属性，并相应地调整其资源用法。

返回：
虚拟机可用的最大处理器数目；从不小于 1
从以下版本开始：
1.4

---

freeMemory

```
public long freeMemory()
```

返回 Java 虚拟机中的空闲内存量。调用 gc 方法可能导致 freeMemory 返回值的增加。
返回：
供将来分配对象使用的当前可用内存的近似总量，以字节为单位。

---

totalMemory

```
public long totalMemory()
```

返回 Java 虚拟机中的内存总量。此方法返回的值可能随时间的推移而变化，这取决于主机环境。

注意，保存任意给定类型的一个对象所需的内存量可能取决于实现方法。

返回：
目前为当前和后续对象提供的内存总量，以字节为单位。

---

maxMemory

```
public long maxMemory()
```

返回 Java 虚拟机试图使用的最大内存量。如果内存本身没有限制，则返回值 Long.MAX_VALUE。
返回：

虚拟机试图使用的最大内存量，以字节为单位。

从以下版本开始：

1.4

---

gc

```
public void gc()
```

运行垃圾回收器。调用此方法意味着 Java 虚拟机做了一些努力来回收未用对象，以便能够快速地重用这些对象当前占用的内存。当控制从方法调用中返回时，虚拟机已经尽最大努力回收了所有丢弃的对象。

名称 gc 代表“垃圾回收器”。虚拟机根据需要在单独的线程中自动执行回收过程，甚至不用显式调用 gc 方法。

方法 System.gc() 是调用此方法的一种传统而便捷的方式。

---

runFinalization

```
public void runFinalization()
```

运行挂起 finalization 的所有对象的终止方法。调用此方法意味着 Java 虚拟机做了一些努力运行已被丢弃对象的 finalize 方法，但是这些对象的 finalize 方法还没有运行。当控制从方法调用中返回时，Java 虚拟机已经尽最大努力去完成所有未执行的终止方法。

如果不显式调用 runFinalization 方法，则 Java 虚拟机会根据需要在单独的线程中自动执行此终止过程。

方法 System.runFinalization() 是调用此方法的一种传统而便捷的方式。

另请参见：

Object.finalize()

---

traceInstructions

public void traceInstructions(boolean on)

启用 / 禁用指令跟踪。如果 boolean 变量为 true，则执行此方法意味着让 Java 虚拟机发送虚拟机中每条指令执行的调试信息。该信息的格式，以及虚拟机所发送的文件或其他输出流的格式取决于主机的环境。如果虚拟机不支持此功能，则忽略这一请求。跟踪输出的目的地取决于系统。

如果 boolean 变量为 false，则执行此方法时将使 Java 虚拟机停止执行详细的指令跟踪。

参数：
on - 为 true 时启用指令跟踪；为 false 时则禁用此功能。

traceMethodCalls

public void traceMethodCalls(boolean on)

启用 / 禁用方法调用跟踪。如果 boolean 变量为 true，则执行此方法意味着让 Java 虚拟机发送虚拟机中每个方法的调试信息。该信息的格式，以及虚拟机所发送的文件或其他输出流的格式取决于主机的环境。如果虚拟机不支持此功能，则忽略这一请求。

使用变量 false 调用此方法意味着虚拟机停止发送每个调用的调试信息。

参数：
on - 为 true 时启用指令跟踪；为 false 时则禁用此功能。

load

public void load(String filename)

加载作为动态库的指定文件名。文件名变量必须是一个完整的路径名，(例如 Runtime.getRuntime().load("/home/avh/lib/libX11.so");) 。

首先，如果有安全管理器，则用 filename 作为参数调用 checkLink 方法。这可能导致安全异常。

这与 loadLibrary(String) 方法类似，但它接受通用文件名作为变量，而不仅仅是库名，从而能够加载所有的本机代码文件。

方法 System.load(String) 是调用此方法的一种传统而便捷的方式。

参数：
filename - 要加载的文件。
抛出：
SecurityException - 如果安全管理器存在，并且其 checkLink 方法不允许加载指定的动态库
UnsatisfiedLinkError - 如果文件不存在。
NullPointerException - 如果 filename 为 null
另请参见：
getRuntime(), SecurityException,
SecurityManager.checkLink(java.lang.String)

loadLibrary

```
public void loadLibrary(String libname)
```

加载具有指定库名的动态库。从以前获取库文件的本地文件系统中加载含有本机代码的文件。这一过程的细节取决于实现方法。可以以某种特定于系统的方式完成从库名到特定文件名的映射。

首先，如果有安全管理器，则用 libname 作为参数调用 checkLink 方法。这可能导致安全性异常。

方法 System.loadLibrary(String) 是调用此方法的一种传统而便捷的方式。如果在某个类实现中使用本机方法，则标准的策略是将本机代码放入一个库文件中（称为 LibFile），然后在类声明中放入一个静态的初始值设定项：

```
static { System.loadLibrary("LibFile"); }
```

当加载并初始化这个类时，也将加载实现本机方法所需的本机代码。

如果用相同库名多次调用此方法，则忽略第二次及后续的调用。

参数：
libname - 库名。
抛出：

SecurityException - 如果安全管理器存在，并且其 checkLink 方法不允许加载指定的动态库
UnsatisfiedLinkError - 如果库不存在。
NullPointerException - 如果 libname 为 null
另请参见：
SecurityException, SecurityManager.checkLink(java.lang.String)

getLocalizedInputStream

```
@Deprecated
    public InputStream getLocalizedInputStream(InputStream in)
```

已过时。 *从 JDK 1.1 开始，将本地编码字节流转换为 Unicode 字符流的首选方法是使用 InputStreamReader 和 BufferedReader 类。*
创建输入流的本地化版本。此方法获取 InputStream，并返回除本地化外其他所有方面都和变量等效的 InputStream，这些方面包括：作为本地字符集中的字符从流中被读取，并将它们从本地字符集自动转换为 Unicode。

如果参数已经是本地化流，则可作为结果返回。

参数：
in - 要本地化的 InputStream
返回：
已本地化的输入流
另请参见：
InputStream, BufferedReader.BufferedReader(java.io.Reader), InputStreamReader.InputStreamReader(java.io.InputStream)

getLocalizedOutputStream

```
@Deprecated
    public OutputStream getLocalizedOutputStream(OutputStream out)
```

已过时。 *从 JDK 1.1 开始，将 Unicode 字符流转换为本地编码字节流的首选方法是使用 OutputStreamWriter、BufferedWriter 和 PrintWriter 类。*

创建输出流的本地化版本。此方法获取 OutputStream，并返回除本地化外其他所有方面都和变量等效的 OutputStream，这些方面包括：作为 Unicode 字符被写入流中，并被自动转换为本地字符集。

如果参数已经是本地流，则可作为结果返回。

参数：
out - 要本地化的 OutputStream
返回：
已本地化的输出流

String 类

String 类代表字符串。Java 程序中的所有字符串字面值（如 "abc" ）都作为此类的实例实现。

字符串是常量；它们的值在创建之后不能更改。字符串缓冲区支持可变的字符串。因为 String 对象是不可变的，所以可以共享。例如：

```
String str = "abc";
```

等效于：

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

下面给出了一些如何使用字符串的更多示例：

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

String 类包括的方法可用于检查序列的单个字符、比较字符串、搜索字符串、提取子字符串、创建字符串副本并将所有字符全部转换为大写或小写。大小写映射基于 Character 类指定的 Unicode 标准版。

Java 语言提供对字符串串联符号（"+"）以及将其他对象转换为字符串的特殊支持。字符串串联是通过 StringBuilder（或 StringBuffer）类及其 append 方法实现的。字符串转换是

通过 toString 方法实现的，该方法由 Object 类定义，并可被 Java 中的所有类继承。有关字符串串联和转换的更多信息，请参阅 Gosling、Joy 和 Steele 合著的 The Java Language Specification。

除非另行说明，否则将 null 参数传递给此类中的构造方法或方法将抛出 NullPointerException。

String 表示一个 UTF-16 格式的字符串，其中的*增补字符* 由*代理项对* 表示（有关详细信息，请参阅 Character 类中的 Unicode 字符表示形式）。索引值是指 char 代码单元，因此增补字符在 String 中占用两个位置。

String 类提供处理 Unicode 代码点（即字符）和 Unicode 代码单元（即 char 值）的方法。

字段详细信息

CASE_INSENSITIVE_ORDER

```
public static final Comparator<String> CASE_INSENSITIVE_ORDER
```

一个对 String 对象进行排序的 Comparator，作用与 compareToIgnoreCase 相同。此比较器是可序列化的。

注意，Comparator *不* 考虑语言环境，因此可能导致在某些语言环境中的排序效果不理想。java.text 包提供 *Collator* 完成与语言环境有关的排序。

构造方法详细信息

String

```
public String()
```

初始化一个新创建的 String 对象，使其表示一个空字符序列。注意，由于 String 是不可变的，所以无需使用此构造方法。

String

```
public String(String original)
```

初始化一个新创建的 String 对象，使其表示一个与参数相同的字符序列；换句话说，新创建的字符串是该参数字符串的副本。由于 String 是不可变的，所以无需使用此构造方法，除非需要 original 的显式副本。

参数：

original - 一个 String。

String

```
public String(char[] value)
```

分配一个新的 String，使其表示字符数组参数中当前包含的字符序列。该字符数组的内容已被复制；后续对字符数组的修改不会影响新创建的字符串。

参数：

value - 字符串的初始值

String

```
public String(char[] value,
              int offset,
              int count)
```

分配一个新的 String，它包含取自字符数组参数一个子数组的字符。offset 参数是子数组第一个字符的索引，count 参数指定子数组的长度。该子数组的内容已被复制；后续对字符数组的修改不会影响新创建的字符串。

参数：

value - 作为字符源的数组。

offset - 初始偏移量。

count - 长度。

抛出：

IndexOutOfBoundsException - 如果 offset 和 count 参数索引字符超出 value 数组的范围。

String

```
public String(int[] codePoints,
```

```
                int offset,
                int count)
```

分配一个新的 String，它包含 Unicode 代码点数组参数一个子数组的字符。offset 参数是该子数组第一个代码点的索引，count 参数指定子数组的长度。将该子数组的内容转换为 char；后续对 int 数组的修改不会影响新创建的字符串。

参数：

codePoints - 作为 Unicode 代码点的源的数组。

offset - 初始偏移量。

count - 长度。

抛出：

IllegalArgumentException - 如果在 codePoints 中发现任何无效的 Unicode 代码点

IndexOutOfBoundsException - 如果 offset 和 count 参数索引字符超出 codePoints 数组的范围。

从以下版本开始：

1.5

String

```
@Deprecated
    public String(byte[] ascii,
                  int hibyte,
                  int offset,
                  int count)
```

已过时。 *该方法无法将字节正确地转换为字符。从 JDK 1.1 开始，完成该转换的首选方法是使用带有 Charset、字符集名称，或使用平台默认字符集的 String 构造方法。*

分配一个新的 String，它是根据一个 8 位整数值数组的子数组构造的。

offset 参数是该子数组的第一个 byte 的索引，count 参数指定子数组的长度。

子数组中的每个 byte 都按照上述方法转换为 char。

参数：

ascii - 要转换为字符的 byte。

hibyte - 每个 16 位 Unicode 代码单元的前 8 位。
offset - 初始偏移量。
count - 长度。
抛出：
IndexOutOfBoundsException - 如果 offset 或 count 参数无效。
另请参见：
String(byte[], int), String(byte[], int, int, java.lang.String), String(byte[], int, int, java.nio.charset.Charset), String(byte[], int, int), String(byte[], java.lang.String), String(byte[], java.nio.charset.Charset), String(byte[])

String

```
@Deprecated
    public String(byte[] ascii,
                  int hibyte)
```

已过时。 *该方法无法将字节正确地转换为字符。从 JDK 1.1 开始，完成该转换的首选方法是使用带有 Charset、字符集名称，或使用平台默认字符集的 String 构造方法。*
分配一个新的 String，它包含根据一个 8 位整数值数组构造的字符。所得字符串中的每个字符 *c* 都是根据 byte 数组中的相应组件 *b* 构造的，如下所示：

$$c == (char)(((hibyte \ \& \ 0xff) << 8) \mid (b \ \& \ 0xff))$$

参数：
ascii - 要转换为字符的 byte。
hibyte - 每个 16 位 Unicode 代码单元的前 8 位。
另请参见：
String(byte[], int, int, java.lang.String), String(byte[], int, int, java.nio.charset.Charset), String(byte[], int, int), String(byte[], java.lang.String), String(byte[], java.nio.charset.Charset), String(byte[])

String

```
public String(byte[] bytes,
              int offset,
              int length,
              String charsetName)
       throws UnsupportedEncodingException
```

通过使用指定的字符集解码指定的 byte 子数组，构造一个新的 String。新 String 的长度是一个字符集函数，因此可能不等于子数组的长度。

当给定 byte 在给定字符集中无效的情况下，此构造方法的行为没有指定。如果需要对解码过程进行更多控制，则应该使用 CharsetDecoder 类。

参数：
bytes - 要解码为字符的 byte
offset - 要解码的第一个 byte 的索引
length - 要解码的 byte 数
charsetName - 受支持 charset 的名称
抛出：
UnsupportedEncodingException - 如果指定的字符集不受支持
IndexOutOfBoundsException - 如果 offset 和 length 参数索引字符超出 bytes 数组的范围
从以下版本开始：
JDK1.1

String

```
public String(byte[] bytes,
              int offset,
              int length,
              Charset charset)
```

通过使用指定的 charset 解码指定的 byte 子数组，构造一个新的 String。新 String 的长度是字符集的函数，因此可能不等于子数组的长度。

此方法总是使用此字符集的默认替代字符串替代错误输入 (malformed-input) 和不可映射字符 (unmappable-character) 序列。如果需要对解码过程进行更多控制，则应该使用 CharsetDecoder 类。

参数：
bytes - 要解码为字符的 byte
offset - 要解码的第一个 byte 的索引
length - 要解码的 byte 数
charset - 用来解码 bytes 的 charset
抛出：
IndexOutOfBoundsException - 如果 offset 和 length 参数索引字符超出 bytes 数组的边界
从以下版本开始：
1.6

String

```
public String(byte[] bytes,
              String charsetName)
       throws UnsupportedEncodingException
```

通过使用指定的 charset 解码指定的 byte 数组，构造一个新的 String。新 String 的长度是字符集的函数，因此可能不等于 byte 数组的长度。

当给定 byte 在给定字符集中无效的情况下，此构造方法的行为没有指定。如果需要对解码过程进行更多控制，则应该使用 CharsetDecoder 类。

参数：
bytes - 要解码为字符的 byte
charsetName - 受支持的 charset 的名称
抛出：
UnsupportedEncodingException - 如果指定字符集不受支持
从以下版本开始：
JDK1.1

String

```
public String(byte[] bytes,
              Charset charset)
```

通过使用指定的 charset 解码指定的 byte 数组，构造一个新的 String。新 String 的长度是字符集的函数，因此可能不等于 byte 数组的长度。

此方法总是使用此字符集的默认替代字符串替代错误输入和不可映射字符序列。如果需要对解码过程进行更多控制，则应该使用 CharsetDecoder 类。

参数：
bytes - 要解码为字符的 byte
charset - 要用来解码 bytes 的 charset
从以下版本开始：
1.6

String

```
public String(byte[] bytes,
              int offset,
              int length)
```

通过使用平台的默认字符集解码指定的 byte 子数组，构造一个新的 String。新 String 的长度是字符集的函数，因此可能不等于该子数组的长度。

当给定 byte 在给定字符集中无效的情况下，此构造方法的行为没有指定。如果需要对解码过程进行更多控制，则应该使用 CharsetDecoder 类。

参数：
bytes - 要解码为字符的 byte
offset - 要解码的第一个 byte 的索引
length - 要解码的 byte 数
抛出：
IndexOutOfBoundsException - 如果 offset 和 length 参数索引字符超出 bytes 数组的范围
从以下版本开始：
JDK1.1

String

```
public String(byte[] bytes)
```

通过使用平台的默认字符集解码指定的 byte 数组，构造一个新的 String。新 String 的长度是字符集的函数，因此可能不等于 byte 数组的长度。

当给定 byte 在给定字符集中无效的情况下，此构造方法的行为没有指定。如果需要对解码过程进行更多控制，则应该使用 CharsetDecoder 类。

参数：
bytes - 要解码为字符的 byte
从以下版本开始：
JDK1.1

String

```
public String(StringBuffer buffer)
```

分配一个新的字符串，它包含字符串缓冲区参数中当前包含的字符序列。该字符串缓冲区的内容已被复制，后续对它的修改不会影响新创建的字符串。
参数：
buffer - 一个 StringBuffer

String

```
public String(StringBuilder builder)
```

分配一个新的字符串，它包含字符串生成器参数中当前包含的字符序列。该字符串生成器的内容已被复制，后续对它的修改不会影响新创建的字符串。

提供此构造方法是为了简化到 StringBuilder 的迁移。通过 toString 方法从字符串生成器中获取字符串可能运行的更快，因此通常作为首选。

参数：
builder - 一个 StringBuilder
从以下版本开始：

1.5

方法详细信息

length

public int length()

返回此字符串的长度。长度等于字符串中 Unicode 代码单元的数量。

指定者：

接口 CharSequence 中的 length

返回：

此对象表示的字符序列的长度。

---

isEmpty

public boolean isEmpty()

当且仅当 length() 为 0 时返回 true。

返回：

如果 length() 为 0，则返回 true；否则返回 false。

从以下版本开始：

1.6

---

charAt

public char charAt(int index)

返回指定索引处的 char 值。索引范围为从 0 到 length() - 1。序列的第一个 char 值位于索引 0 处，第二个位于索引 1 处，依此类推，这类似于数组索引。

如果索引指定的 char 值是代理项，则返回代理项值。

指定者：

接口 CharSequence 中的 charAt

参数：

index - char 值的索引。

返回：
此字符串指定索引处的 char 值。第一个 char 值位于索引 0 处。
抛出：
IndexOutOfBoundsException - 如果 index 参数为负或小于此字符串的长度。

codePointAt

```
public int codePointAt(int index)
```

返回指定索引处的字符（Unicode 代码点）。索引引用 char 值（Unicode 代码单元），其范围从 0 到 length() - 1。

如果给定索引指定的 char 值属于高代理项范围，则后续索引小于此 String 的长度；如果后续索引处的 char 值属于低代理项范围，则返回该代理项对相应的增补代码点。否则，返回给定索引处的 char 值。

参数：
index - char 值的索引
返回：
index 处字符的代码点值
抛出：
IndexOutOfBoundsException - 如果 index 参数为负或小于此字符串的长度。
从以下版本开始：
1.5

codePointBefore

```
public int codePointBefore(int index)
```

返回指定索引之前的字符(Unicode 代码点)。索引引用 char 值(Unicode 代码单元)，其范围从 1 到 length。

如果 (index - 1) 处的 char 值属于低代理项范围，则 (index - 2) 为非负；如果 (index - 2) 处的 char 值属于高低理项范围，则返回该代理项对的增补代码点值。如果 index－1 处的 char 值是未配对的低（高）代理项，则返回代理项值。

参数：
index - 应返回的代码点之后的索引
返回：
给定索引前面的 Unicode 代码点。
抛出：
IndexOutOfBoundsException - 如果 index 参数小于 1 或大于此字符串的长度。
从以下版本开始：
1.5

---

codePointCount

```
public int codePointCount(int beginIndex,
                          int endIndex)
```

返回此 String 的指定文本范围中的 Unicode 代码点数。文本范围始于指定的 beginIndex，一直到索引 endIndex - 1 处的 char。因此，该文本范围的长度（用 char 表示）是 endIndex-beginIndex。该文本范围内每个未配对的代理项计为一个代码点。
参数：
beginIndex - 文本范围的第一个 char 的索引。
endIndex - 文本范围的最后一个 char 之后的索引。
返回：
指定文本范围中 Unicode 代码点的数量
抛出：
IndexOutOfBoundsException - 如果 beginIndex 为负，或 endIndex 大于此 String 的长度，或 beginIndex 大于 endIndex。
从以下版本开始：
1.5

---

offsetByCodePoints

```
public int offsetByCodePoints(int index,
                              int codePointOffset)
```

返回此 String 中从给定的 index 处偏移 codePointOffset 个代码点的索引。文本范围内由 index 和 codePointOffset 给定的未配对代理项各计为一个代码点。

参数：
index - 要偏移的索引
codePointOffset - 代码点中的偏移量
返回：
String 的索引
抛出：
IndexOutOfBoundsException - 如果 index 为负或大于此 String 的长度；或者 codePointOffset 为正，且以 index 开头子字符串的代码点比 codePointOffset 少；如果 codePointOffset 为负，且 index 前面子字符串的代码点比 codePointOffset 的绝对值少。
从以下版本开始：
1.5

---

getChars

```
public void getChars(int srcBegin,
                     int srcEnd,
                     char[] dst,
                     int dstBegin)
```

将字符从此字符串复制到目标字符数组。

要复制的第一个字符位于索引 srcBegin 处；要复制的最后一个字符位于索引 srcEnd-1 处（因此要复制的字符总数是 srcEnd-srcBegin）。要复制到 dst 子数组的字符从索引 dstBegin 处开始，并结束于索引：

```
dstbegin + (srcEnd-srcBegin) - 1
```

参数：
srcBegin - 字符串中要复制的第一个字符的索引。
srcEnd - 字符串中要复制的最后一个字符之后的索引。
dst - 目标数组。
dstBegin - 目标数组中的起始偏移量。
抛出：
IndexOutOfBoundsException - 如果下列任何一项为 true：

- srcBegin 为负。
- srcBegin 大于 srcEnd

- srcEnd 大于此字符串的长度
- dstBegin 为负
- dstBegin+(srcEnd-srcBegin) 大于 dst.length

getBytes

```
@Deprecated
    public void getBytes(int srcBegin,
                         int srcEnd,
                         byte[] dst,
                         int dstBegin)
```

已过时。 *该方法无法将字符正确转换为字节。从 JDK 1.1 起，完成该转换的首选方法是通过 getBytes() 方法，该方法使用平台的默认字符集。*将字符从此字符串复制到目标 byte 数组中。每个 byte 接收相应字符的 8 个低位。不复制每个字符的高位，它们不参与任何方式的转换。

要复制的第一个字符位于索引 srcBegin 处；要复制的最后一个字符位于索引 srcEnd-1 处。要复制的字符总数为 srcEnd-srcBegin。将转换为 byte 的字符复制到 dst 的子数组中，从索引 dstBegin 处开始，并结束于索引：

```
dstbegin + (srcEnd-srcBegin) - 1
```

参数：
srcBegin - 字符串中要复制的第一个字符的索引
srcEnd - 字符串中要复制的最后一个字符之后的索引
dst - 目标数组
dstBegin - 目标数组中的起始偏移量
抛出：
IndexOutOfBoundsException - 如果下列任何一项为 true：

- srcBegin 为负
- srcBegin 大于 srcEnd
- srcEnd 大于此 String 的长度
- dstBegin 为负
- dstBegin+(srcEnd-srcBegin) 大于 dst.length

getBytes

```
public byte[] getBytes(String charsetName)
                    throws UnsupportedEncodingException
```

使用指定的字符集将此 String 编码为 byte 序列，并将结果存储到一个新的 byte 数组中。

当此字符串不能使用给定的字符集编码时，此方法的行为没有指定。如果需要对编码过程进行更多控制，则应该使用 CharsetEncoder 类。

参数：
charsetName - 受支持的 charset 名称
返回：
所得 byte 数组
抛出：
UnsupportedEncodingException - 如果指定的字符集不受支持
从以下版本开始：
JDK1.1

getBytes

```
public byte[] getBytes(Charset charset)
```

使用给定的 charset 将此 String 编码到 byte 序列，并将结果存储到新的 byte 数组。

此方法总是使用此字符集的默认替代 byte 数组替代错误输入和不可映射字符序列。如果需要对编码过程进行更多控制，则应该使用 CharsetEncoder 类。

参数：
charset - 用于编码 String 的 Charset
返回：
所得 byte 数组
从以下版本开始：
1.6

getBytes

public byte[] getBytes()

使用平台的默认字符集将此 String 编码为 byte 序列，并将结果存储到一个新的 byte 数组中。

当此字符串不能使用默认的字符集编码时，此方法的行为没有指定。如果需要对编码过程进行更多控制，则应该使用 CharsetEncoder 类。

返回：
所得 byte 数组
从以下版本开始：
JDK1.1

---

equals

public boolean equals(Object anObject)

将此字符串与指定的对象比较。当且仅当该参数不为 null，并且是与此对象表示相同字符序列的 String 对象时，结果才为 true。
覆盖：
类 Object 中的 equals
参数：
anObject - 与此 String 进行比较的对象。
返回：
如果给定对象表示的 String 与此 String 相等，则返回 true；否则返回 false。
另请参见：
compareTo(String), equalsIgnoreCase(String)

---

contentEquals

public boolean contentEquals(StringBuffer sb)

将此字符串与指定的 StringBuffer 比较。当且仅当此 String 与指定 StringBuffer 表示相同的字符序列时，结果才为 true。
参数：
sb - 要与此 String 比较的 StringBuffer。

返回：
如果此 String 与指定 StringBuffer 表示相同的字符序列，则返回 true；否则返回 false。
从以下版本开始：
1.4

---

contentEquals

```
public boolean contentEquals(CharSequence cs)
```

将此字符串与指定的 CharSequence 比较。当且仅当此 String 与指定序列表示相同的 char 值序列时，结果才为 true。
参数：
cs - 要与此 String 比较的序列
返回：
如果此 String 与指定序列表示相同的 char 值序列，则返回 true；否则返回 false。
从以下版本开始：
1.5

---

equalsIgnoreCase

```
public boolean equalsIgnoreCase(String anotherString)
```

将此 String 与另一个 String 比较，不考虑大小写。如果两个字符串的长度相同，并且其中的相应字符都相等（忽略大小写），则认为这两个字符串是相等的。

在忽略大小写的情况下，如果下列至少一项为 true，则认为 c1 和 c2 这两个字符相同。

- 这两个字符相同（使用 == 运算符进行比较）。
- 对每个字符应用方法 Character.toUpperCase(char) 生成相同的结果。
- 对每个字符应用方法 Character.toLowerCase(char) 生成相同的结果。

参数：
anotherString - 与此 String 进行比较的 String。

返回：
如果参数不为 null，且这两个 String 相等（忽略大小写），则返回 true；否则返回 false。
另请参见：
equals(Object)

compareTo

```
public int compareTo(String anotherString)
```

按字典顺序比较两个字符串。该比较基于字符串中各个字符的 Unicode 值。按字典顺序将此 String 对象表示的字符序列与参数字符串所表示的字符序列进行比较。如果按字典顺序此 String 对象位于参数字符串之前，则比较结果为一个负整数。如果按字典顺序此 String 对象位于参数字符串之后，则比较结果为一个正整数。如果这两个字符串相等，则结果为 0；compareTo 只在方法 equals(Object) 返回 true 时才返回 0。

这是字典排序的定义。如果这两个字符串不同，那么它们要么在某个索引处的字符不同（该索引对二者均为有效索引），要么长度不同，或者同时具备这两种情况。如果它们在一个或多个索引位置上的字符不同，假设 *k* 是这类索引的最小值；则在位置 *k* 上具有较小值的那个字符串（使用 < 运算符确定），其字典顺序在其他字符串之前。在这种情况下，compareTo 返回这两个字符串在位置 k 处两个 char 值的差，即值：

```
this.charAt(k)-anotherString.charAt(k)
```

如果没有字符不同的索引位置，则较短字符串的字典顺序在较长字符串之前。在这种情况下，compareTo 返回这两个字符串长度的差，即值：

```
this.length()-anotherString.length()
```

指定者：
接口 Comparable<String> 中的 compareTo
参数：
anotherString - 要比较的 String。
返回：
如果参数字符串等于此字符串，则返回值 0；如果此字符串按字典顺序小于字符串参数，则返回一个小于 0 的值；如果此字符串按字典顺序大于字符串参数，则返回一个大于 0 的值。

compareToIgnoreCase

```
public int compareToIgnoreCase(String str)
```

按字典顺序比较两个字符串，不考虑大小写。此方法返回一个整数，其符号与使用规范化的字符串调用 compareTo 所得符号相同，规范化字符串的大小写差异已通过对每个字符调用 Character.toLowerCase(Character.toUpperCase(character)) 消除。

注意，此方法*不* 考虑语言环境，因此可能导致在某些语言环境中的排序效果不理想。java.text 包提供 *Collators* 完成与语言环境有关的排序。

参数：
str - 要比较的 String。
返回：
根据指定 String 大于、等于还是小于此 String（不考虑大小写），分别返回一个负整数、0 或一个正整数。
从以下版本开始：
1.2
另请参见：
Collator.compare(String, String)

regionMatches

```
public boolean regionMatches(int toffset,
                             String other,
                             int ooffset,
                             int len)
```

测试两个字符串区域是否相等。

将此 String 对象的一个子字符串与参数 other 的一个子字符串进行比较。如果这两个子字符串表示相同的字符序列，则结果为 true。要比较的此 String 对象的子字符串从索引 toffset 处开始，长度为 len。要比较的 other 的子字符串从索引 ooffset 处开始，长度为 len。当且仅当下列至少一项为 true 时，结果才为 false ：

- toffset 为负。

- ooffset 为负。
- toffset+len 大于此 String 对象的长度。
- ooffset+len 大于另一个参数的长度。
- 存在某个小于 len 的非负整数 *k*，它满足：
  this.charAt(toffset+*k*) != other.charAt(ooffset+*k*)

参数：
toffset - 字符串中子区域的起始偏移量。
other - 字符串参数。
ooffset - 字符串参数中子区域的起始偏移量。
len - 要比较的字符数。
返回：
如果此字符串的指定子区域完全匹配字符串参数的指定子区域，则返回 true；否则返回 false。

regionMatches

```
public boolean regionMatches(boolean ignoreCase,
                             int toffset,
                             String other,
                             int ooffset,
                             int len)
```

测试两个字符串区域是否相等。

将此 String 对象的子字符串与参数 other 的子字符串进行比较。如果这两个子字符串表示相同的字符序列，则结果为 true，当且仅当 ignoreCase 为 true 时忽略大小写。要比较的此 String 对象的子字符串从索引 toffset 处开始，长度为 len。要比较的 other 的子字符串从索引 ooffset 处开始，长度为 len。当且仅当下列至少一项为 true 时，结果才为 false：

- toffset 为负。
- ooffset 为负。
- toffset+len 大于此 String 对象的长度。
- ooffset+len 大于另一个参数的长度。
- ignoreCase 为 false，且存在某个小于 len 的非负整数 *k*，即：
- this.charAt(toffset+k) != other.charAt(ooffset+k)

- ignoreCase 为 true，且存在某个小于 len 的非负整数 *k*，即：
- Character.toLowerCase(this.charAt(toffset+k)) !=
- Character.toLowerCase(other.charAt(ooffset+k))

以及：

Character.toUpperCase(this.charAt(toffset+k)) !=
Character.toUpperCase(other.charAt(ooffset+k))

参数：
ignoreCase - 如果为 true，则比较字符时忽略大小写。
toffset - 此字符串中子区域的起始偏移量。
other - 字符串参数。
toffset - 字符串参数中子区域的起始偏移量。
len - 要比较的字符数。
返回：
如果此字符串的指定子区域匹配字符串参数的指定子区域，则返回 true；否则返回 false。是否完全匹配或考虑大小写取决于 ignoreCase 参数。

startsWith

```
public boolean startsWith(String prefix,
                          int toffset)
```

测试此字符串从指定索引开始的子字符串是否以指定前缀开始。
参数：
prefix - 前缀。
toffset - 在此字符串中开始查找的位置。
返回：
如果参数表示的字符序列是此对象从索引 toffset 处开始的子字符串前缀，则返回 true；否则返回 false。如果 toffset 为负或大于此 String 对象的长度，则结果为 false；否则结果与以下表达式的结果相同：

this.substring(toffset).startsWith(prefix)

startsWith

public boolean startsWith(String prefix)

测试此字符串是否以指定的前缀开始。

参数：

prefix - 前缀。

返回：

如果参数表示的字符序列是此字符串表示的字符序列的前缀，则返回 true；否则返回 false。还要注意，如果参数是空字符串，或者等于此 String 对象（用 equals(Object) 方法确定），则返回 true。

从以下版本开始：

1. 0

---

endsWith

public boolean endsWith(String suffix)

测试此字符串是否以指定的后缀结束。

参数：

suffix - 后缀。

返回：

如果参数表示的字符序列是此对象表示的字符序列的后缀，则返回 true；否则返回 false。注意，如果参数是空字符串，或者等于此 String 对象（用 equals(Object) 方法确定），则结果为 true。

---

hashCode

public int hashCode()

返回此字符串的哈希码。String 对象的哈希码根据以下公式计算：

s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]

使用 int 算法，这里 s[i] 是字符串的第 *i* 个字符，n 是字符串的长度，^ 表示求幂。（空字符串的哈希值为 0。）

覆盖：

类 Object 中的 hashCode

返回：

此对象的哈希码值。

另请参见：
Object.equals(java.lang.Object), Hashtable

indexOf

```
public int indexOf(int ch)
```

返回指定字符在此字符串中第一次出现处的索引。如果在此 String 对象表示的字符序列中出现值为 ch 的字符，则返回第一次出现该字符的索引（以 Unicode 代码单元表示）。对于 0 到 0xFFFF（包括 0 和 0xFFFF）范围内的 ch 的值，返回值是
this.charAt(*k*) == ch

为 true 的最小 *k* 值。对于其他 ch 值，返回值是
this.codePointAt(*k*) == ch

为 true 最小 *k* 值。无论哪种情况，如果此字符串中没有这样的字符，则返回 -1。

参数：
ch - 一个字符（Unicode 代码点）。

返回：
在此对象表示的字符序列中第一次出现该字符的索引；如果未出现该字符，则返回 -1。

indexOf

```
public int indexOf(int ch,
                   int fromIndex)
```

返回在此字符串中第一次出现指定字符处的索引，从指定的索引开始搜索。

在此 String 对象表示的字符序列中，如果带有值 ch 的字符的索引不小于 fromIndex，则返回第一次出现该值的索引。对于 0 到 0xFFFF（包括 0 和 0xFFFF）范围内的 ch 值，返回值是

```
(this.charAt(k) == ch) && (k >= fromIndex)
```

为 true 的最小 *k* 值。对于其他 ch 值，返回值是

```
(this.codePointAt(k) == ch) && (k >= fromIndex)
```

为 true 的最小 *k* 值。无论哪种情况，如果此字符串中 fromIndex 或之后的位置没有这样的字符出现，则返回 -1。

fromIndex 的值没有限制。如果它为负，则与它为 0 的效果同样：将搜索整个字符串。如果它大于此字符串的长度，则与它等于此字符串长度的效果相同：返回 -1。

所有索引都在 char 值中指定（Unicode 代码单元）。

参数：
ch - 一个字符（Unicode 代码点）。
fromIndex - 开始搜索的索引。
返回：
在此对象表示的字符序列中第一次出现的大于或等于 fromIndex 的字符的索引；如果未出现该字符，则返回 -1。

---

lastIndexOf

```
public int lastIndexOf(int ch)
```

返回指定字符在此字符串中最后一次出现处的索引。对于 0 到 0xFFFF（包括 0 和 0xFFFF）范围内的 ch 的值，返回的索引（Unicode 代码单元）是

```
this.charAt(k) == ch
```

为 true 最大 *k* 值。对于其他 ch 值，返回值是

```
this.codePointAt(k) == ch
```

为 true 的最大 *k* 值。无论哪种情况，如果此字符串中没有这样的字符出现，则返回 -1。从最后一个字符开始反向搜索此 String。
参数：
ch - 一个字符（Unicode 代码点）。
返回：
在此对象表示的字符序列中最后一次出现该字符的索引；如果未出现该字符，则返回 -1。

lastIndexOf

```
public int lastIndexOf(int ch,
                       int fromIndex)
```

返回指定字符在此字符串中最后一次出现处的索引，从指定的索引处开始进行反向搜索。对于 0 到 0xFFFF（包括 0 和 0xFFFF）范围内的 ch 值，返回的索引是

```
(this.charAt(k) == ch) && (k <= fromIndex)
```

为 true 的最大 *k* 值。对于 ch 的其他值，返回值是

```
(this.codePointAt(k) == ch) && (k <= fromIndex)
```

为 true 的最大 *k* 值。无论哪种情况，如果此字符串中 fromIndex 或之前的位置没有这样的字符出现，则返回 -1。

所有的索引都以 char 值指定（Unicode 代码单元）。

参数：

ch - 一个字符（Unicode 代码点）。

fromIndex - 开始搜索的索引。fromIndex 的值没有限制。如果它大于等于此字符串的长度，则与它小于此字符串长度减 1 的效果相同：将搜索整个字符串。如果它为负，则与它为 -1 的效果相同：返回 -1。

返回：

在此对象表示的字符序列（小于等于 fromIndex）中最后一次出现该字符的索引；如果在该点之前未出现该字符，则返回 -1。

indexOf

```
public int indexOf(String str)
```

返回指定子字符串在此字符串中第一次出现处的索引。返回的整数是

```
this.startsWith(str, k)
```

为 true 的最小 *k* 值。

参数：

str - 任意字符串。

返回：

如果字符串参数作为一个子字符串在此对象中出现，则返回第一个这种子字符串的第一个字符的索引；如果它不作为一个子字符串出现，则返回 -1。

indexOf

```
public int indexOf(String str,
                   int fromIndex)
```

返回指定子字符串在此字符串中第一次出现处的索引，从指定的索引开始。返回的整数是满足下式的最小 k 值：

```
k >= Math.min(fromIndex, this.length()) && this.startsWith(str, k)
```

如果不存在这样的 *k* 值，则返回 -1。

参数：

str - 要搜索的子字符串。

fromIndex - 开始搜索的索引位置。

返回：

指定子字符串在此字符串中第一次出现处的索引，从指定的索引开始。

lastIndexOf

```
public int lastIndexOf(String str)
```

返回指定子字符串在此字符串中最右边出现处的索引。将最右边的空字符串 "" 视为出现在索引值 this.length() 处。返回的索引是

```
this.startsWith(str, k)
```

为 true 的最大 *k* 值。

参数：

str - 要搜索的子字符串。

返回：

如果字符串参数作为一个子字符串在此对象中出现一次或多次，则返回最后一个这种子字符串的第一个字符。如果它不作为一个子字符串出现，则返回 -1。

lastIndexOf

```
public int lastIndexOf(String str,
                       int fromIndex)
```

返回指定子字符串在此字符串中最后一次出现处的索引，从指定的索引开始反向搜索。返回的整数是满足下式的最大 *k* 值：

```
k <= Math.min(fromIndex,this.length()) && this.startsWith(str, k)
```

如果不存在这样的 *k* 值，则返回 -1。

参数：

str - 要搜索的子字符串。

fromIndex - 开始搜索的索引位置。

返回：

指定子字符串在此字符串中最后一次出现处的索引。

---

substring

```
public String substring(int beginIndex)
```

返回一个新的字符串，它是此字符串的一个子字符串。该子字符串从指定索引处的字符开始，直到此字符串末尾。

示例：

```
"unhappy".substring(2) returns "happy"
"Harbison".substring(3) returns "bison"
"emptiness".substring(9) returns "" (an empty string)
```

参数：

beginIndex - 起始索引（包括）。

返回：

指定的子字符串。

抛出：

IndexOutOfBoundsException - 如果 beginIndex 为负或大于此 String 对象的长度。

---

substring

```
public String substring(int beginIndex,
                        int endIndex)
```

返回一个新字符串，它是此字符串的一个子字符串。该子字符串从指定的 beginIndex 处开始，直到索引 endIndex - 1 处的字符。因此，该子字符串的长度为 endIndex-beginIndex。

示例：

```
"hamburger".substring(4, 8) returns "urge"
"smiles".substring(1, 5) returns "mile"
```

参数：
beginIndex - 起始索引（包括）。
endIndex - 结束索引（不包括）。
返回：
指定的子字符串。
抛出：
IndexOutOfBoundsException - 如果 beginIndex 为负，或 endIndex 大于此 String 对象的长度，或 beginIndex 大于 endIndex。

subSequence

```
public CharSequence subSequence(int beginIndex,
                                int endIndex)
```

返回一个新的字符序列，它是此序列的一个子序列。

此方法这种形式的调用：

```
str.subSequence(begin, end)
```

与以下调用的行为完全相同：

```
str.substring(begin, end)
```

定义此方法使 String 类能够实现 CharSequence 接口。
指定者：
接口 CharSequence 中的 subSequence
参数：
beginIndex - 起始索引（包括）。
endIndex - 结束索引（不包括）。

返回：
指定子序列。
抛出：
IndexOutOfBoundsException - 如果 beginIndex 或 endIndex 为负，如果 endIndex 大于 length() 或 beginIndex 大于 startIndex
从以下版本开始：
1.4

---

concat

```
public String concat(String str)
```

将指定字符串连接到此字符串的结尾。

如果参数字符串的长度为 0，则返回此 String 对象。否则，创建一个新的 String 对象，用来表示由此 String 对象表示的字符序列和参数字符串表示的字符序列连接而成的字符序列。

示例：

```
"cares".concat("s") returns "caress"
"to".concat("get").concat("her") returns "together"
```

参数：
str - 连接到此 String 结尾的 String。
返回：
一个字符串，它表示在此对象字符后连接字符串参数字符而成的字符。

---

replace

```
public String replace(char oldChar,
                      char newChar)
```

返回一个新的字符串，它是通过用 newChar 替换此字符串中出现的所有 oldChar 得到的。

如果 oldChar 在此 String 对象表示的字符序列中没有出现，则返回对此 String 对象的引用。否则，创建一个新的 String 对象，它所表示的

字符序列除了所有的 oldChar 都被替换为 newChar 之外，与此 String 对象表示的字符序列相同。

示例：

```
"mesquite in your cellar".replace('e', 'o')
        returns "mosquito in your collar"
"the war of baronets".replace('r', 'y')
        returns "the way of bayonets"
"sparring with a purple porpoise".replace('p', 't')
        returns "starring with a turtle tortoise"
"JonL".replace('q', 'x') returns "JonL" (no change)
```

参数：
oldChar - 原字符。
newChar - 新字符。
返回：
一个从此字符串派生的字符串，它将此字符串中的所有 oldChar 替代为 newChar。

matches

```
public boolean matches(String regex)
```

告知此字符串是否匹配给定的正则表达式。

调用此方法的 *str*.matches(*regex*) 形式与以下表达式产生的结果完全相同：

```
Pattern.matches(regex, str)
```

参数：
regex - 用来匹配此字符串的正则表达式
返回：
当且仅当此字符串匹配给定的正则表达式时，返回 true
抛出：
PatternSyntaxException - 如果正则表达式的语法无效
从以下版本开始：
1.4

另请参见：
Pattern

contains

```
public boolean contains(CharSequence s)
```

当且仅当此字符串包含指定的 char 值序列时，返回 true。
参数：
s - 要搜索的序列
返回：
如果此字符串包含 s，则返回 true，否则返回 false
抛出：
NullPointerException - 如果 s 为 null
从以下版本开始：
1.5

replaceFirst

```
public String replaceFirst(String regex,
                           String replacement)
```

使用给定的 replacement 替换此字符串匹配给定的正则表达式的第一个子字符串。

调用此方法的 *str*.replaceFirst(*regex*, *repl*) 形式与以下表达式产生的结果完全相同：

```
Pattern.compile(regex).matcher(str).replaceFirst(repl)
```

注意，在替代字符串中使用反斜杠 (\) 和美元符号 ($) 与将其视为字面值替代字符串所得的结果可能不同；请参阅 Matcher.replaceFirst(java.lang.String)。如有需要，可使用 Matcher.quoteReplacement(java.lang.String) 取消这些字符的特殊含义。

参数：
regex - 用来匹配此字符串的正则表达式
replacement - 用来替换第一个匹配项的字符串

返回：
所得 String
抛出：
PatternSyntaxException - 如果正则表达式的语法无效
从以下版本开始：
1.4
另请参见：
Pattern

replaceAll

```
public String replaceAll(String regex,
                         String replacement)
```

使用给定的 replacement 替换此字符串所有匹配给定的正则表达式的子字符串。

调用此方法的 *str*.replaceAll(*regex*, *repl*) 形式与以下表达式产生的结果完全相同：

```
Pattern.compile(regex).matcher(str).replaceAll(repl)
```

注意，在替代字符串中使用反斜杠 (\) 和美元符号 ($) 与将其视为字面值替代字符串所得的结果可能不同；请参阅 Matcher.replaceAll。如有需要，可使用 Matcher.quoteReplacement(java.lang.String) 取消这些字符的特殊含义。

参数：
regex - 用来匹配此字符串的正则表达式
replacement - 用来替换每个匹配项的字符串
返回：
所得 String
抛出：
PatternSyntaxException - 如果正则表达式的语法无效
从以下版本开始：
1.4
另请参见：
Pattern

replace

```
public String replace(CharSequence target,
                      CharSequence replacement)
```

使用指定的字面值替换序列替换此字符串所有匹配字面值目标序列的子字符串。该替换从字符串的开头朝末尾执行，例如，用 "b" 替换字符串 "aaa" 中的 "aa" 将生成 "ba" 而不是 "ab"。

参数：

target - 要被替换的 char 值序列

replacement - char 值的替换序列

返回：

所得 String

抛出：

NullPointerException - 如果 target 或 replacement 为 null。

从以下版本开始：

1.5

---

split

```
public String[] split(String regex,
                      int limit)
```

根据匹配给定的正则表达式来拆分此字符串。

此方法返回的数组包含此字符串的子字符串，每个子字符串都由另一个匹配给定表达式的子字符串终止，或者由此字符串末尾终止。数组中的子字符串按它们在此字符串中出现的顺序排列。如果表达式不匹配输入的任何部分，那么所得数组只具有一个元素，即此字符串。

limit 参数控制模式应用的次数，因此影响所得数组的长度。如果该限制 *n* 大于 0，则模式将被最多应用 *n* - 1 次，数组的长度将不会大于 *n*，而且数组的最后一项将包含所有超出最后匹配的定界符的输入。如果 *n* 为非正，那么模式将被应用尽可能多的次数，而且数组可以是任何长度。如果 *n* 为 0，那么模式将被应用尽可能多的次数，数组可以是任何长度，并且结尾空字符串将被丢弃。

例如，字符串 "boo:and:foo" 使用这些参数可生成以下结果：

Regex	Limit	结果
:	2	{ "boo", "and:foo" }
:	5	{ "boo", "and", "foo" }
:	-2	{ "boo", "and", "foo" }
o	5	{ "b", "", ":and:f", "", "" }
o	-2	{ "b", "", ":and:f", "", "" }
o	0	{ "b", "", ":and:f" }

调用此方法的 *str*.split(*regex*, *n*) 形式与以下表达式产生的结果完全相同：

```
Pattern.compile(regex).split(str, n)
```

参数：

regex - 定界正则表达式

limit - 结果阈值，如上所述

返回：

字符串数组，它是根据给定正则表达式的匹配拆分此字符串确定的

抛出：

PatternSyntaxException - 如果正则表达式的语法无效

从以下版本开始：

1.4

另请参见：

Pattern

---

split

```
public String[] split(String regex)
```

根据给定正则表达式的匹配拆分此字符串。

该方法的作用就像是使用给定的表达式和限制参数 0 来调用两参数 split 方法。因此，所得数组中不包括结尾空字符串。

例如，字符串 "boo:and:foo" 使用这些表达式可生成以下结果：

Regex	结果
:	{ "boo", "and", "foo" }

o	{ "b", "", ":and:f" }

参数：

regex - 定界正则表达式

返回：

字符串数组，它是根据给定正则表达式的匹配拆分此字符串确定的

抛出：

PatternSyntaxException - 如果正则表达式的语法无效

从以下版本开始：

1.4

另请参见：

Pattern

---

toLowerCase

public String toLowerCase(Locale locale)

使用给定 Locale 的规则将此 String 中的所有字符都转换为小写。大小写映射关系基于 Character 类指定的 Unicode 标准版。由于大小写映射关系并不总是 1:1 的字符映射关系，因此所得 String 的长度可能不同于原 String。

下表中给出了几个小写映射关系的示例：

语言环境的代码	大写字母	小写字母	描述
tr (Turkish)	\u0130	\u0069	大写字母 I，上面有点 -> 小写字母 i
tr (Turkish)	\u0049	\u0131	大写字母 I -> 小写字母 i，无点
(all)	French Fries	french fries	将字符串中的所有字符都小写
(all)	ΙΧΘΥΣ	ιχθυς	将字符串中的所有字符都小写

参数：

locale - 使用此语言环境的大小写转换规则

返回：

要转换为小写的 String。

从以下版本开始：

1.1

另请参见：
toLowerCase(), toUpperCase(), toUpperCase(Locale)

toLowerCase

public String toLowerCase()

使用默认语言环境的规则将此 String 中的所有字符都转换为小写。这等效于调用 toLowerCase(Locale.getDefault())。

注： 此方法与语言环境有关，如果用于应独立于语言环境解释的字符串，则可能生成不可预料的结果。示例有编程语言标识符、协议键、HTML 标记。例如，"TITLE".toLowerCase() 在 Turkish（土耳其语）语言环境中返回 "t?tle"，其中“?”是 LATIN SMALL LETTER DOTLESS I 字符。对于与语言环境有关的字符，要获得正确的结果，请使用

toLowerCase(Locale.ENGLISH)。

返回：
要转换为小写的 String。
另请参见：
toLowerCase(Locale)

toUpperCase

public String toUpperCase(Locale locale)

使用给定 Locale 的规则将此 String 中的所有字符都转换为大写。大小写映射关系基于 Character 类指定的 Unicode 标准版。由于大小写映射关系并不总是 1:1 的字符映射关系，因此所得 String 的长度可能不同于原 String。

下表中给出了几个与语言环境有关和 1:M 大小写映射关系的一些示例。

语言环境的代码	小写	大写	描述
tr (Turkish)	\u0069	\u0130	小写字母 i -> 大写字母 I，上面有点
tr (Turkish)	\u0131	\u0049	小写字母 i，无点 -> 大写字

			母 I
(all)	\u00df	\u0053 \u0053	小写字母 sharp s -> 两个字母：SS
(all)	Fahrvergnügen	FAHRVERGNÜN	

参数：
locale - 使用此语言环境的大小写转换规则
返回：
要转换为大写的 String。
从以下版本开始：
1.1
另请参见：
toUpperCase(), toLowerCase(), toLowerCase(Locale)

toUpperCase

```
public String toUpperCase()
```

使用默认语言环境的规则将此 String 中的所有字符都转换为大写。此方法等效于 toUpperCase(Locale.getDefault())。

注： 此方法与语言环境有关，如果用于应独立于语言环境解释的字符串，则可能生成不可预料的结果。示例有编程语言标识符、协议键、HTML 标记。例如，"title".toUpperCase() 在 Turkish（土耳其语）语言环境中返回 "T?TLE"，其中“?”是 LATIN CAPITAL LETTER I WITH DOT ABOVE 字符。对于与语言环境有关的字符，要获得正确的结果，请使用

```
toUpperCase(Locale.ENGLISH)。
```

返回：
要转换为大写的 String。
另请参见：
toUpperCase(Locale)

trim

```
public String trim()
```

返回字符串的副本，忽略前导空白和尾部空白。

如果此 String 对象表示一个空字符序列，或者此 String 对象表示的字符序列的第一个和最后一个字符的代码都大于 '\u0020'（空格字符），则返回对此 String 对象的引用。

否则，若字符串中没有代码大于 '\u0020' 的字符，则创建并返回一个表示空字符串的新 String 对象。

否则，假定 *k* 为字符串中代码大于 '\u0020' 的第一个字符的索引，*m* 为字符串中代码大于 '\u0020' 的最后一个字符的索引。创建一个新的 String 对象，它表示此字符串中从索引 *k* 处的字符开始，到索引 *m* 处的字符结束的子字符串，即 this.substring(*k*, *m*+1) 的结果。

此方法可用于截去字符串开头和末尾的空白（如上所述）。

返回：
此字符串移除了前导和尾部空白的副本；如果没有前导和尾部空白，则返回此字符串。

toString

```
public String toString()
```

返回此对象本身（它已经是一个字符串！）。
指定者：
接口 CharSequence 中的 toString
覆盖：
类 Object 中的 toString
返回：
字符串本身。

toCharArray

```
public char[] toCharArray()
```

将此字符串转换为一个新的字符数组。
返回：
一个新分配的字符数组，它的长度是此字符串的长度，它的内容被初始化为包含此字符串表示的字符序列。

format

```
public static String format(String format,
                            Object... args)
```

使用指定的格式字符串和参数返回一个格式化字符串。

始终使用 Locale.getDefault() 返回的语言环境。

参数：
format - 格式字符串
args - 格式字符串中由格式说明符引用的参数。如果还有格式说明符以外的参数，则忽略这些额外的参数。参数的数目是可变的，可以为 0。参数的最大数目受 Java Virtual Machine Specification 所定义的 Java 数组最大维度的限制。有关 null 参数的行为依赖于转换。
返回：
一个格式化字符串
抛出：
IllegalFormatException - 如果格式字符串中包含非法语法、与给定的参数不兼容的格式说明符，格式字符串给定的参数不够，或者存在其他非法条件。有关所有可能的格式化错误的规范，请参阅 formatter 类规范的详细信息 一节。
NullPointerException - 如果 format 为 null
从以下版本开始：
1.5
另请参见：
Formatter

---

format

```
public static String format(Locale l,
                            String format,
                            Object... args)
```

使用指定的语言环境、格式字符串和参数返回一个格式化字符串。
参数：
l - 格式化过程中要应用的语言环境。如果 l 为 null，则不进行本地化。
format - 格式字符串

args - 格式字符串中由格式说明符引用的参数。如果还有格式说明符以外的参数，则忽略这些额外的参数。参数的数目是可变的，可以为 0。参数的最大数目受 Java Virtual Machine Specification 所定义的 Java 数组最大维度的限制。有关 null 参数的行为依赖于转换。

返回：

一个格式化字符串

抛出：

IllegalFormatException - 如果格式字符串中包含非法语法、与给定参数不兼容的格式说明符，格式字符串给定的参数不够，或存在其他非法条件。有关所有可能的格式化错误的规范，请参阅 formatter 类规范的详细信息 一节。

NullPointerException - 如果 format 为 null

从以下版本开始：

1.5

另请参见：

Formatter

---

valueOf

```
public static String valueOf(Object obj)
```

返回 Object 参数的字符串表示形式。

参数：

obj - 一个 Object。

返回：

如果参数为 null，则字符串等于 "null"；否则，返回 obj.toString() 的值。

另请参见：

Object.toString()

---

valueOf

```
public static String valueOf(char[] data)
```

返回 char 数组参数的字符串表示形式。字符数组的内容已被复制，后续修改不会影响新创建的字符串。

参数：

data - char 数组。

返回：
一个新分配的字符串，它表示包含在字符数组参数中的相同字符序列。

valueOf

```
public static String valueOf(char[] data,
                             int offset,
                             int count)
```

返回 char 数组参数的特定子数组的字符串表示形式。

offset 参数是子数组的第一个字符的索引。count 参数指定子数组的长度。字符数组的内容已被复制，后续修改不会影响新创建的字符串。

参数：
data - 字符数组。
offset - String 值的初始偏移量。
count - String 值的长度。
返回：
一个字符串，它表示在字符数组参数的子数组中包含的字符序列。
抛出：
IndexOutOfBoundsException - 如果 offset 为负，count 为负，或者 offset+count 大于 data.length。

copyValueOf

```
public static String copyValueOf(char[] data,
                                 int offset,
                                 int count)
```

返回指定数组中表示该字符序列的 String。
参数：
data - 字符数组。
offset - 子数组的初始偏移量。
count - 子数组的长度。
返回：
一个 String，它包含字符数组的指定子数组的字符。

copyValueOf

```
public static String copyValueOf(char[] data)
```

返回指定数组中表示该字符序列的 String。

参数：

data - 字符数组。

返回：

一个 String，它包含字符数组的字符。

---

valueOf

```
public static String valueOf(boolean b)
```

返回 boolean 参数的字符串表示形式。

参数：

b - 一个 boolean。

返回：

如果参数为 true，则返回一个等于 "true" 的字符串；否则，返回一个等于 "false" 的字符串。

---

valueOf

```
public static String valueOf(char c)
```

返回 char 参数的字符串表示形式。

参数：

c - 一个 char。

返回：

一个长度为 1 的字符串，它包含参数 c 的单个字符。

---

valueOf

```
public static String valueOf(int i)
```

返回 int 参数的字符串表示形式。

该表示形式恰好是单参数的 Integer.toString 方法返回的结果。

参数：
i - 一个 int。
返回：
int 参数的字符串表示形式。
另请参见：
Integer.toString(int, int)

---

valueOf

```
public static String valueOf(long l)
```

返回 long 参数的字符串表示形式。

该表示形式恰好是单参数的 Long.toString 方法返回的结果。

参数：
l - 一个 long。
返回：
long 参数的字符串表示形式。
另请参见：
Long.toString(long)

---

valueOf

```
public static String valueOf(float f)
```

返回 float 参数的字符串表示形式。

该表示形式恰好是单参数的 Float.toString 方法返回的结果。

参数：
f - 一个 float。
返回：
float 参数的字符串表示形式。
另请参见：
Float.toString(float)

---

valueOf

public static String valueOf(double d)

返回 double 参数的字符串表示形式。

该表示形式恰好是单参数的 Double.toString 方法返回的结果。

参数：
d - 一个 double。
返回：
double 参数的字符串表示形式。
另请参见：
Double.toString(double)

intern

public String intern()

返回字符串对象的规范化表示形式。

一个初始为空的字符串池，它由类 String 私有地维护。

当调用 intern 方法时，如果池已经包含一个等于此 String 对象的字符串（用 equals(Object) 方法确定），则返回池中的字符串。否则，将此 String 对象添加到池中，并返回此 String 对象的引用。

它遵循以下规则：对于任意两个字符串 s 和 t，当且仅当 s.equals(t) 为 true 时，s.intern() == t.intern() 才为 true。

所有字面值字符串和字符串赋值常量表达式都使用 intern 方法进行操作。字符串字面值在 Java Language Specification 的 §3.10.5 定义。

返回：
一个字符串，内容与此字符串相同，但一定取自具有唯一字符串的池。

System 类

System 类包含一些有用的类字段和方法。它不能被实例化。

在 System 类提供的设施中，有标准输入、标准输出和错误输出流；对外部定义的属性和环境变量的访问；加载文件和库的方法；还有快速复制数组的一部分的实用方法。

字段详细信息

in

```
public static final InputStream in
```

“标准”输入流。此流已打开并准备提供输入数据。通常，此流对应于键盘输入或者由主机环境或用户指定的另一个输入源。

---

out

```
public static final PrintStream out
```

“标准”输出流。此流已打开并准备接受输出数据。通常，此流对应于显示器输出或者由主机环境或用户指定的另一个输出目标。

对于简单独立的 Java 应用程序，编写一行输出数据的典型方式是：

```
System.out.println(data)
```

请参阅 PrintStream 类中的 println 方法。

另请参见：

PrintStream.println(), PrintStream.println(boolean),
PrintStream.println(char), PrintStream.println(char[]),
PrintStream.println(double), PrintStream.println(float),
PrintStream.println(int), PrintStream.println(long),
PrintStream.println(java.lang.Object),
PrintStream.println(java.lang.String)

---

err

```
public static final PrintStream err
```

“标准”错误输出流。此流已打开并准备接受输出数据。

通常，此流对应于显示器输出或者由主机环境或用户指定的另一个输出目标。按照惯例，此输出流用于显示错误消息，或者显示那些即使用户输出流（变量 out 的值）已经重定向到通常不被连续监视的某一文件或其他目标，也应该立刻引起用户注意的其他信息。

方法详细信息

setIn

```
public static void setIn(InputStream in)
```

重新分配“标准”输入流。

首先，如果有安全管理器，则通过 RuntimePermission("setIO") 权限调用其 checkPermission 方法，查看是否可以重新分配“标准”输入流。

参数：
in - 新的标准输出流。
抛出：
SecurityException - 如果安全管理器存在并且其 checkPermission 方法不允许重新分配标准输入流。
从以下版本开始：
JDK1.1
另请参见：
SecurityManager.checkPermission(java.security.Permission), RuntimePermission

---

setOut

```
public static void setOut(PrintStream out)
```

重新分配“标准”输出流。

首先，如果有安全管理器，则通过 RuntimePermission("setIO") 权限调用其 checkPermission 方法，查看是否可以重新分配“标准”输出流。

参数：
out - 新的标准输出流

抛出：
SecurityException - 如果安全管理器存在并且其 checkPermission 方法不允许重新分配标准输出流。
从以下版本开始：
JDK1.1
另请参见：
SecurityManager.checkPermission(java.security.Permission), RuntimePermission

---

setErr

```
public static void setErr(PrintStream err)
```

重新分配“标准”错误输出流。

首先，如果有安全管理器，则通过 RuntimePermission("setIO") 权限调用其 checkPermission 方法，查看是否可以重新分配“标准”输出流。

参数：
err - 新的标准错误输出流
抛出：
SecurityException - 如果安全管理器存在并且其 checkPermission 方法不允许重新分配标准错误输出流。
从以下版本开始：
JDK1.1
另请参见：
SecurityManager.checkPermission(java.security.Permission), RuntimePermission

---

console

```
public static Console console()
```

返回与当前 Java 虚拟机关联的唯一 Console 对象（如果有）。
返回：
系统控制台（如果有），否则返回 null。
从以下版本开始：
1.6

---

inheritedChannel

```
public static Channel inheritedChannel()
                                    throws IOException
```

返回从创建此 Java 虚拟机的实体中继承的信道。

此方法返回通过调用系统级默认 SelectorProvider 对象的 inheritedChannel 方法获得的信道。

除了 inheritedChannel 中描述的面向网络的信道之外，此方法以后还可能返回其他种类的信道。

返回：
继承的信道（如果有），否则返回 null。
抛出：
IOException - 如果发生 I/O 错误
SecurityException - 如果安全管理器存在并且它不允许访问信道。
从以下版本开始：
1.5

---

setSecurityManager

```
public static void setSecurityManager(SecurityManager s)
```

设置系统安全性。

如果已经安装了安全管理器，则此方法首先通过 RuntimePermission("setSecurityManager") 权限调用安全管理器的 checkPermission 方法，以确保可以替换现有的安全管理器。这可能导致抛出一个 SecurityException 异常。

否则，将该参数作为当前安全管理器建立。如果参数为 null 并且没有建立安全管理器，则不执行任何操作，并且该方法将自行返回。

参数：
s - 安全管理器。
抛出：
SecurityException - 如果安全管理器已经设置并且其 checkPermission 方法不允许替换该设置。

另请参见：
getSecurityManager(),
SecurityManager.checkPermission(java.security.Permission),
RuntimePermission

---

getSecurityManager

```
public static SecurityManager getSecurityManager()
```

获取系统安全接口。
返回：
如果已经为当前应用程序建立了安全管理器，则返回此安全管理器；否则，返回 null。
另请参见：
setSecurityManager(java.lang.SecurityManager)

---

currentTimeMillis

```
public static long currentTimeMillis()
```

返回以毫秒为单位的当前时间。注意，当返回值的时间单位是毫秒时，值的粒度取决于底层操作系统，并且粒度可能更大。例如，许多操作系统以几十毫秒为单位测量时间。

请参阅 Date 类的描述，了解可能发生在“计算机时间”和协调世界时（UTC）之间的细微差异的讨论。

返回：
当前时间与协调世界时 1970 年 1 月 1 日午夜之间的时间差（以毫秒为单位测量）。
另请参见：
Date

---

nanoTime

```
public static long nanoTime()
```

返回最准确的可用系统计时器的当前值，以毫微秒为单位。

此方法只能用于测量已过的时间，与系统或钟表时间的其他任何时间概念无关。返回值表示从某一固定但任意的时间算起的毫微秒数（或许从以后算起，所以该值可能为负）。此方法提供毫微秒的精度，但不是必要的毫微秒的准确度。它对于值的更改频率没有作出保证。在取值范围大于约 292 年（2^{63} 毫微秒）的连续调用的不同点在于：由于数字溢出，将无法准确计算已过的时间。

例如，测试某些代码执行的时间长度：

```
long startTime = System.nanoTime();
// ... the code being measured ...
long estimatedTime = System.nanoTime() - startTime;
```

返回：
系统计时器的当前值，以毫微秒为单位。
从以下版本开始：
1.5

arraycopy

```
public static void arraycopy(Object src,
                             int srcPos,
                             Object dest,
                             int destPos,
                             int length)
```

从指定源数组中复制一个数组，复制从指定的位置开始，到目标数组的指定位置结束。从 src 引用的源数组到 dest 引用的目标数组，数组组件的一个子序列被复制下来。被复制的组件的编号等于 length 参数。源数组中位置在 srcPos 到 srcPos+length-1 之间的组件被分别复制到目标数组中的 destPos 到 destPos+length-1 位置。

如果参数 src 和 dest 引用相同的数组对象，则复制的执行过程就好像首先将 srcPos 到 srcPos+length-1 位置的组件复制到一个带有 length 组件的临时数组，然后再将此临时数组的内容复制到目标数组的 destPos 到 destPos+length-1 位置一样。

If 如果 dest 为 null，则抛出 NullPointerException 异常。

如果 src 为 null，则抛出 NullPointerException 异常，并且不会修改目标数组。

否则，只要下列任何情况为真，则抛出 ArrayStoreException 异常并且不会修改目标数组：

- src 参数指的是非数组对象。
- dest 参数指的是非数组对象。
- src 参数和 dest 参数指的是那些其组件类型为不同基本类型的数组。
- src 参数指的是具有基本组件类型的数组且 dest 参数指的是具有引用组件类型的数组。
- src 参数指的是具有引用组件类型的数组且 dest 参数指的是具有基本组件类型的数组。

否则，只要下列任何情况为真，则抛出 IndexOutOfBoundsException 异常，并且不会修改目标数组：

- srcPos 参数为负。
- destPos 参数为负。
- length 参数为负。
- srcPos+length 大于 src.length，即源数组的长度。
- destPos+length 大于 dest.length，即目标数组的长度。

否则，如果源数组中 srcPos 到 srcPos+length-1 位置上的实际组件通过分配转换并不能转换成目标数组的组件类型，则抛出 ArrayStoreException 异常。在这种情况下，将 ***k*** 设置为比长度小的最小非负整数，这样就无法将 src[srcPos+*k*] 转换为目标数组的组件类型；当抛出异常时，从 srcPos 到 srcPos+*k*-1 位置上的源数组组件已经被复制到目标数组中的 destPos 到 destPos+*k*-1 位置，而目标数组中的其他位置不会被修改。（因为已经详细说明过的那些限制，只能将此段落有效地应用于两个数组都有引用类型的组件类型的情况。）

参数：
src - 源数组。
srcPos - 源数组中的起始位置。
dest - 目标数组。
destPos - 目标数据中的起始位置。
length - 要复制的数组元素的数量。

抛出：
IndexOutOfBoundsException - 如果复制会导致对数组范围以外的数据的访问。
ArrayStoreException - 如果因为类型不匹配而使得无法将 src 数组中的元素存储到 dest 数组中。
NullPointerException - 如果 src 或 dest 为 null。

identityHashCode

```
public static int identityHashCode(Object x)
```

返回给定对象的哈希码，该代码与默认的方法 hashCode() 返回的代码一样，无论给定对象的类是否重写 hashCode()。null 引用的哈希码为 0。
参数：
x - 要计算其哈希码的对象
返回：
哈希码
从以下版本开始：
JDK1.1

getProperties

```
public static Properties getProperties()
```

确定当前的系统属性。

首先，如果有安全管理器，则不带参数直接调用其 checkPropertiesAccess 方法。这可能导致一个安全性异常。

将 getProperty(String) 方法使用的当前系统属性集合作为 Properties 对象返回。如果没有当前系统属性集合，则先创建并初始化一个系统属性集合。这个系统属性集合总是包含以下键的值：

键	相关值的描述
java.version	Java 运行时环境版本
java.vendor	Java 运行时环境供应商
java.vendor.url	Java 供应商的 URL

java.home	Java 安装目录
java.vm.specification.version	Java 虚拟机规范版本
java.vm.specification.vendor	Java 虚拟机规范供应商
java.vm.specification.name	Java 虚拟机规范名称
java.vm.version	Java 虚拟机实现版本
java.vm.vendor	Java 虚拟机实现供应商
java.vm.name	Java 虚拟机实现名称
java.specification.version	Java 运行时环境规范版本
java.specification.vendor	Java 运行时环境规范供应商
java.specification.name	Java 运行时环境规范名称
java.class.version	Java 类格式版本号
java.class.path	Java 类路径
java.library.path	加载库时搜索的路径列表
java.io.tmpdir	默认的临时文件路径
java.compiler	要使用的 JIT 编译器的名称
java.ext.dirs	一个或多个扩展目录的路径
os.name	操作系统的名称
os.arch	操作系统的架构
os.version	操作系统的版本
file.separator	文件分隔符（在 UNIX 系统中是“/”）
path.separator	路径分隔符（在 UNIX 系统中是“:”）
line.separator	行分隔符（在 UNIX 系统中是“/n”）
user.name	用户的账户名称
user.home	用户的主目录
user.dir	用户的当前工作目录

系统属性值中的多个路径是用平台的路径分隔符分隔的。

注意，即使安全管理器不允许执行 getProperties 操作，它可能也会选择允许执行 getProperty(String) 操作。

返回：
系统属性
抛出：

SecurityException - 如果安全管理器存在并且其
checkPropertiesAccess 方法不允许访问系统属性。
另请参见：
setProperties(java.util.Properties), SecurityException,
SecurityManager.checkPropertiesAccess(), Properties

setProperties

public static void setProperties(Properties props)

将系统属性设置为 Properties 参数。

首先，如果有安全管理器，则不带参数直接调用其
checkPropertiesAccess 方法。这可能导致一个安全性异常。

参数是 getProperty(String) 方法使用的当前系统属性的集合。如果参数为 null，则忽略当前系统属性的集合。

参数：
props - 新的系统属性。
抛出：
SecurityException - 如果安全管理器存在并且其
checkPropertiesAccess 方法不允许访问系统属性。
另请参见：
getProperties(), Properties, SecurityException,
SecurityManager.checkPropertiesAccess()

getProperty

public static String getProperty(String key)

获取指定键指示的系统属性。

首先，如果有安全管理器，则用该键作为其参数来调用
checkPropertyAccess 方法。结果可能导致 SecurityException。

如果没有当前系统属性的集合，则首先用与 getProperties 方法相同的方式创建并初始化系统属性的集合。

参数：
key - 系统属性的名称。
返回：
系统属性的字符串值，如果没有带有此键的属性，则返回 null。
抛出：
SecurityException - 如果安全管理器存在并且其 checkPropertyAccess 方法不允许访问指定的系统属性。
NullPointerException - 如果 key 为 null。
IllegalArgumentException - 如果 key 为空。
另请参见：
setProperty(java.lang.String, java.lang.String),
SecurityException,
SecurityManager.checkPropertyAccess(java.lang.String),
getProperties()

getProperty

```
public static String getProperty(String key,
                                 String def)
```

获取用指定键描述的系统属性。

首先，如果有安全管理器，则用该 key 作为参数调用 checkPropertyAccess 方法。

如果没有当前系统属性的集合，将用与 getProperties 方法相同的方式首先创建并初始化系统属性的集合。

参数：
key - 系统属性的名称。
def - 默认值。
返回：
系统属性的字符串值，如果没有带有此键的属性，则返回默认值。
抛出：
SecurityException - 如果安全管理器存在并且其 checkPropertyAccess 方法不允许访问指定的系统属性。
NullPointerException - 如果 key 为 null。
IllegalArgumentException - 如果 key 为空。
另请参见：

setProperty(java.lang.String, java.lang.String),
SecurityManager.checkPropertyAccess(java.lang.String),
getProperties()

setProperty

```
public static String setProperty(String key,
                                 String value)
```

设置指定键指示的系统属性。

首先，如果安全管理器存在，则通过 PropertyPermission(key, "write") 权限调用其 SecurityManager.checkPermission 方法。这可能导致抛出 SecurityException。如果没有抛出异常，则将指定属性设置为给定值。

参数：
key - 系统属性的名称。
value - 系统属性的值。
返回：
系统属性以前的值，如果没有以前的值，则返回 null。
抛出：
SecurityException - 如果安全管理器存在并且其 checkPermission 方法不允许设置指定属性。
NullPointerException - 如果 key 或 value 为 null。
IllegalArgumentException - 如果 key 为空。
从以下版本开始：
1.2
另请参见：
getProperty(java.lang.String), getProperty(java.lang.String),
getProperty(java.lang.String, java.lang.String),
PropertyPermission,
SecurityManager.checkPermission(java.security.Permission)

clearProperty

```
public static String clearProperty(String key)
```

移除指定键指示的系统属性。

首先，如果安全管理器存在，则通过 PropertyPermission(key, "write") 权限调用其 SecurityManager.checkPermission 方法。这可能导致抛出 SecurityException。如果没有抛出异常，则移除指定的属性。

参数：
key - 要移除的系统属性的名称。
返回：
系统属性以前的字符串值，如果带有此键的属性不存在，则返回 null。
抛出：
SecurityException - 如果安全管理器存在并且其 checkPropertyAccess 方法不允许访问指定的系统属性。
NullPointerException - 如果 key 为 null。
IllegalArgumentException - 如果 key 为空。
从以下版本开始：
1.5
另请参见：
getProperty(java.lang.String), setProperty(java.lang.String, java.lang.String), Properties, SecurityException, SecurityManager.checkPropertiesAccess()

getenv

```
public static String getenv(String name)
```

获取指定的环境变量值。环境变量是一个取决于系统的外部指定的值。

如果安全管理器存在，则通过 RuntimePermission("getenv."+name) 的权限调用其 checkPermission 方法。这可能导致抛出 SecurityException。如果没有抛出异常，则返回变量 name 的值。

从概念上讲，*系统属性* 和*环境变量* 都是名称与值之间的映射。两种机制都能用来将用户定义的信息传递给 Java 进程。环境变量产生更多的全局效应，因为它们不仅对紧接着出现的 Java 子进程可见，而且对于定义它们的进程的所有子进程都是可见的。在不同的操作系统上，它们的语义有细微的差别，比如，不区分大小写。因为这些原因，环境变量更可能有意料不到的副作用。最好在可能的地方使用系统属性。环境变量应该在需要全局效应的时候使用，或者在外部系统接口要求使用环境变量时使用（比如 PATH）。

在 UNIX 系统中，name 的字母大小写通常很重要，而在 Microsoft Windows 系统中，这通常不重要。例如，表达式 System.getenv("FOO").equals(System.getenv("foo")) 在 Microsoft Windows 中可能为真。

参数：
name - 环境变量的名称
返回：
变量的字符串值，如果变量不是在系统环境中定义的，则返回 null
抛出：
NullPointerException - 如果 name 为 null
SecurityException - 如果安全管理器存在并且其 checkPermission 方法不允许访问环境变量 name
另请参见：
getenv(), ProcessBuilder.environment()

---

getenv

```
public static Map<String,String> getenv()
```

返回一个不能修改的当前系统环境的字符串映射视图。该环境是一个取决于系统的从名称到值的映射，它从父进程传递给子进程。

如果系统不支持环境变量，则返回一个空映射。

返回的映射永远不会包含 null 键或 Null 值。如果试图查询 null 键或 Null 值的存在，则会抛出 NullPointerException。如果试图查询不是 String 类型的键或值的存在，则会抛出 ClassCastException。

返回的映射及其集合视图可能没有遵守 Object.equals(java.lang.Object) 和 Object.hashCode() 方法的通用协定。

在所有的平台上，返回的映射通常都是区分大小写的。

如果安全管理器存在，则通过 RuntimePermission("getenv.*") 权限调用 checkPermission 方法。这可能导致抛出 SecurityException。

将信息传递给 Java 子进程时，系统属性一般优先于环境变量。

返回：
作为变量名称到值的映射的环境
抛出：
SecurityException - 如果安全管理器存在并且其 checkPermission 方法不允许访问进程环境
从以下版本开始：
1.5
另请参见：
getenv(String), ProcessBuilder.environment()

---

exit

```
public static void exit(int status)
```

终止当前正在运行的 Java 虚拟机。参数用作状态码；根据惯例，非 0 的状态码表示异常终止。

该方法调用 Runtime 类中的 exit 方法。该方法永远不会正常返回。

调用 System.exit(n) 实际上等效于调用：

```
Runtime.getRuntime().exit(n)
```

参数：
status - 退出状态。
抛出：
SecurityException - 如果安全管理器存在并且其 checkExit 方法不允许以指定状态退出。
另请参见：
Runtime.exit(int)

---

gc

```
public static void gc()
```

运行垃圾回收器。

调用 gc 方法暗示着 Java 虚拟机做了一些努力来回收未用对象，以便能够快速地重用这些对象当前占用的内存。当控制权从方法调用中返回时，虚拟机已经尽最大努力从所有丢弃的对象中回收了空间。

调用 System.gc() 实际上等效于调用：

```
Runtime.getRuntime().gc()
```

另请参见：
Runtime.gc()

runFinalization

```
public static void runFinalization()
```

运行处于挂起终止状态的所有对象的终止方法。

调用该方法说明 Java 虚拟机做了一些努力运行已被丢弃对象的 finalize 方法，但是这些对象的 finalize 方法至今尚未运行。当控制权从方法调用中返回时，Java 虚拟机已经尽最大努力去完成所有未执行的终止方法。

调用 System.runFinalization() 实际上等效于调用：

```
Runtime.getRuntime().runFinalization()
```

另请参见：
Runtime.runFinalization()

runFinalizersOnExit

```
@Deprecated
public static void runFinalizersOnExit(boolean value)
```

已过时。 *该方法具有固有的不安全性。它可能对正在使用的对象调用终结方法，而其他线程同时正在操作这些对象，从而导致不正确的行为或死锁。*

在退出时启用或禁用终结；这样做可指定，拥有未被自动调用终结方法的所有对象的终结方法在退出 Java 运行库前运行。默认情况下，禁用退出时终结。

如果有安全管理器，则首先使用 0 作为参数来调用其 checkExit 方法，以确保允许退出。这可能导致抛出 SecurityException。

参数：
value - 指示启用或禁用终止操作的值
抛出：
SecurityException - 如果安全管理器存在并且其 checkExit 方法不允许退出。
从以下版本开始：
JDK1.1
另请参见：
Runtime.exit(int), Runtime.gc(), SecurityManager.checkExit(int)

load

```
public static void load(String filename)
```

从作为动态库的本地文件系统中以指定的文件名加载代码文件。文件名参数必须是完整的路径名。

调用 System.load(name) 实际上等效于调用：

```
Runtime.getRuntime().load(name)
```

参数：
filename - 要加载的文件。
抛出：
SecurityException - 如果安全管理器存在并且其 checkLink 方法不允许加载指定的动态库。
UnsatisfiedLinkError - 如果文件不存在。
NullPointerException - 如果 filename 为 null
另请参见：
Runtime.load(java.lang.String),
SecurityManager.checkLink(java.lang.String)

loadLibrary

```
public static void loadLibrary(String libname)
```

加载由 libname 参数指定的系统库。将库名映射到实际系统库的方法取决于系统。

调用 System.loadLibrary(name) 实际上等效于调用：

```
Runtime.getRuntime().loadLibrary(name)
```

参数：
libname - 库名。
抛出：
SecurityException - 如果安全管理器存在并且其 checkLink 方法不允许加载指定的动态库。
UnsatisfiedLinkError - 如果库不存在。
NullPointerException - 如果 libname 为 null
另请参见：
Runtime.loadLibrary(java.lang.String), SecurityException, SecurityManager.checkLink(java.lang.String)

---

mapLibraryName

```
public static String mapLibraryName(String libname)
```

将一个库名称映射到特定于平台的、表示本机库的字符串中。
参数：
libname - 库名。
返回：
取决于平台的本机库名称。
抛出：
NullPointerException - 如果 libname 为 null。

java.io

通过数据流、序列化和文件系统提供系统输入和输出。

接口

Closeable 接口

Closeable 是可以关闭的数据源或目标。调用 close 方法可释放对象保存的资源（如打开文件）。

方法详细信息

close

```
void close()
           throws IOException
```

关闭此流并释放与此流关联的所有系统资源。如果已经关闭该流，则调用此方法无效。

抛出：

IOException - 如果发生 I/O 错误

DataInput 接口

DataInput 接口用于从二进制流中读取字节，并根据所有 Java 基本类型数据进行重构。同时还提供根据 UTF-8 修改版格式的数据重构 String 的工具。

对于此接口中的所有数据读取例程来说，如果在读取所需字节数之前已经到达文件末尾 (end of file)，则将抛出 EOFException（IOException 的一种）。如果因为到达文件末尾以外的其他原因无法读取字节，则将抛出 IOException 而不是 EOFException。尤其是，在输入流已关闭的情况下，将抛出 IOException。

方法详细信息

readFully

```
void readFully(byte[] b)
               throws IOException
```

从输入流中读取一些字节，并将它们存储在缓冲区数组 b 中。读取的字节数等于 b 的长度。

在出现以下条件之一以前，此方法将一直阻塞：

- 输入数据的 b.length 个字节是可用的，在这种情况下，正常返回。
- 检测到文件末尾，在这种情况下，抛出 EOFException。
- 发生 I/O 错误，在这种情况下，将抛出 IOException，而不是 EOFException。

如果 b 为 null，则抛出 NullPointerException。如果 b.length 为零，则不读取字节。否则，将读取的第一个字节存储到元素 b[0] 中，下一个字节存储到 b[1] 中，依此类推。如果此方法抛出异常，则可能是因为已经用输入流中的数据更新了 b 的某些（但非全部）字节。

参数：

b - 存储读取数据的缓冲区。

抛出：

EOFException - 如果此流在读取所有字节之前到达末尾。

IOException - 如果发生 I/O 错误。

---

readFully

```
void readFully(byte[] b,
               int off,
               int len)
               throws IOException
```

从输入流中读取 len 个字节。

在出现以下条件之一以前，此方法将一直阻塞：

- 输入数据的 len 个字节是可用的，在这种情况下，正常返回。
- 检测到文件末尾，在这种情况下，抛出 EOFException。
- 如果发生 I/O 错误，在这种情况下，将抛出 IOException，而不是 EOFException。

如果 b 为 null，则抛出 NullPointerException。如果 off 为负，或 len 为负，或者 off+len 大于数组 b 的长度，则抛出 IndexOutOfBoundsException。如果 len 为零，则不读取字节。否则，将读取的第一个字节存储到元素 b[off] 中，下一个字节存储到 b[off+1] 中，依此类推。读取的字节数至多等于 b[0]。

参数：

b - 存储读取数据的缓冲区。

off - 指定数据中的偏移量的 int 值。

len - 指定读取的字节数的 int 值。

抛出：

EOFException - 如果此流在读取所有字节之前到达末尾。

IOException - 如果发生 I/O 错误。

skipBytes

```
int skipBytes(int n)
              throws IOException
```

试图在输入流中跳过数据的 n 个字节，并丢弃跳过的字节。不过，可以跳过更少的字节数，该字节数甚至可以为零。这可能由很多情况引起；在已经跳过 n 个字节前到达文件末尾只是其中的一种可能。此方法从不抛出 EOFException。返回实际跳过的字节数。

参数：

n - 要跳过的字节数。

返回：

实际跳过的字节数。

抛出：

IOException - 如果发生 I/O 错误。

readBoolean

```
boolean readBoolean()
                    throws IOException
```

读取一个输入字节，如果该字节不是零，则返回 true，如果是零，则返回 false。此方法适用于读取用接口 DataOutput 的 writeBoolean 方法写入的字节。

返回：

读取的 boolean 值。

抛出：

EOFException - 如果此流在读取所有字节之前到达末尾。

IOException - 如果发生 I/O 错误。

readByte

```
byte readByte()
              throws IOException
```

读取并返回一个输入字节。该字节被看作是 -128 到 127（包含）范围内的一个有符号值。此方法适用于读取用接口 DataOutput 的 writeByte 方法写入的字节。

返回：

读取的 8 位值。

抛出：

EOFException - 如果此流在读取所有字节之前到达末尾。

IOException - 如果发生 I/O 错误。

readUnsignedByte

```
int readUnsignedByte()
```

```
throws IOException
```

读取一个输入字节，将它左侧补零 (zero-extend) 转变为 int 类型，并返回结果，所以结果的范围是 0 到 255。如果接口 DataOutput 的 writeByte 方法的参数是 0 到 255 之间的值，则此方法适用于读取用 writeByte 写入的字节。

返回：

读取的无符号 8 位值。

抛出：

EOFException - 如果此流在读取所有字节之前到达末尾。

IOException - 如果发生 I/O 错误。

readShort

```
short readShort()
                throws IOException
```

读取两个输入字节并返回一个 short 值。设 a 为第一个读取字节，b 为第二个读取字节。返回的值是：

```
(short)((a << 8) | (b & 0xff))
```

此方法适用于读取用接口 DataOutput 的 writeShort 方法写入的字节。

返回：

读取的 16 位值。

抛出：

EOFException - 如果此流在读取所有字节之前到达末尾。

IOException - 如果发生 I/O 错误。

readUnsignedShort

```
int readUnsignedShort()
                throws IOException
```

读取两个输入字节，并返回 0 到 65535 范围内的一个 int 值。设 a 为第一个读取字节，b 为第二个读取字节。返回的值是：

```
(((a & 0xff) << 8) | (b & 0xff))
```

如果接口 DataOutput 的 writeShort 方法的参数是 0 到 65535 范围内的值，

则此方法适用于读取用 writeShort 写入的字节。

返回：

读取的无符号 16 位值。

抛出：

EOFException - 如果此流在读取所有字节之前到达末尾。

IOException - 如果发生 I/O 错误。

---

readChar

```
char readChar()
              throws IOException
```

读取两个输入字节并返回一个 char 值。设 a 为第一个读取字节，b 为第二个读取字节。返回的值是：

(char)((a << 8) | (b & 0xff))

此方法适用于读取用接口 DataOutput 的 writeChar 方法写入的字节。

返回：

读取的 char 值。

抛出：

EOFException - 如果此流在读取所有字节之前到达末尾。

IOException - 如果发生 I/O 错误。

---

readInt

```
int readInt()
            throws IOException
```

读取四个输入字节并返回一个 int 值。设 a-d 为四个读取字节中的第一个字节。返回的值是：

```
(((a & 0xff) << 24) | ((b & 0xff) << 16) |
 ((c & 0xff) << 8) | (d & 0xff))
```

此方法适用于读取用接口 DataOutput 的 writeInt 方法写入的字节。

返回：

读取的 int 值。

抛出：

EOFException - 如果此流在读取所有字节之前到达末尾。

IOException - 如果发生 I/O 错误。

---

readLong

```
long readLong()
              throws IOException
```

读取八个输入字节并返回一个 long 值。设 a-h 为八个读取字节中的第一个字节。返回的值是：

```
(((long)(a & 0xff) << 56) |
 ((long)(b & 0xff) << 48) |
 ((long)(c & 0xff) << 40) |
 ((long)(d & 0xff) << 32) |
 ((long)(e & 0xff) << 24) |
 ((long)(f & 0xff) << 16) |
 ((long)(g & 0xff) <<  8) |
 ((long)(h & 0xff)))
```

