PHP5 面向对象初步

Created by 刀客羽朋

前言 PHP5 面向对象设计

从 OOP 的视角看，不应区分语言。无论是 C++、无论是 Java、无论是.net 还有更多面向对象的语言，只要你了解了 OO 的真谛，便可以跨越语言，让你的思想轻松的跳跃。便没有对于 Java、.net 、 PHP 之间谁强谁弱的争执了。

希望这个介绍 PHP5 面向对象编程（OOP）的资料能让初学者受益，能让更多的 PHPer 开始转向 OO 的编程过程。

相对 PHP4，PHP5 在面向对象方面改变了很多。我们将只介绍 PHP5 环境下的面向对象。而我们必须改变自己来跟随 PHP5 的发展。如果代码结果在你的环境下执行不一致，请确认你的环境为 PHP5。

我们假设读者没有任何面向对象的知识，即使你是第一次听说 OOP，也可以读懂这文章。但我希望你必须对 PHP 有一些了解。

在后面我们将使用一些例子，来逐渐分析 PHP5 的 OO 基础。

面向对象只解决了两个问题，**代码的可扩展性 、 代码的可维护性。**

关于为什么要使用面向对象？在什么时候使用面向对象？这里不讨论。

感谢www.phpchina.com，编程软件环境使用的ZendStudio5.2。

不得不说句，php 越来越像 Java 了。

在本书之后，建议继续读www.PHPchina.com 翻译的 《PHP设计模式》。

仅以这些给我即将降生的 baby。

刀客羽朋
gaoxiangming@hotmail.com
2006-10-12

目录

第一章 PHP5 面向对象基础

1.1 类和对象

Everything is Object：万事万物皆对象。

面向对象的编程（OOP）思想力图使对计算机语言中对事物的描述与现实世界中该事物的本来面目尽可能的一致。

（面向对象语言与我们的生活是想通的，面向对象语言学习起来其实很简单。在应用中更符合我们的生活逻辑。）

类(Class)是用来描述一个对象(Object)：

类描述了每个对象应包括的数据

类描述了每个对象的行为特征

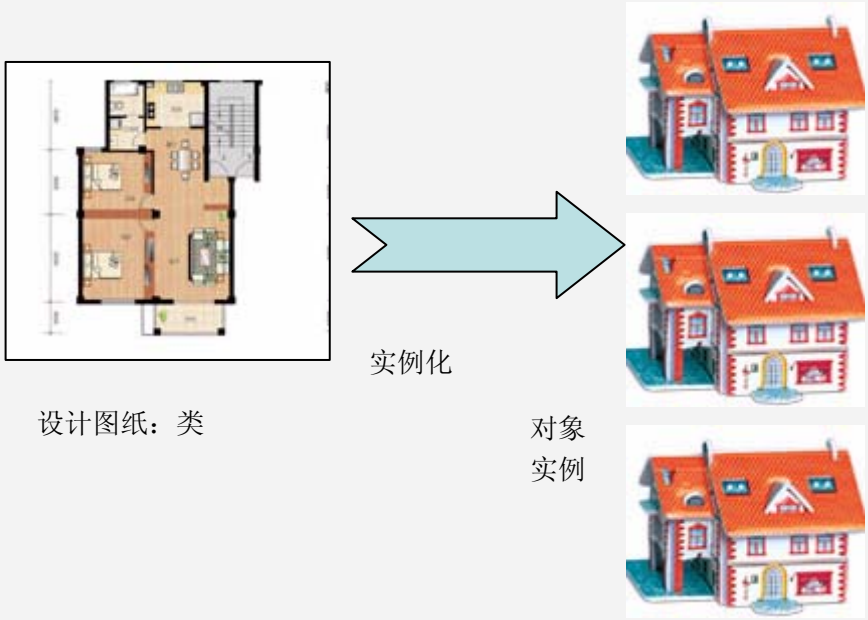
（如上所述，类仅仅是对象的描述，就好像楼房的设计图纸。）

Class/Object：类(class)和对象(object)是面向对象方法的核心概念。

类是对一类事物描述，是抽象的、概念上的定义；
（类好像是在图纸上设计的楼房，楼房设计出来了，但这个楼房并不存在。）

对象是实际存在的该类事物的每个个体，因而也称实例(instance)。
（对象是实实在在存在的，照着楼房的设计图纸，高楼盖起来，可以住进去了。在计算机中，可以理解为，在内存中创建了实实在在存在的一个内存区域存储着这个对象。）

创建对象的过程称为 创建对象 也称为实例化。
看下面的图示，一张楼房的图纸创建了多个别墅（对象）。
思考一下：
它们外观一样么？
它们结构一样么？
它们是一个对象么？