此方法适用于读取用接口 DataOutput 的 writeLong 方法写入的字节。

返回：

读取的 long 值。

抛出：

EOFException - 如果此流在读取所有字节之前到达末尾。

IOException - 如果发生 I/O 错误。

---

readFloat

```
float readFloat()
               throws IOException
```

读取四个输入字节并返回一个 float 值。实现这一点的方法是:先使用与 readInt 方法完全相同的方式构造一个 int 值，然后使用与 Float.intBitsToFloat 方法完全相同的方式将此 int 值转换成一个 float 值。此方法适用于读取用接口 DataOutput 的 writeFloat 方法写入的字节。

返回：

读取的 float 值。

抛出：

EOFException - 如果此流在读取所有字节之前到达末尾。

IOException - 如果发生 I/O 错误。

---

readDouble

```
double readDouble()
                  throws IOException
```

读取八个输入字节并返回一个 double 值。实现这一点的方法是：先使用与 readlong 方法完全相同的方式构造一个 long 值，然后使用与 Double.longBitsToDouble 方法完全相同的方式将此 long 值转换成一个 double 值。此方法适用于读取用接口 DataOutput 的 writeDouble 方法写入的字节。

返回：

读取的 double 值。

抛出：

EOFException - 如果此流在读取所有字节之前到达末尾。

IOException - 如果发生 I/O 错误。

---

readLine

```
String readLine()
                  throws IOException
```

从输入流中读取下一文本行。该方法读取连续的字节，将每个字节分别转换成一个字符，直到遇到行结尾符或到达末尾；然后以 String 形式返回读取的字符。注意，因为此方法用于处理字符，所以它不支持整个 Unicode 字符集的输入。

如果在一个字节都没有读取的时候就到达文件末尾，则返回 null。否则，通过左侧补零将读取的每个字节转换成 char 类型的值。如果遇到字符 '\n'，则丢弃它并且停止读取。如果遇到字符 '\r' 则丢弃它，如果后续字节转变成字符 '\n'，则同样丢弃它并停止读取。如果在遇到字符 '\n' 和 '\r' 之一前到达文件末尾，则停止读取。一旦已停止读取，则返回一个 String，它按顺序包含所有已读取且未丢弃的字符。注意，此字符串中的每个字符的值都将小于 \u0100（即 (char)256）的值。

返回：

输入流中文本的下一行，如果还没有读取一个字节就到达文件末尾，则返回 null。

抛出：

IOException - 如果发生 I/O 错误。

readUTF

```
String readUTF()
                throws IOException
```

读入一个已使用 UTF-8 修改版格式编码的字符串。readUTF 的常规协定是：该方法读取使用 UTF-8 修改版格式编码的 Unicode 字符串的表示形式；然后以 String 的形式返回此字符串。

首先读取两个字节，并使用它们构造一个无符号 16 位整数，构造方式与 readUnsignedShort 方法的方式完全相同。该整数值被称为 *UTF 长度*，它指定要读取的额外字节数。然后成组地将这些字节转换为字符。每组的长度根据该组第一个字节的值计算。紧跟在某个组后面的字节（如果有）是下一组的第一个字节。

如果组的第一个字节与位模式 0xxxxxxx（其中 x 表示“可能为 0 或 1”）匹配，则该组只有这一个字节。该字节被左侧补零，转换成一个字符。

如果组的第一个字节与位模式 110xxxxx 匹配，则该组只由字节 a 和另一个字节 b 组成。如果没有字节 b（因为字节 a 是要读取的最后一个字节），或者字节 b 与位模式 10xxxxxx 不匹配，则抛出 UTFDataFormatException。否则，将该组转换成字符：

```
(char)(((a& 0x1F) << 6) | (b & 0x3F))
```

如果组的第一个字节与位模式 1110xxxx 匹配，则该组由字节 a 和另外两个字节 b 和 c 组成。如果没有字节 c（因为字节 a 是要读取的最后两个字节之一），或者字节 b 或字节 c 与位模式 10xxxxxx 不匹配，则抛出 UTFDataFormatException。否则，将该组转换成字符：

```
(char)(((a & 0x0F) << 12) | ((b & 0x3F) << 6) | (c & 0x3F))
```

如果组的第一个字节与模式 1111xxxx 或模式 10xxxxxx 匹配，则抛出 UTFDataFormatException。

如果在执行整个过程中的任意时间到达文件末尾，则抛出 EOFException。

在通过此过程将每个组转换成字符后，按照从输入流中读取相应组的顺序，将这些字符收集在一起，形成一个 String，然后该字符串将被返回。

可以使用 DataOutput 接口的 writeUTF 方法写入适合此方法读取的数据。

返回：
一个 Unicode 字符串。
抛出：
EOFException - 如果此流在读取所有字节之前到达末尾。
IOException - 如果发生 I/O 错误。
UTFDataFormatException - 如果这些字节不表示一个有效的、UTF-8 修改版编码的字符串。

DataOutput 接口

DataOutput 接口用于将数据从任意 Java 基本类型转换为一系列字节，并将这些字节写入二进制流。同时还提供了一个将 String 转换成 UTF-8 修改版格式并写入所得到的系列字节的工具。

对于此接口中写入字节的所有方法，如果由于某种原因无法写入某个字节，则抛出 IOException。

方法详细信息

write

```
void write(int b)
           throws IOException
```

将参数 b 的八个低位写入输出流。忽略 b 的 24 个高位。

参数：

b - 要写入的字节。

抛出：

IOException - 如果发生 I/O 错误。

---

write

```
void write(byte[] b)
           throws IOException
```

将数组 b 中的所有字节写入输出流。如果 b 为 null，则抛出 NullPointerException。如果 b.length 为零，则不写入字节。否则，首先写入字节 b[0],然后写入字节 b[1],依此类推;最后一个写入字节是 b[b.length-1]。

参数：

b - 数据。

抛出：

IOException - 如果发生 I/O 错误。

---

write

```
void write(byte[] b,
           int off,
           int len)
           throws IOException
```

将数组 b 中的 len 个字节按顺序写入输出流。如果 b 为 null，则抛出 NullPointerException。如果 off 为负，或 len 为负，抑或 off+len 大于数

组 b 的长度，则抛出 IndexOutOfBoundsException。如果 len 为零，则不写入字节。否则，首先写入字节 b[off]，然后写入字节 b[off+1]，依此类推；最后一个写入字节是 b[off+len-1]。

参数：

b - 数据。

off - 数据中的起始偏移量。

len - 要写入的字节数。

抛出：

IOException - 如果发生 I/O 错误。

writeBoolean

```
void writeBoolean(boolean v)
                  throws IOException
```

将一个 boolean 值写入输出流。如果参数 v 为 true，则写入值 (byte)1；如果 v 为 false，则写入值 (byte)0。此方法写入的字节可由 DataInput 接口的 readBoolean 方法读取，然后该方法将返回一个等于 v 的 boolean 值。

参数：

v - 要写入的 boolean 值。

抛出：

IOException - 如果发生 I/O 错误。

writeByte

```
void writeByte(int v)
               throws IOException
```

将参数 v 的八个低位写入输出流。忽略 v 的 24 个高位。(这意味着 writeByte 的作用与使用整数做参数的 write 完全相同。)此方法写入的字节可由 DataInput 接口的 readByte 方法读取，然后该方法将返回一个等于 (byte)v 的 byte 值。

参数：

v - 要写入的字节值。

抛出：

IOException - 如果发生 I/O 错误。

writeShort

```
void writeShort(int v)
                throws IOException
```

将两个字节写入输出流，用它们表示参数值。要写入的字节值（按顺序显示）是：

```
(byte)(0xff & (v >> 8))
(byte)(0xff & v)
```

此方法写入的字节可由 DataInput 接口的 readShort 方法读取，然后该方法将返回一个等于 (short)v 的 short 值。

参数：

v - 要写入的 short 值。

抛出：

IOException - 如果发生 I/O 错误。

writeChar

```
void writeChar(int v)
                throws IOException
```

将一个 char 值写入输出流，该值由两个字节组成。要写入的字节值（按顺序显示）是：

```
(byte)(0xff & (v >> 8))
(byte)(0xff & v)
```

此方法写入的字节可由 DataInput 接口的 readChar 方法读取，然后该方法将返回一个等于 (char)v 的 char 值。

参数：

v - 要写入的 char 值。

抛出：

IOException - 如果发生 I/O 错误。

writeInt

```
void writeInt(int v)
                throws IOException
```

将一个 int 值写入输出流，该值由四个字节组成。要写入的字节值（按顺序显示）是：

```
(byte)(0xff & (v >> 24))
(byte)(0xff & (v >> 16))
(byte)(0xff & (v >>  8))
(byte)(0xff & v)
```

此方法写入的字节可由 DataInput 接口的 readInt 方法读取，然后该方法将返回一个等于 v 的 int 值。

参数：

v - 要写入的 int 值。

抛出：

IOException - 如果发生 I/O 错误。

writeLong

```
void writeLong(long v)
                throws IOException
```

将一个 long 值写入输出流，该值由八个字节组成。要写入的字节值（按顺序显示）是：

```
(byte)(0xff & (v >> 56))
(byte)(0xff & (v >> 48))
(byte)(0xff & (v >> 40))
(byte)(0xff & (v >> 32))
(byte)(0xff & (v >> 24))
(byte)(0xff & (v >> 16))
(byte)(0xff & (v >>  8))
```

```
(byte)(0xff & v)
```

此方法写入的字节可由 DataInput 接口的 readLong 方法读取，然后该方法将返回一个等于 v 的 long 值。

参数：

v - 要写入的 long 值。

抛出：

IOException - 如果发生 I/O 错误。

writeFloat

```
void writeFloat(float v)
                throws IOException
```

将一个 float 值写入输出流，该值由四个字节组成。实现这一点的方式是：首先使用与 Float.floatToIntBits 方法完全相同的方式将此 float 值转换为一个 int 值，然后使用与 writeInt 方法完全相同的方式写入该 int 值。此方法写入的字节可由 DataInput 接口的 readFloat 方法读取，然后该方法将返回一个等于 v 的 float 值。

参数：

v - 要写入的 float 值。

抛出：

IOException - 如果发生 I/O 错误。

writeDouble

```
void writeDouble(double v)
                 throws IOException
```

将一个 double 值写入输出流，该值由八个字节组成。实现这一点的方式是：首先使用与 Double.doubleToLongBits 方法相同的方式将此 double 值转换为一个 long 值，然后使用与 writeLong 方法完全相同的方式写入该 long 值。此方法写入的字节可由 DataInput 接口的 readDouble 方法读取，然后该方法将返回一个等于 v 的 double 值。

参数：

v - 要写入的 double 值。

抛出：

IOException - 如果发生 I/O 错误。

writeBytes

```
void writeBytes(String s)
                throws IOException
```

将一个字符串写入输出流。字符串 s 中的每一个字符被依次写入输出流，每个字符用一个字节表示。如果 s 为 null，则抛出 NullPointerException。

如果 s.length 为零，则不写入字节。否则，首先写入字符 s[0]，然后写入字符 s[1]，依此类推；最后一个写入字符是 s[s.length-1]。使用与 writeByte 方法完全相同的方法为每个字符写入一个低位字节。忽略字符串中每个字符的八个高位。

参数：

s - 要写入的字节字符串。

抛出：

IOException - 如果发生 I/O 错误。

writeChars

```
void writeChars(String s)
                throws IOException
```

将字符串 s 中的所有字符按顺序写入输出流，每个字符用两个字节表示。如果 s 为 null，则抛出 NullPointerException。如果 s.length 为零，则不写入字符。否则，首先写入字符 s[0]，然后写入字符 s[1]，依此类推；最后一个写入字符是 s[s.length-1]。使用与 writeChar 方法完全相同的方法为每个字符实际写入两个字节，先使用高位字节。

参数：

s - 要写入的字符串值。

抛出：

IOException - 如果发生 I/O 错误。

writeUTF

```
void writeUTF(String s)
                throws IOException
```

将表示长度信息的两个字节写入输出流，后跟字符串 s 中每个字符的 UTF-8 修改版表示形式。如果 s 为 null，则抛出 NullPointerException。根据字符的值，将字符串 s 中每个字符转换成一个字节、两个字节或三个字节的字节组。

如果字符 c 在 \u0001 到 \u007f 的范围内，则用一个字节表示：

```
(byte)c
```

如果字符 c 是 \u0000 或者它在 \u0080 到 \u07ff 的范围内，则用两个字节表示，写入顺序如下：

```
(byte)(0xc0 | (0x1f & (c >> 6)))
(byte)(0x80 | (0x3f & c))
```

如果字符 c 在 \u0800 到 \uffff 的范围内，则用三个字节表示，写入顺序如下：

```
(byte)(0xe0 | (0x0f & (c >> 12)))
(byte)(0x80 | (0x3f & (c >>  6)))
(byte)(0x80 | (0x3f & c))
```

首先，要计算表示 s 中所有字符所需的字节总数。如果总数大于 65535，则抛出 UTFDataFormatException。否则，使用与 writeShort 方法完全相同的方式将此长度写入输出流，然后写入字符串 s 中所有字符的 1 字节、2 字节或 3 字节表示形式。

此方法写入的字节可由 DataInput 接口的 readUTF 方法读取，然后该方法将返回一个等于 s 的 String 值。

参数：

s - 要写入的字符串值。

抛出：

IOException - 如果发生 I/O 错误。

Externalizable 接口

Externalizable 实例类的唯一特性是可以被写入序列化流中，该类负责保存和恢复实例内容。 若某个要完全控制某一对象及其超类型的流格式和内容，则它要实现 Externalizable 接口的 writeExternal 和 readExternal 方法。这些方法必须显式与超类型进行协调以保存其状态。这些方法将代替定制的 writeObject 和 readObject 方法实现。

Serialization 对象将使用 Serializable 和 Externalizable 接口。对象持久性机制也可以使用它们。要存储的每个对象都需要检测是否支持 Externalizable 接口。如果对象支持 Externalizable，则调用 writeExternal 方法。如果对象不支持 Externalizable 但实现了 Serializable，则使用 ObjectOutputStream 保存该对象。

在重构 Externalizable 对象时，先使用无参数的公共构造方法创建一个实例，然后调用 readExternal 方法。通过从 ObjectInputStream 中读取 Serializable 对象可以恢复这些对象。

Externalizable 实例可以通过 Serializable 接口中记录的 writeReplace 和 readResolve 方法来指派一个替代对象。

方法详细信息

writeExternal

```
void writeExternal(ObjectOutput out)
                   throws IOException
```

该对象可实现 writeExternal 方法来保存其内容，它可以通过调用 DataOutput 的方法来保存其基本值，或调用 ObjectOutput 的 writeObject 方法来保存对象、字符串和数组。

参数：

out - 要写入对象的流

抛出：

IOException - 包含可能发生的所有 I/O 异常

readExternal

```
void readExternal(ObjectInput in)
                  throws IOException,
                         ClassNotFoundException
```

对象实现 readExternal 方法来恢复其内容，它通过调用 DataInput 的方法来恢复其基础类型，调用 readObject 来恢复对象、字符串和数组。readExternal 方法必须按照与 writeExternal 方法写入值时使用的相同顺序和类型来读取这些值。

参数：

in - 为了恢复对象而从中读取数据的流

抛出：

IOException - 如果发生 I/O 错误

ClassNotFoundException - 如果无法找到需要恢复的某个对象的类。

FileFilter 接口

用于抽象路径名的过滤器。

此接口的实例可传递给 File 类的 listFiles(FileFilter) 方法。

方法详细信息

accept

```
boolean accept(File pathname)
```

测试指定抽象路径名是否应该包含在某个路径名列表中。

参数：

pathname - 要测试的抽象路径名

返回：

当且仅当应该包含 pathname 时返回 true

FilenameFilter 接口

实现此接口的类实例可用于过滤器文件名。Abstract Window Toolkit 的文件对话框组件使用这些实例过滤 File 类的 list 方法中的目录清单。

方法详细信息

accept

```
boolean accept(File dir,
               String name)
```

测试指定文件是否应该包含在某一文件列表中。

参数：

dir - 被找到的文件所在的目录。

name - 文件的名称。

返回：

当且仅当该名称应该包含在文件列表中时返回 true；否则返回 false。

Flushable 接口

Flushable 是可刷新数据的目标地。调用 flush 方法将所有已缓冲输出写入底层流。

方法详细信息

flush

```
void flush()
           throws IOException
```

通过将所有已缓冲输出写入底层流来刷新此流。

抛出：

IOException - 如果发生 I/O 错误

ObjectInput 接口

ObjectInput 扩展 DataInput 接口以包含对象的读操作。DataInput 包括基本类型的输入方法；ObjectInput 扩展了该接口，以包含对象、数组和 String 的输出方法。

方法详细信息

readObject

```
Object readObject()
                    throws ClassNotFoundException,
                           IOException
```

读取并返回对象。实现此接口的类定义从哪里“读取”对象。

返回：

从流读取的对象

抛出：

ClassNotFoundException - 如果无法找到已序列化对象的类。

IOException - 如果发生任何常规 Input/Output 相关的异常。

---

read

```
int read()
          throws IOException
```

读取数据字节。如果不存在可用的输入，此方法将发生阻塞。

返回：

读取的字节；如果已到达流的末尾，则返回 -1。

抛出：

IOException - 如果发生 I/O 错误。

---

read

```
int read(byte[] b)
        throws IOException
```

读入 byte 数组。在某些输入可用之前，此方法将发生阻塞。

参数：

b - 将数据读入的缓冲区

返回：

读取的实际字节数；当到达流的末尾时，返回 -1。

抛出：

IOException - 如果发生 I/O 错误。

read

```
int read(byte[] b,
        int off,
        int len)
        throws IOException
```

读入 byte 数组。在某些输入可用之前，此方法将发生阻塞。

参数：

b - 将数据读入的缓冲区

off - 数据的初始偏移量

len - 读取的最大字节数

返回：

读取的实际字节数；当到达流的末尾时返回 -1。

抛出：

IOException - 如果发生 I/O 错误。

skip

```
long skip(long n)
        throws IOException
```

跳过输入的 n 个字节。

参数：

n - 要跳过的字节数

返回：

跳过的实际字节数。

抛出：

IOException - 如果发生 I/O 错误。

available

```
int available()
              throws IOException
```

返回可以无阻塞地读取的字节数。

返回：

可用字节数。

抛出：

IOException - 如果发生 I/O 错误。

close

```
void close()
           throws IOException
```

关闭输入流。必须调用此方法以释放与流相关的所有资源。

抛出：

IOException - 如果发生 I/O 错误。

ObjectInputValidation 接口

允许验证图形中对象的回调接口。允许在反序列化完整的对象图形后调用对象。

方法详细信息

validateObject

```
void validateObject()
                    throws InvalidObjectException
```

验证对象。

抛出：

InvalidObjectException - 如果对象无法自我验证。

ObjectOutput 接口

ObjectOutput 扩展 DataOutput 接口以包含对象的写入操作。DataOutput 包括基本类型的输出方法；ObjectOutput 扩展了该接口，以包含对象、数组和 String 的输出方法。

方法详细信息

writeObject

```
void writeObject(Object obj)
                    throws IOException
```

将对象写入底层存储或流。实现此接口的类定义如何写入对象。

参数：

obj - 要写入的对象

抛出：

IOException - 如果发生任何与常规 Input/Output 相关的异常。

---

write

```
void write(int b)
                    throws IOException
```

写入字节。在实际写入字节前，此方法将阻塞。

指定者：

接口 DataOutput 中的 write

参数：

b - 字节

抛出：
IOException - 如果发生 I/O 错误

write

```
void write(byte[] b)
           throws IOException
```

写入 byte 数组。在实际写入字节前将发生阻塞。
指定者：
接口 DataOutput 中的 write
参数：
b - 要写入的数据
抛出：
IOException - 如果发生 I/O 错误。

write

```
void write(byte[] b,
           int off,
           int len)
           throws IOException
```

写入字节的子数组。
指定者：
接口 DataOutput 中的 write
参数：
b - 要写入的数据
off - 数据中的初始偏移量
len - 写入的字节数
抛出：
IOException - 如果发生 I/O 错误。

flush

```
void flush()
                throws IOException
```

刷新该流的缓冲。此操作将写入所有已缓冲的输出字节。

抛出：

IOException - 如果发生 I/O 错误。

close

```
void close()
                throws IOException
```

关闭该流。必须调用此方法以释放与此流相关的所有资源。

抛出：

IOException - 如果发生 I/O 错误。

ObjectStreamConstants 接口

写入 Object Serialization Stream 的常量。

方法详细信息

STREAM_MAGIC

```
static final short STREAM_MAGIC
```

写入流头的幻数。

另请参见：

常量字段值

STREAM_VERSION

```
static final short STREAM_VERSION
```

写入流头的版本号。

另请参见：

常量字段值

TC_BASE

static final byte TC_BASE

第一个标记值。

另请参见：

常量字段值

TC_NULL

static final byte TC_NULL

Null 对象引用。

另请参见：

常量字段值

TC_REFERENCE

static final byte TC_REFERENCE

已写入流的对象的引用。

另请参见：

常量字段值

TC_CLASSDESC

static final byte TC_CLASSDESC

新的 Class Descriptor。

另请参见：

常量字段值

TC_OBJECT

static final byte TC_OBJECT

新的 Object。

另请参见：

常量字段值

TC_STRING

static final byte TC_STRING

新的 String。

另请参见：

常量字段值

TC_ARRAY

static final byte TC_ARRAY

新的 Array。

另请参见：

常量字段值

TC_CLASS

static final byte TC_CLASS

Class 的引用。

另请参见：

常量字段值

TC_BLOCKDATA

static final byte TC_BLOCKDATA

可选数据块。跟在标记后面的字节指示此块数据中的字节数。

另请参见：

常量字段值

TC_ENDBLOCKDATA

static final byte TC_ENDBLOCKDATA

可选块数据的末尾因对象而阻塞。

另请参见：

常量字段值

TC_RESET

static final byte TC_RESET

重置流上下文。重置写入流的所有句柄。

另请参见：

常量字段值

TC_BLOCKDATALONG

static final byte TC_BLOCKDATALONG

long Block 数据。跟在标记后面的 long 指示此块数据中的字节数。

另请参见：

常量字段值

TC_EXCEPTION

static final byte TC_EXCEPTION

写入过程中的异常。

另请参见：

常量字段值

TC_LONGSTRING

static final byte TC_LONGSTRING

Long 字符串。

另请参见：

常量字段值

TC_PROXYCLASSDESC

static final byte TC_PROXYCLASSDESC

新的 Proxy Class Descriptor。

另请参见：

常量字段值

TC_ENUM

static final byte TC_ENUM

新的 Enum 常量。

从以下版本开始：

1.5

另请参见：

常量字段值

TC_MAX

static final byte TC_MAX

最后一个标记值。

另请参见：

常量字段值

baseWireHandle

static final int baseWireHandle

分配的第一个句柄。

另请参见：

常量字段值

SC_WRITE_METHOD

static final byte SC_WRITE_METHOD

ObjectStreamClass 标志的位掩码。指示 Serializable 类，该类定义自身的 writeObject 方法。

另请参见：

常量字段值

SC_BLOCK_DATA

static final byte SC_BLOCK_DATA

ObjectStreamClass 标志的位掩码。指示以 Block Data 模式写入的 Externalizable 数据。为 PROTOCOL_VERSION_2 添加。

从以下版本开始：

1.2

另请参见：

PROTOCOL_VERSION_2,常量字段值

SC_SERIALIZABLE

static final byte SC_SERIALIZABLE

ObjectStreamClass 标志的位掩码。指示类为 Serializable。

另请参见：

常量字段值

SC_EXTERNALIZABLE

static final byte SC_EXTERNALIZABLE

ObjectStreamClass 标志的位掩码。指示类为 Externalizable。

另请参见：

常量字段值

SC_ENUM

static final byte SC_ENUM

ObjectStreamClass 标志的位掩码。指示类为 enum 类型。

从以下版本开始：

1.5

另请参见：

常量字段值

SUBSTITUTION_PERMISSION

static final SerializablePermission SUBSTITUTION_PERMISSION

允许在序列化/反序列化期间使用一个对象取代另一个对象。

从以下版本开始：

1.2

另请参见：

ObjectOutputStream.enableReplaceObject(boolean),
ObjectInputStream.enableResolveObject(boolean)

SUBCLASS_IMPLEMENTATION_PERMISSION

static final SerializablePermission SUBCLASS_IMPLEMENTATION_PERMISSION

允许重写 readObject 和 writeObject。

从以下版本开始：

1.2

另请参见：

ObjectOutputStream.writeObjectOverride(Object), ObjectInputStream.readObjectOverride()

PROTOCOL_VERSION_1

static final int PROTOCOL_VERSION_1

流协议的版本。

调用此方法后，所有可扩展的数据都将使用 JDK 1.1 外部数据格式写入。当流中包含 JDK 1.1.6 JVM 之前的版本可读取的 Externalizable 数据时，此版本为写入流所必需的。

从以下版本开始：

1.2

另请参见：

ObjectOutputStream.useProtocolVersion(int),常量字段值

PROTOCOL_VERSION_2

static final int PROTOCOL_VERSION_2

流协议的版本。

此协议由 JVM 1.2 写入。 Externalizable 数据以块数据模式写入，使用 TC_ENDBLOCKDATA 终止。Externalizable 类描述符标志已启用 SC_BLOCK_DATA。JVM 1.1.6 及更高版本可以读取此格式的更改。 允许将

nonSerializable 类描述符写入流。将 nonSerializable 类的 serialVersionUID 设置为 0L。

Serializable 接口

类通过实现 java.io.Serializable 接口以启用其序列化功能。未实现此接口的类将无法使其任何状态序列化或反序列化。可序列化类的所有子类型本身都是可序列化的。序列化接口没有方法或字段，仅用于标识可序列化的语义。

要允许不可序列化类的子类型序列化，可以假定该子类型负责保存和恢复超类型的公用 (public)、受保护的 (protected) 和（如果可访问）包 (package) 字段的状态。仅在子类型扩展的类有一个可访问的无参数构造方法来初始化该类的状态时，才可以假定子类型有此职责。如果不是这种情况，则声明一个类为可序列化类是错误的。该错误将在运行时检测到。

在反序列化过程中，将使用该类的公用或受保护的无参数构造方法初始化不可序列化类的字段。可序列化的子类必须能够访问无参数构造方法。可序列化子类的字段将从该流中恢复。

当遍历一个图形时，可能会遇到不支持 Serializable 接口的对象。在此情况下，将抛出 NotSerializableException，并将标识不可序列化对象的类。

在序列化和反序列化过程中需要特殊处理的类必须使用下列准确签名来实现特殊方法：

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException
private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;
private void readObjectNoData()
    throws ObjectStreamException;
```

writeObject 方法负责写入特定类的对象的状态，以便相应的 readObject 方法可以恢复它。通过调用 out.defaultWriteObject 可以调用保存 Object 的字段的默认机制。该方法本身不需要涉及属于其超类或子类的状态。通过使用 writeObject 方法或使用 DataOutput 支持的用于基本数据类型的方法将各个字段写入 ObjectOutputStream，状态可以被保存。

readObject 方法负责从流中读取并恢复类字段。它可以调用 in.defaultReadObject 来调用默认机制，以恢复对象的非静态和非瞬态字段。defaultReadObject 方法使用流中的信息来分配流中通过当前对象中相应指定字段保存的对象的字段。这用于处理类演化后需要添加新字段的情形。该方法本身不需要涉及属于其超类或子类的状态。通过使用 writeObject 方法或使用 DataOutput 支持的用于基本数据类型的方法将各个字段写入 ObjectOutputStream，状态可以被保存。

在序列化流不列出给定类作为将被反序列化对象的超类的情况下，readObjectNoData 方

法负责初始化特定类的对象状态。这在接收方使用的反序列化实例类的版本不同于发送方，并且接收者版本扩展的类不是发送者版本扩展的类时发生。在序列化流已经被篡改时也将发生；因此，不管源流是“敌意的”还是不完整的，readObjectNoData 方法都可以用来正确地初始化反序列化的对象。

将对象写入流时需要指定要使用的替代对象的可序列化类，应使用准确的签名来实现此特殊方法：

```
ANY-ACCESS-MODIFIER Object writeReplace() throws ObjectStreamException;
```

此 writeReplace 方法将由序列化调用，前提是如果此方法存在，而且它可以通过被序列化对象的类中定义的一个方法访问。因此，该方法可以拥有私有 (private)、受保护的 (protected) 和包私有 (package-private) 访问。子类对此方法的访问遵循 java 访问规则。

在从流中读取类的一个实例时需要指定替代的类应使用的准确签名来实现此特殊方法。

```
ANY-ACCESS-MODIFIER Object readResolve() throws ObjectStreamException;
```

此 readResolve 方法遵循与 writeReplace 相同的调用规则和访问规则。

序列化运行时使用一个称为 serialVersionUID 的版本号与每个可序列化类相关联，该序列号在反序列化过程中用于验证序列化对象的发送者和接收者是否为该对象加载了与序列化兼容的类。如果接收者加载的该对象的类的 serialVersionUID 与对应的发送者的类的版本号不同，则反序列化将会导致 InvalidClassException。可序列化类可以通过声明名为 "serialVersionUID" 的字段（该字段必须是静态 (static)、最终 (final) 的 long 型字段）显式声明其自己的 serialVersionUID：

```
ANY-ACCESS-MODIFIER static final long serialVersionUID = 42L;
```

如果可序列化类未显式声明 serialVersionUID，则序列化运行时将基于该类的各个方面计算该类的默认 serialVersionUID 值，如“Java(TM) 对象序列化规范”中所述。不过，*强烈建议* 所有可序列化类都显式声明 serialVersionUID 值，原因是计算默认的 serialVersionUID 对类的详细信息具有较高的敏感性，根据编译器实现的不同可能千差万别，这样在反序列化过程中可能会导致意外的 InvalidClassException。因此，为保证 serialVersionUID 值跨不同 java 编译器实现的一致性，序列化类必须声明一个明确的 serialVersionUID 值。还强烈建议使用 private 修饰符显示声明 serialVersionUID（如果可能），原因是这种声明仅应用于直接声明类 -- serialVersionUID 字段作为继承成员没有用处。数组类不能声明一个明确的 serialVersionUID，因此它们总是具有默认的计算值，但是数组类没有匹配 serialVersionUID 值的要求。

类

BufferedInputStream 类

BufferedInputStream 为另一个输入流添加一些功能，即缓冲输入以及支持 mark 和 reset 方法的能力。在创建 BufferedInputStream 时，会创建一个内部缓冲区数组。在读取或跳过流中的字节时，可根据需要从包含的输入流再次填充该内部缓冲区，一次填充多个字节。mark 操作记录输入流中的某个点，reset 操作使得在从包含的输入流中获取新字节之前，再次读取自最后一次 mark 操作后读取的所有字节。

字段详细信息

buf

protected volatile byte[] buf

存储数据的内部缓冲区数组。必要时可用另一个不同大小的数组替换它。

count

protected int count

比缓冲区中最后一个有效字节的索引大 1 的索引。此值始终处于 0 到 buf.length 的范围内；从 buf[0] 到 buf[count-1] 的元素包含从底层输入流中获取的缓冲输入数据。

pos

protected int pos

缓冲区中的当前位置。这是将从 buf 数组中读取的下一个字符的索引。

此值始终处于 0 到 count 的范围内。如果此值小于 count，则 buf[pos] 将作为下一个输入字节；如果此值等于 count，则下一次 read 或 skip 操作需要从包含的输入流中读取更多的字节。

另请参见：
buf

markpos

protected int markpos

最后一次调用 mark 方法时 pos 字段的值。

此值始终处于 -1 到 pos 的范围内。如果输入流中没有被标记的位置，则此字段为 -1。如果输入流中有被标记的位置，则 buf[markpos] 将用作 reset 操作后的第一个输入字节。如果 markpos 不是 -1，则从位置 buf[markpos] 到 buf[pos-1] 之间的所有字节都必须保留在缓冲区数组中（尽管对 count、pos 和 markpos 的值进行适当调整后，这些字节可能移动到缓冲区数组中的其他位置）；除非 pos 与 markpos 的差超过 marklimit，否则不能将其丢弃。

另请参见：
mark(int),pos

marklimit

protected int marklimit

调用 mark 方法后，在后续调用 reset 方法失败之前所允许的最大提前读取量。只要 pos 与 markpos 之差超过 marklimit,就可以通过将 markpos 设置为 -1 来删除该标记。

另请参见：
mark(int),reset()

构造方法详细信息

BufferedInputStream

```
public BufferedInputStream(InputStream in)
```

创建一个 BufferedInputStream 并保存其参数，即输入流 in，以便将来使用。创建一个内部缓冲区数组并将其存储在 buf 中。

参数：

in - 底层输入流。

BufferedInputStream

```
public BufferedInputStream(InputStream in,
                           int size)
```

创建具有指定缓冲区大小的 BufferedInputStream 并保存其参数,即输入流 in，以便将来使用。创建一个长度为 size 的内部缓冲区数组并将其存储在 buf 中。

参数：

in - 底层输入流。

size - 缓冲区大小。

抛出：

IllegalArgumentException - 如果 size <= 0

方法详细信息

read

```
public int read()
         throws IOException
```

参见 InputStream 的 read 方法的常规协定。

覆盖：

类 FilterInputStream 中的 read

返回：

下一个数据字节，如果到达流末尾，则返回 -1。

抛出：

IOException - 如果已经调用其 close() 方法关闭了此输入流，或者发生 I/O 错误。

另请参见：

FilterInputStream.in

---

read

```
public int read(byte[] b,
                int off,
                int len)
         throws IOException
```

从此字节输入流中给定偏移量处开始将各字节读取到指定的 byte 数组中。

此方法实现了 InputStream 类相应 read 方法的常规协定。另一个便捷之处在于，它将通过重复地调用底层流的 read 方法，尝试读取尽可能多的字节。这种迭代的 read 会一直继续下去，直到满足以下条件之一：

- 已经读取了指定的字节数，
- 底层流的 read 方法返回 -1，指示文件末尾（end-of-file），或者
- 底层流的 available 方法返回 0，指示将阻塞后续的输入请求。

如果第一次对底层流调用 read 返回 -1（指示文件末尾），则此方法返回 -1。否则此方法返回实际读取的字节数。

鼓励（但不是必须）此类的各个子类以相同的方式尝试读取尽可能多的字节。

覆盖：

类 FilterInputStream 中的 read

参数：

b - 目标缓冲区。

off - 开始存储字节处的偏移量。

len - 要读取的最大字节数。

返回：

读取的字节数；如果已到达流末尾，则返回 -1。

抛出：

IOException - 如果已经调用其 close() 方法关闭了此输入流，或者发生 I/O 错误。

另请参见：

FilterInputStream.in

---

skip

```
public long skip(long n)
          throws IOException
```

参见 InputStream 的 skip 方法的常规协定。

覆盖：

类 FilterInputStream 中的 skip

参数：

n - 要跳过的字节数。

返回：

跳过的实际字节数。

抛出：

IOException - 如果流不支持查找操作；或者已经调用其 close() 方法关闭了此输入流；或者发生 I/O 错误。

---

available

```
public int available()
          throws IOException
```

返回可以从此输入流读取（或跳过）、且不受此输入流接下来的方法调用阻塞的估计字节数。接下来的调用可能是同一个线程，也可能是不同的线程。一次读取或跳过这么多字节将不会受阻塞，但可以读取或跳过数量更少的字节。

此方法返回缓冲区中剩余的待读取字节数（count - pos）与调用 in.available() 的结果之和。

覆盖：

类 FilterInputStream 中的 available

返回：

可以不受阻塞地从此输入流读取（或跳过）的估计字节数。

抛出：

IOException - 如果已经调用其 close() 方法关闭了此输入流，或者发生 I/O 错误。

mark

```
public void mark(int readlimit)
```

参见 InputStream 的 mark 方法的常规协定。

覆盖：

类 FilterInputStream 中的 mark

参数：

readlimit - 在标记位置变为无效之前可以读取字节的最大限制。

另请参见：

reset()

reset

```
public void reset()
                throws IOException
```

参见 InputStream 的 reset 方法的常规协定。

如果 markpos 为 -1（尚未设置标记，或者标记已失效），则抛出 IOException。否则将 pos 设置为与 markpos 相等。

覆盖：

类 FilterInputStream 中的 reset

抛出：

IOException - 如果尚未标记此流；或者标记已失效；或者已经调用其 close() 方法关闭了此输入流；或者发生 I/O 错误。

另请参见：

mark(int)

markSupported

public boolean markSupported()

测试此输入流是否支持 mark 和 reset 方法。BufferedInputStream 的 markSupported 方法返回 true。

覆盖：

类 FilterInputStream 中的 markSupported

返回：

一个 boolean 值，指示此流类型是否支持 mark 和 reset 方法。

另请参见：

InputStream.mark(int),InputStream.reset()

close

public void close()
throws IOException

关闭此输入流并释放与该流关联的所有系统资源。关闭了该流之后，后续的 read()、available()、reset() 或 skip() 调用都将抛出 IOException。关闭之前已关闭的流不会产生任何效果。

指定者：

接口 Closeable 中的 close

覆盖：

类 FilterInputStream 中的 close

抛出：

IOException - 如果发生 I/O 错误。

BufferedOutputStream 类

该类实现缓冲的输出流。通过设置这种输出流，应用程序就可以将各个字节写入底层输出流中，而不必针对每次字节写入调用底层系统。

字段详细信息

buf

protected byte[] buf

存储数据的内部缓冲区。

count

protected int count

缓冲区中的有效字节数。此值始终处于 0 到 buf.length 范围内；元素 buf[0] 到 buf[count-1] 包含有效的字节数据。

构造方法详细信息

BufferedOutputStream

public BufferedOutputStream(OutputStream out)

创建一个新的缓冲输出流，以将数据写入指定的底层输出流。

参数：

out - 底层输出流。

BufferedOutputStream

public BufferedOutputStream(OutputStream out,
int size)

创建一个新的缓冲输出流，以将具有指定缓冲区大小的数据写入指定的底层输出流。

参数：

out - 底层输出流。

size - 缓冲区的大小。

抛出：

IllegalArgumentException - 如果 size <= 0

方法详细信息

write

```
public void write(int b)
           throws IOException
```

将指定的字节写入此缓冲的输出流。

覆盖：

类 FilterOutputStream 中的 write

参数：

b - 要写入的字节。

抛出：

IOException - 如果发生 I/O 错误。

---

write

```
public void write(byte[] b,
                  int off,
                  int len)
           throws IOException
```

将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此缓冲的输出流。

一般来说，此方法将给定数组的字节存入此流的缓冲区中，根据需要将该缓冲区刷新，并转到底层输出流。但是，如果请求的长度至少与此流的缓冲区大小相同，则此方法将刷新该缓冲区并将各个字节直接写入底层输出流。因此多余的 BufferedOutputStream 将不必复制数据。

覆盖：

类 FilterOutputStream 中的 write

参数：

b - 数据。

off - 数据的起始偏移量。

len - 要写入的字节数。

抛出：

IOException - 如果发生 I/O 错误。

另请参见：

FilterOutputStream.write(int)

flush

```
public void flush()
            throws IOException
```

刷新此缓冲的输出流。这迫使所有缓冲的输出字节被写出到底层输出流中。

指定者：

接口 Flushable 中的 flush

覆盖：

类 FilterOutputStream 中的 flush

抛出：

IOException - 如果发生 I/O 错误。

ByteArrayInputStream 类

ByteArrayInputStream 包含一个内部缓冲区，该缓冲区包含从流中读取的字节。内部计数器跟踪 read 方法要提供的下一个字节。

关闭 ByteArrayInputStream 无效。此类中的方法在关闭此流后仍可被调用，而不会产生任何 IOException。

字段详细信息

buf

```
protected byte[] buf
```

由该流的创建者提供的 byte 数组。元素 buf[0] 到 buf[count-1] 是只能从流中读取的字节；元素 buf[pos] 是要读取的下一个字节。

pos

protected int pos

要从输入流缓冲区中读取的下一个字符的索引。此值应该始终是非负数，并且不应大于 count 值。从输入流缓冲区中读取的下一个字节是 buf[pos]。

mark

protected int mark

流中当前的标记位置。构造时默认将 ByteArrayInputStream 对象标记在位置零处。通过 mark() 方法可将其标记在缓冲区内的另一个位置处。通过 reset() 方法将当前缓冲区位置设置为此点。

如果尚未设置标记，则标记值是传递给构造方法的偏移量（如果未提供偏移量，则标记值为 0）。

从以下版本开始：
JDK1.1

count

protected int count

比输入流缓冲区中最后一个有效字符的索引大一的索引。此值应该始终是非负数，并且不应大于 buf 的长度。它比 buf 中最后一个可从输入流缓冲区中读取的字节位置大一。

构造方法详细信息

ByteArrayInputStream

public ByteArrayInputStream(byte[] buf)

创建一个 ByteArrayInputStream，使用 buf 作为其缓冲区数组。该缓冲区数组不是复制得到的。pos 的初始值是 0，count 的初始值是 buf 的长度。

参数：

buf - 输入缓冲区。

ByteArrayInputStream

```
public ByteArrayInputStream(byte[] buf,
                            int offset,
                            int length)
```

创建 ByteArrayInputStream，使用 buf 作为其缓冲区数组。pos 的初始值是 offset，count 的初始值是 offset+length 和 buf.length 中的最小值。该缓冲区数组不是复制得到的。将该缓冲区的标记设置为指定的偏移量。

参数：

buf - 输入缓冲区。

offset - 缓冲区中要读取的第一个字节的偏移量。

length - 从缓冲区中读取的最大字节数。

方法详细信息

read

```
public int read()
```

从此输入流中读取下一个数据字节。返回一个 0 到 255 范围内的 int 字节值。如果因为到达流末尾而没有可用的字节，则返回值 -1。

此 read 方法不会阻塞。

指定者：

类 InputStream 中的 read

返回：

下一个数据字节，如果到达流末尾，则返回 -1。

read

```
public int read(byte[] b,
                int off,
                int len)
```

将最多 len 个数据字节从此输入流读入 byte 数组。如果 pos 等于 count，则返回 -1 指示文件结束。否则，读取的字节数 k 等于 len 和 count-pos 中的较小者。如果 k 是正数，则以 System.arraycopy 执行的方式将 buf[pos] 到 buf[pos+k-1] 的字节复制到 b[off] 到 b[off+k-1] 中。将值 k 与 pos 相加并返回 k。

此 read 方法不会阻塞。

覆盖：

类 InputStream 中的 read

参数：

b - 存储读入数据的缓冲区。

off - 目标数组 b 的起始偏移量。

len - 读取的最大字节数。

返回：

读入缓冲区的总字节数，如果由于已到达流末尾而不再有数据，则返回 -1。

抛出：

NullPointerException - 如果 b 为 null。

IndexOutOfBoundsException - 如果 off 为负，len 为负，或者 len 大于 b.length - off

另请参见：

InputStream.read()

skip

```
public long skip(long n)
```

从此输入流中跳过 n 个输入字节。如果已到达输入流末尾，则可能会跳过较少的字节。实际跳过的字节数 k 等于 n 和 count-pos 中的较小者。将值 k 与 pos 相加并返回 k。

覆盖：

类 InputStream 中的 skip

参数：

n - 要跳过的字节数。

返回：

跳过的实际字节数。

available

```
public int available()
```

返回可从此输入流读取（或跳过）的剩余字节数。

返回值是 count - pos，它是要从输入缓冲区中读取的剩余字节数。

覆盖：

类 InputStream 中的 available

返回：

不受阻塞地从此输入流读取（或跳过）的剩余字节数。

markSupported

```
public boolean markSupported()
```

测试此 InputStream 是否支持 mark/reset。ByteArrayInputStream 的 markSupported 方法始终返回 true。

覆盖：

类 InputStream 中的 markSupported

返回：

如果此输入流实例支持 mark 和 reset 方法，则返回 true；否则返回 false。

从以下版本开始：

JDK1.1

另请参见：

InputStream.mark(int), InputStream.reset()

mark

public void mark(int readAheadLimit)

设置流中的当前标记位置。构造时默认将 ByteArrayInputStream 对象标记在位置零处。通过此方法可将其标记在缓冲区内的另一个位置处。

如果尚未设置标记，则标记值是传递给构造方法的偏移量（如果未提供偏移量，则标记值为 0）。

注：readAheadLimit 对于此类没有意义。

覆盖：

类 InputStream 中的 mark

参数：

readAheadLimit - 在标记位置失效前可以读取字节的最大限制。

从以下版本开始：

JDK1.1

另请参见：

InputStream.reset()

---

reset

public void reset()

将缓冲区的位置重置为标记位置。除非已标记了另一个位置，或者在构造方法中指定了一个偏移量，否则该标记位置是 0。

覆盖：

类 InputStream 中的 reset

另请参见：

InputStream.mark(int),IOException

---

close

public void close()
throws IOException

关闭 ByteArrayInputStream 无效。此类中的方法在关闭此流后仍可被调用，而不会产生任何 IOException。

指定者：
接口 Closeable 中的 close
覆盖：
类 InputStream 中的 close
抛出：
IOException - 如果发生 I/O 错误。

ByteArrayOutputStream 类

此类实现了一个输出流，其中的数据被写入一个 byte 数组。缓冲区会随着数据的不断写入而自动增长。可使用 toByteArray() 和 toString() 获取数据。

关闭 ByteArrayOutputStream 无效。此类中的方法在关闭此流后仍可被调用，而不会产生任何 IOException。

字段详细信息

buf

```
protected byte[] buf
```

存储数据的缓冲区。

---

count

```
protected int count
```

缓冲区中的有效字节数。

构造方法详细信息

ByteArrayOutputStream

```
public ByteArrayOutputStream()
```

创建一个新的 byte 数组输出流。缓冲区的容量最初是 32 字节，如有必要可增加其大小。

ByteArrayOutputStream

```
public ByteArrayOutputStream(int size)
```

创建一个新的 byte 数组输出流，它具有指定大小的缓冲区容量（以字节为单位）。

参数：

size - 初始大小。

抛出：

IllegalArgumentException - 如果 size 为负。

方法详细信息

write

```
public void write(int b)
```

将指定的字节写入此 byte 数组输出流。

指定者：

类 OutputStream 中的 write

参数：

b - 要写入的字节。

write

```
public void write(byte[] b,
                  int off,
                  int len)
```

将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此 byte 数组输出流。

覆盖：

类 OutputStream 中的 write

参数：

b - 数据。

off - 数据的初始偏移量。

len - 要写入的字节数。

writeTo

```
public void writeTo(OutputStream out)
                    throws IOException
```

将此 byte 数组输出流的全部内容写入到指定的输出流参数中，这与使用 out.write(buf, 0, count) 调用该输出流的 write 方法效果一样。

参数：

out - 要写入数据的输出流。

抛出：

IOException - 如果发生 I/O 错误。

reset

```
public void reset()
```

将此 byte 数组输出流的 count 字段重置为零，从而丢弃输出流中目前已累积的所有输出。通过重新使用已分配的缓冲区空间，可以再次使用该输出流。

另请参见：

ByteArrayInputStream.count

toByteArray

```
public byte[] toByteArray()
```

创建一个新分配的 byte 数组。其大小是此输出流的当前大小，并且缓冲区的有效内容已复制到该数组中。

返回：

以 byte 数组的形式返回此输出流的当前内容。

另请参见：

size()

size

public int size()

返回缓冲区的当前大小。

返回：

count 字段的值，这是此输出流中有效字节的数目。

另请参见：

count

toString

public String toString()

使用平台默认的字符集，通过解码字节将缓冲区内容转换为字符串。新 String 的长度是字符集的函数，因此可能不等于缓冲区的大小。

此方法总是使用平台默认字符集的默认替代字符串替代错误输入 (malformed-input) 和不可映射字符 (unmappable-character) 序列。如果需要对解码过程进行更多控制，则应该使用 CharsetDecoder 类。

覆盖：

类 Object 中的 toString

返回：

从缓冲区内容解码的字符串。

从以下版本开始：

JDK1.1

toString

public String toString(String charsetName)
throws UnsupportedEncodingException

使用指定的 charsetName，通过解码字节将缓冲区内容转换为字符串。新 String 的长度是字符集的函数，因此可能不等于缓冲区的大小。

此方法总是使用平台默认字符集的默认替代字符串替代错误输入（malformed-input）和不可映射字符（unmappable-character）序列。如果需要对解码过程进行更多控制，则应该使用 CharsetDecoder 类。

参数：

charsetName - charset 支持的名称

返回：

从缓冲区内容解码的字符串。

抛出：

UnsupportedEncodingException - 如果不支持指定的字符集。

从以下版本开始：

JDK1.1

---

toString

```
@Deprecated
    public String toString(int hibyte)
```

已过时。 *此方法无法将字节正确转换为字符。从 JDK 1.1 开始，完成该转换的首选方法是通过 toString(String enc) 方法（使用一个编码名称参数），或 toString() 方法（使用平台的默认字符编码）。*

创建一个新分配的字符串。其大小是该输出流的当前大小，并且缓冲区的有效内容已复制到其中。得到的字符串中的每个字符 *c* 都根据 byte 数组中的相应元素 *b* 构造，如下所示：

```
c == (char)(((hibyte & 0xff) << 8) | (b & 0xff))
```

参数：

hibyte - 每个结果 Unicode 字符的高次字节。

返回：

以字符串的形式返回输出流的当前内容。

另请参见：

size(),toString(String),toString()

close

```
public void close()
                throws IOException
```

关闭 ByteArrayOutputStream 无效。此类中的方法在关闭此流后仍可被调用，而不会产生任何 IOException 时。

指定者：

接口 Closeable 中的 close

覆盖：

类 OutputStream 中的 close

抛出：

IOException - 如果发生 I/O 错误。

DataInputStream 类

数据输入流允许应用程序以与机器无关方式从底层输入流中读取基本 Java 数据类型。应用程序可以使用数据输出流写入稍后由数据输入流读取的数据。

DataInputStream 对于多线程访问不一定是安全的。 线程安全是可选的，它由此类方法的使用者负责。

构造方法详细信息

DataInputStream

```
public DataInputStream(InputStream in)
```

使用指定的底层 InputStream 创建一个 DataInputStream。

参数：

in - 指定输入流

方法详细信息

read

```
public final int read(byte[] b)
                throws IOException
```

从包含的输入流中读取一定数量的字节，并将它们存储到缓冲区数组 b 中。以整数形式返回实际读取的字节数。在输入数据可用、检测到文件末尾 (end of file) 或抛出异常之前，此方法将一直阻塞。

如果 b 为 null，则抛出 NullPointerException。如果 b 的长度为 0，则不读取字节并返回 0；否则，尝试读取至少一个字节。如果因为流位于文件末尾而没有字节可用，则返回值 -1；否则至少读取一个字节并将其存储到 b 中。

将读取的第一个字节存储到元素 b[0] 中，将下一个字节存储到 b[1] 中，依此类推。读取的字节数至多等于 b 的长度。设 k 为实际读取的字节数；这些字节将存储在从 b[0] 到 b[k-1] 的元素中，b[k] 到 b[b.length-1] 的元素不受影响。

read(b) 方法与以下方法的效果相同：

```
read(b, 0, b.length)
```

覆盖：

类 FilterInputStream 中的 read

参数：

b - 存储读取数据的缓冲区。

返回：

读入缓冲区的字节总数；如果因为已经到达流末尾而没有更多的数据，则返回 -1。

抛出：

IOException - 如果不是因为流位于文件末尾而无法读取第一个字节；该流已关闭并且底层输入流在关闭后不支持读取操作；发生其他 I/O 错误。

另请参见：

FilterInputStream.in,InputStream.read(byte[], int, int)

read

```
public final int read(byte[] b,
                      int off,
                      int len)
               throws IOException
```

从包含的输入流中将最多 len 个字节读入一个 byte 数组中。尽量读取 len 个字节，但读取的字节数可能少于 len 个，也可能为零。以整数形式返回实际读取的字节数。

在输入数据可用、检测到文件末尾或抛出异常之前，此方法将阻塞。

如果 len 为零，则不读取任何字节并返回 0；否则，尝试读取至少一个字节。如果因为流位于文件末尾而没有字节可用，则返回值 -1；否则，至少读取一个字节并将其存储到 b 中。

将读取的第一个字节存储到元素 b[off] 中，将下一个字节存储到 b[off+1] 中，依此类推。读取的字节数至多等于 len。设 *k* 为实际读取的字节数；这些字节将存储在 b[off] 到 b[off+*k*-1] 的元素中，b[off+*k*] 到 b[off+len-1] 的元素不受影响。

在所有情况下，b[0] 到 b[off] 的元素和 b[off+len] 到 b[b.length-1] 的元素都不受影响。

覆盖：

类 FilterInputStream 中的 read

参数：

b - 存储读取数据的缓冲区。

off - 目标数组 b 中的起始偏移量

len - 读取的最大字节数。

返回：

读入缓冲区的字节总数；如果因为已经到达流末尾而没有更多的数据，则返回 -1。

抛出：

NullPointerException - 如果 b 为 null。

IndexOutOfBoundsException - 如果 off 为负，len 为负，或者 len 大于 b.length - off

IOException - 如果不是因为流位于文件末尾而无法读取第一个字节；该流已关闭并且底层输入流在关闭后不支持读取操作；发生其他 I/O 错误。

另请参见：
FilterInputStream.in,InputStream.read(byte[], int, int)

readFully

```
public final void readFully(byte[] b)
                     throws IOException
```

参见 DataInput 的 readFully 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：
接口 DataInput 中的 readFully
参数：
b - 存储读取数据的缓冲区。
抛出：
EOFException - 如果此输入流在读取所有字节之前到达末尾。
IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。
另请参见：
FilterInputStream.in

readFully

```
public final void readFully(byte[] b,
                            int off,
                            int len)
                     throws IOException
```

参见 DataInput 的 readFully 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：
接口 DataInput 中的 readFully
参数：

b - 存储读取数据的缓冲区。
off - 数据的起始偏移量。
len - 要读取的字节数。
抛出：
EOFException - 如果此输入流在读取所有字节之前到达末尾。
IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。
另请参见：
FilterInputStream.in

---

skipBytes

```
public final int skipBytes(int n)
                    throws IOException
```

参见 DataInput 的 skipBytes 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：
接口 DataInput 中的 skipBytes
参数：
n - 要跳过的字节数。
返回：
实际跳过的字节数。
抛出：
IOException - 如果包含的输入流不支持查找操作；该流已关闭并且包含的输入流在关闭后不支持读取操作；发生其他 I/O 错误。

---

readBoolean

```
public final boolean readBoolean()
                    throws IOException
```

参见 DataInput 的 readBoolean 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：
接口 DataInput 中的 readBoolean
返回：
读取的 boolean 值。
抛出：
EOFException - 如果此输入流已经到达末尾。
IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生另其他 I/O 错误。
另请参见：
FilterInputStream.in

---

readByte

```
public final byte readByte()
                    throws IOException
```

参见 DataInput 的 readByte 方法的常规协定。

从所包含的输入流中读取此操作需要的字节。

指定者：
接口 DataInput 中的 readByte
返回：
此输入流的下一个字节，以有符号 8 位 byte 的形式表示。
抛出：
EOFException - 如果此输入流已经到达末尾。
IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。
另请参见：
FilterInputStream.in

---

readUnsignedByte

```
public final int readUnsignedByte()
```

throws IOException

参见 DataInput 的 readUnsignedByte 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：

接口 DataInput 中的 readUnsignedByte

返回：

此输入流的下一个字节，将它解释为一个无符号 8 位数。

抛出：

EOFException - 如果此输入流已经到达末尾。

IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。

另请参见：

FilterInputStream.in

---

readShort

public final short readShort()
throws IOException

参见 DataInput 的 readShort 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：

接口 DataInput 中的 readShort

返回：

此输入流的下两个字节，将它们解释为一个有符号 16 位数。

抛出：

EOFException - 如果此输入流在读取这两个字节之前到达末尾。

IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。

另请参见：

FilterInputStream.in

---

readUnsignedShort

```
public final int readUnsignedShort()
                                    throws IOException
```

参见 DataInput 的 readUnsignedShort 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：
接口 DataInput 中的 readUnsignedShort
返回：
此输入流的下两个字节，将它们解释为一个无符号 16 位整数。
抛出：
EOFException - 如果此输入流在读取这两个字节之前到达末尾。
IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。
另请参见：
FilterInputStream.in

---

readChar

```
public final char readChar()
                           throws IOException
```

参见 DataInput 的 readChar 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：
接口 DataInput 中的 readChar
返回：
此输入流的下两个字节，将它们解释为一个 char。
抛出：
EOFException - 如果此输入流在读取这两个字节之前到达末尾。
IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。
另请参见：

FilterInputStream.in

readInt

```
public final int readInt()
                    throws IOException
```

参见 DataInput 的 readInt 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：
接口 DataInput 中的 readInt
返回：
此输入流的下四个字节，将它们解释为一个 int。
抛出：
EOFException - 如果此输入流在读取这四个字节之前到达末尾。
IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。
另请参见：
FilterInputStream.in

readLong

```
public final long readLong()
                    throws IOException
```

参见 DataInput 的 readLong 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：
接口 DataInput 中的 readLong
返回：
此输入流的下八个字节，将它们解释为一个 long。
抛出：
EOFException - 如果此输入流在读取这八个字节之前到达末尾。

IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。

另请参见：

FilterInputStream.in

---

readFloat

```
public final float readFloat()
                        throws IOException
```

参见 DataInput 的 readFloat 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：

接口 DataInput 中的 readFloat

返回：

此输入流的下四个字节，将它们解释为一个 float。

抛出：

EOFException - 如果此输入流在读取这四个字节之前到达末尾。

IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。

另请参见：

readInt(),Float.intBitsToFloat(int)

---

readDouble

```
public final double readDouble()
                        throws IOException
```

参见 DataInput 的 readDouble 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：

接口 DataInput 中的 readDouble

返回：

此输入流的下八个字节，将它们解释为一个 double。

抛出：

EOFException - 如果此输入流在读取这八个字节之前到达末尾。

IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。

另请参见：

readLong(),Double.longBitsToDouble(long)

---

readLine

```
@Deprecated
    public final String readLine()
                                throws IOException
```

已过时。 *该方法无法将字节正确转换为字符。从 JDK 1.1 开始，读取文本行的首选方法是使用 BufferedReader.readLine() 方法。使用 DataInputStream 类读取文本行的程序可以改为使用 BufferedReader 类，只要将以下形式的代码：*

```
    DataInputStream d = new DataInputStream(in);
```

替换为：

```
    BufferedReader d
        = new BufferedReader(new InputStreamReader(in));
```

参见 DataInput 的 readLine 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：

接口 DataInput 中的 readLine

返回：

此输入流中的下一文本行。

抛出：

IOException - 如果发生 I/O 错误。

另请参见：

BufferedReader.readLine(),FilterInputStream.in

---

readUTF

```
public final String readUTF()
                    throws IOException
```

参见 DataInput 的 readUTF 方法的常规协定。

从包含的输入流中读取此操作需要的字节。

指定者：

接口 DataInput 中的 readUTF

返回：

一个 Unicode 字符串。

抛出：

EOFException - 如果此输入流在读取所有字节之前到达末尾。

IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其他 I/O 错误。

UTFDataFormatException - 如果这些字节不表示一个有效的、UTF-8 修改版编码的字符串。

另请参见：

readUTF(java.io.DataInput)

readUTF

```
public static final String readUTF(DataInput in)
                    throws IOException
```

从流 in 中读取用 UTF-8 修改版格式编码的 Unicode 字符格式的字符串；然后以 String 形式返回此字符串。UTF-8 修改版表示形式的一些细节与 DataInput 的 readUTF 方法完全相同。

参数：

in - 数据输入流。

返回：

一个 Unicode 字符串。

抛出：

EOFException - 如果此输入流在读取所有字节之前到达末尾。

IOException - 该流已关闭并且包含的输入流在关闭后不支持读取操作，或者发生其

他 I/O 错误。

UTFDataFormatException - 如果这些字节不表示一个有效的、UTF-8 修改版编码的 Unicode 字符串。

File 类

文件和目录路径名的抽象表示形式。

用户界面和操作系统使用与系统相关的路径名字符串 来命名文件和目录。此类呈现分层路径名的一个抽象的、与系统无关的视图。抽象路径名 有两个组件：

一个可选的与系统有关的前缀 字符串，比如盘符，"/" 表示 UNIX 中的根目录，"\\\\" 表示 Microsoft Windows UNC 路径名。

零个或更多字符串名称 的序列。

抽象路径名中的第一个名称是目录名，对于 Microsoft Windows UNC 路径名则是主机名。抽象路径名中第一个名称之后的每个名称表示一个目录；最后一个名称既可以表示目录，也可以表示文件。空 抽象路径名没有前缀和名称序列。

路径名字符串与抽象路径名之间的转换与系统有关。将抽象路径名转换为路径名字符串时，每个名称与下一个名称之间用一个默认分隔符 隔开。默认名称分隔符由系统属性 file.separator 定义，可通过此类的公共静态字段 separator 和 separatorChar 使其可用。将路径名字符串转换为抽象路径名时，可以使用默认名称分隔符或者底层系统支持的任何其他名称分隔符来分隔其中的名称。

无论是抽象路径名还是路径名字符串，都可以是绝对 路径名或相对 路径名。绝对路径名是完整的路径名，不需要任何其他信息就可以定位它所表示的文件。相反，相对路径名必须使用取自其他路径名的信息进行解释。默认情况下，java.io 包中的类总是根据当前用户目录来解析相对路径名。此目录由系统属性 user.dir 指定，通常是 Java 虚拟机的调用目录。

调用此类的 getParent() 方法可以获取抽象路径名的父 路径名，它由路径名前缀以及路径名名称序列中的每个名称（最后一个除外）组成。对于任何具有绝对抽象路径名的 File 对象，如果其绝对抽象路径名以某个目录的绝对路径名开头，那么该目录的绝对路径名是该 File 对象的祖先。例如，抽象路径名 "/usr" 表示的目录是路径名 "/usr/local/bin" 所表示目录的一个祖先。

在处理 UNIX 平台的根目录，以及 Microsoft Windows 平台的盘符、根目录和 UNC 路径名时，将用到前缀这一概念。如下所示：

对于 UNIX 平台，绝对路径名的前缀始终是 "/"。相对路径名没有前缀。表示根目录的绝对路径名的前缀为 "/" 且名称序列为空。

对于 Microsoft Windows 平台，包含盘符的路径名前缀由驱动器号和一个 ":" 组成。如果路径名是绝对路径名，还可能后跟 "\\"。UNC 路径名的前缀是 "\\\\"；主机名和共享名是名称序列中的前两个名称。没有指定驱动器的相对路径名没有前缀。

此类的实例可能表示（也可能不表示）实际文件系统对象，如文件或目录。如果它表示这种对象，那么该对象驻留在一个分区 中。分区是文件系统特定于操作系统的存储分区。一个存储设备（例如，物理磁盘驱动器、闪存、CD-ROM）可以包含多个分区。对象（如果有）将驻留在此路径名（绝对形式）某个祖先指定的分区上。

文件系统可以实现对实际文件系统对象上的某些操作（比如，读、写、执行）进行限制。这些限制统称为访问权限。文件系统可以对一个对象设置多个访问权限。例如，一个设置可能适用于对象的所有者，另一个设置则可能适用于所有其他用户。对象上的访问权限可能导致此类的某些方法执行失败。

File 类的实例是不可变的；也就是说，一旦创建，File 对象表示的抽象路径名将永不改变。

字段详细信息

separatorChar

public static final char separatorChar

与系统有关的默认名称分隔符。此字段被初始化为包含系统属性 file.separator 值的第一个字符。在 UNIX 系统上，此字段的值为 '/'；在 Microsoft Windows 系统上，它为 '\\'。

另请参见：

System.getProperty(java.lang.String)

separator

public static final String separator

与系统有关的默认名称分隔符，为了方便，它被表示为一个字符串。此字符串只包含一个字符，即 separatorChar。

pathSeparatorChar

public static final char pathSeparatorChar

与系统有关的路径分隔符。此字段被初始为包含系统属性 path.separator 值的第

一个字符。此字符用于分隔以*路径列表* 形式给定的文件序列中的文件名。在 UNIX 系统上，此字段为 ':'；在 Microsoft Windows 系统上，它为 ';'。

另请参见：

System.getProperty(java.lang.String)

pathSeparator

```
public static final String pathSeparator
```

与系统有关的路径分隔符，为了方便，它被表示为一个字符串。此字符串只包含一个字符，即 pathSeparatorChar。

构造方法详细信息

File

```
public File(String pathname)
```

通过将给定路径名字符串转换为抽象路径名来创建一个新 File 实例。如果给定字符串是空字符串，那么结果是空抽象路径名。

参数：

pathname - 路径名字符串

抛出：

NullPointerException - 如果 pathname 参数为 null

File

```
public File(String parent,
            String child)
```

根据 parent 路径名字符串和 child 路径名字符串创建一个新 File 实例。

如果 parent 为 null，则创建一个新的 File 实例，这与调用以给定 child 路径名字符串作为参数的单参数 File 构造方法效果一样。

否则，parent 路径名字符串用于表示目录，child 路径名字符串用于表示目录或文件。如果 child 路径名字符串是绝对路径名，则用与系统有关的方式将它转换为一个相对路径名。如果 parent 是空字符串，则通过将 child 转换为抽象路径名，并根据与系统有关的默认目录解析结果来创建新的 File 实例。否则，将每个路径名字符串转换为一个抽象路径名，并根据父抽象路径名解析子抽象路径名。

参数：

parent - 父路径名字符串

child - 子路径名字符串

抛出：

NullPointerException - 如果 child 为 null

File

```
public File(File parent,
            String child)
```

根据 parent 抽象路径名和 child 路径名字符串创建一个新 File 实例。

如果 parent 为 null，则创建一个新的 File 实例，这与调用给定 child 路径名字符串的单参数 File 构造方法的效果一样。

否则，parent 抽象路径名用于表示目录，child 路径名字符串用于表示目录或文件。如果 child 路径名字符串是绝对路径名，则用与系统有关的方式将它转换为一个相对路径名。如果 parent 是空抽象路径名，则通过将 child 转换为抽象路径名，并根据与系统有关的默认目录解析结果来创建新的 File 实例。否则，将每个路径名字符串转换为一个抽象路径名，并根据父抽象路径名解析子抽象路径名。

参数：

parent - 父抽象路径名

child - 子路径名字符串

抛出：

NullPointerException - 如果 child 为 null

File

public File(URI uri)

通过将给定的 file: URI 转换为一个抽象路径名来创建一个新的 File 实例。

file: URI 的具体形式与系统有关，因此，由此构造方法执行的转换也与系统有关。

对于某个给定抽象路径名 *f*，可以保证：

new File(f.toURI()).equals(f.getAbsoluteFile())

只要原始抽象路径名、URI 和新抽象路径名都是在同一 Java 虚拟机（或者它的不同调用）中创建的。但是，当在某一操作系统上的虚拟机中创建的 file: URI 在不同操作系统上的虚拟机中被转换为抽象路径名时，这种关系通常是不成立的。

参数：

uri - 一个绝对分层 URI，由一个等于 ″file″ 的 scheme、非空的 path 组件，以及未定义的 authority、query 和 fragment 组件组成

抛出：

NullPointerException - 如果 uri 为 null

IllegalArgumentException - 如果关于参数的前提不成立

从以下版本开始：

1.4

另请参见：

toURI(),URI

方法详细信息

getName

public String getName()

返回由此抽象路径名表示的文件或目录的名称。该名称是路径名名称序列中的最后一个名称。如果路径名名称序列为空，则返回空字符串。

返回：

此抽象路径名表示的文件或目录的名称；如果路径名的名称序列为空，则返回空字符串

getParent

public String getParent()

返回此抽象路径名父目录的路径名字符串；如果此路径名没有指定父目录，则返回 null。

抽象路径名的父 路径名由路径名的前缀（如果有），以及路径名名称序列中最后一个名称以外的所有名称组成。如果名称序列为空，那么该路径名没有指定父目录。

返回：
此抽象路径名指定父目录的路径名字符串；如果此路径名没有指定父目录，则返回 null

getParentFile

public File getParentFile()

返回此抽象路径名父目录的抽象路径名；如果此路径名没有指定父目录，则返回 null。

抽象路径名的父 路径名由路径名的前缀（如果有），以及路径名名称序列中最后一个名称以外的所有名称组成。如果名称序列为空，那么该路径名没有指定父目录。

返回：
此抽象路径名指定父目录的抽象路径名；如果此路径名没有指定父目录，则返回 null
从以下版本开始：
1.2

getPath

public String getPath()

将此抽象路径名转换为一个路径名字符串。所得字符串使用默认名称分隔符分隔名

称序列中的名称。

返回：

此抽象路径名的字符串形式

isAbsolute

```
public boolean isAbsolute()
```

测试此抽象路径名是否为绝对路径名。绝对路径名的定义与系统有关。在 UNIX 系统上，如果路径名的前缀是 "/"，那么该路径名是绝对路径名。在 Microsoft Windows 系统上，如果路径名的前缀是后跟 "\\" 的盘符，或者是 "\\\\"，那么该路径名是绝对路径名。

返回：

如果此抽象路径名是绝对路径名，则返回 true；否则返回 false

getAbsolutePath

```
public String getAbsolutePath()
```

返回此抽象路径名的绝对路径名字符串。

如果此抽象路径名已经是绝对路径名，则返回该路径名字符串，这与 getPath() 方法一样。如果此抽象路径名是空抽象路径名，则返回当前用户目录的路径名字符串，该目录由系统属性 user.dir 指定。否则，使用与系统有关的方式解析此路径名。在 UNIX 系统上，根据当前用户目录解析相对路径名，可使该路径名成为绝对路径名。在 Microsoft Windows 系统上，根据路径名指定的当前驱动器目录（如果有）解析相对路径名，可使该路径名成为绝对路径名；否则，可以根据当前用户目录解析它。

返回：

绝对路径名字符串，它与此抽象路径名表示相同的文件或目录

抛出：

SecurityException - 如果无法访问所需的系统属性值。

另请参见：

isAbsolute()

getAbsoluteFile

```
public File getAbsoluteFile()
```

返回此抽象路径名的绝对路径名形式。等同于 new File(this.getAbsolutePath())。

返回：

绝对抽象路径名，它与此抽象路径名表示相同的文件或目录

抛出：

SecurityException - 如果无法访问所需的系统属性值。

从以下版本开始：

1.2

getCanonicalPath

```
public String getCanonicalPath()
                        throws IOException
```

返回此抽象路径名的规范路径名字符串。

规范路径名是绝对路径名，并且是惟一的。规范路径名的准确定义与系统有关。如有必要，此方法首先将路径名转换为绝对路径名，这与调用 getAbsolutePath() 方法的效果一样，然后用与系统相关的方式将它映射到其惟一路径名。这通常涉及到从路径名中移除多余的名称（比如 ″.″ 和 ″..″）、解析符号连接（对于 UNIX 平台），以及将驱动器号转换为标准大小写形式（对于 Microsoft Windows 平台）。

每个表示现存文件或目录的路径名都有一个惟一的规范形式。每个表示不存在文件或目录的路径名也有一个惟一的规范形式。不存在文件或目录路径名的规范形式可能不同于创建文件或目录之后同一路径名的规范形式。同样，现存文件或目录路径名的规范形式可能不同于删除文件或目录之后同一路径名的规范形式。

返回：

规范路径名字符串，它与此抽象路径名表示相同的文件或目录

抛出：

IOException - 如果发生 I/O 错误（可能是因为构造规范路径名需要进行文件系统查询）

SecurityException - 如果无法访问所需的系统属性值，或者存在安全管理器，且其 SecurityManager.checkRead(java.io.FileDescriptor) 方法拒绝对文件进行读访问

从以下版本开始：

JDK1.1

---

getCanonicalFile

```
public File getCanonicalFile()
                        throws IOException
```

返回此抽象路径名的规范形式。等同于 new File(this.getCanonicalPath())。

返回：

规范路径名字符串，它与此抽象路径名表示相同的文件或目录

抛出：

IOException - 如果发生 I/O 错误（可能是因为构造规范路径名需要进行文件系统查询）

SecurityException - 如果无法访问所需的系统属性值，或者存在安全管理器，且其 SecurityManager.checkRead(java.io.FileDescriptor) 方法拒绝对文件进行读访问

从以下版本开始：

1.2

---

toURL

```
@Deprecated
public URL toURL()
                throws MalformedURLException
```

已过时。 *此方法不会自动转义 URL 中的非法字符。建议新的代码使用以下方式将抽象路径名转换为 URL：首先通过 toURI 方法将其转换为 URI，然后通过 URI.toURL 方法将 URI 装换为 URL。*

将此抽象路径名转换为一个 file: URL。该 URL 的具体形式与系统有关。如果可以确定此抽象路径名表示的文件是一个目录，那么所得 URL 将以斜杠结束。

返回：

表示等价文件 URL 的 URL 对象

抛出：
MalformedURLException - 如果无法将路径解析为 URL
从以下版本开始：
1.2
另请参见：
toURI(),URI,URI.toURL(),URL

toURI

public URI toURI()

构造一个表示此抽象路径名的 file: URI。

该 URI 的具体形式与系统有关。如果可以确定此抽象路径名表示的文件是一个目录，那么所得 URI 将以斜杠结束。

对于某个给定抽象路径名 *f*，可保证：

new File(*f*.toURI()).equals(*f*.getAbsoluteFile())
只要原始抽象路径名、URI 和新抽象路径名都是在同一 Java 虚拟机（或者它的不同调用）中创建的。但是，由于抽象路径名与系统有关的特性，当在某一操作系统上的虚拟机中创建的 file: URI 在不同操作系统上的虚拟机中被转换为抽象路径名时，这种关系通常是不成立的。
返回：
一个绝对分层 URI，由一个等于 "file" 的 scheme、表示此抽象路径名的 path，以及未定义的 authority、query 和 fragment 组件组成
抛出：
SecurityException - 如果不能访问请求的系统属性值。
从以下版本开始：
1.4
另请参见：
File(java.net.URI),URI,URI.toURL()

canRead

public boolean canRead()

测试应用程序是否可以读取此抽象路径名表示的文件。

返回：

当且仅当此抽象路径名指定的文件存在*且* 可被应用程序读取时，返回 true；否则返回 false

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对文件进行读访问

---

canWrite

```
public boolean canWrite()
```

测试应用程序是否可以修改此抽象路径名表示的文件。

返回：

当且仅当文件系统实际包含此抽象路径名表示的文件*且* 允许应用程序对该文件进行写入时，返回 true；否则返回 false。

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkWrite(java.lang.String) 方法拒绝对文件进行写访问

---

exists

```
public boolean exists()
```

测试此抽象路径名表示的文件或目录是否存在。

返回：

当且仅当此抽象路径名表示的文件或目录存在时，返回 true；否则返回 false

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对文件或目录进行写访问

---

isDirectory

public boolean isDirectory()

测试此抽象路径名表示的文件是否是一个目录。

返回：

当且仅当此抽象路径名表示的文件存在*且* 是一个目录时，返回 true；否则返回 false

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对文件进行读访问

---

isFile

public boolean isFile()

测试此抽象路径名表示的文件是否是一个标准文件。如果该文件不是一个目录，并且满足其他与系统有关的标准，那么该文件是*标准* 文件。由 Java 应用程序创建的所有非目录文件一定是标准文件。

返回：

当且仅当此抽象路径名表示的文件存在*且* 是一个标准文件时，返回 true；否则返回 false

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对文件进行读访问

---

isHidden

public boolean isHidden()

测试此抽象路径名指定的文件是否是一个隐藏文件。*隐藏* 的具体定义与系统有关。在 UNIX 系统上，如果文件名以句点字符 ('.') 开头，则认为该文件被隐藏。在 Microsoft Windows 系统上，如果在文件系统中文件被标记为隐藏，则认为该文件被隐藏。

返回：

当且仅当此抽象路径名表示的文件根据底层平台约定是隐藏文件时，返回 true

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对文件进行读访问

从以下版本开始：

1.2

---

lastModified

```
public long lastModified()
```

返回此抽象路径名表示的文件最后一次被修改的时间。

返回：

表示文件最后一次被修改的时间的 long 值，用与时间点（1970 年 1 月 1 日，00:00:00 GMT）之间的毫秒数表示；如果该文件不存在，或者发生 I/O 错误，则返回 0L

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对文件进行读访问

---

length

```
public long length()
```

返回由此抽象路径名表示的文件的长度。如果此路径名表示一个目录，则返回值是不确定的。

返回：

此抽象路径名表示的文件的长度，以字节为单位；如果文件不存在，则返回 0L。对于表示特定于系统的实体（比如设备或管道）的路径名，某些操作系统可能返回 0L。

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对文件进行读访问

createNewFile

```
public boolean createNewFile()
                    throws IOException
```

当且仅当不存在具有此抽象路径名指定名称的文件时，不可分地创建一个新的空文件。检查文件是否存在，若不存在则创建该文件，这是单个操作，对于其他所有可能影响该文件的文件系统活动来说，该操作是不可分的。

注：此方法*不应该* 用于文件锁定，因为所得协议可能无法可靠地工作。应该使用 FileLock 机制替代。

返回：

如果指定的文件不存在并成功地创建，则返回 true；如果指定的文件已经存在，则返回 false

抛出：

IOException - 如果发生 I/O 错误

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkWrite(java.lang.String) 方法拒绝对文件进行写访问

从以下版本开始：

1.2

delete

```
public boolean delete()
```

删除此抽象路径名表示的文件或目录。如果此路径名表示一个目录，则该目录必须为空才能删除。

返回：

当且仅当成功删除文件或目录时，返回 true；否则返回 false

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkDelete(java.lang.String) 方法拒绝对文件进行删除访问

deleteOnExit

public void deleteOnExit()

在虚拟机终止时，请求删除此抽象路径名表示的文件或目录。 文件（或目录）将以与注册相反的顺序删除。调用此方法删除已注册为删除的文件或目录无效。根据 Java 语言规范中的定义，只有在虚拟机正常终止时，才会尝试执行删除操作。

一旦请求了删除操作，就无法取消该请求。所以应小心使用此方法。

注：此方法*不应该* 用于文件锁定，因为所得协议可能无法可靠地工作。应该使用 FileLock 设施替代。

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkDelete(java.lang.String) 方法拒绝对文件进行删除访问

从以下版本开始：

1.2

另请参见：

delete()

list

public String[] list()

返回一个字符串数组，这些字符串指定此抽象路径名表示的目录中的文件和目录。

如果此抽象路径名不表示一个目录，那么此方法将返回 null。否则返回一个字符串数组，每个数组元素对应目录中的每个文件或目录。表示目录本身及其父目录的名称不包括在结果中。每个字符串是一个文件名，而不是一条完整路径。

不保证所得数组中的相同字符串将以特定顺序出现，特别是不保证它们按字母顺序出现。

返回：

字符串数组，这些字符串指定此抽象路径名表示的目录中的文件和目录。如果目录为

空，那么数组也将为空。如果此抽象路径名不表示一个目录，或者发生 I/O 错误，则返回 null。

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对目录进行读访问

---

list

public String[] list(FilenameFilter filter)

返回一个字符串数组,这些字符串指定此抽象路径名表示的目录中满足指定过滤器的文件和目录。除了返回数组中的字符串必须满足过滤器外，此方法的行为与 list() 方法相同。如果给定 filter 为 null，则接受所有名称。否则，当且仅当在此抽象路径名及其表示的目录中的文件名或目录名上调用过滤器的 FilenameFilter.accept(java.io.File, java.lang.String) 方法返回 true 时，该名称才满足过滤器。

参数：

filter - 文件名过滤器

返回：

字符串数组，这些字符串指定此抽象路径名表示的目录中给定 filter 能接受的文件和目录。如果目录为空，或者没有名称被过滤器接受，那么该数组将为空。如果抽象路径名不表示一个目录，或者发生 I/O 错误，则返回 null。

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对目录进行读访问

---

listFiles

public File[] listFiles()

返回一个抽象路径名数组，这些路径名表示此抽象路径名表示的目录中的文件。

如果此抽象路径名不表示一个目录，那么此方法将返回 null。否则返回一个 File 对象数组，每个数组元素对应目录中的每个文件或目录。表示

目录本身及其父目录的名称不包括在结果中。得到的每个抽象路径名都是根据此抽象路径名，使用 File(File, String) 构造方法构造的。所以，如果此路径名是绝对路径名，那么得到的每个路径名都是绝对路径名；如果此路径名是相对路径名，那么得到的每个路径名都是相对于同一目录的路径名。

不保证所得数组中的相同字符串将以特定顺序出现，特别是不保证它们按字母顺序出现。

返回：

抽象路径名数组，这些路径名表示此抽象路径名表示的目录中的文件和目录。如果目录为空，那么数组也将为空。如果抽象路径名不表示一个目录，或者发生 I/O 错误，则返回 null。

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对目录进行读访问

从以下版本开始：

1.2

---

listFiles

public File[] listFiles(FilenameFilter filter)

返回抽象路径名数组，这些路径名表示此抽象路径名表示的目录中满足指定过滤器的文件和目录。除了返回数组中的路径名必须满足过滤器外，此方法的行为与 listFiles() 方法相同。如果给定 filter 为 null，则接受所有路径名。否则，当且仅当在此抽象路径名及其表示的目录中的文件名或目录名上调用过滤器的 FilenameFilter.accept(java.io.File, java.lang.String) 方法返回 true 时，该路径名才满足过滤器。

参数：

filter - 文件名过滤器

返回：

抽象路径名数组，这些路径名表示此抽象路径名表示的目录中的文件和目录。如果目录为空，那么数组也将为空。如果此抽象路径名不表示一个目录，或者发生 I/O 错误，则返回 null。

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对目录进行读访问

从以下版本开始：

1.2

---

listFiles

```
public File[] listFiles(FileFilter filter)
```

返回抽象路径名数组，这些路径名表示此抽象路径名表示的目录中满足指定过滤器的文件和目录。除了返回数组中的路径名必须满足过滤器外，此方法的行为与 listFiles() 方法相同。如果给定 filter 为 null，则接受所有路径名。否则，当且仅当在路径名上调用过滤器的 FileFilter.accept(java.io.File) 方法返回 true 时，该路径名才满足过滤器。

参数：

filter - 文件过滤器

返回：

抽象路径名数组，这些路径名表示此抽象路径名表示的目录中的文件和目录。如果目录为空，那么数组也将为空。如果抽象路径名不表示一个目录，或者发生 I/O 错误，则返回 null。

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对目录进行读访问

从以下版本开始：

1.2

---

mkdir

```
public boolean mkdir()
```

创建此抽象路径名指定的目录。

返回：

当且仅当已创建目录时，返回 true；否则返回 false

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkWrite(java.lang.String) 方法不允许创建指定的目录

mkdirs

public boolean mkdirs()

创建此抽象路径名指定的目录，包括所有必需但不存在的父目录。注意，此操作失败时也可能已经成功地创建了一部分必需的父目录。

返回：

当且仅当已创建目录以及所有必需的父目录时，返回 true；否则返回 false

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法不允许验证指定目录和所有必需的父目录是否存在；或者 SecurityManager.checkWrite(java.lang.String) 方法不允许创建指定的目录和所有必需的父目录

renameTo

public boolean renameTo(File dest)

重新命名此抽象路径名表示的文件。

此方法行为的许多方面都是与平台有关的：重命名操作无法将一个文件从一个文件系统移动到另一个文件系统，该操作不是不可分的，如果已经存在具有目标抽象路径名的文件，那么该操作可能无法获得成功。应该始终检查返回值，以确保重命名操作成功。

参数：

dest - 指定文件的新抽象路径名

返回：

当且仅当重命名成功时，返回 true；否则返回 false

抛出：

SecurityException - 如果存在安全管理器，且其

SecurityManager.checkWrite(java.lang.String) 方法拒绝对原路径名和新路径名进行写访问

NullPointerException - 如果参数 dest 为 null

setLastModified

public boolean setLastModified(long time)

设置此抽象路径名指定的文件或目录的最后一次修改时间。

所有平台都支持将文件修改时间设置为最接近的秒数，而且一些平台会提供更精确的值。该参数将被截取，以满足受支持的精度。如果该操作成功，并且没有在文件上发生其他干扰操作，则下一次调用 lastModified() 方法将返回传递给此方法的 time 参数（可能被截取）。

参数：

time - 新的最后一次修改时间，用与时间点（1970 年 1 月 1 日，00:00:00 GMT）之间的毫秒数表示

返回：

当且仅当该操作成功时，返回 true；否则返回 false

抛出：

IllegalArgumentException - 如果该参数为负

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkWrite(java.lang.String) 方法拒绝对指定文件进行写访问

从以下版本开始：

1.2

setReadOnly

public boolean setReadOnly()

标记此抽象路径名指定的文件或目录，从而只能对其进行读操作。调用此方法后，可以保证在被删除或被标记为允许写访问之前，文件或目录不会发生更改。是否可以删除某个只读文件或目录则取决于底层系统。

返回：

当且仅当该操作成功时，返回 true；否则返回 false

抛出：

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkWrite(java.lang.String) 方法拒绝对指定文件进行写访问

从以下版本开始：

1.2

---

setWritable

```
public boolean setWritable(boolean writable,
                           boolean ownerOnly)
```

设置此抽象路径名的所有者或所有用户的写权限。

参数：

writable - 如果为 true，则设置允许写操作的访问权限；如果为 false，则不允许写操作。

ownerOnly - 如果为 true，则写权限只适用于所有者的写权限；否则适用于所有用户。如果底层文件系统不能区分所有者写权限与其他写权限，那么无论该参数为何值，写权限将适用于所有用户。

返回：

当且仅当操作成功时返回 true。如果用户不具有更改此抽象路径名访问权限的权限，那么操作将失败。

抛出：

SecurityException - 如果安全管理器存在且其 SecurityManager.checkWrite(java.lang.String) 方法拒绝对指定文件进行写访问。

从以下版本开始：

1.6

---

setWritable

```
public boolean setWritable(boolean writable)
```

设置此抽象路径名所有者写权限的一个便捷方法。

此方法 file.setWritable(arg) 形式的调用与以下调用的行为完全相同：

```
file.setWritable(arg, true)
```

参数：

writable - 如果为 true，则设置允许写操作的访问权限；如果为 false，则不允许写操作。

返回：

当且仅当操作成功时返回 true。如果用户不具有更改此抽象路径名访问权限的权限，那么操作将失败。

抛出：

SecurityException - 如果安全管理器存在且其 SecurityManager.checkWrite(java.lang.String) 方法拒绝对文件进行写访问。

从以下版本开始：

1.6

setReadable

```
public boolean setReadable(boolean readable,
                           boolean ownerOnly)
```

设置此抽象路径名的所有者或所有用户的读权限。

参数：

readable - 如果为 true，则设置允许读操作的访问权限；如果为 false，则不允许读操作。

ownerOnly - 如果为 true，则读权限只适用于所有者的读权限；否则适用于所有用户。如果底层文件系统不能区分所有者读权限与其他读权限，那么无论该参数为何值，读权限将适用于所有用户。

返回：

当且仅当操作成功时返回 true。如果用户不具有更改此抽象路径名访问权限的权限，那么操作将失败。如果 readable 为 false，并且底层文件系统不实现读权限，那么操作也将失败。

抛出：

SecurityException - 如果安全管理器存在且其 SecurityManager.checkWrite(java.lang.String) 方法拒绝对文件进行写访问。

从以下版本开始：

1.6

setReadable

```
public boolean setReadable(boolean readable)
```

设置此抽象路径名所有者读权限的一个便捷方法。

此方法 file.setReadable(arg) 形式的调用与以下调用的行为完全相同：

```
file.setReadable(arg, true)
```

参数：

readable - 如果为 true，则设置允许读操作的访问权限；如果为 false，则不允许读操作。

返回：

当且仅当操作成功时返回 true。如果用户不具有更改此抽象路径名访问权限的权限，那么操作将失败。如果 readable 为 false，并且底层文件系统不实现读权限，那么操作也将失败。

抛出：

SecurityException - 如果安全管理器存在且其 SecurityManager.checkWrite(java.lang.String) 方法拒绝对文件进行写访问。

从以下版本开始：

1.6

setExecutable

```
public boolean setExecutable(boolean executable,
                             boolean ownerOnly)
```

设置此抽象路径名的所有者或所有用户的执行权限。

参数：

executable - 如果为 true，则设置允许执行操作的访问权限；如果为 false，则不允许执行操作。

ownerOnly - 如果为 true，则执行权限只适用于所有者的执行权限；否则适用于所有用户。如果底层文件系统不能区分所有者执行权限与其他执行权限，那么无论该参数为何值，执行权限将适用于所有用户。

返回：

当且仅当操作成功时返回 true。如果用户不具有更改此抽象路径名访问权限的权限，那么操作将失败。如果 executable 为 false，并且底层文件系统不实现执行权限，那么操作也将失败。

抛出：

SecurityException - 如果安全管理器存在且其 SecurityManager.checkWrite(java.lang.String) 方法拒绝对文件进行写访问。

从以下版本开始：

1.6

setExecutable

```
public boolean setExecutable(boolean executable)
```

设置此抽象路径名所有者执行权限的一个便捷方法。

此方法 file.setExcutable(arg) 形式的调用与以下调用的行为完全相同：

```
file.setExecutable(arg, true)
```

参数：

executable - 如果为 true，则设置允许执行操作的访问权限；如果为 false，则不允许执行操作。

返回：

当且仅当操作成功时返回 true。如果用户不具有更改此抽象路径名访问权限的权限，那么操作将失败。如果 executable 为 false，并且底层文件系统不实现执行权限，那么操作也将失败。

抛出：

SecurityException - 如果安全管理器存在且其 SecurityManager.checkWrite(java.lang.String) 方法拒绝对文件进行写访问。

从以下版本开始：

1.6

canExecute

public boolean canExecute()

测试应用程序是否可以执行此抽象路径名表示的文件。

返回：

当且仅当抽象路径名存在*且* 允许应用程序执行文件时返回 true。

抛出：

SecurityException - 如果安全管理器存在且其 SecurityManager.checkExec(java.lang.String) 方法拒绝对文件进行执行访问。

从以下版本开始：

1.6

listRoots

public static File[] listRoots()

列出可用的文件系统根。

特定 Java 平台可以支持零个或更多个分层组织的文件系统。每个文件系统有一个 root 目录，可以从这里到达文件系统中的所有其他文件。例如，Windows 平台为每个活动驱动器提供了一个根目录；UNIX 平台只有一个根目录，即 "/"。可用文件系统根的设置受各种系统级操作的影响，比如可移动介质的插入和弹出，以及断开或卸载那些物理磁盘或虚拟磁盘。

此方法返回一个 File 对象数组，这些对象表示可用文件系统根的根目录。可以保证本地机器上物理存在的任何文件的规范路径名都以此方法返回的根之一开始。

位于其他一些机器上的文件的规范路径名是通过远程文件系统协议（比如 SMB 或 NFS）访问的，它们可能以此方法返回的根之一开始，也可能不是这样。如果远程文件的路径名在语法上无法与本地文件的路径名进行区分，那么它将以此方法返回的根之一开始。例如，此方法将返回表示 Windows 平台上映射为网络驱动器根目录的 File 对象，而不返回包含 UNC 路径名的 File 对象。

与此类中的大多数方法不同，此方法不抛出安全性异常。如果存在安全管理器，且其 SecurityManager.checkRead(java.lang.String) 方法拒绝对特定根目录进行读访问，那么该目录将不会出现在结果中。

返回：

表示可用文件系统根的 File 对象数组；如果无法确定根集，则返回 null。如果没有文件系统，那么该数组将为空。

从以下版本开始：

1.2

---

getTotalSpace

```
public long getTotalSpace()
```

返回此抽象路径名指定的分区大小。

返回：

分区的大小，以字节为单位；如果此抽象路径名没有指定分区，则返回 0L。

抛出：

SecurityException - 如果安装了安全管理器，并且安全管理器拒绝 RuntimePermission("getFileSystemAttributes") ，或者其 SecurityManager.checkRead(String) 方法拒绝对此抽象路径名指定的文件进行读访问

从以下版本开始：

1.6

---

getFreeSpace

```
public long getFreeSpace()
```

返回此抽象路径名指定的分区中未分配的字节数。

返回的未分配字节数是一个提示，而不是一个保证，只能使用这些字节的一部分。未分配字节数很可能在此调用后立即与实际相符。某些外部 I/O 操作可能使其与实际不符，这些操作包括那些在此虚拟机外部系统上进行的操作。此方法不保证对此文件系统的写操作将成功。

返回：

分区上的未分配字节数；如果抽象路径名没有指定分区，则返回 0L。返回值将小于等于 getTotalSpace() 返回的总文件系统大小。

抛出：

SecurityException - 如果安装了安全管理器，并且安全管理器拒绝 RuntimePermission("getFileSystemAttributes") ，或者其 SecurityManager.checkRead(String) 方法拒绝对此抽象路径名指定的文件进行读访问

从以下版本开始：

1.6

---

getUsableSpace

```
public long getUsableSpace()
```

返回此抽象路径名指定的分区上可用于此虚拟机的字节数。若有可能，此方法将检查写权限和其他操作系统限制，因此与 getFreeSpace() 相比，此方法能更准确地估计可实际写入的新数据数。

返回的可用字节数是一个提示，而不是一个保证，只能使用这些字节的一部分。未分配字节数很可能在此调用后立即与实际相符。某些外部 I/O 操作可能使其与实际不符，这些操作包括那些在此虚拟机外部系统上进行的操作。此方法不保证对此文件系统的写操作将成功。

返回：

分区上的可用字节数；如果抽象路径名没有指定分区，则返回 0L。在此信息不可用的系统上，此方法等效于调用 getFreeSpace()。

抛出：

SecurityException - 如果安装了安全管理器，并且安全管理器拒绝 RuntimePermission("getFileSystemAttributes") ，或者其 SecurityManager.checkRead(String) 方法拒绝对此抽象路径名指定的文件进行读访问

从以下版本开始：

1.6

---

createTempFile

```
public static File createTempFile(String prefix,
                                  String suffix,
                                  File directory)
                           throws IOException
```

在指定目录中创建一个新的空文件，使用给定的前缀和后缀字符串生成其名称。如果此方法成功返回，则可以保证：

1. 由返回的抽象路径名表示的文件在此方法被调用之前不存在。
2. 此方法及其所有变体都不会在虚拟机的当前调用中再次返回相同的抽象路径名。

此方法只提供了临时文件的部分功能。要安排自动删除此方法创建的文件，可使用 deleteOnExit() 方法。

prefix 参数至少必须是三个字节长。建议前缀使用一个短的、有意义的字符串，比如 "hjb" 或 "mail"。suffix 参数可以为 null，在这种情况下，将使用后缀 ".tmp"。

要创建新文件，可能首先要调整前缀和后缀，使其满足底层平台的限制。如果前缀太长，则将它截断，但前三个字符将始终保留。如果后缀太长，则将它截断，但如果它以句点字符 ('.') 开始，则该句点以及后跟的前三个字符将始终保留。进行了这些调整后，通过连接前缀、五个或更多个内部生成的字符以及后缀，便生成了新文件的名称。

如果 directory 参数为 null，则使用与系统有关的默认临时文件目录。默认临时文件目录由系统属性 java.io.tmpdir 指定。在 UNIX 系统上，此属性的默认值通常是 "/tmp" 或 "/var/tmp"；在 Microsoft Windows 系统上，该值通常是 "C:\\WINNT\\TEMP"。在调用 Java 虚拟机时，可为此系统属性提供不同的值，但不保证使用程序更改此属性会对此方法使用的临时目录产生影响。

参数：

prefix - 用于生成文件名的前缀字符串；必须至少是三字符长

suffix - 用于生成文件名的后缀字符串；可以为 null，在这种情况下，将使用后缀 ".tmp"

directory - 将创建的文件所在的目录；如果使用默认临时文件目录，则该参数为 null

返回：

表示新建空文件的抽象路径名

抛出：

IllegalArgumentException - 如果 prefix 参数包含的字符少于三个

IOException - 如果无法创建文件

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkWrite(java.lang.String) 方法不允许创建文件

从以下版本开始：

1.2

---

createTempFile

```
public static File createTempFile(String prefix,
                                  String suffix)
                           throws IOException
```

在默认临时文件目录中创建一个空文件，使用给定前缀和后缀生成其名称。调用此方法等同于调用 createTempFile(prefix, suffix, null)。

参数：

prefix - 用于生成文件名的前缀字符串；必须至少是三字符长

suffix - 用于生成文件名的后缀字符串；可以为 null，在这种情况下，将使用后缀 ".tmp"

返回：

表示新建空文件的抽象路径名

抛出：

IllegalArgumentException - 如果 prefix 参数包含的字符少于三个

IOException - 如果无法创建文件

SecurityException - 如果存在安全管理器，且其 SecurityManager.checkWrite(java.lang.String) 方法不允许创建文件

从以下版本开始：

1.2

---

compareTo

```
public int compareTo(File pathname)
```

按字母顺序比较两个抽象路径名。此方法定义的顺序取决于底层系统。在 UNIX 系

统上，比较路径名时，字母大小写通常很重要，而在 Microsoft Windows 系统上，这通常不重要。

指定者：

接口 Comparable<File> 中的 compareTo

参数：

pathname - 将与此抽象路径名进行比较的抽象路径名

返回：

如果该参数等于此抽象路径名，则返回零；如果此抽象路径名在字母顺序上小于该参数，则返回小于零的值；如果此抽象路径名在字母顺序上大于该参数，则返回大于零的值

从以下版本开始：

1.2

---

equals

```
public boolean equals(Object obj)
```

测试此抽象路径名与给定对象是否相等。当且仅当该参数不是 null，而是一个与此抽象路径名表示相同的文件或目录的抽象路径名时，返回 true。两个抽象路径名是否相等取决于底层系统。在 UNIX 系统上，比较路径名时，字母大小写通常很重要，而在 Microsoft Windows 系统上，这通常不重要。

覆盖：

类 Object 中的 equals

参数：

obj - 要与此抽象路径名进行比较的对象

返回：

当且仅当对象相同时，返回 true；否则返回 false

另请参见：

Object.hashCode(),Hashtable

---

hashCode

```
public int hashCode()
```

计算此抽象路径名的哈希码。因为抽象路径名的相等性与系统有关，所以对其哈希码的计算也与系统有关。在 UNIX 系统上，抽象路径名的哈希码等于其路径名字符串

和十进制值 1234321 的哈希码的*异或*。在 Microsoft Windows 系统上，哈希码等于其转换为小写的路径名字符串和十进制值 1234321 的哈希码的*异或*。在将路径名字符串转换为小写时不考虑语言环境。

覆盖：

类 Object 中的 hashCode

返回：

此抽象路径名的哈希码

另请参见：

Object.equals(java.lang.Object),Hashtable

---

toString

```
public String toString()
```

返回此抽象路径名的路径名字符串。该字符串就是 getPath() 方法返回的字符串。

覆盖：

类 Object 中的 toString

返回：

此抽象路径名的字符串形式

FileInputStream 类

FileInputStream 从文件系统中的某个文件中获得输入字节。哪些文件可用取决于主机环境。

FileInputStream 用于读取诸如图像数据之类的原始字节流。要读取字符流，请考虑使用 FileReader。

构造方法详细信息

FileInputStream

```
public FileInputStream(String name)
                       throws FileNotFoundException
```

通过打开一个到实际文件的连接来创建一个 FileInputStream，该文件通过文件系统中的路径名 name 指定。创建一个新 FileDescriptor 对象来表示此文件连接。

首先，如果有安全管理器，则用 name 作为参数调用其 checkRead 方法。

如果指定文件不存在，或者它是一个目录，而不是一个常规文件，抑或因为其他某些原因而无法打开进行读取，则抛出 FileNotFoundException。

参数：

name - 与系统有关的文件名。

抛出：

FileNotFoundException - 如果该文件不存在，或者它是一个目录，而不是一个常规文件，抑或因为其他某些原因而无法打开进行读取。

SecurityException - 如果存在安全管理器，且其 checkRead 方法拒绝对文件进行读取访问。

另请参见：

SecurityManager.checkRead(java.lang.String)

FileInputStream

```
public FileInputStream(File file)
                throws FileNotFoundException
```

通过打开一个到实际文件的连接来创建一个 FileInputStream，该文件通过文件系统中的 File 对象 file 指定。创建一个新 FileDescriptor 对象来表示此文件连接。

首先，如果有安全管理器，则用 file 参数表示的路径作为参数调用其 checkRead 方法。

如果指定文件不存在，或者它是一个目录，而不是一个常规文件，抑或因为其他某些原因而无法打开进行读取，则抛出 FileNotFoundException。

参数：

file - 为了进行读取而打开的文件。

抛出：

FileNotFoundException - 如果该文件不存在，或者它是一个目录，而不是一个常规文

件，抑或因为其他某些原因而无法打开进行读取。

SecurityException - 如果存在安全管理器，且其 checkRead 方法拒绝对文件进行读取访问。

另请参见：

File.getPath(),SecurityManager.checkRead(java.lang.String)

FileInputStream

```
public FileInputStream(FileDescriptor fdObj)
```

通过使用文件描述符 fdObj 创建一个 FileInputStream，该文件描述符表示到文件系统中某个实际文件的现有连接。

首先，如果有安全管理器，则用文件描述符 fdObj 作为参数调用其 checkRead 方法，以查看它是否可以读取该文件描述符。如果拒绝对该文件描述符进行读取访问，则抛出 SecurityException。

如果 fdObj 为 null，则抛出 NullPointerException。

参数：

fdObj - 为了进行读取而打开的文件描述符。

抛出：

SecurityException - 如果存在安全管理器，且其 checkRead 方法拒绝对文件描述符进行读取访问

另请参见：

SecurityManager.checkRead(java.io.FileDescriptor)

方法详细信息

read

```
public int read()
            throws IOException
```

从此输入流中读取一个数据字节。如果没有输入可用，则此方法将阻塞。

指定者：

类 InputStream 中的 read

返回：

下一个数据字节；如果已到达文件末尾，则返回 -1。

抛出：

IOException - 如果发生 I/O 错误。

read

```
public int read(byte[] b)
         throws IOException
```

从此输入流中将最多 b.length 个字节的数据读入一个 byte 数组中。在某些输入可用之前，此方法将阻塞。

覆盖：

类 InputStream 中的 read

参数：

b - 存储读取数据的缓冲区。

返回：

读入缓冲区的字节总数，如果因为已经到达文件末尾而没有更多的数据，则返回 -1。

抛出：

IOException - 如果发生 I/O 错误。

另请参见：

InputStream.read(byte[], int, int)

read

```
public int read(byte[] b,
                int off,
                int len)
         throws IOException
```

从此输入流中将最多 len 个字节的数据读入一个 byte 数组中。如果 len 不为 0，则在输入可用之前，该方法将阻塞；否则，不读取任何字节并返回 0。

覆盖：

类 InputStream 中的 read

参数：

b - 存储读取数据的缓冲区。
off - 目标数组 b 中的起始偏移量。
len - 读取的最大字节数。
返回：
读入缓冲区的字节总数，如果因为已经到达文件末尾而没有更多的数据，则返回 -1。
抛出：
NullPointerException - 如果 b 为 null。
IndexOutOfBoundsException - 如果 off 为负、len 为负，或者 len 大于 b.length - off
IOException - 如果发生 I/O 错误。
另请参见：
InputStream.read()

skip

```
public long skip(long n)
          throws IOException
```

从输入流中跳过并丢弃 n 个字节的数据。

出于各种原因，skip 方法最终跳过的字节数可能更少一些，甚至可能为 0。如果 n 为负，则抛出 IOException，即使 InputStream 超类的 skip 方法在这种情况下没有执行任何操作。返回实际跳过的字节数。

此方法跳过的字节可能多于底层实现文件中剩余的字节。这不会产生异常，并且跳过的字节数可能包括底层实现文件的 EOF（文件结束符）之后的一些字节数。如果试图在跳过末尾之后读取流，那么会返回 -1（指示文件结束）。

覆盖：
类 InputStream 中的 skip
参数：
n - 要跳过的字节数。
返回：
实际跳过的字节数。
抛出：
IOException - 如果 n 为负，如果该流不支持查找操作，或者发生 I/O 错误。

available

```
public int available()
                    throws IOException
```

返回下一次对此输入流调用的方法可以不受阻塞地从此输入流读取（或跳过）的估计剩余字节数。下一次调用可能是同一个线程，也可能是另一个线程。一次读取或跳过此数量个字节不会发生阻塞，但读取或跳过的字节可能小于该数。

在某些情况下，非阻塞的读取（或跳过）操作在执行很慢时看起来受阻塞，例如，在网速缓慢的网络上读取大文件时。

覆盖：
类 InputStream 中的 available
返回：
可以不受阻塞地从此输入流中读取（或跳过）的估计剩余字节数。
抛出：
IOException - 如果此文件输入流已通过调用 close 关闭，或者发生 I/O 错误。

close

```
public void close()
                throws IOException
```

关闭此文件输入流并释放与此流有关的所有系统资源。

如果此流有一个与之关联的通道，则关闭该通道。

指定者：
接口 Closeable 中的 close
覆盖：
类 InputStream 中的 close
抛出：
IOException - 如果发生 I/O 错误。

getFD

public final FileDescriptor getFD()
throws IOException

返回表示到文件系统中实际文件的连接的 FileDescriptor 对象,该文件系统正被此 FileInputStream 使用。

返回:

与此流有关的文件描述符对象。

抛出:

IOException - 如果发生 I/O 错误。

另请参见:

FileDescriptor

getChannel

public FileChannel getChannel()

返回与此文件输入流有关的唯一 FileChannel 对象。

所返回通道的初始 position 将等于到目前为止从文件中读取的字节数。从此流中读取的字节会使通道的位置递增。显式地或通过读取来更改通道的位置会更改此流的文件位置。

返回:

与此文件输入流有关的文件通道

从以下版本开始:

1.4

finalize

protected void finalize()
throws IOException

确保在不再引用文件输入流时调用其 close 方法。

覆盖:

类 Object 中的 finalize

抛出:

IOException - 如果发生 I/O 错误。

另请参见：
close()

FileOutputStream 类

文件输出流是用于将数据写入 File 或 FileDescriptor 的输出流。文件是否可用或能否可以被创建取决于基础平台。特别是某些平台一次只允许一个 FileOutputStream（或其他文件写入对象）打开文件进行写入。在这种情况下，如果所涉及的文件已经打开，则此类中的构造方法将失败。

FileOutputStream 用于写入诸如图像数据之类的原始字节的流。要写入字符流，请考虑使用 FileWriter。

构造方法详细信息

FileOutputStream

```
public FileOutputStream(String name)
                 throws FileNotFoundException
```

创建一个向具有指定名称的文件中写入数据的输出文件流。创建一个新 FileDescriptor 对象来表示此文件连接。

首先，如果有安全管理器，则用 name 作为参数调用 checkWrite 方法。

如果该文件存在，但它是一个目录，而不是一个常规文件；或者该文件不存在，但无法创建它；抑或因为其他某些原因而无法打开它，则抛出 FileNotFoundException。

参数：
name - 与系统有关的文件名
抛出：
FileNotFoundException - 如果文件存在，但它是一个目录，而不是一个常规文件；或者该文件不存在，但无法创建它；抑或因为其他某些原因而无法打开它
SecurityException - 如果存在安全管理器，且其 checkWrite 方法拒绝对文件进行写入访问。

另请参见：
SecurityManager.checkWrite(java.lang.String)

FileOutputStream

```
public FileOutputStream(String name,
                        boolean append)
                 throws FileNotFoundException
```

创建一个向具有指定 name 的文件中写入数据的输出文件流。如果第二个参数为 true，则将字节写入文件末尾处，而不是写入文件开始处。创建一个新 FileDescriptor 对象来表示此文件连接。

首先，如果有安全管理器，则用 name 作为参数调用 checkWrite 方法。

如果该文件存在，但它是一个目录，而不是一个常规文件；或者该文件不存在，但无法创建它；抑或因为其他某些原因而无法打开它，则抛出 FileNotFoundException。

参数：
name - 与系统有关的文件名
append - 如果为 true，则将字节写入文件末尾处，而不是写入文件开始处
抛出：
FileNotFoundException - 如果该文件存在，但它是一个目录，而不是一个常规文件；或者该文件不存在，但无法创建它；抑或因为其他某些原因而无法打开它。
SecurityException - 如果存在安全管理器，且其 checkWrite 方法拒绝对文件进行写入访问。
从以下版本开始：
JDK1.1
另请参见：
SecurityManager.checkWrite(java.lang.String)

FileOutputStream

```
public FileOutputStream(File file)
                 throws FileNotFoundException
```

创建一个向指定 File 对象表示的文件中写入数据的文件输出流。创建一个新 FileDescriptor 对象来表示此文件连接。

首先，如果有安全管理器，则用 file 参数表示的路径作为参数来调用 checkWrite 方法。

如果该文件存在，但它是一个目录，而不是一个常规文件；或者该文件不存在，但无法创建它；抑或因为其他某些原因而无法打开，则抛出 FileNotFoundException。

参数：

file - 为了进行写入而打开的文件。

抛出：

FileNotFoundException - 如果该文件存在，但它是一个目录，而不是一个常规文件；或者该文件不存在，但无法创建它；抑或因为其他某些原因而无法打开

SecurityException - 如果存在安全管理器，且其 checkWrite 方法拒绝对文件进行写入访问。

另请参见：

File.getPath(), SecurityException, SecurityManager.checkWrite(java.lang.String)

FileOutputStream

```
public FileOutputStream(File file,
                        boolean append)
                 throws FileNotFoundException
```

创建一个向指定 File 对象表示的文件中写入数据的文件输出流。如果第二个参数为 true，则将字节写入文件末尾处，而不是写入文件开始处。创建一个新 FileDescriptor 对象来表示此文件连接。

首先，如果有安全管理器，则用 file 参数表示的路径作为参数来调用 checkWrite 方法。

如果该文件存在，但它是一个目录，而不是一个常规文件；或者该文件不存在，但无法创建它；抑或因为其他某些原因而无法打开它，则抛出 FileNotFoundException。

参数：
file - 为了进行写入而打开的文件。
append - 如果为 true，则将字节写入文件末尾处，而不是写入文件开始处
抛出：
FileNotFoundException - 如果该文件存在，但它是一个目录，而不是一个常规文件；或者该文件不存在，但无法创建它；抑或因为其他某些原因而无法打开它
SecurityException - 如果存在安全管理器，且其 checkWrite 方法拒绝对文件进行写入访问。
从以下版本开始：
1.4
另请参见：
File.getPath(), SecurityException, SecurityManager.checkWrite(java.lang.String)

FileOutputStream

public FileOutputStream(FileDescriptor fdObj)

创建一个向指定文件描述符处写入数据的输出文件流，该文件描述符表示一个到文件系统中的某个实际文件的现有连接。

首先，如果有安全管理器，则用文件描述符 fdObj 参数作为参数来调用 checkRead 方法。

参数：
fdObj - 为进行写入而打开的文件描述符
抛出：
SecurityException - 如果存在安全管理器，且其 checkWrite 方法拒绝对文件描述符进行写入访问
另请参见：
SecurityManager.checkWrite(java.io.FileDescriptor)

方法详细信息

write

```
public void write(int b)
                throws IOException
```

将指定字节写入此文件输出流。实现 OutputStream 的 write 方法。

指定者：

类 OutputStream 中的 write

参数：

b - 要写入的字节。

抛出：

IOException - 如果发生 I/O 错误。

write

```
public void write(byte[] b)
                throws IOException
```

将 b.length 个字节从指定 byte 数组写入此文件输出流中。

覆盖：

类 OutputStream 中的 write

参数：

b - 数据。

抛出：

IOException - 如果发生 I/O 错误。

另请参见：

OutputStream.write(byte[], int, int)

write

```
public void write(byte[] b,
                  int off,
                  int len)
                throws IOException
```

将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此文件输出流。

覆盖：

类 OutputStream 中的 write

参数：

b - 数据。
off - 数据中的起始偏移量。
len - 要写入的字节数。
抛出：
IOException - 如果发生 I/O 错误。

close

```
public void close()
                throws IOException
```

关闭此文件输出流并释放与此流有关的所有系统资源。此文件输出流不能再用于写入字节。

如果此流有一个与之关联的通道，则关闭该通道。

指定者：
接口 Closeable 中的 close
覆盖：
类 OutputStream 中的 close
抛出：
IOException - 如果发生 I/O 错误。

getFD

```
public final FileDescriptor getFD()
                                  throws IOException
```

返回与此流有关的文件描述符。
返回：
表示到文件系统中的某个文件的连接的 FileDescriptor 对象，该文件系统正被此 FileOutputStream 对象使用。
抛出：
IOException - 如果发生 I/O 错误。
另请参见：
FileDescriptor

getChannel

```
public FileChannel getChannel()
```

返回与此文件输出流有关的唯一 FileChannel 对象。

所返回通道的初始
java.nio.channels.FileChannel#position()position 将等于到目前为止从文件中读取的字节数，除非此流处于挂起模式，在这种情况下，它将等于文件的大小。将字节写入此流中会使通道的位置相应地增加。显式地或通过写入来更改通道的位置会更改此流的文件位置。

返回：
与此文件输出流有关的文件通道
从以下版本开始：
1.4

---

finalize

```
protected void finalize()
                    throws IOException
```

清理到文件的连接，并确保在不再引用此文件输出流时调用此流的 close 方法。
覆盖：
类 Object 中的 finalize
抛出：
IOException - 如果发生 I/O 错误。

FilterInputStream 类

FilterInputStream 包含其他一些输入流，它将这些流用作其基本数据源，它可以直接传输数据或提供一些额外的功能。FilterInputStream 类本身只是简单地重写那些将所有请求传递给所包含输入流的 InputStream 的所有方法。FilterInputStream 的子类可进一步重写这些方法中的一些方法，并且还可以提供一些额外的方法和字段。

字段详细信息

in

```
protected volatile InputStream in
```

要过滤的输入流。

构造方法详细信息

FilterInputStream

```
protected FilterInputStream(InputStream in)
```

将参数 in 分配给字段 this.in，以便记住它供以后使用，通过这种方式创建一个 FilterInputStream。

参数：

in - 底层输入流，如果要在没有底层流的情况下创建此实例，则该参数为 null。

方法详细信息

read

```
public int read()
            throws IOException
```

从此输入流中读取下一个数据字节。返回一个 0 到 255 范围内的 int 字节值。如果因为已经到达流末尾而没有字节可用，则返回 -1。在输入数据可用、检测到流末尾或抛出异常之前，此方法将一直阻塞。

此方法只执行 in.read() 并返回结果。

指定者：

类 InputStream 中的 read

返回：

下一个数据字节；如果已到达流末尾，则返回 -1。

抛出：

IOException - 如果发生 I/O 错误。

另请参见：

in

read

```
public int read(byte[] b)
         throws IOException
```

从此输入流中将 byte.length 个字节的数据读入一个 byte 数组中。在某些输入可用之前，此方法将阻塞。

此方法只执行 read(b, 0, b.length) 调用并返回结果。注意到它*不* 执行 in.read(b) 很重要；FilterInputStream 的某些子类依赖于实际使用的实现策略。

覆盖：

类 InputStream 中的 read

参数：

b - 存储读取数据的缓冲区。

返回：

读入缓冲区的字节总数，如果因为已经到达流末尾而没有更多的数据，则返回 -1。

抛出：

IOException - 如果发生 I/O 错误。

另请参见：

read(byte[], int, int)

read

```
public int read(byte[] b,
                int off,

         int len)
```

throws IOException

从此输入流中将 len 个字节的数据读入一个 byte 数组中。如果 len 不为 0，则在输入可用前，此方法将阻塞；否则，不读取任何字节并且返回 0。

此方法只执行 in.read(b, off, len) 并返回结果。

覆盖：

类 InputStream 中的 read

参数：

b - 存储读取数据的缓冲区。

off - 目标数组 b 中的起始偏移量。

len - 读取的最大字节数。

返回：

读入缓冲区的字节总数，如果因为已经到达流末尾而没有更多的数据，则返回 -1。

抛出：

NullPointerException - 如果 b 为 null。

IndexOutOfBoundsException - 如果 off 为负、len 为负，或者 len 大于 b.length - off

IOException - 如果发生 I/O 错误。

另请参见：

in

skip

public long skip(long n)
throws IOException

跳过和丢弃此输入流中数据的 n 个字节。出于各种原因，skip 方法结束时跳过的字节数可能小于该数，也可能为 0。导致这种情况的原因很多，跳过 n 个字节之前已到达文件末尾只是其中一种可能。返回跳过的实际字节数。如果 n 为负，则不跳过任何字节。

此类的 skip 方法创建一个 byte 数组，然后重复将字节读入其中，直到读够 n 个字节或已到达流末尾为止。建议子类提供此方法更为有效的实现。例如，可依赖搜索能力的实现。

此方法只执行 in.skip(n)。

覆盖：
类 InputStream 中的 skip
参数：
n - 要跳过的字节数。
返回：
跳过的实际字节数。
抛出：
IOException - 如果流不支持搜索，或者发生其他 I/O 错误。

available

```
public int available()
                  throws IOException
```

返回下一次对此输入流调用的方法可以不受阻塞地从此输入流读取（或跳过）的估计剩余字节数。下一个调用者可能是同一个线程，也可能是另一个线程。一次读取或跳过此数量个字节不会发生阻塞，但读取或跳过的字节可能小于该数。

此方法返回 in.available() 的结果。

覆盖：
类 InputStream 中的 available
返回：
可以不受阻塞地从此输入流中读取（或跳过）的估计字节数。
抛出：
IOException - 如果发生 I/O 错误。

close

```
public void close()
                throws IOException
```

关闭此输入流并释放与此流关联的所有系统资源。此方法只执行 in.close()。
指定者：
接口 Closeable 中的 close
覆盖：

类 InputStream 中的 close

抛出：

IOException - 如果发生 I/O 错误。

另请参见：

in

mark

```
public void mark(int readlimit)
```

在输入流中的当前位置上作标记。reset 方法的后续调用将此流重新定位在最后标记的位置上，以便后续读取操作重新读取相同的字节。

readlimit 参数告知此输入流在标记位置无效之前允许读取的字节数。

此方法只执行 in.mark(readlimit)。

覆盖：

类 InputStream 中的 mark

参数：

readlimit - 在标记位置变得无效前可以读取字节的最大限制。

另请参见：

in,reset()

reset

```
public void reset()
                throws IOException
```

将此流重新定位到对此输入流最后调用 mark 方法时的位置。

此方法只执行 in.reset()。

在需要提前读取一小部分数据以查看流中有什么的情况下，可以使用流的标记。通过调用通用解析器常常最容易做到这一点。如果流属于通过解析处理的类型，那么解析起来就很容易。如果流不属于那种类型，那么解析

器应该在解析失败时抛出一个异常。如果这发生在 readlimit 个字节内，那么它允许外部代码重置流，并尝试另一种解析器。

覆盖：

类 InputStream 中的 reset

抛出：

IOException - 如果已经标记了该流，或者标记已经无效。

另请参见：

in,mark(int)

---

markSupported

```
public boolean markSupported()
```

测试此输入流是否支持 mark 和 reset 方法。此方法只执行 in.markSupported()。

覆盖：

类 InputStream 中的 markSupported

返回：

如果此流类型支持 mark 和 reset 方法，则返回 true；否则返回 false。

FilterOutputStream 类

此类是过滤输出流的所有类的超类。这些流位于已存在的输出流（基础 输出流）之上，它们将已存在的输出流作为其基本数据接收器，但可能直接传输数据或提供一些额外的功能。

FilterOutputStream 类本身只是简单地重写那些将所有请求传递给所包含输出流的 OutputStream 的所有方法。FilterOutputStream 的子类可进一步地重写这些方法中的一些方法，并且还可以提供一些额外的方法和字段。

字段详细信息

out

protected OutputStream out

要过滤的基础输出流。

构造方法详细信息

FilterOutputStream

public FilterOutputStream(OutputStream out)

创建一个构建在指定基础输出流之上的输出流过滤器。

参数：

out - 分配给字段 this.out 以便以后使用的基础输出流，如果将在没有底层流的情况下创建此实例，则该参数为 null。

方法详细信息

write

public void write(int b)
throws IOException

将指定 byte 写入此输出流。

FilterOutputStream 的 write 方法调用其基础输出流的 write 方法，也就是说，它执行 out.write(b)。

实现 OutputStream 的抽象 write 方法。

指定者：

类 OutputStream 中的 write

参数：

b - byte。

抛出：

IOException - 如果发生 I/O 错误。

write

```
public void write(byte[] b)
                  throws IOException
```

将 b.length 个字节写入此输出流。

FilterOutputStream 的 write 方法将 b、0 和 b.length 作为三个参数来调用 write 方法。

注意，此方法不调用其底层流的只带有单个参数 b 的 write 方法。

覆盖：
类 OutputStream 中的 write
参数：
b - 要写入的数据。
抛出：
IOException - 如果发生 I/O 错误。
另请参见：
write(byte[], int, int)

write

```
public void write(byte[] b,
                  int off,
                  int len)
                  throws IOException
```

将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此输出流。

FilterOutputStream 的 write 方法依次调用带一个参数的 write 方法来输出每个 byte。

注意，此方法不会调用其基础输入流的带有相同参数的 write 方法。FilterOutputStream 的子类应该提供此方法更有效的实现。

覆盖：
类 OutputStream 中的 write

参数：
b - 数据。
off - 数据中的起始偏移量。
len - 要写入的字节数。
抛出：
IOException - 如果发生 I/O 错误。
另请参见：
write(int)

flush

```
public void flush()
            throws IOException
```

刷新此输出流，并强制将所有已缓冲的输出字节写入该流中。

FilterOutputStream 的 flush 方法调用其基础输出流的 flush 方法。

指定者：
接口 Flushable 中的 flush
覆盖：
类 OutputStream 中的 flush
抛出：
IOException - 如果发生 I/O 错误。
另请参见：
out

close

```
public void close()
            throws IOException
```

关闭此输出流并释放与此流有关的所有系统资源。

FilterOutputStream 的 close 方法先调用其 flush 方法，然后调用其基础输出流的 close 方法。

指定者：
接口 Closeable 中的 close
覆盖：
类 OutputStream 中的 close
抛出：
IOException - 如果发生 I/O 错误。

LineNumberInputStream 类

此类是一个输入流过滤器，它提供跟踪当前行号的附加功能。

行是以回车符 ('\r')、换行符 ('\n') 或回车符后面紧跟换行符结尾的字节序列。在所有这三种情况下，都以单个换行符形式返回行终止字符。

行号以 0 开头，并在 read 返回换行符时递增 1。

构造方法详细信息

LineNumberInputStream

```
public LineNumberInputStream(InputStream in)
```

已过时。
构造从指定输入流读取其输入的新行号输入流。
参数：
in - 基础输入流。

方法详细信息

read

```
public int read()
            throws IOException
```

已过时。
从此输入流读取下一个数据字节。返回 0 到 255 范围内的 int 字节值。如果因流

的末尾已到达而没有可用的字节，则返回值 -1。在输入数据可用、检测到流的末尾或者抛出异常前，此方法一直阻塞。

LineNumberInputStream 的 read 方法调用基础输入流的 read 方法。它检查输入中的回车和换行符，并相应地修改当前行号。回车符或后跟换行符的回车两者都可转换为单个换行符。

覆盖：

类 FilterInputStream 中的 read

返回：

下一个数据字节，如果已到达此流的末尾，则返回 -1。

抛出：

IOException - 如果发生 I/O 错误。

另请参见：

FilterInputStream.in,getLineNumber()

---

read

```
public int read(byte[] b,
                int off,
                int len)
         throws IOException
```

已过时。

将最多 len 个数据字节从此输入流读入 byte 数组。在某个输入可用前，此方法一直阻塞。

LineNumberInputStream 的 read 方法重复调用 0 参数的 read 方法来填充 byte 数组。

覆盖：

类 FilterInputStream 中的 read

参数：

b - 读入数据的缓冲区。

off - 数据的初始偏移量。

len - 读取的最大字节数。

返回：

读入缓冲区的总字节数，如果由于已到达此流的末尾而不再有数据，则返回 -1。

抛出：

IOException - 如果发生 I/O 错误。

另请参见：

read()

skip

```
public long skip(long n)
          throws IOException
```

已过时。

跳过和放弃此输入流中的 n 个数据字节。出于各种原因，该 skip 方法跳过某些较小的字节数（可能是 0）后结束。返回跳过的实际字节数。如果 n 为负，则不跳过任何字节。

LineNumberInputStream 的 skip 方法创建 byte 数组，然后重复向其读入，直到读够 n 个字节或已到达流的末尾为止。

覆盖：

类 FilterInputStream 中的 skip

参数：

n - 要跳过的字节数。

返回：

跳过的实际字节数。

抛出：

IOException - 如果发生 I/O 错误。

另请参见：

FilterInputStream.in

setLineNumber

```
public void setLineNumber(int lineNumber)
```

已过时。

将行号设置为指定参数。

参数：
lineNumber - 新行号。
另请参见：
getLineNumber()

getLineNumber

```
public int getLineNumber()
```

已过时。
返回当前行号。
返回：
当前行号。
另请参见：
setLineNumber(int)

available

```
public int available()
                  throws IOException
```

已过时。
无阻塞情况下返回可以从此输入流中读取的字节数。

注意，如果基础输入流能够在无阻塞情况下提供 *k* 个输入字符，则 LineNumberInputStream 可以保证在无阻塞情况下只提供 *k*/2 个字符，因为基础输入流中的 *k* 个字符可能由 *k*/2 个 '\r' 和 '\n' 对组成，这些对只能转换为 *k*/2 '\n' 个字符。

覆盖：
类 FilterInputStream 中的 available
返回：
无阻塞情况下可以从此输入流读取的字节数。
抛出：
IOException - 如果发生 I/O 错误。
另请参见：

FilterInputStream.in

mark

```
public void mark(int readlimit)
```

已过时。

在此输入流中标记当前的位置。对 reset 方法的后续调用会在最后标记的位置重新定位此流，以使后续读取重新读取相同的字节。

LineNumberInputStream 的 mark 方法会记住 private 变量中的当前行号，然后调用基础输入流的 mark 方法。

覆盖：

类 FilterInputStream 中的 mark

参数：

readlimit - 在标记位置失效前可以读取字节的最大限制。

另请参见：

FilterInputStream.in,reset()

reset

```
public void reset()
                throws IOException
```

已过时。

将此流重新定位到对此输入流最后调用 mark 方法时的位置。

LineNumberInputStream 的 reset 方法会重新将行号设置为调用 mark 方法时的行号，然后调用基础输入流的 reset 方法。

在需要提前从流中读取少量数据以查看流内容的情况下，可以使用流标记。通常，调用某些常规解析器可以非常方便地完成这项工作。如果流属于解析器处理的类型，则会非常顺利地完成。如果流不属于该类型，则解析器应该在其失败时抛出异常，如果这是在 readlimit 字节内发生的，它将允许使用外部代码重新设置该流并尝试其他解析器。

覆盖：
类 FilterInputStream 中的 reset
抛出：
IOException - 如果发生 I/O 错误。

PipedInputStream 类

管道输入流应该连接到管道输出流；管道输入流提供要写入管道输出流的所有数据字节。通常，数据由某个线程从 PipedInputStream 对象读取，并由其他线程将其写入到相应的 PipedOutputStream。不建议对这两个对象尝试使用单个线程，因为这样可能死锁线程。管道输入流包含一个缓冲区，可在缓冲区限定的范围内将读操作和写操作分离开。如果向连接管道输出流提供数据字节的线程不再存在，则认为该管道已损坏。

字段详细信息

PIPE_SIZE

```
protected static final int PIPE_SIZE
```

管道循环输入缓冲区的默认大小。
从以下版本开始：
JDK1.1
另请参见：
常量字段值

---

buffer

```
protected byte[] buffer
```

放置传入数据的循环缓冲区。
从以下版本开始：
JDK1.1

---

in

```
protected int in
```

循环缓冲区中位置的索引，当从连接的管道输出流中接收到下一个数据字节时，会将其存储到该位置。in<0 意味着缓冲区为空，in==out 意味着缓冲区已满。

从以下版本开始：

JDK1.1

out

```
protected int out
```

循环缓冲区中位置的索引，此管道输入流将从该位置读取下一个数据字节。

从以下版本开始：

JDK1.1

构造方法详细信息

PipedInputStream

```
public PipedInputStream(PipedOutputStream src)
                        throws IOException
```

创建 PipedInputStream，使其连接到管道输出流 src。写入 src 的数据字节可用作此流的输入。

参数：

src - 要连接的流。

抛出：

IOException - 如果发生 I/O 错误。

PipedInputStream

```
public PipedInputStream(PipedOutputStream src,
```

int pipeSize)
throws IOException

创建一个 PipedInputStream，使其连接到管道输出流 src，并对管道缓冲区使用指定的管道大小。 写入 src 的数据字节可用作此流的输入。

参数：

src - 要连接的流。

pipeSize - 管道缓冲区的大小。

抛出：

IOException - 如果发生 I/O 错误。

IllegalArgumentException - 如果 pipeSize <= 0。

从以下版本开始：

1.6

PipedInputStream

public PipedInputStream()

创建尚未连接的 PipedInputStream。在使用前必须将其连接到 PipedOutputStream。

PipedInputStream

public PipedInputStream(int pipeSize)

创建一个尚未连接的 PipedInputStream，并对管道缓冲区使用指定的管道大小。在使用前必须将其连接到 PipedOutputStream。

参数：

pipeSize - 管道缓冲区的大小。

抛出：

IllegalArgumentException - 如果 pipeSize <= 0。

从以下版本开始：

1.6

方法详细信息

connect

```
public void connect(PipedOutputStream src)
                    throws IOException
```

使此管道输入流连接到管道输出流 src。如果此对象已经连接到其他某个管道输出流，则抛出 IOException。

如果 src 为未连接的管道输出流，snk 为未连接的管道输入流，则可以通过以下任一调用使其连接：

snk.connect(src)

或：

src.connect(snk)

这两个调用的效果相同。

参数：
src - 要连接的管道输出流。
抛出：
IOException - 如果发生 I/O 错误。

receive

```
protected void receive(int b)
                       throws IOException
```

接收数据字节。如果不存在可用的输入，此方法将发生阻塞。
参数：
b - 将接收的字节
抛出：
IOException - 如果管道损坏、未连接、关闭，或者发生 I/O 错误。
从以下版本开始：

JDK1.1

read

```
public int read()
            throws IOException
```

读取此管道输入流中的下一个数据字节。返回 0 到 255 范围内的 int 字节值。在输入数据可用、检测到流的末尾或者抛出异常前，此方法一直阻塞。

指定者：

类 InputStream 中的 read

返回：

下一个数据字节；如果已到达流末尾，则返回 -1。

抛出：

IOException - 如果管道未连接、损坏、关闭，或者发生 I/O 错误。

read

```
public int read(byte[] b,
                int off,
                int len)
         throws IOException
```

将最多 len 个数据字节从此管道输入流读入 byte 数组。如果已到达数据流的末尾，或者 len 超出管道缓冲区大小，则读取的字节数将少于 len。如果 len 为 0，则不读取任何字节并返回 0；否则，在至少 1 个输入字节可用、检测到流末尾、抛出异常前，该方法将一直阻塞。

覆盖：

类 InputStream 中的 read

参数：

b - 读入数据的缓冲区。

off - 目标数组 b 中的初始偏移量。

len - 读取的最多字节数。

返回：

读入缓冲区的总字节数；如果由于已到达流末尾而不再有数据，则返回 -1。

抛出：

NullPointerException - 如果 b 为 null。
IndexOutOfBoundsException - 如果 off 为负，len 为负，或者 len 大于 b.length - off
IOException - 如果管道损坏、未连接、关闭，或者发生 I/O 错误。
另请参见：
InputStream.read()

---

available

```
public int available()
                  throws IOException
```

返回可以不受阻塞地从此输入流中读取的字节数。
覆盖：
类 InputStream 中的 available
返回：
可以不受阻塞地从此输入流读取的字节数;如果已经调用 close() 方法关闭此输入流、管道未连接或已损坏，则返回 0。
抛出：
IOException - 如果发生 I/O 错误。
从以下版本开始：
JDK1.0.2

---

close

```
public void close()
                throws IOException
```

关闭此管道输入流并释放与该流相关的所有系统资源。
指定者：
接口 Closeable 中的 close
覆盖：
类 InputStream 中的 close
抛出：
IOException - 如果发生 I/O 错误。

PipedOutputStream 类

可以将管道输出流连接到管道输入流来创建通信管道。管道输出流是管道的发送端。通常，数据由某个线程写入 PipedOutputStream 对象，并由其他线程从连接的 PipedInputStream 读取。不建议对这两个对象尝试使用单个线程，因为这样可能会造成该线程死锁。如果某个线程正从连接的管道输入流中读取数据字节，但该线程不再处于活动状态，则该管道被视为处于 毁坏 状态。

构造方法详细信息

PipedOutputStream

```
public PipedOutputStream(PipedInputStream snk)
                  throws IOException
```

创建连接到指定管道输入流的管道输出流。写入此流的数据字节稍后将用作 snk 的输入。

参数：

snk - 要连接的管道输入流。

抛出：

IOException - 如果发生 I/O 错误。

PipedOutputStream

```
public PipedOutputStream()
```

创建尚未连接到管道输入流的管道输出流。必须在使用之前将管道输出流连接到管道输入流（既可由接收者连接，也可由发送者连接）。

方法详细信息

connect

```
public void connect(PipedInputStream snk)
                    throws IOException
```

将此管道输出流连接到接收者。如果此对象已经连接到其他某个管道输入流，则抛出 IOException。

如果 snk 是未连接的管道输入流，而 src 是未连接的管道输出流，则可以通过以下调用之一连接它们：

```
src.connect(snk)
```

或：

```
snk.connect(src)
```

这两个调用的效果相同。

参数：

snk - 要连接的管道输入流。

抛出：

IOException - 如果发生 I/O 错误。

---

write

```
public void write(int b)
                  throws IOException
```

将指定 byte 写入传送的输出流。

实现 OutputStream 的 write 方法。

指定者：

类 OutputStream 中的 write

参数：

b - 要写入的 byte。

抛出：

IOException - 如果管道处于毁坏或未连接状态，或者发生 I/O 错误。

---

write

```
public void write(byte[] b,
                  int off,
                  int len)
           throws IOException
```

将 len 字节从初始偏移量为 off 的指定 byte 数组写入该管道输出流。在将所有字节写入输出流之前，此方法一直处于阻塞状态。

覆盖：

类 OutputStream 中的 write

参数：

b - 数据。

off - 数据中的初始偏移量。

len - 要写入的字节数。

抛出：

IOException - 如果管道处于 毁坏或未连接状态，或者发生 I/O 错误。

---

flush

```
public void flush()
           throws IOException
```

刷新此输出流并强制写出所有缓冲的输出字节。这将通知所有 reader，告知它们管道中的字符处于等待中。

指定者：

接口 Flushable 中的 flush

覆盖：

类 OutputStream 中的 flush

抛出：

IOException - 如果发生 I/O 错误。

---

close

```
public void close()
           throws IOException
```

关闭此管道输出流并释放与此流有关的所有系统资源。此流可能无法再用于写入字节。

指定者：
接口 Closeable 中的 close
覆盖：
类 OutputStream 中的 close
抛出：
IOException - 如果发生 I/O 错误。

PrintStream 类

PrintStream 为其他输出流添加了功能，使它们能够方便地打印各种数据值表示形式。它还提供其他两项功能。与其他输出流不同，PrintStream 永远不会抛出 IOException；而是，异常情况仅设置可通过 checkError 方法测试的内部标志。另外，为了自动刷新，可以创建一个 PrintStream；这意味着可在写入 byte 数组之后自动调用 flush 方法，可调用其中一个 println 方法，或写入一个换行符或字节 ('\n')。

PrintStream 打印的所有字符都使用平台的默认字符编码转换为字节。在需要写入字符而不是写入字节的情况下，应该使用 PrintWriter 类。

构造方法详细信息

PrintStream

```
public PrintStream(OutputStream out)
```

创建新的打印流。此流将不会自动刷新。
参数：
out - 将向其打印值和对象的输出流
另请参见：
PrintWriter.PrintWriter(java.io.OutputStream)

PrintStream

```
public PrintStream(OutputStream out,
                   boolean autoFlush)
```

创建新的打印流。

参数：

out - 将向其打印值和对象的输出流

autoFlush - boolean 变量；如果为 true，则每当写入 byte 数组、调用其中一个 println 方法或写入换行符或字节 ('\n') 时都会刷新输出缓冲区

另请参见：

PrintWriter.PrintWriter(java.io.OutputStream, boolean)

PrintStream

```
public PrintStream(OutputStream out,
                   boolean autoFlush,
                   String encoding)
            throws UnsupportedEncodingException
```

创建新的打印流。

参数：

out - 将向其打印值和对象的输出流

autoFlush - boolean 变量；如果为 true，则每当写入 byte 数组、调用其中一个 println 方法或写入换行符或字节 ('\n') 时都会刷新缓冲区

encoding - 受支持的字符编码的名称

抛出：

UnsupportedEncodingException - 如果不支持指定的编码

从以下版本开始：

1.4

PrintStream

```
public PrintStream(String fileName)
            throws FileNotFoundException
```

创建具有指定文件名称且不带自动行刷新的新打印流。此便捷构造方法创建必要的中间 OutputStreamWriter，后者将使用此 Java 虚拟机实例的默认 charset 进行字符编码。

参数：

fileName - 要用作此打印流目标的文件名称。如果存在该文件，则将其大小截取为

零；否则，创建一个新文件。将输出写入文件中，并对其进行缓冲处理。

抛出：

FileNotFoundException - 如果给定的文件对象不表示现有的可写常规文件，并且无法创建该名称的新常规文件，或者在打开或创建文件时发生其他一些错误

SecurityException - 如果存在安全管理器，并且 checkWrite(fileName) 拒绝对文件进行写入访问。

从以下版本开始：

1.5

PrintStream

```
public PrintStream(String fileName,
                   String csn)
            throws FileNotFoundException,
                   UnsupportedEncodingException
```

创建具有指定文件名称和字符集且不带自动行刷新的新打印流。此便捷构造方法创建必要的中间 OutputStreamWriter，后者将使用提供的字符集进行字符编码。

参数：

fileName - 要用作此打印流目标的文件名称。如果存在该文件，则将其大小截取为零；否则，创建一个新文件。将输出写入文件中，并对其进行缓冲处理。

csn - 受支持的字符集的名称

抛出：

FileNotFoundException - 如果给定的文件对象不表示现有的可写常规文件，并且无法创建该名称的新常规文件，或者在打开或创建文件时发生其他一些错误

SecurityException - 如果存在安全管理器，并且 checkWrite(fileName) 拒绝对文件进行写入访问。

UnsupportedEncodingException - 如果不支持指定字符集

从以下版本开始：

1.5

PrintStream

```
public PrintStream(File file)
            throws FileNotFoundException
```

创建具有指定文件且不带自动行刷新的新打印流。此便捷构造方法创建必要的中间 OutputStreamWriter，后者将使用此 Java 虚拟机实例的默认 charset 进行字符编码。

参数：

file - 要用作此打印流目标的文件。如果存在该文件，则将其大小截取为零；否则，创建一个新文件。将输出写入文件中，并对其进行缓冲处理。

抛出：

FileNotFoundException - 如果给定的文件对象不表示现有的可写常规文件，并且无法创建该名称的新常规文件，或者在打开或创建文件时发生其他一些错误

SecurityException - 如果存在安全管理器，并且 checkWrite(file.getPath()) 拒绝对文件进行写入访问。

从以下版本开始：

1.5

PrintStream

```
public PrintStream(File file,
                   String csn)
            throws FileNotFoundException,
                   UnsupportedEncodingException
```

创建具有指定文件名称和字符集且不带自动行刷新的新打印流。此便捷构造方法创建必要的中间 OutputStreamWriter，后者将使用提供的字符集进行字符编码。

参数：

file - 要用作此打印流目标的文件。如果存在该文件，则将其大小截取为零；否则，创建一个新文件。将输出写入文件中，并对其进行缓冲处理。

csn - 受支持的字符集的名称

抛出：

FileNotFoundException - 如果给定的文件对象不表示现有的可写常规文件，并且无法创建该名称的新常规文件，或者在打开或创建文件时发生其他一些错误

SecurityException - 如果存在安全管理器，并且 checkWrite(file.getPath()) 拒绝对文件进行写入访问

UnsupportedEncodingException - 如果不支持指定字符集

从以下版本开始：

1.5

方法详细信息

flush

```
public void flush()
```

刷新该流的缓冲。通过将所有缓冲的输出字节写入到底层输出流然后刷新该流的缓冲，完成此操作。

指定者：

接口 Flushable 中的 flush

覆盖：

类 FilterOutputStream 中的 flush

另请参见：

OutputStream.flush()

---

close

```
public void close()
```

关闭流。通过刷新流然后关闭底层输出流，完成此操作。

指定者：

接口 Closeable 中的 close

覆盖：

类 FilterOutputStream 中的 close

另请参见：

OutputStream.close()

---

checkError

```
public boolean checkError()
```

刷新流并检查其错误状态。当底层输出流抛出 IOException 而不是 InterruptedIOException 时，以及调用 setError 方法时，内部错误状态将被设置为 true。如果对底层输出流的操作抛出 InterruptedIOException，则通过调用以下方法或等效方法，PrintStream 将该异常转换回中断状态：

```
Thread.currentThread().interrupt();
```

返回：

当且仅当此流遇到 IOException 而不是 InterruptedIOException 时，或已调用 setError 方法时，返回 true。

---

setError

```
protected void setError()
```

将该流的错误状态设置为 true。

在调用 clearError() 之前，此方法将导致 checkError() 的后续调用返回 true。

从以下版本开始：

JDK1.1

---

clearError

```
protected void clearError()
```

清除此流的内部错误状态。

在另一个写入操作失败并调用 setError() 之前，此方法将导致 checkError() 的后续调用返回 false。

从以下版本开始：

1.6

---

write

```
public void write(int b)
```

将指定的字节写入此流。如果字节为新行且启用了自动刷新，则调用 flush 方法。

要注意按给定写入字节；要写入将根据平台的默认字符编码转换的字符，请使用 print(char) 或 println(char) 方法。

覆盖：
类 FilterOutputStream 中的 write
参数：
b - 要写入的字节
另请参见：
print(char),println(char)

write

```
public void write(byte[] buf,
                  int off,
                  int len)
```

将 len 字节从指定的初始偏移量为 off 的 byte 数组写入此流。如果启用自动刷新，则调用 flush 方法。

注意，要按给定写入字节；要写入将根据平台的默认字符编码转换的字符，请使用 print(char) 或 println(char) 方法。

覆盖：
类 FilterOutputStream 中的 write
参数：
buf - byte 数组
off - 相对于开始写入字节处的偏移量
len - 要写入的字节数
另请参见：
FilterOutputStream.write(int)

print

```
public void print(boolean b)
```

打印 boolean 值。按照平台的默认字符编码将 String.valueOf(boolean) 生成

的字符串转换为字节，并完全以 write(int) 方法的方式写入这些字节。

参数：

b - 要打印的 boolean

---

print

```
public void print(char c)
```

打印字符。按照平台的默认字符编码将字符转换为一个或多个字节，并完全以 write(int) 方法的方式写入这些字节。

参数：

c - 要打印的 char

---

print

```
public void print(int i)
```

打印整数。按照平台的默认字节编码将 String.valueOf(int) 生成的字符串转换为字节，并完全以 write(int) 方法的方式写入这些字节。

参数：

i - 要打印的 int

另请参见：

Integer.toString(int)

---

print

```
public void print(long l)
```

打印 long 整数。按照平台的默认字符编码将 String.valueOf(long) 生成的字符串转换为字节，并完全以 write(int) 方法的方式写入这些字节。

参数：

l - 要打印的 long

另请参见：

Long.toString(long)

---

print

public void print(float f)

打印浮点数。按照平台的默认字符编码将 String.valueOf(float) 生成的字符串转换为字节，并完全以 write(int) 方法的方式写入这些字节。

参数：

f - 要打印的 float

另请参见：

Float.toString(float)

print

public void print(double d)

打印双精度浮点数。按照平台的默认字符编码将 String.valueOf(double) 生成的字符串转换为字节，并完全以 write(int) 方法的方式写入这些字节。

参数：

d - 要打印的 double

另请参见：

Double.toString(double)

print

public void print(char[] s)

打印字符数组。按照平台的默认字符编码将字符转换为字节，并完全以 write(int) 方法的方式写入这些字节。

参数：

s - 要打印的字符数组

抛出：

NullPointerException - 如果 s 为 null

print

public void print(String s)

打印字符串。如果参数为 null，则打印字符串 "null"。否则，按照平台的默认字符编码将字符串的字符转换为字节，并完全以 write(int) 方法的方式写入这些字节。

参数：

s - 要打印的 String

---

print

public void print(Object obj)

打印对象。按照平台的默认字符串编码将 String.valueOf(Object) 方法生成的字符串转换为字节，并完全以 write(int) 方法的方式写入这些字节。

参数：

obj - 要打印的 Object

另请参见：

Object.toString()

---

println

public void println()

通过写入行分隔符字符串终止当前行。行分隔符字符串由系统属性 line.separator 定义，不一定是单个换行符 ('\n')。

---

println

public void println(boolean x)

打印 boolean 值，然后终止行。此方法的行为就像先调用 print(boolean) 然后调用 println() 一样。

参数：

x - 要打印的 boolean

---

println

public void println(char x)

打印字符，然后终止该行。此方法的行为就像先调用 print(char) 然后调用 println() 一样。

参数：

x - 要打印的 char。

println

public void println(int x)

打印整数，然后终止该行。此方法的行为就像先调用 print(int) 然后调用 println() 一样。

参数：

x - 要打印的 int。

println

public void println(long x)

打印 long，然后终止该行。此方法的行为就像先调用 print(long) 然后调用 println() 一样。

参数：

x - 要打印的 long。

println

public void println(float x)

打印 float，然后终止该行。此方法的行为就像先调用 print(float) 然后调用 println() 一样。

参数：

x - 要打印的 float。

println

public void println(double x)

打印 double，然后终止该行。此方法的行为就像先调用 print(double) 然后调用 println() 一样。

参数：

x - 要打印的 double。

println

public void println(char[] x)

打印字符数组，然后终止该行。此方法的行为就像先调用 print(char[]) 然后调用 println() 一样。

参数：

x - 要打印的字符数组。

println

public void println(String x)

打印 String，然后终止该行。此方法的行为就像先调用 print(String) 然后调用 println() 一样。

参数：

x - 要打印的 String。

println

public void println(Object x)

打印 Object，然后终止该行。此方法首先调用 String.valueOf(x) 获取打印对象的字符串值，然后的行为如同先调用 print(String) 再调用 println() 一样。

参数：

x - 要打印的 Object。

printf

```
public PrintStream printf(String format,
                          Object... args)
```

使用指定格式字符串和参数将格式化的字符串写入此输出流的便捷方法。

调用此方法的 out.printf(format, args) 形式，行为与以下调用完全相同：

```
out.format(format, args)
```

参数：

format - 在格式字符串的语法中描述的格式字符串

args - 格式字符串中的格式说明符引用的参数。如果参数多于格式说明符，则忽略额外的参数。参数的数量是可变的，并且可以为零。参数的最大数量受到 Java Virtual Machine Specification 定义的 Java 数组的最大维数的限制。针对 null 参数的行为依赖于 conversion。

返回：

此输出流

抛出：

IllegalFormatException - 如果格式字符串包含非法语法、与给定参数不兼容的格式说明符、对给定格式字符串而言不够充足的参数或其他非法条件。有关所有可能的格式错误的规范，请参阅 formatter 类规范的详细信息部分。

NullPointerException - 如果 format 为 null

从以下版本开始：

1.5

printf

```
public PrintStream printf(Locale l,
                          String format,
                          Object... args)
```

使用指定格式字符串和参数将格式化的字符串写入此输出流的便捷方法。

调用此方法的 out.printf(l, format, args) 形式，行为与以下调用完全相同：

```
out.format(l, format, args)
```

参数：

l - 格式化过程中应用的 locale。如果 l 为 null，则不应用本地化。

format - 在格式字符串的语法中描述的格式字符串

args - 格式字符串中的格式说明符引用的参数。如果参数多于格式说明符，则忽略额外的参数。参数的数量是可变的，并且可以为零。参数的最大数量受到 Java Virtual Machine Specification 定义的 Java 数组的最大维数的限制。针对 null 参数的行为依赖于 conversion。

返回：

此输出流

抛出：

IllegalFormatException - 如果格式字符串包含非法语法、与给定参数不兼容的格式说明符、对给定格式字符串而言不够充足的参数或其他非法条件。有关所有可能的格式错误的规范，请参阅 formatter 类规范的详细信息部分。

NullPointerException - 如果 format 为 null

从以下版本开始：

1.5

---

format

```
public PrintStream format(String format,
                          Object... args)
```

使用指定格式字符串和参数将格式化字符串写入此输出流中。

始终使用的语言环境是由 Locale.getDefault() 返回的语言环境，不管以前在此对象上调用了其他什么样的格式化方法。

参数：

format - 在格式字符串的语法中描述的格式字符串

args - 格式字符串中的格式说明符引用的参数。如果参数多于格式说明符，则忽略额外的参数。参数的数量是可变的，并且可以为零。参数的最大数量受到 Java Virtual

Machine Specification 定义的 Java 数组的最大维数的限制。针对 null 参数的行为依赖于 conversion。

返回：

此输出流

抛出：

IllegalFormatException - 如果格式字符串包含非法语法、与给定参数不兼容的格式说明符、对给定格式字符串而言不够充足的参数或其他非法条件。有关所有可能的格式错误的规范，请参阅 formatter 类规范的详细信息部分。

NullPointerException - 如果 format 为 null

从以下版本开始：

1.5

---

format

```
public PrintStream format(Locale l,
                          String format,
                          Object... args)
```

使用指定格式字符串和参数将格式化字符串写入此输出流中。

参数：

l - 格式化过程中应用的 locale。如果 l 为 null，则不应用本地化。

format - 在格式字符串的语法中描述的格式字符串

args - 格式字符串中的格式说明符引用的参数。如果参数多于格式说明符，则忽略额外的参数。参数的数量是可变的，并且可以为零。参数的最大数量受到 Java Virtual Machine Specification 定义的 Java 数组的最大维数的限制。针对 null 参数的行为依赖于 conversion。

返回：

此输出流

抛出：

IllegalFormatException - 如果格式字符串包含非法语法、与给定参数不兼容的格式说明符、对给定格式字符串而言不够充足的参数或其他非法条件。有关所有可能的格式错误的规范，请参阅 formatter 类规范的详细信息部分。

NullPointerException - 如果 format 为 null

从以下版本开始：

1.5

---

append

```
public PrintStream append(CharSequence csq)
```

将指定字符序列添加到此输出流。

此方法调用 out.append(csq) 的行为与调用下列方法完全相同：

```
out.print(csq.toString())
```

可能不添加整个序列，也可能添加，具体取决于字符序列 csq 的 toString 规范。例如，调用一个字符缓冲区的 toString 方法将返回一个子序列，其内容取决于缓冲区的位置和限制。

指定者：

接口 Appendable 中的 append

参数：

csq - 要添加的字符序列。如果 csq 为 null，则向此输出流添加四个字符"null"。

返回：

此输出流

从以下版本开始：

1.5

---

append

```
public PrintStream append(CharSequence csq,
                          int start,
                          int end)
```

将指定字符序列的子序列添加到此输出流。

当 csq 不为 null 时，调用该方法的 out.append(csq、start、end) 形式，行为与以下调用完全相同：

```
out.print(csq.subSequence(start, end).toString())
```

指定者：

接口 Appendable 中的 append

参数：

csq - 要添加子序列的字符序列。如果 csq 为 null，则添加这些字符，就好像 csq 包含四个字符 "null" 一样。

start - 子序列中第一个字符的索引

end - 子序列中最后一个字符后面的字符的索引

返回：

此输出流

抛出：

IndexOutOfBoundsException - 如果 start 或 end 为负，而 start 大于 end 或者 end 大于 csq.length()

从以下版本开始：

1.5

---

append

```
public PrintStream append(char c)
```

将指定字符添加到此输出流。

调用此方法的 out.append(c) 形式，行为与以下调用完全相同：

```
out.print(c)
```

指定者：

接口 Appendable 中的 append

参数：

c - 要添加的 16 位字符

返回：

此输出流

java.awt

包含用于创建用户界面和绘制图形图像的所有类。

接口

ActiveEvent 接口

知道如何对自身进行指派的事件的接口。通过实现此接口，可以使用 EventDispatchThread 将一个事件放置到事件队列，并且指派该事件时将调用其 dispatch() 方法。

这是一种非常有用的避免死锁的机制。如果线程正在执行某个关键部分（即它已经进入了一个或多个监视器），调用其他同步代码可能导致死锁。为了避免潜在的死锁，可以创建一个 ActiveEvent，以便以后运行代码的第二部分。如果存在监视器争用，那么在第一个线程已经完成工作并退出监视器之前，第二个线程将一直处于阻塞状态。

出于安全性考虑，使用 ActiveEvent 来避免从一个关键线程中调用不受信任的代码通常是很值得的。例如，同位体实现可以使用此设施来避免从系统线程调用用户代码。这样做可以避免潜在的死锁和拒绝服务攻击。

方法详细信息

dispatch

```
void dispatch()
```

将事件指派给目标、事件源侦听器，或者做任何希望此事件去做的事情。

Adjustable 接口

此接口用于那些具有可调整数值的对象，数值应包含在有限范围的值之内。

字段详细信息

HORIZONTAL

static final int HORIZONTAL

指示 Adjustable 具有水平方向。

另请参见：

常量字段值

VERTICAL

static final int VERTICAL

指示 Adjustable 具有垂直方向。

另请参见：

常量字段值

NO_ORIENTATION

static final int NO_ORIENTATION

指示 Adjustable 不具有方向。

另请参见：

常量字段值

方法详细信息

getOrientation

int getOrientation()

获取此可调整对象的方向。

返回：

可调整对象的方向；可以是 HORIZONTAL、VERTICAL 或者 NO_ORIENTATION

setMinimum

void setMinimum(int min)

设置可调整对象的最小值。

参数：

min - 最小值

getMinimum

int getMinimum()

获取可调整对象的最小值。

返回：

可调整对象的最小值

setMaximum

void setMaximum(int max)

设置可调整对象的最大值。

参数：

max - 最大值

getMaximum

int getMaximum()

获取可调整对象的最大值。

返回：

可调整对象的最大值

setUnitIncrement

void setUnitIncrement(int u)

设置可调整对象的单位值增量。

参数：

u - 单位增量

getUnitIncrement

int getUnitIncrement()

获取可调整对象的单位值增量。

返回：

可调整对象的单位值增量

setBlockIncrement

void setBlockIncrement(int b)

设置可调整对象的块值增量。

参数：

b - 块值增量

getBlockIncrement

int getBlockIncrement()

获取可调整对象的块值增量。

返回：

可调整对象的块值增量

setVisibleAmount

void setVisibleAmount(int v)

设置可调整对象的比例指示器的长度。

参数：

v - 指示器的长度

getVisibleAmount

int getVisibleAmount()

获取比例指示器的长度。

返回：

比例指示器的长度

setValue

void setValue(int v)

设置可调整对象的当前值。如果提供的值小于 minimum 或者大于 maximum - visibleAmount，则适当地替换其中一个值。

调用此方法不会激发 AdjustmentEvent。

参数：

v - 当前值，它位于 minimum 和 maximum - visibleAmount 之间

getValue

int getValue()

获取可调整对象的当前值。

返回：

可调整对象的当前值

addAdjustmentListener

void addAdjustmentListener(AdjustmentListener l)

添加侦听器，以便在可调整对象的值更改时接收调整事件。

参数：

l - 接受事件的侦听器

另请参见：

AdjustmentEvent

removeAdjustmentListener

void removeAdjustmentListener(AdjustmentListener l)

移除一个调整侦听器。

参数：

l - 要移除的侦听器

Composite 接口

Composite 接口和 CompositeContext 一起定义了用基础图形区域组合绘图基本元素的方法。在 Graphics2D 上下文中设置 Composite 后，此接口组合形状、文本或图像，这些内容已经用根据预定义的规则所呈现的颜色进行了呈现。实现此接口的类提供规则和方法来创建特定操作的上下文。CompositeContext 是复合操作所使用的环境，由 Graphics2D 在操作开始前创建。CompositeContext 包含复合操作所需的私有信息和资源。当不再需要 CompositeContext 时，Graphics2D 对象会移除它，以便回收为操作所分配的资源。

实现 Composite 的类实例必须是不可变的，因为将这些对象作为 setComposite 方法的属性来设置时，或者 Graphics2D 对象被复制时，Graphics2D 并不复制这些对象。这是为了避免 Graphics2D 未定义的呈现行为，这种行为是在 Graphics2D 上下文中设置 Composite

对象后再对其进行修改所导致的。

由于此接口必须向可能的任意代码公开目标设备或图像上的像素内容，所以在直接向屏幕设备呈现内容时，由 readDisplayPixels AWTPermission 负责管理那些实现此接口的自定义对象的使用。将这样的自定义对象传递给 Graphics2D（从 Component 检索）的 setComposite 方法时会进行权限检查。

方法详细信息

createContext

```
CompositeContext createContext(ColorModel srcColorModel,
                               ColorModel dstColorModel,
                               RenderingHints hints)
```

创建一个上下文，其中包含用于执行复合操作的状态。在多线程环境中，单个 Composite 对象可同时有多个上下文。

参数：

srcColorModel - 源的 ColorModel

dstColorModel - 目标的 ColorModel

hints - 上下文对象用于选择呈现方法的提示

返回：

用于执行复合操作的 CompositeContext 对象。

CompositeContext 接口

CompositeContext 接口为复合操作定义了已封装和已优化的环境。CompositeContext 对象维护复合操作的状态。在多线程环境中，单个 Composite 对象可同时有多个上下文。

方法详细信息

dispose

```
void dispose()
```

释放分配给上下文的资源。

compose

```
void compose(Raster src,
             Raster dstIn,
             WritableRaster dstOut)
```

组合两个源 Raster 对象并将结果置于目标 WritableRaster 中。注意，目标可以是第一个或第二个源对象。注意，dstIn 和 dstOut 必须与传递给 Composite 接口的 (java.awt.image.ColorModel、java.awt.image.ColorModel、java.awt.RenderingHints) createContext 方法的 dstColorModel 兼容。

参数：

src - 复合操作的第一个源

dstIn - 复合操作的第二个源

dstOut - 存储操作结果的 WritableRaster

ItemSelectable 接口

包含零个或多个可选则项的集合的对象接口。

方法详细信息

getSelectedObjects

```
Object[] getSelectedObjects()
```

返回选定项；如果没有选定的项，则返回 null。

addItemListener

```
void addItemListener(ItemListener l)
```

添加侦听器，以接收用户更改项状态时的项事件。以编程方式设置项状态时，不发送项事件。如果 l 为 null，则不抛出异常也不执行操作。

参数：

l - 接收事件的侦听器

另请参见：

ItemEvent

removeItemListener

```
void removeItemListener(ItemListener l)
```

移除项侦听器。如果 l 为 null，则不抛出异常也不执行操作。

参数：

l - 正在移除的侦听器

KeyEventDispatcher 接口

在指派所有 KeyEvent 和为其确定目标时，KeyEventDispatcher 和当前 KeyboardFocusManager 一起使用。在当前 KeyboardFocusManager 中注册的 KeyEventDispatcher 在 KeyEvent 被指派到目标前接收它们，允许每个 KeyEventDispatcher 重定向事件，使用它，指派事件本身，或者进行其他更改。

注意，KeyboardFocusManager 本身即可实现 KeyEventDispatcher。默认情况下，当前 KeyboardFocusManager 是已注册 KeyEventDispatcher 未指派的所有 KeyEvent 的接受器。无法将当前 KeyboardFocusManager 作为 KeyEventDispatcher 完全注销。但是，如果 KeyEventDispatcher 报告指派了 KeyEvent，则不管是否实际指派了，KeyboardFocusManager 都将不对 KeyEvent 进行进一步的操作。（尽管客户端代码将当前的 KeyboardFocusManager 一次或多次注册为 KeyEventDispatcher 是可能的，但通常不必要也不建议这样做。）

方法详细信息

dispatchKeyEvent

```
boolean dispatchKeyEvent(KeyEvent e)
```

当前 KeyboardFocusManager 调用此方法，请求此 KeyEventDispatcher 为其指派指定的事件。此 KeyEventDispatcher 可以随意重定向事件，使用它，指派它本身，或者进行其他更改。通常使用此功能将 KeyEvent 传递给 Component 而不是焦点所有者。例如，可以使用此功能在可访问环境中导航不可聚焦 Window 的子级。注意，如果 KeyEventDispatcher 指派 KeyEvent 本身，则其必须使用 redispatchEvent 来防止当前 KeyboardFocusManager 递归式请求此 KeyEventDispatcher 再次指派事件。

如果此方法的实现返回 false，则将 KeyEvent 传入链中的下一个 KeyEventDispatcher，以当前的 KeyboardFocusManager 结尾。如果实现返回 true，则假定已经指派了该 KeyEvent（尽管事实不一定如此），当前 KeyboardFocusManager 将不对该 KeyEvent 进行任何进一步的操作。在此情况下，KeyboardFocusManager.dispatchEvent 也应该返回 true。如果实现使用 KeyEvent，但是返回 false，则仍将所使用的事件传入链中的下一个 KeyEventDispatcher。开发人员在将 KeyEvent 指派到目标前首先检查它是否已使用至关重要。默认情况下，当前 KeyboardFocusManager 不指派使用的 KeyEvent。

参数：

e - 要指派的 KeyEvent

返回：

如果 KeyboardFocusManager 不应对该 KeyEvent 进行进一步操作，则返回 true；否则，返回 false

KeyEventPostProcessor 接口

在最终解析所有未使用的 KeyEvent 时，需要将 KeyEventPostProcessor 和当前的 KeyboardFocusManager 一起使用。在当前 KeyboardFocusManager 中注册的 KeyEventPostProcessor 在 KeyEvent 被指派到目标并由目标处理后接收它们。因为目前应用程序中不存在拥有该焦点的 Component，所以也可以将这些 KeyEvent 转发到已注册的 KeyEventPostProcessor 中，否则它们将被丢弃。这允许应用程序实现需要全局 KeyEvent 后处理的功能，如菜单快捷键。

注意，KeyboardFocusManager 本身实现了 KeyEventPostProcessor。默认情况下，当前的 KeyboardFocusManager 将是链中的最终 KeyEventPostProcessor。无法将当前 KeyboardFocusManager 作为 KeyEventPostProcessor 完全注销。但是，如果 KeyEventPostProcessor 报告不应进一步后处理 KeyEvent，则 AWT 将认为已完全处理了该事件，不再对它执行任何其他操作。(尽管客户端代码将当前的 KeyboardFocusManager 一次

或多次注册为 KeyEventPostProcessor 是可能的，但通常不必要也不建议这样做。）

方法详细信息

postProcessKeyEvent

boolean postProcessKeyEvent(KeyEvent e)

当前 KeyboardFocusManager 调用此方法，请求此 KeyEventPostProcessor 执行应该作为 KeyEvent 最终解析的一部分的所需后处理。在调用此方法时，通常已经将该 KeyEvent 指派到其目标并由其目标进行了处理。但是，如果目前应用程序中任何 Component 均不拥有该焦点，则表示尚未将该 KeyEvent 指派给任何 Component。通常情况下，KeyEvent 后处理用于实现需要全局 KeyEvent 后处理的功能，如菜单快捷键。注意，如果 KeyEventPostProcessor 希望指派 KeyEvent，则其必须使用 redispatchEvent 来防止 AWT 递归式请求此 KeyEventPostProcessor 再次后处理该事件。

如果此方法的实现返回 false，则将 KeyEvent 传入链中的下一个 KeyEventPostProcessor，以当前的 KeyboardFocusManager 结尾。如果实现返回 true，则假定已经完全处理了该 KeyEvent（尽管事实不一定如此），AWT 将不对该 KeyEvent 进行任何进一步的操作。如果实现使用了 KeyEvent，但是返回 false，则仍将所使用的事件传入链中的下一个 KeyEventPostProcessor。开发人员在后处理 KeyEvent 之前检查是否已经使用了该 KeyEvent 至关重要。默认情况下，当前 KeyboardFocusManager 不执行任何响应已使用 KeyEvent 的后处理操作。

参数：

e - 要后处理的 KeyEvent

返回：

如果 AWT 不应该对该 KeyEvent 执行进一步操作，则返回 true；否则返回 false

LayoutManager 接口

定义知道如何布置 Container 类的接口。

Swing 的绘制架构假定 JComponent 的子组件不发生重叠。如果 JComponent 的 LayoutManager 允许子组件重叠，则 JComponent 必须重写 isOptimizedDrawingEnabled 以返回 false。

方法详细信息

createContext

```
CompositeContext createContext(ColorModel srcColorModel,
                               ColorModel dstColorModel,
                               RenderingHints hints)
```

创建一个上下文，其中包含用于执行复合操作的状态。在多线程环境中，单个 Composite 对象可同时有多个上下文。

参数：

srcColorModel - 源的 ColorModel

dstColorModel - 目标的 ColorModel

hints - 上下文对象用于选择呈现方法的提示

返回：

用于执行复合操作的 CompositeContext 对象。

LayoutManager2 接口

为类定义接口，该类知道如何根据布局约束对象来布置 Container。 此接口扩展了 LayoutManager 接口，以根据约束对象显式地处理布局，该约束对象指定应该如何以及在何处将组件添加到布局。

LayoutManager 的这一最小扩展可供想要创建基于约束的布局的工具提供者使用。但是，它不对基于约束的自定义布局管理器提供完整的通用支持。

方法详细信息

addLayoutComponent

void addLayoutComponent(Component comp,
Object constraints)

使用指定约束对象，将指定组件添加到布局。

参数：

comp - 要添加的组件

constraints - 如何以及在何处将组件添加到布局。

maximumLayoutSize

Dimension maximumLayoutSize(Container target)

给定指定容器的组件，计算该容器的最大大小维数。

另请参见：

Component.getMaximumSize(),LayoutManager

getLayoutAlignmentX

float getLayoutAlignmentX(Container target)

返回沿 X 轴的对齐方式。它指定如何相对于其他组件对齐该组件。值应该是一个介于 0 和 1 之间的数，其中 0 表示顶部对齐，1 表示底部对齐，0.5 表示居中对齐等。

getLayoutAlignmentY

float getLayoutAlignmentY(Container target)

返回沿 Y 轴的对齐方式。它指定如何相对于其他组件对齐该组件。值应该是一个介于 0 和 1 之间的数，其中 0 表示顶部对齐，1 表示底部对齐，0.5 表示居中对齐等。

invalidateLayout

```
void invalidateLayout(Container target)
```

使布局失效，指示如果布局管理器缓存了信息，则应该将其丢弃。

MenuContainer 接口

所有菜单相关容器的超类。

方法详细信息

getFont

```
Font getFont()
```

remove

```
void remove(MenuComponent comp)
```

postEvent

```
@Deprecated
boolean postEvent(Event evt)
```

Paint 接口

此 Paint 接口定义如何为 Graphics2D 操作生成颜色模式。将实现 Paint 接口的类添加到 Graphics2D 上下文中，以便定义 draw 和 fill 方法所使用的颜色模式。

实现 Paint 的类的实例必须是只读的，因为将这些对象作为 setPaint 方法的属性来设置时，或者 Graphics2D 对象本身被复制时，Graphics2D 并不复制这些对象。

方法详细信息

createContext

```
PaintContext createContext(ColorModel cm,
                           Rectangle deviceBounds,
                           Rectangle2D userBounds,
                           AffineTransform xform,
                           RenderingHints hints)
```

创建并返回用来生成颜色模式的 PaintContext。因为传递给 createContext 的 ColorModel 参数只是一个提示，所以 Paint 的实现应该接受 ColorModel 的 null 参数。注意，如果应用程序没有首选的特定 ColorModel，则为 null 的 ColorModel 参数将给予 Paint 实现完全的选择余地，使其在光栅处理中使用其首选最高效的 ColorModel。

因为 API 文档在 1.4 版本之前没有关于此项的具体描述，因此可能有一些 Paint 实现不能接受 null ColorModel 参数。如果开发人员正在编写代码将 null ColorModel 参数从任意源传递给 Paint 对象的 createContext 方法，则为了实现安全的编码，应该为这些对象构造一个非 null ColorModel，使其抛出 NullPointerException。

参数：

cm - 接收 Paint 数据的 ColorModel。这只用作一个提示。
deviceBounds - 正在呈现的图形图元的设备空间边界框
userBounds - 正在呈现的图形图元的用户空间边界框
xform - 从用户空间到设备空间的 AffineTransform
hints - 上下文对象用于选择所呈现内容的提示

返回：

生成颜色模式的 PaintContext

PaintContext 接口

PaintContext 接口定义了经过封装和优化的环境，使用此环境可以生成设备空间中的颜色模式，此模式用于在 Graphics2D 上进行填充操作或笔划操作。PaintContext 采用与

ColorModel 相关联的 Raster 的方式提供 Graphics2D 操作所必需的颜色。PaintContext 保持着进行特殊着色操作的状态。在多线程环境中，针对单个 Paint 对象可同时有多个上下文。

方法详细信息

dispose

```
void dispose()
```

释放为操作分配的资源。

---

getColorModel

```
ColorModel getColorModel()
```

返回输出的 ColorModel。注意，此 ColorModel 可能与在 Paint 的 createContext 方法中指定的提示是不同的。不是所有的 PaintContext 对象都能够生成任意 ColorModel 的颜色模式。

返回：

输出的 ColorModel。

---

getRaster

```
Raster getRaster(int x,
                 int y,
                 int w,
                 int h)
```

返回包含为图形操作生成的颜色的 Raster。

参数：

x - 为其生成颜色的设备空间区域的 x 坐标。

y - 为其生成颜色的设备空间区域的 y 坐标。

w - 设备空间区域的宽度

h - 设备空间区域的高度

返回：

一个 Raster，它表示指定的矩形区域，该区域包含为图形操作生成的颜色。

PrintGraphics 接口

提供页面的打印图形上下文的抽象类。

方法详细信息

getPrintJob

```
PrintJob getPrintJob()
```

返回从其产生 PrintGraphics 对象的 PrintJob 对象。

Shape 接口

Shape 接口提供了表示一些几何形状的对象的定义。Shape 是由 PathIterator 对象描述的，它可以表示 Shape 的轮廓以及确定该轮廓如何将 2D 平面划分成内点和外点的规则。每个 Shape 对象都提供回调，以获取几何形状的边框，确定点或矩形是部分还是全部位于 Shape 内部，并检索一个描述 Shape 轮廓的轨迹路径的 PathIterator 对象。

内部定义：当且仅当以下条件成立时，才认为某个点位于 Shape 内：

该点完全位于 Shape 边界内，或者

该点恰好位于 Shape 边界上，并且 X 轴正方向上紧邻该点的空间完全处于边界之内，或者

该点恰好在水平边界分段上，并且 Y 轴正方向上紧邻该点的空间完全处于边界之内。

contains 和 intersects 方法将 Shape 内部视为可以填充的封闭区域。这意味着为了确定某个 shape 是否包含矩形或与矩形相交，或者确定某个 shape 是否包含一个点，这些方法将隐式地认为未闭合的 shape 是闭合的。

方法详细信息

getBounds

Rectangle getBounds()

返回一个完全包围 Shape 的整型 Rectangle。注意，不保证返回的 Rectangle 是包围 Shape 的最小边界框，只保证 Shape 完全位于指示的 Rectangle 中。如果 Shape 超出了整数数据类型的有效范围，则返回的 Rectangle 也可能不完全包围 Shape。getBounds2D 方法由于在表示形式上具有更大的灵活性，所以通常返回更紧密的边界框。

返回：

完全包围 Shape 的整型 Rectangle。

从以下版本开始：

1.2

另请参见：

getBounds2D()

getBounds2D

Rectangle2D getBounds2D()

返回一个高精度的、比 getBounds 方法更准确的 Shape 边界框。注意，不保证返回的 Rectangle2D 是包围 Shape 的最小边界框，只保证 Shape 完全位于指示的 Rectangle2D 中。此方法返回的边界框通常比 getBounds 方法返回的更紧密，而且永远不会因为溢出问题而出错，因为返回值可以是一个使用双精度值存储尺寸的 Rectangle2D 实例。

返回：

一个 Rectangle2D 实例，它是 Shape 的高精度边界框。

从以下版本开始：

1.2

另请参见：

getBounds()

contains

boolean contains(double x,
double y)

测试指定坐标是否在 Shape 的边界内。

参数：

x - 要测试的指定的 X 坐标
y - 要测试的指定的 Y 坐标
返回：
如果指定坐标在 Shape 边界内，则返回 true；否则返回 false。
从以下版本开始：
1.2

---

contains

```
boolean contains(Point2D p)
```

测试指定的 Point2D 是否在 Shape 的边界内。
参数：
p - 要测试的指定的 Point2D
返回：
如果指定的 Point2D 在 Shape 边界内，则返回 true；否则返回 false。
从以下版本开始：
1.2

---

intersects

```
boolean intersects(double x,
                   double y,
                   double w,
                   double h)
```

测试 Shape 内部是否与指定矩形区域的内部相交。如果任何一个点既包含在 Shape 内，又包含在指定矩形区域内，则认为矩形区域与 Shape 相交。

在下列情况下，Shape.intersects() 方法允许 Shape 实现谨慎地返回 true：

- 矩形区域与 Shape 相交的可能性很大，但是
- 精确确定相交的计算代价太高。

这意味着对于某些 Shape，即使矩形区域没有与该 Shape 相交，此方法也可能返

回 true。如果需要更精确的答案，由于 Area 类比大多数 Shape 对象更为准确地计算几何相交，因此可以使用该类。

参数：

x - 指定矩形区域左上角的 X 坐标

y - 指定矩形区域左上角的 Y 坐标

w - 指定矩形区域的宽度

h - 指定矩形区域的高度

返回：

如果 Shape 的内部区域与矩形的内部区域相交，或者相交的可能性很大且执行计算的代价太高，则返回 true；否则返回 false。

从以下版本开始：

1.2

另请参见：

Area

---

intersects

boolean intersects(Rectangle2D r)

测试 Shape 内部是否与指定 Rectangle2D 内部相交。在下列情况下，Shape.intersects() 方法允许 Shape 实现谨慎地返回 true：

- Rectangle2D 与 Shape 相交的可能性很大，但是
- 精确确定相交的计算代价太高。

这意味着对于某些 Shape，即使 Rectangle2D 没有与该 Shape 相交，此方法也可能返回 true。如果需要更精确的答案，由于 Area 类比大多数 Shape 对象更为准确地计算几何相交，因此可以使用该类。

参数：

r - 指定的 Rectangle2D

返回：

如果 Shape 内部与指定 Rectangle2D 内部相交，或者相交的可能性很大且执行计算的代价太高，则返回 true；否则返回 false。

从以下版本开始：

1.2

另请参见：

intersects(double, double, double, double)

---

contains