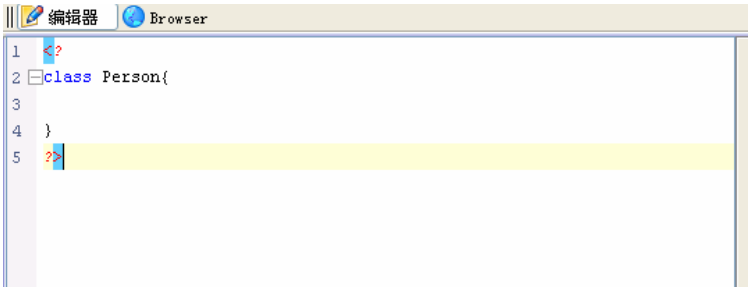


1.2 PHP5 中的类和对象

我们先建立一个基础的类。

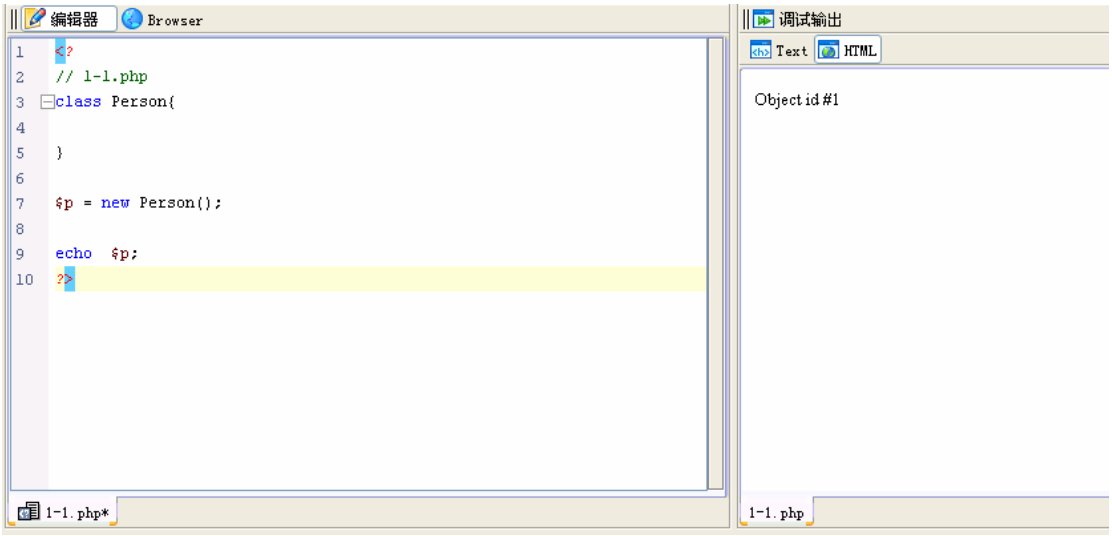
PHP 中使用关键字 **class** 来定义一个类。类的命名一般使用首字符大写，而后每个单词首字符大写连接的方式方便阅读。

例 1-1.PHP



这样，我们就拥有了第一个 PHP 类。

我们继续来使用这个类，使用 **new** 这个关键字创建对象，用 echo 打印$p。



我们定义了一个变量 $p ，使用 **new** 这个关键字创建了一个 Person 的对象。

打印变量$p，我们看到输出 Object id #1 提示这是一个对象。

> $p = new Person();也可以写成 $p = new Person;
> 但不建议使用后面的这种方式。

1.3 PHP5 中的属性

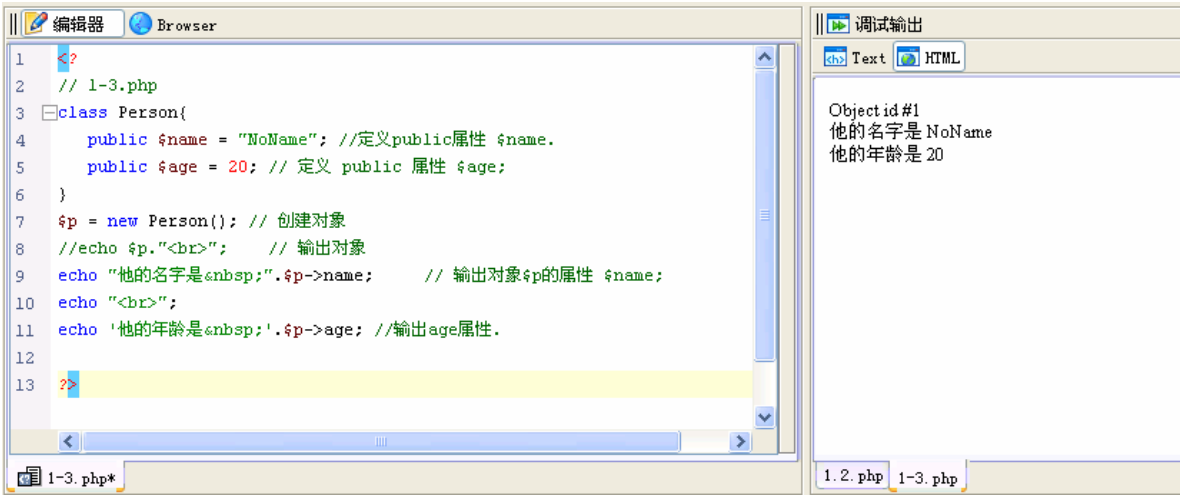
属性：用来描述对象的数据元素称为对象的属性（也称为数据/状态）

在 PHP5 中，属性指在 class 中声明的变量。在声明变量时，必须使用 public private protected 之一进行修饰，定义变量的访问权限。

- Public（公开）：可以自由的在类的内部外部读取、修改。
- Private（私有）：只能在这个当前类的内部读取、修改。
- Protected（受保护）：能够在这个类和类的子类中读取和修改。