```
boolean contains(double x,
                 double y,
                 double w,
                 double h)
```

测试 Shape 内部是否完全包含指定矩形区域。矩形区域内的所有坐标都必须位于 Shape 中，才可以认为整个矩形区域包含在 Shape中。

在下列情况下，Shape.contains() 方法允许 Shape 实现谨慎地返回 false：

- intersect 方法返回 true 并且
- 计算 Shape 是否完全包含矩形区域的代价太高。

这意味着对于某些 Shape，即使 Shape 包含矩形区域，此方法也可能返回 false。如果需要更精确的答案，由于 Area 类比大多数 Shape 对象更为准确地执行几何计算，因此可以使用该类。

参数：

x - 指定矩形区域左上角的 X 坐标
y - 指定矩形区域左上角的 Y 坐标
w - 指定矩形区域的宽度
h - 指定矩形区域的高度

返回：

如果 Shape 内部完全包含指定矩形区域，则返回 true；否则，如果 Shape 包含矩形区域、intersects 方法返回 true 且执行包含计算代价太高，则返回 false。

从以下版本开始：

1.2

另请参见：

Area,intersects(double, double, double, double)

contains

```
boolean contains(Rectangle2D r)
```

测试 Shape 内部是否完全包含指定的 Rectangle2D。在下列情况下，

Shape.contains() 方法允许 Shape 实现谨慎地返回 false：

- intersect 方法返回 true 并且
- 计算 Shape 是否完全包含 Rectangle2D 的代价太高。

这意味着对于某些 Shape，即使 Shape 包含 Rectangle2D，此方法也可能返回 false。如果需要更精确的答案，由于 Area 类比大多数 Shape 对象更为准确地执行几何计算，因此可以使用该类。

参数：

r - 指定的 Rectangle2D

返回：

如果 Shape 内部完全包含 Rectangle2D，则返回 true；否则，如果 Shape 包含 Rectangle2D、intersects 方法返回 true 且执行包含计算代价太高，则返回 false。

从以下版本开始：

1.2

另请参见：

contains(double, double, double, double)

getPathIterator

PathIterator getPathIterator(AffineTransform at)

返回一个沿着 Shape 边界迭代并提供对 Shape 轮廓几何形状的访问的迭代器对象。如果指定一个可选 AffineTransform，则相应地转换迭代中返回的坐标。

每次调用此方法都会返回一个最新的、遍历 Shape 对象几何形状的 PathIterator 对象，该对象独立于其他所有同时使用的 PathIterator 对象。

建议但不保证实现 Shape 接口的对象将进行中的迭代与该迭代期间可能对原始对象几何形状所做的任何更改隔离开来。

参数：

at - 一个可选的 AffineTransform，用于在迭代中返回的坐标，如果需要未转换的坐标，则返回 null

返回：

一个新的、独立地遍历 Shape 的几何形状的 PathIterator 对象。

从以下版本开始：
1.2

getPathIterator

PathIterator getPathIterator(AffineTransform at,
double flatness)

返回一个沿着 Shape 边界迭代并提供对 Shape 轮廓几何形状的平面视图访问的迭代器对象。

迭代器只返回 SEG_MOVETO、SEG_LINETO 和 SEG_CLOSE 点类型。

如果指定一个可选 AffineTransform，则相应地转换迭代中返回的坐标。

flatness 参数控制曲线段分段的数量，指定转换的不平曲线上任一点能够偏离返回的变平路径段的最大距离。注意，变平路径的精度限制可能稍微受到影响，使非常小的变平参数作为较大的值来处理。如果有这样的限制，那么该限制是由使用的特定实现定义的。

每次调用此方法都会返回一个最新的、遍历 Shape 对象几何形状的 PathIterator 对象，该对象独立于其他所有同时使用的 PathIterator 对象。

建议但不保证实现 Shape 接口的对象将进行中的迭代与该迭代期间可能对原始对象几何形状所做的任何更改隔离开来。

参数：
at - 一个可选 AffineTransform，用于在迭代中返回的坐标，如果需要未转换的坐标，则返回 null
flatness - 用来近似曲线段的直线段偏离原始曲线上任一点的最大距离
返回：
一个新的、独立地遍历 Shape 几何形状的平面视图的 PathIterator。

Stroke 接口

Stroke 接口允许 Graphics2D 对象获得一个 Shape，该 Shape 是指定 Shape 的装饰轮廓，或该轮廓的风格表示形式。描绘一个 Shape 如同使用一支具有适当大小和形状的画笔描绘其轮廓。画笔着墨的区域是轮廓 Shape 封闭的区域。

Graphics2D 接口中使用由 Stroke 对象返回的轮廓 Shape 的方法包括：draw 和其他所有根据此方法实现的方法，比如 drawLine、drawRect、drawRoundRect、drawOval、drawArc、drawPolyline 和 drawPolygon。

实现 Stroke 的类的对象必须是只读的，因为在使用 setStroke 方法将这些对象设置为一个属性时，或者在 Graphics2D 对象没有复制自身时，Graphics2D 不会复制这些对象。如果在 Graphics2D 上下文中设置了一个 Stroke 对象后修改它，则后续呈现行为将是不确定的。

方法详细信息

createStrokedShape

```
Shape createStrokedShape(Shape p)
```

返回一个轮廓 Shape，它封闭了应该在根据规则描绘 Shape 时绘制的区域，这些规则是由实现 Stroke 接口的对象定义的。

参数：

p - 要描绘的 Shape

返回：

描绘的轮廓 Shape。

Transparency 接口

Transparency 接口定义用于实现类的通用透明模式。

字段详细信息

OPAQUE

```
static final int OPAQUE
```

表示保证完全不透明的图像数据，意味着所有像素 alpha 值都为 1.0。

另请参见：
常量字段值

BITMASK

static final int BITMASK

表示保证完全不透明的图像数据（alpha 值为 1.0）或完全透明的图像数据（alpha 值为 0.0）。
另请参见：
常量字段值

TRANSLUCENT

static final int TRANSLUCENT

表示包含或可能包含位于 0.0 和 1.0（含两者）之间的任意 alpha 值的图像数据。

方法详细信息

getTransparency

int getTransparency()

返回此 Transparency 的类型。
返回：
此 Transparency 的字段类型，具体为 OPAQUE、BITMASK 或 TRANSLUCENT。

类

AlphaComposite 类

AlphaComposite 类实现一些基本的 alpha 合成规则，将源色与目标色组合，在图形和图像中实现混合和透明效果。此类实现的特定规则是 T. Porter 和 T. Duff 合著的 "Compositing Digital Images", SIGGRAPH 84, 253-259 中描述的 12 条基本规则集。本文档的其余部分假定读者熟悉上述论文中概括的定义和概念。

此类扩展了 Porter 和 Duff 定义的方程，包含一个额外的因子。AlphaComposite 类的实例可以包含一个 alpha 值，在将该值用于混合方程之前，可以用它来修改不透明度和每个源像素的覆盖率。

要重点注意的是，Porter 和 Duff 的论文中定义的方程完全是为颜色分量方面的操作定义的，这些颜色分量都要用它们对应的 alpha 分量预乘。因为 ColorModel 和 Raster 类允许以预乘和非预乘的方式存储像素数据，所以在将所有输入数据应用到方程之前，必须将它们标准化为预乘形式，并且在存储像素值之前，可能需要将所有结果都调整回目标所需的形式。

还要注意的是，此类只定义了以纯数学方式组合颜色和 alpha 值的方程。方程的具体应用取决于从数据源中检索数据和将它们存储到其目标中的方式。有关更多信息，请参阅实现警告事项。

Porter 和 Duff 的论文在混合方程的描述中使用了以下因子：

因子	定义
As	源像素的 alpha 分量
Cs	源像素的预乘形式的颜色分量
Ad	目标像素的 alpha 分量
Cd	目标像素的预乘形式的颜色分量
Fs	用于输出的源像素的分数值
Fd	用于输出的目标像素的分数值
Ar	结果 alpha 分量
Cr	结果预乘形式的颜色分量

使用这些因子，Porter 和 Duff 定义了 12 种选择混合因子 *Fs* 和 *Fd* 的方法，从而产生了 12 种令人满意的可视效果。在对 12 个指定可视效果的静态字段的描述中，给出了具有确定 *Fs* 和 *Fd* 值的方程。例如，对 SRC_OVER 的描述指定了 *Fs* = 1 和 *Fd* = (1-*As*)。

一旦已知一组确定混合因子的方程，就可以使用以下方程组将它们应用于每个像素以生成结果：

$$Fs = f(Ad)$$
$$Fd = f(As)$$
$$Ar = As*Fs + Ad*Fd$$
$$Cr = Cs*Fs + Cd*Fd$$

在我们对 Porter 和 Duff 论文混合方程的扩展中，用到了以下因子：

因子	定义
Csr	源像素的原始颜色分量之一
Cdr	目标像素的原始颜色分量之一
Aac	取自 AlphaComposite 实例的“额外的”alpha 分量
Asr	源像素的原始 alpha 分量
Adr	目标像素的原始 alpha 分量
Adf	目标中存储的最终 alpha 分量
Cdf	目标中存储的最终原始颜色分量

准备输入

AlphaComposite 类定义一个应用于源 alpha 的额外 alpha 值。应用此值就好像首先将隐式的 SRC_IN 规则应用于源像素（通过将原始源 alpha 和原始源色乘以 AlphaComposite 中的 alpha 值获得），而不是应用于具有指定 alpha 值的像素。这产生了以下方程，该方程生成 Porter 和 Duff 的混合方程中使用的 alpha 值：

$$As = Asr * Aac$$

所有这些原始源色分量都必须乘以 AlphaComposite 实例中的 alpha 值。此外，如果源色分量不是预乘形式的，那么还需要将颜色分量乘以源 alpha 值。因此，用来生成 Porter 和 Duff 方程源色分量的方程取决于源像素是否已经被预乘：

$$Cs = Csr * Asr * Aac$$ （如果源像素没有被预乘）
$$Cs = Csr * Aac$$ （如果源像素被预乘）

无需对目标 alpha 进行调整：

$$Ad = Adr$$

仅当目标色分量不是预乘形式时，才需要对它们进行调整：

$$Cd = Cdr * Ad$$ （如果目标色分量没有被预乘）
$$Cd = Cdr$$ （如果目标色分量被预乘）

应用混合方程

调整后的 *As*、*Ad*、*Cs* 和 *Cd* 将用于标准的 Porter 和 Duff 方程，以计算混合因子 *Fs* 和 *Fd*，然后计算得到的预乘分量 *Ar* 和 *Cr*。

准备结果

仅当结果要存储回保存未预乘数据的目标缓冲区时，才需要使用以下方程调整结果：

$$Adf = Ar$$

Cdf = Cr　　（如果目标数据被预乘）

Cdf = Cr / Ar　　（如果目标数据没有被预乘）

注意，在所得 alpha 为零时除法是不明确的，所以在这种情况下会忽略除法以避免“除以零”的情况，此时颜色分量都将为零。

性能考虑事项

出于性能方面的原因，传递给 CompositeContext 对象（由 AlphaComposite 类创建）compose 方法的 Raster 对象最好有预乘数据。不过，如果源 Raster 或目标 Raster 没有被预乘，那么可以在合成操作之前或之后执行适当的转换。

实现警告事项

许多源图像，比如 BufferedImage 类中列出的一些不透明图像类型，没有为它们的像素存储 alpha 值。这类源图像为它们所有的像素提供了值为 1.0 的 alpha 值。

许多目标也没有地方存储 alpha 值（这些值是此类执行混合计算的结果）。这类目标会隐式丢弃此类产生的 alpha 值。建议这类目标将它们存储的颜色值作为未预乘的值对待，并在存储颜色值和丢弃 alpha 值之前，将得到的颜色值除以得到的 alpha 值。

结果的精度取决于将像素存储到目标中的方式。对于一连串十二种合成操作中的少数操作，将为每种颜色和 alpha 分量提供至少 8 个存储位的图像格式作为目标无论如何都应该足够了。在舍入误差支配结果之前，为每个分量提供少于 8 个存储位的图像格式仅限用于一或两个合成操作。对于任何半透明混合的类型而言，不单独存储颜色分量的图像格式不是一个好的候选方式。例如，不应将 BufferedImage.TYPE_BYTE_INDEXED 用作混合操作的目标，因为需要从限定的调色板选择像素，以匹配混合方程的结果，所以每个操作都可能引入大量错误。

几乎所有的格式都将像素存储为离散整数，而不是将它们存储为上述方程中使用的浮点值。该实现可以将整数像素值缩放成范围在 0.0 到 1.0 之间的浮点值，或者使用稍作修改的方程，完全在整数域内操作，从而生成与上述方程类似的结果。

通常，整数值与浮点值存在某种相关性：整数 0 等于浮点值 0.0，整数 2^*n*-1（其中 *n* 是表示形式中的位数）等于 1.0。对于 8 位的表示形式，这意味着 0x00 表示 0.0，0xff 表示 1.0。

内部实现可能近似于某些方程，它们也可以消除一些步骤，以避免不必要的操作。例如，可以考虑一个离散整数图像，它带有未预乘的 alpha 值，并为存储每个分量使用了 8 个位。接近透明的暗红色存储的值可能是：

```
(A, R, G, B) = (0x01, 0xb0, 0x00, 0x00)
```

如果正在使用整数运算，并且此值在 SRC 模式下没有与任何 alpha 值合成，则该运算将指示此运算的结果为（整数格式）：

```
(A, R, G, B) = (0x01, 0x01, 0x00, 0x00)
```

注意中间的值，它总是以已预乘的形式出现，并且只允许整数 red 分量为 0x00 或 0x01。当试图将此结果存储回未预乘的目标中时，用 alpha 值相除之后，对于未预乘的 red 值，可进行的选择很少。在这种情况下，在整数空间内执行运算（没有快捷方式）的实现最后可能提供以下最终像素值：

(A, R, G, B) = (0x01, 0xff, 0x00, 0x00)

（注意，0x01 除以 0x01 得到的是 1.0，等于 8 位存储格式的值 0xff。）

另一方面，使用浮点运算的实现可以生成更精确的结果，并以返回原始像素值结束，该值有可能带有少量的舍入误差。或者，使用整数运算的实现可能决定是否可以将未涉及的像素传输给目标，完全避免所有运算。因为如果在浮点空间内执行运算，方程可简单归结为颜色值上的一个虚拟 NOP。

这些实现都试图遵守相同的方程，但使用经过权衡的不同整数和浮点运算，并使用部分的或全部的方程。为了说明这些不同之处，可能最好只期望获得已预乘形式的结果，以在实现和图像格式之间进行匹配。在这种情况下，以预乘形式表示的两种答案将等同于：

(A, R, G, B) = (0x01, 0x01, 0x00, 0x00)

并且它们将是完全匹配的。

因为那些通过简化方程使计算更有效的技术，在非预乘的目标上遇到值为 0.0 的结果 alpha 值时，一些实现的执行可能会有所不同。注意，如果分母 (alpha) 为 0，则在 SRC 规则下移除除以 alpha 这一简化操作技术上是不合法的。但是，因为在以预乘形式查看时只期望结果是精确的，所以，结果为 0 的 alpha 值呈现的实质上是所得到的不相关颜色分量，因此，在这种情况下，具体的行为应该是无法预料的。

字段详细信息

CLEAR

public static final int CLEAR

目标色和目标 alpha 值都被清除（Porter-Duff Clear 规则）。源色和目标色都不被用作输入。

$F_s = 0$ 和 $F_d = 0$，因此：

$$A_r = 0$$

$$C_r = 0$$

另请参见：
常量字段值

SRC

public static final int SRC

将源色复制到目标色（Porter-Duff Source 规则）。目标色不被用作输入。

F_s = 1 和 F_d = 0，因此：

$$A_r = A_s$$

$$C_r = C_s$$

另请参见：

常量字段值

DST

public static final int DST

目标色不变（Porter-Duff Destination 规则）。

F_s = 0 和 F_d = 1，因此：

$$A_r = A_d$$

$$C_r = C_d$$

从以下版本开始：

1.4

另请参见：

常量字段值

SRC_OVER

public static final int SRC_OVER

在目标色之上合成源色（Porter-Duff Source Over Destination 规则）。

F_s = 1 和 F_d = (1−A_s)，因此：

$$A_r = A_s + A_d*(1-A_s)$$

$$C_r = C_s + C_d*(1-A_s)$$

另请参见：

常量字段值

DST_OVER

public static final int DST_OVER

在源色之上合成目标色，产生的结果将替代目标色（Porter-Duff Destination Over Source 规则）。

F_s = (1−A_d) 和 F_d = 1，因此：

$$A_r = A_s*(1-A_d) + A_d$$
$$C_r = C_s*(1-A_d) + C_d$$

另请参见：

常量字段值

SRC_IN

public static final int SRC_IN

目标色中的源色部分将替换目标色（Porter-Duff Source In Destination 规则）。

F_s = A_d 和 F_d = 0，因此：

$$A_r = A_s*A_d$$
$$C_r = C_s*A_d$$

另请参见：

常量字段值

DST_IN

public static final int DST_IN

源色中的目标色部分将替换目标色（Porter-Duff Destination In Source 规则）。

F_s = 0 和 F_d = A_s，因此：

$$A_r = A_d * A_s$$
$$C_r = C_d * A_s$$

另请参见：

常量字段值

SRC_OUT

public static final int SRC_OUT

目标色以外的源色部分将替换目标色（Porter-Duff Source Held Out By Destination 规则）。

$F_s = (1-A_d)$ 和 $F_d = 0$，因此：

$$A_r = A_s * (1-A_d)$$
$$C_r = C_s * (1-A_d)$$

另请参见：

常量字段值

DST_OUT

public static final int DST_OUT

源色以外的目标色部分将替换目标色（Porter-Duff Destination Held Out By Source 规则）。

$F_s = 0$ 和 $F_d = (1-A_s)$，因此：

$$A_r = A_d * (1-A_s)$$
$$C_r = C_d * (1-A_s)$$

另请参见：

常量字段值

SRC_ATOP

public static final int SRC_ATOP

目标色中的源色部分将被合成到目标色中（Porter-Duff Source Atop Destination 规则）。

$F_s = A_d$ 和 $F_d = (1-A_s)$，因此：

$$A_r = A_s*A_d + A_d*(1-A_s) = A_d$$
$$C_r = C_s*A_d + C_d*(1-A_s)$$

从以下版本开始：

1.4

另请参见：

常量字段值

DST_ATOP

public static final int DST_ATOP

在源色之上合成源色中的目标色部分，并将替换目标色（Porter-Duff Destination Atop Source 规则）。

$F_s = (1-A_d)$ 和 $F_d = A_s$，因此：

$$A_r = A_s*(1-A_d) + A_d*A_s = A_s$$
$$C_r = C_s*(1-A_d) + C_d*A_s$$

从以下版本开始：

1.4

另请参见：

常量字段值

XOR

public static final int XOR

将目标色之外的源色部分与源色之外的目标色部分结合到一起（Porter-Duff Source Xor Destination 规则）。

$F_s = (1-A_d)$ 和 $F_d = (1-A_s)$，因此：

$$A_r = A_s*(1-A_d) + A_d*(1-A_s)$$
$$C_r = C_s*(1-A_d) + C_d*(1-A_s)$$

从以下版本开始：

1.4

另请参见：

常量字段值

Clear

public static final AlphaComposite Clear

实现不透明 CLEAR 规则的 AlphaComposite 对象，具有 1.0f 的 alpha 值。

另请参见：

CLEAR

Src

public static final AlphaComposite Src

实现不透明 SRC 规则的 AlphaComposite 对象，具有 1.0f 的 alpha 值。

另请参见：

SRC

Dst

public static final AlphaComposite Dst

实现不透明 DST 规则的 AlphaComposite 对象，具有 1.0f 的 alpha 值。

从以下版本开始：

1.4

另请参见：

DST

SrcOver

public static final AlphaComposite SrcOver

实现不透明 SRC_OVER 规则的 AlphaComposite 对象，具有 1.0f 的 alpha 值。

另请参见：

SRC_OVER

DstOver

public static final AlphaComposite DstOver

实现不透明 DST_OVER 规则的 AlphaComposite 对象，具有 1.0f 的 alpha 值。

另请参见：

DST_OVER

SrcIn

public static final AlphaComposite SrcIn

实现不透明 SRC_IN 规则的 AlphaComposite 对象，具有 1.0f 的 alpha 值。

另请参见：

SRC_IN

DstIn

public static final AlphaComposite DstIn

实现不透明 DST_IN 规则的 AlphaComposite 对象，具有 1.0f 的 alpha 值。

另请参见：

DST_IN

SrcOut

public static final AlphaComposite SrcOut

实现不透明 SRC_OUT 规则的 AlphaComposite 对象，具有 1.0f 的 alpha 值。

另请参见：
SRC_OUT

DstOut

public static final AlphaComposite DstOut

实现不透明 DST_OUT 规则的 AlphaComposite 对象，具有 1.0f 的 alpha 值。
另请参见：
DST_OUT

SrcAtop

public static final AlphaComposite SrcAtop

实现不透明 SRC_ATOP 规则的 AlphaComposite 对象，具有 1.0f 的 alpha 值。
从以下版本开始：
1.4
另请参见：
SRC_ATOP

DstAtop

public static final AlphaComposite DstAtop

实现不透明 DST_ATOP 规则的 AlphaComposite 对象，具有 1.0f 的 alpha 值。
从以下版本开始：
1.4
另请参见：
DST_ATOP

Xor

public static final AlphaComposite Xor

实现不透明 XOR 规则的 AlphaComposite 对象，具有 1.0f 的 alpha 值。

方法详细信息

getInstance

public static AlphaComposite getInstance(int rule)

创建一个具有指定规则的 AlphaComposite 对象。

参数：

rule - 合成规则

抛出：

IllegalArgumentException - 如果 rule 不是以下规则之一：CLEAR、SRC、DST、SRC_OVER、DST_OVER、SRC_IN、DST_IN、SRC_OUT、DST_OUT、SRC_ATOP、DST_ATOP 或 XOR

---

getInstance

public static AlphaComposite getInstance(int rule,
float alpha)

创建一个 AlphaComposite 对象，它具有指定的规则和用来乘源色 alpha 值的常量 alpha 值。在将源色与目标色合成前，要将源色乘以指定的 alpha 值。

参数：

rule - 合成规则

alpha - 将乘源色的 alpha 值的常量 alpha 值。alpha 必须是范围 [0.0, 1.0] 之内（包含边界值）的一个浮点数字。

抛出：

IllegalArgumentException - 如果 alpha 小于 0.0 或大于 1.0，或者 rule 是以下规则之一：CLEAR、SRC、DST、SRC_OVER、DST_OVER、SRC_IN、DST_IN、SRC_OUT、DST_OUT、SRC_ATOP、DST_ATOP 或 XOR

---

createContext

```
public CompositeContext createContext(ColorModel srcColorModel,
                                      ColorModel dstColorModel,
                                      RenderingHints hints)
```

为合成操作创建一个上下文。上下文中包括执行合成操作过程中使用的状态。

指定者：

接口 Composite 中的 createContext

参数：

srcColorModel - 源色的 ColorModel

dstColorModel - 目标色的 ColorModel

hints - 上下文对象用于选择呈现方法的提示

返回：

用来执行合成操作的 CompositeContext 对象。

getAlpha

```
public float getAlpha()
```

返回此 AlphaComposite 的 alpha 值。如果此 AlphaComposite 没有 alpha 值，则返回 1.0。

返回：

此 AlphaComposite 的 alpha 值。

getRule

```
public int getRule()
```

返回此 AlphaComposite 的合成规则。

返回：

此 AlphaComposite 的合成规则。

derive

```
public AlphaComposite derive(int rule)
```

返回一个类似的 AlphaComposite 对象，该对象使用指定的复合规则。如果此对象

已经使用指定的复合规则，则返回此对象。

参数：

rule - 复合规则

返回：

派生于使用指定复合规则的对象的 AlphaComposite 对象。

抛出：

IllegalArgumentException - 如果 rule 不是以下规则之一： CLEAR、SRC、DST、SRC_OVER、DST_OVER、SRC_IN、DST_IN、SRC_OUT、DST_OUT、SRC_ATOP、DST_ATOP 或 XOR

从以下版本开始：

1.6

---

derive

```
public AlphaComposite derive(float alpha)
```

返回一个类似的 AlphaComposite 对象，该对象使用指定的 alpha 值。如果此对象已经具有指定的 alpha 值，则返回此对象。

参数：

alpha - 常量 alpha，将乘以源的 alpha 值。alpha 必须是闭区间 [0.0, 1.0] 中的一个浮点数字。

返回：

派生于使用指定 alpha 值的对象的 AlphaComposite 对象。

抛出：

IllegalArgumentException - 如果 alpha 小于 0.0 或大于 1.0

从以下版本开始：

1.6

---

hashCode

```
public int hashCode()
```

返回此合成的哈希码。

覆盖：

类 Object 中的 hashCode

返回：

此合成的哈希码。

另请参见：

Object.equals(java.lang.Object),Hashtable

equals

```
public boolean equals(Object obj)
```

确定指定的对象是否等于此 AlphaComposite。

当且仅当参数不为 null，并且是一个与此对象具有相同组合规则和 alpha 值的 AlphaComposite 对象时，结果才为 true。

覆盖：

类 Object 中的 equals

参数：

obj - 要测试相等性的 Object

返回：

如果 obj 等于此 AlphaComposite，则返回 true；否则返回 false。

BorderLayout 类

这是一个布置容器的边框布局，它可以对容器组件进行安排，并调整其大小，使其符合下列五个区域：北、南、东、西、中。每个区域最多只能包含一个组件，并通过相应的常量进行标识：NORTH、SOUTH、EAST、WEST、CENTER。当使用边框布局将一个组件添加到容器中时，要使用这五个常量之一，例如：

```
Panel p = new Panel();
p.setLayout(new BorderLayout());
p.add(new Button("Okay"), BorderLayout.SOUTH);
```

为了方便起见，BorderLayout 将缺少字符串说明的情况解释为常量 CENTER：

```
Panel p2 = new Panel();
p2.setLayout(new BorderLayout());
p2.add(new TextArea());  // Same as p.add(new TextArea(), BorderLayout.CENTER);
```

此外，BorderLayout 支持相对定位常量 PAGE_START、PAGE_END、LINE_START 和 LINE_END。在 ComponentOrientation 设置为 ComponentOrientation.LEFT_TO_RIGHT 的容器中，这些常量分别映射到 NORTH、SOUTH、WEST 和 EAST。

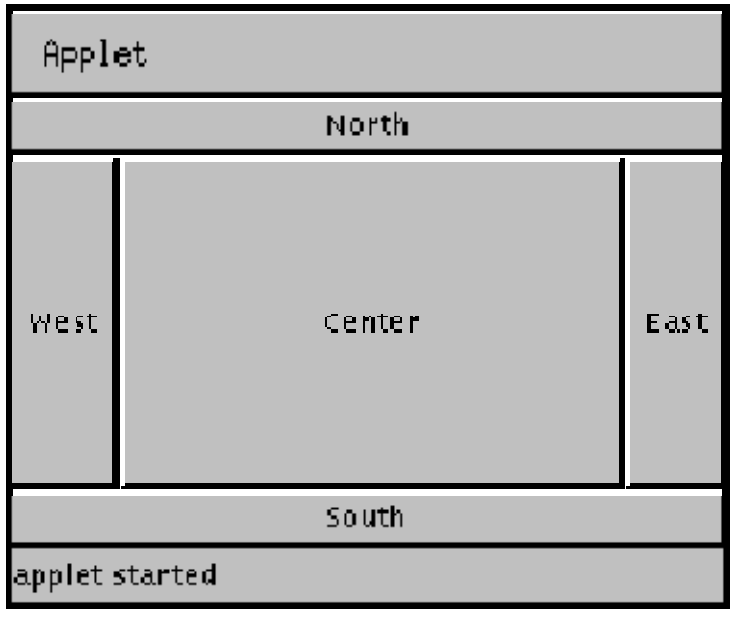
为了与以前的版本兼容，BorderLayout 还包括相对定位常量 BEFORE_FIRST_LINE、AFTER_LAST_LINE、BEFORE_LINE_BEGINS 和 AFTER_LINE_ENDS。这些常量分别等同于 PAGE_START、PAGE_END、LINE_START 和 LINE_END。为了与其他组件使用的相对定位常量一致，应优先使用后一组常量。

将绝对定位常量与相对定位常量混合会产生无法预料的结果。如果两种类型的常量都使用，则优先采用相对常量。例如，如果同时使用 NORTH 和 PAGE_START 常量在方向性为 LEFT_TO_RIGHT 的容器中添加组件，则只体现 PAGE_START 布局。

注：目前，在 Java 2 Platform v1.2 中，BorderLayout 不支持垂直方向。不考虑容器的 ComponentOrientation 上 isVertical 设置。

根据其首选大小和容器大小的约束 (constraints) 对组件进行布局。NORTH 和 SOUTH 组件可以在水平方向上拉伸；而 EAST 和 WEST 组件可以在垂直方向上拉伸；CENTER 组件可同时在水平和垂直方向上拉伸，从而填充所有剩余空间。

以下是一个使用 BorderLayout 布局管理器的例子，它对一个 applet 中的五个按钮进行布局：



此 applet 的代码如下：

```
import java.awt.*;
```

```
import java.applet.Applet;

public class buttonDir extends Applet {
   public void init() {
      setLayout(new BorderLayout());
      add(new Button("North"), BorderLayout.NORTH);
      add(new Button("South"), BorderLayout.SOUTH);
      add(new Button("East"), BorderLayout.EAST);
      add(new Button("West"), BorderLayout.WEST);
      add(new Button("Center"), BorderLayout.CENTER);
   }
}
```

字段详细信息

NORTH

```
public static final String NORTH
```

北区域的布局约束（容器顶部）。

另请参见：

常量字段值

SOUTH

```
public static final String SOUTH
```

南区域的布局约束（容器底部）。

另请参见：

常量字段值

EAST

public static final String EAST

东区域的布局约束（容器右边）。

另请参见：

常量字段值

WEST

public static final String WEST

西区域的布局约束（容器左边）。

另请参见：

常量字段值

CENTER

public static final String CENTER

中间区域的布局约束（容器中央）。

另请参见：

常量字段值

BEFORE_FIRST_LINE

public static final String BEFORE_FIRST_LINE

与 PAGE_START 同义。为了与以前版本兼容而存在。首选 PAGE_START。

从以下版本开始：

1.2

另请参见：

PAGE_START, 常量字段值

AFTER_LAST_LINE

public static final String AFTER_LAST_LINE

与 PAGE_END 同义。为了与以前版本兼容而存在。首选 PAGE_END。

从以下版本开始：

1.2

另请参见：

PAGE_END, 常量字段值

BEFORE_LINE_BEGINS

public static final String BEFORE_LINE_BEGINS

与 LINE_START 同义。为了与以前版本兼容而存在。首选 LINE_START。

从以下版本开始：

1.2

另请参见：

LINE_START, 常量字段值

AFTER_LINE_ENDS

public static final String AFTER_LINE_ENDS

与 LINE_END 同义。为了与以前版本兼容而存在。首选 LINE_END。

从以下版本开始：

1.2

另请参见：

LINE_END, 常量字段值

PAGE_START

public static final String PAGE_START

组件出现在第一行布局内容之前。对于 Western，方向是从左到右、从上到下，等同于 NORTH。

从以下版本开始：

1.4

另请参见：

Component.getComponentOrientation(),常量字段值

PAGE_END

public static final String PAGE_END

组件出现在最后一行布局内容之后。对于 Western，方向是从左到右、从上到下，等同于 SOUTH。

从以下版本开始：

1.4

另请参见：

Component.getComponentOrientation(),常量字段值

LINE_START

public static final String LINE_START

组件出现在布局的行方向的开始处。对于 Western，方向是从左到右、从上到下，等同于 WEST。

从以下版本开始：

1.4

另请参见：

Component.getComponentOrientation(),常量字段值

LINE_END

public static final String LINE_END

组件出现在布局的行方向的结尾处。对于 Western，方向是从左到右、从上到下，等同于 EAST。

从以下版本开始：

1.4

另请参见：

Component.getComponentOrientation(),常量字段值

构造方法详细信息

BorderLayout

```
public BorderLayout()
```

构造一个组件之间没有间距的新边框布局。

BorderLayout

```
public BorderLayout(int hgap,
                    int vgap)
```

构造一个具有指定组件间距的边框布局。水平间距由 hgap 指定，垂直间距由 vgap 指定。

参数：

hgap - 水平间距。

vgap - 垂直间距。

方法详细信息

getHgap

```
public int getHgap()
```

返回组件之间的水平间距。

从以下版本开始：

JDK1.1

---

setHgap

```
public void setHgap(int hgap)
```

设置组件之间的水平间距。

参数：
hgap - 组件之间的水平间距
从以下版本开始：
JDK1.1

---

getVgap

```
public int getVgap()
```

返回组件之间的垂直间距。
从以下版本开始：
JDK1.1

---

setVgap

```
public void setVgap(int vgap)
```

设置组件之间的垂直间距。
参数：
vgap - 组件之间的垂直间距
从以下版本开始：
JDK1.1

---

addLayoutComponent

```
public void addLayoutComponent(Component comp,
                               Object constraints)
```

使用指定的约束对象将指定组件添加到布局中。对于边框布局，约束必须是以下约束之一：NORTH、SOUTH、EAST、WEST 或 CENTER。

大多数应用程序并不直接调用此方法。当使用 Container.add 方法将组件添加到容器中时，可以使用相同的参数类型调用此方法。

指定者：

接口 LayoutManager2 中的 addLayoutComponent

参数：

comp - 要添加的组件。

constraints - 指定将组件添加到布局中的方式和位置的对象。

抛出：

IllegalArgumentException - 如果约束对象不是一个字符串，或者它不是五种指定约束之一。

从以下版本开始：

JDK1.1

另请参见：

Container.add(java.awt.Component, java.lang.Object)

---

addLayoutComponent

```
@Deprecated
    public void addLayoutComponent(String name,
                                   Component comp)
```

已过时。 *由 addLayoutComponent(Component, Object) 取代。*

从接口 LayoutManager 复制的描述

如果布局管理器使用每组件字符串，则将组件 comp 添加到布局，并将它与 name 指定的字符串关联。

指定者：

接口 LayoutManager 中的 addLayoutComponent

参数：

name - 要与组件关联的字符串

comp - 要添加的组件

---

removeLayoutComponent

```
    public void removeLayoutComponent(Component comp)
```

从此边框布局中移除指定组件。当容器调用其 remove 或 removeAll 方法时，可调用此方法。大多数应用程序并不直接调用此方法。

指定者：

接口 LayoutManager 中的 removeLayoutComponent

参数：

comp - 要移除的组件。

另请参见：

Container.remove(java.awt.Component),Container.removeAll()

getLayoutComponent

```
public Component getLayoutComponent(Object constraints)
```

获取使用给定约束添加的组件。

参数：

constraints - 所需的约束，它是 CENTER、NORTH、SOUTH、WEST、EAST、PAGE_START、PAGE_END、LINE_START、LINE_END 之一

返回：

给定位置上的组件；如果此位置为空，则返回 null

抛出：

IllegalArgumentException - 如果约束对象不是九个指定约束之一

从以下版本开始：

1.5

另请参见：

addLayoutComponent(java.awt.Component, java.lang.Object)

getLayoutComponent

```
public Component getLayoutComponent(Container target,
                                    Object constraints)
```

基于目标 Container 的组件方向，返回给定约束位置对应的组件。利用相对约束 PAGE_START、PAGE_END、LINE_START 和 LINE_END 添加的组件优先于利用显式约束 NORTH、SOUTH、WEST 和 EAST 添加的组件。Container 的组件方向用于确定利用 LINE_START 和 LINE_END 添加的组件的位置。

参数：

constraints - 所需的绝对位置，CENTER、NORTH、SOUTH、EAST 和 WEST 之一

target - Container，用来获取基于目标 Container 组件方向的约束位置。

返回：

指定位置上的组件；如果此位置为空，则返回 null

抛出：
IllegalArgumentException - 如果约束对象不是五个指定约束之一
NullPointerException - 如果目标参数为 null
从以下版本开始：
1.5
另请参见：
addLayoutComponent(java.awt.Component, java.lang.Object)

getConstraints

public Object getConstraints(Component comp)

获取指定组件的约束
参数：
comp - 要查询的组件
返回：
指定组件的约束；如果组件为 null 或不在此布局中，则返回 null
从以下版本开始：
1.5
另请参见：
addLayoutComponent(java.awt.Component, java.lang.Object)

minimumLayoutSize

public Dimension minimumLayoutSize(Container target)

使用此布局管理器确定 target 容器的最小大小。

当容器调用其 getMinimumSize 方法时，可以调用此方法。大多数应用程序并不直接调用此方法。

指定者：
接口 LayoutManager 中的 minimumLayoutSize
参数：
target - 在其中进行布局的容器。
返回：

对指定容器的子组件进行布局所需的最小尺寸。

另请参见：

Container, preferredLayoutSize(java.awt.Container), Container.getMinimumSize()

preferredLayoutSize

public Dimension preferredLayoutSize(Container target)

基于容器中的组件，使用此布局管理器确定 target 容器的首选大小。

大多数应用程序并不直接调用此方法。容器调用其 getPreferredSize 方法时将调用此方法。

指定者：

接口 LayoutManager 中的 preferredLayoutSize

参数：

target - 在其中进行布局的容器。

返回：

对指定容器的子组件进行布局所需的首选尺寸。

另请参见：

Container, minimumLayoutSize(java.awt.Container), Container.getPreferredSize()

maximumLayoutSize

public Dimension maximumLayoutSize(Container target)

在给出指定目标容器中的组件的前提下，返回此布局的最大尺寸。

指定者：

接口 LayoutManager2 中的 maximumLayoutSize

参数：

target - 需要对其进行布局的组件

另请参见：

Container, minimumLayoutSize(java.awt.Container), preferredLayoutSize(java.awt.Container)

getLayoutAlignmentX

public float getLayoutAlignmentX(Container parent)

返回沿 x 轴的对齐方式。这指出了相对于其他组件将如何排列该组件。该值应该在 0 到 1 之间，其中 0 表示根据原点进行对齐，1 表示根据距原点最远的点对齐，0.5 表示居中对齐等等。

指定者：

接口 LayoutManager2 中的 getLayoutAlignmentX

getLayoutAlignmentY

public float getLayoutAlignmentY(Container parent)

返回沿 y 轴的对齐方式。这指出了相对于其他组件将如何排列该组件。该值应该在 0 到 1 之间，其中 0 表示根据原点进行对齐，1 表示根据距原点最远的点对齐，0.5 表示居中对齐等等。

指定者：

接口 LayoutManager2 中的 getLayoutAlignmentY

invalidateLayout

public void invalidateLayout(Container target)

使布局无效，指示如果布局管理器缓存了信息，则应该将其丢弃。

指定者：

接口 LayoutManager2 中的 invalidateLayout

layoutContainer

public void layoutContainer(Container target)

使用此边框布局对容器参数进行布局。

为了满足此 BorderLayout 对象的约束条件，此方法实际上会重塑指定容器中的组件。NORTH 和 SOUTH 组件（如果有）分别放置在容器的顶部和底部。WEST 和 EAST 组件分别放置在容器的左边和右边。最后，CENTER 对象放置在中间的任何剩余空间内。

大多数应用程序并不直接调用此方法。容器调用其 doLayout 方法时将调用此方法。

指定者：

接口 LayoutManager 中的 layoutContainer

参数：

target - 在其中进行布局的容器。

另请参见：

Container,Container.doLayout()

toString

```
public String toString()
```

返回此边框布局的状态的字符串表示形式。

覆盖：

类 Object 中的 toString

返回：

此边框布局的字符串表示形式。

Button 类

此类创建一个标签按钮。当按下该按钮时，应用程序能执行某项动作。以下图像描绘了在 Solaris 操作系统下，"Quit" 按钮所表现的三种视图：

第一幅视图展示该按钮在一般情况下的外观。第二幅视图展示该按钮成为输入焦点时的

外观。其边框变黑，让用户知道它是一个活动对象。第三幅视图展示用户在该按钮上单击鼠标，从而请求执行某个动作时该按钮的外观。

用鼠标单击按钮这一动作与 ActionEvent 的一个实例相关，在按下鼠标和释放按钮的时候都会用到该类。如果应用程序希望知道何时按钮作为一个单独动作被按下但未释放，它可以特殊化 processMouseEvent，或者通过调用 addMouseListener 将自身注册为鼠标事件的侦听器。这两种方法都是由所有组件的抽象超类 Component 定义的。

当按下按钮并释放时，AWT 通过调用按钮的 processEvent，将 ActionEvent 的一个实例发送给按钮。按钮的 processEvent 方法接收按钮的所有事件；同时，它通过调用自身的 processActionEvent 方法传递一个动作事件。后一种方法将动作事件传递给为关注此按钮生成的动作事件而注册的任何动作侦听器。

如果应用程序想要执行基于按下并释放按钮的某个动作，则它应该实现 ActionListener 并注册新的侦听器，以便通过调用按钮的 addActionListener 方法来接收发自此按钮的事件。应用程序可以按消息传递协议使用按钮的动作命令。

构造方法详细信息

Button

```
public Button()
        throws HeadlessException
```

构造一个标签字符串为空的按钮。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless()

Button

```
public Button(String label)
        throws HeadlessException
```

构造一个带指定标签的按钮。

参数：

label - 按钮的字符串标签，如果没有标签，则为 null

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless()

方法详细信息

addNotify

```
public void addNotify()
```

创建按钮的同位体。按钮的同位体允许应用程序更改按钮的外观。而不更改其功能。

覆盖：

类 Component 中的 addNotify

另请参见：

Toolkit.createButton(java.awt.Button),Component.getToolkit()

---

getLabel

```
public String getLabel()
```

获取此按钮的标签。

返回：

按钮的标签，如果按钮没有标签，则返回 null。

另请参见：

setLabel(java.lang.String)

---

setLabel

```
public void setLabel(String label)
```

将按钮的标签设置为指定的字符串。

参数：

label - 新的标签，如果按钮没有标签，则为 null。

另请参见：

getLabel()

setActionCommand

public void setActionCommand(String command)

设置此按钮激发的动作事件的命令名称。在默认情况下，此动作命令设置为与按钮标签相匹配。

参数：

command - 用于设置按钮动作命令的字符串。如果字符串为 null，则动作命令设置为与按钮标签相匹配。

从以下版本开始：

JDK1.1

另请参见：

ActionEvent

getActionCommand

public String getActionCommand()

返回此按钮激发的动作事件的命令名称。如果命令名称为 null（默认），则此方法返回按钮的标签。

addActionListener

public void addActionListener(ActionListener l)

添加指定的动作侦听器，以接收发自此按钮的动作事件。当用户在此按钮上按下或释放鼠标时，发生动作事件。如果 l 为 null，则不抛出任何异常，也不执行任何动作。

有关 AWT 的线程模型的细节信息，请参阅 AWT 线程问题。

参数：

l - 动作侦听器

从以下版本开始：

JDK1.1

另请参见：
removeActionListener(java.awt.event.ActionListener),
getActionListeners(),ActionListener

removeActionListener

public void removeActionListener(ActionListener l)

移除指定的动作侦听器，以便它不再接收发自此按钮的动作事件。当用户在此按钮上按下或释放鼠标时，发生动作事件。如果 l 为 null，则不抛出任何异常，也不执行任何动作。

有关 AWT 的线程模型的细节信息，请参阅 AWT 线程问题。

参数：
l - 动作侦听器
从以下版本开始：
JDK1.1
另请参见：
addActionListener(java.awt.event.ActionListener),
getActionListeners(),ActionListener

getActionListeners

public ActionListener[] getActionListeners()

返回在此按钮上注册的所有动作侦听器的一个数组。
返回：
此按钮的所有 ActionListener；如果当前没有注册任何动作侦听器，则返回一个空数组。
从以下版本开始：
1.4
另请参见：
addActionListener(java.awt.event.ActionListener),
removeActionListener(java.awt.event.ActionListener),
ActionListener

getListeners

public <T extends EventListener> T[] getListeners(Class<T> listenerType)

返回当前在此 Button 上注册为 *Foo*Listener 的所有对象的数组。*Foo*Listener 是用 add*Foo*Listener 方法注册的。

可以使用 class 字面值（如 *Foo*Listener.class）来指定 listenerType 参数。例如，可以使用以下代码来查询 Button b 的动作侦听器：

```
ActionListener[] als = (ActionListener[])(b.getListeners(ActionListener.class));
```

如果不存在这样的侦听器，则此方法将返回一个空数组。

覆盖：

类 Component 中的 getListeners

参数：

listenerType - 请求的侦听器类型；此参数应该指定一个从 java.util.EventListener 继承的接口

返回：

在此按钮上作为 *Foo*Listener 注册的所有对象的数组；如果尚未添加这样的侦听器，则返回一个空数组

抛出：

ClassCastException - 如果 listenerType 未指定一个实现 java.util.EventListener 的类或接口

从以下版本开始：

1.3

另请参见：

getActionListeners()

---

processEvent

protected void processEvent(AWTEvent e)

处理此按钮上的事件。如果事件是 ActionEvent 的一个实例，则此方法将调用 processActionEvent 方法。否则，它调用超类的 processEvent 方法。

注意，如果事件参数为 null，则行为未指定，并可能导致一个异常。

覆盖：

类 Component 中的 processEvent

参数：

e - 事件

从以下版本开始：

JDK1.1

另请参见：

ActionEvent,processActionEvent(java.awt.event.ActionEvent)

processActionEvent

protected void processActionEvent(ActionEvent e)

处理发生在此按钮上的动作事件，方法是将这些事件指派给所有已注册的 ActionListener 对象。

只有启用了此按钮的动作事件，此方法才有可能被调用。当发生以下事件之一时会激活动作事件：

- 通过 addActionListener 注册了 ActionListener 对象。
- 通过 enableEvents 激活了动作事件。

注意，如果事件参数为 null，则行为未指定，并可能导致一个异常。

参数：

e - 动作事件

从以下版本开始：

JDK1.1

另请参见：

ActionListener, addActionListener(java.awt.event.ActionListener), Component.enableEvents(long)

paramString

protected String paramString()

返回此 Button 状态的字符串表示形式。此方法仅在进行调试的时候使用，对于这两个实现，返回的字符串的内容和格式可能有所不同。返回的字符串可能为空，但不

可能为 null。

覆盖：

类 Component 中的 paramString

返回：

此按钮的参数字符串

getAccessibleContext

```
public AccessibleContext getAccessibleContext()
```

获取与此 Button 相关的 AccessibleContext。对于按钮，AccessibleContext 采用 AccessibleAWTButton 的形式。如果有必要，创建一个新的 AccessibleAWTButton 实例。

指定者：

接口 Accessible 中的 getAccessibleContext

覆盖：

类 Component 中的 getAccessibleContext

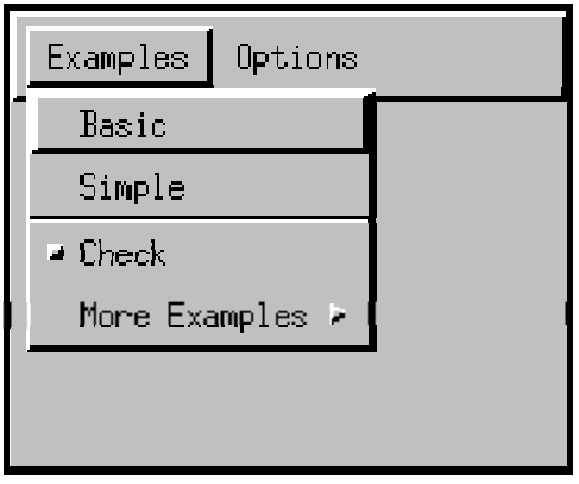
返回：

一个 AccessibleAWTButton，它将充当此 Button 的 AccessibleContext

CheckboxMenuItem 类

此类表示一个可包括在菜单中的复选框。选择菜单中的复选框可以将其状态从 "开" 更改为 "关" 或者从 "关" 更改为 "开"。

下图描绘了一个包含 CheckBoxMenuItem 实例的菜单：



Check 项显示了一个处于 "关" 状态的复选框菜单项。

当选定一个复选框菜单项时，AWT 将一个项事件发送到该项。因为该事件是 ItemEvent 的一个实例，所以 processEvent 方法检查该事件，同时将它传递给 processItemEvent。后一种方法将该事件重定向到任何为关注此菜单项生成的项事件而注册的 ItemListener 对象。

构造方法详细信息

CheckboxMenuItem

```
public CheckboxMenuItem()
                    throws HeadlessException
```

创建具有空标签的复选框菜单项。菜单项的状态初始设置为 "关"。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：

JDK1.1

另请参见：

GraphicsEnvironment.isHeadless()

CheckboxMenuItem

```
public CheckboxMenuItem(String label)
```

throws HeadlessException

创建具有指定标签的复选框菜单项。菜单项的状态初始设置为 "关"。

参数：

label - 复选框菜单项的字符串标签，对于没有标签的菜单项，该参数为 null。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless()

CheckboxMenuItem

public CheckboxMenuItem(String label,
boolean state)
throws HeadlessException

创建 具有指定标签和状态的复选框菜单项。

参数：

label - 复选框菜单项的字符串标签，对于没有标签的菜单项，该参数为 null。

state - 菜单项的最初状态，其中 true 指示 "开"，false 指示 "关"。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

方法详细信息

addNotify

public void addNotify()

创建复选框项的同位体。此同位体允许更改复选框外观，而不更改其功能。大多数应用程序并不直接调用此方法。

覆盖：

类 MenuItem 中的 addNotify

另请参见：

Toolkit.createCheckboxMenuItem(java.awt.CheckboxMenuItem),
Component.getToolkit()

getState

public boolean getState()

确定此复选框菜单项的状态是 "开" 还是 "关"。

返回：

复选框菜单项的状态，其中 true 指示 "开"，false 指示 "关"

另请参见：

setState(boolean)

setState

public void setState(boolean b)

将此复选框菜单项设置为指定的状态。boolean 值 true 指示 "开"，而 false 指示 "关"。

注意，此方法应主要用于初始化复选框菜单项的状态。以编程方式设置复选框菜单项的状态*不会* 触发 ItemEvent。触发 ItemEvent 的唯一方式是通过用户交互。

参数：

b - 如果复选框菜单项处于打开状态，则该参数为 true，否则为 false

另请参见：

getState()

getSelectedObjects

public Object[] getSelectedObjects()

返回一个数组（长度为 1），它包含复选框菜单项的标签，如果没有选中复选框，则返回 null。

指定者：

接口 ItemSelectable 中的 getSelectedObjects

另请参见：

ItemSelectable

addItemListener

public void addItemListener(ItemListener l)

添加指定的项侦听器，以接收发自此复选框菜单项的项事件。发送项事件，以响应用户动作，但不响应对 setState() 的调用。如果 l 为 null，则不抛出任何异常，也不执行任何动作。有关 AWT 的线程模型的细节信息，请参阅 AWT 线程问题。

指定者：

接口 ItemSelectable 中的 addItemListener

参数：

l - 项侦听器

从以下版本开始：

JDK1.1

另请参见：

removeItemListener(java.awt.event.ItemListener), getItemListeners(), setState(boolean), ItemEvent, ItemListener

removeItemListener

public void removeItemListener(ItemListener l)

移除指定的项侦听器，以便它不再接收发自此复选框菜单项的项事件。如果 l 为 null，则不抛出任何异常，也不执行任何动作。有关 AWT 的线程模型的细节信息，请参阅 AWT 线程问题。

指定者：

接口 ItemSelectable 中的 removeItemListener

参数：

l - 项侦听器

从以下版本开始：

JDK1.1

另请参见：

addItemListener(java.awt.event.ItemListener), getItemListeners(), ItemEvent, ItemListener

getItemListeners

public ItemListener[] getItemListeners()

返回在此复选框菜单项上注册的所有项侦听器组成的数组。

返回：

此复选框菜单的所有 ItemListener；如果当前没有已注册的项侦听器，则返回一个空数组。

从以下版本开始：

1.4

另请参见：

addItemListener(java.awt.event.ItemListener), removeItemListener(java.awt.event.ItemListener), ItemEvent, ItemListener

getListeners

public <T extends EventListener> T[] getListeners(Class<T> listenerType)

返回当前在此 CheckboxMenuItem 上注册为 *Foo*Listener 的所有对象组成的数组。*Foo*Listener 是用 add*Foo*Listener 方法注册的。

可以使用 class 字面值（如 *Foo*Listener.class）来指定 listenerType 参数。例如，可以通过以下代码查询 CheckboxMenuItem c，以获得它的项侦听器：

```
ItemListener[] ils =
(ItemListener[])(c.getListeners(ItemListener.class));
```

如果不存在这样的侦听器，则此方法将返回一个空数组。

覆盖：

类 MenuItem 中的 getListeners

参数：

listenerType - 请求的侦听器类型；此参数应该指定一个从 java.util.EventListener 继承的接口

返回：

在此复选框菜单上作为 *Foo*Listener 注册的所有对象组成的数组；如果尚未添加这样的侦听器，则返回一个空数组

抛出：

ClassCastException - 如果 listenerType 未指定一个实现 java.util.EventListener 的类或接口

从以下版本开始：
1.3
另请参见：
getItemListeners()

processEvent

protected void processEvent(AWTEvent e)

处理此复选框菜单项上的事件。如果事件是 ItemEvent 的一个实例，则此方法将调用 processItemEvent 方法。如果事件不是一个项事件，则它将调用超类的 processEvent。

复选框菜单项目前只支持项事件。

注意，如果事件参数为 null，则行为是不确定的，并可能导致一个异常。

覆盖：
类 MenuItem 中的 processEvent
参数：
e - 事件
从以下版本开始：
JDK1.1
另请参见：
ItemEvent,processItemEvent(java.awt.event.ItemEvent)

processItemEvent

protected void processItemEvent(ItemEvent e)

处理发生在此复选框菜单项上的项事件，方法是将这些事件指派给所有已注册的 ItemListener 对象。

只有此菜单项启用了项事件，才会调用此方法。当出现以下情况时，将启用项事件：

- 通过 addItemListener 注册了 ItemListener 对象。

- 通过 enableEvents 启用了项事件。

注意，如果事件参数为 null，则行为是不确定的，并可能导致异常。

参数：
e - 项事件
从以下版本开始：
JDK1.1
另请参见：
ItemEvent, ItemListener, addItemListener(java.awt.event.ItemListener), MenuItem.enableEvents(long)

---

paramString

public String paramString()

返回表示此 CheckBoxMenuItem 状态的字符串。此方法仅在进行调试的时候使用，对于这两个实现，返回的字符串的内容和格式可能有所不同。返回的字符串可能为空，但不可能为 null。
覆盖：
类 MenuItem 中的 paramString
返回：
此复选框菜单项的参数字符串

---

getAccessibleContext

public AccessibleContext getAccessibleContext()

获取与此 CheckboxMenuItem 相关的 AccessibleContext。对于复选框菜单来说，AccessibleContext 采用 AccessibleAWTCheckboxMenuItem 的形式。如果必要，可以创建一个新的 AccessibleAWTCheckboxMenuItem。
指定者：
接口 Accessible 中的 getAccessibleContext
覆盖：
类 MenuItem 中的 getAccessibleContext

返回：

一个 AccessibleAWTCheckboxMenuItem，它充当此 CheckboxMenuItem 的 AccessibleContext

Choice 类

Choice 类表示一个弹出式选择菜单。当前的选择显示为菜单的标题。

下列代码示例产生了一个弹出式菜单：

```
Choice ColorChooser = new Choice();
ColorChooser.add("Green");
ColorChooser.add("Red");
ColorChooser.add("Blue");
```

在将此选择菜单添加到一个面板中后，它显示为以下正常状态：

在这个图像中，"Green" 为当前选择。在对象上按下鼠标按键，这将显示一个菜单，菜单的当前选择被高亮显示。

一些本机平台不支持任意调整 Choice 组件的大小，setSize()/getSize() 的行为受到这类限制的约束。本机 GUI Choice 组件的大小常受到一些属性的限制，比如字体大小和 Choice 中包含的项的长度。

构造方法详细信息

Choice

```
public Choice()
        throws HeadlessException
```

创建一个新的选择菜单。最初，此菜单中没有任何项。

默认情况下，在用户通过调用 select 方法之一进行不同的选择之前，给选择菜单添加的第一个项将成为选定项。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

方法详细信息

addNotify

```
public void addNotify()
```

创建 Choice 的同位体。此同位体允许更改 Choice 的外观，而不更改其功能。

覆盖：

类 Component 中的 addNotify

另请参见：

Toolkit.createChoice(java.awt.Choice),Component.getToolkit()

---

getItemCount

```
public int getItemCount()
```

返回此 Choice 菜单中项的数量。

返回：

此 Choice 菜单中项的数量

从以下版本开始：

JDK1.1

另请参见：

getItem(int)

---

countItems

```
@Deprecated
public int countItems()
```

已过时。 *从 JDK version 1.1 开始，由 getItemCount() 取代。*

---

getItem

public String getItem(int index)

获取此 Choice 菜单中指定索引上的字符串。

参数：

index - 起始索引

另请参见：

getItemCount()

add

public void add(String item)

将一个项添加到此 Choice 菜单中。

参数：

item - 要添加的项

抛出：

NullPointerException - 如果这个项的值为 null

从以下版本开始：

JDK1.1

addItem

public void addItem(String item)

如果对于 Java 2 platform v1.1 已过时，请使用 add 方法替换。

将一个项添加到此 Choice 菜单中。

参数：

item - 要添加的菜单项

抛出：

NullPointerException - 如果这个项的值等于 null

insert

```
public void insert(String item,
                   int index)
```

将菜单项插入此选择的指定位置上。如果某个索引上存在的项大于等于 index，则将它上移一位，以容纳新的项。如果 index 大于等于此选择中项的数目，则将 item 添加到此选择的结尾处。

如果此项是第一个添加到该选择中的项，则这个项将成为选定项。否则，如果选定项是上移项中的一个，则该选择中的第一个项将成为选定项。如果选定项不在这些上移项中，则它仍然是选定项。

参数：

item - 将插入的非 null 项

index - 应该插入项的位置

抛出：

IllegalArgumentException - 如果索引小于 0

remove

```
public void remove(String item)
```

移除 Choice 菜单中第一个出现的 item。如果正被移除的项是目前选定的项，则该选择中的第一个项将成为选定项。否则，当前选定项仍然是选定项（并且选定的索引被相应地更新）。

参数：

item - 将从此 Choice 菜单中移除的项

抛出：

IllegalArgumentException - 如果该项不在此选择菜单中

从以下版本开始：

JDK1.1

remove

public void remove(int position)

从选择菜单的指定位置上移除一个项。如果正被移除的项是目前选定的项，则该选择中的第一个项将成为选定项。否则，当前选定项将仍然是选定项（并且选定的索引被相应地更新）。

参数：

position - 项的位置

抛出：

IndexOutOfBoundsException - 如果指定位置超出边界

从以下版本开始：

JDK1.1

---

removeAll

public void removeAll()

从选择菜单中移除所有的项。

从以下版本开始：

JDK1.1

另请参见：

remove(java.lang.String)

---

getSelectedItem

public String getSelectedItem()

获取当前选择的字符串表示形式。

返回：

此选择菜单中当前选定项的字符串表示形式

另请参见：

getSelectedIndex()

---

getSelectedObjects

public Object[] getSelectedObjects()

返回包含当前选定项的数组（长度为 1）。如果此选择菜单中没项，则返回 null。

指定者：

接口 ItemSelectable 中的 getSelectedObjects

另请参见：

ItemSelectable

---

getSelectedIndex

```
public int getSelectedIndex()
```

返回当前选定项的索引。如果没有选定任何内容，则返回 -1。

返回：

当前选定项的索引，如果目前没有选定任何内容，则返回 -1

另请参见：

getSelectedItem()

---

select

```
public void select(int pos)
```

将此 Choice 菜单中的选定项设置为指定位置上的项。

注意，此方法主要应该用于对此组件中某个项的选择的初始化。以编程方式调用此方法*不* 会触发 ItemEvent。触发 ItemEvent 的唯一方式是通过用户交互。

参数：

pos - 指定项的位置

抛出：

IllegalArgumentException - 如果指定位置大于项的数量或小于零

另请参见：

getSelectedItem(),getSelectedIndex()

---

select

public void select(String str)

将此 Choice 菜单中的选定项设置为其名称等于指定字符串的项。如果有多个项与指定的字符串匹配（相等），则选择具有最小索引的那一个项。

注意，此方法主要应该用于初始化对此组件中某个项的选择。以编程方式调用此方法*不* 会触发 ItemEvent。触发 ItemEvent 的唯一方式是通过用户交互。

参数：

str - 指定的字符串

另请参见：

getSelectedItem(),getSelectedIndex()

addItemListener

public void addItemListener(ItemListener l)

添加指定的项侦听器，以接收发自此 Choice 菜单的项事件。通过发送项事件来响应用户输入，但不响应对 select 的调用。如果 l 为 null，则不会抛出异常，并且不执行任何动作。有关 AWT 的线程模型的细节信息，请参阅 AWT 线程问题。

指定者：

接口 ItemSelectable 中的 addItemListener

参数：

l - 项侦听器

从以下版本开始：

JDK1.1

另请参见：

removeItemListener(java.awt.event.ItemListener),
getItemListeners(),select(int),ItemEvent,ItemListener

removeItemListener

public void removeItemListener(ItemListener l)

移除指定的项侦听器，以便它不再接收发自此 Choice 菜单的项事件。如果 l 为 null，则不会抛出异常，并且不执行任何动作。有关 AWT 的线程模型的细节信息，

请参阅 AWT 线程问题。

指定者：

接口 ItemSelectable 中的 removeItemListener

参数：

l - 项侦听器

从以下版本开始：

JDK1.1

另请参见：

addItemListener(java.awt.event.ItemListener), getItemListeners(), ItemEvent,ItemListener

---

getItemListeners

```
public ItemListener[] getItemListeners()
```

返回已在此选择上注册的所有项侦听器组成的数组。

返回：

此选择的所有 ItemListener，如果没有当前已注册的项侦听器，则返回一个空数组

从以下版本开始：

1.4

另请参见：

addItemListener(java.awt.event.ItemListener), removeItemListener(java.awt.event.ItemListener), ItemEvent, ItemListener

---

getListeners

```
public <T extends EventListener> T[] getListeners(Class<T> listenerType)
```

返回目前已在此 Choice 上注册为 *Foo*Listener 的所有对象组成的数组。*Foo*Listener 是使用 add*Foo*Listener 方法注册的。

可以用一个 class 字面值（比如 *Foo*Listener.class）指定 listenerType 参数。例如，可以使用以下代码查询 Choice c，以获得它的项侦听器：

```
ItemListener[] ils = (ItemListener[])(c.getListeners(ItemListener.class));
```

如果不存在这样的侦听器，则此方法将返回一个空数组。

覆盖：

类 Component 中的 getListeners

参数：

listenerType - 所请求侦听器的类型；该参数应该指定一个从 java.util.EventListener 遗传下来的接口

返回：

在此选择上注册为 *Foo*Listener 的所有对象组成的数组，如果没有添加这样的侦听器，则返回一个空数组

抛出：

ClassCastException - 如果 listenerType 没有指定实现 java.util.EventListener 的类或接口

从以下版本开始：

1.3

另请参见：

getItemListeners()

processEvent

```
protected void processEvent(AWTEvent e)
```

处理关于此选择的事件。如果事件是 ItemEvent 的一个实例，则此方法将调用 processItemEvent 方法。否则，它将调用其超类的 processEvent 方法。

注意，如果事件参数为 null，则行为是不确定的，并且可能导致异常。

覆盖：

类 Component 中的 processEvent

参数：

e - 事件

从以下版本开始：

JDK1.1

另请参见：

ItemEvent,processItemEvent(java.awt.event.ItemEvent)

processItemEvent

protected void processItemEvent(ItemEvent e)

处理发生在此 Choice 菜单上的项事件，实现方式是将这些事件指派给所有已注册的 ItemListener 对象。

只有为此组件启用项事件之后，才调用此方法。项事件是在出现以下情况之一时启用：

- ItemListener 对象是通过 addItemListener 注册的。
- 可以通过 enableEvents 启用项事件。

注意，如果事件参数为 null，则行为是不确定的，并且可能导致异常。

参数：

e - 项事件

从以下版本开始：

JDK1.1

另请参见：

ItemEvent, ItemListener, addItemListener(ItemListener), Component.enableEvents(long)

paramString

protected String paramString()

返回表示此 Choice 菜单的状态的字符串。此方法仅在进行调试的时候使用，对于这两个实现，所返回字符串的内容和格式可能有所不同。返回的字符串可能为空，但不可能为 null。

覆盖：

类 Component 中的 paramString

返回：

此 Choice 菜单的参数字符串

getAccessibleContext

public AccessibleContext getAccessibleContext()

获取与此 Choice 相关的 AccessibleContext。对于 Choice 组件，AccessibleContext 采用的是 AccessibleAWTChoice 的形式。如有必要，可以创建一个新的 AccessibleAWTChoice 实例。

指定者：

接口 Accessible 中的 getAccessibleContext

覆盖：

类 Component 中的 getAccessibleContext

返回：

充当此 Choice 的 AccessibleContext 的 AccessibleAWTChoice

Dialog 类

Dialog 是一个带标题和边界的顶层窗口，边界一般用于从用户处获得某种形式的输入。Dialog 的大小包括边界所指定的任何区域。边界区的维度可以使用 getInsets 方法获得，但是，由于这些维度是依赖于平台的，因此只有通过调用 pack 或 show 将 dialog 设置为可显示的，才能获得有效的 insets 值。由于 dialog 的总大小包括了边界区，因此边界有效地模糊了 dialog 的部分区域，约束了可用于在矩形中呈现或显示子部件的区域，矩形左上角的位置为 (insets.left, insets.top)，宽度为 width - (insets.left + insets.right)，长度为 height - (insets.top + insets.bottom)。

Dialog 的默认布局为 BorderLayout。

Dialog 可以使用 setUndecorated 关闭本机装饰（例如 Frame & Titlebar）。只有在 dialog 不是 displayable 时才能完成此操作。

在构造 dialog 时，dialog 可以拥有另一个窗口作为它自己的窗口。当可见的 dialog 的所有者窗口被最小化时，dialog 会自动隐藏为对用户不可见。当所有者窗口被还原时，dialog 重新又变为用户可见的。

在多屏幕环境中，可以在不同于其所有者的屏幕设备上创建 Dialog。

字段详细信息

DEFAULT_MODALITY_TYPE

public static final Dialog.ModalityType DEFAULT_MODALITY_TYPE

有模式 dialog 的默认模式类型。默认模式类型是 APPLICATION_MODAL。调用旧式 setModal(true) 等效于调用 setModalityType(DEFAULT_MODALITY_TYPE)。

构造方法详细信息

Dialog

```
public Dialog(Frame owner)
```

构造一个最初不可见的、无模式的 Dialog，它带有指定所有者 Frame 和一个空标题。

参数：

owner - dialog 的所有者，如果此 dialog 没有所有者，则该参数为 null

抛出：

IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自某一屏幕设备

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

另请参见：

GraphicsEnvironment.isHeadless(), Component.setSize(int, int), Component.setVisible(boolean)

Dialog

```
public Dialog(Frame owner,
              boolean modal)
```

构造一个最初不可见的 Dialog，它带有指定所有者 Frame、模式和一个空标题。

参数：

owner - dialog 的所有者，如果此 dialog 没有所有者，则该参数为 null

modal - 指定在显示的时候是否阻止用户将内容输入到其他顶级窗口中。如果该参数为 false，则 dialog 是 MODELESS；如果该参数为 true，则模式类型属性被设置为 DEFAULT_MODALITY_TYPE

抛出：

IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自某一屏幕设备

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

另请参见：
Dialog.ModalityType, Dialog.ModalityType.MODELESS, DEFAULT_MODALITY_TYPE, setModal(boolean), setModalityType(java.awt.Dialog.ModalityType), GraphicsEnvironment.isHeadless()

Dialog

```
public Dialog(Frame owner,
              String title)
```

构造一个最初不可见的、无模式的 Dialog，它带有指定的所有者 Frame 和标题。
参数：
owner - dialog 的所有者，如果此 dialog 没有所有者，则该参数为 null
title - dialog 的标题，如果此 dialog 没有标题，则该参数为 null
抛出：
IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自某一屏幕设备
HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时
另请参见：
GraphicsEnvironment.isHeadless(), Component.setSize(int, int), Component.setVisible(boolean)

Dialog

```
public Dialog(Frame owner,
              String title,
              boolean modal)
```

构造一个最初不可见的 Dialog，它带有指定的所有者 Frame、标题和模式。
参数：
owner - dialog 的所有者，如果此 dialog 没有所有者，则该参数为 null
title - dialog 的标题，如果此 dialog 没有标题，则该参数为 null
modal - 指定在显示的时候是否阻止用户将内容输入到其他顶级窗口中。如果该参数为 false，则 dialog 是 MODELESS；如果该参数为 true，则模式类型属性被设置为 DEFAULT_MODALITY_TYPE

抛出：

IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自某一屏幕设备

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

另请参见：

Dialog.ModalityType, Dialog.ModalityType.MODELESS, DEFAULT_MODALITY_TYPE, setModal(boolean), setModalityType(java.awt.Dialog.ModalityType), GraphicsEnvironment.isHeadless(), Component.setSize(int, int), Component.setVisible(boolean)

Dialog

```
public Dialog(Frame owner,
              String title,
              boolean modal,
              GraphicsConfiguration gc)
```

构造一个最初不可见的 Dialog，它带有指定的所有者 Frame、标题、模式和 GraphicsConfiguration。

参数：

owner - dialog 的所有者，如果此 dialog 没有所有者，则该参数为 null

title - dialog 的标题，如果此 dialog 没有标题，则该参数为 null

modal - 指定在显示的时候是否阻止用户将内容输入到其他顶级窗口中。如果该参数为 false，则 dialog 是 MODELESS；如果该参数为 true，则模式类型属性被设置为 DEFAULT_MODALITY_TYPE

gc - 目标屏幕设备的 GraphicsConfiguration；如果该参数为 null，则假定默认系统为 GraphicsConfiguration

抛出：

IllegalArgumentException - 如果 gc 不是来自某一屏幕设备

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

从以下版本开始：

1.4

另请参见：

Dialog.ModalityType, Dialog.ModalityType.MODELESS, DEFAULT_MODALITY_TYPE, setModal(boolean), setModalityType(java.awt.Dialog.ModalityType),

GraphicsEnvironment.isHeadless(), Component.setSize(int, int), Component.setVisible(boolean)

Dialog

public Dialog(Dialog owner)

构造一个最初不可见的、无模式的 Dialog，它带有指定所有者 Dialog 和一个空标题。

参数：

owner - dialog 的所有者，如果此 dialog 没有所有者，则该参数为 null

抛出：

IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自某一屏幕设备

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

从以下版本开始：

1.2

另请参见：

GraphicsEnvironment.isHeadless()

Dialog

public Dialog(Dialog owner,
String title)

构造一个最初不可见的、无模式的 Dialog，它带有指定的所有者 Dialog 和标题。

参数：

owner - dialog 的所有者，如果此 dialog 没有所有者，则该参数为 null

title - dialog 的标题，如果此 dialog 没有标题，则该参数为 null

抛出：

IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自某一屏幕设备

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

从以下版本开始：

1.2

另请参见：

GraphicsEnvironment.isHeadless()

Dialog

```
public Dialog(Dialog owner,
              String title,
              boolean modal)
```

构造一个最初不可见的 Dialog，它带有指定的所有者 Dialog、标题和模式。

参数：

owner - dialog 的所有者，如果此 dialog 没有所有者，则该参数为 null

title - dialog 的标题，如果此 dialog 没有标题，则该参数为 null

modal - 指定在显示的时候是否阻止用户将内容输入到其他顶级窗口中。如果该参数为 false，则 dialog 是 MODELESS；如果该参数为 true，则模式类型属性被设置为 DEFAULT_MODALITY_TYPE

抛出：

IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自某一屏幕设备

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

从以下版本开始：

1.2

另请参见：

Dialog.ModalityType, Dialog.ModalityType.MODELESS, DEFAULT_MODALITY_TYPE, setModal(boolean), setModalityType(java.awt.Dialog.ModalityType), GraphicsEnvironment.isHeadless()

Dialog

```
public Dialog(Dialog owner,
              String title,
              boolean modal,
              GraphicsConfiguration gc)
```

构造一个最初不可见的 Dialog，它带有指定的所有者 Dialog、标题、模式和 GraphicsConfiguration。

参数：

owner - dialog 的所有者，如果此 dialog 没有所有者，则该参数为 null

title - dialog 的标题，如果此 dialog 没有标题，则该参数为 null

modal - 指定在显示的时候是否阻止用户将内容输入到其他顶级窗口中。如果该参数为 false，则 dialog 是 MODELESS；如果该参数为 true，则模式类型属性被设置为 DEFAULT_MODALITY_TYPE

gc - 目标屏幕设备的 GraphicsConfiguration；如果该参数为 null，则假定默认系统为 GraphicsConfiguration

抛出：

IllegalArgumentException - 如果 gc 不是来自某一屏幕设备

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

从以下版本开始：

1.4

另请参见：

Dialog.ModalityType, Dialog.ModalityType.MODELESS, DEFAULT_MODALITY_TYPE, setModal(boolean), setModalityType(java.awt.Dialog.ModalityType), GraphicsEnvironment.isHeadless(), Component.setSize(int, int), Component.setVisible(boolean)

Dialog

public Dialog(Window owner)

构造一个最初不可见的、无模式的 Dialog，它带有指定所有者 Window 和一个空标题。

参数：

owner - dialog 的所有者。该所有者必须是 Dialog、Frame 以及它们的任何后代的实例，或者是 null

抛出：

IllegalArgumentException - 如果 owner 不是 Dialog 或 Frame 实例

IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自屏幕设备

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

从以下版本开始：

1.6

另请参见：

GraphicsEnvironment.isHeadless()

Dialog

```
public Dialog(Window owner,
              String title)
```

构造一个最初不可见的、无模式的 Dialog，它带有指定的所有者 Window 和标题。

参数：

owner - dialog 的所有者。该所有者必须是 Dialog、Frame 以及它们的任何后代的实例，或者是 null

title - dialog 的标题，如果此 dialog 没有标题，则该参数为 null

抛出：

IllegalArgumentException - 如果 owner 不是 Dialog 或 Frame 实例

IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自屏幕设备

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

从以下版本开始：

1.6

另请参见：

GraphicsEnvironment.isHeadless()

Dialog

```
public Dialog(Window owner,
              Dialog.ModalityType modalityType)
```

构造一个最初不可见的 Dialog，它带有指定所有者 Window、模式和一个空标题。

参数：

owner - dialog 的所有者。该所有者必须是 Dialog、Frame 以及它们的任何后代的实例，或者是 null

modalityType - 指定 dialog 是否阻止用户在显示的时候将内容输入其他窗口。null 值和不受支持的模式类型等效于 MODELESS

抛出：

IllegalArgumentException - 如果 owner 不是 Dialog 或 Frame 实例

IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自

某个屏幕设备
HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时
SecurityException - 如果调用线程没有使用给定 modalityType 创建有模式 dialog 的权限
从以下版本开始：
1.6
另请参见：
Dialog.ModalityType, setModal(boolean), setModalityType(java.awt.Dialog.ModalityType), GraphicsEnvironment.isHeadless(), Toolkit.isModalityTypeSupported(java.awt.Dialog.ModalityType)

Dialog

```
public Dialog(Window owner,
              String title,
              Dialog.ModalityType modalityType)
```

构造一个最初不可见的 Dialog，它带有指定的所有者 Window、标题和模式。
参数：
owner - dialog 的所有者。该所有者必须是 Dialog、Frame 以及它们的任何后代的实例，或者是 null
title - dialog 的标题，如果此 dialog 没有标题，则该参数为 null
modalityType - 指定 dialog 是否阻止用户在显示的时候将内容输入其他窗口中。null 值和不受支持的模式类型等效于 MODELESS
抛出：
IllegalArgumentException - 如果 owner 不是 Dialog 或 Frame 实例
IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自某个屏幕设备
HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时
SecurityException - 如果调用线程没有使用给定 modalityType 创建有模式 dialog 的权限
从以下版本开始：
1.6
另请参见：
Dialog.ModalityType, setModal(boolean), setModalityType(java.awt.Dialog.ModalityType),

GraphicsEnvironment.isHeadless(),
Toolkit.isModalityTypeSupported(java.awt.Dialog.ModalityType)

Dialog

```
public Dialog(Window owner,
              String title,
              Dialog.ModalityType modalityType,
              GraphicsConfiguration gc)
```

构造一个最初不可见的 Dialog，它带有指定的所有者 Window、标题、模式和 GraphicsConfiguration。

参数：

owner - dialog 的所有者。该所有者必须是 Dialog、Frame 以及它们的任何后代的实例，或者是 null

title - dialog 的标题，如果此 dialog 没有标题，则该参数为 null

modalityType - 指定 dialog 是否阻止用户在显示的时候将内容输入其他窗口。null 值和不受支持的模式类型等效于 MODELESS

gc - 目标屏幕设备的 GraphicsConfiguration；如果该参数为 null，则假定默认系统为 GraphicsConfiguration

抛出：

IllegalArgumentException - 如果 owner 不是 Dialog 或 Frame 实例

IllegalArgumentException - 如果 gc 不是来自某一屏幕设备

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

SecurityException - 如果调用线程没有使用给定 modalityType 创建有模式 dialog 的权限

方法详细信息

addNotify

```
public void addNotify()
```

通过将此 Dialog 连接到本机屏幕资源，从而使其成为可显示的。如果 dialog 是可显示的，则其所有子级也将成为可显示的。由工具包在内部调用此方法，而不应该直接由程序调用。

覆盖：
类 Window 中的 addNotify
另请参见：
Component.isDisplayable(),Window.removeNotify()

isModal

```
public boolean isModal()
```

指出 dialog 是否为有模式的。

此方法已过时，保留它只是为了后向兼容。可以使用 getModalityType() 代替。

返回：
如果此 dialog 窗口是有模式的，则返回 true；否则返回 false
另请参见：
DEFAULT_MODALITY_TYPE, Dialog.ModalityType.MODELESS, setModal(boolean), getModalityType(), setModalityType(java.awt.Dialog.ModalityType)

setModal

```
public void setModal(boolean modal)
```

指定此 dialog 是否应该是有模式的。

此方法已过时，保留它只是为了后向兼容。可以使用 setModalityType() 代替。

注：更改可见 dialog 的模式只在隐藏该 dialog 并再次显示它时产生效果。

参数：
modal - 指定 dialog 是否阻止在显示的时候将内容输入其他窗口；调用 setModal(true) 等效于 setModalityType(Dialog.DEFAULT_MODALITY_TYPE)，而调用

setModal(false) 等效于 setModalityType(Dialog.ModalityType.MODELESS)

从以下版本开始：

1.1

另请参见：

DEFAULT_MODALITY_TYPE, Dialog.ModalityType.MODELESS, isModal(), getModalityType(),setModalityType(java.awt.Dialog.ModalityType)

---

getModalityType

public Dialog.ModalityType getModalityType()

返回此 dialog 的模式类型。

返回：

此 dialog 的模式类型

从以下版本开始：

1.6

另请参见：

setModalityType(java.awt.Dialog.ModalityType)

---

setModalityType

public void setModalityType(Dialog.ModalityType type)

设置此 dialog 的模式类型。有关可能的模式类型，请参阅 ModalityType。

如果给定模式类型不受支持，则使用 MODELESS。您可能想在调用此方法之后调用 getModalityType()，以确保模式类型已被设置。

注：更改可见 dialog 的模式只在隐藏该 dialog 并再次显示它时产生效果。

参数：

type - 指定 dialog 是否阻止在显示的时候将内容输入其他窗口。null 值和不受支持的模式类型等效于 MODELESS

抛出：

SecurityException - 如果调用线程没有使用给定 modalityType 创建有模式 dialog 的权限

从以下版本开始：

1.6

另请参见：

getModalityType(),
Toolkit.isModalityTypeSupported(java.awt.Dialog.ModalityType)

---

getTitle

```
public String getTitle()
```

获取 dialog 的标题。标题显示在 dialog 的边界中。

返回：

此 dialog 窗口的标题。标题可以为 null。

另请参见：

setTitle(java.lang.String)

---

setTitle

```
public void setTitle(String title)
```

设置 Dialog 的标题。

参数：

title - 显示在 dialog 的边界中的标题；null 值会导致一个空标题

另请参见：

getTitle()

---

setVisible

```
public void setVisible(boolean b)
```

根据参数 b 的值显示或隐藏此 Dialog。

覆盖：

类 Window 中的 setVisible

参数：

b - 如果该参数为 true，则使 Dialog 可见，否则隐藏 Dialog。如果 dialog 和/或其所有者尚不可显示，则使它们都变成可显示的。在使 dialog 可见之前，将验证该 dialog。如果该参数为 false，则隐藏 Dialog，并且随后导致返回 setVisible(true)（如果它目前受阻塞）。

对有模式 dialog 的注释。

- setVisible(true)：如果 dialog 尚不可见，则在通过调用 setVisible(false) 或 dispose 隐藏 dialog 之前，此调用不会返回。
- setVisible(false)：隐藏 dialog，然后返回 setVisible(true)（如果它目前受阻塞）。
- 从事件指派线程调用此方法是受允许的，因为在此方法受阻塞时，工具包会确保其他事件不受阻塞。

另请参见：

Window.setVisible(boolean), Window.dispose(), Component.isDisplayable(),Component.validate(),isModal()

---

show

```
@Deprecated
    public void show()
```

已过时。 *从 JDK 版本 1.5 开始，由 setVisible(boolean) 取代。*

使 Dialog 可见。如果 dialog 和/或其所有者尚不可显示，则使它们都变成可显示的。在使 dialog 可见之前，将验证该 dialog。如果 dialog 已经可见，则此方法会使 dialog 显示在前面。

如果 dialog 是有模式的并且尚不可见，则在通过调用 hide 或 dispose 方法隐藏 dialog 之前，此调用不会返回。从事件指派线程显示有模式 dialog 是受允许的，因为调用此方法的线程受阻塞时，工具包会确保另一个事件队列的运行。

覆盖：

类 Window 中的 show

另请参见：

Component.hide(), Component.isDisplayable(), Component.validate(),

isModal(),Window.setVisible(boolean)

hide

@Deprecated
public void hide()

已过时。 *从 JDK 版本 1.5 开始，由 setVisible(boolean) 取代。*
隐藏 Dialog 并随后导致返回 show（如果它目前受阻塞）。
覆盖：
类 Window 中的 hide
另请参见：
Window.show(),Window.dispose(),Window.setVisible(boolean)

toBack

public void toBack()

如果此窗口是可视的，则将此窗口置于后方，如果它是焦点窗口或活动窗口，则会导致丢失焦点或活动状态。

在此虚拟机中，将此窗口放在堆栈顺序的底部，并在其他所有窗口之后显示此窗口。如果此窗口不可见，则不发生任何操作。有些平台不允许其他窗口拥有的窗口出现在其所有者下方。将进行所有尝试来移动此窗口，使其位于堆栈顺序中尽可能靠后的位置；不过，开发人员不应假定此方法在所有情况下都可以将此窗口移到所有其他窗口之下。

由于本机窗口系统多种多样，因此无法保证对焦点窗口和活动窗口的更改能够实现。在此窗口接收 WINDOW_LOST_FOCUS 或 WINDOW_DEACTIVATED 事件之前，开发人员不得假定此窗口不再是焦点窗口或活动窗口。在顶层窗口为焦点窗口的平台上，此方法**可能**导致此窗口不再是焦点状态。在此情况下，此虚拟机中紧跟其后的可作为焦点的窗口将成为焦点窗口。在堆栈顺序通常不影响焦点窗口的平台上，此方法**可能**维持焦点窗口和活动窗口不变。

如果此 dialog 是有模式的并且阻止某些窗口，则所有 dialog 都被发送回来，使它们位于正受阻塞的 dialog 下。

覆盖：

类 Window 中的 toBack

另请参见：

Window.toBack()

isResizable

```
public boolean isResizable()
```

指出此 dialog 是否可以由用户调整大小。默认情况下，所有 dialog 最初都是可调整的。

返回：

如果用户可以调整 dialog 的大小，则返回 true；否则返回 false。

另请参见：

setResizable(boolean)

setResizable

```
public void setResizable(boolean resizable)
```

设置此 dialog 是否可以由用户调整大小。

参数：

resizable - 如果用户可以调整 dialog 的大小，则返回 true；否则返回 false。

另请参见：

isResizable()

setUndecorated

```
public void setUndecorated(boolean undecorated)
```

禁用或启用此 dialog 的装饰。只有在 dialog 不可显示时才调用此方法。

参数：

undecorated - 如果没有启用 dialog 装饰，则为 true；如果启用了 dialog 装饰，则为 false。

抛出：

IllegalComponentStateException - 如果 dialog 是可显示的。

从以下版本开始：

1.4

另请参见：

isUndecorated(),Component.isDisplayable()

---

isUndecorated

```
public boolean isUndecorated()
```

指出此 dialog 是否未装饰。在默认情况下，所有 dialog 最初都是已装饰的。

返回：

如果 dialog 为未装饰，则为 true；否则为 false。

从以下版本开始：

1.4

另请参见：

setUndecorated(boolean)

---

paramString

```
protected String paramString()
```

返回表示此 dialog 状态的字符串。此方法仅用于调试目的，对于这两种实现，返回字符串的内容和格式可能有所不同。返回的字符串可能为空，但不可能为 null。

覆盖：

类 Container 中的 paramString

返回：

此 dialog 窗口的参数字符串。

---

getAccessibleContext

public AccessibleContext getAccessibleContext()

获取与此 Dialog 有关的 AccessibleContext。对于 dialog 来说，AccessibleContext 采用 AccessibleAWTDialog 的某种形式。必要时创建一个新的 AccessibleAWTDialog 实例。

指定者：

接口 Accessible 中的 getAccessibleContext

覆盖：

类 Window 中的 getAccessibleContext

返回：

AccessibleAWTDialog，用作此 Dialog 的 AccessibleContext

Dimension 类

Dimension 类封装单个对象中组件的宽度和高度（精确到整数）。该类与组件的某个属性关联。由 Component 类和 LayoutManager 接口定义的一些方法将返回 Dimension 对象。

通常，width 和 height 的值是非负整数。允许创建 dimension 的构造方法不会阻止您为这些属性设置负值。如果 width 或 height 的值为负，则由其他对象定义的一些方法的行为是不明确的。

字段详细信息

width

public int width

dimension 的宽度，可以使用负值。

从以下版本开始：

1.0

另请参见：

getSize(),setSize(double, double)

---

height