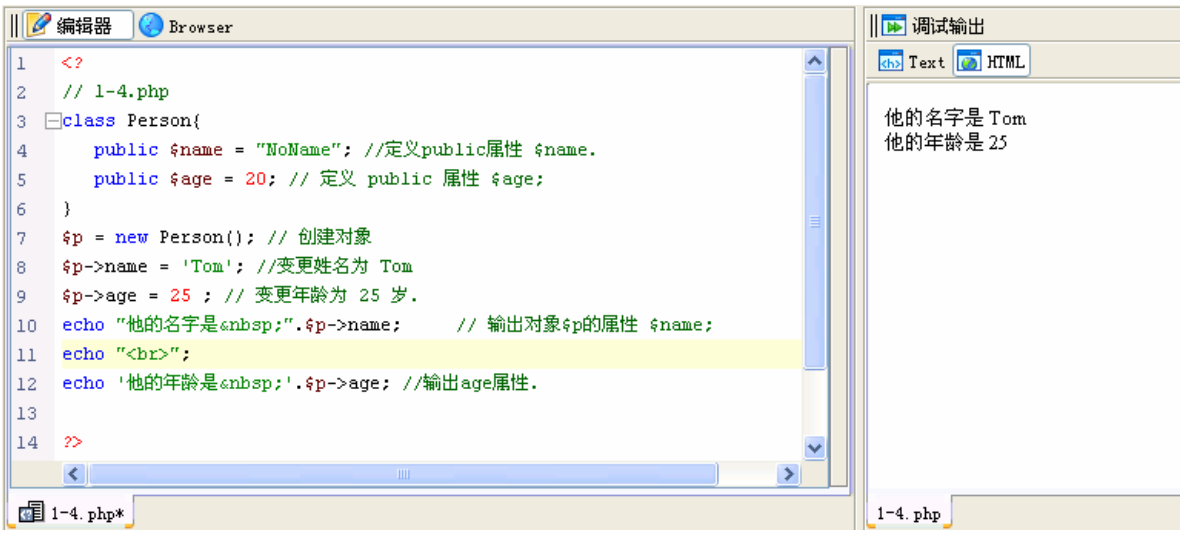
属性的使用：通过引用变量的 -> 符号调用变量指向对象的属性。
在方法内部通过 $this-> 符号调用同一对象的属性。

例 1-3：public 的属性。



Person 类有两个属性，$name 和 $age，在实例化后，使用$p->name 和 $p->age 打印出属性的内容。
当然，你可以在属性定义时不设置初始值，那样的话，就打印不出任何结果了。

例 1-4:
改变对象的属性，注意 8 行和 9 行代码，还有输出结果的变化。我们看到输出的属性值被改变了。

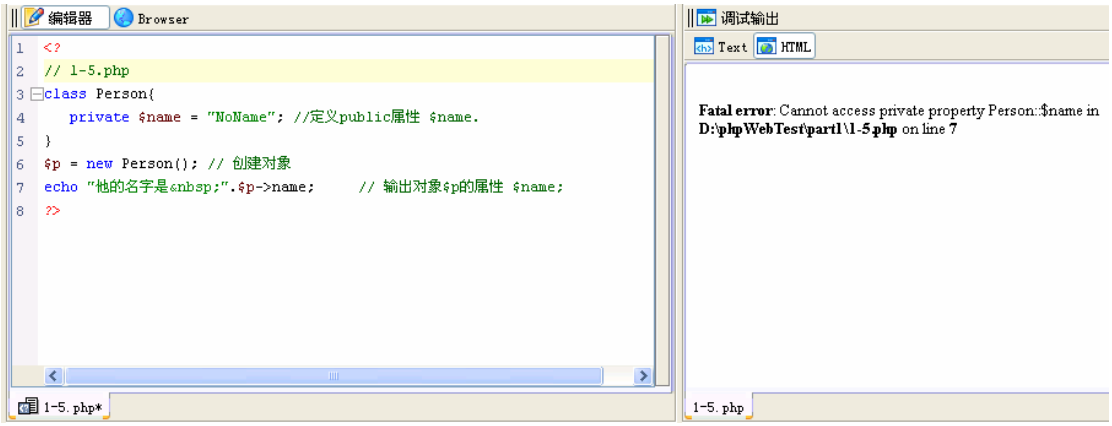


创建一个 Person 的对象，改变这个对象的属性。为它命名，查看它的名字。你就是机器里面这个 Person 对象的上帝，按照你定义的规则，这个实实在在在内存中的 Person 对象被创建了，而且它有了可以改变的属性。
现在，我们就是计算机世界的上帝，准备好创造世界吧。

Private 修饰的属性，在当前对象以外不能访问。设置私有属性是为了进行数据的隐藏。

隐藏：指对象的一种保护机制，使得它的属性或方法不被外部的程序直接访问。

例 1-5：访问时会报错如下。



私有属性不能被外部访问，在下一节介绍这样做的好处。

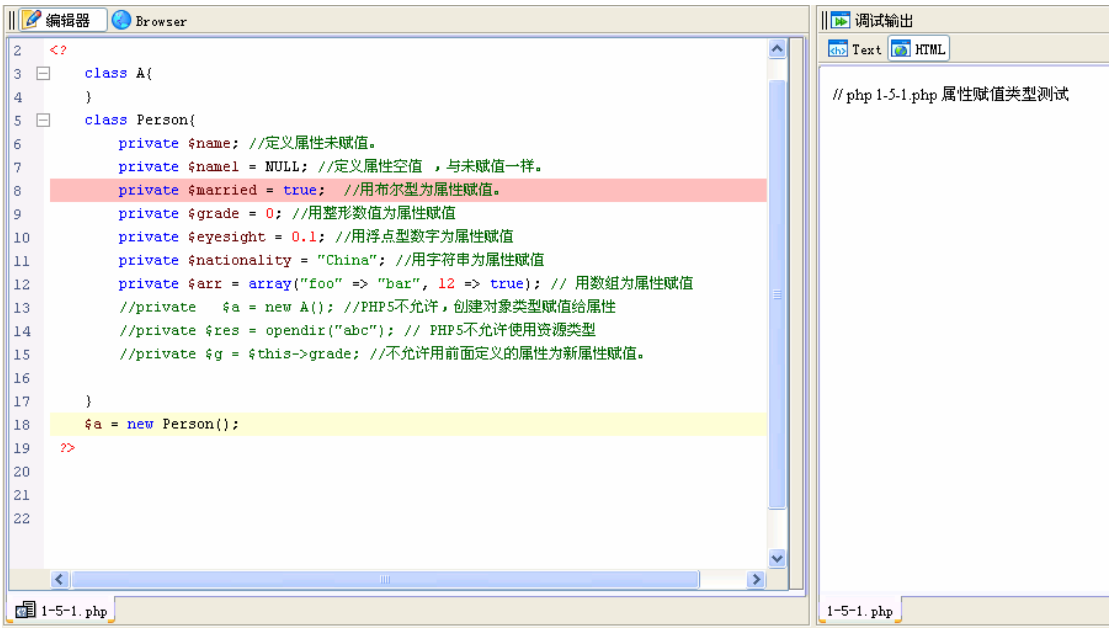
■ 属性的初值

在 PHP5 中，在属性定义可以不设置初值，或者赋予以下红色类型的初值。

PHP 中简单类型有 8 种，分别是：

- **数值类型**
  - **boolean** 布尔类型
  - **integer** 整型
  - **float** 浮点型，也称为 **double** 双精度浮点型
  - **string** 字符串
- **复合类型**
  - **array** 数组
  - **object** 对象
- **特殊类型**
  - resource 资源
  - **NULL**

例：1-5-1.php




```php
<?
    class A{
    }
    class Person{
        private $name; //定义属性未赋值。
        private $name1 = NULL; //定义属性空值 ，与未赋值一样。
        private $married = true;  //用布尔型为属性赋值。
        private $grade = 0; //用整形数值为属性赋值
        private $eyesight = 0.1; //用浮点型数字为属性赋值
        private $nationality = "China"; //用字符串为属性赋值
        private $arr = array("foo" => "bar", 12 => true); // 用数组为属性赋值
        //private   $a = new A(); //PHP5不允许，创建对象类型赋值给属性
        //private $res = opendir("abc"); // PHP5不允许使用资源类型
        //private $g = $this->grade; //不允许用前面定义的属性为新属性赋值。

    }
    $a = new Person();
?>
```