```
public int height
```

dimension 的高度，可以使用负值。

构造方法详细信息

Dimension

```
public Dimension()
```

创建 Dimension 的一个实例（宽度为零，高度为零）。

Dimension

```
public Dimension(Dimension d)
```

创建 Dimension 的一个实例（宽度和高度与指定的 dimension 相同）。

参数：

d - 带有 width 和 height 值的指定 dimension

Dimension

```
public Dimension(int width,
                 int height)
```

构造一个 Dimension，并将其初始化为指定宽度和高度。

参数：

width - 指定宽度

height - 指定高度

方法详细信息

getWidth

```
public double getWidth()
```

返回此 Dimension 的宽度（以 double 精度表示）。

指定者：

类 Dimension2D 中的 getWidth

返回：

此 Dimension 的宽度。

从以下版本开始：

1.2

getHeight

```
public double getHeight()
```

返回此 Dimension 的高度（以 double 精度表示）。

指定者：

类 Dimension2D 中的 getHeight

返回：

此 Dimension 的高度。

从以下版本开始：

1.2

setSize

```
public void setSize(double width,
                    double height)
```

将此 Dimension 对象的大小设置为指定的宽度和高度（以双精度表示）。注意，如果 width 或 height 大于 Integer.MAX_VALUE，则将其重新设置为 Integer.MAX_VALUE。

指定者：

类 Dimension2D 中的 setSize

参数：

width - Dimension 对象的新宽度

height - Dimension 对象的新高度

从以下版本开始：

1.2

getSize

```
public Dimension getSize()
```

获取此 Dimension 对象的大小。包含此方法是出于完整性考虑，它与 Component 定义的 getSize 方法相似。

返回：

此 dimension 的大小，一个具有相同宽度和高度的 Dimension 的新实例

从以下版本开始：

1.1

另请参见：

setSize(double, double),Component.getSize()

setSize

```
public void setSize(Dimension d)
```

将 Dimension 对象的大小设置为指定大小。包含此方法是出于完整性考虑，它与 Component 定义的 setSize 方法相似。

参数：

d - Dimension 对象的新大小

从以下版本开始：

1.1

另请参见：

getSize(),Component.setSize(int, int)

setSize

```
public void setSize(int width,
                    int height)
```

将此 Dimension 对象的大小设置为指定的宽度和高度。包含此方法是出于完整性考虑，它与 Component 定义的 setSize 方法相似。

参数：

width - Dimension 对象的新宽度
height - Dimension 对象的新高度
从以下版本开始：
1.1
另请参见：
getSize(),Component.setSize(int, int)

equals

```
public boolean equals(Object obj)
```

检查两个 dimension 对象是否具有相同的值。
覆盖：
类 Object 中的 equals
参数：
obj - 要与之比较的引用对象。
返回：
如果此对象与 obj 参数相同，则返回 true；否则返回 false。
另请参见：
Object.hashCode(),Hashtable

hashCode

```
public int hashCode()
```

返回此 Dimension 的哈希码。
覆盖：
类 Object 中的 hashCode
返回：
此 Dimension 的哈希码
另请参见：
Object.equals(java.lang.Object),Hashtable

toString

public String toString()

返回此 Dimension 对象的 height 和 width 字段的字符串表示形式。此方法仅用于调试目的，对于这两种实现，返回字符串的内容和格式可能有所不同。返回的字符串可以为空，但不可以为 null。

覆盖：

类 Object 中的 toString

返回：

该 Dimension 对象的字符串表示形式

Event 类

注：Event 类已废弃，只可用于向后兼容。它已经由 AWTEvent 类及其子类所取代。

Event 是一个与平台无关的类，它封装了 Java 1.0 事件模型中来自平台图形用户界面的事件。在 Java 1.1 和以后的版本中，Event 类只针对向后兼容进行维护。此类描述中的信息可以协助程序员将 Java 1.0 程序转换成新的事件模型。

在 Java 1.0 事件模型中，包含 id 字段的事件指出它是什么类型的事件，以及与该事件有关的其他 Event 变量。

对于键盘事件来说，key 所包含的值指出要激活哪个键，modifiers 包含该事件的修饰符。对于 KEY_PRESS 和 KEY_RELEASE 事件的 id，key 的值是该键的 unicode 字符代码。对于 KEY_ACTION 和 KEY_ACTION_RELEASE，key 的值是 Event 类（PGUP、PGDN、F1、F2 等）中所定义的某个动作-键标识符。

字段详细信息

SHIFT_MASK

public static final int SHIFT_MASK

此标志指示当事件发生时按下 Shift 键。

另请参见：

常量字段值

CTRL_MASK

public static final int CTRL_MASK

此标志指示当事件发生时按下 Control 键。

另请参见：

常量字段值

META_MASK

public static final int META_MASK

此标志指示当事件发生时按下 Meta 键。对于鼠标事件，此标志指示按下或释放右边按钮。

另请参见：

常量字段值

ALT_MASK

public static final int ALT_MASK

此标志指示当事件发生时按下 Alt 键。对于鼠标事件，此标志指示按下或释放中间鼠标按钮。

另请参见：

常量字段值

HOME

public static final int HOME

Home 键，一个非 ASCII 操作键。

另请参见：

常量字段值

END

public static final int END

End 键，一个非 ASCII 操作键。

另请参见：

常量字段值

PGUP

public static final int PGUP

Page Up 键，一个非 ASCII 操作键。

另请参见：

常量字段值

PGDN

public static final int PGDN

Page Down 键，一个非 ASCII 操作键。

另请参见：

常量字段值

UP

public static final int UP

Up Arrow 键，一个非 ASCII 操作键。

另请参见：

常量字段值

DOWN

public static final int DOWN

Down Arrow 键，一个非 ASCII 操作键。

另请参见：
常量字段值

LEFT

public static final int LEFT

Left Arrow 键，一个非 ASCII 操作键。
另请参见：
常量字段值

RIGHT

public static final int RIGHT

Right Arrow 键，一个非 ASCII 操作键。
另请参见：
常量字段值

F1

public static final int F1

F1 功能键，一个非 ASCII 操作键。
另请参见：
常量字段值

F2

public static final int F2

F2 功能键，一个非 ASCII 操作键。
另请参见：
常量字段值

F3

public static final int F3

F3 功能键，一个非 ASCII 操作键。

另请参见：

常量字段值

F4

public static final int F4

F4 功能键，一个非 ASCII 操作键。

另请参见：

常量字段值

F5

public static final int F5

F5 功能键，一个非 ASCII 操作键。

另请参见：

常量字段值

F6

public static final int F6

F6 功能键，一个非 ASCII 操作键。

另请参见：

常量字段值

F7

public static final int F7

F7 功能键，一个非 ASCII 操作键。

另请参见：

常量字段值

F8

public static final int F8

F8 功能键，一个非 ASCII 操作键。

另请参见：

常量字段值

F9

public static final int F9

F9 功能键，一个非 ASCII 操作键。

另请参见：

常量字段值

F10

public static final int F10

F10 功能键，一个非 ASCII 操作键。

另请参见：

常量字段值

F11

public static final int F11

F11 功能键，一个非 ASCII 操作键。

另请参见：

常量字段值

F12

public static final int F12

F12 功能键，一个非 ASCII 操作键。

另请参见：

常量字段值

PRINT_SCREEN

public static final int PRINT_SCREEN

Print Screen 键，一个非 ASCII 操作键。

另请参见：

常量字段值

SCROLL_LOCK

public static final int SCROLL_LOCK

Scroll Lock 键，一个非 ASCII 操作键。

另请参见：

常量字段值

CAPS_LOCK

public static final int CAPS_LOCK

Caps Lock 键，一个非 ASCII 操作键。

另请参见：
常量字段值

NUM_LOCK

```
public static final int NUM_LOCK
```

Num Lock 键，一个非 ASCII 操作键。
另请参见：
常量字段值

PAUSE

```
public static final int PAUSE
```

Pause 键，一个非 ASCII 操作键。
另请参见：
常量字段值

INSERT

```
public static final int INSERT
```

Insert 键，一个非 ASCII 操作键。
另请参见：
常量字段值

ENTER

```
public static final int ENTER
```

Enter 键。
另请参见：
常量字段值

BACK_SPACE

public static final int BACK_SPACE

BackSpace 键。

另请参见：

常量字段值

TAB

public static final int TAB

Tab 键。

另请参见：

常量字段值

ESCAPE

public static final int ESCAPE

Escape 键。

另请参见：

常量字段值

DELETE

public static final int DELETE

Delete 键。

另请参见：

常量字段值

WINDOW_DESTROY

public static final int WINDOW_DESTROY

用户要求窗口管理程序关掉窗口。

另请参见：

常量字段值

WINDOW_EXPOSE

public static final int WINDOW_EXPOSE

用户要求窗口管理程序公开窗口。

另请参见：

常量字段值

WINDOW_ICONIFY

public static final int WINDOW_ICONIFY

用户要求窗口管理程序图标化窗口。

另请参见：

常量字段值

WINDOW_DEICONIFY

public static final int WINDOW_DEICONIFY

用户要求窗口管理程序取消窗口图标化。

另请参见：

常量字段值

WINDOW_MOVED

public static final int WINDOW_MOVED

用户要求窗口管理程序移动窗口。

另请参见：

常量字段值

KEY_PRESS

public static final int KEY_PRESS

用户已经按下一个常规键。

另请参见：

常量字段值

KEY_RELEASE

public static final int KEY_RELEASE

用户已经释放一个常规键。

另请参见：

常量字段值

KEY_ACTION

public static final int KEY_ACTION

用户已经按下一个非 ASCII *action* 键。key 字段包含一个值，指示发生在某个操作键上的事件，操作键包括 12 个功能键、箭头（光标）键、Page Up、Page Down、Home、End、Print Screen、Scroll Lock、Caps Lock、Num Lock、Pause 和 Insert。

另请参见：

常量字段值

KEY_ACTION_RELEASE

public static final int KEY_ACTION_RELEASE

用户已经释放一个非 ASCII *action* 键。key 字段包含一个值，指示发生在某个操作键中的事件，操作键包括 12 个功能键、箭头（光标）键、Page Up、Page Down、Home、End、Print Screen、Scroll Lock、Caps Lock、Num Lock、Pause 和 Insert。

另请参见：

常量字段值

MOUSE_DOWN

public static final int MOUSE_DOWN

用户已经按下鼠标按钮。ALT_MASK 标志指示中间按钮已经被按下。META_MASK 标志指示右按钮已经被按下。

另请参见：

ALT_MASK,META_MASK,常量字段值

MOUSE_UP

public static final int MOUSE_UP

用户已经释放鼠标按钮。ALT_MASK 标志指示中间按钮已经被释放。META_MASK 标志指示右边按钮已经被释放。

另请参见：

ALT_MASK,META_MASK,常量字段值

MOUSE_MOVE

public static final int MOUSE_MOVE

鼠标已经移动，没有按钮被按下。

另请参见：

常量字段值

MOUSE_ENTER

public static final int MOUSE_ENTER

鼠标已经进入了一个组件。

另请参见：

常量字段值

MOUSE_EXIT

public static final int MOUSE_EXIT

鼠标已经退出了一个组件。

另请参见：

常量字段值

MOUSE_DRAG

public static final int MOUSE_DRAG

鼠标已经移动，同时按钮被按下。ALT_MASK 标志指示中间按钮正在被按下。META_MASK 标志指示右边按钮正在被按下。

另请参见：

ALT_MASK,META_MASK,常量字段值

SCROLL_LINE_UP

public static final int SCROLL_LINE_UP

用户已经激活了滚动条的 *line up* 区域。

另请参见：

常量字段值

SCROLL_LINE_DOWN

public static final int SCROLL_LINE_DOWN

用户已经激活了滚动条的 *line down* 区域。

另请参见：

常量字段值

SCROLL_PAGE_UP

public static final int SCROLL_PAGE_UP

用户已经激活了滚动条的 *page up* 区域。

另请参见：

常量字段值

SCROLL_PAGE_DOWN

public static final int SCROLL_PAGE_DOWN

用户已经激活了滚动条的 *page down* 区域。

另请参见：

常量字段值

SCROLL_ABSOLUTE

public static final int SCROLL_ABSOLUTE

用户已经将滚动条中的“气泡”（即翻阅标志）移到一个“绝对的”位置，而不是相对最后位置的偏移量。

另请参见：

常量字段值

SCROLL_BEGIN

public static final int SCROLL_BEGIN

滚动开始事件。

另请参见：

常量字段值

SCROLL_END

```
public static final int SCROLL_END
```

滚动结束事件。

另请参见：

常量字段值

LIST_SELECT

```
public static final int LIST_SELECT
```

已经选择了列表中的一项。

另请参见：

常量字段值

LIST_DESELECT

```
public static final int LIST_DESELECT
```

已经取消选择列表中的一项。

另请参见：

常量字段值

ACTION_EVENT

```
public static final int ACTION_EVENT
```

此事件指示用户想要某个事件发生。

另请参见：

常量字段值

LOAD_FILE

public static final int LOAD_FILE

文件加载事件。

另请参见：

常量字段值

SAVE_FILE

public static final int SAVE_FILE

文件保存事件。

另请参见：

常量字段值

GOT_FOCUS

public static final int GOT_FOCUS

组件获得焦点。

另请参见：

常量字段值

LOST_FOCUS

public static final int LOST_FOCUS

组件失去焦点。

另请参见：

常量字段值

target

public Object target

目标组件。这指示事件针对该组件发生，或事件与该组件关联。此对象已经由 AWTEvent.getSource() 取代。

另请参见：

EventObject.getSource()

when

public long when

时间戳。由 InputEvent.getWhen() 取代。

另请参见：

InputEvent.getWhen()

id

public int id

指示事件是什么类型的事件，其他哪个 Event 变量与该事件有关。它已经由 AWTEvent.getID() 取代。

另请参见：

AWTEvent.getID()

x

public int x

事件的 *x* 坐标。由 MouseEvent.getX() 取代。

另请参见：

MouseEvent.getX()

y

public int y

事件的 *y* 坐标。由 MouseEvent.getY() 取代。

另请参见：

MouseEvent.getY()

key

public int key

在键盘事件中被按下的键的键代码。这已经由 KeyEvent.getKeyCode() 取代。

另请参见：

KeyEvent.getKeyCode()

modifiers

public int modifiers

修饰符键的状态。这已经由 InputEvent.getModifiers() 取代。在 java 1.1 中，MouseEvent 和 KeyEvent 是 InputEvent 的子类。

另请参见：

InputEvent.getModifiers()

clickCount

public int clickCount

对于 MOUSE_DOWN 事件来说，此字段指示连续点击的次数。对于其他事件，其值为 0。此字段由 MouseEvent.getClickCount() 取代。

另请参见：

MouseEvent.getClickCount().

arg

```
public Object arg
```

事件的任意参数。此字段的值取决于事件的类型。arg 已经由事件指定的属性取代。

evt

```
public Event evt
```

下一事件。将事件放入到链接列表时设置此字段。这已经由 EventQueue 取代。

构造方法详细信息

Event

```
public Event(Object target,
             long when,
             int id,
             int x,
             int y,
             int key,
             int modifiers,
             Object arg)
```

注：Event 类已废弃，只可用于向后兼容。它已经由 AWTEvent 类及其子类所取代。

创建 Event 的一个实例，具有指定的目标组件、时间戳、事件类型、*x* 和 *y* 坐标、键盘键、修饰符键的状态、参数。

参数：

target - 目标组件。
when - 时间戳。
id - 事件类型。
x - *x* 坐标。
y - *y* 坐标。

key - 在键盘事件中按下的键。
modifiers - 修饰符键的状态。
arg - 指定参数。

Event

```
public Event(Object target,
             long when,
             int id,
             int x,
             int y,
             int key,
             int modifiers)
```

注：Event 类已废弃，只可用于向后兼容。它已经由 AWTEvent 类和其子类所取代。

创建 Event 的一个实例，具有指定的目标组件、时间戳、事件类型、*x* 和 *y* 坐标、键盘键、修饰符键的状态、一个设置为 null 的参数。

参数：
target - 目标组件。
when - 时间戳。
id - 事件类型。
x - *x* 坐标。
y - *y* 坐标。
key - 在键盘事件中按下的键。
modifiers - 修饰符键的状态。

Event

```
public Event(Object target,
             int id,
             Object arg)
```

注：Event 类已废弃，只可用于向后兼容。它已经由 AWTEvent 类和其子类所取代。

使用指定的目标组件、事件类型和参数创建 Event 的一个实例。

参数：
target - 目标组件。
id - 事件类型。
arg - 指定参数。

方法详细信息

translate

```
public void translate(int dx,
                      int dy)
```

注：Event 类已废弃，只可用于向后兼容。它已经由 AWTEvent 类和其子类所取代。

转换此事件，使其 *x* 和 *y* 坐标分别以 *dx* 和 *dy* 递增。

此方法转换与给定组件相关的事件。这至少涉及到将坐标转换成给定组件的本地坐标系。它还可能涉及到在公开事件中转换一个区域

参数：
dx - 要转换 *x* 坐标的距离。
dy - 要转换 *y* 坐标的距离。

---

shiftDown

```
public boolean shiftDown()
```

注：Event 类已废弃，只可用于向后兼容。它已经由 AWTEvent 类和其子类所取代。

检查 Shift 键是否被按下。

返回：
如果该键被按下，则返回 true；否则返回 false。
另请参见：
modifiers,controlDown(),metaDown()

controlDown

public boolean controlDown()

注：Event 类已废弃，只可用于向后兼容。它已经由 AWTEvent 类和其子类所取代。

检查 Control 键是否被按下。

返回：

如果该键被按下，则返回 true；否则返回 false。

另请参见：

modifiers,shiftDown(),metaDown()

metaDown

public boolean metaDown()

注：Event 类已废弃，只可用于向后兼容。它已经由 AWTEvent 类和其子类所取代。

检查 Meta 键是否被按下。

返回：

如果该键被按下，则返回 true；否则返回 false。

另请参见：

modifiers,shiftDown(),controlDown()

paramString

protected String paramString()

注：Event 类已废弃，只可用于向后兼容。它已经由 AWTEvent 类和其子类所取代。

返回此 Event 状态的字符串表示形式。此方法仅用于调试目的，对于这两种实现，返回字符串的内容和格式可能有所不同。返回的字符串可能为空，但不可能为 null。

返回：
此事件的参数字符串

toString

```
public String toString()
```

注：Event 类已废弃，只可用于向后兼容。它已经由 AWTEvent 类和其子类所取代。

返回此事件的值的字符串表示形式。

覆盖：
类 Object 中的 toString
返回：
字符串，表示事件及其成员字段的值。

FileDialog 类

FileDialog 类显示一个对话框窗口，用户可以从中选择文件。

由于它是一个模式对话框，当应用程序调用其 show 方法来显示对话框时，它将阻塞其余应用程序，直到用户选择一个文件。

字段详细信息

LOAD

```
public static final int LOAD
```

此常量值指示文件对话框窗口的作用是查找要读取的文件。
另请参见：
常量字段值

SAVE

public static final int SAVE

此常量值指示文件对话框窗口的作用是查找要写入的文件。

构造方法详细信息

FileDialog

public FileDialog(Frame parent)

创建一个文件对话框，用于加载文件。文件对话框的标题最初是空的。这是 FileDialog(parent, ″″, LOAD) 的一个便捷方法。

参数：

parent - 对话框的所有者

从以下版本开始：

JDK1.1

FileDialog

public FileDialog(Frame parent,
String title)

创建一个具有指定标题的文件对话框窗口，用于加载文件。显示的文件是当前目录中的文件。这是 FileDialog(parent, title, LOAD) 的一个便捷方法。

参数：

parent - 对话框的所有者

title - 对话框的标题

FileDialog

public FileDialog(Frame parent,
String title,
int mode)

创建一个具有指定标题的文件对话框窗口，用于加载或保存文件。

如果 mode 的值为 LOAD，那么文件对话框将查找要读取的文件，所显示的文件是当前目录中的文件。如果 mode 的值为 SAVE，则文件对话框将查找要写入文件的位置。

参数：

parent - 对话框的所有者

title - 对话框的标题

mode - 对话框的模式，可以是 FileDialog.LOAD 或 FileDialog.SAVE

抛出：

IllegalArgumentException - 如果提供了非法的文件对话框模式

另请参见：

LOAD,SAVE

FileDialog

```
public FileDialog(Dialog parent)
```

创建一个文件对话框，用于加载文件。文件对话框的标题最初是空的。这是 FileDialog(parent, "", LOAD) 的一个便捷方法。

参数：

parent - 对话框的所有者

抛出：

IllegalArgumentException - 如果 parent 的 GraphicsConfiguration 不是来自屏幕设备；

IllegalArgumentException - 如果 parent 为 null；当 GraphicsEnvironment.isHeadless 返回 true 时始终抛出此异常

从以下版本开始：

1.5

另请参见：

GraphicsEnvironment.isHeadless()

FileDialog

```
public FileDialog(Dialog parent,
                  String title)
```

创建一个具有指定标题的文件对话框，用于加载文件。显示的文件是当前目录中的文件。这是 FileDialog(parent, title, LOAD) 的一个便捷方法。

参数：

parent - 对话框的所有者

title - 对话框的标题；接受 null 值时不会抛出 NullPointerException

抛出：

IllegalArgumentException - 如果 parent 的 GraphicsConfiguration 不是来自屏幕设备；

IllegalArgumentException - 如果 parent 为 null；当 GraphicsEnvironment.isHeadless 返回 true 时始终抛出此异常

从以下版本开始：

1.5

另请参见：

GraphicsEnvironment.isHeadless()

FileDialog

```
public FileDialog(Dialog parent,
                  String title,
                  int mode)
```

创建一个具有指定标题的文件对话框窗口，用于加载或保存文件。

如果 mode 的值为 LOAD，那么文件对话框将查找要读取的文件，所显示的文件是当前目录中的文件。如果 mode 的值为 SAVE，则文件对话框将查找要写入文件的位置。

参数：

parent - 对话框的所有者

title - 对话框的标题；接受 null 值时不会抛出 NullPointerException

mode - 对话框的模式，FileDialog.LOAD 或 FileDialog.SAVE

抛出：

IllegalArgumentException - 如果提供了非法的文件对话框模式；

IllegalArgumentException - 如果 parent 的 GraphicsConfiguration 不是来自屏幕设备；

IllegalArgumentException - 如果 parent 为 null；当 GraphicsEnvironment.isHeadless 返回 true 时始终抛出此异常

方法详细信息

addNotify

public void addNotify()

创建文件对话框的同位体。同位体允许我们更改文件对话框的外观而不更改其功能。

覆盖：

类 Dialog 中的 addNotify

另请参见：

Component.isDisplayable(),Window.removeNotify()

---

getMode

public int getMode()

指示此文件对话框是用于从文件加载内容还是将内容保存到文件。

返回：

文件对话框窗口的模式，可以是 FileDialog.LOAD 或 FileDialog.SAVE

另请参见：

LOAD,SAVE,setMode(int)

---

setMode

public void setMode(int mode)

设置文件对话框的模式。如果 mode 不是一个合法值，则抛出一个异常，并且不设置 mode。

参数：

mode - 文件对话框的模式，可以是 FileDialog.LOAD 或 FileDialog.SAVE

抛出：

IllegalArgumentException - 如果提供了非法的文件对话框模式

从以下版本开始：

JDK1.1

另请参见：

LOAD, SAVE, getMode()

getDirectory

public String getDirectory()

获取此文件对话框的目录。

返回：

此 FileDialog 的目录（可能为 null 或无效）

另请参见：

setDirectory(java.lang.String)

setDirectory

public void setDirectory(String dir)

将此文件对话框窗口的目录设置为指定目录。指定 null 或无效目录意味着指定由实现所定义的默认值。但是，在用户选择一个文件之前，此默认值不会被实现。在此之前，getDirectory() 将返回传递到此方法的值。

指定 "" 作为目录，完全等同于指定 null 作为目录。

参数：

dir - 指定的目录

另请参见：

getDirectory()

getFile

public String getFile()

获取此文件对话框的选定文件。如果用户选择 CANCEL，则返回文件为 null。

返回：

此文件对话框窗口当前所选择的文件；如果没有文件被选择，则返回 null

另请参见：

setFile(java.lang.String)

setFile

public void setFile(String file)

将此文件对话框窗口的选定文件设置为指定文件。如果这么设置，那么在文件对话框窗口第一次显示之前，此文件就成为默认文件。

指定 "" 作为文件，完全等同于指定 null 作为文件。

参数：

file - 正被设置的文件 set

另请参见：

getFile()

getFilenameFilter

public FilenameFilter getFilenameFilter()

确定此文件对话框的文件名过滤器。文件名过滤器允许用户指定哪个文件出现在文件对话框窗口。Microsoft Windows 的文件名过滤器在 Sun 的参考实现中不起作用。

返回：

此文件对话框的文件名过滤器

另请参见：

FilenameFilter,setFilenameFilter(java.io.FilenameFilter)

setFilenameFilter

public void setFilenameFilter(FilenameFilter filter)

将此文件对话框窗口的文件名过滤器设置为指定的过滤器。Microsoft Windows 的文件名过滤器在 Sun 的参考实现中不起作用。

参数：

filter - 指定的过滤器

另请参见：
FilenameFilter,getFilenameFilter()

paramString

protected String paramString()

返回表示此 FileDialog 窗口状态的字符串。此方法仅在进行调试的时候使用，对于不同的实现，返回的字符串的内容和格式可能有所不同。返回的字符串可能为空，但不可能为 null。

覆盖：
类 Dialog 中的 paramString

返回：
此文件对话框窗口的参数字符串

FlowLayout 类

流布局用于安排有向流中的组件，这非常类似于段落中的文本行。流的方向取决于容器的 componentOrientation 属性，它可能是以下两个值中的一个：

ComponentOrientation.LEFT_TO_RIGHT

ComponentOrientation.RIGHT_TO_LEFT

流布局一般用来安排面板中的按钮。它使得按钮呈水平放置，直到同一条线上再也没有适合的按钮。线的对齐方式由 align 属性确定。可能的值为：

LEFT

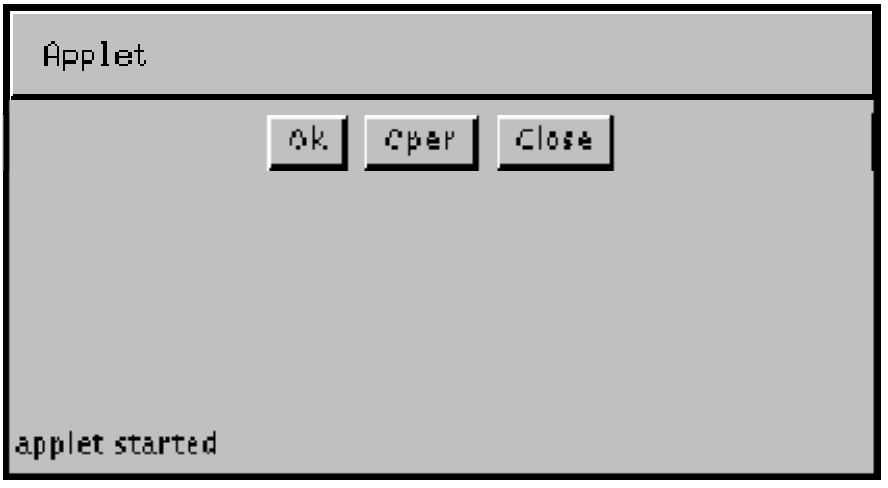
RIGHT

CENTER

LEADING

TRAILING

例如，以下图片显示了使用流布局管理器（它的默认布局管理器）来定位三个按钮的 applet：



以下是此 applet 的代码：

```
import java.awt.*;
import java.applet.Applet;

public class myButtons extends Applet {
    Button button1, button2, button3;
    public void init() {
        button1 = new Button("Ok");
        button2 = new Button("Open");
        button3 = new Button("Close");
        add(button1);
        add(button2);
        add(button3);
    }
}
```

流布局把每个组件都假定为它的自然（首选）大小。

字段详细信息

LEFT

public static final int LEFT

此值指示每一行组件都应该是左对齐的。

另请参见：

常量字段值

CENTER

public static final int CENTER

此值指示每一行组件都应该是居中的。

另请参见：

常量字段值

RIGHT

public static final int RIGHT

此值指示每一行组件都应该是右对齐的。

另请参见：

常量字段值

LEADING

public static final int LEADING

此值指示每一行组件都应该与容器方向的开始边对齐，例如，对于从左到右的方向，则与左边对齐。

从以下版本开始：

1.2

另请参见：

Component.getComponentOrientation(),ComponentOrientation,常量字段值

TRAILING

```
public static final int TRAILING
```

此值指示每行组件都应该与容器方向的结束边对齐，例如，对于从左到右的方向，则与右边对齐。

构造方法详细信息

FlowLayout

```
public FlowLayout()
```

构造一个新的 FlowLayout，它是居中对齐的，默认的水平和垂直间隙是 5 个单位。

FlowLayout

```
public FlowLayout(int align)
```

构造一个新的 FlowLayout，它具有指定的对齐方式，默认的水平和垂直间隙是 5 个单位。align 参数的值必须是以下值之一：FlowLayout.LEFT、FlowLayout.RIGHT、FlowLayout.CENTER、FlowLayout.LEADING 或 FlowLayout.TRAILING。

参数：

align - 对齐值

FlowLayout

```
public FlowLayout(int align,
                  int hgap,
                  int vgap)
```

创建一个新的流布局管理器，它具有指定的对齐方式以及指定的水平和垂直间隙。

align 参数的值必须是以下值之一：FlowLayout.LEFT、FlowLayout.RIGHT、FlowLayout.CENTER、FlowLayout.LEADING 或 FlowLayout.TRAILING。

参数：

align - 对齐值
hgap - 组件之间以及组件与 Container 的边之间的水平间隙
vgap - 组件之间以及组件与 Container 的边之间的垂直间隙

方法详细信息

getAlignment

```
public int getAlignment()
```

获取此布局的对齐方式。可能的值是 FlowLayout.LEFT、FlowLayout.RIGHT、FlowLayout.CENTER、FlowLayout.LEADING 或 FlowLayout.TRAILING。

返回：

此布局的对齐值

从以下版本开始：

JDK1.1

另请参见：

setAlignment(int)

setAlignment

```
public void setAlignment(int align)
```

设置此布局的对齐方式。可能的值是

- FlowLayout.LEFT
- FlowLayout.RIGHT
- FlowLayout.CENTER
- FlowLayout.LEADING
- FlowLayout.TRAILING

参数：
align - 上面显示的对齐值之一
从以下版本开始：
JDK1.1
另请参见：
getAlignment()

getHgap

```
public int getHgap()
```

获取组件之间以及组件与 Container 的边之间的水平间隙。
返回：
组件之间以及组件与 Container 的边之间的水平间隙
从以下版本开始：
JDK1.1
另请参见：
setHgap(int)

setHgap

```
public void setHgap(int hgap)
```

设置组件之间以及组件与 Container 的边之间的水平间隙。
参数：
hgap - 组件之间以及组件与 Container 的边之间的水平间隙
从以下版本开始：
JDK1.1
另请参见：
getHgap()

getVgap

```
public int getVgap()
```

获取组件之间以及组件与 Container 的边之间的垂直间隙。
返回：
组件之间以及组件与 Container 的边之间的垂直间隙
从以下版本开始：
JDK1.1
另请参见：
setVgap(int)

setVgap

```
public void setVgap(int vgap)
```

设置组件之间以及组件与 Container 的边之间的垂直间隙。
参数：
vgap - 组件之间以及组件与 Container 的边之间的垂直间隙
从以下版本开始：
JDK1.1
另请参见：
getVgap()

setAlignOnBaseline

```
public void setAlignOnBaseline(boolean alignOnBaseline)
```

设置组件是否应该沿着其基线垂直对齐。没有基线的组件将中心对齐。默认值为 false。
参数：
alignOnBaseline - 组件是否应该沿着其基线垂直对齐
从以下版本开始：
1.6

getAlignOnBaseline

```
public boolean getAlignOnBaseline()
```

如果组件将沿其基线垂直对齐，则返回 true。默认值为 false。

返回：

true 如果组件将沿其基线垂直对齐

从以下版本开始：

1.6

---

addLayoutComponent

```
public void addLayoutComponent(String name,
                               Component comp)
```

将指定的组件添加到布局中。不能被此类使用。

指定者：

接口 LayoutManager 中的 addLayoutComponent

参数：

name - 组件的名称

comp - 要添加的组件

---

removeLayoutComponent

```
public void removeLayoutComponent(Component comp)
```

从布局中移除指定的组件。不能被此类使用。

指定者：

接口 LayoutManager 中的 removeLayoutComponent

参数：

comp - 要移除的组件

另请参见：

Container.removeAll()

---

preferredLayoutSize

```
public Dimension preferredLayoutSize(Container target)
```

给出指定目标容器中的 *visible* 组件，返回此布局的首选维数。

指定者：
接口 LayoutManager 中的 preferredLayoutSize
参数：
target - 需要布置的容器
返回：
布置指定容器的子组件的首选维数
另请参见：
Container, minimumLayoutSize(java.awt.Container), Container.getPreferredSize()

minimumLayoutSize

public Dimension minimumLayoutSize(Container target)

返回需要布置 *visible* 组件的最小维数，该组件包含在指定的目标容器中。
指定者：
接口 LayoutManager 中的 minimumLayoutSize
参数：
target - 需要布置的容器
返回：
布置指定容器的子组件的最小维数
另请参见：
preferredLayoutSize(java.awt.Container), Container, Container.doLayout()

layoutContainer

public void layoutContainer(Container target)

布置该容器。此方法让每个 *visible* 组件都采用它的首选大小，这通过对目标容器中的组件进行整形得以实现，以满足此 FlowLayout 对象的对齐方式。
指定者：
接口 LayoutManager 中的 layoutContainer
参数：
target - 正被布置的指定组件
另请参见：

Container,Container.doLayout()

toString

```
public String toString()
```

返回此 FlowLayout 对象及其值的字符串表示形式。

覆盖：

类 Object 中的 toString

返回：

此布局的字符串表示形式

Font 类

Font 类表示字体，可以使用它以可见方式呈现文本。字体提供将*字符* 序列映射到*字形* 序列所需要的信息，以便在 Graphics 对象和 Component 对象上呈现字形序列。

字符和字形

字符 是表示项的符号（如字母、数字或具有抽象意义的标点）。例如，'g'（G 的拉丁小写字母）是一个字符。

字形 是用来呈现字符或字符序列的一种形状。在简单的书写系统（如拉丁文）中，通常一个字形就表示一个字符。但在一般情况下，字符和字形并没有一对一的对应关系。例如，字符 'á' A 的带重音符号的拉丁小写字母 可以由两个字形表示：一个是 'a'，一个是 '´'。另一方面，两字符的字符串 "fi" 可以由单个字形 "fi" 连字表示。在复杂的书写系统（如阿拉伯语或南亚和东南亚语言）中，字符与字形之间的关系可能更复杂，涉及到依赖于上下文的字形选择以及字形重排序。字体封装了呈现所选择的字符集所需的字形集，还封装了将字符序列映射到相应的字形序列所需的表。

物理字体和逻辑字体

Java Platform 可以区分两种字体：*物理* 字体和*逻辑* 字体。

物理 字体是实际的字体库，包含字形数据和表，这些数据和表使用字体技术（如 TrueType 或 PostScript Type 1）将字符序列映射到字形序列。Java Platform 的所有实现都支持 TrueType 字体；对其他字体技术的支持是与实现相关的。物理字体可以使用字体名称，如 Helvetica、Palatino、HonMincho 或任意数量的其他字体名称。通常，每种物理字体只支持有限的书写系统集合，例如，只支持拉丁文字符，或者只支持日文和基本拉丁文。可用的物理字体集合随配置的不同而有所不同。要求特定字体的应用程序可以使用 createFont 方法来捆绑这些字体，并对其进行实例化。

逻辑 字体是由必须受所有 Java 运行时环境支持的 Java 平台所定义的五种字体系列：Serif、SansSerif、Monospaced、Dialog 和 DialogInput。这些逻辑字体不是实际的字体库。此外，由 Java 运行时环境将逻辑字体名称映射到物理字体。映射关系与实现和通常语言环境相关，因此它们提供的外观和规格各不相同。通常，为了覆盖庞大的字符范围，每种逻辑字体名称都映射到几种物理字体。

同级的 AWT 组件（如 Label 和 TextField）只可以使用逻辑字体。

有关使用物理字体或逻辑字体的相对优势和劣势的讨论，请参阅 Internationalization FAQ 文档。

字体外观和名称

Font 可以有多个外观，如 heavy、medium、oblique、gothic 和 regular。所有这些外观都有相似的排字设计。

可以从 Font 对象获得三种不同的名称。*逻辑字体名称* 只是用来构造字体的名称。*字体外观名称*，或*外观名称* 是特定字体外观的名称，如 Helvetica Bold。*系列名称* 是字体系列的名称，用来确定多种外观（如 Helvetica）的排字设计。

Font 类表示字体外观集合中字体外观的实例，字体外观集合位于主机系统的系统资源中。例如，Arial Bold 和 Courier Bold Italic 是字体外观。有几个 Font 对象与字体外观相关，每个对象在大小、样式、变换和字体特征上都有所不同。

GraphicsEnvironment 类的 getAllFonts 方法返回系统中所有可用字体外观组成的数组。这些字体外观被作为 Font 对象返回，对象的大小为 1，具有恒等变换和默认字体特征。这些基本字体可以用来派生新的 Font 对象，使之具有变化的大小、样式、变换和字体特征，这些可以通过此类中的 deriveFont 方法实现。

Font 和 TextAttribute

Font 支持大多数 TextAttribute。这使得一些操作（如呈现带下划线的文本）变得更方便，因为没必要再显式构造一个 TextLayout 对象。通过使用 TextAttribute 值的 Map 来构造或派生属性，可以在 Font 上设置属性。

一些 TextAttributes 值是不可序列化的，因此尝试序列化具有这样的值的 Font 实例不会使它们获得序列化。这意味着从这样一个流实现序列化的 Font 不会等同于包含不可序列化属性的原始 Font。这应该是很少出现的一个问题，因为这些属性通常只使用在特殊情况下并且不可能被序列化。

FOREGROUND 和 BACKGROUND 使用 Paint 值。子类 Color 是可序列化的，而 GradientPaint 和 TexturePaint 是不可序列化的。

CHAR_REPLACEMENT 使用 GraphicAttribute 值。子类 ShapeGraphicAttribute 和 ImageGraphicAttribute 是不可序列化的。

INPUT_METHOD_HIGHLIGHT 使用 InputMethodHighlight 值，它是不可序列化的。请参阅 InputMethodHighlight。

创建 Paint 和 GraphicAttribute 的自定义子类的客户机可以使它们可序列化并避免此问题。使用输入方法高亮显示的客户机可以将这些转换为特定于平台的属性，以便在当前平台上高亮显示它们并在 Font 上设置它们，以此作为解决方法。

基于 Map 的构造方法和 deriveFont API 忽略了 FONT 属性，并且没有通过 Font 保留它；如果 FONT 属性可能存在，则应该使用 static getFont(java.util.Map) 方法。有关更多信息，请参阅 TextAttribute.FONT。

一些属性会带来额外的呈现开销，并且可能调用布局。如果某一 Font 具有这样的属性，那么 hasLayoutAttributes 方法将返回 true。

注：字体旋转可能导致文本基线也跟着旋转。为了应对这种（很少见的）可能性，可以指定字体 API 返回字体规格，并使参数“位于相对于基线的坐标中”。这将 'x' 坐标映射到沿基线的某段距离处（正 x 表示沿基线的正方向），将 'y' 坐标映射到位于 'x' 坐标的基线垂直方向的某段距离处（正 y 表示从基线向量顺时针旋转 90 度）。用于实现这些的 API 特别重要，它们被调出，就像具有“相对于基线的坐标”一样。

字段详细信息

DIALOG

public static final String DIALOG

逻辑字体 "Dialog" 的规范系列名称的 String 常量。它在 Font 构造中很有用，可提供该名称的编译时验证。

从以下版本开始：

1.6

另请参见：

常量字段值

DIALOG_INPUT

public static final String DIALOG_INPUT

逻辑字体 "DialogInput" 的规范系列名称的 String 常量。它在 Font 构造中很有用，可提供该名称的编译时验证。

从以下版本开始：

1.6

另请参见：

常量字段值

SANS_SERIF

public static final String SANS_SERIF

逻辑字体 "SansSerif" 的规范系列名称的 String 常量。它在 Font 构造中很有用，可提供该名称的编译时验证。

从以下版本开始：

1.6

另请参见：

常量字段值

SERIF

public static final String SERIF

逻辑字体 "Serif" 的规范系列名称的 String 常量。它在 Font 构造中很有用，可提供该名称的编译时验证。

从以下版本开始：

1.6

另请参见：

常量字段值

MONOSPACED

public static final String MONOSPACED

逻辑字体 "Monospaced" 的规范系列名称的 String 常量。它在 Font 构造中很有用，可提供该名称的编译时验证。

从以下版本开始：

1.6

另请参见：

常量字段值

PLAIN

public static final int PLAIN

普通样式常量。

另请参见：

常量字段值

BOLD

public static final int BOLD

粗体样式常量。可与其他样式常量（PLAIN 除外）进行组合，从而得到混合样式。

另请参见：

常量字段值

ITALIC

public static final int ITALIC

斜体样式常量。可与其他样式常量（PLAIN 除外）进行组合，从而得到混合样式。

另请参见：

常量字段值

ROMAN_BASELINE

public static final int ROMAN_BASELINE

布置文本时，在大多数 Roman 脚本中使用的基线。

另请参见：

常量字段值

CENTER_BASELINE

public static final int CENTER_BASELINE

布置文本时，在表意文字的脚本（如中文、日文和韩文）中使用的基线。

另请参见：
常量字段值

HANGING_BASELINE

public static final int HANGING_BASELINE

布置文本时，在 Devanigiri 和类似脚本中使用的基线。
另请参见：
常量字段值

TRUETYPE_FONT

public static final int TRUETYPE_FONT

标识 TRUETYPE 类型的字体资源。可用它为 createFont(int, java.io.InputStream) 方法指定一个 TrueType 字体资源。
从以下版本开始：
1.3
另请参见：
常量字段值

TYPE1_FONT

public static final int TYPE1_FONT

标识 TYPE1 类型的字体资源。可用它为 createFont(int, java.io.InputStream) 方法指定一个 Type1 字体资源。
从以下版本开始：
1.5
另请参见：
常量字段值

name

protected String name

Font 的逻辑名称，它被传递到构造方法中。

从以下版本开始：

JDK1.0

另请参见：

getName()

style

protected int style

Font 的样式，它被传递到构造方法中。此样式可以为 PLAIN、BOLD、ITALIC 或 BOLD+ITALIC。

从以下版本开始：

JDK1.0

另请参见：

getStyle()

size

protected int size

Font 的磅值大小，舍入为整数。

从以下版本开始：

JDK1.0

另请参见：

getSize()

pointSize

protected float pointSize

以 float 形式表示的 Font 的磅值大小。

另请参见：

getSize(),getSize2D()

LAYOUT_LEFT_TO_RIGHT

public static final int LAYOUT_LEFT_TO_RIGHT

layoutGlyphVector 的标志，指示文本是从左到右的，这由 Bidi 分析确定。

另请参见：

常量字段值

LAYOUT_RIGHT_TO_LEFT

public static final int LAYOUT_RIGHT_TO_LEFT

layoutGlyphVector 的标志，指示文本是从右到左的，这由 Bidi 分析确定。

另请参见：

常量字段值

LAYOUT_NO_START_CONTEXT

public static final int LAYOUT_NO_START_CONTEXT

layoutGlyphVector 的标志，指示不应该检查指定 start 之前的 char 数组中的文本。

另请参见：

常量字段值

LAYOUT_NO_LIMIT_CONTEXT

public static final int LAYOUT_NO_LIMIT_CONTEXT

layoutGlyphVector 的标志，指示不应该检查指定 limit 之后的 char 数组中的文本。

构造方法详细信息

Font

```
public Font(String name,
            int style,
            int size)
```

根据指定名称、样式和磅值大小，创建一个新 Font。

字体名称可以是字体外观名称或字体系列名称。它与样式一起使用，以查找合适的字体外观。如果指定了字体系列名称，则使用样式参数从系列中选择最合适的外观。如果指定了字体外观名称，则合并外观的样式和样式参数，以便从同一个系列查找最匹配的字体。例如，如果指定外观名称 "Arial Bold" 及样式 Font.ITALIC，则字体系统在 "Arial" 系列中寻找既是粗体又是斜体的外观，可以将字体实例与物理字体外观 "Arial Bold Italic" 相关联。将样式参数与指定外观的样式合并，而不是执行添加或减去操作。这意味着，指定粗体外观和粗体样式并不会双倍加粗字体，而指定粗体外观和普通样式也不会变细字体。

如果无法找到所要求样式的外观，则字体系统可以应用样式设计算法来获得所需的样式。例如，如果要求 ITALIC，但是没有可用的斜体外观，则可以通过算法使普通外观倾斜。

字体名称查找是区分大小写的，可以使用 US 语言环境的大小写转换规则。

如果 name 参数表示逻辑字体以外的事物，例如表示为物理字体外观或系列名称，并且无法通过实现物理字体或可兼容的替代物映射它，则字体系统会将 Font 实例映射到 "Dialog"，因此，通过 getFamily 报告的字体系列名称将是 "Dialog"。

参数：

name - 字体名称。可以是字体外观名称或字体系列名称，并且可表示此 GraphicsEnvironment 中找到的逻辑字体或物理字体。逻辑字体的系列名称有：Dialog、DialogInput、Monospaced、Serif 或 SansSerif。预定义 String 常量是为所有这些名称（如 DIALOG）而存在。如果 name 为 null，则将新 Font 的*逻辑字体名称*（由 getName() 返回）设置为 "Default"。

style - Font 的样式常量。样式参数是整数位掩码，可以为 PLAIN，或 BOLD 和 ITALIC 的按位或（例如，ITALIC 或 BOLD|ITALIC）。如果样式参数不符合任何一个期望的整数位掩码，则将样式设置为 PLAIN。
size - Font 的磅值大小
从以下版本开始：
JDK1.0
另请参见：
GraphicsEnvironment.getAllFonts(),
GraphicsEnvironment.getAvailableFontFamilyNames()

Font

public Font(Map<? extends AttributedCharacterIterator.Attribute,?> attributes)

创建一个具有指定属性的新 Font。只有 TextAttribute 中定义的键被识别。此外，FONT 属性也没有被此构造方法识别（请参阅 getAvailableAttributes()）。只有具有有效类型值的属性会影响新的 Font。

如果 attributes 为 null，则使用默认值初始化新 Font。

参数：
attributes - 分配给新 Font 的属性，或 null
另请参见：
TextAttribute

Font

protected Font(Font font)

根据指定 font 创建一个新 Font。此构造方法由子类使用。
参数：
font - 用来创建此 Font 的字体。
抛出：
NullPointerException - 如果 font 为 null
从以下版本开始：
1.6

方法详细信息

getPeer

```
@Deprecated
    public java.awt.peer.FontPeer getPeer()
```

已过时。 *现在，字体呈现与平台无关。*

获取此 Font 的同位体。

返回：

Font 的同位体。

从以下版本开始：

JDK1.1

---

getFont

```
public static Font getFont(Map<? extends AttributedCharacterIterator.Attribute,?> attributes)
```

返回适合于这些属性的 Font。如果 attributes 包含一个使用有效 Font 作为其值的 FONT 属性，那么它将与其余所有属性合并。有关更多信息，请参阅 TextAttribute.FONT。

参数：

attributes - 分配给新 Font 的属性

返回：

使用指定属性创建的新 Font

抛出：

NullPointerException - 如果 attributes 为 null。

从以下版本开始：

1.2

另请参见：

TextAttribute

---

createFont

```
public static Font createFont(int fontFormat,
                              InputStream fontStream)
                       throws FontFormatException,
                              IOException
```

返回一个使用指定字体类型和输入数据的新 Font。创建的新 Font 的磅值大小为 1，样式为 PLAIN。然后，基本字体可以与此类中的 deriveFont 方法一起使用，从而派生出新的 Font 对象，这些对象具有变化的大小、样式、变换和字体特征。此方法不会关闭 InputStream。

为了使 Font 可用于 Font 构造方法，必须通过调用 registerFont(Font) 在 GraphicsEnviroment 中注册返回的 Font。

参数：

fontFormat - Font 的类型，如果指定了 TrueType 资源，则类型为 TRUETYPE_FONT，如果指定了 Type 1 资源，则类型为 TYPE1_FONT。

fontStream - InputStream 对象，表示字体的输入数据。

返回：

使用指定字体类型创建的新 Font。

抛出：

IllegalArgumentException - 如果 fontFormat 不为 TRUETYPE_FONT 或 TYPE1_FONT。

FontFormatException - 如果 fontStream 数据不包含指定格式所需的字体表。

IOException - 如果无法完全读取 fontStream。

从以下版本开始：

1.3

另请参见：

GraphicsEnvironment.registerFont(Font)

createFont

```
public static Font createFont(int fontFormat,
                              File fontFile)
                       throws FontFormatException,
                              IOException
```

返回一个使用指定字体类型和指定字体文件的新 Font。创建的新 Font 的磅值大小为 1，样式为 PLAIN。基本字体可以与此类中的 deriveFont 方法一起使用，

从而派生出新的 Font 对象，这些对象具有变化的大小、样式、变换和字体特征。

参数：

fontFormat - Font 的类型，如果指定了 TrueType 资源，则类型为 TRUETYPE_FONT，如果指定了 Type 1 资源，则类型为 TYPE1_FONT。只要引用了返回的字体，或它派生的字体，实现就可以继续访问 fontFile 以检索字体数据。因此，如果文件被更改，或变成不可访问的，其结果是不确定的。

为了使 Font 可用于 Font 构造方法，必须通过调用 registerFont(Font) 在 GraphicsEnviroment 中注册返回的 Font。

fontFile - File 对象，表示字体的输入数据。

返回：

使用指定字体类型创建的新 Font。

抛出：

IllegalArgumentException - 如果 fontFormat 不为 TRUETYPE_FONTorTYPE1_FONT。

NullPointerException - 如果 fontFile 为 null。

IOException - 如果无法读取 fontFile。

FontFormatException - 如果 fontFile 不包含指定格式所需的字体表。

SecurityException - 如果正在执行的代码没有从文件中读取的权限。

从以下版本开始：

1.5

另请参见：

GraphicsEnvironment.registerFont(Font)

getTransform

```
public AffineTransform getTransform()
```

返回与此 Font 相关的变换的副本。此转换对于用于构造字体的方法而言是没必要的。如果字体具有算法上标（algorithmic superscripting）或宽度调整，那么这将合并到返回的 AffineTransform 中。

通常不会转换字体。客户机通常应该先调用 isTransformed()，如果 isTransformed 返回 true，则只调用此方法。

返回：

AffineTransform 对象，表示此 Font 对象的变换属性。

getFamily

public String getFamily()

返回此 Font 的系列名称。

字体的系列名称是特定于字体的。两个字体 Helvetica Italic 和 Helvetica Bold 具有相同的系列名称 *Helvetica*，但它们的字体外观名称是 *Helvetica Bold* 和 *Helvetica Italic*。可用的系列名称列表可以通过使用 GraphicsEnvironment.getAvailableFontFamilyNames() 方法得到。

使用 getName 可以获取字体的逻辑名称。使用 getFontName 可以获取字体的字体外观名称。

返回：
一个 String，此 Font 的系列名称。
从以下版本开始：
JDK1.1
另请参见：
getName(),getFontName()

getFamily

public String getFamily(Locale l)

返回此 Font 的系列名称（已经针对指定语言环境进行了本地化）。

字体的系列名称是特定于字体的。两个字体 Helvetica Italic 和 Helvetica Bold 具有相同的系列名称 *Helvetica*，但它们的字体外观名称是 *Helvetica Bold* 和 *Helvetica Italic*。可用的系列名称列表可以通过使用 GraphicsEnvironment.getAvailableFontFamilyNames() 方法得到。

使用 getFontName 可以获取字体的字体外观名称。

参数：
l - 语言环境，要获得该语言环境的系列名称
返回：
一个 String，表示字体的系列名称（已经针对指定语言环境进行了本地化）。
从以下版本开始：
1.2
另请参见：
getFontName(),Locale

---

getPSName

```
public String getPSName()
```

返回此 Font 的 postscript 名称。使用 getFamily 可以获取字体的系列名称。使用 getFontName 可以获取字体的字体外观名称。
返回：
一个 String，表示此 Font 的 postscript 名称。
从以下版本开始：
1.2

---

getName

```
public String getName()
```

返回此 Font 的逻辑名称。使用 getFamily 可以获取字体的系列名称。使用 getFontName 可以获取字体的字体外观名称。
返回：
一个 String，表示此 Font 的逻辑名称。
从以下版本开始：
JDK1.0
另请参见：
getFamily(),getFontName()

---

getFontName

public String getFontName()

返回此 Font 的字体外观名称。例如，Helvetica Bold 可以作为字体外观名称返回。使用 getFamily 可以获取字体的系列名称。使用 getName 可以获取字体的逻辑名称。

返回：

一个 String，表示此 Font 的字体外观名称。

从以下版本开始：

1.2

另请参见：

getFamily(),getName()

---

getFontName

public String getFontName(Locale l)

返回此 Font 的字体外观名称（已经针对指定语言环境进行了本地化）。例如，Helvetica Fett 可以作为字体外观名称返回。使用 getFamily 可以获取字体的系列名称。

参数：

l - 语言环境，要获得该语言环境的字体外观名称

返回：

一个 String，表示字体外观名称（已经针对指定语言环境进行了本地化）。

另请参见：

getFamily(),Locale

---

getStyle

public int getStyle()

返回此 Font 的样式。样式可以为 PLAIN、BOLD、ITALIC 或 BOLD+ITALIC。

返回：

此 Font 的样式

从以下版本开始：

JDK1.0

另请参见：

isPlain(),isBold(),isItalic()

getSize

public int getSize()

返回此 Font 的磅值大小，舍入为整数。大多数用户都熟悉使用*磅值大小* 的概念，它用于指定字体中字形的大小。磅值大小定义了单间距文本文档中某行的基线到下一行的基线之间的测量。磅值大小是基于*排字磅值* 的，大约为 1/72 英寸。

Java(tm)2D API 规定：1 磅等于用户坐标中的 1 个单位。当使用规范化变换将用户空间坐标转换为设备空间坐标时，72 个用户空间单位等于设备空间中的 1 英寸。在这种情况下，1 磅就是 1/72 英寸。

返回：

Font 的磅值大小（以 1/72 英寸为单位）。

从以下版本开始：

JDK1.0

另请参见：

getSize2D(), GraphicsConfiguration.getDefaultTransform(), GraphicsConfiguration.getNormalizingTransform()

getSize2D

public float getSize2D()

返回此 Font 的磅值大小（以 float 值表示）。

返回：

Font 的磅值大小（以 float 值表示）。

从以下版本开始：

1.2

另请参见：

getSize()

isPlain

public boolean isPlain()

指示此 Font 对象的样式是否为 PLAIN。

返回：

如果此 Font 样式为 PLAIN，则返回 true；否则返回 false。

从以下版本开始：

JDK1.0

另请参见：

getStyle()

isBold

public boolean isBold()

指示此 Font 对象的样式是否为 BOLD。

返回：

如果此 Font 对象的样式为 BOLD，则返回 true；否则返回 false。

从以下版本开始：

JDK1.0

另请参见：

getStyle()

isItalic

public boolean isItalic()

指示此 Font 对象的样式是否为 ITALIC。

返回：

如果此 Font 对象的样式为 ITALIC，则返回 true；否则返回 false。

从以下版本开始：

JDK1.0

另请参见：

getStyle()

isTransformed

public boolean isTransformed()

指示此 Font 对象是否具有影响其大小以及 Size 属性的变换。

返回：

如果此 Font 对象具有无标识的 AffineTransform 属性，则返回 true；否则返回 false。

从以下版本开始：

1.4

另请参见：

getTransform()

hasLayoutAttributes

public boolean hasLayoutAttributes()

如果此 Font 包含需要额外布局处理的属性，则返回 true。

返回：

如果该字体已经具有布局属性，则返回 true

从以下版本开始：

1.6

getFont

public static Font getFont(String nm)

从系统属性列表返回一个 Font 对象。nm 被视为要获得的系统属性的名称。然后，根据 Font.decode(String) 规范，将此属性的 String 值解释为一个 Font 对象。如果无法找到指定属性，或者执行代码没有读取该属性的权限，则返回 null。

参数：

nm - 属性名称

返回：

属性名称描述的 Font 对象，如果没有这样的属性存在，则返回 null。

抛出：

NullPointerException - 如果 nm 为 null。

从以下版本开始：

1.2

另请参见：

decode(String)

decode

public static Font decode(String str)

返回 str 参数所描述的 Font。为了确保此方法返回所需的 Font，可以使用以下方式之一格式化 str 参数。

- *fontname-style-pointsize*
- *fontname-pointsize*
- *fontname-style*
- *fontname*
- *fontname style pointsize*
- *fontname pointsize*
- *fontname style*
- *fontname*

其中，*style* 是以下四个区分大小写的字符串之一：″PLAIN″、″BOLD″、″BOLDITALIC″ 和 ″ITALIC″，pointsize 是磅值大小的正十进制整数表示形式。例如，如果想要的字体是 Arial、粗体、磅值大小为 18，则按以下方式调用此方法："Arial-BOLD-18"。这等同于调用以下 Font 构造方法：new Font(″Arial″, Font.BOLD, 18);，值被解释为构造方法所指定的形式。

有效的尾部十进制字段总是被解释为 pointsize。因此，后跟十进制值的 fontname 不应该在只在 fontname 的格式中使用。

如果样式名称字段不是有效的样式字符串，则它被解释为样式名称的一部分，并且使用默认样式。

只有 ’ ’ 或 ’-’ 可以用来分隔输入中的字段。所标识的分隔符是最接近字符串末尾的字符，它将有效的 pointsize 或有效的样式名称与字符串的其余部分分隔开来。Null（空）pointsize 和样式字段被视为有效字段，具有该字段的默认值。

有些字体名称可以包括分隔符 ’ ’ 或 ’-’。如果 str 不是由三部分构成，例如，style 或 pointsize 字段不出现在 str 中，并且 fontname 还包

含确定为分隔符的字符，则这些打算作为 fontname 一部分出现的字符也可以解释为分隔符，因此字体名称无法正确识别。

默认大小为 12，默认样式为 PLAIN。如果 str 没有指定有效大小，则返回的 Font 大小为 12。如果 str 没有指定有效样式，则返回的 Font 样式为 PLAIN。如果没有在 str 参数中指定有效的字体名称，则此方法将返回系列名称为 "Dialog" 的字体。要确定系统上可以使用哪些字体系列名称，可以使用 GraphicsEnvironment.getAvailableFontFamilyNames() 方法。如果 str 为 null，则返回的新 Font 的系列名称为 "Dialog"，大小为 12，样式为 PLAIN。

参数：

str - 字体名称，或 null

返回：

str 描述的 Font 对象，如果 str 为 null，则返回新的默认 Font。

从以下版本开始：

JDK1.1

另请参见：

getFamily()

getFont

```
public static Font getFont(String nm,
                           Font font)
```

从系统属性列表获取指定的 Font。在 System 的 getProperty 方法中，第一个参数被视为要获取的系统属性的名称。接着将此属性的 String 值解释为 Font 对象。

属性值应该是 Font.decode(String) 所接受的形式之一。如果无法找到指定属性，或者执行代码没有读取该属性的权限，则返回 font 参数。

参数：

nm - 不区分大小写的属性名称

font - 如果没有定义属性 nm，则为要返回的默认 Font

返回：

属性的 Font 值。

抛出：

NullPointerException - 如果 nm 为 null。

另请参见：

decode(String)

hashCode

```
public int hashCode()
```

返回此 Font 的哈希码。

覆盖：

类 Object 中的 hashCode

返回：

此 Font 的哈希码值。

从以下版本开始：

JDK1.0

另请参见：

Object.equals(java.lang.Object),Hashtable

equals

```
public boolean equals(Object obj)
```

将此 Font 对象与指定 Object 进行比较。

覆盖：

类 Object 中的 equals

参数：

obj - 要比较的 Object

返回：

如果对象相同，或参数是 Font 对象，且描述的字体与此对象相同，则返回 true；否则，返回 false。

从以下版本开始：

JDK1.0

另请参见：

Object.hashCode(),Hashtable

toString

```
public String toString()
```

将此 Font 对象转换为 String 表示形式。

覆盖：

类 Object 中的 toString

返回：

此 Font 对象的 String 表示形式。

从以下版本开始：

JDK1.0

---

getNumGlyphs

```
public int getNumGlyphs()
```

返回此 Font 中的字形数量。此 Font 的字形代码，范围从 0 到 getNumGlyphs() - 1。

返回：

此 Font 的字形数量。

从以下版本开始：

1.2

---

getMissingGlyphCode

```
public int getMissingGlyphCode()
```

返回此 Font 不具有指定的 unicode 字形时所使用的 glyphCode 代码点。

返回：

此 Font 的 glyphCode。

从以下版本开始：

1.2

---

getBaselineFor

public byte getBaselineFor(char c)

返回适合用来显示此字符的基线。

大字体可以支持不同的书写系统，并且每个系统都可以使用不同的基线。字符参数确定要使用的书写系统。客户不应该假定所有字符都使用相同的基线。

参数：

c - 用来标识书写系统的字符

返回：

适合于指定字符的基线。

从以下版本开始：

1.2

另请参见：

LineMetrics.getBaselineOffsets(), ROMAN_BASELINE, CENTER_BASELINE, HANGING_BASELINE

---

getAttributes

public Map<TextAttribute,?> getAttributes()

返回此 Font 中可用的字体属性的映射。属性包括诸如连字和字形替换之类的事情。

返回：

此 Font 的属性映射。

---

getAvailableAttributes

public AttributedCharacterIterator.Attribute[] getAvailableAttributes()

返回由此 Font 支持的所有属性的键。这些属性可以用来派生其他字体。

返回：

一个数组，包含受此 Font 支持的所有属性的键。

从以下版本开始：

1.2

---

deriveFont

```
public Font deriveFont(int style,
                       float size)
```

通过复制此 Font 对象并应用新样式和大小，创建一个新 Font 对象。

参数：

style - 新 Font 的样式

size - 新 Font 的大小

返回：

新 Font 对象。

从以下版本开始：

1.2

deriveFont

```
public Font deriveFont(int style,
                       AffineTransform trans)
```

通过复制此 Font 对象并应用新样式和变换，创建一个新 Font 对象。

参数：

style - 新 Font 的样式

trans - 与新 Font 相关的 AffineTransform

返回：

一个新 Font 对象。

抛出：

IllegalArgumentException - 如果 trans 为 null

从以下版本开始：

1.2

deriveFont

```
public Font deriveFont(float size)
```

通过复制当前 Font 对象并应用新的大小，创建一个新 Font 对象。

参数：

size - 新 Font 的大小。

返回：

新 Font 对象。

从以下版本开始：

1.2

---

deriveFont

```
public Font deriveFont(AffineTransform trans)
```

通过复制当前 Font 对象并应用新的变换，创建一个新 Font 对象。

参数：

trans - 与新 Font 相关的 AffineTransform

返回：

一个新的 Font 对象。

抛出：

IllegalArgumentException - 如果 trans 为 null

从以下版本开始：

1.2

---

deriveFont

```
public Font deriveFont(int style)
```

通过复制当前的 Font 对象，并应用新的样式，创建一个新 Font 对象。

参数：

style - 新 Font 的样式

返回：

新 Font 对象。

从以下版本开始：

1.2

---

deriveFont

public Font deriveFont(Map<? extends AttributedCharacterIterator.Attribute,?> attributes)

通过复制当前 Font 对象并应用新的字体属性集，创建一个新 Font 对象。

参数：

attributes - 为新 Font 所启用的属性映射

返回：

新 Font 对象。

从以下版本开始：

1.2

---

canDisplay

public boolean canDisplay(char c)

检查此 Font 是否具有指定字符的字形。

注：此方法无法处理增补字符。要支持所有 Unicode 字符，包括增补字符，可以使用 canDisplay(int) 方法或 canDisplayUpTo 方法。

参数：

c - 需要字形的字符

返回：

如果此 Font 具有该字符的字形，则返回 true；否则返回 false。

从以下版本开始：

1.2

---

canDisplay

public boolean canDisplay(int codePoint)

检查此 Font 是否具有指定字符的字形。

参数：

codePoint - 需要字形的字符（Unicode 代码点）。

返回：

如果此 Font 具有该字符的字形，则返回 true；否则返回 false。

抛出：

IllegalArgumentException - 如果代码点不是一个有效的 Unicode 代码点。

从以下版本开始：

1.5

另请参见：

Character.isValidCodePoint(int)

canDisplayUpTo

```
public int canDisplayUpTo(String str)
```

指示此 Font 是否可以显示指定的 String。对于使用 Unicode 编码的字符串，知道特定字体是否可以显示该字符串是很重要的。此方法返回 String str 中的一个偏移量，这是此 Font 不使用缺少的字形代码就无法显示的第一个字符。如果 Font 可以显示所有字符，则返回 -1。

参数：

str - 一个 String 对象

返回：

str 中的一个偏移量，指向 str 中此 Font 无法显示的第一个字符；如果此 Font 可以显示 str 中的所有字符，则返回 -1。

从以下版本开始：

1.2

canDisplayUpTo

```
public int canDisplayUpTo(char[] text,
                          int start,
                          int limit)
```

指示此 Font 是否可以显示指定 text 中从 start 开始至 limit 结束的所有字符。此方法是一次方便的重载。

参数：

text - 指定 char 值的数组

start - 指定 char 值数组中的指定起始偏移量 (in char)

limit - 指定 char 值数组中的指定结束偏移量 (in char)

返回：

text 中的偏移量，指向此 Font 无法显示的 text 中的第一个字符；如果此 Font 可以显示 text 中的所有字符，则返回 -1。

从以下版本开始：

1.2

canDisplayUpTo

```
public int canDisplayUpTo(CharacterIterator iter,
                          int start,
                          int limit)
```

指示此 Font 是否可以显示由 iter 指定的文本（从 start 开始至 limit 结束）。

参数：

iter - CharacterIterator 对象

start - 指定 CharacterIterator 中的指定起始偏移量。

limit - 指定 CharacterIterator 中的指定结束偏移量。

返回：

iter 中的偏移量，它指向 iter 中此 Font 无法显示的第一个字符；如果此 Font 可以显示 iter 中的所有字符，则返回 -1。

从以下版本开始：

1.2

getItalicAngle

```
public float getItalicAngle()
```

返回此 Font 的斜角。斜角是与此 Font 的动作最匹配的 caret 的反斜面。

返回：

此 Font 的 ITALIC 样式的角度

另请参见：

TextAttribute.POSTURE

hasUniformLineMetrics

```
public boolean hasUniformLineMetrics()
```

检查此 Font 是否具有统一的行规格。逻辑 Font 可以是复合字体，这意味着它由不同的物理字体组成，可以覆盖不同的代码范围。其中每一种字体都可能有不同的 LineMetrics。如果逻辑 Font 是单一字体，则规格将是统一的。

返回：

如果此 Font 具有统一的行规格，则返回 true；否则返回 false。

getLineMetrics

```
public LineMetrics getLineMetrics(String str,
                                  FontRenderContext frc)
```

返回一个使用指定 String 和 FontRenderContext 创建的 LineMetrics 对象。

参数：

str - 指定的 String

frc - 指定的 FontRenderContext

返回：

使用指定 String 和 FontRenderContext 创建的 LineMetrics 对象。

getLineMetrics

```
public LineMetrics getLineMetrics(String str,
                                  int beginIndex,
                                  int limit,
                                  FontRenderContext frc)
```

返回使用指定参数创建的 LineMetrics 对象。

参数：

str - 指定的 String

beginIndex - str 的初始偏移量

limit - str 的结束偏移量

frc - 指定的 FontRenderContext

返回：

使用指定参数创建的 LineMetrics 对象。

getLineMetrics

```
public LineMetrics getLineMetrics(char[] chars,
                                  int beginIndex,
                                  int limit,
                                  FontRenderContext frc)
```

返回使用指定参数创建的 LineMetrics 对象。

参数：

chars - 字符数组

beginIndex - chars 的初始偏移量

limit - chars 的结束偏移量

frc - 指定的 FontRenderContext

返回：

使用指定参数创建的 LineMetrics 对象。

getLineMetrics

```
public LineMetrics getLineMetrics(CharacterIterator ci,
                                  int beginIndex,
                                  int limit,
                                  FontRenderContext frc)
```

返回使用指定参数创建的 LineMetrics 对象。

参数：

ci - 指定的 CharacterIterator

beginIndex - ci 中的初始偏移量

limit - ci 的结束偏移量

frc - 指定的 FontRenderContext

返回：

使用指定参数创建的 LineMetrics 对象。

getStringBounds

```
public Rectangle2D getStringBounds(String str,
```

```
                                                    FontRenderContext frc)
```

返回指定 FontRenderContext 中指定 String 的逻辑边界。逻辑边界包含 origin、ascent、advance 和 height，其中包括了 leading。逻辑边界并不总是包围所有文本。例如，在某些语言和字体中，accent 标记可以位于 ascent 之上，或 descent 之下。要得到可视的边界框（它包围了所有文本），可以使用 TextLayout 的 getBounds 方法。

注：返回的边界在相对于基线的坐标中（请参阅 class notes）。

参数：

str - 指定的 String

frc - 指定的 FontRenderContext

返回：

一个 Rectangle2D，它是指定 FontRenderContext 中指定 String 的边界框。

从以下版本开始：

1.2

另请参见：

FontRenderContext,
createGlyphVector(java.awt.font.FontRenderContext,
java.lang.String)

getStringBounds

```
public Rectangle2D getStringBounds(String str,
                                   int beginIndex,
                                   int limit,
                                   FontRenderContext frc)
```

返回指定 FontRenderContext 中指定 String 的逻辑边界。逻辑边界包含 origin、ascent、advance 和 height，其中包括了 leading。逻辑边界并不总是包围所有文本。例如，在某些语言和字体中，accent 标记可以位于 ascent 之上，或 descent 之下。要得到可视的边界框（它包围了所有文本），可以使用 TextLayout 的 getBounds 方法。

注：返回的边界在相对于基线的坐标中（请参阅 class notes）。

参数：

str - 指定的 String
beginIndex - str 的初始偏移量
limit - str 的结束偏移量
frc - 指定的 FontRenderContext
返回：
一个 Rectangle2D，它是指定 FontRenderContext 中指定 String 的边界框。
抛出：
IndexOutOfBoundsException - 如果 beginIndex 小于零，或 limit 大于 str 的长度，或 beginIndex 大于 limit。
从以下版本开始：
1.2
另请参见：
FontRenderContext,
createGlyphVector(java.awt.font.FontRenderContext,
java.lang.String)

getStringBounds

```
public Rectangle2D getStringBounds(char[] chars,
                                   int beginIndex,
                                   int limit,
                                   FontRenderContext frc)
```

返回指定 FontRenderContext 指定字符数组的逻辑边界。逻辑边界包含 origin、ascent、advance 和 height，其中包括了 leading。逻辑边界并不总是包围所有文本。例如，在某些语言和字体中，accent 标记可以位于 ascent 之上，或 descent 之下。要得到可视的边界框（它包围了所有文本），可以使用 TextLayout 的 getBounds 方法。

注：返回的边界在相对于基线的坐标中（请参阅 class notes）。

参数：
chars - 字符数组
beginIndex - 字符数组的初始偏移量
limit - 字符数组的结束偏移量
frc - 指定的 FontRenderContext
返回：

一个 Rectangle2D，它是指定 FontRenderContext 中指定字符数组的边界框。

抛出：

IndexOutOfBoundsException - 如果 beginIndex 小于零，或 limit 大于 chars 的长度，或 beginIndex 大于 limit。

从以下版本开始：

1.2

另请参见：

FontRenderContext,
createGlyphVector(java.awt.font.FontRenderContext, java.lang.String)

getStringBounds

```
public Rectangle2D getStringBounds(CharacterIterator ci,
                                   int beginIndex,
                                   int limit,
                                   FontRenderContext frc)
```

返回指定 FontRenderContext 中针对指定 CharacterIterator 进行索引的字符的逻辑边界。逻辑边界包含 origin、ascent、advance 和 height，其中包括了 leading。逻辑边界并不总是包围所有文本。例如，在某些语言和字体中，accent 标记可以位于 ascent 之上，或 descent 之下。要得到可视的边界框（它包围了所有文本），可以使用 TextLayout 的 getBounds 方法。

注：返回的边界在相对于基线的坐标中（请参阅 class notes）。

参数：

ci - 指定的 CharacterIterator
beginIndex - ci 中的初始偏移量
limit - ci 中的结束偏移量
frc - 指定的 FontRenderContext

返回：

一个 Rectangle2D，它是指定 FontRenderContext 中针对指定 CharacterIterator 进行索引的字符的边界框。

抛出：

IndexOutOfBoundsException - 如果 beginIndex 小于 ci 的起始索引，或 limit 大于 ci 的结束索引，或 beginIndex 大于 limit

从以下版本开始：
1.2
另请参见：
FontRenderContext,
createGlyphVector(java.awt.font.FontRenderContext,
java.lang.String)

getMaxCharBounds

```
public Rectangle2D getMaxCharBounds(FontRenderContext frc)
```

返回最大边界定义在 FontRenderContext 中的字符的边界。

注：返回的边界在相对于基线的坐标中（请参阅 class notes）。

参数：
frc - 指定的 FontRenderContext
返回：
一个 Rectangle2D，它是具有最大边界的字符的边界框。

createGlyphVector

```
public GlyphVector createGlyphVector(FontRenderContext frc,
                                     String str)
```

根据此 Font 中的 Unicode cmap 将字符一一映射到字形，从而创建一个 GlyphVector。除了字形到字符的映射之外，此方法不做任何其他处理。这意味着，此方法对于某些脚本是无用的，如 Arabic、Hebrew、Thai 和 Indic，它们要求进行重排序、整形或连字替换。
参数：
frc - 指定的 FontRenderContext
str - 指定的 String
返回：
使用指定 String 和指定 FontRenderContext 创建的新 GlyphVector。

createGlyphVector

```
public GlyphVector createGlyphVector(FontRenderContext frc,
                                     char[] chars)
```

根据此 Font 中的 Unicode cmap 将字符一一映射到字形，从而创建一个 GlyphVector。除了字形到字符的映射之外，此方法不做任何其他处理。这意味着，此方法对于某些脚本是无用的，如 Arabic、Hebrew、Thai 和 Indic，它们要求进行重排序、整形或连字替换。

参数：

frc - 指定的 FontRenderContext

chars - 指定的字符数组

返回：

使用指定字符数组和指定 FontRenderContext 创建的 GlyphVector。

createGlyphVector

```
public GlyphVector createGlyphVector(FontRenderContext frc,
                                     CharacterIterator ci)
```

根据此 Font 中的 Unicode cmap 将指定字符一一映射到字形，从而创建一个 GlyphVector。除了字形到字符的映射之外，此方法不做任何其他处理。这意味着，此方法对于某些脚本是无用的，如 Arabic、Hebrew、Thai 和 Indic，它们要求进行重排序、整形或连字替换。

参数：

frc - 指定的 FontRenderContext

ci - 指定的 CharacterIterator

返回：

使用指定 CharacterIterator 和指定 FontRenderContext 创建的 GlyphVector。

createGlyphVector

```
public GlyphVector createGlyphVector(FontRenderContext frc,
                                     int[] glyphCodes)
```

根据此 Font 中的 Unicode cmap 将字符一一映射到字形，从而创建一个 GlyphVector。除了字形到字符的映射之外，此方法不做任何其他处理。这意味着，此方法对于某些脚本是无用的，如 Arabic、Hebrew、Thai 和 Indic，它们要求进行重排序、整形或连字替换。

参数：

frc - 指定的 FontRenderContext

glyphCodes - 指定的整数数组

返回：

使用指定整数数组和指定 FontRenderContext 创建的 GlyphVector。

layoutGlyphVector

```
public GlyphVector layoutGlyphVector(FontRenderContext frc,
                                     char[] text,
                                     int start,
                                     int limit,
                                     int flags)
```

返回一个新 GlyphVector 对象，执行完整的文本布局（如有可能）。复杂文本要求有完整布局，如 Arabic 或 Hindi。对不同脚本的支持取决于字体和实现。

Layout requires bidi analysis, as performed by Bidi 只应该在具有统一方向的文本上执行。用 flags 参数来指示方向，通过使用 LAYOUT_RIGHT_TO_LEFT 来指示从右到左（Arabic 和 Hebrew）的运行方向，或通过使用 LAYOUT_LEFT_TO_RIGHT 来指示从左到右 (English) 的运行方向。

此外，有些操作（比如 Arabic 整形）需要上下文，这样在开始和结束处的字符才会有合适的形状。有时，在所提供范围之外的缓冲区中的数据并不是有效数据。可以将值 LAYOUT_NO_START_CONTEXT 和 LAYOUT_NO_LIMIT_CONTEXT 添加到 flags 参数中，分别指示在 start 之前或 limit 之后的文本不应该作为上下文进行检查。

flags 参数的所有其他值均被保留。

参数：

frc - 指定的 FontRenderContext

text - 要布局的文本

start - GlyphVector 所要使用的文本的起始处
limit - GlyphVector 所要使用的文本的结束处
flags - 如上所述的控制标志

返回：

一个新 GlyphVector，表示 start 和 limit 之间的文本，具有经过选择和定位的字形，以便以最佳方式表示文本

抛出：

ArrayIndexOutOfBoundsException - 如果 start 或 limit 超出了边界

从以下版本开始：

1.4

另请参见：

Bidi, LAYOUT_LEFT_TO_RIGHT, LAYOUT_RIGHT_TO_LEFT, LAYOUT_NO_START_CONTEXT, LAYOUT_NO_LIMIT_CONTEXT

finalize

```
protected void finalize()
                    throws Throwable
```

移除本机 Font 对象。

覆盖：

类 Object 中的 finalize

抛出：

Throwable - 此方法抛出的 Exception

Frame 类

Frame 是带有标题和边框的顶层窗口。

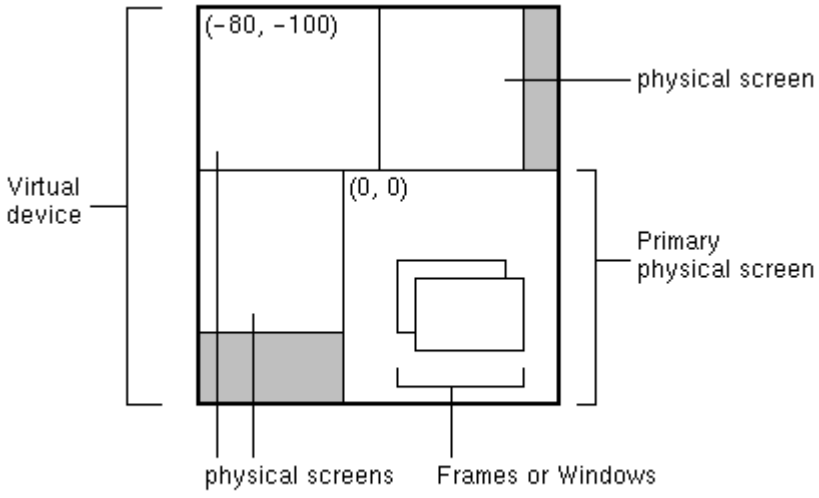
窗体的大小包括为边框指定的所有区域。边框区域的尺寸可以使用 getInsets 方法获得，但是，由于这些尺寸与平台相关，因此在通过调用 pack 或 show 将窗体设置为可显示之前，将无法获得有效的 insets 值。由于窗体的总大小包括了边框区，因此边框实际上遮掩了窗体的部分区域，并将可用于在矩形中呈现和/或显示子部件的区域限制在一个矩形内，该矩形左上角的位置为 (insets.left, insets.top)，宽度为 width - (insets.left + insets.right)，长度为 height - (insets.top + insets.bottom)。

窗体的默认布局为 BorderLayout。

使用 setUndecorated，窗体可以关闭本机装饰（即 Frame 和 Titlebar）。但只有在窗体不是 displayable 时才能这样做。

在多屏幕环境中，通过使用 Frame(GraphicsConfiguration) 或 Frame(String title, GraphicsConfiguration) 构造 Frame，可以在不同的屏幕设备上创建 Frame。GraphicsConfiguration 对象是目标屏幕设备的 GraphicsConfiguration 对象之一。

在虚拟设备多屏幕环境中（其中桌面区域可以跨越多物理屏幕设备），所有配置的边界都是相对于虚拟坐标系的。虚拟坐标系的原点位于主物理屏幕的左上角。是否使用负坐标取决于主物理屏幕在虚拟设备中的位置，如下图所示。



在此环境中调用 setLocation 时，必须传递一个虚拟坐标到此方法中。类似地，对 Frame 调用 getLocationOnScreen 将返回虚拟设备坐标。调用 GraphicsConfiguration 的 getBounds 方法可查找它在虚拟坐标系中的原点。

以下代码将 Frame 的位置设置为 (10, 10)（相对于相应 GraphicsConfiguration 的物理屏幕的原点）。如果不考虑 GraphicsConfiguration 的边界，则 Frame 的位置将被设置为 (10, 10)（相对于虚拟坐标系），并出现在主物理屏幕上，主物理屏幕不同于指定的 GraphicsConfiguration 的物理屏幕。

```
Frame f = new Frame(GraphicsConfiguration gc);
Rectangle bounds = gc.getBounds();
f.setLocation(10 + bounds.x, 10 + bounds.y);
```

窗体能够生成以下类型的 WindowEvent：

WINDOW_OPENED

WINDOW_CLOSING:

在处理此事件时，如果程序没有显式地隐藏或释放窗口，则取消窗口关闭操作。

WINDOW_CLOSED

WINDOW_ICONIFIED
WINDOW_DEICONIFIED
WINDOW_ACTIVATED
WINDOW_DEACTIVATED
WINDOW_GAINED_FOCUS
WINDOW_LOST_FOCUS
WINDOW_STATE_CHANGED

字段详细信息

DEFAULT_CURSOR

```
@Deprecated
    public static final int DEFAULT_CURSOR
```

已过时。 *由 Cursor.DEFAULT_CURSOR 取代。*
另请参见：
常量字段值

CROSSHAIR_CURSOR

```
@Deprecated
    public static final int CROSSHAIR_CURSOR
```

已过时。 *由 Cursor.CROSSHAIR_CURSOR 取代。*
另请参见：
常量字段值

TEXT_CURSOR

```
@Deprecated
    public static final int TEXT_CURSOR
```

已过时。 *由 Cursor.TEXT_CURSOR 取代。*
另请参见：

常量字段值

WAIT_CURSOR

```
@Deprecated
public static final int WAIT_CURSOR
```

已过时。 *由 Cursor.WAIT_CURSOR 取代。*

另请参见：

常量字段值

SW_RESIZE_CURSOR

```
@Deprecated
public static final int SW_RESIZE_CURSOR
```

已过时。 *由 Cursor.SW_RESIZE_CURSOR 取代。*

另请参见：

常量字段值

SE_RESIZE_CURSOR

```
@Deprecated
public static final int SE_RESIZE_CURSOR
```

已过时。 *由 Cursor.SE_RESIZE_CURSOR 取代。*

另请参见：

常量字段值

NW_RESIZE_CURSOR

```
@Deprecated
public static final int NW_RESIZE_CURSOR
```

已过时。 *由 Cursor.NW_RESIZE_CURSOR 取代。*
另请参见：
常量字段值

NE_RESIZE_CURSOR

```
@Deprecated
    public static final int NE_RESIZE_CURSOR
```

已过时。 *由 Cursor.NE_RESIZE_CURSOR 取代。*
另请参见：
常量字段值

N_RESIZE_CURSOR

```
@Deprecated
    public static final int N_RESIZE_CURSOR
```

已过时。 *由 Cursor.N_RESIZE_CURSOR 取代。*
另请参见：
常量字段值

S_RESIZE_CURSOR

```
@Deprecated
    public static final int S_RESIZE_CURSOR
```

已过时。 *由 Cursor.S_RESIZE_CURSOR 取代。*
另请参见：
常量字段值

W_RESIZE_CURSOR

```
@Deprecated
    public static final int W_RESIZE_CURSOR
```

已过时。 *由 Cursor.W_RESIZE_CURSOR 取代。*
另请参见：
常量字段值

E_RESIZE_CURSOR

```
@Deprecated
    public static final int E_RESIZE_CURSOR
```

已过时。 *由 Cursor.E_RESIZE_CURSOR 取代。*
另请参见：
常量字段值

HAND_CURSOR

```
@Deprecated
    public static final int HAND_CURSOR
```

已过时。 *由 Cursor.HAND_CURSOR 取代。*
另请参见：
常量字段值

MOVE_CURSOR

```
@Deprecated
    public static final int MOVE_CURSOR
```

已过时。 *由 Cursor.MOVE_CURSOR 取代。*
另请参见：
常量字段值

NORMAL

public static final int NORMAL

窗体处于 "normal" 状态。此符号常量指定所有状态位均被清除的窗体状态。

另请参见：

setExtendedState(int),getExtendedState(),常量字段值

ICONIFIED

public static final int ICONIFIED

此状态位指示将窗体图标化。

另请参见：

setExtendedState(int),getExtendedState(),常量字段值

MAXIMIZED_HORIZ

public static final int MAXIMIZED_HORIZ

此状态位指示在水平方向将窗体最大化。

从以下版本开始：

1.4

另请参见：

setExtendedState(int),getExtendedState(),常量字段值

MAXIMIZED_VERT

public static final int MAXIMIZED_VERT

此状态位指示在垂直方向将窗体最大化。

从以下版本开始：

1.4

另请参见：

setExtendedState(int),getExtendedState(),常量字段值

MAXIMIZED_BOTH

```
public static final int MAXIMIZED_BOTH
```

此状态位掩码指示将窗体完全最大化（水平和垂直方向）。它只是 MAXIMIZED_VERT | MAXIMIZED_HORIZ 的一种便捷别名。

注意，要正确测试窗体是否完全最大化，使用以下方法：

```
(state & Frame.MAXIMIZED_BOTH) == Frame.MAXIMIZED_BOTH
```

要测试窗体是否在*某个* 方向最大化，使用：

```
(state & Frame.MAXIMIZED_BOTH) != 0
```

从以下版本开始：
1.4
另请参见：
setExtendedState(int),getExtendedState(),常量字段值

构造方法详细信息

Frame

```
public Frame()
      throws HeadlessException
```

构造一个最初不可见的 Frame 新实例（）。Frame 的标题为空。
抛出：
HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时
另请参见：
GraphicsEnvironment.isHeadless(), Component.setSize(int, int), Component.setVisible(boolean)

Frame

```
public Frame(GraphicsConfiguration gc)
```

使用指定的 GraphicsConfiguration 构造一个最初不可见的新 Frame。

参数：

gc - 目标屏幕设备的 GraphicsConfiguration。如果 gc 为 null，则假定它为系统默认的 GraphicsConfiguration。

抛出：

IllegalArgumentException - 如果 gc 不是来自屏幕设备。

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

从以下版本开始：

1.3

另请参见：

GraphicsEnvironment.isHeadless()

Frame

```
public Frame(String title)
        throws HeadlessException
```

构造一个新的、最初不可见的、具有指定标题的 Frame 对象。

参数：

title - 要显示在窗体边框中的标题。null 值视为空字符串 ""。

抛出：

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

另请参见：

GraphicsEnvironment.isHeadless(), Component.setSize(int, int), Component.setVisible(boolean),GraphicsConfiguration.getBounds()

Frame

```
public Frame(String title,
             GraphicsConfiguration gc)
```

构造一个新的、初始不可见的、具有指定标题和 GraphicsConfiguration 的

Frame 对象。

参数：

title - 要显示在窗体边框中的标题。null 值视为空字符串 ""。

gc - 目标屏幕设备的 GraphicsConfiguration。如果 gc 为 null，则假定它为系统默认的 GraphicsConfiguration。

抛出：

IllegalArgumentException - 如果 gc 不是来自屏幕设备。

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

从以下版本开始：

1.3

另请参见：

GraphicsEnvironment.isHeadless(), Component.setSize(int, int), Component.setVisible(boolean),GraphicsConfiguration.getBounds()

方法详细信息

addNotify

```
public void addNotify()
```

通过将此窗体连接到本机屏幕资源，使其成为可显示的。如果窗体是可显示的，则其所有子级也将成为可显示的。此方法由工具包内部调用，不应直接由程序调用。

覆盖：

类 Window 中的 addNotify

另请参见：

Component.isDisplayable(),removeNotify()

---

getTitle

```
public String getTitle()
```

获得窗体的标题。标题显示在窗体的边框中。

返回：

此窗体的标题，如果此窗体没有标题，则返回一个空字符串 ("") 。

另请参见：

setTitle(String)

setTitle

public void setTitle(String title)

将此窗体的标题设置为指定的字符串。

参数：

title - 要显示在窗体边框中的标题。null 值视为空字符串 ""。

另请参见：

getTitle()

getIconImage

public Image getIconImage()

返回要作为此窗体图标显示的图像。

此方法已废弃，保留它只是为了向后兼容性。使用 Window.getIconImages() 代替。

如果将几个图像的列表指定为 Window 的图标，则此方法返回列表的第一项。

返回：

此窗体的图标图像；如果此窗体没有图标图像，则返回 null。

另请参见：

setIconImage(Image), Window.getIconImages(), Window.setIconImages(java.util.List)

setIconImage

public void setIconImage(Image image)

设置要作为此窗口图标显示的图像。

将单个图像指定为窗口的图标时，可以使用此方法代替 setIconImages()。

以下语句：

```
setIconImage(image);
```

等价于：

```
ArrayList imageList = new ArrayList ();
imageList.add(image);
setIconImages(imageList);
```

注：根据上下文的不同（例如，窗口装饰、窗口列表、任务栏等），本机窗口系统可以使用不同尺寸的不同图像表示一个窗口。也可以对所有上下文使用一个图像，或者根本不用图像。

覆盖：
类 Window 中的 setIconImage
参数：
image - 要显示的图标图像。
另请参见：
Window.setIconImages(java.util.List),Window.getIconImages()

---

getMenuBar

```
public MenuBar getMenuBar()
```

获取此窗体的菜单栏。
返回：
此窗体的菜单栏；如果此窗体没有菜单栏，则返回 null。
另请参见：
setMenuBar(MenuBar)

---

setMenuBar

public void setMenuBar(MenuBar mb)

将此窗体的菜单栏设置为指定的菜单栏。

参数：

mb - 正被设置的菜单栏。如果此参数为 null，则移除此窗体上现有的所有菜单栏。

另请参见：

getMenuBar()

isResizable

public boolean isResizable()

指示此窗体是否可由用户调整大小。在默认情况下，所有窗体最初都可以调整大小。

返回：

如果用户可以调整窗体的大小，则返回 true；否则返回 false。

另请参见：

setResizable(boolean)

setResizable

public void setResizable(boolean resizable)

设置此窗体是否可由用户调整大小。

参数：

resizable - 如果此窗体是可调整大小的，则为 true；否则为 false。

另请参见：

isResizable()

setState

public void setState(int state)

设置此窗体的状态（已废弃）。

在较早的 JDK 版本中，窗体的状态只能为 NORMAL 或 ICONIFIED。自从 JDK 1.4 以来，受支持的窗体状态集合扩大了，窗体状态表示为逐位掩码。

为了与原有的程序兼容，此方法仍然接受 Frame.NORMAL 和 Frame.ICONIFIED，但它只改变窗体的图标状态，而不影响窗体其他方面的状态。

参数：

state - Frame.NORMAL 或 Frame.ICONIFIED。

另请参见：

getState(),setExtendedState(int)

setExtendedState

public void setExtendedState(int state)

设置此窗体的状态。该状态表示为逐位掩码。

- NORMAL
  指示不设置任何状态位。
- ICONIFIED
- MAXIMIZED_HORIZ
- MAXIMIZED_VERT
- MAXIMIZED_BOTH
  串连 MAXIMIZED_HORIZ 和 MAXIMIZED_VERT。

注意，如果该状态在给定平台上不受支持，则什么也不会发生。应用程序可以通过 java.awt.Toolkit#isFrameStateSupported(int state) 方法确定特定的状态是否可用。

参数：

state - 窗体状态常量的逐位掩码

从以下版本开始：

1.4

另请参见：

getExtendedState(),Toolkit.isFrameStateSupported(int)

getState

public int getState()

获取此窗体的状态（已废弃）。

在较早的 JDK 版本中，窗体的状态只能为 NORMAL 或 ICONIFIED。自从 JDK 1.4 以来，受支持的窗体状态集合扩大了，窗体状态表示为逐位掩码。

为了与原有的程序兼容，此方法仍然返回 Frame.NORMAL 和 Frame.ICONIFIED，但它只报告窗体的图标状态，而不报告窗体其他方面的状态。

返回：

Frame.NORMAL 或 Frame.ICONIFIED。

另请参见：

setState(int),getExtendedState()

getExtendedState

public int getExtendedState()

获取此窗体的状态。该状态表示为逐位掩码。

- NORMAL
  指示不设置任何状态位。
- ICONIFIED
- MAXIMIZED_HORIZ
- MAXIMIZED_VERT
- MAXIMIZED_BOTH
  串连 MAXIMIZED_HORIZ 和 MAXIMIZED_VERT。

返回：

窗体状态常量的逐位掩码

从以下版本开始：

1.4

另请参见：

setExtendedState(int)

setMaximizedBounds

public void setMaximizedBounds(Rectangle bounds)

设置此窗体的最大化边界。

当窗体处于最大化状态时，系统提供默认边界。此方法允许重写系统提供的部分或全部值。

如果 bounds 为 null，则接受系统提供的边界。如果 bound 不为 null，则可以重写系统提供的某些值而接受其他值，这通过将希望从系统接受的那些字段设置为 Integer.MAX_VALUE 来完成。

在某些系统中，只考虑部分边界的大小。

参数：

bounds - 最大化状态的边界

从以下版本开始：

1.4

另请参见：

getMaximizedBounds()

getMaximizedBounds

public Rectangle getMaximizedBounds()

获取此窗体的最大化边界。有些字段可能包含 Integer.MAX_VALUE，指示必须使用系统为该字段提供的值。

返回：

此窗体的最大化边界；可以为 null

从以下版本开始：

1.4

另请参见：

setMaximizedBounds(Rectangle)

setUndecorated

public void setUndecorated(boolean undecorated)

禁用或启用此窗体的装饰。只有在窗体不可显示时才调用此方法。

参数：

undecorated - 如果没有启用窗体装饰，则为 true；如果启用了窗体装饰，则为 false。

抛出：

IllegalComponentStateException - 如果窗体是可显示的。

从以下版本开始：

1.4

另请参见：

isUndecorated(), Component.isDisplayable(), JFrame.setDefaultLookAndFeelDecorated(boolean)

---

isUndecorated

public boolean isUndecorated()

指示此窗体是否未装饰。在默认情况下，所有窗体初始时都是已装饰的。

返回：

如果窗体未装饰，则返回 true；否则返回 false。

从以下版本开始：

1.4

另请参见：

setUndecorated(boolean)

---

remove

public void remove(MenuComponent m)

从此窗体移除指定的菜单栏。

指定者：

接口 MenuContainer 中的 remove

覆盖：

类 Component 中的 remove

参数：

m - 要移除的菜单组件。如果 m 为 null，则不执行任何操作。

另请参见：
Component.add(PopupMenu)

removeNotify

```
public void removeNotify()
```

通过移除与本机屏幕资源的连接，将此窗体设置为不可显示的。如果窗体是不可显示的，则其所有子级也将成为不可显示的。此方法由工具包内部调用，不应直接由程序调用。

覆盖：
类 Window 中的 removeNotify

另请参见：
Component.isDisplayable(),addNotify()

paramString

```
protected String paramString()
```

返回表示此 Frame 状态的字符串。此方法仅用于调试目的，对于各个实现，所返回字符串的内容和格式可能有所不同。返回的字符串可能为空，但不可能为 null。

覆盖：
类 Container 中的 paramString

返回：
此窗体的参数字符串

setCursor

```
@Deprecated
public void setCursor(int cursorType)
```

已过时。 *从 JDK version 1.1 开始，由 Component.setCursor(Cursor) 取代。*

getCursorType

@Deprecated
public int getCursorType()

已过时。 *从 JDK version 1.1 开始，由 Component.getCursor() 取代。*

getFrames

public static Frame[] getFrames()

返回一个此应用程序创建的所有 Frame 所组成的数组。如果从 applet 调用，则数组只包括该 applet 可访问的 Frame。

警告： 此方法可能返回系统创建的窗体，如 Swing 使用的共享的、隐藏的窗体。应用程序不应该假定这些窗口存在，也不应该假定与这些窗体有关的任何内容（如组件位置、LayoutManager 或序列化）存在。

注：若要获得没有所有者的窗口列表，包括没有所有者的 Dialog（在版本 1.6 中引入），请使用 Window.getOwnerlessWindows。

从以下版本开始：
1.2
另请参见：
Window.getWindows(sun.awt.AppContext),
Window.getOwnerlessWindows()

getAccessibleContext

public AccessibleContext getAccessibleContext()

获取与此窗体有关的 AccessibleContext。对于 frame，AccessibleContext 采用 AccessibleAWTFrame 的形式。如有必要，创建一个新的 AccessibleAWTFrame 实例。
指定者：
接口 Accessible 中的 getAccessibleContext
覆盖：
类 Window 中的 getAccessibleContext
返回：
用作此窗体 AccessibleContext 的 AccessibleAWTFrame。

GridBagLayout 类

GridBagLayout 类是一个灵活的布局管理器，它不要求组件的大小相同便可以将组件垂直、水平或沿它们的基线对齐。每个 GridBagLayout 对象维持一个动态的矩形单元网格，每个组件占用一个或多个这样的单元，该单元被称为显示区域。

每个由 GridBagLayout 管理的组件都与 GridBagConstraints 的实例相关联。Constraints 对象指定组件的显示区域在网格中的具体放置位置，以及组件在其显示区域中的放置方式。除了 Constraints 对象之外，GridBagLayout 还考虑每个组件的最小大小和首选大小，以确定组件的大小。

网格的总体方向取决于容器的 ComponentOrientation 属性。对于水平的从左到右的方向，网格坐标 (0,0) 位于容器的左上角，其中 X 向右递增，Y 向下递增。对于水平的从右到左的方向，网格坐标 (0,0) 位于容器的右上角，其中 X 向左递增，Y 向下递增。

为了有效使用网格包布局，必须自定义与组件关联的一个或多个 GridBagConstraints 对象。可以通过设置一个或多个实例变量来自定义 GridBagConstraints 对象：

GridBagConstraints.gridx、GridBagConstraints.gridy

指定包含组件显示区域的前导角的单元，在此显示区域中，位于网格原点的单元地址是 gridx = 0, gridy = 0。对于水平的从左到右的布局，组件的前导角是其左上角。对于水平的从右到左的布局，组件的前导角是其右上角。使用 GridBagConstraints.RELATIVE（默认值），指定会将组件直接放置在之前刚添加到容器中的组件的后面（沿 X 轴向为 gridx 或 Y 轴向为 gridy）。

GridBagConstraints.gridwidth、GridBagConstraints.gridheight

指定组件的显示区域中行（针对 gridwidth）或列（针对 gridheight）中的单元数。默认值为 1。使用 GridBagConstraints.REMAINDER 指定组件的显示区域，该区域的范围是从 gridx 到该行（针对 gridwidth）中的最后一个单元，或者从 gridy 到该列（针对 gridheight）中的最后一个单元。 使用 GridBagConstraints.RELATIVE 指定组件的显示区域，该区域的范围是从 gridx 到其所在行（针对 gridwidth）的倒数第二个单元，或者从 gridy 到其所在列（针对 gridheight）的倒数第二个单元。

GridBagConstraints.fill

当组件的显示区域大于组件的所需大小时，用于确定是否（以及如何）调整组件。可能的值为 GridBagConstraints.NONE（默认值）、GridBagConstraints.HORIZONTAL（加宽组件直到它足以在水平方向上填满其显示区域，但不更改其高度）、GridBagConstraints.VERTICAL（加高组件直到它足以在垂直方向上填满其显示区域，但不更改其宽度）和 GridBagConstraints.BOTH（使组件完全填满其显示区域）。

GridBagConstraints.ipadx、GridBagConstraints.ipady

指定布局中组件的内部填充，即对组件最小大小的添加量。组件的宽度至少为其最小宽

度加上 ipadx 像素。类似地，组件的高度至少为其最小高度加上 ipady 像素。

GridBagConstraints.insets

指定组件的外部填充，即组件与其显示区域边缘之间间距的最小量。

GridBagConstraints.anchor

指定组件应置于其显示区域中何处。可能的值有三种：绝对值、相对于方向的值和相对于基线的值。相对于方向的值是相对于容器的 ComponentOrientation 属性进行解释的，而绝对值则不然。相关于基线的值是相对于基线进行计算的。

GridBagConstraints.weightx、GridBagConstraints.weighty

用于确定分布空间的方式，这对于指定调整行为至关重要。除非在行 (weightx) 和列 (weighty) 中至少指定一个组件的权重，否则所有组件都会聚集在其容器的中央。这是因为，当权重为零（默认值）时，GridBagLayout 对象会将所有额外空间置于其单元网格和容器边缘之间。

每行可以有一条基线，具体取决于该行中具有有效基线并沿此基线对齐的组件（组件的锚值是 BASELINE、BASELINE_LEADING 或 BASELINE_TRAILING 其中之一）。如果行中没有具有有效基线的组件，则该行没有基线。

如果组件跨多行，则它与起始行（如果基线调整行为是 CONSTANT_ASCENT）或结束行（如果基线调整行为是 CONSTANT_DESCENT）的基线对齐。用来对齐组件的行称为主导行。

下图显示了基线布局并包括横跨行的组件：

此布局由三个组件组成：

起始于 0 行并结束于 1 行的面板。该面板有一个 CONSTANT_DESCENT 基线调整行为以及 BASELINE 的锚。因为基线调整行为是 CONSTANT_DESCENT，所以该面板的主要行是第 1 行。

两个按钮，每个按钮都带有 CENTER_OFFSET 基线调整行为和 BASELINE 的锚。

因为第二个按钮和面板共享相同的主要行，所以它们都沿其基线对齐。

使用一个相对于基线的值定位的组件调整不同于使用绝对值或相对于方向的值。组件更改的方式由主要行的基线更改方式指示。如果基于相同主导行的所有组件具有 CONSTANT_DESCENT 基线调整行为，则基线定位到显示区域底部；否则，基线定位到显示区域顶部。下述规则指示调整大小的行为：

位于基线上方的可调整大小的组件只能增长到与该基线一样高。例如，如果基线为 100 且位于顶部，则位于基线上方的可调整大小的组件的增长永远都不能超过 100 个单位。

同样地，位于基线下方的可调整大小的组件只能增长到和显示高度与基线之间的差值一样高。

仅当调整了大小的基线适应显示区域时，才能调整基线上具有 OTHER 基线调整行为的可调整大小组件。如果不能将基线放入显示区域，则该组件不能调整大小。

位于该基线上没有 OTHER 基线调整行为的组件只能增长到和显示高度 - 基线 + 组件基线一样高。

如果沿基线放置一个组件，但该组件没有有效的基线，那么它将在其空间中垂直居中对齐。同样地，如果已放置一个相对于基线的组件且行中的所有组件都没有有效的基线，则该组件垂直居中对齐。

下图显示了由网格包布局管理的十个组件（均为按钮）。图 2 显示水平方向从左到右的容器的布局，图 3 显示水平方向从右到左的容器的布局。

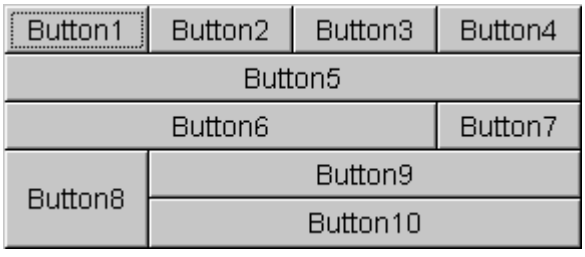


图 2：水平方向，从左到右

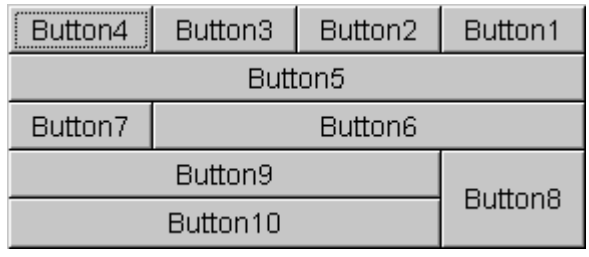


图 3：水平方向，从右到左

十个组件的每一个组件都会将与之相关的 GridBagConstraints 对象的 fill 字段设置为 GridBagConstraints.BOTH。此外，这些组件还具有以下非默认值约束 (Constraints)：

Button1、Button2、Button3： weightx = 1.0

Button4：weightx = 1.0、gridwidth = GridBagConstraints.REMAINDER

Button5：gridwidth = GridBagConstraints.REMAINDER

Button6：gridwidth = GridBagConstraints.RELATIVE

Button7：gridwidth = GridBagConstraints.REMAINDER

Button8：gridheight = 2、weighty = 1.0

Button9、Button 10：gridwidth = GridBagConstraints.REMAINDER

下面是实现上述示例的代码：

```
import java.awt.*;
import java.util.*;
import java.applet.Applet;

public class GridBagEx1 extends Applet {

    protected void makebutton(String name,
                              GridBagLayout gridbag,
                              GridBagConstraints c) {
        Button button = new Button(name);
        gridbag.setConstraints(button, c);
        add(button);
```

```
}

public void init() {
    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();

    setFont(new Font("SansSerif", Font.PLAIN, 14));
    setLayout(gridbag);

    c.fill = GridBagConstraints.BOTH;
    c.weightx = 1.0;
    makebutton("Button1", gridbag, c);
    makebutton("Button2", gridbag, c);
    makebutton("Button3", gridbag, c);

      c.gridwidth = GridBagConstraints.REMAINDER; //end row
    makebutton("Button4", gridbag, c);

    c.weightx = 0.0;                    //reset to the default
    makebutton("Button5", gridbag, c); //another row

      c.gridwidth = GridBagConstraints.RELATIVE; //next-to-last in row
    makebutton("Button6", gridbag, c);

      c.gridwidth = GridBagConstraints.REMAINDER; //end row
    makebutton("Button7", gridbag, c);

      c.gridwidth = 1;                  //reset to the default
      c.gridheight = 2;
    c.weighty = 1.0;
    makebutton("Button8", gridbag, c);

    c.weighty = 0.0;                    //reset to the default
      c.gridwidth = GridBagConstraints.REMAINDER; //end row
      c.gridheight = 1;                 //reset to the default
    makebutton("Button9", gridbag, c);
    makebutton("Button10", gridbag, c);
```

```
            setSize(300, 100);
        }

        public static void main(String args[]) {
            Frame f = new Frame("GridBag Layout Example");
            GridBagEx1 ex1 = new GridBagEx1();

            ex1.init();

            f.add("Center", ex1);
            f.pack();
            f.setSize(f.getPreferredSize());
            f.show();
        }
    }
```

字段详细信息

MAXGRIDSIZE

```
protected static final int MAXGRIDSIZE
```

此字段不再用于保留数组和保持向后兼容性。以前，此是网格包布局布置的网格位置（既包括水平的位置，也包括垂直的位置）的最大数。当前实现不会对网格大小产生任何影响。

另请参见：

常量字段值

MINSIZE

```
protected static final int MINSIZE
```

网格包布局可以布置的最小网格。

另请参见：
常量字段值

PREFERREDSIZE

protected static final int PREFERREDSIZE

网格包布局可以布置的首选网格大小。

另请参见：
常量字段值

comptable

protected Hashtable<Component,GridBagConstraints> comptable

此哈希表维持组件与其网格包约束之间的关联。comptable 中的键是组件，值是 GridBagConstraints 的实例。

另请参见：
GridBagConstraints

defaultConstraints

protected GridBagConstraints defaultConstraints

此字段保持包含默认值的网格包约束实例，因此如果某个组件没有与其相关联的网格包约束，则会分配给该组件一个 defaultConstraints 的副本。

另请参见：
getConstraints(Component), setConstraints(Component, GridBagConstraints),lookupConstraints(Component)

layoutInfo

protected GridBagLayoutInfo layoutInfo

此字段保持网格包的布局信息。此字段中的信息基于最近验证的网格包。如果 layoutInfo 为 null，这指示网格包中不存在组件，或者即使存在，也是尚未经过验证的组件。

另请参见：

getLayoutInfo(Container, int)

columnWidths

public int[] columnWidths

此字段保持对列最小宽度的重写。如果此字段为非 null，则在计算全部最小列宽度之后将这些值应用到网格包。如果 columnWidths 的元素数多于列数，则在网格包中添加列以匹配 columnWidth 中的元素数。

另请参见：

getLayoutDimensions()

rowHeights

public int[] rowHeights

此字段保持对行最小高度的重写。如果此字段为非 null，则在计算全部最小行高度之后将这些值应用到网格包。如果 rowHeights 的元素数多于行数，则在网格包中添加行以匹配 rowHeights 中的元素数。

另请参见：

getLayoutDimensions()

columnWeights

public double[] columnWeights

此字段保持对列权重的重写。如果此字段为非 null，则在计算全部列权重之后将这些值应用到网格包。如果 columnWeights[i] 大于列 i 的权重，则将 columnWeights[i] 中的权重分配给列 i。如果 columnWeights 的元素数多于列数，则多余的元素将被忽略——而不会相应地创建更多列。

rowWeights

```
public double[] rowWeights
```

此字段保持对行权重的重写。如果此字段为非 null，则在计算全部行权重之后将这些值应用到网格包。如果 rowWeights[i] 大于行 i 的权重，则将 rowWeights[i] 中的权重分配给行 i。如果 rowWeights 的元素多于行数，则多余的元素将被忽略——它们不会导致更多行的创建。

构造方法详细信息

GridBagLayout

```
public GridBagLayout()
```

创建网格包布局管理器。

方法详细信息

setConstraints

```
public void setConstraints(Component comp,
                           GridBagConstraints constraints)
```

设置此布局中指定组件的约束条件。

参数：

comp - 要修改的组件

constraints - 要应用的约束条件

getConstraints

```
public GridBagConstraints getConstraints(Component comp)
```

获取指定组件的约束。返回实际 GridBagConstraints 对象的副本。

参数：

comp - 要查询的组件
返回：
此网格包布局中指定组件的约束；返回实际约束对象的副本

lookupConstraints

protected GridBagConstraints lookupConstraints(Component comp)

检索指定组件的约束。返回值不是副本，而是布局机制使用的实际 GridBagConstraints 对象。

如果 comp 不在 GridBagLayout 中，则返回一组默认的 GridBagConstraints。值为 null 的 comp 值是无效的，返回 null。

参数：
comp - 要查询的组件
返回：
指定组件的约束

getLayoutOrigin

public Point getLayoutOrigin()

在目标容器的图形坐标空间确定布局区域的原点。此值表示布局区域的左上角的像素坐标，不管容器的 ComponentOrientation 值如何。这与单元坐标给定的网格原点 (0,0) 不同。大多数应用程序并不直接调用此方法。
返回：
布局网格左上角的单元的图形原点
从以下版本开始：
JDK1.1
另请参见：
ComponentOrientation

getLayoutDimensions

```
public int[][] getLayoutDimensions()
```

确定布局网格的列宽度和行高度。

大多数应用程序并不直接调用此方法。

返回：

一个二维数组，包含布局列的宽度和布局行的高度

从以下版本开始：

JDK1.1

---

getLayoutWeights

```
public double[][] getLayoutWeights()
```

确定布局网格的行与列的权重。如果布局具有额外空间来填充，则权重用于计算给定的行或列可以伸展得超过首选大小多少。

大多数应用程序不直接调用此方法。

返回：

一个二维数组，表示布局列的水平权重和布局行的垂直权重。

从以下版本开始：

JDK1.1

---

location

```
public Point location(int x,
                      int y)
```

确定在布局网格中哪个单元包含由（x， y）指定的点。每个单元由其列索引（范围从 0 到列数减 1）和其行索引（范围从 0 到行数减 1）来标识。

如果（x， y）点位于网格的外部，则使用以下规则。如果 x 位于从左到右容器布局的左边或位于从右到左容器布局的右边，则列索引的返回值为0。如果 x 位于从左到右容器布局的右边或位于从右到左容器布局的左边，则列索引的返回值是列数。如果 y 位于布局的上边，则行索引的返

回值为零；如果 y 位于布局的下边，则行索引的返回值为行数。容器的方向由其 ComponentOrientation 属性确定。

参数：

x - 点的 *x* 坐标

y - 点的 *y* 坐标

返回：

索引的有序对，指示布局网格中的哪个单元包含点 (*x*,*y*)。

从以下版本开始：

JDK1.1

另请参见：

ComponentOrientation

---

addLayoutComponent

```
public void addLayoutComponent(String name,
                               Component comp)
```

无效，因为此布局管理器不使用每组件字符串。

指定者：

接口 LayoutManager 中的 addLayoutComponent

参数：

name - 要与组件关联的字符串

comp - 要添加的组件

---

addLayoutComponent

```
public void addLayoutComponent(Component comp,
                               Object constraints)
```

使用指定 constraints 对象将指定组件添加到布局中。注意，约束条件是可变的，因此缓存时应该复制。

指定者：

接口 LayoutManager2 中的 addLayoutComponent

参数：

comp - 要添加的组件

constraints - 确定如何将组件添加到布局的对象

抛出：

IllegalArgumentException - 如果 constraints 不是 GridBagConstraint

removeLayoutComponent

```
public void removeLayoutComponent(Component comp)
```

从此布局移除指定组件。

大多数应用程序不直接调用此方法。

指定者：

接口 LayoutManager 中的 removeLayoutComponent

参数：

comp - 要移除的组件。

另请参见：

Container.remove(java.awt.Component),Container.removeAll()

preferredLayoutSize

```
public Dimension preferredLayoutSize(Container parent)
```

使用此网络包布局确定 parent 容器的首选大小。

大多数应用程序不直接调用此方法。

指定者：

接口 LayoutManager 中的 preferredLayoutSize

参数：

parent - 在其中进行布局的容器

返回：

parent 容器的首选大小

另请参见：

Container.getPreferredSize()

minimumLayoutSize

public Dimension minimumLayoutSize(Container parent)

使用此网格包布局确定 parent 容器的最小大小。

大多数应用程序不直接调用此方法。

指定者：

接口 LayoutManager 中的 minimumLayoutSize

参数：

parent - 在其中进行布局的容器

返回：

parent 容器的最小大小

另请参见：

Container.doLayout()

maximumLayoutSize

public Dimension maximumLayoutSize(Container target)

在给出指定目标容器中的组件的前提下，返回此布局的最大维数

指定者：

接口 LayoutManager2 中的 maximumLayoutSize

参数：

target - 需要布置的容器

返回：

此布局的最大维数

另请参见：

Container, minimumLayoutSize(Container), preferredLayoutSize(Container)

getLayoutAlignmentX

public float getLayoutAlignmentX(Container parent)

返回沿 X 轴的对齐方式。它指定该组件相对于其他组件如何对齐。该值应该是一个介于 0 和 1 之间的数，其中 0 表示沿原点对齐，1 表示按距原点最远的点对齐，0.5 表示居中对齐等。

指定者：

接口 LayoutManager2 中的 getLayoutAlignmentX

返回：

指示居中对齐的值 0.5f

---

getLayoutAlignmentY

```
public float getLayoutAlignmentY(Container parent)
```

返回沿 y 轴的对齐方式。它指定该组件相对于其他组件如何对齐。该值应该是一个介于 0 和 1 之间的数，其中 0 表示沿原点对齐，1 表示按距原点最远的点对齐，0.5 表示居中对齐等。

指定者：

接口 LayoutManager2 中的 getLayoutAlignmentY

返回：

指示居中对齐的值 0.5f

---

invalidateLayout

```
public void invalidateLayout(Container target)
```

使布局失效，指示如果布局管理器缓存了信息，则应该将其丢弃。

指定者：

接口 LayoutManager2 中的 invalidateLayout

---

layoutContainer

```
public void layoutContainer(Container parent)
```

使用此网格包布局布置指定容器。为了满足此 GridBagLayout 对象的约束条件，此方法会重塑指定容器中的组件。

大多数应用程序不直接调用此方法。

指定者：
接口 LayoutManager 中的 layoutContainer
参数：
parent - 要在其中进行布局的容器
另请参见：
Container,Container.doLayout()

toString

```
public String toString()
```

返回此网格包布局的值的字符串表示形式。
覆盖：
类 Object 中的 toString
返回：
此网格包布局的字符串表示形式。

getLayoutInfo

```
protected GridBagLayoutInfo getLayoutInfo(Container parent,
                                          int sizeflag)
```

为当前受管子级的集合填充 GridBagLayoutInfo 的实例。这需要通过子级集合传递三次：

1.计算布局网格的维数。
2.确定组件占用的单元。
3.在行/列中分布权重和最小大小。

第一次遇到子级时，此操作还会将所有子级的最小大小缓存起来（因此后续循环不必再次请求）。

此方法仅供 GridBagLayout 内部使用。

参数：

parent - 布局容器
sizeflag - PREFERREDSIZE 或 MINSIZE
返回：
子级集合的 GridBagLayoutInfo
从以下版本开始：
1.4

GetLayoutInfo

```
protected GridBagLayoutInfo GetLayoutInfo(Container parent,
                                          int sizeflag)
```

此方法已过时，仅为提供向后兼容性；新代码应该调用 getLayoutInfo 来代替。此方法与 getLayoutInfo 相同；请参阅 getLayoutInfo 以获取参数及返回值的详细信息。

adjustForGravity

```
protected void adjustForGravity(GridBagConstraints constraints,
                                Rectangle r)
```

根据约束几何结构和填充将 x、y、宽度和高度四个字段调整为正确值。此方法仅供 GridBagLayout 内部使用。
参数：
constraints - 要应用的约束
r - 要调整的 Rectangle
从以下版本开始：
1.4

AdjustForGravity

```
protected void AdjustForGravity(GridBagConstraints constraints,
                                Rectangle r)
```

此方法已过时，仅为提供向后兼容性；新代码应该调用 adjustForGravity 来代

替。此方法与 adjustForGravity 相同；请参阅 adjustForGravity 以获取参数及返回值的详细信息。

getMinSize

protected Dimension getMinSize(Container parent,
GridBagLayoutInfo info)

基于 getLayoutInfo 中的信息计算其所有者的最小大小。此方法仅供 GridBagLayout 内部使用。

参数：

parent - 布局容器

info - 此父级的布局信息

返回：

包含最小大小的 Dimension 对象

从以下版本开始：

1.4

GetMinSize

protected Dimension GetMinSize(Container parent,
GridBagLayoutInfo info)

此方法已过时，仅为提供向后兼容性；新代码应该调用 getMinSize 来代替。此方法与 getMinSize 相同；请参阅 getMinSize 以获取参数及返回值的详细信息。

arrangeGrid

protected void arrangeGrid(Container parent)

布置网格。此方法仅供 GridBagLayout 内部使用。

参数：

parent - 布局容器

从以下版本开始：

1.4

ArrangeGrid

```
protected void ArrangeGrid(Container parent)
```

此方法已过时，仅为提供向后兼容性；新代码应该调用 arrangeGrid 来代替。此方法与 arrangeGrid 相同；请参阅 arrangeGrid 以获取参数及返回值的详细信息。

GridLayout 类

GridLayout 类是一个布局处理器，它以矩形网格形式对容器的组件进行布置。容器被分成大小相等的矩形，一个矩形中放置一个组件。例如，下面是一个将六个按钮布置到三行两列中的 applet：

```
import java.awt.*;
import java.applet.Applet;
public class ButtonGrid extends Applet {
    public void init() {
        setLayout(new GridLayout(3,2));
        add(new Button("1"));
        add(new Button("2"));
        add(new Button("3"));
        add(new Button("4"));
        add(new Button("5"));
        add(new Button("6"));
    }
}
```

如果容器的 ComponentOrientation 属性是水平从左到右的，则上述示例生成图 1 中所示的输出。如果容器的 ComponentOrientation 属性是水平从右到左的，则该示例生成图 2 所示的输出。

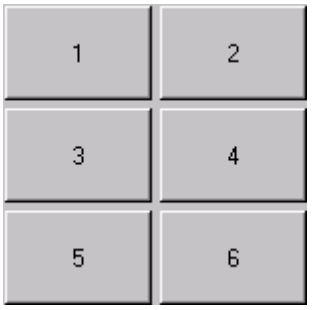


图 1：水平，从左到右

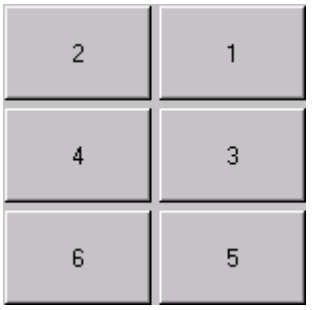


图 2：水平，从右到左

通过构造方法或 setRows 和 setColumns 方法将行数和列数都设置为非零值时，指定的列数将被忽略。列数通过指定的行数和布局中的组件总数来确定。因此，例如，如果指定了三行和两列，在布局中添加了九个组件，则它们将显示为三行三列。仅当将行数设置为零时，指定列数才对布局有效。

构造方法详细信息

GridLayout

```
public GridLayout()
```

创建具有默认值的网格布局，即每个组件占据一行一列。

从以下版本开始：

JDK1.1

GridLayout

```
public GridLayout(int rows,
                  int cols)
```

创建具有指定行数和列数的网格布局。给布局中的所有组件分配相等的大小。

rows 和 cols 中的一个可以为零（但不能两者同时为零），这表示可以将任何数目的对象置于行或列中。

参数：

rows - 该 rows 具有表示任意行数的值零。

cols - 该 cols 具有表示任意列数的值零。

GridLayout

```
public GridLayout(int rows,
                  int cols,
                  int hgap,
                  int vgap)
```

创建具有指定行数和列数的网格布局。给布局中的所有组件分配相等的大小。

此外，将水平和垂直间距设置为指定值。水平间距将置于列与列之间。将垂直间距将置于行与行之间。

rows 和 cols 中的一个可以为零（但不能两者同时为零），这表示可以将任何数目的对象置于行或列中。

所有 GridLayout 构造方法都服从这一规定。

参数：

rows - 该 rows 具有表示任意行数的值零
cols - 该 cols 具有表示任意列数的值零
hgap - 水平间距
vgap - 垂直间距

抛出：

IllegalArgumentException - 如果将 rows 和 cols 的值都设置为零

方法详细信息

getRows

```
public int getRows()
```

获取此布局中的行数。

返回：

此布局中的行数

从以下版本开始：

JDK1.1

setRows

```
public void setRows(int rows)
```

将此布局中的行数设置为指定值。

参数：

rows - 此布局中的行数

抛出：

IllegalArgumentException - 如果将 rows 和 cols 的值都设置为零

从以下版本开始：

JDK1.1

getColumns

```
public int getColumns()
```

获取此布局中的列数。

返回：

此布局中的列数

从以下版本开始：

JDK1.1

setColumns

```
public void setColumns(int cols)
```

将此布局中的列数设置为指定值。如果构造方法或 setRows 方法指定的行数为非零，则列数的设置对布局没有影响。在这种情况下，布局中显示的列数由组件的总数和指定的行数确定。

参数：

cols - 此布局中的列数

抛出：

IllegalArgumentException - 如果将 rows 和 cols 的值都设置为零
从以下版本开始：
JDK1.1

getHgap

```
public int getHgap()
```

获取组件之间的水平间距。
返回：
组件之间的水平间距
从以下版本开始：
JDK1.1

setHgap

```
public void setHgap(int hgap)
```

将组件之间的水平间距设置为指定值。
参数：
hgap - 组件之间的水平间距
从以下版本开始：
JDK1.1

getVgap

```
public int getVgap()
```

获取组件之间的垂直间距。
返回：
组件之间的垂直间距
从以下版本开始：
JDK1.1

setVgap

```
public void setVgap(int vgap)
```

将组件之间的垂直间距设置为指定值。

参数：

vgap - 组件之间的垂直间距

从以下版本开始：

JDK1.1

addLayoutComponent

```
public void addLayoutComponent(String name,
                               Component comp)
```

将具有指定名称的指定组件添加到布局。

指定者：

接口 LayoutManager 中的 addLayoutComponent

参数：

name - 组件名

comp - 要添加的组件

removeLayoutComponent

```
public void removeLayoutComponent(Component comp)
```

从布局移除指定组件。

指定者：

接口 LayoutManager 中的 removeLayoutComponent

参数：

comp - 要移除的组件

preferredLayoutSize

public Dimension preferredLayoutSize(Container parent)

使用此网格布局确定容器参数的首选大小。

网络布局的首选宽度等于容器中所有组件的最大首选宽度乘以列数，加上水平填充乘以列数减去一，再加上目标容器的左右 insets。

网络布局的首选高度等于容器中所有组件的最大首选高度乘以行数，加上垂直填充乘以行数减去一，再加上目标容器的上下 insets。

指定者：

接口 LayoutManager 中的 preferredLayoutSize

参数：

parent - 要在其中进行布局的容器

返回：

布置指定容器子组件的首选维数

另请参见：

minimumLayoutSize(java.awt.Container),
Container.getPreferredSize()

minimumLayoutSize

public Dimension minimumLayoutSize(Container parent)

使用此网络布局确定最小大小的容器参数。

网络布局的最小宽度等于容器中所有组件的最大最小宽度乘以列数，加上水平填充乘以列数减去一，再加上目标容器的左右 insets。

网络布局的最小高度等于容器中所有组件的最大最小高度乘以行数，加上垂直填充乘以行数减去一，再加上目标容器的上下 insets。

指定者：

接口 LayoutManager 中的 minimumLayoutSize

参数：

parent - 要在其中进行布局的容器

返回：

布置指定容器的子组件所需的最小维数

另请参见：

preferredLayoutSize(java.awt.Container),Container.doLayout()

layoutContainer

public void layoutContainer(Container parent)

使用此布局布置指定容器。

为了满足 GridLayout 对象的约束条件，此方法会重塑指定目标中的组件。

网络布局管理器根据布局中的行数和列数，通过将容器中的自由空间分割成相等大小的部分来确定单个组件的大小。容器的自由空间等于容器的大小减去所有 insets 和所有指定的水平和垂直间距。给网络布局中的所有组件分配相同的大小。

指定者：

接口 LayoutManager 中的 layoutContainer

参数：

parent - 要在其中进行布局的容器

另请参见：

Container,Container.doLayout()

toString

public String toString()

返回此网格布局的值的字符串表示形式。

覆盖：

类 Object 中的 toString

返回：

此网格布局的字符串表示形式

Label 类

Label 对象是一个可在容器中放置文本的组件。一个标签只显示一行只读文本。文本可由应用程序更改，但是用户不能直接对其进行编辑。

例如，代码……

```
setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
add(new Label("Hi There!"));
add(new Label("Another Label"));
```

生成以下标签：

字段详细信息

LEFT

```
public static final int LEFT
```

指示标签文本应左对齐。

另请参见：

常量字段值

CENTER

```
public static final int CENTER
```

指示标签文本应居中。

另请参见：

常量字段值

RIGHT

```
public static final int RIGHT
```

指示标签文本应右对齐。

构造方法详细信息

Label

```
public Label()
        throws HeadlessException
```

构造一个空标签。此标签的文本为空字符串 ″″。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

另请参见：

GraphicsEnvironment.isHeadless()

Label

```
public Label(String text)
        throws HeadlessException
```

使用指定的文本字符串构造一个新的标签，其文本对齐方式为左对齐。

参数：

text - 此标签显示的字符串。将接受 null 值，而不不会导致抛出 NullPointerException。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

另请参见：

GraphicsEnvironment.isHeadless()

Label

```
public Label(String text,
             int alignment)
      throws HeadlessException
```

构造一个显示指定的文本字符串的新标签，其文本对齐方式为指定的方式。可能的 alignment 值有 Label.LEFT、Label.RIGHT 和 Label.CENTER。

参数：

text - 标签显示的字符串。将接受 null 值，并且不会导致抛出 NullPointerException。

alignment - 对齐方式的值。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

方法详细信息

addNotify

```
public void addNotify()
```

创建此标签的同位体。此同位体可在不更改标签功能的情况下修改其外观。

覆盖：

类 Component 中的 addNotify

另请参见：

Component.isDisplayable(),Component.removeNotify()

---

getAlignment

```
public int getAlignment()
```

获取此标签的当前对齐方式。可能的值有 Label.LEFT、Label.RIGHT 和 Label.CENTER。

另请参见：

setAlignment(int)

---

setAlignment

public void setAlignment(int alignment)

将此标签的对齐方式设置为指定的方式。可能的值有 Label.LEFT、Label.RIGHT 和 Label.CENTER。

参数：

alignment - 要设置的对齐方式。

抛出：

IllegalArgumentException - 如果 alignment 赋值不正确。

另请参见：

getAlignment()

getText

public String getText()

获取此标签的文本。

返回：

此标签的文本，如果此文本已设置为 null，则返回 null。

另请参见：

setText(java.lang.String)

setText

public void setText(String text)

将此标签的文本设置为指定的文本。

参数：

text - 此标签显示的文本。如果 text 为 null，则将其作为一个空字符串 "" 显示。

另请参见：

getText()

paramString

protected String paramString()

返回一个表示此 Label 状态的字符串。此方法仅在进行调试的时候使用，对于这两个实现，返回的字符串的内容和格式可能有所不同。返回的字符串可能为空，但不可能为 null。

覆盖：

类 Component 中的 paramString

返回：

此标签的参数字符串

getAccessibleContext

```
public AccessibleContext getAccessibleContext()
```

获取与此标签相关的 AccessibleContext。对于标签，AccessibleContext 采用 AccessibleAWTLabel 的形式。如有必要，则会创建一个新的 AccessibleAWTLabel 实例。

指定者：

接口 Accessible 中的 getAccessibleContext

覆盖：

类 Component 中的 getAccessibleContext

返回：

一个 AccessibleAWTLabel，该 AccessibleAWTLabel 将用作此标签的 AccessibleContext。

List 类

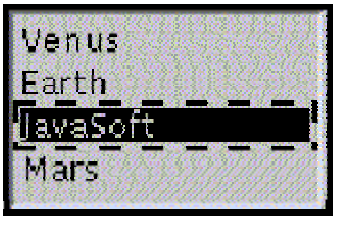
List 组件为用户提供了一个可滚动的文本项列表。可设置此 list，使其允许用户进行单项或多项选择。

例如以下代码：

```
List lst = new List(4, false);
lst.add("Mercury");
lst.add("Venus");
lst.add("Earth");
lst.add("JavaSoft");
lst.add("Mars");
```

```
        lst.add("Jupiter");
        lst.add("Saturn");
        lst.add("Uranus");
        lst.add("Neptune");
        lst.add("Pluto");
        cnt.add(lst);
```

当 cnt 为容器时，将生成以下滚动列表：



如果 List 允许进行多项选择，则单击已选中的项时，将取消选中该项。在上面的示例中，一次只能从滚动列表中选择一项，因为创建新的滚动列表时，第二个参数为 false。如果 List 不允许进行多项选择，则选择某一项会导致其他选中的项取消选中。

注意，本示例中显示的列表是用四个可视行创建的。创建该列表之后，不可更改可视行的数量。默认的 List 是用四行创建的，所以 lst = new List() 与 list = new List(4, false) 等效。

从 Java 1.1 开始，AWT（Abstract Window Toolkit，抽象窗口工具包）会把列表上发生的所有鼠标、键盘和焦点事件发送给 List 对象。（维护旧的 AWT 事件模型的目的是为了向后兼容，不推荐使用它。）

当用户选中或取消选中某项时，AWT 将向列表发送一个 ItemEvent 实例。当用户双击滚动列表中的某一项时，AWT 会在紧随项事件后向列表发送一个 ActionEvent 实例。当用户选中列表中的某项，按下 return 键时，AWT 也会生成一个动作事件。

如果应用程序需要基于此列表中用户选中或激活的项执行某个动作，则应该相应地实现 ItemListener 或 ActionListener，并注册新的侦听器，以便在此列表中接收事件。

对于多项选择滚动列表，使用外部动作（如单击按钮）来触发动作被认为是一种更好的用户界面。

构造方法详细信息

List

```
        public List()
```

```
        throws HeadlessException
```

创建新的滚动列表。默认情况下，有四个可视行，并且不允许进行多项选择。注意，这是 List(0, false) 的一种便捷方法。还要注意，列表中的可视行数一旦创建就不能更改。

抛出：

HeadlessException - 如果为 GraphicsEnvironment.isHeadless()，则返回 true。

另请参见：

GraphicsEnvironment.isHeadless()

List

```
public List(int rows)
        throws HeadlessException
```

创建一个用指定可视行数初始化的新滚动列表。默认情况下，不允许进行多项选择。注意，这是 List(rows, false) 的一种便捷方法。还要注意，列表中的可视行数一旦创建就不能更改。

参数：

rows - 要显示的项数。

抛出：

HeadlessException - 如果为 GraphicsEnvironment.isHeadless()，则返回 true。

从以下版本开始：

JDK1.1

另请参见：

GraphicsEnvironment.isHeadless()

List

```
public List(int rows,
            boolean multipleMode)
        throws HeadlessException
```

创建一个初始化为显示指定行数的新滚动列表。注意，如果指定了零行，则会按默认的四行创建列表。还要注意，列表中的可视行数一旦创建就不能更改。如果 multipleMode 的值为 true，则用户可从列表中选择多项。如果为 false，则一次只能选择一项。

参数：

rows - 要显示的项数。

multipleMode - 如果为 true，则允许进行多项选择；否则，一次只能选择一项。

抛出：

HeadlessException - 如果为 GraphicsEnvironment.isHeadless()，则返回 true。

方法详细信息

addNotify

```
public void addNotify()
```

创建列表的同位体。同位体允许我们在不更改列表功能的情况下修改其外观。

覆盖：

类 Component 中的 addNotify

另请参见：

Component.isDisplayable(),Component.removeNotify()

---

removeNotify

```
public void removeNotify()
```

移除此列表的同位体。同位体允许我们在不更改列表功能的情况下修改其外观。

覆盖：

类 Component 中的 removeNotify

另请参见：

Component.isDisplayable(),Component.addNotify()

---

getItemCount

```
public int getItemCount()
```

获取列表中的项数。

返回：

列表中的项数

从以下版本开始：
JDK1.1
另请参见：
getItem(int)

countItems

```
@Deprecated
    public int countItems()
```

已过时。 *从 JDK version 1.1 开始，由 getItemCount() 取代。*

getItem

```
    public String getItem(int index)
```

获取与指定索引关联的项。
参数：
index - 项的位置
返回：
一个与指定索引关联的项
另请参见：
getItemCount()

getItems

```
    public String[] getItems()
```

获取列表中的项。
返回：
一个包含列表中项的字符串数组
从以下版本开始：
JDK1.1
另请参见：
select(int),deselect(int),isIndexSelected(int)

add

```
public void add(String item)
```

向滚动列表的末尾添加指定的项。

参数：

item - 要添加的项

从以下版本开始：

JDK1.1

addItem

```
@Deprecated
public void addItem(String item)
```

已过时。 *由 add(String) 取代。*

add

```
public void add(String item,
                int index)
```

向滚动列表中索引指示的位置添加指定的项。索引是从零开始的。如果索引值小于零，或者索引值大于或等于列表中的项数，则将该项添加到列表的末尾。

参数：

item - 要添加的项；如果此参数为 null，则该项被视为空字符串 ""

index - 添加项的位置

从以下版本开始：

JDK1.1

addItem

```
@Deprecated
    public void addItem(String item,
                        int index)
```

已过时。 *由 add(String, int) 取代。*

replaceItem

```
    public void replaceItem(String newValue,
                            int index)
```

使用新字符串替换滚动列表中指定索引处的项。

参数：

newValue - 一个替换现有项的新字符串

index - 要替换项的位置

抛出：

ArrayIndexOutOfBoundsException - 如果 index 超出范围。

removeAll

```
    public void removeAll()
```

从此列表中移除所有项。

从以下版本开始：

JDK1.1

另请参见：

remove(java.lang.String),delItems(int, int)

clear

```
@Deprecated
    public void clear()
```

已过时。 *从 JDK version 1.1 开始，由 removeAll() 取代。*

remove

public void remove(String item)

从列表中移除项的第一次出现。如果选中了指定的项，并且该项是列表中唯一选中的项，则列表将被设置为无选择。

参数：

item - 从列表中移除的项

抛出：

IllegalArgumentException - 如果列表中不存在该项

从以下版本开始：

JDK1.1

---

remove

public void remove(int position)

从此滚动列表中移除指定位置处的项。如果选中了指定位置的项，并且该项是列表中惟一选中的项，则列表将被设置为无选择。

参数：

position - 要删除项的索引

抛出：

ArrayIndexOutOfBoundsException - 如果 position 小于零或大于 getItemCount()-1

从以下版本开始：

JDK1.1

另请参见：

add(String, int)

---

delItem

@Deprecated

public void delItem(int position)

已过时。 *由 remove(String) 和 remove(int) 取代。*

---

getSelectedIndex

public int getSelectedIndex()

获取列表中选中项的索引。

返回：

选中项的索引；如果没有选中的项，或者选中了多项，则返回 -1。

另请参见：

select(int),deselect(int),isIndexSelected(int)

getSelectedIndexes

public int[] getSelectedIndexes()

获取列表中选中的索引。

返回：

此滚动列表中选中索引的一个数组；如果没有选中的项，则返回一个零长度的数组。

另请参见：

select(int),deselect(int),isIndexSelected(int)

getSelectedItem

public String getSelectedItem()

获取此滚动列表中选中的项。

返回：

列表中选中的项；如果没有选中的项，或者选中了多项，则返回 null。

另请参见：

select(int),deselect(int),isIndexSelected(int)

getSelectedItems

public String[] getSelectedItems()

获取此滚动列表中选中的项。

返回：
此滚动列表中选中项的数组；如果没有选中的项，则返回一个零长度的数组。
另请参见：
select(int),deselect(int),isIndexSelected(int)

getSelectedObjects

```
public Object[] getSelectedObjects()
```

获取对象数组中此滚动列表的选中项。
指定者：
接口 ItemSelectable 中的 getSelectedObjects
返回：
表示此滚动列表中选中项的 Object 数组；如果没有选中的项，则会返回一个零长度的数组。
另请参见：
getSelectedItems(),ItemSelectable

select

```
public void select(int index)
```

选择滚动列表中指定索引处的项。

注意，超出参数范围是无效的，并且将导致未指定的行为。

注意，此方法应主要用于初始选择此组件中的项。以编程方式调用此方法*不* 会触发 ItemEvent。触发 ItemEvent 的唯一方式是通过用户交互。

参数：
index - 要选择的项位置
另请参见：
getSelectedItem(),deselect(int),isIndexSelected(int)

deselect

public void deselect(int index)

取消选择指定索引处的项。

注意，超出参数范围是无效的，并且将导致未指定的行为。

如果指定索引处的项未选中，则忽略操作。

参数：

index - 要取消选择项的位置

另请参见：

select(int),getSelectedItem(),isIndexSelected(int)

isIndexSelected

public boolean isIndexSelected(int index)

确定是否已选中此滚动列表中的指定项。

参数：

index - 要检查的项

返回：

如果已选中指定的项，则返回 true；否则返回 false

从以下版本开始：

JDK1.1

另请参见：

select(int),deselect(int)

isSelected

@Deprecated

public boolean isSelected(int index)

已过时。 *从 JDK version 1.1 开始，由 isIndexSelected(int) 取代。*

getRows

public int getRows()

获取此列表中的可视行数。注意，一旦完成 List 创建，行数将永远不变。

返回：

此滚动列表的可视行数

isMultipleMode

public boolean isMultipleMode()

确定此列表是否允许进行多项选择。

返回：

如果此列表允许进行多项选择，则返回 true；否则返回 false

从以下版本开始：

JDK1.1

另请参见：

setMultipleMode(boolean)

allowsMultipleSelections

@Deprecated

public boolean allowsMultipleSelections()

已过时。 *从 JDK version 1.1 开始，由 isMultipleMode() 取代。*

setMultipleMode

public void setMultipleMode(boolean b)

设置确定此列表是否允许进行多项选择的标志。当选择模式从多项选择更改为单项选择时，选中的项将会发生以下变化：如果选中的项具有位置光标，则只有该项仍保持选中。如果无选中项具有位置光标，则所有项都将取消选中。

参数：

b - 如果为 true，则允许进行多项选择；否则，只能一次从列表中选择一项。

从以下版本开始：

JDK1.1

另请参见：
isMultipleMode()

setMultipleSelections

```
@Deprecated
    public void setMultipleSelections(boolean b)
```

已过时。 *从 JDK version 1.1 开始，由 setMultipleMode(boolean) 取代。*

getVisibleIndex

```
    public int getVisibleIndex()
```

获取上次由 makeVisible 方法使其可视的项的索引。
返回：
上次变得可视的项的索引
另请参见：
makeVisible(int)

makeVisible

```
    public void makeVisible(int index)
```

使指定索引处的项可视。
参数：
index - 项的位置
另请参见：
getVisibleIndex()

getPreferredSize

```
    public Dimension getPreferredSize(int rows)
```

获取具有指定行数的列表的首选维数。

参数：

rows - 列表的行数

返回：

给出此滚动列表必须可视的指定行数，返回用于显示滚动列表的首选维数

从以下版本开始：

JDK1.1

另请参见：

Component.getPreferredSize()

---

preferredSize

```
@Deprecated
    public Dimension preferredSize(int rows)
```

已过时。 *从 JDK version 1.1 开始，由 getPreferredSize(int) 取代。*

---

getPreferredSize

```
    public Dimension getPreferredSize()
```

获取此滚动列表的首选大小。

覆盖：

类 Component 中的 getPreferredSize

返回：

用于显示此滚动列表的首选维数

从以下版本开始：

JDK1.1

另请参见：

Component.getPreferredSize()

---

preferredSize

```
@Deprecated
```

public Dimension preferredSize()

已过时。 *从 JDK version 1.1 开始，由 getPreferredSize() 取代。*

覆盖：

类 Component 中的 preferredSize

getMinimumSize

public Dimension getMinimumSize(int rows)

获取具有指定行数的列表的最少维数。

参数：

rows - 列表的行数

返回：

给出此滚动列表必须可视的指定行数，返回用于显示滚动列表的最少维数

从以下版本开始：

JDK1.1

另请参见：

Component.getMinimumSize()

minimumSize

@Deprecated

public Dimension minimumSize(int rows)

已过时。 *从 JDK version 1.1 开始，由 getMinimumSize(int) 取代。*

getMinimumSize

public Dimension getMinimumSize()

确定此滚动列表的最小大小。

覆盖：

类 Component 中的 getMinimumSize

返回：

显示此滚动列表所需的最少维数

从以下版本开始：
JDK1.1
另请参见：
Component.getMinimumSize()

minimumSize

```
@Deprecated
    public Dimension minimumSize()
```

已过时。 *从 JDK version 1.1 开始，由 getMinimumSize() 取代。*
覆盖：
类 Component 中的 minimumSize

addItemListener

```
    public void addItemListener(ItemListener l)
```

添加指定的项侦听器以接收此列表的项事件。发送项事件以响应用户输入，但不响应对 select 或 deselect 的调用。如果侦听器 l 为 null，则不会抛出异常，并且不执行动作。

有关 AWT 的线程模型的详细信息，请参阅 AWT 线程问题。

指定者：
接口 ItemSelectable 中的 addItemListener
参数：
l - 项侦听器
从以下版本开始：
JDK1.1
另请参见：
removeItemListener(java.awt.event.ItemListener), getItemListeners(), select(int), deselect(int), ItemEvent, ItemListener

removeItemListener

public void removeItemListener(ItemListener l)

移除指定的项侦听器，以便不再从此列表接收项事件。如果侦听器 l 为 null，则不会抛出异常，并且不执行动作。

有关 AWT 的线程模型的详细信息，请参阅 AWT 线程问题。

指定者：

接口 ItemSelectable 中的 removeItemListener

参数：

l - 项侦听器

从以下版本开始：

JDK1.1

另请参见：

addItemListener(java.awt.event.ItemListener), getItemListeners(), ItemEvent,ItemListener

getItemListeners

public ItemListener[] getItemListeners()

返回已在此列表上注册的所有项侦听器的数组。

返回：

此列表的所有 ItemListener，如果当前没有已注册的项侦听器，则返回一个空数组。

从以下版本开始：

1.4

另请参见：

addItemListener(java.awt.event.ItemListener), removeItemListener(java.awt.event.ItemListener), ItemEvent, ItemListener

addActionListener

public void addActionListener(ActionListener l)

添加指定的动作侦听器以从此列表接收动作事件。当用户双击列表项时或者在此列表具有键盘焦点时按 Enter 键，将发生动作事件。

如果侦听器 l 为 null，则不会抛出异常，并且不执行动作。

有关 AWT 的线程模型的详细信息，请参阅 AWT 线程问题。

参数：
l - 动作侦听器
从以下版本开始：
JDK1.1
另请参见：
removeActionListener(java.awt.event.ActionListener),
getActionListeners(),ActionEvent,ActionListener

removeActionListener

public void removeActionListener(ActionListener l)

移除指定的动作侦听器，以便不再从此列表接收动作事件。当用户双击列表项时，将发生动作事件。如果侦听器 l 为 null，则不会抛出异常，并且不执行动作。

有关 AWT 的线程模型的详细信息，请参阅 AWT 线程问题。

参数：
l - 动作侦听器
从以下版本开始：
JDK1.1
另请参见：
addActionListener(java.awt.event.ActionListener),
getActionListeners(),ActionEvent,ActionListener

getActionListeners

public ActionListener[] getActionListeners()

返回已在此列表上注册的所有动作侦听器的数组。

返回：

此列表的所有 ActionListener，如果当前没有已注册的动作侦听器，则返回一个空数组。

从以下版本开始：

1.4

另请参见：

addActionListener(java.awt.event.ActionListener), removeActionListener(java.awt.event.ActionListener), ActionEvent, ActionListener

getListeners

```
public <T extends EventListener> T[] getListeners(Class<T> listenerType)
```

返回目前已在此 List 上注册为 *Foo*Listener 的所有对象的数组。*Foo*Listener 是用 add*Foo*Listener 方法注册的。

可以使用 class 字面值来指定 listenerType 参数，如 *Foo*Listener.class。例如，可以查询其项侦听器具有以下代码的 List l：

```
ItemListener[] ils = (ItemListener[])(l.getListeners(ItemListener.class));
```

如果不存在这样的侦听器，则此方法将返回一个空数组。

覆盖：

类 Component 中的 getListeners

参数：

listenerType - 请求的侦听器类型；此参数应该指定一个从 java.util.EventListener 遗传下来的接口

返回：

在此列表上注册为 *Foo*Listener 的所有对象的数组

抛出：

ClassCastException - 如果 listenerType 未指定实现 java.util.EventListener 的类或接口。

从以下版本开始：

1.3

另请参见：

getItemListeners()

processEvent

protected void processEvent(AWTEvent e)

此滚动列表的进程事件。如果事件是 ItemEvent 的一个实例，则该事件将调用 processItemEvent 方法。另外，如果事件是 ActionEvent 的一个实例，则它将调用 processActionEvent。如果事件不是一个项事件或动作事件，则它将调用超类的 processEvent。

注意，如果事件参数为 null，则行为未指定，并可能抛出异常。

覆盖：
类 Component 中的 processEvent
参数：
e - 事件
从以下版本开始：
JDK1.1
另请参见：
ActionEvent, ItemEvent, processActionEvent(java.awt.event.ActionEvent), processItemEvent(java.awt.event.ItemEvent)

processItemEvent

protected void processItemEvent(ItemEvent e)

处理发生在此列表上的项事件，方法是将这些事件指派给所有已注册的 ItemListener 对象。

除非此组件启用了项事件，才会调用此方法。当出现以下情况时，将启用项事件：

- ItemListener 对象通过 addItemListener 注册。
- 项事件通过 enableEvents 启用。

注意，如果事件参数为 null，则行为未指定，并可能抛出异常。

参数：

e - 项事件
从以下版本开始：
JDK1.1
另请参见：
ItemEvent, ItemListener, addItemListener(java.awt.event.ItemListener), Component.enableEvents(long)

processActionEvent

protected void processActionEvent(ActionEvent e)

处理发生在此列表上的动作事件，方法是将这些事件指派给所有已注册的 ActionListener 对象。

除非此组件启用了动作事件，才会调用此方法。当出现以下情况时，将启用动作事件：

- ActionListener 对象通过 addActionListener 注册。
- 动作事件通过 enableEvents 启用。

注意，如果事件参数为 null，则行为未指定，并可能导致一个异常。

参数：
e - 动作事件
从以下版本开始：
JDK1.1
另请参见：
ActionEvent, ActionListener, addActionListener(java.awt.event.ActionListener), Component.enableEvents(long)

paramString

protected String paramString()

返回表示此滚动列表状态的参数字符串。此字符串对调试很有用。

覆盖：
类 Component 中的 paramString
返回：
此滚动列表的参数字符串

delItems

```
@Deprecated
    public void delItems(int start,
                         int end)
```

已过时。 *从 JDK version 1.1 开始，后来不再公开使用。只希望作为包私有方法时保留此方法。*

getAccessibleContext

```
    public AccessibleContext getAccessibleContext()
```

获取与此 List 关联的 AccessibleContext。对于列表，AccessibleContext 采用 AccessibleAWTList 的形式。如有必要，可以创建一个新的 AccessibleAWTList 实例。
指定者：
接口 Accessible 中的 getAccessibleContext
覆盖：
类 Component 中的 getAccessibleContext
返回：
一个 AccessibleAWTList，它将充当此 List 的 AccessibleContext。

Menu 类

Menu 对象是从菜单栏部署的下拉式菜单组件。

菜单可以是任意分离式 菜单。可以打开分离式菜单，并从其父菜单栏或菜单中拖开。释放鼠标按钮之后，它仍然在屏幕上。分离菜单的机制与平台有关，因为分离式菜单的外观由其同位体确定。对于不支持分离式菜单的平台，分离属性会被忽略。

菜单中的每一项都必须属于 MenuItem 类。它可以是 MenuItem 的一个实例、子菜单（Menu 的一个实例）、或复选框（CheckboxMenuItem 的一个实例）。

构造方法详细信息

Menu

```
public Menu()
      throws HeadlessException
```

构造具有空标签的新菜单。此菜单不是分离式菜单。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

从以下版本开始：

JDK1.1

另请参见：

GraphicsEnvironment.isHeadless()

Menu

```
public Menu(String label)
      throws HeadlessException
```

构造具有指定标签的新菜单。此菜单不是分离式菜单。

参数：

label - 菜单栏或其他菜单（此菜单是其子菜单）中菜单的标签。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

另请参见：

GraphicsEnvironment.isHeadless()

Menu

```
public Menu(String label,
```

```
                    boolean tearOff)
            throws HeadlessException
```

构造具有指定标签的新菜单，指示该菜单是否可以分离。

可能并非 AWT 的所有实现都支持分离功能。如果特定的实现不支持分离式菜单，则此值会被默认忽略。

参数：

label - 菜单栏或其他菜单（此菜单是其子菜单）中菜单的标签。

tearOff - 如果为 true，则该菜单为分离式菜单。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

方法详细信息

addNotify

```
public void addNotify()
```

创建该菜单的同位体。该同位体允许我们修改该菜单的外观，而不更改其功能。

覆盖：

类 MenuItem 中的 addNotify

---

removeNotify

```
public void removeNotify()
```

移除该菜单的同位体。该同位体允许我们修改该菜单的外观，而不更改其功能。

覆盖：

类 MenuComponent 中的 removeNotify

---

isTearOff

```
public boolean isTearOff()
```

指示此菜单是否为分离式菜单。

AWT 的所有实现都不支持分离功能。如果特定的实现不支持分离式菜单，则此值会被默认忽略。

返回：

如果这是一个分离式菜单，则返回 true；否则返回 false。

getItemCount

```
public int getItemCount()
```

获取此菜单中的项数。

返回：

此菜单中的项数。

从以下版本开始：

JDK1.1

countItems

```
@Deprecated
public int countItems()
```

已过时。 *从 JDK version 1.1 开始，已由 getItemCount() 取代。*

getItem

```
public MenuItem getItem(int index)
```

获取此菜单的指定索引处的项。

参数：

index - 要返回的项的位置。

返回：

指定索引处的项。

add

```
public MenuItem add(MenuItem mi)
```

将指定的菜单项添加到此菜单。如果该菜单项为另一个菜单的一部分，则从该菜单移除它。

参数：

mi - 要添加的菜单项

返回：

已添加的菜单项

另请参见：

insert(java.lang.String, int),insert(java.awt.MenuItem, int)

add

```
public void add(String label)
```

将带有指定标签的项添加到此菜单。

参数：

label - 该项上的文本

另请参见：

insert(java.lang.String, int),insert(java.awt.MenuItem, int)

insert

```
public void insert(MenuItem menuitem,
                   int index)
```

将菜单项插入到此菜单的指定位置。

参数：

menuitem - 要插入的菜单项。

index - 菜单项应插入的位置。

抛出：

IllegalArgumentException - 如果 index 的值小于零

从以下版本开始：

JDK1.1

另请参见：
add(java.lang.String),add(java.awt.MenuItem)

insert

```
public void insert(String label,
                   int index)
```

将带有指定标签的菜单项插入到此菜单的指定位置。这是 insert(menuItem, index) 的一个便捷方法。

参数：
label - 项的文本
index - 菜单项应插入的位置

抛出：
IllegalArgumentException - 如果 index 的值小于零

从以下版本开始：
JDK1.1

另请参见：
add(java.lang.String),add(java.awt.MenuItem)

addSeparator

```
public void addSeparator()
```

将一个分隔线或连字符添加到菜单的当前位置。

另请参见：
insertSeparator(int)

insertSeparator

```
public void insertSeparator(int index)
```

在指定的位置插入分隔符。

参数：
index - 菜单分隔符应插入的位置。

抛出：
IllegalArgumentException - 如果 index 的值小于 0。
从以下版本开始：
JDK1.1
另请参见：
addSeparator()

remove

```
public void remove(int index)
```

从此菜单移除指定索引处的菜单项。
参数：
index - 要移除的项的位置。

remove

```
public void remove(MenuComponent item)
```

从此菜单移除指定的菜单项。
指定者：
接口 MenuContainer 中的 remove
参数：
item - 要从该菜单移除的项。如果 item 为 null，或不在此菜单中，则此方法不执行任何操作。

removeAll

```
public void removeAll()
```

从此菜单移除所有项。
从以下版本开始：
JDK1.0.

paramString

public String paramString()

返回表示此 Menu 状态的字符串。此方法仅用于调试目的，对于这两个实现，返回的字符串的内容和格式可能有所不同。返回的字符串可能为空，但不可能为 null。

覆盖：

类 MenuItem 中的 paramString

返回：

此菜单的参数字符串

getAccessibleContext

public AccessibleContext getAccessibleContext()

获取与此菜单关联的 AccessibleContext。对于菜单，AccessibleContext 会采用 AccessibleAWTMenu 的窗体。必要时创建一个新的 AccessibleAWTDialog 实例。

指定者：

接口 Accessible 中的 getAccessibleContext

覆盖：

类 MenuItem 中的 getAccessibleContext

返回：

用作此菜单的 AccessibleContext 的 AccessibleAWTMenu

Panel 类

Panel 是最简单的容器类。应用程序可以将其他组件放在面板提供的空间内，这些组件包括其他面板。

面板的默认布局管理器是 FlowLayout 布局管理器。

构造方法详细信息

Panel

public Panel()

使用默认的布局管理器创建新面板。所有面板的默认布局管理器都是 FlowLayout 类。

Panel

public Panel(LayoutManager layout)

创建具有指定布局管理器的新面板。

参数：

layout - 此面板的布局管理器。

方法详细信息

addNotify

public void addNotify()

创建 Panel 的同位体。此同位体允许修改面板的外观而不会更改其功能。

覆盖：

类 Container 中的 addNotify

另请参见：

Component.isDisplayable(),Container.removeNotify()

getAccessibleContext

public AccessibleContext getAccessibleContext()

获取与 Panel 相关的 AccessibleContext。对于面板，AccessibleContext 采用 AccessibleAWTPanel 的形式。如有必要，则创建一个新的 AccessibleAWTPanel 实例。

指定者：

接口 Accessible 中的 getAccessibleContext

覆盖：

类 Component 中的 getAccessibleContext

返回：

一个 AccessibleAWTPanel，它作为此 Panel 的 AccessibleContext

Point 类

表示（x, y）坐标空间中的位置的点，以整数精度指定。

字段详细信息

x

public int x

此 Point 的 X 坐标。如果未设置 X 坐标，则默认为 0。

从以下版本开始：

1.0

另请参见：

getLocation(),move(int, int)

---

y

public int y

此 Point 的 Y 坐标。如果未设置 Y 坐标，则默认为 0。

构造方法详细信息

Point

public Point()

在坐标空间的原点 (0,0) 构造并初始化一个点。

从以下版本开始：

1.1

Point

```
public Point(Point p)
```

构造并初始化一个与指定的 Point 对象具有相同位置的点。

参数：

p - 点

从以下版本开始：

1.1

Point

```
public Point(int x,
             int y)
```

在坐标空间中指定的 (x, y) 位置构造并初始化一个点。

参数：

x - 新构造的 Point 的 X 坐标

y - 新构造的 Point 的 Y 坐标

方法详细信息

getX

```
public double getX()
```

以 double 精度返回此 Point2D 的 X 坐标。

指定者：

类 Point2D 中的 getX

返回：

此 Point2D 的 X 坐标。

从以下版本开始：

1.2

getY

public double getY()

以 double 精度返回此 Point2D 的 Y 坐标。

指定者：

类 Point2D 中的 getY

返回：

此 Point2D 的 Y 坐标。

从以下版本开始：

1.2

---

getLocation

public Point getLocation()

返回此点的位置。包含此方法是出于完整性考虑，它与 Component 的 getLocation 方法相似。

返回：

此点在相同位置的一个副本

从以下版本开始：

1.1

另请参见：

Component.getLocation(), setLocation(java.awt.Point), setLocation(int, int)

---

setLocation

public void setLocation(Point p)

将点的位置设为指定位置。包含此方法是出于完整性考虑，它与 Component 的 setLocation 方法相似。

参数：

p - 点，此点的新位置

从以下版本开始：

1.1

另请参见：
Component.setLocation(java.awt.Point),getLocation()

setLocation

```
public void setLocation(int x,
                        int y)
```

将此点更改为具有指定位置。

包含此方法是出于完整性考虑，它与 Component 的 setLocation 方法相似。其行为与 move(int, int) 相同。

参数：
x - 新位置的 X 坐标
y - 新位置的 Y 坐标
从以下版本开始：
1.1
另请参见：
Component.setLocation(int, int),getLocation(),move(int, int)

setLocation

```
public void setLocation(double x,
                        double y)
```

将此点的位置设为指定的双精度坐标。双精度的值将被舍入为整数值。任何小于 Integer.MIN_VALUE 的数都将被重置为 MIN_VALUE，任何大于 Integer.MAX_VALUE 的数都将被重置为 MAX_VALUE。
指定者：
类 Point2D 中的 setLocation
参数：
x - 新位置的 X 坐标
y - 新位置的 Y 坐标
另请参见：
getLocation()

move

```
public void move(int x,
                 int y)
```

将此点移动到 (x,y) 坐标平面中的指定位置。此方法与 setLocation(int, int) 相同。

参数：

x - 新位置的 X 坐标

y - 新位置的 Y 坐标

另请参见：

Component.setLocation(int, int)

translate

```
public void translate(int dx,
                      int dy)
```

平移 (x,y) 位置的点，沿 x 轴平移 dx，沿 y 轴平移 dy，移动后得到点 (x+dx,y+dy)。

参数：

dx - 沿 X 轴平移此点的距离

dy - 沿 Y 轴平移此点的距离

equals

```
public boolean equals(Object obj)
```

确定两个点是否相等。如果 Point2D 的两个实例的 x 和 y 成员字段的值相同，则它们是相等的，其中成员字段的值表示实例在坐标空间中的位置。

覆盖：

类 Point2D 中的 equals

参数：

obj - 将与此 Point2D 的实例进行比较的对象

返回：

如果要比较的对象是 Point2D 的实例且有相同的值，则返回 true；否则返回

false。

另请参见：

Object.hashCode(),Hashtable

toString

```
public String toString()
```

返回此点的字符串表示形式及其在（x, y）坐标空间中的位置。此方法仅用于调试目的，对于各个实现，所返回字符串的内容和格式可能有所不同。返回的字符串可以为空，但不可以为 null。

覆盖：

类 Object 中的 toString

返回：

此点的字符串表示形式

Polygon 类

Polygon 类封装了坐标空间中封闭的二维区域的描述。此区域以任意条线段为边界，每条线段都是多边形的一条边。在内部，一个多边形包含一列 (x,y) 坐标对，其中每个坐标对（coordinate pair）定义多边形的一个顶点，且两个连续的坐标对是多边形一条边的端点。第一个和最后一个 (x,y) 坐标对通过一条线段相连，形成一个封闭的多边形。此 Polygon 是按奇-偶性旋绕规则来定义的。有关奇-偶性旋绕规则的定义，请参见 WIND_EVEN_ODD。此类的目标测试方法使用 Shape 类注释中描述的 insideness 定义，目标测试方法包括 contains、intersects 和 inside 方法。

字段详细信息

npoints

```
public int npoints
```

点的总数。npoints 的值表示在此 Polygon 中有效的点的数量，该值可以小于 xpoints 或 ypoints 中元素的个数。此值可以为 NULL。

从以下版本开始：

1.0

另请参见：

addPoint(int, int)

xpoints

public int[] xpoints

X 坐标的数组。此数组中元素的个数可以大于此 Polygon 中 X 坐标的个数。额外的元素允许新的点添加到此 Polygon 中，而无需重新创建此数组。npoints 的值等于此 Polygon 中有效点的个数。

从以下版本开始：

1.0

另请参见：

addPoint(int, int)

ypoints

public int[] ypoints

Y 坐标的数组。此数组中元素的个数可以大于此 Polygon 中 Y 坐标的个数。额外的元素允许新的点添加到 Polygon 中，而无需重新创建此数组。npoints 的值等于此 Polygon 中有效点的个数。

从以下版本开始：

1.0

另请参见：

addPoint(int, int)

bounds

protected Rectangle bounds

此 Polygon 的边界。此值可以为 null。

从以下版本开始：

1.0

另请参见：

getBoundingBox(),getBounds()

构造方法详细信息

Polygon

```
public Polygon()
```

创建空的多边形。

从以下版本开始：

1.0

Polygon

```
public Polygon(int[] xpoints,
               int[] ypoints,
               int npoints)
```

根据指定的参数构造并初始化新的 Polygon。

参数：

xpoints - X 坐标的数组

ypoints - Y 坐标的数组

npoints - 此 Polygon 中点的总数

抛出：

NegativeArraySizeException - 如果 npoints 为负值。

IndexOutOfBoundsException - 如果 npoints 大于 xpoints 或 ypoints 的长度。

NullPointerException - 如果 xpoints 或 ypoints 为 null。

方法详细信息

reset

public void reset()

将此 Polygon 对象重置为一个空多边形。坐标数组及其中的数据不发生改变，但点的个数被重置为零，以便将旧的顶点数据标记为无效，并且开始从头累积新的顶点数据。与旧顶点相关的所有内部缓冲的数据都将被丢弃。注意，由于重用了重置之前的坐标数组，因此在将创建一个新的空 Polygon 与重置当前多边形相比时，如果新多边形数据的顶点数远远少于重置之前的数据的顶点数，则重新创建将会提高内存的效率。

从以下版本开始：

1.4

另请参见：

invalidate()

invalidate

public void invalidate()

所有内部缓冲数据的失效或刷新都依赖于此 Polygon 的顶点坐标。此方法应该在完成对 xpoints 或 ypoints 数组中坐标的直接操作之后被调用，以避免产生与 getBounds 或 contains 这样的方法不一致的结果，后者这些方法可能从与顶点坐标相关联的更早的计算中缓冲数据。

从以下版本开始：

1.4

另请参见：

getBounds()

translate

public void translate(int deltaX,
int deltaY)

对 Polygon 的顶点进行平移，沿 x 轴移动 deltaX，沿 y 移动 deltaY。

参数：

deltaX - 沿 X 轴移动的量

deltaY - 沿 Y 轴移动的量

从以下版本开始：

1.1

addPoint

```
public void addPoint(int x,
                     int y)
```

将指定的坐标追加到此 Polygon。

如果已经执行了计算此 Polygon 的边界框的操作，例如 getBounds 或 contains，则此方法将更新边界框。

参数：

x - 指定的 X 坐标

y - 指定的 Y 坐标

从以下版本开始：

1.0

另请参见：

getBounds(),contains(java.awt.Point)

getBounds

```
public Rectangle getBounds()
```

获取此 Polygon 的边界框。边界框是最小的 Rectangle，其边平行于坐标空间的 x 轴和 y 轴，且能够完全包含 Polygon。

指定者：

接口 Shape 中的 getBounds

返回：

返回定义此 Polygon 边界的 Rectangle。

从以下版本开始：

1.1

另请参见：

Shape.getBounds2D()

getBoundingBox

```
@Deprecated
public Rectangle getBoundingBox()
```

已过时。 *从 JDK version 1.1 开始，由 getBounds() 取代。*

返回此 Polygon 的边界。

返回：

返回此 Polygon 的边界。

从以下版本开始：

1.0

---

contains

```
public boolean contains(Point p)
```

确定指定的 Point 是否位于此 Polygon 的内部。

参数：

p - 要测试的指定的 Point

返回：

如果 Polygon 包含 Point，则返回 true；否则返回 false。

从以下版本开始：

1.0

另请参见：

contains(double, double)

---

contains

```
public boolean contains(int x,
                        int y)
```

确定指定的坐标是否位于此 Polygon 的内部。

参数：

x - 要测试的指定的 X 坐标

y - 要测试的指定的 Y 坐标

返回：

如果此 Polygon 包含指定的坐标 (x,y)，则返回 true；否则返回 false。

从以下版本开始：

1.1

另请参见：
contains(double, double)

inside

```
@Deprecated
    public boolean inside(int x,
                          int y)
```

已过时。 *从 JDK version 1.1 开始, 此函数为 contains(int, int)。*
确定此 Polygon 是否包含指定的坐标。
参数：
x - 要测试的指定的 X 坐标
y - 要测试的指定的 Y 坐标
返回：
如果此 Polygon 包含指定的坐标 (x, y)，则返回 true；否则返回 false。
从以下版本开始：
1.0
另请参见：
contains(double, double)

getBounds2D

```
    public Rectangle2D getBounds2D()
```

返回一个高精度的、比 getBounds 方法更准确的 Shape 边界框。注意，不保证返回的 Rectangle2D 是包围 Shape 的最小边界框，只保证 Shape 完全位于指示的 Rectangle2D 中。此方法返回的边界框通常比 getBounds 方法返回的更紧密，而且永远不会因为溢出问题而出错，因为返回值可以是一个使用双精度值存储尺寸的 Rectangle2D 实例。
指定者：
接口 Shape 中的 getBounds2D
返回：
一个 Rectangle2D 实例，它是 Shape 的高精度边界框。
从以下版本开始：
1.2

另请参见：
Shape.getBounds()

contains

```
public boolean contains(double x,
                        double y)
```

测试指定坐标是否在 Shape 的边界内。
指定者：
接口 Shape 中的 contains
参数：
x - 要测试的指定的 X 坐标
y - 要测试的指定的 Y 坐标
返回：
如果指定坐标在 Shape 边界内，则返回 true；否则返回 false。
从以下版本开始：
1.2

contains

```
public boolean contains(Point2D p)
```

测试指定的 Point2D 是否在 Shape 的边界内。
指定者：
接口 Shape 中的 contains
参数：
p - 要测试的指定的 Point2D
返回：
如果指定的 Point2D 在 Shape 边界内，则返回 true；否则返回 false。
从以下版本开始：
1.2

intersects

```
public boolean intersects(double x,
                          double y,
                          double w,
                          double h)
```

测试 Shape 内部是否与指定矩形区域的内部相交。如果任何一个点既包含在 Shape 内，又包含在指定矩形区域内，则认为矩形区域与 Shape 相交。

在下列情况下，Shape.intersects() 方法允许 Shape 实现谨慎地返回 true：

- 矩形区域与 Shape 相交的可能性很大，但是
- 精确确定相交的计算代价太高。

这意味着对于某些 Shape，即使矩形区域没有与该 Shape 相交，此方法也可能返回 true。如果需要更精确的答案，由于 Area 类比大多数 Shape 对象更为准确地计算几何相交，因此可以使用该类。

指定者：

接口 Shape 中的 intersects

参数：

x - 指定矩形区域左上角的 X 坐标

y - 指定矩形区域左上角的 Y 坐标

w - 指定矩形区域的宽度

h - 指定矩形区域的高度

返回：

如果 Shape 的内部区域与矩形的内部区域相交，或者相交的可能性很大且执行计算的代价太高，则返回 true；否则返回 false。

从以下版本开始：

1.2

另请参见：

Area

intersects

```
public boolean intersects(Rectangle2D r)
```

测试 Shape 内部是否与指定 Rectangle2D 内部相交。在下列情况下，Shape.intersects() 方法允许 Shape 实现谨慎地返回 true：

- Rectangle2D 与 Shape 相交的可能性很大，但是
- 精确确定相交的计算代价太高。

这意味着对于某些 Shape，即使 Rectangle2D 没有与该 Shape 相交，此方法也可能返回 true。如果需要更精确的答案，由于 Area 类比大多数 Shape 对象更为准确地计算几何相交，因此可以使用该类。

指定者：

接口 Shape 中的 intersects

参数：

r - 指定的 Rectangle2D

返回：

如果 Shape 内部与指定 Rectangle2D 内部相交，或者相交的可能性很大且执行计算的代价太高，则返回 true；否则返回 false。

从以下版本开始：

1.2

另请参见：

Shape.intersects(double, double, double, double)

---

contains

```
public boolean contains(double x,
                        double y,
                        double w,
                        double h)
```

测试 Shape 内部是否完全包含指定矩形区域。矩形区域内的所有坐标都必须位于 Shape 中，才可以认为整个矩形区域包含在 Shape中。

在下列情况下，Shape.contains() 方法允许 Shape 实现谨慎地返回 false：

- intersect 方法返回 true 并且
- 计算 Shape 是否完全包含矩形区域的代价太高。

这意味着对于某些 Shape，即使 Shape 包含矩形区域，此方法也可能返回 false。如果需要更精确的答案，由于 Area 类比大多数 Shape 对象更为准确地执行几何计算，因此可以使用该类。

指定者：

接口 Shape 中的 contains

参数：

x - 指定矩形区域左上角的 X 坐标

y - 指定矩形区域左上角的 Y 坐标

w - 指定矩形区域的宽度

h - 指定矩形区域的高度

返回：

如果 Shape 内部完全包含指定矩形区域，则返回 true；否则，如果 Shape 包含矩形区域、intersects 方法返回 true 且执行包含计算代价太高，则返回 false。

从以下版本开始：

1.2

另请参见：

Area,Shape.intersects(double, double, double, double)

contains

```
public boolean contains(Rectangle2D r)
```

测试 Shape 内部是否完全包含指定的 Rectangle2D。在下列情况下，Shape.contains() 方法允许 Shape 实现谨慎地返回 false：

- intersect 方法返回 true 并且
- 计算 Shape 是否完全包含 Rectangle2D 的代价太高。

这意味着对于某些 Shape，即使 Shape 包含 Rectangle2D，此方法也可能返回 false。如果需要更精确的答案，由于 Area 类比大多数 Shape 对象更为准确地执行几何计算，因此可以使用该类。

指定者：

接口 Shape 中的 contains

参数：

r - 指定的 Rectangle2D

返回：

如果 Shape 内部完全包含 Rectangle2D，则返回 true；否则，如果 Shape 包含 Rectangle2D、intersects 方法返回 true 且执行包含计算代价太高，则返回 false。

从以下版本开始：

1.2

另请参见：

Shape.contains(double, double, double, double)

getPathIterator

public PathIterator getPathIterator(AffineTransform at)

返回迭代器对象，此对象沿此 Polygon 的边界进行迭代，并且提供对此 Polygon 轮廓的几何形状的访问。可以指定一个可选的 AffineTransform，以便对迭代中返回的坐标进行相应的转换。

指定者：

接口 Shape 中的 getPathIterator

参数：

at - 坐标在迭代中返回时，要应用于这些坐标的可选的 AffineTransform，或者需要撤消转换时为 null

返回：

PathIterator 对象，该对象提供对此 Polygon 的几何形状的访问。

从以下版本开始：

1.2

getPathIterator

public PathIterator getPathIterator(AffineTransform at,
double flatness)

返回迭代器对象，此对象沿 Shape 的边界进行迭代，并且提供了对 Shape 轮廓的几何形状的访问。迭代器只返回 SEG_MOVETO、SEG_LINETO 和 SEG_CLOSE 点类型。由于多边形是平面的，因此可以忽略 flatness 参数。可以指定可选的 AffineTransform，在这种情况下相应地转换在迭代返回的坐标。

指定者：

接口 Shape 中的 getPathIterator

参数：

at - 坐标在迭代中返回时，要应用于这些坐标的可选的 AffineTransform，或者需要撤消转换时为 null

flatness - 在使用连接端点的直线取代细分的曲线之前，给定曲线的控制点可以从共线变化的最大量。由于多平形是平面的，因此可以忽略 flatness 参数。

返回：

返回 PathIterator 对象，此对象提供对 Shape 对象的几何形状的访问。

Rectangle 类

Rectangle 指定坐标空间中的一个区域，通过坐标空间中 Rectangle 对象左上方的点(x,y)、宽度和高度可以定义这个区域。

Rectangle 对象的 width 和 height 是 public 字段。创建 Rectangle 的构造方法，以及可以修改该对象的方法，都允许将 width 和 height 设置为负值。

对于 width 或 height 正好为 0 的 Rectangle，它在相应轴上存在维数为 0 的位置，但在这种情况下，也可将其视为空的 Rectangle。对于这种 Rectangle，isEmpty() 方法将返回 ture。 如果两个维数中任一维数为 0，则测试空 Rectangle 是否包含某个点或是否与某个矩形相交的方法将始终返回 false。用点或矩形合并这种 Rectangle 的方法将在结果中包含 Rectangle 在该轴上的位置，正如调用 add(Point) 方法一样。

对于 width 或 height 为负值的 Rectangle，它在相应轴上并不存在维数为负的位置或维数。这种 Rectangle 将被视为不存在相应的轴上。对于包含计算，这种 Rectangle 也为空，测试它是否包含某个点或是否与某个矩形相交的方法将始终返回 false。用点或矩形合并这种 Rectangle 的方法将在生成结果时完全忽略 Rectangle。如果合并了两个 Rectangle 对象，且每个对象都具有一个负维数，则结果中至少有一个负维数。

对于仅影响 Rectangle 位置的方法，无论 Rectangle 是否沿任一轴具有负维数或 0 维数，都将在其位置上进行操作。

注意，用默认不带参数的构造方法构造的 Rectangle 将具有 0x0 维数，因此为空。Rectangle 仍然具有 (0,0) 的位置，且将该位置用于合并和添加操作。因此，尝试累积点集合范围的代码最初应使用具体的负 width 和负 height 来构造 Rectangle，或者应使用该集合中的第一个点构造 Rectangle。例如：