注意：

在上面例子中，第 13 行，尝试创建对象并将值赋予属性$a 会报错。

第 14 行，建立资源并复制给$res 会出现错误。

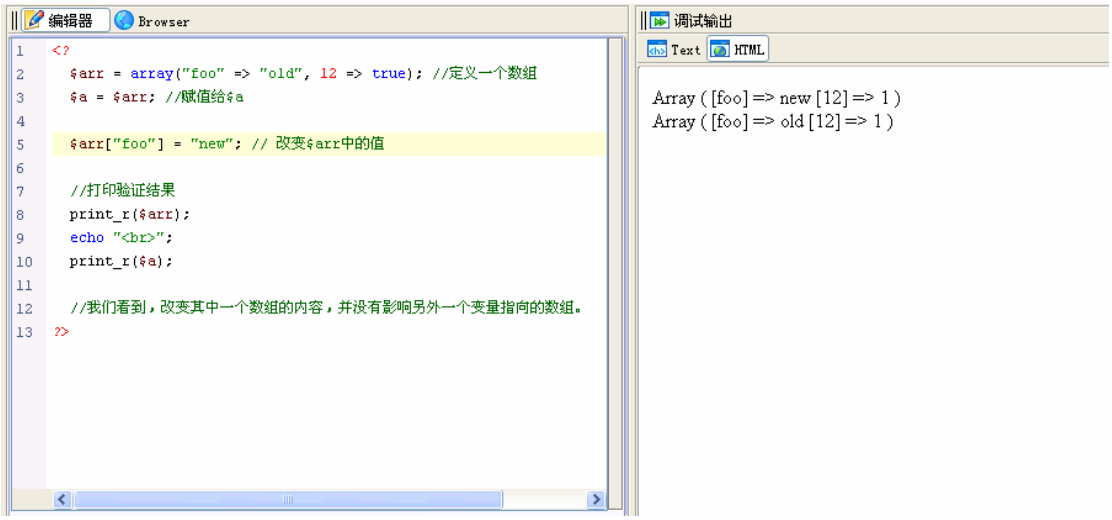
第 15 行，使用上面定义的属性为新属性赋值也会产生错误。

（在 Java 中，可以作 13 行和 15 行这样的操作。

PHP5 中定义属性的默认值，被限制到最简单的方式。其它的操作，交给构造方法操作，后面内容中将讲解构造方法。）

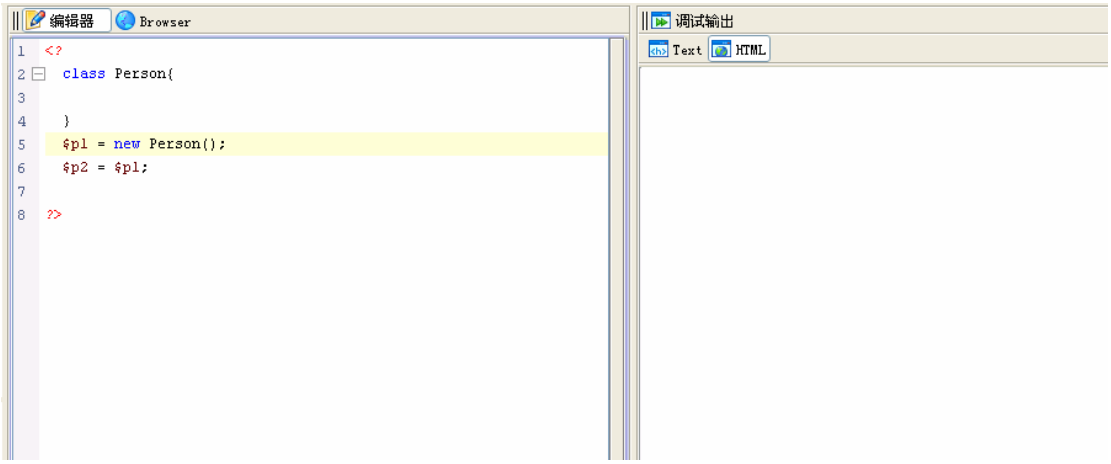
- 变量与引用变量

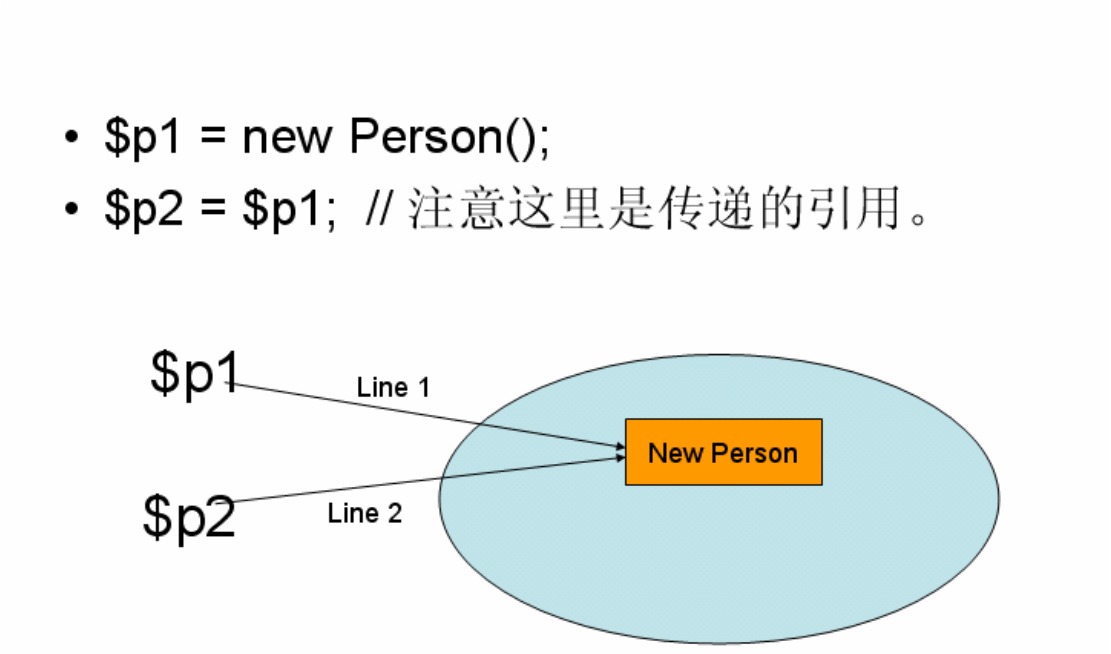
普通变量间的传值方法，就是值的赋值。比如数组。



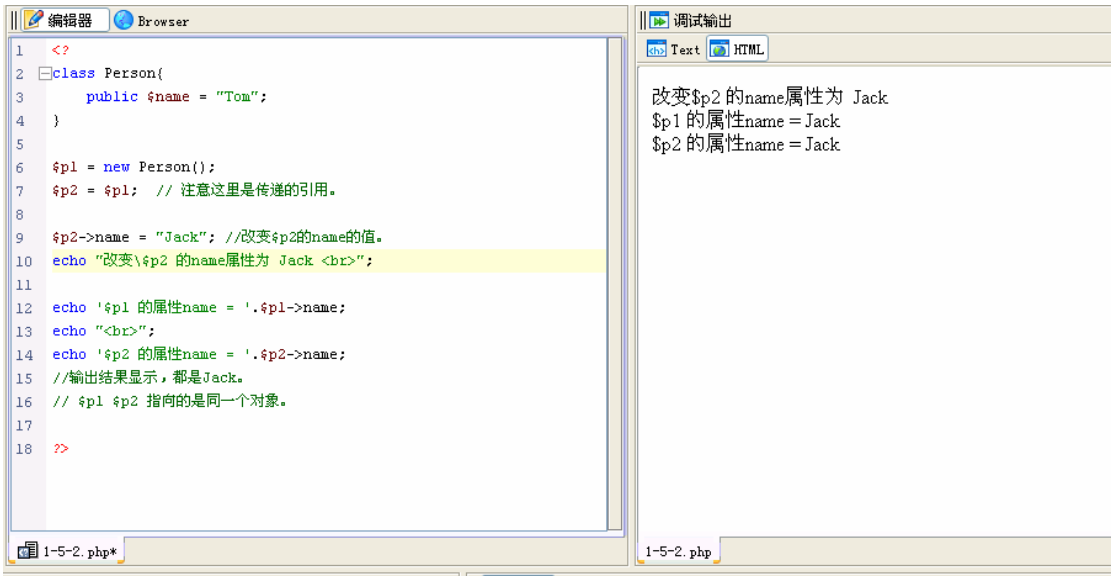

```php
<?
  $arr = array("foo" => "old", 12 => true); //定义一个数组
  $a = $arr; //赋值给$a

  $arr["foo"] = "new"; // 改变$arr中的值

  //打印验证结果
  print_r($arr);
  echo "<br>";
  print_r($a);

  //我们看到，改变其中一个数组的内容，并没有影响另外一个变量指向的数组。
?>
```

而指向对象的变量，是一个引用变量。在这个变量里面存储的是所指向对象的内存地址。引用变量传值时，传递的是这个对象的指向。而非复制这个对象。





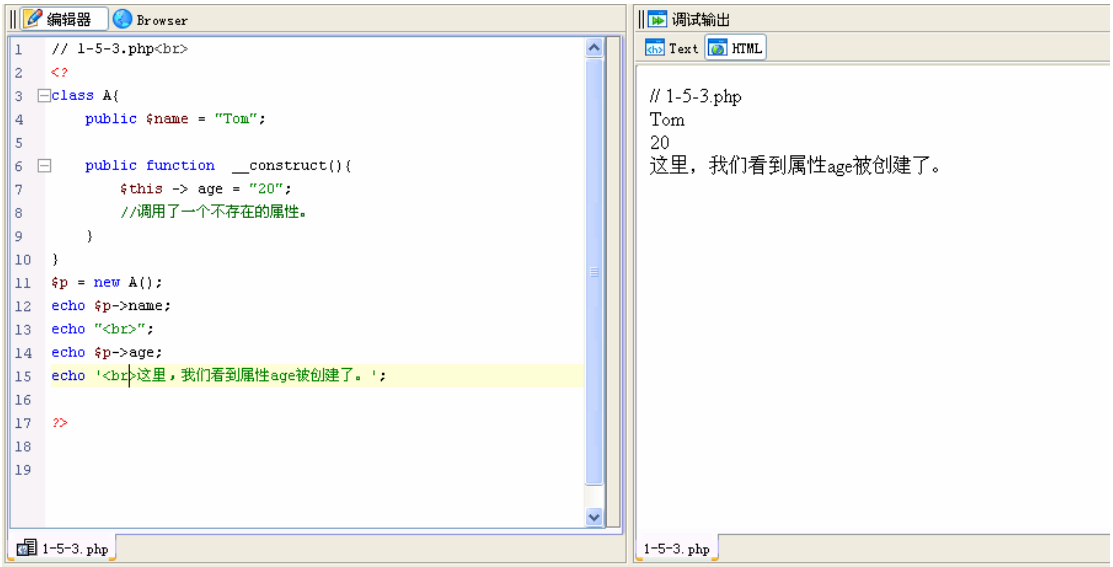
例 1-5-1.php 说明了这个问题。



- 属性的扩充

$this 指当前对象。
$this-> 调用当前对象的 属性或者方法。

在类中使用$this-> 调用一个未定义的属性时，PHP5 会自动创建一个属性供使用。
这个被创建的属性，默认的方法权限是 public。
例：1-5-3.php



1.4 PHP5 中的方法

方法：对对象的属性进行的操作称为对象的方法（也称为行为/操作）。

过程 函数 方法

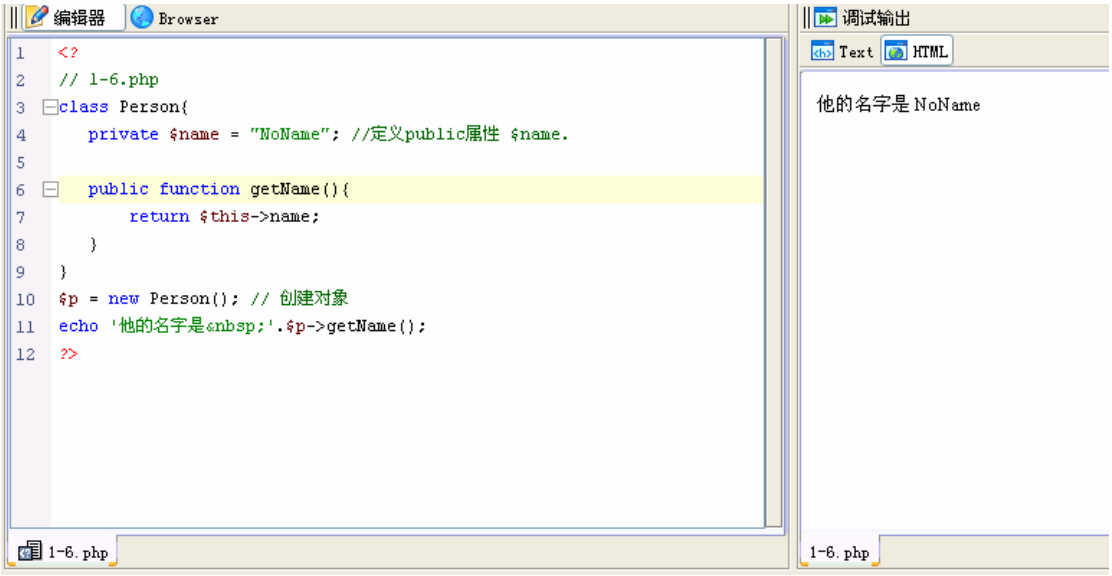
过程：过程是编制程序时定义的一个语句序列，用来完成某种指定的操作。

函数：函数有返回值，也是定义的语句序列。

方法：在面向对象概念中，类里面的一段语句序列。

一般来说，在面向对象概念中，函数和方法两个名词是通用的。

例 1-6：通过方法读取属性。



上面的例子将属性设置为 private ，同时声明了 public 的 getName()方法，用来获取属性$name 的值，调用 getName()方法就会通过 return $this->name 返回 $name 的值。

注意：这里，方法内部调用本地属性时，使用 $this->name 来获取属性。在这个例子中，设置了公开的 getName()方法，即用户只能获取$name，而无法改变他的值。这就是封装的好处。

- **封装指的是将对象的状态信息（属性）和行为（方法）捆绑为一个逻辑单元的机制。**
- **PHP5 中通过将数据封装、声明为私有的(private)，再提供一个或多个公开的（public）方法实现对该属性的操作，以实现下述目的：**
  - 隐藏一个类的实现细节；
  - 防止对封装数据的未经授权的访问。使用者只能通过事先定制好的方法来访问数据，可以方便地加入控制逻辑，限制对属性的不合理操作；
  - 有利于保证数据的完整性；
  - 便于修改，增强代码的可维护性；

- **方法的参数**

通过方法定义时的参数，可以向方法内部传递变量。

如下第 5 行，定义方法时定义了方法参数$_a,。 使用这个方法时，可以向方法内传递参数变量。方法内接受到的变量是局部变量，仅在方法内部有效。可以通过向属性传递变量值的方式，让这个变量应用于整个对象。

```
3  class Person{
4      private $a;
5      function setA($_a){
6          $this->a = $_a; // 局部变量$_a 传值给属性$a;
7      }
8      function getA(){
9          return $this->a; // 返回属性$a;
10     }
11 }
12
13 $p = new Person(); // 创建Person的对象。
14 $p->setA(100);
15 echo $p->getA();
```

```
100
```