```
Rectangle bounds = new Rectangle(0, 0, -1, -1);
for (int i = 0; i < points.length; i++) {
bounds.add(points[i]);
    }
```

如果知道点数组中至少包含了一个点，那么也可以：

```
Rectangle bounds = new Rectangle(points[0]);
for (int i = 1; i < points.length; i++) {
bounds.add(points[i]);
    }
```

此类用 32 位整数存储其位置和维数。通常情况下，这些操作可能生成超过 32 位整数范围的结果。这些方法将以某种方式计算它们的结果，该方式可以避免任何针对中间结果的 32 位溢出，然后将选择最佳的表示方式，将最终结果存储回保存位置和维数的 32 位字段中。通过将真实结果修改为最接近的 32 位值，将得到的位置存储到 x 和 y 字段中。存储到 width 和 height 维数字段中的值将被作为 32 位值选择，这些值尽可能包含真实结果的最大部分。通常情况下，这意味着维数将被独立地修改为 32 位整数范围内，但以下情况除外：如果必须移动位置才能将其存储到 32 位字段对中，则将相对于位置的“最佳表示形式”对维数进行调整；如果真实结果有一个负维数，并因此沿一个或两个轴不存在，则存储的维数在这些轴中将为负数；如果真实结果有一个位置可以在 32 位整数范围内表示，但沿一个或两个轴维数为 0，则存储的维数在这些轴中将为 0。

字段详细信息

x

```
public int x
```

Rectangle 左上角的 X 坐标。

从以下版本开始：

1.0

另请参见：

setLocation(int, int),getLocation()

---

y

```
public int y
```

Rectangle 左上角的 Y 坐标。

从以下版本开始：

1.0

另请参见：

setLocation(int, int),getLocation()

---

width

public int width

Rectangle 的宽度。

从以下版本开始：

1.0

另请参见：

setSize(int, int),getSize()

height

public int height

Rectangle 的高度。

构造方法详细信息

Rectangle

public Rectangle()

构造一个新的 Rectangle，其左上角的坐标为 (0,0)，宽度和高度均为 0。

Rectangle

public Rectangle(Rectangle r)

构造一个新的 Rectangle，并将其初始化，以与指定 Rectangle 的值匹配。

参数：

r - Rectangle，要从中将初始值复制到新构造的 Rectangle

从以下版本开始：

1.1

Rectangle

```
public Rectangle(int x,
                 int y,
                 int width,
                 int height)
```

构造一个新的 Rectangle，其左上角被指定为 (x, y)，其宽度和高度由同名的参数指定。

参数：

x - 指定的 X 坐标

y - 指定的 Y 坐标

width - Rectangle 的宽度

height - Rectangle 的高度

从以下版本开始：

1.0

Rectangle

```
public Rectangle(int width,
                 int height)
```

构造一个新的 Rectangle，其左上角的坐标为 (0,0)，其宽度和高度由同名的参数指定。

参数：

width - Rectangle 的宽度

height - Rectangle 的高度

Rectangle

```
public Rectangle(Point p,
                 Dimension d)
```

构造一个新的 Rectangle，其左上角由 Point 参数指定，其宽度和高度由 Dimension 参数指定。

参数：

p - 一个 Point，它是 Rectangle 的左上角
d - 一个 Dimension，表示 Rectangle 的宽度和高度

Rectangle

public Rectangle(Point p)

构造一个新的 Rectangle，其左上角是指定的 Point，其宽度和高度均为 0。
参数：
p - 一个 Point，它是 Rectangle 左上角的顶点

Rectangle

public Rectangle(Dimension d)

构造一个新的 Rectangle，其左上角为 (0,0)，其宽度和高度由 Dimension 参数指定。
参数：
d - 一个 Dimension，用于指定宽度和高度

方法详细信息

getX

public double getX()

以 double 精度返回边界 Rectangle 的 X 坐标。
指定者：
类 RectangularShape 中的 getX
返回：
边界 Rectangle 的 X 坐标。

getY

public double getY()

以 double 精度返回边界 Rectangle 的 Y 坐标。

指定者：

类 RectangularShape 中的 getY

返回：

边界 Rectangle 的 Y 坐标。

getWidth

public double getWidth()

以 double 精度返回边界 Rectangle 的宽度。

指定者：

类 RectangularShape 中的 getWidth

返回：

边界 Rectangle 的宽度。

getHeight

public double getHeight()

以 double 精度返回边界 Rectangle 的高度。

指定者：

类 RectangularShape 中的 getHeight

返回：

边界 Rectangle 的高度。

getBounds

public Rectangle getBounds()

获取此 Rectangle 的边界 Rectangle。

包含此方法是出于完整性考虑，以对应 Component 的 getBounds 方法。

指定者：
接口 Shape 中的 getBounds
覆盖：
类 RectangularShape 中的 getBounds
返回：
一个新 Rectangle，它等同于此 Rectangle 的边界 Rectangle。
从以下版本开始：
1.1
另请参见：
Component.getBounds(), setBounds(Rectangle), setBounds(int, int, int, int)

getBounds2D

```
public Rectangle2D getBounds2D()
```

返回一个高精度的、比 getBounds 方法更准确的 Shape 边界框。注意，不保证返回的 Rectangle2D 是包围 Shape 的最小边界框，只保证 Shape 完全位于指示的 Rectangle2D 中。此方法返回的边界框通常比 getBounds 方法返回的更紧密，而且永远不会因为溢出问题而出错，因为返回值可以是一个使用双精度值存储尺寸的 Rectangle2D 实例。
指定者：
接口 Shape 中的 getBounds2D
覆盖：
类 Rectangle2D 中的 getBounds2D
返回：
一个 Rectangle2D 实例，它是 Shape 的高精度边界框。
从以下版本开始：
1.2
另请参见：
Shape.getBounds()

setBounds

```
public void setBounds(Rectangle r)
```

设置此 Rectangle 的边界 Rectangle，以匹配指定的 Rectangle。

包含此方法是出于完整性考虑，以对应 Component 的 setBounds 方法。

参数：
r - 指定的 Rectangle
从以下版本开始：
1.1
另请参见：
getBounds(),Component.setBounds(java.awt.Rectangle)

setBounds

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

将此 Rectangle 的边界 Rectangle 设置为指定的 x、y、width 和 height。

包含此方法是出于完整性考虑，以对应 Component 的 setBounds 方法。

参数：
x - 此 Rectangle 左上角的新 X 坐标
y - 此 Rectangle 左上角的新 Y 坐标
width - 此 Rectangle 的新宽度
height - 此 Rectangle 的新高度
从以下版本开始：
1.1
另请参见：
getBounds(),Component.setBounds(int, int, int, int)

setRect

```
public void setRect(double x,
                    double y,
```

```
                                double width,
                                double height)
```

将此 Rectangle 的边界设置为整数边界，它包含指定的 x、y、width 和 height。如果这些参数指定的 Rectangle 超出整数的最大范围，则结果将为与最大整数边界相交的指定 Rectangle 的最佳表示形式。

指定者：

类 Rectangle2D 中的 setRect

参数：

x - 指定矩形左上角的 X 坐标

y - 指定矩形左上角的 Y 坐标

width - 指定矩形的宽度

height - 指定矩形的新高度

reshape

```
@Deprecated
    public void reshape(int x,
                        int y,
                        int width,
                        int height)
```

已过时。 *从 JDK version 1.1 开始，由 setBounds(int, int, int, int) 取代。*

将此 Rectangle 的边界 Rectangle 设置为指定的 x、y、width 和 height。

参数：

x - 此 Rectangle 左上角的新 X 坐标

此 - Rectangle 左上角的新 Y 坐标

width - 此 Rectangle 的新宽度

height - 此 Rectangle 的新高度

getLocation

```
    public Point getLocation()
```

返回此 Rectangle 的位置。

包含此方法是出于完整性考虑，以对应 Component 的 getLocation 方法。

返回：

一个 Point，它是此 Rectangle 左上角的顶点。

从以下版本开始：

1.1

另请参见：

Component.getLocation(),setLocation(Point),setLocation(int, int)

---

setLocation

```
public void setLocation(Point p)
```

将此 Rectangle 移动到指定位置。

包含此方法是出于完整性考虑，以对应 Component 的 setLocation 方法。

参数：

p - 指定此 Rectangle 新位置的 Point

从以下版本开始：

1.1

另请参见：

Component.setLocation(java.awt.Point),getLocation()

---

setLocation

```
public void setLocation(int x,
                        int y)
```

将此 Rectangle 移动到指定位置。

包含此方法是出于完整性考虑，以对应 Component 的 setLocation 方法。

参数：

x - 新位置的 X 坐标
y - 新位置的 Y 坐标
从以下版本开始：
1.1
另请参见：
getLocation(),Component.setLocation(int, int)

move

```
@Deprecated
    public void move(int x,
                     int y)
```

已过时。 *从 JDK version 1.1 开始，由 setLocation(int, int) 取代。*
将此 Rectangle 移动到指定位置。
参数：
x - 新位置的 X 坐标
新位置的 - Y 坐标

translate

```
    public void translate(int dx,
                          int dy)
```

将此 Rectangle 沿 X 坐标轴向右，沿 Y 坐标轴向下平移指定距离。
参数：
dx - 沿 X 轴移动此 Rectangle 的距离
dy - 沿 Y 轴移动此 Rectangle 的距离
另请参见：
setLocation(int, int),setLocation(java.awt.Point)

getSize

```
    public Dimension getSize()
```

获取此 Rectangle 的大小，用返回的 Dimension 表示。

包含此方法是出于完整性考虑，以对应 Component 的 getSize 方法。

返回：
一个 Dimension，表示此 Rectangle 的大小。
从以下版本开始：
1.1
另请参见：
Component.getSize(),setSize(Dimension),setSize(int, int)

setSize

```
public void setSize(Dimension d)
```

设置此 Rectangle 的大小，以匹配指定的 Dimension。

包含此方法是出于完整性考虑，以对应 Component 的 setSize 方法。

参数：
d - Dimension 对象的新大小
从以下版本开始：
1.1
另请参见：
Component.setSize(java.awt.Dimension),getSize()

setSize

```
public void setSize(int width,
                    int height)
```

将此 Rectangle 的大小设置为指定的宽度和高度。

包含此方法是出于完整性考虑，以对应 Component 的 setSize 方法。

参数：
width - 此 Rectangle 的新宽度

height - 此 Rectangle 的新高度
从以下版本开始：
1.1
另请参见：
Component.setSize(int, int),getSize()

---

resize

```
@Deprecated
    public void resize(int width,
                       int height)
```

已过时。 *从 JDK version 1.1 开始，由 setSize(int, int) 取代。*
将此 Rectangle 的大小设置为指定的宽度和高度。
参数：
width - 此 Rectangle 的新宽度
height - 此 Rectangle 的新高度

---

contains

```
    public boolean contains(Point p)
```

检查此 Rectangle 是否包含指定的 Point。
参数：
p - 要测试的 Point
返回：
如果指定的 Point 位于此 Rectangle 中，则返回 true；否则返回 false。
从以下版本开始：
1.1

---

contains

```
    public boolean contains(int x,
                            int y)
```

检查此 Rectangle 是否包含位于指定位置 (x,y) 的点。

参数：

x - 指定的 X 坐标

y - 指定的 Y 坐标

返回：

如果点 (x,y) 位于此 Rectangle 中，则返回 true；否则返回 false。

从以下版本开始：

1.1

---

contains

```
public boolean contains(Rectangle r)
```

检查此 Rectangle 是否完全包含指定的 Rectangle。

参数：

r - 指定的 Rectangle

返回：

如果 Rectangle 完全包含在此 Rectangle 中，则返回 true；否则返回 false

从以下版本开始：

1.2

---

contains

```
public boolean contains(int X,
                        int Y,
                        int W,
                        int H)
```

检查此 Rectangle 是否完全包含位于指定位置 (X,Y) 且具有指定维数 (W,H) 的 Rectangle。

参数：

X - 指定的 X 坐标

Y - 指定的 Y 坐标

W - Rectangle 的宽度

H - Rectangle 的高度

返回：

如果由 (X, Y, W, H) 指定的 Rectangle 完全包含在此 Rectangle 中，则返回 true；否则返回 false。

从以下版本开始：

1.1

---

inside

```
@Deprecated
    public boolean inside(int X,
                          int Y)
```

已过时。 *从 JDK version 1.1 开始，由 contains(int, int) 取代。*

检查此 Rectangle 是否包含位于指定位置 (X,Y) 的点。

参数：

X - 指定的 X 坐标

Y - 指定的 Y 坐标

返回：

如果点 (X,Y) 位于此 Rectangle 中，则返回 true；否则返回 false。

---

intersects

```
    public boolean intersects(Rectangle r)
```

确定此 Rectangle 是否与指定的 Rectangle 相交。如果两个矩形的交集为非空，则它们是相交的。

参数：

r - 指定的 Rectangle

返回：

如果指定的 Rectangle 与此 Rectangle 相交，则返回 true；否则返回 false。

---

intersection

```
    public Rectangle intersection(Rectangle r)
```

计算此 Rectangle 与指定 Rectangle 的交集。返回一个新的 Rectangle，它表

示两个矩形的交集。如果两个矩形没有相交，则结果将是一个空矩形。

参数：

r - 指定的 Rectangle

返回：

同时包含在指定 Rectangle 和此 Rectangle 中的最大 Rectangle；如果这些矩形没有相交，则返回一个空矩形。

---

union

```
public Rectangle union(Rectangle r)
```

计算此 Rectangle 与指定 Rectangle 的并集。返回一个新的 Rectangle，它表示两个矩形的并集。

如果两个 Rectangle 的任何维数小于 0，则应用针对不存在矩形的规则。如果只有一个维数小于 0，那么结果将是另一个 Rectangle 的副本。如果两个维数都小于 0，则结果中将至少有一个维数小于 0。

如果得到的 Rectangle 维数太大而无法表示为 int，则结果中将沿该维数有一个为 Integer.MAX_VALUE 的维数。

参数：

r - 指定的 Rectangle

返回：

同时包含指定 Rectangle 和此 Rectangle 的最小 Rectangle。

---

add

```
public void add(int newx,
                int newy)
```

将一个由整数参数 newx,newy 指定的点添加到此 Rectangle 的边界。

如果此 Rectangle 的任何维数小于 0，则应用针对不存在矩形的规则。在这种情况下，此 Rectangle 的新边界的位置将等于指定的坐标，且宽度和高度等于 0。

在添加一个点后，调用将添加的点作为参数的 contains 方法不一定返回 true。对于 Rectangl 右边和底部边界线上的点，econtains 方法不返回 true。所以，如果添加的点落在放大的 Rectangle 右边和底部边界线上，则 contains 将针对该点返回 false。如果指定的点必须包含在新 Rectangle 中，则应添加 1x1 矩形：

```
r.add(newx, newy, 1, 1);
```

参数：

newx - 新点的 X 坐标

newy - 新点的 Y 坐标

add

```
public void add(Point pt)
```

将指定的 Point 添加到此 Rectangle 的边界。

如果此 Rectangle 的任何维数小于 0，则应用针对不存在矩形的规则。在这种情况下，此 Rectangle 新边界的位置将等于指定 Point 的坐标，且宽度和高度等于 0。

在添加一个 Point 后，调用将添加的 Point 作为参数的 contains 方法不一定返回 true。对于 Rectangl 右边和底部边界线上的点，econtains 方法不返回 true。所以，如果添加的 Point 落在放大的 Rectangle 右边和底部边界线上，则 contains 将针对该 Point 返回 false。如果指定的点必须包含在新 Rectangle 中，则应添加 1x1 矩形：

```
r.add(pt.x, pt.y, 1, 1);
```

参数：

pt - 添加到此 Rectangle 中的新 Point

add

```
public void add(Rectangle r)
```

将一个 Rectangle 添加到此 Rectangle 中。得到的 Rectangle 是两个矩形的并集。

如果任何一个 Rectangle 任何维数小于 0，则结果将是另一个 Rectangle 的维数。如果两个 Rectangle 都至少有一个维数小于 0，则结果中至少有一个维数小于 0。

如果任何一个 Rectangle 中有一个或两个维数等于 0，则沿那些轴具有 0 维数的结果将等同于通过将相应原坐标添加到沿该轴的结果矩形中所获得的结果，这类似于 add(Point) 方法的操作，但没有超过该坐标的维数。

如果得到的 Rectangle 维数太大而无法表示为 int，则结果中将沿该维数有一个为 Integer.MAX_VALUE 的维数。

参数：

r - 指定的 Rectangle

grow

```
public void grow(int h,
                 int v)
```

在水平方向和垂直方向上同时调整 Rectangle 的大小。

此方法修改 Rectangle，使它左边和右边都增加 h 个单位，顶部和底部都增加 v 个单位。

新的 Rectangle 的左上角是 (x - h, y - v)，宽度是 (width + 2h)，高度是 (height + 2v)。

如果为 h 和 v 提供的是负值，则 Rectangle 的大小也相应地缩小。grow 方法检查整数溢出或下溢，但不检查得到的 width 和 height 值是否从负值增大到非负值或从非负值减小到负值。

参数：

h - 水平扩展

v - 垂直扩展

isEmpty

```
public boolean isEmpty()
```

确定 RectangularShape 是否为空。当 RectangularShape 为空时，它不封闭任何区域。

指定者：

类 RectangularShape 中的 isEmpty

返回：

如果 RectangularShape 为空，则返回 true；否则返回 false。

从以下版本开始：

1.2

outcode

```
public int outcode(double x,
                   double y)
```

确定指定坐标相对于此 Rectangle2D 的位置。此方法计算适当掩码值的二进制或 (OR)，这些掩码值针对此 Rectangle2D 的每个边指示指定坐标是否在此 Rectangle2D 其余边缘的同一侧。

指定者：

类 Rectangle2D 中的 outcode

参数：

x - 指定的 X 坐标

y - 指定的 Y 坐标

返回：

所有适当外码的逻辑或。

从以下版本开始：

1.2

另请参见：

Rectangle2D.OUT_LEFT, Rectangle2D.OUT_TOP, Rectangle2D.OUT_RIGHT, Rectangle2D.OUT_BOTTOM

createIntersection

public Rectangle2D createIntersection(Rectangle2D r)

返回一个新的 Rectangle2D 对象，它表示此 Rectangle2D 与指定 Rectangle2D 的交集。

指定者：

类 Rectangle2D 中的 createIntersection

参数：

r - 与此 Rectangle2D 相交的 Rectangle2D

返回：

同时被指定 Rectangle2D 和此 Rectangle2D 包含的最大 Rectangle2D。

从以下版本开始：

1.2

---

createUnion

public Rectangle2D createUnion(Rectangle2D r)

返回一个新的 Rectangle2D 对象，它表示此 Rectangle2D 与指定 Rectangle2D 的并集。

指定者：

类 Rectangle2D 中的 createUnion

参数：

r - 与此 Rectangle2D 合并的 Rectangle2D

返回：

包含指定 Rectangle2D 和此 Rectangle2D 的最小 Rectangle2D。

从以下版本开始：

1.2

---

equals

public boolean equals(Object obj)

检查两个矩形是否相等。

当且仅当参数不是 null，而是一个与此 Rectangle 具有相同左上角、宽度和高度的 Rectangle 对象时，结果才为 true。

覆盖：
类 Rectangle2D 中的 equals
参数：
obj - 要与此 Rectangle 进行比较的 Object
返回：
如果对象相等，则返回 true；否则返回 false。
另请参见：
Object.hashCode(),Hashtable

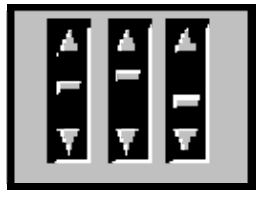
toString

```
public String toString()
```

返回表示此 Rectangle 及其值的 String。
覆盖：
类 Object 中的 toString
返回：
表示此 Rectangle 对象的坐标和大小值的 String。

Scrollbar 类

Scrollbar 类描述了一个滚动条，这是大家都很熟悉的用户界面对象。滚动条提供了一个允许用户在一定范围的值中进行选择的便捷方式。可以将以下三个垂直滚动条用作滑动块控件，以选择红、绿和蓝三种颜色的分量：



在此例中，每个滚动条都是使用类似于下面的代码创建的：

```
redSlider=new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, 255);
add(redSlider);
```

此外，滚动条还可以表示某一范围的值。例如，如果滚动条用于滚动文本，则可以使用“滑动块（bubble）”（也称为 "thumb" 或“滚动框”）的宽度来表示可视的文本数。下面是表示某一范围的滚动条：

在这个例子中，滑动块表示的值范围是可见量。此例中的水平滚动条可以用以下代码来创建：

```
ranger = new Scrollbar(Scrollbar.HORIZONTAL, 0, 60, 0, 300);
add(ranger);
```

注意，滚动条的实际最大值是 maximum 减去 visible amount。在前面的例子中，因为 maximum 是 300，visible amount 是 60，所以实际最大值是 240。滚动条轨道的范围是 0 - 300。滑动块的左侧指示了滚动条的值。

通常，用户通过使用鼠标来改变滚动条的值。例如，用户可以上下拖动滚动条的滑动块，或单击滚动条的单位增量或块增量区域。键盘动作也能映射到滚动条。按照惯例，Page Up 和 Page Down 键等同于单击滚动条的块增量和块减量区域。

当用户改变滚动条的值时，滚动条接收一个 AdjustmentEvent 实例。滚动条处理此事件，将它传递给所有已注册的侦听器。

任何希望滚动条值发生变化时被通知的对象都应该实现包 java.awt.event 中定义的 AdjustmentListener 接口。调用 addAdjustmentListener 和 removeAdjustmentListener 方法能动态地添加或删除侦听器。

AdjustmentEvent 类定义了五种调整事件，如下所示：

- 当用户拖动滚动条的滑动块时，发送 AdjustmentEvent.TRACK。
- 当用户单击水平滚动条的左箭头或垂直滚动条的上箭头，或从键盘做出等效动作时，发送 AdjustmentEvent.UNIT_INCREMENT。

- 当用户单击水平滚动条的右箭头或垂直滚动条的下箭头，或从键盘做出等效动作时，发送 AdjustmentEvent.UNIT_DECREMENT。
- 当用户单击水平滚动条滑动块左边的轨道，或垂直滚动条滑动块上边的轨道时，发送 AdjustmentEvent.BLOCK_INCREMENT。按照惯例，如果用户使用定义了 Page Up 键的键盘，则 Page Up 键是等效的。
- 当用户单击水平滚动条滑动块右边的轨道，或垂直滚动条滑动块下边的轨道时，发送 AdjustmentEvent.BLOCK_DECREMENT。按照惯例，如果用户使用定义了 Page Down 键的键盘，则 Page Down 键是等效的。

为了获得向后兼容，JDK 1.0 事件系统也受到支持，但是该平台的新版本不鼓励使用它。JDK 1.1 中介绍的五种调整事件与以前该平台版本中的有关滚动条的五种事件对应。下面列表给出了调整事件类型和它对应的 JDK 1.0 中的替换事件类型。

- AdjustmentEvent.TRACK 替换 Event.SCROLL_ABSOLUTE
- AdjustmentEvent.UNIT_INCREMENT 替换 Event.SCROLL_LINE_UP
- AdjustmentEvent.UNIT_DECREMENT 替换 Event.SCROLL_LINE_DOWN
- AdjustmentEvent.BLOCK_INCREMENT 替换 Event.SCROLL_PAGE_UP
- AdjustmentEvent.BLOCK_DECREMENT 替换 Event.SCROLL_PAGE_DOWN

注：我们建议只对值的选择使用 Scrollbar。如果想在容器中实现一个可滚动的组件，那么建议您使用 ScrollPane。如果使用 Scrollbar 来实现这一目的，那么可能会遇到绘制、键处理、大小调整和定位问题。

字段详细信息

HORIZONTAL

```
public static final int HORIZONTAL
```

指示一个水平滚动条的常量。

另请参见：

常量字段值

VERTICAL

```
public static final int VERTICAL
```

指示一个垂直滚动条的常量。

构造方法详细信息

Scrollbar

```
public Scrollbar()
            throws HeadlessException
```

构造一个新的垂直滚动条。滚动条的默认属性列在下表中：

属性	描述	默认值
方向	指示滚动条是垂直的或水平的	Scrollbar.VERTICAL
值	控制滚动条的滑动块位置的值	0
可见量	滚动条范围的可见量，通常由滚动条的滑动块的大小表示	10
最小值	滚动条的最小值	0
最大值	滚动条的最大值	100
单元增量	在按下 Line Up 或 Line Down 键时，或者单击滚动条的末端箭头时，值更改的量	1
块增量	在按下 Page Up 或 Page Down 键时，或在滑动块的两侧单击滚动条轨道时，值更改的量	10

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

另请参见：

GraphicsEnvironment.isHeadless()

Scrollbar

```
public Scrollbar(int orientation)
            throws HeadlessException
```

构造一个具有指定方向的新滚动条。

orientation 参数必须是 Scrollbar.HORIZONTAL 或 Scrollbar.VERTICAL 这两个值之一，它们分别指示滚动条是水平滚动条，还是垂直滚动条。

参数：

orientation - 指示滚动条的方向

抛出：

IllegalArgumentException - 在提供不合法的 orientation 参数值时

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

另请参见：

GraphicsEnvironment.isHeadless()

Scrollbar

```
public Scrollbar(int orientation,
                 int value,
                 int visible,
                 int minimum,
                 int maximum)
          throws HeadlessException
```

构造一个新的滚动条，它具有指定的方向、初始值、可见量、最小值和最大值。

orientation 参数必须是 Scrollbar.HORIZONTAL 或 Scrollbar.VERTICAL 这两个值之一，分别指示滚动条是水平滚动条，还是垂直滚动条。

为此构造方法提供的参数受到 setValues(int, int, int, int) 中描述的约束 (Constraints) 的限制。

参数：

orientation - 指示滚动条的方向。

value - 滚动条的初始值

visible - 滚动条的可见量，通常由滑动块的大小表示

minimum - 滚动条的最小值

maximum - 滚动条的最大值

抛出：

IllegalArgumentException - 当提供不合法的 orientation 参数值时

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

方法详细信息

addNotify

public void addNotify()

创建 Scrollbar 的同位体。此同位体允许在不更改 Scrollbar 功能的情况下修改其外观。

覆盖：

类 Component 中的 addNotify

另请参见：

Component.isDisplayable(),Component.removeNotify()

---

getOrientation

public int getOrientation()

返回此滚动条的方向。

指定者：

接口 Adjustable 中的 getOrientation

返回：

此滚动条的方向，可以是 Scrollbar.HORIZONTAL 或 Scrollbar.VERTICAL

另请参见：

setOrientation(int)

---

setOrientation

public void setOrientation(int orientation)

设置此滚动条的方向。

参数：

orientation - 此滚动条的方向，可以是 Scrollbar.HORIZONTAL 或 Scrollbar.VERTICAL

抛出：
IllegalArgumentException - 如果为 orientation 提供的值是非法值
从以下版本开始：
JDK1.1
另请参见：
getOrientation()

---

getValue

```
public int getValue()
```

获取此滚动条的当前值。
指定者：
接口 Adjustable 中的 getValue
返回：
此滚动条的当前值
另请参见：
getMinimum(),getMaximum()

---

setValue

```
public void setValue(int newValue)
```

将此滚动条的值设置为指定值。

如果指定的值小于当前 minimum 或大于当前 maximum - visibleAmount，则对 minimum 或 maximum - visibleAmount 进行适当的替换。

通常，程序应该只通过调用 setValues 更改滚动条的值。setValues 方法同时且同步地设置滚动条的最小值、最大值、可见量和值属性，因此它们是一致的。

调用此方法不会触发 AdjustmentEvent。

指定者：
接口 Adjustable 中的 setValue
参数：

newValue - 滚动条的新值

另请参见：

setValues(int, int, int, int), getValue(), getMinimum(), getMaximum()

getMinimum

```
public int getMinimum()
```

获取此滚动条的最小值。

指定者：

接口 Adjustable 中的 getMinimum

返回：

此滚动条的最小值

另请参见：

getValue(),getMaximum()

setMinimum

```
public void setMinimum(int newMinimum)
```

设置此滚动条的最小值。

在调用 setMinimum 时，最小值会发生更改，并且其他值（包括最大值、可视量和滚动条的当前值）都会随新的最小值发生一致的更改。

通常，程序应该只通过调用 setValues 更改滚动条的最小值。setValues 方法同时且同步地设置滚动条的最小值、最大值、可见量和值属性，因此它们是相互一致的。

注意，将最小值设置为 Integer.MAX_VALUE 将导致新的最小值被设置为 Integer.MAX_VALUE - 1。

指定者：

接口 Adjustable 中的 setMinimum

参数：

newMinimum - 此滚动条的新的最小值
从以下版本开始：
JDK1.1
另请参见：
setValues(int, int, int, int),setMaximum(int)

getMaximum

public int getMaximum()

获取此滚动条的最大值。
指定者：
接口 Adjustable 中的 getMaximum
返回：
此滚动条的最大值
另请参见：
getValue(),getMinimum()

setMaximum

public void setMaximum(int newMaximum)

设置此滚动条的最大值。

在调用 setMaximum 时，最大值会发生更改，并且其他值（包括最小值、可见量和滚动条的当前值）都会随新的最大值发生一致的更改。

通常，程序应该只通过调用 setValues 更改滚动条的最大值。setValues 方法同时且同步地设置滚动条的最小值、最大值、可见量和值属性，因此它们是相互一致的。

注意，将最大值设置为 Integer.MIN_VALUE 将导致新的最大值被设置为 Integer.MIN_VALUE + 1。

指定者：
接口 Adjustable 中的 setMaximum
参数：

newMaximum - 此滚动条的新的最大值
从以下版本开始：
JDK1.1
另请参见：
setValues(int, int, int, int),setMinimum(int)

getVisibleAmount

```
public int getVisibleAmount()
```

获取此滚动条的可见量。

当使用滚动条来选择某一范围的值时，可以使用可见量来表示当前可视值的范围。滚动条的滑动块（也称为 thumb 或滚动框）的大小通常给出了滚动条范围与可见量之间关系的直观表示。

在滚动条的滑动块不可移动时（例如，当它充斥滚动条轨道的整个长度时，或者当禁用滚动条时），它可能不被显示。是否显示滑动块不会影响 getVisibleAmount 返回的值。

指定者：
接口 Adjustable 中的 getVisibleAmount
返回：
此滚动条的可见量
从以下版本开始：
JDK1.1
另请参见：
setVisibleAmount(int)

getVisible

```
@Deprecated
public int getVisible()
```

已过时。 *从 JDK version 1.1 开始，由 getVisibleAmount() 取代。*

setVisibleAmount

public void setVisibleAmount(int newAmount)

设置此滚动条的可见量。

当使用滚动条来选择某一范围的值时，可以使用可见量来表示当前可视值的范围。滚动条的滑动块（也称为 thumb 或滚动框）的大小通常给出了滚动条范围与可见量之间关系的直观表示。

在滚动条的滑动块不可移动时（例如，当它充斥滚动条轨道的整个长度时，或者当禁用滚动条时），它可能不被显示。是否显示滑动块不会影响 getVisibleAmount 返回的值。

如果提供的可见量小于 one 或大于当前 maximum - minimum，则对 one 或 maximum - minimum 进行适当的替换。

通常，程序应该只通过调用 setValues 更改滚动条的值。setValues 方法同时且同步地设置滚动条的最小值、最大值、可见量和值属性，因此它们是一致的。

指定者：

接口 Adjustable 中的 setVisibleAmount

参数：

newAmount - 新的可见量

从以下版本开始：

JDK1.1

另请参见：

getVisibleAmount(),setValues(int, int, int, int)

setUnitIncrement

public void setUnitIncrement(int v)

设置此滚动条的单位增量。

单位增量是用户激活滚动条的单位增量区域时增加或减少的值，通常通过鼠标或键盘来实现，滚动条将该动作作为一个调整事件来接收。单位增量必须大于零。试图将单位增量设置为小于 1 的值将使该值被设置为 1。

指定者：
接口 Adjustable 中的 setUnitIncrement
参数：
v - 滚动条值增加或减少的量
从以下版本开始：
JDK1.1
另请参见：
getUnitIncrement()

---

setLineIncrement

```
@Deprecated
    public void setLineIncrement(int v)
```

已过时。 *从 JDK version 1.1 开始，由 setUnitIncrement(int) 取代。*

---

getUnitIncrement

```
    public int getUnitIncrement()
```

获取此滚动条的单位增量。

单位增量是用户激活滚动条的单位增量区域时增加或减少的值，通常通过鼠标或键盘来实现，滚动条将该动作作为一个调整事件来接收。单位增量必须大于零。

指定者：
接口 Adjustable 中的 getUnitIncrement
返回：
此滚动条的单位增量
从以下版本开始：
JDK1.1

另请参见：
setUnitIncrement(int)

getLineIncrement

```
@Deprecated
    public int getLineIncrement()
```

已过时。 *从 JDK version 1.1 开始，由 getUnitIncrement() 取代。*

setBlockIncrement

```
    public void setBlockIncrement(int v)
```

设置此滚动条的块增量。

块增量是用户激活滚动条的块增量区域时增加或减少的值，通常通过鼠标或键盘来实现，滚动条将该动作作为一个调整事件来接收。块增量必须大于零。试图将块增量设置为小于 1 的值将使该值被设置为 1。

指定者：
接口 Adjustable 中的 setBlockIncrement
参数：
v - 滚动条值增加或减少的量
从以下版本开始：
JDK1.1
另请参见：
getBlockIncrement()

setPageIncrement

```
@Deprecated
    public void setPageIncrement(int v)
```

已过时。 *从 JDK version 1.1 开始，由 setBlockIncrement() 取代。*

getBlockIncrement

```
public int getBlockIncrement()
```

获取此滚动条的块增量。

块增量是用户激活滚动条的块增量区域时增加或减少的值，通常通过鼠标或键盘来实现，滚动条将该动作作为一个调整事件来接收。块增量必须大于零。

指定者：
接口 Adjustable 中的 getBlockIncrement
返回：
此滚动条的块增量
从以下版本开始：
JDK1.1
另请参见：
setBlockIncrement(int)

getPageIncrement

```
@Deprecated
public int getPageIncrement()
```

已过时。 *从 JDK version 1.1 开始，由 getBlockIncrement() 取代。*

setValues

```
public void setValues(int value,
                      int visible,
                      int minimum,
                      int maximum)
```

设置此滚动条的四个属性值：value、visibleAmount、minimum 和 maximum。如果为这些属性提供的值是不一致或不正确的，则将更改它们，确保它们一致。

此方法同时且同步地设置滚动条的四个属性值，以确保它们相互一致。它强制执行了以下约束：maximum 必须大于 minimum，maximum - minimum 不能大于 Integer.MAX_VALUE，visibleAmount 必须大于零。visibleAmount 不能大于 maximum - minimum，value 不能小于 minimum，并且 value 不能大于 maximum - visibleAmount

调用此方法不会触发 AdjustmentEvent。

参数：
value - 当前窗口的位置
visible - 滚动条的可见量
minimum - 滚动条的最小值
maximum - 滚动条的最大值
另请参见：
setMinimum(int), setMaximum(int), setVisibleAmount(int), setValue(int)

getValueIsAdjusting

public boolean getValueIsAdjusting()

如果该值作为用户执行动作的结果正处于更改过程中，则返回 true。
返回：
valueIsAdjusting 属性的值
从以下版本开始：
1.4
另请参见：
setValueIsAdjusting(boolean)

setValueIsAdjusting

public void setValueIsAdjusting(boolean b)

设置 valueIsAdjusting 属性。
参数：
b - 新的调整正处于进行 (adjustment-in-progress) 状态

从以下版本开始：
1.4
另请参见：
getValueIsAdjusting()

---

addAdjustmentListener

public void addAdjustmentListener(AdjustmentListener l)

添加指定的调整侦听器，以接收发自此滚动条的 AdjustmentEvent 实例。如果 l 为 null，则不会抛出异常并且不执行任何动作。

有关 AWT 的线程模型的详细信息，请参阅 AWT 线程问题。

指定者：
接口 Adjustable 中的 addAdjustmentListener
参数：
l - 调整侦听器
从以下版本开始：
JDK1.1
另请参见：
removeAdjustmentListener(java.awt.event.AdjustmentListener), getAdjustmentListeners(),AdjustmentEvent,AdjustmentListener

---

removeAdjustmentListener

public void removeAdjustmentListener(AdjustmentListener l)

移除指定的调整侦听器，不再接收发自此滚动条的 AdjustmentEvent 实例。如果 l 为 null，则不会抛出异常并且不执行任何动作。

有关 AWT 的线程模型的详细信息，请参阅 AWT 线程问题。

指定者：
接口 Adjustable 中的 removeAdjustmentListener
参数：

l - 调整侦听器
从以下版本开始：
JDK1.1
另请参见：
addAdjustmentListener(java.awt.event.AdjustmentListener), getAdjustmentListeners(),AdjustmentEvent,AdjustmentListener

getAdjustmentListeners

```
public AdjustmentListener[] getAdjustmentListeners()
```

返回在此滚动条上所有已注册调整侦听器组成的数组。
返回：
此滚动条的所有 AdjustmentListener，如果当前没有已注册的调整侦听器，则返回一个空数组
从以下版本开始：
1.4
另请参见：
addAdjustmentListener(java.awt.event.AdjustmentListener), removeAdjustmentListener(java.awt.event.AdjustmentListener), AdjustmentEvent,AdjustmentListener

getListeners

```
public <T extends EventListener> T[] getListeners(Class<T> listenerType)
```

返回目前已在此 Scrollbar 上注册为 *Foo*Listener 的所有对象组成的数组。*Foo*Listener 是用 add*Foo*Listener 方法注册的。

可以使用 class 字面值来指定 listenerType 参数，如 *Foo*Listener.class。例如，可以使用以下代码来查询 Scrollbar c 的鼠标侦听器：

```
MouseListener[] mls = (MouseListener[])(c.getListeners(MouseListener.class));
```

如果不存在这样的侦听器，则此方法将返回一个空数组。
覆盖：

类 Component 中的 getListeners

参数：

listenerType - 所请求的侦听器类型；此参数应该指定一个从 java.util.EventListener 继承的接口

返回：

一个在此组件上作为 *Foo*Listener 注册的所有对象组成的数组，如果尚未添加这样的侦听器，则返回一个空数组

抛出：

ClassCastException - 如果 listenerType 未指定一个实现 java.util.EventListener 的类或接口

从以下版本开始：

1.3

另请参见：

Component.getComponentListeners(), Component.getFocusListeners(), Component.getHierarchyListeners(), Component.getHierarchyBoundsListeners(), Component.getKeyListeners(), Component.getMouseListeners(), Component.getMouseMotionListeners(), Component.getMouseWheelListeners(), Component.getInputMethodListeners(),#getPropertyChangeListener

processEvent

```
protected void processEvent(AWTEvent e)
```

处理在滚动条上发生的事件。如果事件是一个 AdjustmentEvent 实例，则此方法将调用 processAdjustmentEvent 方法。否则，它将调用其超类的 processEvent 方法。

注意，如果事件参数为 null，则行为是不确定的，并且可能导致异常。

覆盖：

类 Component 中的 processEvent

参数：

e - 事件

从以下版本开始：

JDK1.1

另请参见：
AdjustmentEvent,
processAdjustmentEvent(java.awt.event.AdjustmentEvent)

processAdjustmentEvent

protected void processAdjustmentEvent(AdjustmentEvent e)

处理此滚动条上发生的调整事件，方法是将其指派到任意已注册的 AdjustmentListener 对象。

如果没有在组件上启用调整事件，则不调用此方法。出现以下情况之一时启用调整事件：

- 通过 addAdjustmentListener 注册 AdjustmentListener 对象。
- 通过 enableEvents 启用调整事件。

注意，如果事件参数为 null，则行为是不确定的，并且可能导致异常。

参数：
e - 调整事件
从以下版本开始：
JDK1.1
另请参见：
AdjustmentEvent, AdjustmentListener,
addAdjustmentListener(java.awt.event.AdjustmentListener),
Component.enableEvents(long)

paramString

protected String paramString()

返回表示此 Scrollbar 当前状态的字符串表示形式。此方法仅用于调试目的，对于各个实现，所返回字符串的内容和格式可能有所不同。返回的字符串可以为空，但不可以为 null。
覆盖：
类 Component 中的 paramString

返回：
此滚动条的参数字符串

getAccessibleContext

public AccessibleContext getAccessibleContext()

获取与此 Scrollbar 相关的 AccessibleContext。对于滚动条，AccessibleContext 采用的是 AccessibleAWTScrollBar 的形式。如有必要，可以创建一个新的 AccessibleAWTScrollBar 实例。

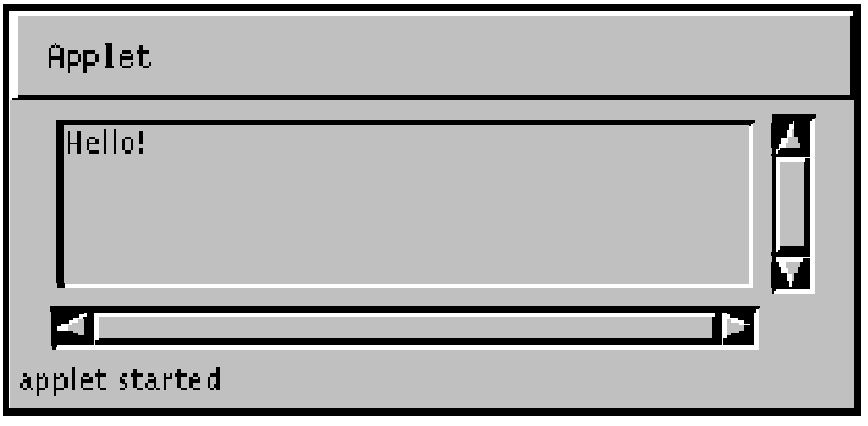
指定者：
接口 Accessible 中的 getAccessibleContext

覆盖：
类 Component 中的 getAccessibleContext

返回：
充当此 ScrollBar 的 AccessibleContext 的 AccessibleAWTScrollBar

TextArea 类

TextArea 对象是显示文本的多行区域。可以将它设置为允许编辑或只读。
下图显示了文本区的外观：



此文本区可以使用以下代码行来创建：

```
new TextArea("Hello", 5, 40);
```

字段详细信息

SCROLLBARS_BOTH

```
public static final int SCROLLBARS_BOTH
```

创建并显示垂直和水平滚动条。

从以下版本开始：

JDK1.1

另请参见：

常量字段值

SCROLLBARS_VERTICAL_ONLY

```
public static final int SCROLLBARS_VERTICAL_ONLY
```

只创建并显示垂直滚动条。

从以下版本开始：

JDK1.1

另请参见：

常量字段值

SCROLLBARS_HORIZONTAL_ONLY

```
public static final int SCROLLBARS_HORIZONTAL_ONLY
```

只创建并显示水平滚动条。

从以下版本开始：

JDK1.1

另请参见：

常量字段值

SCROLLBARS_NONE

```
public static final int SCROLLBARS_NONE
```

不为文本区创建或显示任何滚动条。

构造方法详细信息

TextArea

```
public TextArea()
            throws HeadlessException
```

构造一个将空字符串作为文本的新文本区。此文本区是在滚动条可见性等于 SCROLLBARS_BOTH 的情况下创建的，所以垂直滚动条和水平滚动条对于文本区都将是可视的。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless 返回 true

另请参见：

GraphicsEnvironment.isHeadless()

TextArea

```
public TextArea(String text)
            throws HeadlessException
```

构造具有指定文本的新文本区。文本区是在滚动条可见性等于 SCROLLBARS_BOTH 的情况下创建的，所以垂直滚动条和水平滚动条对于文本区都将是可视的。

参数：

text - 要显示的文本；如果 text 为 null，则显示空字符串 ""

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless 返回 true

另请参见：

GraphicsEnvironment.isHeadless()

TextArea

```
public TextArea(int rows,
                int columns)
         throws HeadlessException
```

构造一个新文本区，该文本区具有指定的行数和列数，并将空字符串作为文本。列是近似平均字符宽度，它与平台有关。文本区是在滚动条可见性等于 SCROLLBARS_BOTH 的情况下创建的，所以垂直滚动条和水平滚动条对于文本区都将是可视的。

参数：

rows - 行数

columns - 列数

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless 返回 true

另请参见：

GraphicsEnvironment.isHeadless()

TextArea

```
public TextArea(String text,
                int rows,
                int columns)
         throws HeadlessException
```

构造一个新文本区，该文本区具有指定的文本，以及指定的行数和列数。列是近似平均字符宽度，它与平台有关。文本区是在滚动条可见性等于 SCROLLBARS_BOTH 的情况下创建的，所以垂直滚动条和水平滚动条对于文本区都将是可视的。

参数：

text - 要显示的文本；如果 text 为 null，则显示空字符串 ""

rows - 行数

columns - 列数

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless 返回 true

另请参见：

GraphicsEnvironment.isHeadless()

TextArea

```
public TextArea(String text,
                int rows,
                int columns,
                int scrollbars)
         throws HeadlessException
```

构造一个新文本区，该文本区具有指定的文本，以及指定的行数、列数和滚动条可见性。所有 TextArea 构造方法都服从这一规定。

TextArea 类定义一些可以作为 scrollbars 参数值提供的常量：

- SCROLLBARS_BOTH,
- SCROLLBARS_VERTICAL_ONLY,
- SCROLLBARS_HORIZONTAL_ONLY,
- SCROLLBARS_NONE.

其他所有用于 scrollbars 参数的值都是无效的，并会使创建此文本区时使用的滚动条可见性等于 SCROLLBARS_BOTH 的默认值。

参数：

text - 要显示的文本；如果 text 为 null，则显示空字符串 ""

rows - 行数；如果 rows 小于 0，则将 rows 设置为 0

columns - 列数；如果 columns 小于 0，则将 columns 设置为 0

scrollbars - 确定为查看文本区创建的滚动条类型的常量

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless 返回 true

方法详细信息

addNotify

```
public void addNotify()
```

创建 TextArea 的同位体。该同位体允许修改 TextArea 的外观，而不更改其功能。

覆盖：

类 TextComponent 中的 addNotify

另请参见：

TextComponent.removeNotify()

---

insert

```
public void insert(String str,
                   int pos)
```

在此文本区的指定位置插入指定文本。

注意，传递 null 或不一致的参数是无效的，并且将导致不确定的行为。

参数：

str - 要插入的非 null 文本

pos - 插入的位置

从以下版本开始：

JDK1.1

另请参见：

TextComponent.setText(java.lang.String), replaceRange(java.lang.String, int, int), append(java.lang.String)

---

insertText

```
@Deprecated
public void insertText(String str,
                       int pos)
```

已过时。 *从 JDK version 1.1 开始，由 insert(String, int) 取代。*

---

append

```
public void append(String str)
```

将给定文本追加到文本区的当前文本。

注意，传递 null 或不一致的参数是无效的，并且将导致不确定的行为。

参数：

str - 要追加的非 null 文本

从以下版本开始：

JDK1.1

另请参见：

insert(java.lang.String, int)

---

appendText

```
@Deprecated
public void appendText(String str)
```

已过时。 *从 JDK version 1.1 开始，由 append(String) 取代。*

---

replaceRange

```
public void replaceRange(String str,
                         int start,
                         int end)
```

用指定替换文本替换指定开始位置与结束位置之间的文本。结束处的文本不会被替换。开始位置处的文本会被替换（除非开始位置与结束位置相同）。文本位置是从零开始的。插入子字符串的长度可以与所替换文本的长度不同。

注意，传递 null 或不一致的参数是无效的，并且将导致不确定的行为。

参数：

str - 用于替换的非 null 文本

start - 开始位置

end - 结束位置

从以下版本开始：

JDK1.1

另请参见：
insert(java.lang.String, int)

replaceText

```
@Deprecated
    public void replaceText(String str,
                            int start,
                            int end)
```

已过时。 *从 JDK version 1.1 开始，由 replaceRange(String, int, int) 取代。*

getRows

```
    public int getRows()
```

返回此文本区的行数。
返回：
此文本区中的行数
从以下版本开始：
JDK1
另请参见：
setRows(int),getColumns()

setRows

```
    public void setRows(int rows)
```

设置此文本区的行数。
参数：
rows - 行数
抛出：
IllegalArgumentException - 如果为 rows 提供的值小于 0
从以下版本开始：
JDK1.1

另请参见：
getRows(),setColumns(int)

getColumns

public int getColumns()

返回此文本区中的列数。
返回：
此文本区中的列数
另请参见：
setColumns(int),getRows()

setColumns

public void setColumns(int columns)

设置此文本区的列数。
参数：
columns - 列数
抛出：
IllegalArgumentException - 如果为 columns 提供的值小于 0
从以下版本开始：
JDK1.1
另请参见：
getColumns(),setRows(int)

getScrollbarVisibility

public int getScrollbarVisibility()

返回指示文本区使用何种滚动条的枚举值。

TextArea 类定义了四个整数常量，用来指定哪些滚动条是可用的。
TextArea 有一个给出滚动条上的应用方向的构造方法。

返回：
指示使用哪种滚动条的整数
从以下版本开始：
JDK1.1
另请参见：
SCROLLBARS_BOTH, SCROLLBARS_VERTICAL_ONLY, SCROLLBARS_HORIZONTAL_ONLY, SCROLLBARS_NONE, TextArea(java.lang.String, int, int, int)

---

getPreferredSize

```
public Dimension getPreferredSize(int rows,
                                  int columns)
```

确定具有指定行数和列数的文本区的首选大小。
参数：
rows - 行数
columns - 列数
返回：
显示具有指定行数和列数的文本区所需的首选尺寸
从以下版本开始：
JDK1.1
另请参见：
Component.getPreferredSize()

---

preferredSize

```
@Deprecated
public Dimension preferredSize(int rows,
                               int columns)
```

已过时。 *从 JDK version 1.1 开始，由 getPreferredSize(int, int) 取代。*

---

getPreferredSize

```
public Dimension getPreferredSize()
```

确定此文本区的首选大小。

覆盖：

类 Component 中的 getPreferredSize

返回：

此文本区所需的首选尺寸

从以下版本开始：

JDK1.1

另请参见：

Component.getPreferredSize()

preferredSize

```
@Deprecated
public Dimension preferredSize()
```

已过时。 *从 JDK version 1.1 开始，由 getPreferredSize() 取代。*

覆盖：

类 Component 中的 preferredSize

getMinimumSize

```
public Dimension getMinimumSize(int rows,
                                int columns)
```

确定具有指定行数和列数的文本区的最小大小。

参数：

rows - 行数

columns - 列数

返回：

显示具有指定行数和列数的文本区所需的最小尺寸

从以下版本开始：

JDK1.1

另请参见：

Component.getMinimumSize()

minimumSize

```
@Deprecated
    public Dimension minimumSize(int rows,
                                 int columns)
```

已过时。 *从 JDK version 1.1 开始，由 getMinimumSize(int, int) 取代。*

getMinimumSize

```
    public Dimension getMinimumSize()
```

确定此文本区的最小大小。

覆盖：

类 Component 中的 getMinimumSize

返回：

此文本区所需的首选尺寸

从以下版本开始：

JDK1.1

另请参见：

Component.getPreferredSize()

minimumSize

```
@Deprecated
    public Dimension minimumSize()
```

已过时。 *从 JDK version 1.1 开始，由 getMinimumSize() 取代。*

覆盖：

类 Component 中的 minimumSize

paramString

```
    protected String paramString()
```

返回表示此 TextArea 状态的字符串。此方法仅用于调试目的，对于各个实现，返回的字符串的内容和格式可能有所不同。返回的字符串可能为空，但不可能为 null。

覆盖：

类 TextComponent 中的 paramString

返回：

此文本区的参数字符串

---

getAccessibleContext

```
public AccessibleContext getAccessibleContext()
```

返回与此 TextArea 相关的 AccessibleContext。对于文本区，AccessibleContext 采用 AccessibleAWTTextArea 的形式。如有必要，可创建一个新的 AccessibleAWTTextArea 实例。

指定者：

接口 Accessible 中的 getAccessibleContext

覆盖：

类 TextComponent 中的 getAccessibleContext

返回：

用作此 TextArea 的 AccessibleContext 的 AccessibleAWTTextArea

TextComponent 类

TextComponent 类是所有允许编辑文本的组件的超类。

文本组件具体化文本字符串。TextComponent 类定义了一组判断此文本是否可编辑的方法。如果组件是可编辑的，则该类还定义了另一组支持文件插入符的方法。

此外，该类定义了用于维护文本当前*选择* 的方法。文本选择是组件文本的子字符串，是编辑操作的目标。它也被称为*选定文本*。

字段详细信息

textListener

protected transient TextListener textListener

方法详细信息

enableInputMethods

public void enableInputMethods(boolean enable)

启用或禁用此文本组件的输入法支持。如果启用输入法支持并且文本组件也能处理按键事件，则将传入事件提供给当前输入法；如果输入法没有使用它们，则它们只能由组件处理或被指派给它们的侦听器。默认情况下，启用还是禁用，以及如何启用或禁用此文本组建的输入法支持与实现有关。

覆盖：

类 Component 中的 enableInputMethods

参数：

enable - 为 true 表示启用；为 false 表示禁用

从以下版本开始：

1.2

另请参见：

Component.processKeyEvent(java.awt.event.KeyEvent)

getInputMethodRequests

public InputMethodRequests getInputMethodRequests()

从类 Component 复制的描述

获取输入方法请求处理程序，该处理程序支持此组件输入方法发出的请求。支持当场文本输入的组件必须重写此方法，以便返回一个 InputMethodRequests 实例。同时，还必须处理输入方法事件。

覆盖：

类 Component 中的 getInputMethodRequests

返回：

组件的输入方法请求处理程序，默认为 null

另请参见：

Component.addInputMethodListener(java.awt.event.InputMethodListener)

addNotify

public void addNotify()

通过将此 Component 连接到一个本机屏幕资源，使其成为可显示的。此方法由工具包内部调用，不应直接由程序调用。

覆盖：

类 Component 中的 addNotify

另请参见：

removeNotify()

---

removeNotify

public void removeNotify()

移除 TextComponent 的同位体。该同位体允许我们修改 TextComponent 的外观，而不更改其功能。

覆盖：

类 Component 中的 removeNotify

另请参见：

Component.isDisplayable(),Component.addNotify()

---

setText

public void setText(String t)

将此文本组件显示的文本设置为指定文本。

参数：

t - 新文本；如果此参数为 null，则将文本设置为空字符串 ""

另请参见：

getText()

---

getText

public String getText()

返回此文本组件表示的文本。默认情况下，此文本是一个空字符串。

返回：

此 TextComponent 的值

另请参见：

setText(java.lang.String)

getSelectedText

public String getSelectedText()

返回此文本组件所表示文本的选定文本。

返回：

此文本组件的选定文本

另请参见：

select(int, int)

isEditable

public boolean isEditable()

指示此文本组件是否可编辑。

返回：

如果此文本组件是可编辑的，则返回 true；否则返回 false。

从以下版本开始：

JDK1.0

另请参见：

setEditable(boolean)

setEditable

public void setEditable(boolean b)

设置判断此文本组件是否可编辑的标志。

如果将该标志设置为 true，则此文本组件变成用户可编辑的。如果将该标志设置为 false，则用户无法更改此文本组件的文本。默认情况下，不可编辑的文本组件的背景色为 SystemColor.control。通过调用 setBackground 可以重写此默认值。

参数：

b - 指示此文本组件是否是用户可编辑的标志。

从以下版本开始：

JDK1.0

另请参见：

isEditable()

---

getBackground

```
public Color getBackground()
```

获得此文本组件的背景色。默认情况下，不可编辑的文本组件的背景色为 SystemColor.control。通过调用 setBackground 可以重写此默认值。

覆盖：

类 Component 中的 getBackground

返回：

此文本组件的背景色。如果此文本组件没有背景色，则返回其父组件的背景色。

从以下版本开始：

JDK1.0

另请参见：

setBackground(Color)

---

setBackground

```
public void setBackground(Color c)
```

设置此文本组件的背景色。

覆盖：

类 Component 中的 setBackground

参数：

c - 成为此文本组件颜色的颜色。如果此参数为 null，则此文本组件将继承其父组件

的背景色。
从以下版本开始：
JDK1.0
另请参见：
getBackground()

getSelectionStart

```
public int getSelectionStart()
```

获取此文本组件中选定文本的开始位置。
返回：
选定文本的开始位置
另请参见：
setSelectionStart(int),getSelectionEnd()

setSelectionStart

```
public void setSelectionStart(int selectionStart)
```

将此文本组件的选定开始位置设置为指定位置。新的起始点限制在当前选定结束位置处或之前。并且不能将它设置为小于零，即小于组件文本的开始处。如果调用者提供的 selectionStart 值超出限度，则该方法暗中执行这些限制，操作不会失败。
参数：
selectionStart - 选定文本的开始位置
从以下版本开始：
JDK1.1
另请参见：
getSelectionStart(),setSelectionEnd(int)

getSelectionEnd

```
public int getSelectionEnd()
```

获取此文本组件中选定文本的结束位置。

返回：
选定文本的结束位置
另请参见：
setSelectionEnd(int),getSelectionStart()

setSelectionEnd

```
public void setSelectionEnd(int selectionEnd)
```

将此文本组件的选定结束位置设置为指定位置。新的结束点限制在当前选定开始位置处或之后。并且不能将它设置为超出组件文本末端。如果调用者提供的 selectionEnd 值超出限度，则该方法暗中执行这些限制，操作不会失败。
参数：
selectionEnd - 选定文本的结束位置
从以下版本开始：
JDK1.1
另请参见：
getSelectionEnd(),setSelectionStart(int)

select

```
public void select(int selectionStart,
                   int selectionEnd)
```

选择指定开始位置和结束位置之间的文本。

此方法设置选定文本的开始位置和结束位置，并强行限制开始位置必须大于等于零。结束位置必须大于等于开始位置，并小于等于文本组件的文本长度。字符位置的索引从零开始。选择文本的长度是 endPosition - startPosition，因此没有选定 endPosition 处的字符。如果选定文本的开始位置和结束位置是相等的，则取消对所有文本的选择。

如果调用者提供不一致或超出限度的值，则该方法暗中执行这些限制，操作不会失败。具体来说，如果开始位置或结束位置大于文本长度，则将它重置为等于文本长度。如果开始位置小于零，则将它重置为零，如果结束位置小于开始位置，则将它重置为开始位置。

参数：

selectionStart - 要选定的首字符（char 值）从零开始的索引

selectionEnd - 要选定的文本从零开始的结束位置；selectionEnd 处的字符（char 值）未被选定

另请参见：

setSelectionStart(int),setSelectionEnd(int),selectAll()

selectAll

```
public void selectAll()
```

选择此文本组件中的所有文本。

另请参见：

select(int, int)

setCaretPosition

```
public void setCaretPosition(int position)
```

设置文本插入符的位置。插入符的位置被限制在 0 和文本最后一个字符（包括）之间。如果传入值大于此范围，则将该值设置为最后一个字符（如果 TextComponent 不包含文本，则将该值设置为 0），并且不返回任何错误。如果传入值小于 0，则抛出 IllegalArgumentException。

参数：

position - 文本插入符的位置

抛出：

IllegalArgumentException - 如果 position 小于零

从以下版本开始：

JDK1.1

getCaretPosition

```
public int getCaretPosition()
```

返回文本插入符的位置。插入符的位置被限制在 0 和文本最后一个字符（包括）之

间。如果没有设置文本或插入符，则默认插入符的位置为 0。

返回：

文本插入符的位置

从以下版本开始：

JDK1.1

另请参见：

setCaretPosition(int)

addTextListener

```
public void addTextListener(TextListener l)
```

添加指定的文本事件侦听器，以接收此文本组件发出的文本事件。如果 l 为 null，则不抛出任何异常，也不执行任何操作。

有关 AWT 线程模式的详细信息，请参考 AWT Threading Issues。

参数：

l - 文本事件侦听器

另请参见：

removeTextListener(java.awt.event.TextListener), getTextListeners(),TextListener

removeTextListener

```
public void removeTextListener(TextListener l)
```

移除指定的文本事件侦听器，不再接收此文本组件发出的文本事件。如果 l 为 null，则不抛出任何异常，也不执行任何操作。

有关 AWT 线程模式的详细信息，请参考 AWT Threading Issues。

参数：

l - 文本侦听器

从以下版本开始：

JDK1.1

另请参见：

addTextListener(java.awt.event.TextListener), getTextListeners(), TextListener

getTextListeners

```
public TextListener[] getTextListeners()
```

返回在此文本组件上所有已注册文本侦听器的数组。

返回：

此文本组件的所有 TextListener；如果当前没有已注册的文本侦听器，则返回一个空数组

从以下版本开始：

1.4

另请参见：

addTextListener(java.awt.event.TextListener), removeTextListener(java.awt.event.TextListener)

getListeners

```
public <T extends EventListener> T[] getListeners(Class<T> listenerType)
```

返回当前已在此 TextComponent 上注册为 *Foo*Listener 的所有对象的数组。*Foo*Listener 是用 add*Foo*Listener 方法注册的。

可以使用 class 字面值来指定 listenerType 参数，如 *Foo*Listener.class。例如，可以使用以下代码来查询 TextComponent t 的文本侦听器：

```
TextListener[] tls = (TextListener[])(t.getListeners(TextListener.class));
```

如果不存在这样的侦听器，则此方法返回一个空数组。

覆盖：

类 Component 中的 getListeners

参数：

listenerType - 请求的侦听器类型；此参数应该指定一个从 java.util.EventListener 继承的接口

返回：

在此文本组件上作为 *Foo*Listener 注册的所有对象的数组；如果尚未添加这样的侦听器，则返回一个空数组

抛出：

ClassCastException - 如果 listenerType 未指定一个实现 java.util.EventListener 的类或接口

从以下版本开始：

1.3

另请参见：

getTextListeners()

---

processEvent

protected void processEvent(AWTEvent e)

处理此文本组件上发生的事件。如果事件是 TextEvent，那么它将调用 processTextEvent 方法，否则它将调用其超类的 processEvent。

注意，如果事件参数为 null，则行为是不明确的，并可能导致异常。

覆盖：

类 Component 中的 processEvent

参数：

e - 事件

另请参见：

Component.processComponentEvent(java.awt.event.ComponentEvent), Component.processFocusEvent(java.awt.event.FocusEvent), Component.processKeyEvent(java.awt.event.KeyEvent), Component.processMouseEvent(java.awt.event.MouseEvent), Component.processMouseMotionEvent(java.awt.event.MouseEvent), Component.processInputMethodEvent(java.awt.event.InputMethodEvent), Component.processHierarchyEvent(java.awt.event.HierarchyEvent), Component.processMouseWheelEvent(java.awt.event.MouseWheelEvent)

---

processTextEvent

protected void processTextEvent(TextEvent e)

处理发生在此文本组件上的文本事件：将这些事件指派给所有已注册的 TextListener 对象。

注：如果没有对此组件启用文本事件，则不调用此方法。这发生在出现以下情况之一时：

- 通过 addTextListener 注册一个 TextListener 对象。
- 通过 enableEvents 启用文本事件

注意，如果事件参数为 null，则行为是不明确的，并可能导致异常。

参数：

e - 文本事件

另请参见：

Component.enableEvents(long)

paramString

protected String paramString()

返回表示此 TextComponent 状态的字符串。此方法仅用于调试目的，对于各个实现，返回字符串的内容和格式可能有所不同。返回的字符串可能为空，但不可能为 null。

覆盖：

类 Component 中的 paramString

返回：

此文本组件的参数字符串

getAccessibleContext

public AccessibleContext getAccessibleContext()

获取与此 TextComponent 关联的 AccessibleContext。对于文本组件，AccessibleContext 采用 AccessibleAWTTextComponent 的形式。如有必要，可创建一个新的 AccessibleAWTTextComponent 实例。

指定者：

接口 Accessible 中的 getAccessibleContext

覆盖：

类 Component 中的 getAccessibleContext

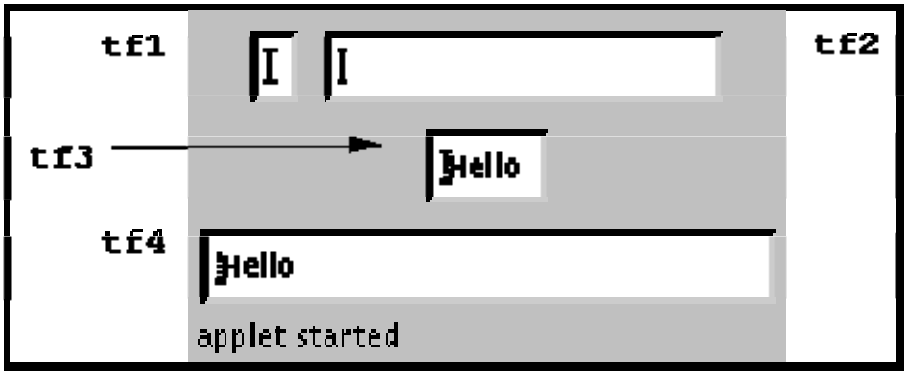
返回：

一个 AccessibleAWTTextComponent，它充当此 TextComponent 的 AccessibleContext

TextField 类

TextField 对象是允许编辑单行文本的文本组件。

例如，下图描绘了带有四个宽度各异的文本字段的窗体。其中两个文本字段显示预定义文本 "Hello"。



以下是产生四个文本字段的代码：

```
TextField tf1, tf2, tf3, tf4;
// a blank text field
tf1 = new TextField();
// blank field of 20 columns
tf2 = new TextField("", 20);
// predefined text displayed
tf3 = new TextField("Hello!");
// predefined text in 30 columns
tf4 = new TextField("Hello", 30);
```

每次用户在文本字段中键入一个键时，就有一个或更多键事件被发送到该文本字段。KeyEvent 是以下三种类型之一：keyPressed、keyReleased 或 keyTyped。键事件的属性指示

事件是哪一种类型，以及关于事件的其他信息，比如对键事件应用哪种修饰符和事件发生的时间。

键事件被传递给每一个 KeyListener 或 KeyAdapter 对象，这些对象使用组件的 addKeyListener 方法注册，以接收这类事件。（KeyAdapter 对象实现 KeyListener 接口。）

TextField 还可能触发 ActionEvent。如果对文本字段启用操作事件，则可以通过按下 Return 键触发它们。

TextField 类的 processEvent 方法检查操作事件，并将它们传递给 processActionEvent。后一种方法将该事件重定向到为接收此文本字段生成的操作事件而注册的所有 ActionListener 对象。

构造方法详细信息

TextField

```
public TextField()
            throws HeadlessException
```

构造新文本字段。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

另请参见：

GraphicsEnvironment.isHeadless()

TextField

```
public TextField(String text)
            throws HeadlessException
```

构造使用指定文本初始化的新文本字段。

参数：

text - 要显示的文本。如果 text 为 null，则显示空字符串 ""。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

另请参见：

GraphicsEnvironment.isHeadless()

TextField

```
public TextField(int columns)
                 throws HeadlessException
```

构造具有指定列数的新空文本字段。列是近似平均字符宽度，它与平台有关。

参数：

columns - 列数。如果 columns 小于 0，则将 columns 设置为 0。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

另请参见：

GraphicsEnvironment.isHeadless()

TextField

```
public TextField(String text,
                 int columns)
                 throws HeadlessException
```

构造使用要显示的指定文本初始化的新文本字段，宽度足够容纳指定列数。列是近似平均字符宽度，它与平台有关。

参数：

text - 要显示的文本。如果 text 为 null，则显示空字符串 ""。

columns - 列数。如果 columns 小于 0，则将 columns 设置为 0。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

方法详细信息

addNotify

```
public void addNotify()
```

创建 TextField 的同位体。此同位体允许修改 TextField 的外观，并且不更改其功能。

覆盖：

类 TextComponent 中的 addNotify

另请参见：
TextComponent.removeNotify()

getEchoChar

```
public char getEchoChar()
```

获取用于回显的字符。

回显字符对于不应将用户输入回显到屏幕上的文本字段有用，例如输入密码的文本字段。如果 echoChar = 0，则将用户输入不作更改地回显到屏幕。

Java 平台实现只能支持有限的、非空回显字符集。此函数返回 setEchoChar() 最初请求的回显字符。TextField 实现实际使用的回显字符可能不同。

返回：
此文本字段的回显字符。
另请参见：
echoCharIsSet(),setEchoChar(char)

setEchoChar

```
public void setEchoChar(char c)
```

设置此文本字段的回显字符。

回显字符对于不应将用户输入回显到屏幕上的文本字段很有用，例如输入密码的文本字段。设置 echoChar = 0 允许将用户输入再次回显到屏幕。

Java 平台实现只能支持有限的、非空回显字符集。试图设置不受支持的回显字符将导致使用默认回显字符。对 getEchoChar() 的后续调用将返回最初请求的回显字符。返回的回显字符有可能与 TextField 实现实际使用的相同，也有可能不同。

参数：

c - 此文本字段的回显字符。

从以下版本开始：

JDK1.1

另请参见：

echoCharIsSet(),getEchoChar()

---

setEchoCharacter

```
@Deprecated
    public void setEchoCharacter(char c)
```

已过时。 *从 JDK version 1.1 开始，由 setEchoChar(char) 取代。*

---

setText

```
    public void setText(String t)
```

将此文本组件显示的文本设置为指定文本。

覆盖：

类 TextComponent 中的 setText

参数：

t - 新文本。

另请参见：

TextComponent.getText()

---

echoCharIsSet

```
    public boolean echoCharIsSet()
```

指示此文本字段是否有一个回显字符集。

回显字符对于不应将用户输入回显到屏幕上的文本字段有用，例如输入密码的文本字段。

返回：

如果此文本字段有一个回显字符集，则返回 true；否则返回 false。

另请参见：

setEchoChar(char),getEchoChar()

getColumns

public int getColumns()

获取此文本字段中的列数。列是近似平均字符宽度，它与平台有关。

返回：

列数。

从以下版本开始：

JDK1.1

另请参见：

setColumns(int)

setColumns

public void setColumns(int columns)

设置此文本字段中的列数。列是近似平均字符宽度，它与平台有关。

参数：

columns - 列数。

抛出：

IllegalArgumentException - 如果为 columns 提供的值小于 0。

从以下版本开始：

JDK1.1

另请参见：

getColumns()

getPreferredSize

public Dimension getPreferredSize(int columns)

获取具有指定列数的文本字段的首选大小。

参数：
columns - 此文本字段中的列数。
返回：
显示此文本字段的首选尺寸。
从以下版本开始：
JDK1.1

---

preferredSize

```
@Deprecated
    public Dimension preferredSize(int columns)
```

已过时。 *从 JDK version 1.1 开始，由 getPreferredSize(int) 取代。*

---

getPreferredSize

```
    public Dimension getPreferredSize()
```

获取此文本字段的首选大小。
覆盖：
类 Component 中的 getPreferredSize
返回：
显示此文本字段的首选尺寸。
从以下版本开始：
JDK1.1
另请参见：
Component.getMinimumSize(),LayoutManager

---

preferredSize

```
@Deprecated
    public Dimension preferredSize()
```

已过时。 *从 JDK version 1.1 开始，由 getPreferredSize() 取代。*
覆盖：

类 Component 中的 preferredSize

getMinimumSize

```
public Dimension getMinimumSize(int columns)
```

获取具有指定列数的文本字段的最小尺寸。

参数：

columns - 此文本字段中的列数。

从以下版本开始：

JDK1.1

minimumSize

```
@Deprecated
public Dimension minimumSize(int columns)
```

已过时。 *从 JDK version 1.1 开始，由 getMinimumSize(int) 取代。*

getMinimumSize

```
public Dimension getMinimumSize()
```

获取此文本字段的最小尺寸。

覆盖：

类 Component 中的 getMinimumSize

返回：

显示此文本字段的最小尺寸。

从以下版本开始：

JDK1.1

另请参见：

Component.getPreferredSize(),LayoutManager

minimumSize

@Deprecated
public Dimension minimumSize()

已过时。 *从 JDK version 1.1 开始，由 getMinimumSize() 取代。*
覆盖：
类 Component 中的 minimumSize

---

addActionListener

public void addActionListener(ActionListener l)

添加指定的操作侦听器，以从此文本字段接收操作事件。如果 l 为 null，则不抛出任何异常，也不执行任何操作。

有关 AWT 线程模式的详细信息，请参考 AWT Threading Issues。

参数：
l - 操作侦听器。
从以下版本开始：
JDK1.1
另请参见：
removeActionListener(java.awt.event.ActionListener), getActionListeners(),ActionListener

---

removeActionListener

public void removeActionListener(ActionListener l)

移除指定的操作侦听器，不再从此文本字段接收操作事件。如果 l 为 null，则不抛出任何异常，也不执行任何操作。

有关 AWT 线程模式的详细信息，请参考 AWT Threading Issues。

参数：
l - 操作侦听器。
从以下版本开始：
JDK1.1
另请参见：

addActionListener(java.awt.event.ActionListener),
getActionListeners(),ActionListener

getActionListeners

```
public ActionListener[] getActionListeners()
```

返回此文本字段上已注册的所有操作侦听器的数组。

返回：

此文本字段的所有 ActionListener；如果当前没有注册任何操作侦听器，则返回一个空数组。

从以下版本开始：

1.4

另请参见：

addActionListener(java.awt.event.ActionListener),
removeActionListener(java.awt.event.ActionListener),
java.awt.event

getListeners

```
public <T extends EventListener> T[] getListeners(Class<T> listenerType)
```

返回当前已在此 TextField 上注册为 *Foo*Listener 的所有对象的数组。*Foo*Listener 是用 add*Foo*Listener 方法注册的。

可以使用 class 字面值（如 *Foo*Listener.class）来指定 listenerType 参数。例如，可以使用以下代码来查询 TextField t 的操作侦听器：

```
ActionListener[] als = (ActionListener[])(t.getListeners(ActionListener.class));
```

如果不存在这样的侦听器，则此方法将返回一个空数组。

覆盖：

类 TextComponent 中的 getListeners

参数：

listenerType - 所请求的侦听器类型；此参数应该指定一个从 java.util.EventListener 继承的接口

返回：

在此文本字段上作为 *Foo*Listener 注册的所有对象的数组；如果尚未添加这样的侦听器，则返回一个空数组

抛出：

ClassCastException - 如果 listenerType 未指定一个实现 java.util.EventListener 的类或接口

从以下版本开始：

1.3

另请参见：

getActionListeners()

processEvent

protected void processEvent(AWTEvent e)

处理此文本字段上的事件。如果事件是 ActionEvent 的一个实例，则此方法将调用 processActionEvent 方法。否则，它将调用超类的 processEvent 方法。

注意，如果事件参数为 null，则行为是不明确的，并可能导致异常。

覆盖：

类 TextComponent 中的 processEvent

参数：

e - 事件

从以下版本开始：

JDK1.1

另请参见：

ActionEvent,processActionEvent(java.awt.event.ActionEvent)

processActionEvent

protected void processActionEvent(ActionEvent e)

处理发生在此文本字段上的操作事件：将这些事件指派给所有已注册的 ActionListener 对象。

只有此组件启用了操作事件时，才调用此方法。发生以下事件之一时将启用操作事件：

- ActionListener 对象通过 addActionListener 注册。
- 操作事件通过 enableEvents 启用。

注意，如果事件参数为 null，则行为是不明确的，并可能导致异常。

参数：
e - 操作事件
从以下版本开始：
JDK1.1
另请参见：
ActionListener, addActionListener(java.awt.event.ActionListener), Component.enableEvents(long)

---

paramString

protected String paramString()

返回表示此 TextField 状态的字符串。此方法仅用于调试目的，对于各个实现，返回字符串的内容和格式可能有所不同。返回的字符串可能为空，但不可能为 null。
覆盖：
类 TextComponent 中的 paramString
返回：
此文本字段的参数字符串

---

getAccessibleContext

public AccessibleContext getAccessibleContext()

获取与此 TextField 关联的 AccessibleContext。对于文本字段，AccessibleContext 采用 AccessibleAWTTextField 的形式。如有必要，可创建一个新的 AccessibleAWTTextField 实例。
指定者：
接口 Accessible 中的 getAccessibleContext
覆盖：

类 TextComponent 中的 getAccessibleContext

返回：

一个 AccessibleAWTTextField，它充当此 TextField 的 AccessibleContext。

Toolkit 类

此类是所有 Abstract Window Toolkit 实际实现的抽象超类。Toolkit 的子类被用于将各种组件绑定到特定本机工具包实现。

许多 GUI 操作可以异步执行。这意味着如果设置某一组件的状态，随后立刻查询该状态，则返回的值可能并没有反映所请求的更改。这包括但不局限于以下操作：

滚动到指定位置。

例如，如果原始请求没有被处理，那么调用 ScrollPane.setScrollPosition 并随后调用 getScrollPosition 可能返回一个不正确的值。

将焦点从一个组件移动到另一个组件。

有关更多信息，请参阅 The Swing Tutorial 的 Timing Focus Transfers 一节。

使顶层容器可见。

对 Window、Frame 或 Dialog 调用 setVisible(true) 可能异步发生。

设置顶层容器的大小或位置。

对 Window、Frame 或 Dialog 调用 setSize、setBounds 或 setLocation 将被转发到底层窗口管理系统，并可能被忽略或修改。有关更多信息，请参阅 Window。

大多数应用程序不应直接调用该类中的任何方法。Toolkit 定义的方法是一种“胶水”，将 java.awt 包中与平台无关的类与 java.awt.peer 中的对应物连接起来。Toolkit 定义的一些方法能直接查询本机操作系统。

字段详细信息

desktopProperties

```
protected final Map<String,Object> desktopProperties
```

desktopPropsSupport

protected final PropertyChangeSupport desktopPropsSupport

构造方法详细信息

Toolkit

public Toolkit()

方法详细信息

createDesktopPeer

protected abstract java.awt.peer.DesktopPeer createDesktopPeer(Desktop target)
throws HeadlessException

使用指定的同位体接口创建此工具包的 Desktop 实现。

参数：

target - 要实现的 Desktop

返回：

此工具包的 Desktop 实现

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：

1.6

另请参见：

GraphicsEnvironment.isHeadless(),Desktop,DesktopPeer

---

createButton

protected abstract java.awt.peer.ButtonPeer createButton(Button target)
throws HeadlessException

使用指定同位体接口创建此工具包的 Button 实现。

参数：

target - 要实现的按钮。

返回：

此工具包的 Button 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),Button,ButtonPeer

---

createTextField

protected abstract java.awt.peer.TextFieldPeer createTextField(TextField target) throws HeadlessException

使用指定同位体接口创建此工具包的 TextField 实现。

参数：

target - 要实现的文本字段。

返回：

此工具包的 TextField 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),TextField,TextFieldPeer

---

createLabel

protected abstract java.awt.peer.LabelPeer createLabel(Label target) throws HeadlessException

使用指定同位体接口创建此工具包的 Label 实现。

参数：

target - 要实现的标签。

返回：

此工具包的 Label 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),Label,LabelPeer

createList

protected abstract java.awt.peer.ListPeer createList(List target)
throws HeadlessException

使用指定同位体接口创建此工具包的 List 实现。

参数：

target - 要实现的列表。

返回：

此工具包的 List 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),List,ListPeer

createCheckbox

protected abstract java.awt.peer.CheckboxPeer createCheckbox(Checkbox target)
throws HeadlessException

使用指定同位体接口创建此工具包的 Checkbox 实现。

参数：

target - 要实现的复选框。

返回：

此工具包的 Checkbox 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),Checkbox,CheckboxPeer

createScrollbar

protected abstract java.awt.peer.ScrollbarPeer createScrollbar(Scrollbar target) throws HeadlessException

使用指定同位体接口创建此工具包的 Scrollbar 实现。

参数：

target - 要实现的滚动条。

返回：

此工具包的 Scrollbar 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),Scrollbar,ScrollbarPeer

---

createScrollPane

protected abstract java.awt.peer.ScrollPanePeer createScrollPane(ScrollPane target) throws HeadlessException

使用指定同位体接口创建此工具包的 ScrollPane 实现。

参数：

target - 要实现的滚动窗格。

返回：

此工具包的 ScrollPane 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：

JDK1.1

另请参见：

GraphicsEnvironment.isHeadless(),ScrollPane,ScrollPanePeer

---

createTextArea

protected abstract java.awt.peer.TextAreaPeer createTextArea(TextArea target) throws HeadlessException

使用指定同位体接口创建此工具包的 TextArea 实现。

参数：

target - 要实现的文本字区域。

返回：

此工具包的 TextArea 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),TextArea,TextAreaPeer

---

createChoice

protected abstract java.awt.peer.ChoicePeer createChoice(Choice target) throws HeadlessException

使用指定同位体接口创建此工具包的 Choice 实现。

参数：

target - 要实现的选择。

返回：

此工具包的 Choice 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),Choice,ChoicePeer

---

createFrame

protected abstract java.awt.peer.FramePeer createFrame(Frame target) throws HeadlessException

使用指定同位体接口创建此工具包的 Frame 实现。

参数：

target - 要实现的窗体。

返回：

此工具包的 Frame 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),Frame,FramePeer

---

createCanvas

protected abstract java.awt.peer.CanvasPeer createCanvas(Canvas target)

使用指定同位体接口创建此工具包的 Canvas 实现。

参数：

target - 要实现的画布。

返回：

此工具包的 Canvas 实现。

另请参见：

Canvas,CanvasPeer

---

createPanel

protected abstract java.awt.peer.PanelPeer createPanel(Panel target)

使用指定同位体接口创建此工具包的 Panel 实现。

参数：

target - 要实现的面板。

返回：

此工具包的 Panel 实现。

另请参见：

Panel,PanelPeer

---

createWindow

protected abstract java.awt.peer.WindowPeer createWindow(Window target)

throws HeadlessException

使用指定同位体接口创建此工具包的 Window 实现。

参数：

target - 要实现的窗口。

返回：

此工具包的 Window 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),Window,WindowPeer

createDialog

protected abstract java.awt.peer.DialogPeer createDialog(Dialog target) throws HeadlessException

使用指定同位体接口创建此工具包的 Dialog 实现。

参数：

target - 要实现的对话框。

返回：

此工具包的 Dialog 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),Dialog,DialogPeer

createMenuBar

protected abstract java.awt.peer.MenuBarPeer createMenuBar(MenuBar target) throws HeadlessException

使用指定同位体接口创建此工具包的 MenuBar 实现。

参数：

target - 要实现的菜单栏。

返回：

此工具包的 MenuBar 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),MenuBar,MenuBarPeer

---

createMenu

```
protected abstract java.awt.peer.MenuPeer createMenu(Menu target)
                                                     throws HeadlessException
```

使用指定同位体接口创建此工具包的 Menu 实现。

参数：

target - 要实现的菜单。

返回：

此工具包的 Menu 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),Menu,MenuPeer

---

createPopupMenu

```
protected abstract java.awt.peer.PopupMenuPeer
createPopupMenu(PopupMenu target)
                                                     throws
HeadlessException
```

使用指定同位体接口创建此工具包的 PopupMenu 实现。

参数：

target - 要实现的弹出菜单。

返回：

此工具包的 PopupMenu 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：

JDK1.1

另请参见：

GraphicsEnvironment.isHeadless(),PopupMenu,PopupMenuPeer

---

createMenuItem

protected abstract java.awt.peer.MenuItemPeer createMenuItem(MenuItem target)

throws HeadlessException

使用指定同位体接口创建此工具包的 MenuItem 实现。

参数：

target - 要实现的菜单项。

返回：

此工具包的 MenuItem 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),MenuItem,MenuItemPeer

---

createFileDialog

protected abstract java.awt.peer.FileDialogPeer createFileDialog(FileDialog target)

throws HeadlessException

使用指定同位体接口创建此工具包的 FileDialog 实现。

参数：

target - 要实现的文件对话框。

返回：

此工具包的 FileDialog 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(),FileDialog,FileDialogPeer

createCheckboxMenuItem

protected abstract java.awt.peer.CheckboxMenuItemPeer createCheckboxMenuItem(CheckboxMenuItem target)

throws HeadlessException

使用指定同位体接口创建此工具包的 CheckboxMenuItem 实现。

参数：

target - 要实现的复选菜单项。

返回：

此工具包的 CheckboxMenuItem 实现。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless(), CheckboxMenuItem, CheckboxMenuItemPeer

getMouseInfoPeer

protected java.awt.peer.MouseInfoPeer getMouseInfoPeer()

使用指定同位体接口创建此工具包的 MouseInfo 实现。

返回：

此工具包的 MouseInfo 的帮助器实现

抛出：

UnsupportedOperationException - 如果没有实现此操作

从以下版本开始：

1.5

另请参见：

MouseInfoPeer,MouseInfo

createComponent

protected java.awt.peer.LightweightPeer createComponent(Component target)

创建组件或容器的同位体。此同位体是无窗口的，允许直接扩展 Component 和 Container 类，以创建完全在 Java 中定义的无窗口组件。

参数：

target - 要创建的 Component。

---

getFontPeer

@Deprecated

protected abstract java.awt.peer.FontPeer getFontPeer(String name,

int style)

已过时。 *请参阅 java.awt.GraphicsEnvironment#getAllFonts*

使用指定同位体接口创建此工具包的 Font 实现。

参数：

name - 要实现的字体

style - 字体样式，比如 PLAIN、BOLD、ITALIC 或它们的组合

返回：

此工具包的 Font 实现

另请参见：

Font,FontPeer,GraphicsEnvironment.getAllFonts()

---

loadSystemColors

protected void loadSystemColors(int[] systemColors)
throws HeadlessException

使用当前系统颜色值填充作为参数提供的整数数组。

参数：

systemColors - 一个整数数组。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：

JDK1.1

另请参见：

GraphicsEnvironment.isHeadless()

setDynamicLayout

```
public void setDynamicLayout(boolean dynamic)
                             throws HeadlessException
```

控制 Container 的布局是在调整大小期间动态地生效，还是在完成调整大小后静态地生效。注意，并非所有平台都支持此功能，反之，在某些平台上不能关闭此功能。在不支持（或总是支持）调整大小期间动态布局的平台上设置此属性无效。注意，在某些平台上，此功能可以作为操作系统或窗口管理器的一个属性来设置或取消设置。在这种平台上，在此方法生效前，必须在操作系统或窗口管理器级别上设置动态调整大小的属性。此方法不能改变底层操作系统或窗口管理器的支持或设置。使用 getDesktopProperty("awt.dynamicLayoutSupported") 可以查询 OS/WM 支持。

参数：

dynamic - 如果该参数为 true，则在调整 Container 大小时重新布置其组件。如果该参数为 false，则布局将在重新调整大小后生效。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：

1.4

另请参见：

isDynamicLayoutSet(), isDynamicLayoutActive(), getDesktopProperty(String propertyName), GraphicsEnvironment.isHeadless()

isDynamicLayoutSet

```
protected boolean isDynamicLayoutSet()
                                throws HeadlessException
```

返回 Container 的布局是在调整大小期间动态地生效，还是在完成调整大小后静态地生效。注：此方法返回的值是以编程方式设置的，不会影响调整大小时对动态布局的操作系统或窗口管理器级的支持，也不会影响当前操作系统或窗口管理器设置。使用 getDesktopProperty("awt.dynamicLayoutSupported") 可以查询 OS/WM 支持。

返回：

如果动态地使 Container 有效，则返回 true，如果在结束对大小的调整后使 Containers 有效，则返回 false。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：

1.4

另请参见：

setDynamicLayout(boolean dynamic), isDynamicLayoutActive(), getDesktopProperty(String propertyName), GraphicsEnvironment.isHeadless()

isDynamicLayoutActive

```
public boolean isDynamicLayoutActive()
                                throws HeadlessException
```

返回当前是否激活了调整大小时的动态布局（以编程方式设置，并受底层操作系统和/或窗口管理器支持）。使用 getDesktopProperty("awt.dynamicLayoutSupported") 可以查询 OS/WM 支持。

返回：

如果当前激活了调整大小时的动态布局，则返回 true；否则返回 false。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：

1.4

另请参见：

setDynamicLayout(boolean dynamic), isDynamicLayoutSet(), getDesktopProperty(String propertyName), GraphicsEnvironment.isHeadless()

getScreenSize

```
public abstract Dimension getScreenSize()
                                throws HeadlessException
```

获取屏幕的大小。在具有多个显示屏的系统上，使用主显示屏。从

GraphicsConfiguration 和 GraphicsDevice 可以获得多屏幕感知显示尺寸。

返回：

此工具包的屏幕大小，以像素为单位。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsConfiguration.getBounds(),
GraphicsDevice.getDisplayMode(),GraphicsEnvironment.isHeadless()

---

getScreenResolution

```
public abstract int getScreenResolution()
                                    throws HeadlessException
```

返回屏幕分辨率，以每英寸的点数为单位。

返回：

此工具包的屏幕分辨率，以每英寸的点数为单位。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

另请参见：

GraphicsEnvironment.isHeadless()

---

getScreenInsets

```
public Insets getScreenInsets(GraphicsConfiguration gc)
                                    throws HeadlessException
```

获得屏幕的 insets。

参数：

gc - 一个 GraphicsConfiguration

返回：

此工具包屏幕的 insets，以像素为单位。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：

1.4

另请参见：
GraphicsEnvironment.isHeadless()

getColorModel

```
public abstract ColorModel getColorModel()
                                    throws HeadlessException
```

确定此工具包屏幕的颜色模型。

ColorModel 是一个抽象类，封装了图像的像素值及其红色、绿色、蓝色和 alpha 组件之间的转换能力。

此工具包方法由 Component 类的 getColorModel 方法调用。

返回：
此工具包屏幕的颜色模型。
抛出：
HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true
另请参见：
GraphicsEnvironment.isHeadless(), ColorModel, Component.getColorModel()

getFontList

```
@Deprecated
public abstract String[] getFontList()
```

已过时。 *请参阅 GraphicsEnvironment.getAvailableFontFamilyNames()*
返回此工具包中可用字体的名称。

在 1.1 中，以下字体名称已经过时，括号内是替代名称：

- TimesRoman（使用 Serif 替代）
- Helvetica（使用 SansSerif 替代）
- Courier（使用 Monospaced 替代）

在 1.1 中字体名称 ZapfDingbats 也已经过时，但 Unicode 中定义的字符从 0x2700 开始。从 1.1 开始，Java 支持这些字符。

返回：
此工具包中可用字体的名称。
另请参见：
GraphicsEnvironment.getAvailableFontFamilyNames()

---

getFontMetrics

```
@Deprecated
    public abstract FontMetrics getFontMetrics(Font font)
```

已过时。 *从 JDK version 1.2 开始，由 Font 方法 getLineMetrics 取代。*
获取呈现字体的屏幕设备规格。
参数：
font - 字体
返回：
此工具包中指定字体的屏幕规格
另请参见：
LineMetrics, Font.getLineMetrics(java.lang.String, java.awt.font.FontRenderContext), GraphicsEnvironment.getScreenDevices()

---

sync

```
    public abstract void sync()
```

同步此工具包的图形状态。某些窗口系统可能会缓存图形事件。

此方法确保显示是最新的。这在动画制作时很有用。

---

getDefaultToolkit

public static Toolkit getDefaultToolkit()

获取默认工具包。

如果名为 "java.awt.headless" 的系统属性被设置为 true，则使用 Toolkit 的 headless 实现。

如果不存在 "java.awt.headless" 或 "java.awt.headless" 被设置为 false，且存在名为 "awt.toolkit" 的系统属性，则该属性将被视为 Toolkit 子类的名称；否则将使用特定于平台的默认 Toolkit 实现。

还可以使用 Sun 引用实现中指定的属性 'assistive_technologies' 将其他类加载到 VM 中，该属性是在 'accessibility.properties' 文件的一个行中指定的。形式是 "assistive_technologies=..."，其中 "..." 是以逗号分隔的、要加载的辅助技术类的列表。每个类都以给定的顺序加载，并且要使用 Class.forName(class).newInstance() 创建每个类的单独实例。此操作在创建 AWT 工具包之后进行。所有错误都通过 AWTError 异常来处理。

返回：

默认工具包。

抛出：

AWTError - 如果不能找到工具包，或者不能访问或实例化工具包。

getImage

public abstract Image getImage(String filename)

返回一幅图像，该图像从指定文件中获取像素数据，图像格式可以是 GIF、JPEG 或 PNG。底层工具包试图对具有相同文件名的多个请求返回相同的 Image。

因为便利 Image 对象共享所需的机制可能在一段不明确的时间内继续保存不再使用的图像，所以鼓励开发者在所有可能的地方使用 createImage 变体实现自己的图像缓存。如果包含在指定文件中的图像数据发生了更改，则此方法返回的 Image 对象仍然包含前一个调用之后从该文件加载的旧信息。通过对返回的 Image 调用 flush 方法，可以手动丢弃以前加载的信息。

此方法首先检查是否安装了安全管理器。如果安装了安全管理器，则该方法使用指定的文件调用安全管理器的 checkRead 方法，以确保允许访问该图像。

参数：

filename - 以可识别文件格式包含像素数据的文件名。

返回：

从指定文件中获取像素数据的图像。

抛出：

SecurityException - 如果存在安全管理器且其 checkListen 方法不允许该操作。

另请参见：

createImage(java.lang.String)

getImage

public abstract Image getImage(URL url)

返回一幅图像，该图像从指定 URL 获取像素数据。通过指定 URL 引用的像素数据必须使用以下格式之一：GIF、JPEG 或 PNG。底层工具包试图对具有相同 URL 的多个请求返回相同的 Image。

因为便利 Image 对象共享所需的机制可能在一段不明确的时间内继续保存不再使用的图像，所以鼓励开发者在所有可能的地方使用 createImage 变体实现自己的图像缓存。如果包含在指定 URL 中的图像数据发生了更改，则此方法返回的 Image 对象仍然包含前一个调用之后从该 URL 加载的旧信息。通过对返回的 Image 调用 flush 方法，可以手动丢弃以前加载的信息。

此方法首先检查是否安装了安全管理器。如果安装了安全管理器，则该方法通过 url.openConnection().getPermission() 权限调用安全管理器的 checkRead 方法，以确保允许访问该图像。为了与 1.2 以前的安全管理器兼容，如果通过 FilePermission 或 SocketPermission 的访问被拒绝，而对应的 1.1 样式的 SecurityManager.checkXXX 也拒绝访问，则该方法抛出 SecurityException。

参数：

url - 用来获取像素数据的 URL。

返回：

从指定 URL 获取像素数据的图像。

抛出：

SecurityException - 如果存在安全管理器且其 checkPermission 方法不允许该操作。

另请参见：

createImage(java.net.URL)

createImage

public abstract Image createImage(String filename)

返回从指定文件获取像素数据的图像。返回的 Image 是一个新对象，该对象不再由此方法的其他任何调用者或其 getImage 变体共享。

此方法首先检查是否安装了安全管理器。如果安装了安全管理器，则该方法使用指定的文件调用安全管理器的 checkRead 方法，以确保允许创建该图像。

参数：

filename - 以可识别文件格式包含像素数据的文件名。

返回：

从指定文件获取像素数据的图像。

抛出：

SecurityException - 如果存在安全管理器且其 checkRead 方法不允许该操作。

另请参见：

getImage(java.lang.String)

createImage

public abstract Image createImage(URL url)

返回一幅图像，该图像从指定 URL 获取像素数据。返回的 Image 是一个新对象，该对象不再由此方法的其他任何调用者或其 getImage 变体共享。

此方法首先检查是否安装了安全管理器。如果安装了安全管理器，则该方法通过 url.openConnection().getPermission() 权限调用安全管理器的 checkPermission 方法，以确保允许创建图像。为了与 1.2 以前的安全管理器兼容，如果通过 FilePermission 或 SocketPermission 的访问

被拒绝，而对应的 1.1 样式的 SecurityManager.checkXXX 也拒绝访问，则该方法抛出 SecurityException。

参数：
url - 用来获取像素数据的 URL。
返回：
从指定 URL 获取像素数据的图像。
抛出：
SecurityException - 如果存在安全管理器且其 checkPermission 方法不允许该操作。
另请参见：
getImage(java.net.URL)

---

prepareImage

```
public abstract boolean prepareImage(Image image,
                                     int width,
                                     int height,
                                     ImageObserver observer)
```

准备一个用于呈现的图像。

如果 width 和 height 参数的值均为 -1，则此方法在默认屏幕上准备一个用于呈现的图像；否则此方法在默认屏幕上以指定宽度和高度准备一个用于呈现的图像。

图像数据由另一个线程异步下载，并将生成适当缩放的图像屏幕表示形式。

此方法由组件的 prepareImage 方法调用。

可在 ImageObserver 接口的定义中找到有关此方法返回的标志的信息。

参数：
image - 准备屏幕显示的图像。
width - 需要的屏幕显示宽度，或为 -1。
height - 需要的屏幕显示高度，或为 -1。
observer - 在准备图像时要通知的 ImageObserver 对象。
返回：

如果已完全准备好了图像，则返回 true；否则返回 false。

另请参见：

Component.prepareImage(java.awt.Image, java.awt.image.ImageObserver), Component.prepareImage(java.awt.Image, int, int, java.awt.image.ImageObserver), ImageObserver

checkImage

```
public abstract int checkImage(Image image,
                               int width,
                               int height,
                               ImageObserver observer)
```

指示正准备显示的指定图像的构造状态。

如果 width 和 height 参数的值均为 -1，则此方法返回此工具包中指定图像屏幕表示形式的构造状态。否则，此方法以指定宽度和高度返回图像缩放表示形式的构造状态。

此方法不会导致开始加载图像。应用程序必须调用 prepareImage 来强制加载图像。

此方法由组件的 checkImage 方法调用。

可在 ImageObserver 接口的定义中找到有关此方法返回的标志的信息。

参数：

image - 要检查状态的图像。
width - 要检查状态的图像的缩放宽度，或为 -1。
height - 要检查状态的图像的缩放高度，或为 -1。
observer - 在准备图像时要通知的 ImageObserver 对象。

返回：

当前可用图像数据的 ImageObserver 标志的逐位 **OR**。

另请参见：

prepareImage(java.awt.Image, int, int, java.awt.image.ImageObserver), Component.checkImage(java.awt.Image,

java.awt.image.ImageObserver),
Component.checkImage(java.awt.Image, int, int,
java.awt.image.ImageObserver),ImageObserver

createImage

public abstract Image createImage(ImageProducer producer)

使用指定的图像生成器创建一幅图像。

参数：

producer - 要使用的图像生成器。

返回：

使用指定图像生成器创建的图像。

另请参见：

Image, ImageProducer,
Component.createImage(java.awt.image.ImageProducer)

createImage

public Image createImage(byte[] imagedata)

创建一幅图像，该图像对存储在指定字节数组中的图像进行解码。

数据必须使用此工具包支持的图像格式存储，比如 GIF 或 JPEG。

参数：

imagedata - 字节数组，表示用受支持图像格式存储的图像数据。

返回：

一幅图像。

从以下版本开始：

JDK1.1

createImage

public abstract Image createImage(byte[] imagedata,

```
                                    int imageoffset,
                                    int imagelength)
```

创建一幅图像，该图像对存储在指定字节数组中指定偏移量和长度处的图像进行解码。数据必须用受此工具包支持的图像格式存储，比如 GIF 或 JPEG。

参数：

imagedata - 字节数组，表示用受支持图像格式存储的图像。

imageoffset - 数组中数据开始的偏移量。

imagelength - 数组中数据的长度。

返回：

一幅图像。

从以下版本开始：

JDK1.1

getPrintJob

```
public abstract PrintJob getPrintJob(Frame frame,
                                     String jobtitle,
                                     Properties props)
```

获取一个 PrintJob 对象，该对象是在工具包平台上初始化某个打印操作的结果。

此方法的每个实际实现都应该先检查是否安装了安全管理器。如果安装了安全管理器，则该方法应该调用安全管理器的 checkPrintJobAccess 方法，以确保允许初始化打印操作。如果使用 checkPrintJobAccess 的默认实现（即未重写该方法），则将导致使用 RuntimePermission("queuePrintJob") 权限调用安全管理器的 checkPermission 方法。

参数：

frame - 打印对话框的父容器。不能为 null。

jobtitle - PrintJob 的标题。null 标题等同于 ""。

props - 包含零个或更多属性的 Properties 对象。这些属性没有被标准化，并且在各个实现之间不一致。因此，要求作业和页面控制的 PrintJob 应该使用此方法带有 JobAttributes 和 PageAttributes 对象的版本。可以更新此对象来反映用户离开时的作业选择。可以为 null。

返回：

一个 PrintJob 对象；如果用户取消打印作业，则返回 null。

抛出：

NullPointerException - 如果 frame 为 null。GraphicsEnvironment.isHeadless() 返回 true 时总是抛出此异常。

SecurityException - 如果此线程不允许初始化打印作业请求

从以下版本开始：

JDK1.1

另请参见：

GraphicsEnvironment.isHeadless(),PrintJob,RuntimePermission

---

getPrintJob

```
public PrintJob getPrintJob(Frame frame,
                            String jobtitle,
                            JobAttributes jobAttributes,
                            PageAttributes pageAttributes)
```

获取一个 PrintJob 对象，该对象是在工具包平台上初始化某个打印操作的结果。

此方法的每个实际实现都应该先检查是否安装了安全管理器。如果安装了安全管理器，则该方法应该调用安全管理器的 checkPrintJobAccess 方法，以确保允许初始化打印操作。如果使用 checkPrintJobAccess 的默认实现（即未重写该方法），则将导致使用 RuntimePermission("queuePrintJob") 权限调用安全管理器的 checkPermission 方法。

参数：

frame - 打印对话框的父级。当且仅当 jobAttributes 不为 null 且 jobAttributes.getDialog() 返回 JobAttributes.DialogType.NONE 或 JobAttributes.DialogType.COMMON 时，该参数可为 null。

jobtitle - PrintJob 的标题。null 标题等同于 ""。

jobAttributes - 控制 PrintJob 的作业属性集合。这些属性将被更新，以反映用户的选择，这概括在 JobAttributes 文档中。可以为 null。

pageAttributes - 控制 PrintJob 的页面属性集合。这些属性将应用于作业中的每个页面。这些属性将被更新，以反映用户的选择，这概括在 PageAttributes 文档中。可以为 null。

返回：

一个 PrintJob 对象；如果用户取消打印作业，则返回 null。

抛出：

NullPointerException - 如果 frame 为 null，并且 jobAttributes 为 null 或者 jobAttributes.getDialog() 返回 JobAttributes.DialogType.NATIVE。

IllegalArgumentException - 如果 pageAttributes 通过馈送然后解析馈送这一过程来指定不同之处。如果此线程可访问文件系统且 jobAttributes 指定打印到文件，则出现以下情况时也将抛出该异常：指定目标文件存在但它是一个目录而不是常规文件；指定目标文件不存在但不能创建；指定目标文件由于某些其他原因无法打开。但是，在指定打印到文件的情况下，如果同时请求了显示对话框，那么用户将有机会选择一个文件并继续打印。在从此方法返回前，该对话框将保证选择的输出文件有效。

GraphicsEnvironment.isHeadless() 返回 true 时总是抛出此异常。

SecurityException - 如果此线程不允许初始化打印作业请求，或者 jobAttributes 指定打印到文件，并且此线程不允许访问该文件系统

从以下版本开始：

1.3

另请参见：

PrintJob, GraphicsEnvironment.isHeadless(), RuntimePermission, JobAttributes,PageAttributes

---

beep

```
public abstract void beep()
```

发出一个音频嘟嘟声。

从以下版本开始：

JDK1.1

---

getSystemClipboard

```
public abstract Clipboard getSystemClipboard()
                                    throws HeadlessException
```

获取系统 Clipboard 的一个实例，该 Clipboard 可作为本机平台提供的剪贴板工具的接口。该剪贴板使数据能够在 Java 应用程序和使用本机剪贴板工具的本机应用程序之间传输。

除了 flavormap.properties 文件（或 AWT.DnD.flavorMapFileURL Toolkit 属性指定的其他文件）中指定的所有格式之外，系统 Clipboard 的 getTransferData() 方法返回的文本在以下 flavor 中是可用的：

- DataFlavor.stringFlavor
- DataFlavor.plainTextFlavor（**已过时**）

在使用 java.awt.datatransfer.StringSelection 时，如果要求的 flavor 是 DataFlavor.plainTextFlavor 或等价 flavor，则返回一个 Reader。**注：**系统 Clipboard 针对 DataFlavor.plainTextFlavor 和等价 DataFlavor 的 getTransferData() 方法的行为与 DataFlavor.plainTextFlavor 的定义是不一致的。因此，对 DataFlavor.plainTextFlavor 和等价 flavor 的支持**已过时**。

此方法的每个实际实现都应该先检查是否安装了安全管理器。如果安装了安全管理器，则此方法应该调用安全管理器的 checkSystemClipboardAccess 方法，以确保可以访问系统剪贴板。如果使用 checkSystemClipboardAccess 的默认实现（即未重写该方法），则将导致使用 AWTPermission("accessClipboard") 权限调用安全管理器的 checkPermission 方法。

返回：
系统 Clipboard
抛出：
HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true
从以下版本开始：
JDK1.1
另请参见：
GraphicsEnvironment.isHeadless(), Clipboard, StringSelection, DataFlavor.stringFlavor, DataFlavor.plainTextFlavor, Reader, AWTPermission

---

getSystemSelection

```
public Clipboard getSystemSelection()
                                    throws HeadlessException
```

以 Clipboard 对象的形式获取系统选择的一个实例。这使应用程序能够读取和修改当前系统范围内的选择。

无论用户在何时使用鼠标或键盘选择了文本，应用程序都要负责更新系统选择。通常，实现方式是在所有支持文本选择的 Component 上，以及传递给该 Component 的 FOCUS_GAINED 和 FOCUS_LOST 事件之间安装一个 FocusListener，并在 Component 内的选择发生更改时更新系统选择 Clipboard。恰当地更新系统选择确保了 Java 应用程序与本机应用程序和同时运行在系统上的其他 Java 应用程序正确交互。注意，java.awt.TextComponent 和 javax.swing.text.JTextComponent 已支持此策略。在使用这些类及其子类时，开发人员不需要编写任何额外的代码。

一些平台不支持系统选择 Clipboard。在这些平台上，此方法将返回 null。在这种情况下，应用程序不再有责任更新系统选择 Clipboard（如上所述）。

此方法的每个实际实现都应该先检查是否安装了 SecurityManager。如果是安装了 SecurityManager，则应调用 SecurityManager 的 checkSystemClipboardAccess 方法，以确保可以访问系统剪贴板。如果使用 checkSystemClipboardAccess 的默认实现（即如果未重写该方法），则将导致使用 AWTPermission("accessClipboard") 权限调用 SecurityManager 的 checkPermission 方法。

返回：

以 Clipboard 形式返回系统选择，如果本机平台不支持系统选择 Clipboard，则返回 null

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：

1.4

另请参见：

Clipboard, FocusListener, FocusEvent.FOCUS_GAINED, FocusEvent.FOCUS_LOST, TextComponent, JTextComponent, AWTPermission, GraphicsEnvironment.isHeadless()

getMenuShortcutKeyMask

public int getMenuShortcutKeyMask()
throws HeadlessException

确定哪一个修改键是菜单快捷方式的适当加速键。

菜单快捷方式嵌入在 MenuShortcut 类中，由 MenuBar 类处理。

默认情况下，此方法返回 Event.CTRL_MASK。如果 Control 键不是正确的加速键，则工具包实现应该重写此方法。

返回：
此工具包中用于菜单快捷方式的 Event 类的修饰符掩码。
抛出：
HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true
从以下版本开始：
JDK1.1
另请参见：
GraphicsEnvironment.isHeadless(),MenuBar,MenuShortcut

getLockingKeyState

public boolean getLockingKeyState(int keyCode)
throws UnsupportedOperationException

返回键盘上给定锁定键是否处于 "on" 状态。有效键代码是 VK_CAPS_LOCK、VK_NUM_LOCK、VK_SCROLL_LOCK 和 VK_KANA_LOCK。
抛出：
IllegalArgumentException - 如果 keyCode 不是有效键代码之一
UnsupportedOperationException - 如果主机系统不允许以编程方式获取此键的状态，或者键盘没有此键
HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true
从以下版本开始：
1.3
另请参见：
GraphicsEnvironment.isHeadless()

setLockingKeyState

```
public void setLockingKeyState(int keyCode,
                               boolean on)
                        throws UnsupportedOperationException
```

设置键盘上给定锁定键的状态。有效键代码是 VK_CAPS_LOCK、VK_NUM_LOCK、VK_SCROLL_LOCK 和 VK_KANA_LOCK。

根据不同的平台，设置锁定键的状态可能涉及事件处理，因此不能立即通过 getLockingKeyState 观察到。

抛出：

IllegalArgumentException - 如果 keyCode 不是有效键代码之一

UnsupportedOperationException - 如果主机系统不允许以编程方式设置此键的状态，或者键盘没有此键

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：

1.3

另请参见：

GraphicsEnvironment.isHeadless()

getNativeContainer

```
protected static Container getNativeContainer(Component c)
```

使本机同位体能够查询给定本机组件的本机容器（例如，直接父组件可以是轻量级的）。

createCustomCursor

```
public Cursor createCustomCursor(Image cursor,
                                 Point hotSpot,
                                 String name)
                          throws IndexOutOfBoundsException,
                                 HeadlessException
```

创建一个新的自定义光标对象。如果要显示的图像无效，则隐藏光标（使其完全透明），并将热点 (hotspot) 设置为（0，0）。

注意，多帧图像是无效的，可能造成此方法被挂起。

参数：

cursor - 激活光标时要显示的图像

hotSpot - 大光标热点的 X 和 Y 坐标；hotSpot 值必须小于 getBestCursorSize 返回的 Dimension

name - 光标的本地化描述，用于 Java Accessibility

抛出：

IndexOutOfBoundsException - 如果 hotSpot 值超出光标边界

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：

1.2

另请参见：

GraphicsEnvironment.isHeadless()

getBestCursorSize

```
public Dimension getBestCursorSize(int preferredWidth,
                                   int preferredHeight)
                            throws HeadlessException
```

返回最接近所需大小的受支持光标尺寸。无论所需的大小如何，只支持单个光标大小的系统将返回该大小。不支持自定义光标的系统返回的尺寸将是 0, 0。

注：如果使用的图像尺寸不符合受支持的大小（由此方法返回的尺寸），则 Toolkit 实现会试着将图像的大小调整为受支持的大小。因为转换低分辨率图像存在一些困难，所以不保证不受支持大小的光标图像的质量。因此建议调用此方法并使用合适的图像，从而不需要转换图像。

参数：

preferredWidth - 组件将使用的首选光标宽度。

preferredHeight - 组件将使用的首选光标高度。

返回：

最接近的受支持光标大小；如果 Toolkit 实现不支持自定义光标，则返回的尺寸为 0,0。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true

从以下版本开始：
1.2
另请参见：
GraphicsEnvironment.isHeadless()

getMaximumCursorColors

```
public int getMaximumCursorColors()
                                    throws HeadlessException
```

返回自定义光标调色板中 Toolkit 支持的最大颜色数。

注：如果图像调色板中使用的颜色超过所支持的最大颜色数，则 Toolkit 实现会试图将调色板所用颜色调整为最大颜色数。因为转换低分辨率图像存在一些困难，所以不保证颜色多于系统支持颜色的那些图像的质量。因此建议调用此方法并使用合适的图像，从而不需要转换图像。

返回：
最大颜色数；如果 Toolkit 实现不支持自定义光标，则返回零。
抛出：
HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true
从以下版本开始：
1.2
另请参见：
GraphicsEnvironment.isHeadless()

isFrameStateSupported

```
public boolean isFrameStateSupported(int state)
                                      throws HeadlessException
```

返回 Toolkit 是否支持 Frame 状态。此方法判断是否支持最大化或图标化的 *UI*。对于 Frame.ICONIFIED|Frame.MAXIMIZED_VERT 之类的“复合”状态，该方法总是返回 false。换句话说，根据经验只有使用单窗体状态常量作为参数的查询才是有意义的。
参数：

state - 指定的窗体状态常量之一。

返回：

如果此 Toolkit 实现支持窗体状态，则返回 true；否则返回 false。

抛出：

HeadlessException - 如果 GraphicsEnvironment.isHeadless() 返回 true。

从以下版本开始：

1.4

另请参见：

Frame.setExtendedState(int)

getProperty

```
public static String getProperty(String key,
                                 String defaultValue)
```

获取具有指定键和默认值的属性。如果未找到属性，此方法将返回 defaultValue。

getSystemEventQueue

```
public final EventQueue getSystemEventQueue()
```

获取应用程序或 applet 的 EventQueue 实例。根据 Toolkit 实现，对于不同的 applet，可能返回不同的 EventQueue。所以 applet 不能假设此方法返回的 EventQueue 实例将由其他 applet 或系统共享。

如果存在安全管理器，则首先调用其 checkAwtEventQueueAccess 方法。如果使用 checkAwtEventQueueAccess 的默认实现（即未重写该方法），则将导致使用 AWTPermission("accessEventQueue") 权限调用安全管理器的 checkPermission 方法。

返回：

EventQueue 对象

抛出：

SecurityException - 如果存在安全管理器且其 SecurityManager.checkAwtEventQueueAccess() 方法拒绝访问 EventQueue

另请参见：
AWTPermission

getSystemEventQueueImpl

```
protected abstract EventQueue getSystemEventQueueImpl()
```

获取应用程序或 applet 的 EventQueue 实例，不检查访问权。出于安全原因，只能从 Toolkit 子类调用此方法。

返回：
EventQueue 对象

createDragSourceContextPeer

```
public abstract java.awt.dnd.peer.DragSourceContextPeer
createDragSourceContextPeer(DragGestureEvent dge)

throws InvalidDnDOperationException
```

创建 DragSourceContext 的同位体。如果 GraphicsEnvironment.isHeadless() 返回 true，则总是抛出 InvalidDndOperationException。

抛出：
InvalidDnDOperationException

另请参见：
GraphicsEnvironment.isHeadless()

createDragGestureRecognizer

```
public <T extends DragGestureRecognizer> T
createDragGestureRecognizer(Class<T> abstractRecognizerClass,

DragSource ds,

Component c,

int srcActions,
```

DragGestureListener dgl)

创建所请求的抽象 DragGestureRecognizer 类的具体的、与平台有关的子类，并将它与指定的 DragSource、Component 和 DragGestureListener 关联。子类应该重写此方法，以提供自己的实现。

参数：

abstractRecognizerClass - 所需识别器的抽象类

ds - DragSource

c - DragGestureRecognizer 的 Component 目标

srcActions - 允许用于该动作的操作

dgl - DragGestureListener

返回：

新的对象或 null。如果 GraphicsEnvironment.isHeadless() 返回 true，则总是返回 null。

另请参见：

GraphicsEnvironment.isHeadless()

getDesktopProperty

public final Object getDesktopProperty(String propertyName)

获取指定桌面属性的值。桌面属性是作为 Toolkit 全局变量资源的惟一指定值。通常它也是与底层平台有关的桌面设置的抽象表示形式。 有关 AWT 支持的桌面属性的更多信息，请参阅 AWT Desktop Properties。

setDesktopProperty

protected final void setDesktopProperty(String name,
Object newValue)

将指定桌面属性设置为指定值，并触发一个属性更改事件，以通知所有侦听器该值已更改。

lazilyLoadDesktopProperty

protected Object lazilyLoadDesktopProperty(String name)

一个对桌面属性值延迟计算的机会。

initializeDesktopProperties

protected void initializeDesktopProperties()

initializeDesktopProperties

addPropertyChangeListener

public void addPropertyChangeListener(String name,
PropertyChangeListener pcl)

为指定的桌面属性添加指定的属性更改侦听器。如果 pcl 为 null，则不抛出任何异常，且不执行任何操作。

参数：

name - 要监听的属性的名称

pcl - 属性更改侦听器

从以下版本开始：

1.2

removePropertyChangeListener

public void removePropertyChangeListener(String name,
PropertyChangeListener pcl)

为指定的桌面属性移除指定的属性更改侦听器。如果 pcl 为 null，则不抛出任何异常，且不执行任何操作。

参数：

name - 要移除的属性的名称

pcl - 属性更改侦听器

从以下版本开始：

1.2

getPropertyChangeListeners

public PropertyChangeListener[] getPropertyChangeListeners()

返回在此工具包上所有已注册的属性更改侦听器所组成的数组。

返回：

此工具包的所有 PropertyChangeListener；如果当前没有注册的属性更改侦听器，则返回一个空数组

从以下版本开始：

1.4

---

getPropertyChangeListeners

public PropertyChangeListener[] getPropertyChangeListeners(String propertyName)

返回与指定属性关联的所有 PropertyChangeListener 所组成的数组。

参数：

propertyName - 指定的属性

返回：

与指定属性关联的所有 PropertyChangeListener；如果没有添加这样的侦听器，则返回一个空数组

从以下版本开始：

1.4

---

isAlwaysOnTopSupported

public boolean isAlwaysOnTopSupported()

返回此工具包是否支持 always-on-top 模式。要检测特定窗口是否支持 always-on-top 模式，请使用 Window.isAlwaysOnTopSupported()。

返回：

如果当前工具包支持 always-on-top 模式，则返回 true；否则返回 false

从以下版本开始：

1.6

另请参见：

Window.isAlwaysOnTopSupported(),Window.setAlwaysOnTop(boolean)

isModalityTypeSupported

public abstract boolean isModalityTypeSupported(Dialog.ModalityType modalityType)

返回此工具包是否支持给定的模式类型。如果创建了带有不受支持的模式类型的对话框，则使用 Dialog.ModalityType.MODELESS。

参数：

modalityType - 要检测是否受此工具包支持的模式类型

返回：

如果当前工具包支持给定模式类型，则返回 true；否则返回 false

从以下版本开始：

1.6

另请参见：

Dialog.ModalityType, Dialog.getModalityType(), Dialog.setModalityType(java.awt.Dialog.ModalityType)

isModalExclusionTypeSupported

public abstract boolean isModalExclusionTypeSupported(Dialog.ModalExclusionType modalExclusionType)

返回此工具包是否支持给定的模式排斥类型。如果在窗口上设置了不受支持的模式排斥类型，则使用 Dialog.ModalExclusionType.NO_EXCLUDE。

参数：

modalExclusionType - 要检测是否受此工具包支持的模式排斥类型

返回：

如果当前工具包支持给定模式排斥类型，则返回 true；否则返回 false

从以下版本开始：

1.6

另请参见：

Dialog.ModalExclusionType, Window.getModalExclusionType(), Window.setModalExclusionType(java.awt.Dialog.ModalExclusionType)

addAWTEventListener

```
public void addAWTEventListener(AWTEventListener listener,
                                 long eventMask)
```

添加一个 AWTEventListener，以接收与给定 eventMask 一致的系统范围内指派的所有 AWTEvent。

首先，如果存在安全管理器，则使用 AWTPermission("listenToAllAWTEvents") 权限调用 checkPermission 方法。这可能导致 SecurityException 异常。

eventMask 是要接收的事件类型的位掩码。它是通过对 AWTEvent 中定义的事件掩码进行逐位 OR 得到的。

注：对于一般的应用程序，不推荐使用事件侦听器，它只用于支持特定目的工具，包括支持可访问性、事件记录/回放和诊断跟踪。如果 listener 为 null，则不抛出任何异常，且不执行任何操作。

参数：

listener - 事件侦听器。

eventMask - 要接收的事件类型的位掩码

抛出：

SecurityException - 如果存在安全管理器且其 checkPermission 方法不允许进行此操作。

从以下版本开始：

1.2

另请参见：

removeAWTEventListener(java.awt.event.AWTEventListener), getAWTEventListeners(), SecurityManager.checkPermission(java.security.Permission), AWTEvent,AWTPermission,AWTEventListener,AWTEventListenerProxy

removeAWTEventListener

```
public void removeAWTEventListener(AWTEventListener listener)
```

从正接收的指派 AWTEvent 中移除一个 AWTEventListener。

首先，如果存在安全管理器，则使用 AWTPermission("listenToAllAWTEvents") 权限调用 checkPermission 方法。这可能导致 SecurityException 异常。

注：对于一般的应用程序，不推荐使用事件侦听器，它只用于支持特定目的工具，包括支持可访问性、事件记录/回放和诊断跟踪。如果 listener 为 null，则不抛出任何异常，且不执行任何操作。

参数：

listener - 事件侦听器。

抛出：

SecurityException - 如果存在安全管理器且其 checkPermission 方法不允许进行此操作。

从以下版本开始：

1.2

另请参见：

addAWTEventListener(java.awt.event.AWTEventListener, long), getAWTEventListeners(), SecurityManager.checkPermission(java.security.Permission), AWTEvent,AWTPermission,AWTEventListener,AWTEventListenerProxy

getAWTEventListeners

```
public AWTEventListener[] getAWTEventListeners()
```

返回在此工具包上所有已注册 AWTEventListener 所组成的数组。如果存在安全管理器，则使用 AWTPermission("listenToAllAWTEvents") 权限调用它的 checkPermission 方法。这可能导致 SecurityException 异常。侦听器可以在 AWTEventListenerProxy 对象中返回，该对象还包含给定侦听器的事件掩码。注意，多次添加的侦听器对象在返回数组中只出现一次。

返回：

所有 AWTEventListener；如果当前没有已注册侦听器，则返回一个空数组

抛出：

SecurityException - 如果存在安全管理器且其 checkPermission 方法不允许进行此操作。

从以下版本开始：

1.4

另请参见：

addAWTEventListener(java.awt.event.AWTEventListener, long), removeAWTEventListener(java.awt.event.AWTEventListener), SecurityManager.checkPermission(java.security.Permission), AWTEvent,AWTPermission,AWTEventListener,AWTEventListenerProxy

getAWTEventListeners

public AWTEventListener[] getAWTEventListeners(long eventMask)

返回已在此工具包上注册的所有 AWTEventListener 所组成的数组，这些侦听器侦听用 eventMask 参数指定的所有事件类型。如果存在安全管理器，则使用 AWTPermission("listenToAllAWTEvents") 权限调用它的 checkPermission 方法。这可能导致 SecurityException 异常。侦听器可以在 AWTEventListenerProxy 对象中返回，该对象还包含给定侦听器的事件掩码。注意，多次添加的侦听器对象在返回数组中只出现一次。

参数：

eventMask - 要监听的事件类型的位掩码

返回：

已在此工具包上为指定事件类型注册的所有 AWTEventListener；如果当前没有这样的已注册侦听器，则返回一个空数组

抛出：

SecurityException - 如果存在安全管理器且其 checkPermission 方法不允许该操作。

从以下版本开始：

1.4

另请参见：

addAWTEventListener(java.awt.event.AWTEventListener, long), removeAWTEventListener(java.awt.event.AWTEventListener), SecurityManager.checkPermission(java.security.Permission), AWTEvent,AWTPermission,AWTEventListener,AWTEventListenerProxy

mapInputMethodHighlight

public abstract Map<TextAttribute,?> mapInputMethodHighlight(InputMethodHighlight highlight)

throws HeadlessException

返回给定输入方法高亮区的抽象级别描述的可视属性映射，如果不存在映射关系，则返回 null。输入方法高亮区的样式字段被忽略。返回的映射是不可修改的。

参数：

highlight - 输入方法高亮区

返回：

样式属性映射，或者返回 null

抛出：

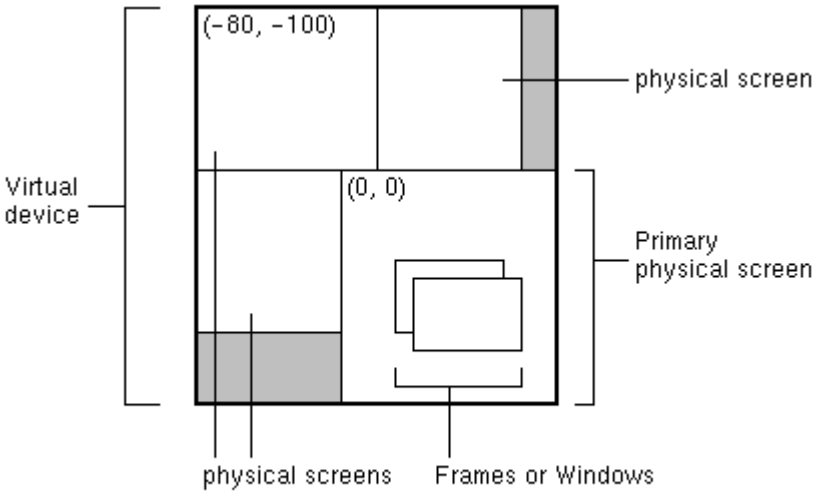
HeadlessException - 如果 GraphicsEnvironment.isHeadless 返回 true

Window 类

Window 对象是一个没有边界和菜单栏的顶层窗口。窗口的默认布局是 BorderLayout。构造窗口时，它必须拥有窗体、对话框或其他作为其所有者定义的窗口。

在多屏幕环境中，通过使用 Window(Window, GraphicsConfiguration) 构造 Window，可以在不同的屏幕设备上创建 Window。GraphicsConfiguration 对象是目标屏幕设备的 GraphicsConfiguration 对象之一。

在虚拟设备多屏幕环境中（其中桌面区域可以跨越多物理屏幕设备），所有配置的边界都是相对于虚拟设备坐标系的。虚拟坐标系的原点位于主物理屏幕的左上角。是否使用负坐标取决于主物理屏幕在虚拟设备中的位置，如下图所示。



在此环境中，调用 setLocation 时，必须传递一个虚拟坐标到此方法中。类似地，对 Frame 调用 getLocationOnScreen 将返回虚拟设备坐标。调用 GraphicsConfiguration 的 getBounds 方法，以查找它在虚拟坐标系中的原点。

以下代码将 Window 的位置设置在 (10, 10)（相对于相应 GraphicsConfiguration 物理屏幕的原点）。如果不考虑 GraphicsConfiguration 的边界，则 Window 位置应设置在 (10, 10)（相对于虚拟坐标系），并显示在主物理屏幕上，该屏幕可能不同于指定 GraphicsConfiguration 的物理屏幕。

```
Window w = new Window(Window owner, GraphicsConfiguration gc);
Rectangle bounds = gc.getBounds();
w.setLocation(10 + bounds.x, 10 + bounds.y);
```

注：顶层窗口（包括 Window、Frame 和 Dialog）的位置和大小受桌面窗口管理系统的控制。对 setLocation、setSize 和 setBounds 的调用是转发到窗口管理系统的请求（不是指令）。将尽所有努力响应这样的请求。但是，在某些情况下，窗口管理系统可以忽略这样的请求，或修改请求的几何结构，以放置和调整 Window 的大小，使之更好地与桌面设置匹配。

由于本机事件处理的异步特性，在处理完最后一个请求前，getBounds、getLocation、getLocationOnScreen 和 getSize 返回的结果可能不反映屏幕上窗口的实际几何结构。在处理后续请求的过程中，窗口管理系统满足这些请求时，这些值可能会相应地改变。

应用程序可以随意设置不可见 Window 的大小和位置，但是当 Window 可见时，窗口管理系统可以随后更改它的大小和/或位置。将生成一个或多个 ComponentEvent 来表示新的几何结构。

窗口能够生成以下 WindowEvents：WindowOpened、WindowClosed、indowGainedFocus、indowLostFocus。

构造方法详细信息

Window

```
public Window(Frame owner)
```

构造一个新的、最初不可见的窗口，使用指定的 Frame 作为其所有者。该窗口将不可聚焦，除非其所有者正显示在屏幕上。

如果存在安全管理器，则此方法首先调用安全管理器的 checkTopLevelWindow 方法（this 作为其参数），以确定是否必须使用警告标志显示该窗口。

参数：
owner - 要充当所有者的 Frame；如果此窗口没有所有者，则该参数为 null
抛出：
IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自屏幕设备
HeadlessException - 当 GraphicsEnvironment.isHeadless 返回 true 时
另请参见：
GraphicsEnvironment.isHeadless(),
SecurityManager.checkTopLevelWindow(java.lang.Object),
isShowing()

Window

public Window(Window owner)

构造一个新的、最初不可见的窗口，使用指定 Window 作为其所有者。此窗口将不可聚焦，除非其最近拥有的 Frame 或 Dialog 正显示在屏幕上。

如果存在安全管理器，则此方法首先调用安全管理器的 checkTopLevelWindow 方法（this 作为其参数），以确定是否必须使用警告标志显示该窗口。

参数：
owner - 要充当所有者的 Window；如果此窗口没有所有者，则该参数为 null
抛出：
IllegalArgumentException - 如果 owner 的 GraphicsConfiguration 不是来自屏幕设备
HeadlessException - 当 GraphicsEnvironment.isHeadless 返回 true 时
从以下版本开始：
1.2
另请参见：
GraphicsEnvironment.isHeadless(),
SecurityManager.checkTopLevelWindow(java.lang.Object),
isShowing()

Window

```
public Window(Window owner,
              GraphicsConfiguration gc)
```

构造一个新的、最初不可见的窗口，使用指定的所有者 Window 和屏幕设备的 GraphicsConfiguration。该窗口将不可聚焦，除非其最近拥有的 Frame 或 Dialog 正显示在屏幕上。

如果存在安全管理器，则此方法首先调用安全管理器的 checkTopLevelWindow 方法（this 作为其参数），以确定是否必须使用警告标志显示该窗口。

参数：

owner - 要作为所有者的窗口；如果此窗口没有所有者，则该参数为 null

gc - 目标屏幕设备的 GraphicsConfiguration；如果 gc 为 null，则使用系统默认的 GraphicsConfiguration

抛出：

IllegalArgumentException - 如果 gc 不是来自屏幕设备

HeadlessException - 当 GraphicsEnvironment.isHeadless() 返回 true 时

从以下版本开始：

1.3

另请参见：

GraphicsEnvironment.isHeadless(), SecurityManager.checkTopLevelWindow(java.lang.Object), GraphicsConfiguration.getBounds(),isShowing()

方法详细信息

getIconImages

```
public List<Image> getIconImages()
```

返回要作为此窗口的图标显示的图像序列。

此方法返回内部存储列表的一个副本，所以对返回对象的任何操作都不会影响窗口的行为。

返回：

此窗口的图标图像列表的副本；如果此窗口没有图标图像，则返回空列表。

从以下版本开始：

1.6
另请参见：
setIconImages(java.util.List),setIconImage(Image)

setIconImages

public void setIconImages(List<? extends Image> icons)

设置要作为此窗口的图标显示的图像序列。随后调用 getIconImages 将总是返回 icons 列表的一个副本。

各平台根据自身能力的不同，使用不同数量和尺寸的图像作为窗口的图标。

一开始便扫描 icons 列表，寻找最佳尺寸的图像。如果列表包含几个大小相同的图像，则使用第一个图像。

未指定图标的、没有所有者的窗口将使用平台默认的图标。如果没有显示重写，被拥有的窗口的图标可以从其所有者继承。将图标设置为 null 或空列表将恢复默认行为。

注：根据上下文的不同（例如，窗口装饰、窗口列表、任务栏等），本机窗口系统可以使用不同尺寸的不同图像表示一个窗口。也可以对所有上下文使用一个图像，或者根本不用图像。

参数：
icons - 要显示的图标图像列表。
从以下版本开始：
1.6
另请参见：
getIconImages(),setIconImage(Image)

setIconImage

public void setIconImage(Image image)

设置要作为此窗口图标显示的图像。

将单个图像指定为窗口的图标时，可以使用此方法代替 setIconImages()。

以下语句：

```
setIconImage(image);
```

等价于：

```
ArrayList        imageList = new ArrayList        ();
imageList.add(image);
setIconImages(imageList);
```

注：根据上下文的不同（例如，窗口装饰、窗口列表、任务栏等），本机窗口系统可以使用不同尺寸的不同图像表示一个窗口。也可以对所有上下文使用一个图像，或者根本不用图像。

参数：

image - 要显示的图标图像。

从以下版本开始：

1.6

另请参见：

setIconImages(java.util.List),getIconImages()

---

addNotify

```
public void addNotify()
```

通过创建到本机屏幕资源的连接，使此窗口变得可显示。此方法由工具包在内部进行调用，并且不应由程序直接调用。

覆盖：

类 Container 中的 addNotify

从以下版本开始：

JDK1.0

另请参见：

Component.isDisplayable(),Container.removeNotify()

---

removeNotify

public void removeNotify()

通过移除此 Container 到其本机屏幕资源的连接，使其不可显示。使容器变得不可显示会使其所有子容器都变得不可显示。此方法是通过工具包内部调用的，不应该通过程序直接调用它。

覆盖：

类 Container 中的 removeNotify

另请参见：

Component.isDisplayable(),Container.addNotify()

pack

public void pack()

调整此窗口的大小，以适合其子组件的首选大小和布局。如果该窗口和/或其所有者还不可显示，则在计算首选大小之前都将变得可显示。在计算首选大小之后，将会验证该窗口。

另请参见：

Component.isDisplayable()

setMinimumSize

public void setMinimumSize(Dimension minimumSize)

将此窗口的最小大小设置为一个常量值。随后调用 getMinimumSize 将总是返回此值。如果当前窗口的大小小于 minimumSize，则窗口的大小将自动增大到最小大小。

如果随后调用 setSize 或 setBounds 方法，且调用时使用的宽度或高度小于 setMinimumSize 指定的宽度或高度，则窗口将自动增大到 minimumSize 值。将最小大小设置为 null 将还原默认行为。

如果用户试图将窗口大小调整到 minimumSize 值以下，则该调整操作将受到限制。此行为与平台有关。

覆盖：

类 Component 中的 setMinimumSize

参数：

minimumSize - 此窗口新的最小大小

从以下版本开始：

1.6

另请参见：

Component.setMinimumSize(java.awt.Dimension), Container.getMinimumSize(), Component.isMinimumSizeSet(), setSize(Dimension)

setSize

```
public void setSize(Dimension d)
```

调整组件的大小，使其宽度为 d.width，高度为 d.height。

如果 d.width 值或 d.height 值小于之前调用 setMinimumSize 指定的最小大小，则它将自动增大。

覆盖：

类 Component 中的 setSize

参数：

d - 指定组件新大小的尺寸

从以下版本开始：

1.6

另请参见：

Component.getSize(), setBounds(int, int, int, int), setMinimumSize(java.awt.Dimension)

setSize

```
public void setSize(int width,
                    int height)
```

调整组件的大小，使其宽度为 width，高度为 height。

如果 width 值或 height 值小于之前调用 setMinimumSize 指定的最小大小，则它将自动增大。

覆盖：

类 Component 中的 setSize

参数：

width - 组件的新宽度，单位是像素

height - 组件的新高度，单位是像素

从以下版本开始：

1.6

另请参见：

Component.getSize(), setBounds(int, int, int, int), setMinimumSize(java.awt.Dimension)

reshape

```
@Deprecated
    public void reshape(int x,
                        int y,
                        int width,
                        int height)
```

已过时。 *从 JDK 1.1 版开始，由 setBounds(int, int, int, int) 取代。*

覆盖：

类 Component 中的 reshape

setVisible

```
    public void setVisible(boolean b)
```

根据参数 b 的值显示或隐藏此 Window。

覆盖：

类 Component 中的 setVisible

参数：

b - 如为 true，则使 Window 可见，否则隐藏 Window。如果 Window 和/或其所有者还不能显示，则都不显示。在使 Window 可见之前将验证它。如果 Window 已

经可见，则此方法将 Window 带到最前面。

如为 false，则隐藏此 Window、此 Window 的子组件，以及它拥有的所有子级。调用 #setVisible(true) 可以重新使 Window 及其子组件可见。

另请参见：

Component.isDisplayable(), Component.setVisible(boolean), toFront(),dispose()

show

```
@Deprecated
    public void show()
```

已过时。 *从 JDK 1.5 版开始，由 Component.setVisible(boolean) 取代。*

使窗口可见。如果窗口和/或其所有者还不能显示，则都不显示。在使窗口可见之前将验证它。如果窗口已经可见，则此方法将窗口带到最前面。

覆盖：

类 Component 中的 show

另请参见：

Component.isDisplayable(),toFront()

hide

```
@Deprecated
    public void hide()
```

已过时。 *从 JDK 1.5 版开始，由 Component.setVisible(boolean) 取代。*

隐藏此窗口、此窗口的子组件，以及它拥有的所有子级。调用 show 可以重新使窗口及其子组件可见。

覆盖：

类 Component 中的 hide

另请参见：

show(),dispose()

dispose

public void dispose()

释放由此 Window、其子组件及其拥有的所有子组件所使用的所有本机屏幕资源。即这些 Component 的资源将被破坏，它们使用的所有内存都将返回到操作系统，并将它们标记为不可显示。

通过随后调用 pack 或 show 重新构造本机资源，可以再次显示 Window 及其子组件。重新创建的 Window 及其子组件的状态与释放 Window 时这些对象的状态一致（不考虑这些操作之间的其他更改）。

注：当 Java 虚拟机 (VM) 中最后的可显示窗口被释放后，虚拟机可能会终止。有关更多信息，请参阅 AWT Threading Issues。

另请参见：
Component.isDisplayable(),pack(),show()

toFront

public void toFront()

如果此窗口是可见的，则将此窗口置于前端，并可以将其设为焦点 Window。

将此窗口放在堆栈顺序的顶层，并在此虚拟机中显示在所有其他窗口的上面。如果此窗口不可见，则不会发生任何操作。有些平台不允许拥有其他窗口的窗口显示在它所拥有的窗口之上。有些平台可能不允许此虚拟机将其窗口放在本机应用程序窗口或其他虚拟机窗口之上。此权限可能取决于此虚拟机中的窗口是否已被设为焦点窗口。将进行所有尝试来移动此窗口，使其位于堆栈顺序中尽可能靠前的位置；但是，开发人员不应假定此方法在所有情况下都可以将此窗口移到所有其他窗口之上。

由于本机窗口系统多种多样，因此无法保证对焦点窗口和活动窗口的更改能够实现。在此窗口接收 WINDOW_GAINED_FOCUS 或 WINDOW_ACTIVATED 事件之前，开发人员不得假定此窗口是焦点窗口或活动窗口。在顶层窗口是焦点窗口的平台上，此方法**可能**使此窗口成为焦点窗口（如果它还不是焦点窗口）。在堆栈顺序通常不影响焦点窗口的平台上，此方法**可能**维持焦点窗口和活动窗口不变。

如果此方法导致此窗口成为焦点窗口，而且此窗口是一个 Frame 或 Dialog，则它也将被激活。如果此窗口是焦点窗口，但它不是一个 Frame 或 Dialog，则拥有此窗口的第一个 Frame 或 Dialog 将被激活。

如果此窗口被模式对话框 (modal dialog) 阻塞，则阻塞对话框将置于最前端，仍然处于被阻塞窗口的前方。

另请参见：
toBack()

---

toBack

```
public void toBack()
```

如果此窗口是可视的，则将此窗口置于后方，如果它是焦点窗口或活动窗口，则会导致丢失焦点或活动状态。

在此虚拟机中，将此窗口放在堆栈顺序的底部，并在其他所有窗口之后显示此窗口。如果此窗口不可见，则不发生任何操作。有些平台不允许其他窗口拥有的窗口出现在其所有者下方。将进行所有尝试来移动此窗口，使其位于堆栈顺序中尽可能靠后的位置；不过，开发人员不应假定此方法在所有情况下都可以将此窗口移到所有其他窗口之下。

由于本机窗口系统多种多样，因此无法保证对焦点窗口和活动窗口的更改能够实现。在此窗口接收 WINDOW_LOST_FOCUS 或 WINDOW_DEACTIVATED 事件之前，开发人员不得假定此窗口不再是焦点窗口或活动窗口。在顶层窗口为焦点窗口的平台上，此方法**可能**导致此窗口不再是焦点状态。在此情况下，此虚拟机中紧跟其后的可作为焦点的窗口将成为焦点窗口。在堆栈顺序通常不影响焦点窗口的平台上，此方法**可能**维持焦点窗口和活动窗口不变。

另请参见：
toFront()

---

getToolkit

public Toolkit getToolkit()

返回此窗体的工具包。

覆盖：

类 Component 中的 getToolkit

返回：

此窗口的工具包。

另请参见：

Toolkit,Toolkit.getDefaultToolkit(),Component.getToolkit()

getWarningString

public final String getWarningString()

获取此窗口中显示的警告字符串。如果此窗口不安全，则警告字符串将在此窗口的可视区域内显示。如果存在安全管理器，并且在窗口作为一个参数传递到安全管理器的 checkTopLevelWindow 方法时，该方法返回 false，那么窗口是不安全的。

如果此窗口是安全的，则 getWarningString 返回 null。如果此窗口是不安全的，则此方法检查系统属性 awt.appletWarning，并返回此属性的字符串值。

返回：

此窗口的警告字符串。

另请参见：

SecurityManager.checkTopLevelWindow(java.lang.Object)

getLocale

public Locale getLocale()

如果设置了区域，则获取与此窗口关联的 Locale 对象。如果未设置区域，则返回默认的区域。

覆盖：

类 Component 中的 getLocale

返回：

为此窗口设置的区域。

从以下版本开始：
JDK1.1
另请参见：
Locale

getInputContext

```
public InputContext getInputContext()
```

获取此窗口的输入上下文。窗口始终具有一个输入上下文，如果子组件没有创建并设置自己的输入上下文，则可以共享该输入上下文。
覆盖：
类 Component 中的 getInputContext
返回：
组件使用的输入上下文，如果可以确定没有上下文，则返回 null
从以下版本开始：
1.2
另请参见：
Component.getInputContext()

setCursor

```
public void setCursor(Cursor cursor)
```

设置指定光标的光标图像。

如果 Java 平台实现和/或本机系统不支持更改鼠标光标形状，那么此方法将没有任何视觉效果。

覆盖：
类 Component 中的 setCursor
参数：
cursor - 由 Cursor 类定义的常量之一。如果此参数为 null，则此窗口的光标将被设置为类型 Cursor.DEFAULT_CURSOR。
从以下版本开始：
JDK1.1

另请参见：
Component.getCursor(),Cursor

getOwner

```
public Window getOwner()
```

返回此窗口的所有者。
从以下版本开始：
1.2

getOwnedWindows

```
public Window[] getOwnedWindows()
```

返回包含此窗口当前拥有的所有窗口的数组。
从以下版本开始：
1.2

getWindows

```
public static Window[] getWindows()
```

返回一个数组，该数组由此应用程序创建的所有 Window（包括被拥有的和不被拥有的）组成。如果从 applet 调用此方法，则返回数组只包括该 applet 可访问的 Window。

警告： 此方法可以返回系统创建的窗口，如打印对话框。应用程序不应该假定这些对话框存在，也不应该假定任何与这些对话框有关的内容（如组件位置、LayoutManager 或序列化）存在。

从以下版本开始：
1.6
另请参见：
Frame.getFrames(),getOwnerlessWindows()

getOwnerlessWindows

public static Window[] getOwnerlessWindows()

返回一个数组，该数组由此应用程序创建的所有没有所有者的 Window 组成。它们包括 Frame 以及没有所有者的 Dialog 和 Window。如果从 applet 调用此方法，则返回数组只包括该 applet 可访问的 Window。

警告： 此方法可以返回系统创建的窗口，如打印对话框。应用程序不应该假定这些对话框存在，也不应该假定任何与这些对话框有关的内容（如组件位置、LayoutManager 或序列化）存在。

从以下版本开始：

1.6

另请参见：

Frame.getFrames(),getWindows(sun.awt.AppContext)

setModalExclusionType

public void setModalExclusionType(Dialog.ModalExclusionType exclusionType)

指定此窗口的模式排斥类型。如果一个窗口是模式排斥的，则该窗口不会被某些模式对话框阻塞。有关可能的模式排斥类型，请参阅 Dialog.ModalExclusionType。

如果不支持给定类型，则使用 NO_EXCLUDE。

注：如果将可见窗口更改为模式排斥类型，则只有在该窗口被隐藏然后再次出现后才有效。

参数：

exclusionType - 此窗口的模式排斥类型；null 值等价于 NO_EXCLUDE

抛出：

SecurityException - 如果调用线程没有权限使用给定 exclusionType 设置窗口的模式排斥属性

从以下版本开始：

1.6

另请参见：

Dialog.ModalExclusionType, getModalExclusionType(),

Toolkit.isModalExclusionTypeSupported(java.awt.Dialog.ModalExclusionType)

getModalExclusionType

public Dialog.ModalExclusionType getModalExclusionType()

返回此窗口的模式排斥类型。

返回：

此窗口的模式排斥类型

从以下版本开始：

1.6

另请参见：

Dialog.ModalExclusionType,
setModalExclusionType(java.awt.Dialog.ModalExclusionType)

addWindowListener

public void addWindowListener(WindowListener l)

添加指定的窗口侦听器，以从此窗口接收窗口事件。如果 l 为 null，则不抛出任何异常，且不执行任何操作。

有关 AWT 线程模式的详细信息，请参考 AWT Threading Issues。

参数：

l - 窗口侦听器

另请参见：

removeWindowListener(java.awt.event.WindowListener),
getWindowListeners()

addWindowStateListener

public void addWindowStateListener(WindowStateListener l)

添加指定的窗口状态侦听器，以从此窗口接收窗口事件。如果 l 为 null，则不抛出任何异常，且不执行任何操作。

有关 AWT 线程模式的详细信息，请参考 AWT Threading Issues。

参数：

l - 窗口状态侦听器

从以下版本开始：

1.4

另请参见：

removeWindowStateListener(java.awt.event.WindowStateListener), getWindowStateListeners()

---

addWindowFocusListener

public void addWindowFocusListener(WindowFocusListener l)

添加指定的窗口焦点侦听器，以从此窗口接收窗口事件。如果 l 为 null，则不抛出任何异常，且不执行任何操作。

有关 AWT 线程模式的详细信息，请参考 AWT Threading Issues。

参数：

l - 窗口焦点侦听器

从以下版本开始：

1.4

另请参见：

removeWindowFocusListener(java.awt.event.WindowFocusListener), getWindowFocusListeners()

---

removeWindowListener

public void removeWindowListener(WindowListener l)

移除指定的窗口侦听器，以便不再从此窗口接收窗口事件。如果 l 为 null，则不抛出任何异常，且不执行任何操作。

有关 AWT 线程模式的详细信息，请参考 AWT Threading Issues。

参数：
l - 窗口侦听器
另请参见：
addWindowListener(java.awt.event.WindowListener),
getWindowListeners()

removeWindowStateListener

public void removeWindowStateListener(WindowStateListener l)

移除指定的窗口状态侦听器，以便不再从此窗口接收窗口事件。如果 l 为 null，则不抛出任何异常，且不执行任何操作。

有关 AWT 线程模式的详细信息，请参考 AWT Threading Issues。

参数：
l - 窗口状态侦听器
从以下版本开始：
1.4
另请参见：
addWindowStateListener(java.awt.event.WindowStateListener),
getWindowStateListeners()

removeWindowFocusListener

public void removeWindowFocusListener(WindowFocusListener l)

移除指定的窗口焦点侦听器，以便不再从此窗口接收窗口事件。如果 l 为 null，则不抛出任何异常，且不执行任何操作。

有关 AWT 线程模式的详细信息，请参考 AWT Threading Issues。

参数：
l - 窗口焦点侦听器
从以下版本开始：

1.4

另请参见：

addWindowFocusListener(java.awt.event.WindowFocusListener), getWindowFocusListeners()

getWindowListeners

```
public WindowListener[] getWindowListeners()
```

返回在此窗口注册的所有窗口侦听器所组成的数组。

返回：

此窗口的所有 WindowListener；如果当前未注册窗口侦听器，则返回一个空数组

从以下版本开始：

1.4

另请参见：

addWindowListener(java.awt.event.WindowListener), removeWindowListener(java.awt.event.WindowListener)

getWindowFocusListeners

```
public WindowFocusListener[] getWindowFocusListeners()
```

返回在此窗口注册的所有窗口焦点侦听器所组成的数组。

返回：

此窗口的所有 WindowFocusListener；如果当前未注册窗口焦点侦听器，则返回一个空数组

从以下版本开始：

1.4

另请参见：

addWindowFocusListener(java.awt.event.WindowFocusListener), removeWindowFocusListener(java.awt.event.WindowFocusListener)

getWindowStateListeners

public WindowStateListener[] getWindowStateListeners()

返回在此窗口注册的所有窗口状态侦听器所组成的数组。

返回：

此窗口的所有 WindowStateListener；如果当前未注册窗口状态侦听器，则返回一个空数组

从以下版本开始：

1.4

另请参见：

addWindowStateListener(java.awt.event.WindowStateListener), removeWindowStateListener(java.awt.event.WindowStateListener)

getListeners

public <T extends EventListener> T[] getListeners(Class<T> listenerType)

返回当前在此 Window 上注册为 *Foo*Listener 的所有对象所组成的数组。*Foo*Listener 是使用 add*Foo*Listener 方法注册的。

可以使用类文本指定 listenerType 参数，如 *Foo*Listener.class。例如，可以使用以下代码查询 Window w，获取其窗口侦听器：

```
WindowListener[] wls =
(WindowListener[])(w.getListeners(WindowListener.clas
s));
```

如果不存在此类侦听器，则此方法将返回一个空数组。

覆盖：

类 Container 中的 getListeners

参数：

listenerType - 请求的侦听器的类型；此参数应指定一个从 java.util.EventListener 继承的接口

返回：

在此窗口注册为 *Foo*Listener 的所有对象所组成的数组；如果未添加此类侦听器，则返回一个空数组

抛出：

ClassCastException - 如果 listenerType 未指定一个实现 java.util.EventListener 的类或接口

从以下版本开始：

1.3

另请参见：
getWindowListeners()

processEvent

protected void processEvent(AWTEvent e)

处理此窗口上的事件。如果此事件为一个 WindowEvent，它将调用 processWindowEvent 方法，否则将调用其超类的 processEvent。

注意，如果此事件参数为 null，则此行为是不明确的，可能导致异常。

覆盖：
类 Container 中的 processEvent
参数：
e - 事件
另请参见：
Component.processComponentEvent(java.awt.event.ComponentEvent),
Component.processFocusEvent(java.awt.event.FocusEvent),
Component.processKeyEvent(java.awt.event.KeyEvent),
Component.processMouseEvent(java.awt.event.MouseEvent),
Component.processMouseMotionEvent(java.awt.event.MouseEvent),
Component.processInputMethodEvent(java.awt.event.InputMethodEvent),
Component.processHierarchyEvent(java.awt.event.HierarchyEvent),
Component.processMouseWheelEvent(java.awt.event.MouseWheelEvent)

processWindowEvent

protected void processWindowEvent(WindowEvent e)

处理此窗口上发生的窗口事件：将它们指派到任何注册的 WindowListener 对象。注：只有为此组件启用了窗口事件时，此方法才有可能被调用；这在存在以下条件之一时发生：

- 通过 addWindowListener 注册 WindowListener 对象
- 通过 enableEvents 启用窗口事件

注意，如果此事件参数为 null，则此行为是不明确的，可能导致异常。

参数：
e - 窗口事件
另请参见：
Component.enableEvents(long)

---

processWindowFocusEvent

protected void processWindowFocusEvent(WindowEvent e)

处理此窗口上发生的窗口焦点事件：将它们指派到任何注册的 WindowFocusListener 对象。注：只有为此组件启用了窗口事件时，此方法才有可能被调用。这在存在以下条件之一时发生：

- 通过 addWindowFocusListener 注册 WindowFocusListener
- 通过 enableEvents 启用窗口焦点事件

注意，如果此事件参数为 null，则此行为是不明确的，可能导致异常。

参数：
e - 窗口焦点事件
从以下版本开始：
1.4
另请参见：
Component.enableEvents(long)

---

processWindowStateEvent

protected void processWindowStateEvent(WindowEvent e)

处理此窗口上发生的窗口状态事件：将它们指派到任何注册的 WindowStateListener 对象。注：只有为此组件启用了窗口事件时，此方法才有可能被调用。这在存在以下条件之一时发生：

- 通过 addWindowStateListener 注册 WindowStateListener
- 通过 enableEvents 启用窗口状态事件

注意，如果此事件参数为 null，则此行为是不明确的，可能导致异常。

参数：
e - 窗口状态事件
从以下版本开始：
1.4
另请参见：
Component.enableEvents(long)

setAlwaysOnTop

```
public final void setAlwaysOnTop(boolean alwaysOnTop)
                                 throws SecurityException
```

设置此窗口是否应该始终位于其他窗口上方。如果存在多个 always-on-top 窗口，则它们的相对顺序没有指定，该顺序与平台有关。

如果某个其他窗口已经是 always-on-top，则不指定这些窗口之间的相互顺序（与平台有关）。任何窗口都不会置于 always-on-top 窗口之上，除非它也是一个 always-on-top 窗口。

所有 always-on-top 窗口拥有的窗口将继承此状态，自动成为 always-on-top 窗口。如果某个窗口不再是 always-on-top 窗口，则它所拥有的窗口也不再是 always-on-top 窗口。当 always-on-top 窗口调用 toBack 时，其 always-on-top 状态将设置为 false。

如果对某个窗口调用此方法且参数值为 true，并且该窗口可见，平台也支持此窗口为 always-on-top，那么该窗口将立即向前，“锁定”在最顶层位置。如果窗口当前不可见，那么此方法将 always-on-top 状态设置为 true，但窗口不会向前。如果以后显示该窗口，那么它将 always-on-top。

如果 alwaysOnTop 为 true，则此方法可使窗口 always-on-top。如果窗口是可见的（这包括将窗口前置，即 toFront），则将其“锁定”在最顶层的位置。如果窗口不可见，则除了设置 always-on-top 的属性之外，

不执行任何操作。如果以后窗口显示出来，则该窗口将 always-on-top。如果窗口已经位于顶层，则此调用不执行任何操作。

如果对某个窗口调用此方法且参数值为 false，则 always-on-top 状态被设置为常规。该窗口保持在最顶层，但对于任何其他窗口来说，其 z-order 可以更改。对已经是常规状态的窗口调用此方法且参数值为 false 将无效。如果没有其他 always-on-top 窗口，将 always-on-top 状态设置为 false 对窗口的相对 z-order 没有影响。

注：有些平台可能不支持 always-on-top 窗口。要检测当前平台是否支持 always-on-top 窗口，请使用 Toolkit.isAlwaysOnTopSupported() 和 isAlwaysOnTopSupported()。如果工具包或此窗口不支持 always-on-top 模式，则调用此方法无效。

如果安装了 SecurityManager，则必须授予调用线程 AWTPermission "setWindowAlwaysOnTop" 权限，才能设置此属性值。如果未授予此权限，则此方法将抛出 SecurityException，并且属性的当前值保持不变。

参数：

alwaysOnTop - 如果窗口应该始终位于其他窗口上方，则该参数为 true

抛出：

SecurityException - 如果调用线程无权设置 always-on-top 属性值。

从以下版本开始：

1.5

另请参见：

isAlwaysOnTop(), toFront(), toBack(), AWTPermission, isAlwaysOnTopSupported(),Toolkit.isAlwaysOnTopSupported()

isAlwaysOnTopSupported

```
public boolean isAlwaysOnTopSupported()
```

返回此窗口是否支持 always-on-top 模式。某些平台不支持 always-on-top 窗口，而另一些只支持某种 top-level（顶层）窗口；例如，某个平台不支持 always-on-top 模式对话框。

返回：

如果工具包和此窗口支持 always-on-top 模式，则返回 true；如果此窗口不支持 always-on-top 模式，或者工具包不支持 always-on-top 窗口，则返回 false。

从以下版本开始：

1.6

另请参见：

setAlwaysOnTop(boolean),Toolkit.isAlwaysOnTopSupported()

isAlwaysOnTop

public final boolean isAlwaysOnTop()

返回此窗口是否为 always-on-top 窗口。

返回：

如果此窗口处于 always-on-top 状态，则返回 true，否则，返回 false

从以下版本开始：

1.5

另请参见：

setAlwaysOnTop(boolean)

getFocusOwner

public Component getFocusOwner()

如果此窗口为焦点窗口，则返回是焦点窗口的子组件；否则返回 null。

返回：

具有焦点的子组件；如果此窗口不是焦点窗口，则返回 null

另请参见：

getMostRecentFocusOwner(),isFocused()

getMostRecentFocusOwner

public Component getMostRecentFocusOwner()

返回此窗口的子组件，该子组件在此窗口为焦点窗口时将接收焦点。如果此窗口当前为焦点窗口，则此方法将返回与 getFocusOwner() 相同的组件。如果此窗口不是焦点窗口，则返回最近请求焦点的子组件。如果没有子组件请求过焦点，并且这是一个可成为焦点的窗口，则返回此窗口最初可成为焦点的组件。如果没有子组件请求过

焦点，并且这是一个不能成为焦点的窗口，则返回 null。

返回：

在此窗口成为焦点窗口时，将接收焦点的子组件

从以下版本开始：

1.4

另请参见：

getFocusOwner(),isFocused(),isFocusableWindow()

isActive

```
public boolean isActive()
```

返回此窗口是否为活动窗口。仅有一个 Frame 或 Dialog 可以处于活动状态。本机窗口系统表示具有特殊修饰的活动窗口或其子窗口，如高亮的标题栏。活动窗口始终是焦点窗口，或者是拥有该焦点窗口的第一个 Frame 或 Dialog。

返回：

此窗口是否为活动窗口。

从以下版本开始：

1.4

另请参见：

isFocused()

isFocused

```
public boolean isFocused()
```

返回此窗口是否为焦点窗口。如果存在焦点所有者，则焦点窗口就是（或者包含）焦点所有者的窗口。如果不存在焦点所有者，则没有作为焦点的窗口。

如果焦点窗口是一个 Frame 或 Dialog，那么它同时也是一个活动窗口。否则，活动窗口将是拥有焦点窗口的第一个 Frame 或 Dialog。

返回：

此窗口是否为焦点窗口。

从以下版本开始：

1.4

另请参见：
isActive()

getFocusTraversalKeys

public Set<AWTKeyStroke> getFocusTraversalKeys(int id)

获取此窗口的焦点遍历键。（有关每个键的完整描述，请参阅 setFocusTraversalKeys。）

如果未为此窗口显式设置遍历键，则返回此窗口的父窗口的遍历键。如果未为此窗口的任何祖先显式设置遍历键，则返回当前 KeyboardFocusManager 的默认遍历键。

覆盖：
类 Container 中的 getFocusTraversalKeys
参数：
id - KeyboardFocusManager.FORWARD_TRAVERSAL_KEYS、KeyboardFocusManager.BACKWARD_TRAVERSAL_KEYS、KeyboardFocusManager.UP_CYCLE_TRAVERSAL_KEYS 或 KeyboardFocusManager.DOWN_CYCLE_TRAVERSAL_KEYS 之一
返回：
指定键的 AWTKeyStroke
抛出：
IllegalArgumentException - 如果 id 不是 KeyboardFocusManager.FORWARD_TRAVERSAL_KEYS、KeyboardFocusManager.BACKWARD_TRAVERSAL_KEYS、KeyboardFocusManager.UP_CYCLE_TRAVERSAL_KEYS 或 KeyboardFocusManager.DOWN_CYCLE_TRAVERSAL_KEYS 之一
从以下版本开始：
1.4
另请参见：
Container.setFocusTraversalKeys(int, java.util.Set), KeyboardFocusManager.FORWARD_TRAVERSAL_KEYS, KeyboardFocusManager.BACKWARD_TRAVERSAL_KEYS, KeyboardFocusManager.UP_CYCLE_TRAVERSAL_KEYS, KeyboardFocusManager.DOWN_CYCLE_TRAVERSAL_KEYS

setFocusCycleRoot

public final void setFocusCycleRoot(boolean focusCycleRoot)

不执行任何操作，因为窗口必须始终是焦点遍历循环的根。忽略传入的值。

覆盖：

类 Container 中的 setFocusCycleRoot

参数：

focusCycleRoot - 忽略此值

从以下版本开始：

1.4

另请参见：

isFocusCycleRoot(), Container.setFocusTraversalPolicy(java.awt.FocusTraversalPolicy),Container.getFocusTraversalPolicy()

isFocusCycleRoot

public final boolean isFocusCycleRoot()

始终返回 true，因为所有窗口必须是焦点遍历循环的根。

覆盖：

类 Container 中的 isFocusCycleRoot

返回：

true

从以下版本开始：

1.4

另请参见：

setFocusCycleRoot(boolean), Container.setFocusTraversalPolicy(java.awt.FocusTraversalPolicy),Container.getFocusTraversalPolicy()

getFocusCycleRootAncestor

public final Container getFocusCycleRootAncestor()

始终返回 null，因为窗口没有祖先；它们表示组件层次结构的顶层。

覆盖：

类 Component 中的 getFocusCycleRootAncestor

返回：

null

从以下版本开始：

1.4

另请参见：

Container.isFocusCycleRoot()

isFocusableWindow

public final boolean isFocusableWindow()

返回此窗口是否可以成为焦点窗口，也就是说，此窗口或其任何子组件是否可以成为焦点所有者。对于可成为焦点的 Frame 或 Dialog，其可成为焦点的窗口状态必须设置为 true。对于不可成为焦点的 Frame 或 Dialog 的窗口，其可成为焦点的窗口状态必须设置为 true，其最近拥有的 Frame 或 Dialog 必须显示在屏幕上，而且它必须至少包含其焦点遍历循环中的一个组件。如果未满足这些条件中的任何一个条件，则此窗口及其任何子组件都不能成为焦点所有者。

返回：

如果此窗口可以成为焦点窗口，则返回 true；否则返回 false

从以下版本开始：

1.4

另请参见：

getFocusableWindowState(), setFocusableWindowState(boolean), isShowing(),Component.isFocusable()

getFocusableWindowState

public boolean getFocusableWindowState()

返回如果此窗口满足 isFocusableWindow 中列出的其他要求，其是否可以成为焦点窗口。如果此方法返回 false，则 isFocusableWindow 也将返回 false。如果此方法返回 true，则 isFocusableWindow 可能返回 true，也可能返回 false，具体取决于窗口要成为焦点窗口所必须满足的其他要求。

默认情况下，所有窗口都具有 focusable 的窗口状态 true。

返回：
此窗口是否可成为焦点窗口
从以下版本开始：
1.4
另请参见：
isFocusableWindow(),setFocusableWindowState(boolean),isShowing(),Component.setFocusable(boolean)

setFocusableWindowState

public void setFocusableWindowState(boolean focusableWindowState)

设置如果此窗口满足 isFocusableWindow 中列出的其他要求，其是否可以成为焦点窗口。如果此窗口可成为焦点窗口状态设置为 false，则 isFocusableWindow 将返回 false。如果此窗口的可成为焦点的窗口状态设置为 true，则 isFocusableWindow 可能返回 true，也可能返回 false，具体取决于要成为焦点的窗口所必须满足的其他要求。

将窗口的焦点状态设置为 false 是 AWT 标识应用程序的标准机制，AWT 是可用作浮动调色板或工具栏的窗口，因此应为是不可为焦点的窗口。在可见的 Window 上设置焦点状态，可能在某些平台上产生延迟的结果 — 只有在 Window 成为隐藏窗口，然后再可见后才发生实际的更改。为确保行为跨平台的一致性，当 Window 不可见时设置 Window 的焦点状态，然后再显示它。

参数：
focusableWindowState - 此窗口是否可以成为焦点窗口
从以下版本开始：
1.4
另请参见：
isFocusableWindow(), getFocusableWindowState(), isShowing(),Component.setFocusable(boolean)

addPropertyChangeListener

public void addPropertyChangeListener(PropertyChangeListener listener)

将 PropertyChangeListener 添加到侦听器列表。为此类的所有绑定 (bound) 属性注册侦听器，包括：

- 此窗口的字体 ("font")
- 此窗口的背景色 ("background")
- 此窗口的前景色 ("foreground")
- 此窗口的可聚焦性 ("focusable")
- 此窗口的焦点遍历键启用状态 ("focusTraversalKeysEnabled")
- 此窗口的 FORWARD_TRAVERSAL_KEYS 集 ("forwardFocusTraversalKeys")
- 此窗口的 BACKWARD_TRAVERSAL_KEYS 集 ("backwardFocusTraversalKeys")
- 此窗口的 UP_CYCLE_TRAVERSAL_KEYS 集 ("upCycleFocusTraversalKeys")
- 此窗口的 DOWN_CYCLE_TRAVERSAL_KEYS 集 ("downCycleFocusTraversalKeys")
- 此窗口的焦点遍历策略 ("focusTraversalPolicy")
- 此窗口的可成为焦点的窗口状态 ("focusableWindowState")
- 此窗口的 always-on-top 状态 ("alwaysOnTop")

注意，如果此窗口在继承一个绑定属性，则不触发任何事件来响应继承属性中的更改。

如果侦听器为 null，则不抛出任何异常，且不执行任何操作。

覆盖：

类 Container 中的 addPropertyChangeListener

参数：

listener - 要添加的 PropertyChangeListener

另请参见：

Component.removePropertyChangeListener(java.beans.PropertyChangeListener), addPropertyChangeListener(java.lang.String, java.beans.PropertyChangeListener)

addPropertyChangeListener

public void addPropertyChangeListener(String propertyName, PropertyChangeListener listener)

将 PropertyChangeListener 添加到某个特定属性的侦听器列表。指定的属性可以是用户定义的，也可以是下列属性之一：

- 此窗口的字体 ("font")
- 此窗口的背景色 ("background")
- 此窗口的前景色 ("foreground")
- 此窗口的可聚焦性 ("focusable")
- 此窗口的焦点遍历键启用状态 ("focusTraversalKeysEnabled")
- 此窗口的 FORWARD_TRAVERSAL_KEYS 集 ("forwardFocusTraversalKeys")
- 此窗口的 BACKWARD_TRAVERSAL_KEYS 集 ("backwardFocusTraversalKeys")
- 此窗口的 UP_CYCLE_TRAVERSAL_KEYS 集 ("upCycleFocusTraversalKeys")
- 此窗口的 DOWN_CYCLE_TRAVERSAL_KEYS 集 ("downCycleFocusTraversalKeys")
- 此窗口的焦点遍历策略 ("focusTraversalPolicy")
- 此窗口的可成为焦点的窗口状态 ("focusableWindowState")
- 此窗口的 always-on-top 状态 ("alwaysOnTop")

注意，如果此窗口在继承一个绑定属性，则不触发任何事件来响应继承属性中的更改。

如果侦听器为 null，则不抛出任何异常，且不执行任何操作。

覆盖：
类 Container 中的 addPropertyChangeListener
参数：
propertyName - 上文列出的属性名之一
listener - 要添加的 PropertyChangeListener
另请参见：
addPropertyChangeListener(java.beans.PropertyChangeListener), Component.removePropertyChangeListener(java.beans.PropertyChangeListener)

---

postEvent

```
@Deprecated
    public boolean postEvent(Event e)
```

已过时。 *从 JDK version 1.1 开始，由 dispatchEvent(AWTEvent) 取代。*

指定者：
接口 MenuContainer 中的 postEvent
覆盖：
类 Component 中的 postEvent

isShowing

```
public boolean isShowing()
```

检查此窗口是否显示在屏幕上。
覆盖：
类 Component 中的 isShowing
返回：
如果正在显示组件，则返回 true；否则返回 false
另请参见：
Component.setVisible(boolean)

applyResourceBundle

```
@Deprecated
public void applyResourceBundle(ResourceBundle rb)
```

已过时。从 J2SE 1.4 开始，由 Component.applyComponentOrientation 取代。

applyResourceBundle

```
@Deprecated
public void applyResourceBundle(String rbName)
```

已过时。从 J2SE 1.4 开始，由 Component.applyComponentOrientation 取代。

getAccessibleContext

public AccessibleContext getAccessibleContext()

获取与此窗口关联的 AccessibleContext。对于窗口，AccessibleContext 采取 AccessibleAWTWindow 的形式。如有必要，创建新的 AccessibleAWTWindow 实例。

指定者：

接口 Accessible 中的 getAccessibleContext

覆盖：

类 Component 中的 getAccessibleContext

返回：

充当此窗口的 AccessibleContext 的 AccessibleAWTWindow

从以下版本开始：

1.3

getGraphicsConfiguration

public GraphicsConfiguration getGraphicsConfiguration()

此方法返回由此窗口使用的 GraphicsConfiguration。

覆盖：

类 Component 中的 getGraphicsConfiguration

返回：

此 Component 使用的 GraphicsConfiguration，或者返回 null

从以下版本开始：

1.3

setLocationRelativeTo

public void setLocationRelativeTo(Component c)

设置窗口相对于指定组件的位置。

如果组件当前未显示，或者 c 为 null，则此窗口将置于屏幕的中央。中点可以使用 GraphicsEnvironment.getCenterPoint 确定。

如果该组件的底部在屏幕外，则将该窗口放置在 Component 最接近窗口中心的一侧。因此，如果 Component 在屏幕的右部，则 Window 将被放置在左部，反之亦然。

参数：
c - 确定窗口位置涉及的组件
从以下版本开始：
1.4
另请参见：
GraphicsEnvironment.getCenterPoint()

createBufferStrategy

```
public void createBufferStrategy(int numBuffers)
```

为此组件上的多缓冲创建一个新策略。多缓冲对于呈现性能非常有用。此方法试图根据提供的缓冲区数创建可用的最佳策略。它将始终根据该缓冲区数创建 BufferStrategy。首先尝试 page-flipping 策略，然后使用加速缓冲区尝试 blitting 策略。最后使用不加速的 blitting 策略。

每次调用此方法时，将丢弃此组件的现有缓冲策略。

参数：
numBuffers - 要创建的缓冲区数
抛出：
IllegalArgumentException - 如果 numBuffers 小于 1。
IllegalStateException - 如果组件不可显示
从以下版本开始：
1.4
另请参见：
Component.isDisplayable(),getBufferStrategy()

createBufferStrategy

```
public void createBufferStrategy(int numBuffers,
                                  BufferCapabilities caps)
                           throws AWTException
```

根据所需缓冲区能力为此组件上的多缓冲创建新策略。在只需要加速的内存或页面翻转（由缓冲区能力指定）时，这很有用。

每次调用此方法时，将丢弃此组件的现有缓冲策略。

参数：

numBuffers - 要创建的缓冲区数，包括前缓冲区

caps - 创建缓冲策略所需的能力；不能为 null

抛出：

AWTException - 如果提供的能力不受支持或未得到满足；这是有可能发生的，例如，如果当前无足够的可用加速内存，或者指定了页翻转，但不可能实现。

IllegalArgumentException - 如果 numBuffers 小于 1，或者 caps 为 null

从以下版本开始：

1.4

另请参见：

getBufferStrategy()

---

getBufferStrategy

public BufferStrategy getBufferStrategy()

返回此组件使用的 BufferStrategy。如果尚未创建或已经释放了 BufferStrategy，那么此方法将返回 null。

返回：

此组件使用的缓冲策略

从以下版本开始：

1.4

另请参见：

createBufferStrategy(int)

---

setLocationByPlatform

public void setLocationByPlatform(boolean locationByPlatform)

设置窗口下次可见时应该出现的位置：本机窗口系统的默认位置，还是当前位置（由 getLocation 返回）。此行为模拟显示的本机窗口，而不是以编程方式设置其位置。如果未显式设置窗口的位置，那么大多数窗口操作系统将重叠显示窗口。一旦窗口显示在屏幕上，其实际位置就被确定。

还可以通过将系统属性 "java.awt.Window.locationByPlatform" 设置为 "true" 来启用此行为，但应优先考虑调用此方法。

在调用 setLocationByPlatform 清除窗口的此属性之后，调用 setVisible、setLocation 和 setBounds。

例如，在执行以下代码后：

```
setLocationByPlatform(true);
setVisible(true);
boolean flag = isLocationByPlatform();
```

窗口将显示在平台的默认位置，flag 将为 false。

在以下示例中：

```
setLocationByPlatform(true);
setLocation(10, 10);
boolean flag = isLocationByPlatform();
setVisible(true);
```

窗口将显示在 (10, 10) 的位置，flag 将为 false。

参数：

locationByPlatform - 如果此窗口应出现在默认位置，则为 true，如果应出现在当前位置，则为 false

抛出：

IllegalComponentStateException - 如果此窗口显示在屏幕上，且 locationByPlatform 为 true。

从以下版本开始：

1.5

另请参见：

Component.setLocation(int, int), isShowing(), setVisible(boolean), isLocationByPlatform(), System.getProperty(String)

isLocationByPlatform

```
public boolean isLocationByPlatform()
```

如果此窗口下次可见时，出现在本机窗口操作系统的默认位置，则返回 true。如果窗口显示在屏幕上，则此方法始终返回 false。

返回：

此窗口是否将出现在默认位置

从以下版本开始：

1.5

另请参见：

setLocationByPlatform(boolean),isShowing()

setBounds

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

移动组件并调整其大小。由 x 和 y 指定左上角的新位置，由 width 和 height 指定新的大小。

如果 width 值或 height 值小于之前调用 setMinimumSize 指定的最小大小，则它的值将自动增加。

覆盖：

类 Component 中的 setBounds

参数：

x - 组件的新 *x* 坐标

y - 组件的新 *y* 坐标

width - 组件的新 width

height - 组件的新 height

从以下版本开始：

1.6

另请参见：

Component.getBounds(), Component.setLocation(int, int), Component.setLocation(Point), setSize(int, int), setSize(Dimension), setMinimumSize(java.awt.Dimension), setLocationByPlatform(boolean),isLocationByPlatform()

setBounds

public void setBounds(Rectangle r)

移动组件并调整其大小，使其符合新的有界矩形 r。由 r.x 和 r.y 指定组件的新位置，由 r.width 和 r.height 指定组件的新大小

如果 r.width 值或 r.height 值小于之前调用 setMinimumSize 指定的最小大小，则它的值将自动增加。

覆盖：

类 Component 中的 setBounds

参数：

r - 此组件的新的有界矩形