如果声明这个方法有参数，而调用这个方法时没有传递参数，或者参数数量不足，系统会报出错误。

第 14 行调用方法 setA 的时候，没有传递参数。

编辑器 Browser

```
1  <? // 1-6-2.php
2
3  class Person{
4      private $a;
5      function setA($_a){
6          $this->a = $_a; // 局部变量$_a 传值给属性$a;
7      }
8      function getA(){
9          return $this->a; // 返回属性$a;
10     }
11 }
12
13 $p = new Person(); // 创建Person的对象。
14 $p->setA();
15 echo $p->getA();
16 ?>
17
18
19
```

1-6-2.php

调试输出

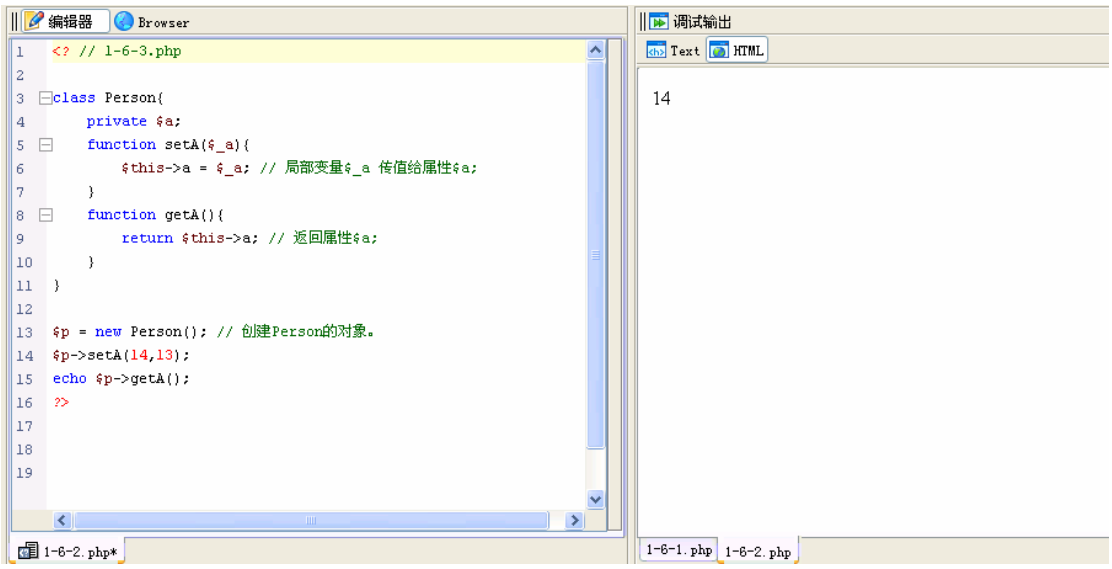
Text HTML

Warning: Missing argument 1 for Person::setA(), called in D:\phpWebTest\part1\1-6-2.php on line 14 and defined in **D:\phpWebTest\part1\1-6-2.php** on line **5**

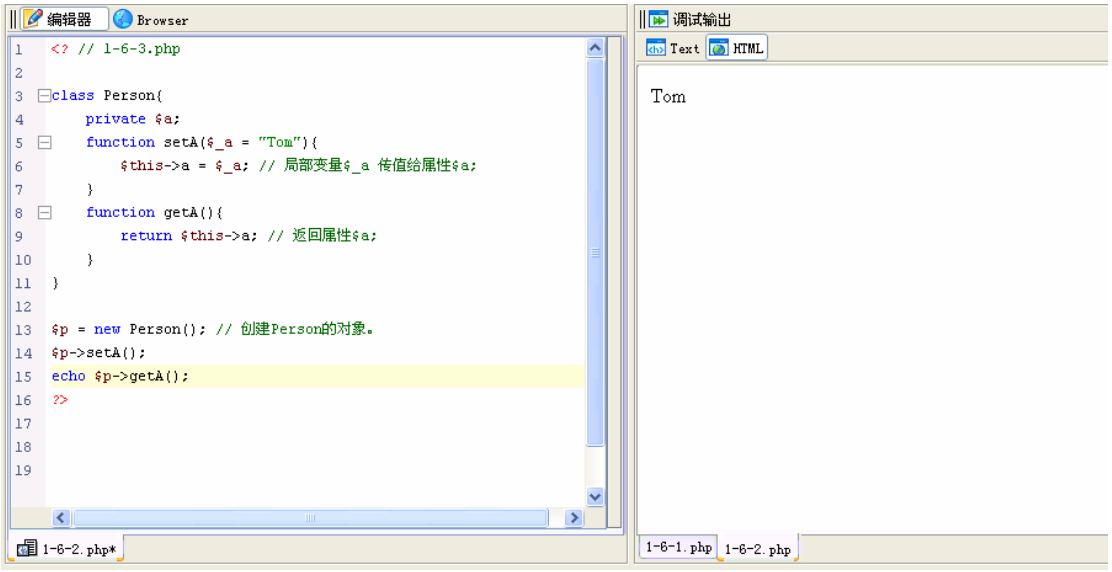
1-6-1.php 1-6-2.php

如果参数数量超过方法定义参数的数量，PHP 就忽略多于的参数。不会报错。

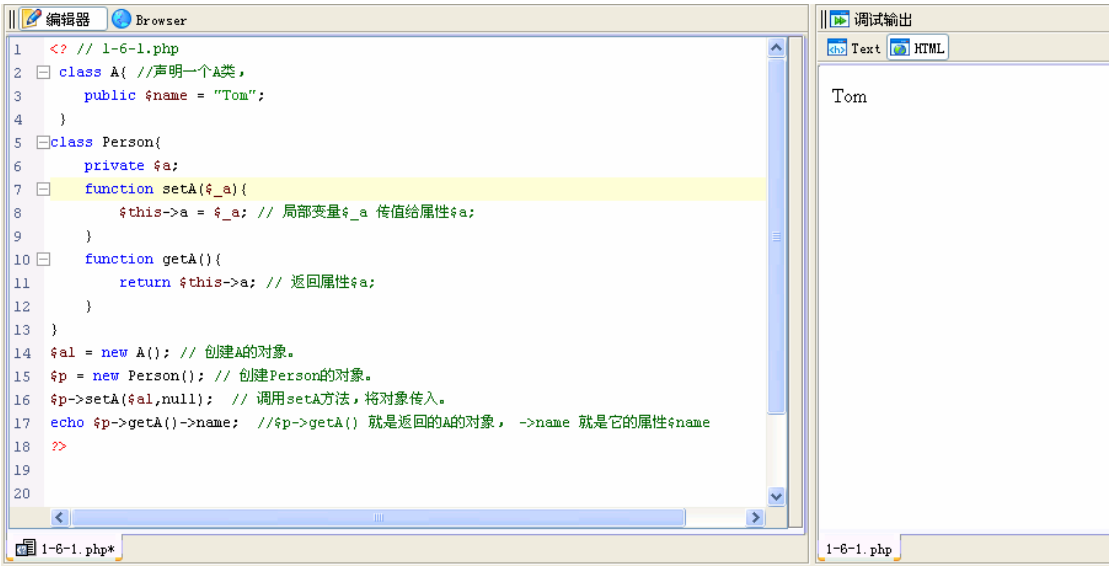
注意第 14 行。



可以在函数定义时为参数设定默认值。
在调用方法时，如果没有传递参数，将使用默认值填充这个参数变量。



可以向一个方法内部传递另外一个对象的引用变量。
在方法内部，这个引用可以一直传递，在需要的时候，调用这个对象的属性和方法。
例 1-6-1.php



再次提示

在 PHP5 中，指向对象的变量是引用变量。在这个变量里面存储的是所指向对象的内存地址。引用变量传值时，传递的是这个对象的指向。而非复制这个对象。

这与其它类型赋值有所不同。

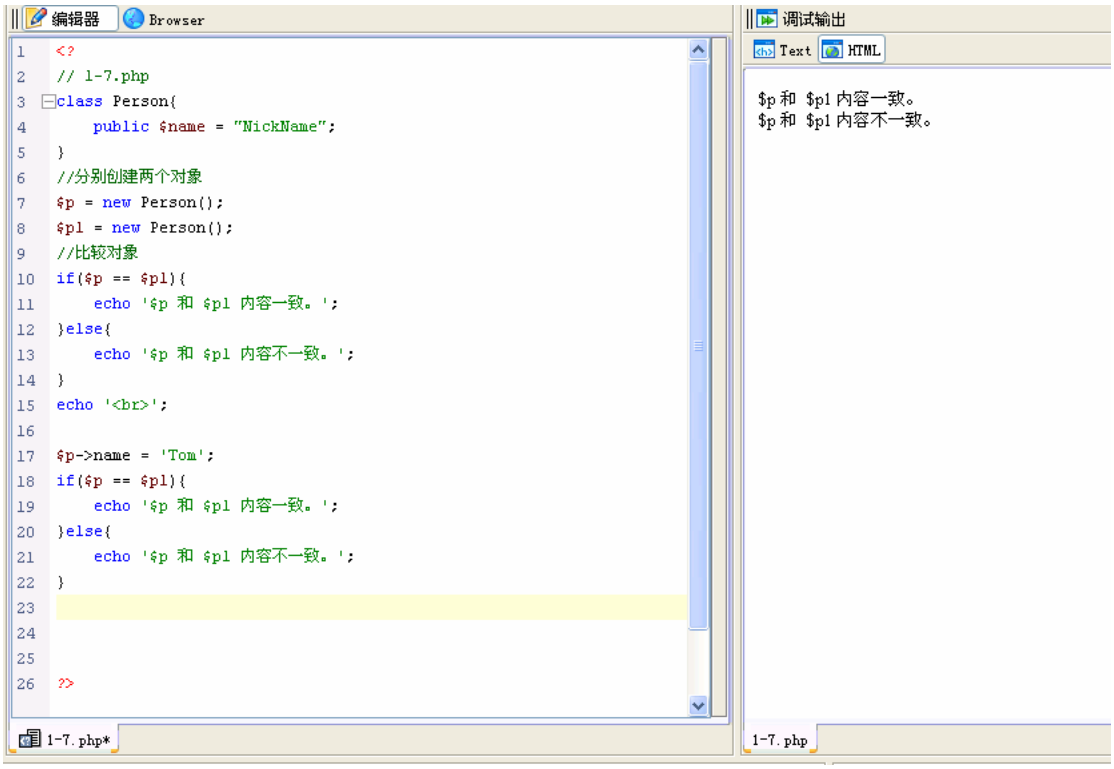
1.5 对象的比较

在 PHP 中有 = 赋值符号、== 等于符号 和 === 全等于符号，这些符号代表什么意思？

- 当使用比较操作符（==）时，对象以一种很简单的规则比较：当两个对象有相同的属性和值，属于同一个类且被定义在相同的命名空间中，则两个对象相等。等于符号比较对象时，比较对象是否有相同的属性和值。

  注意：== 比较两个不同的对象的时候，可能相等也可能不等。

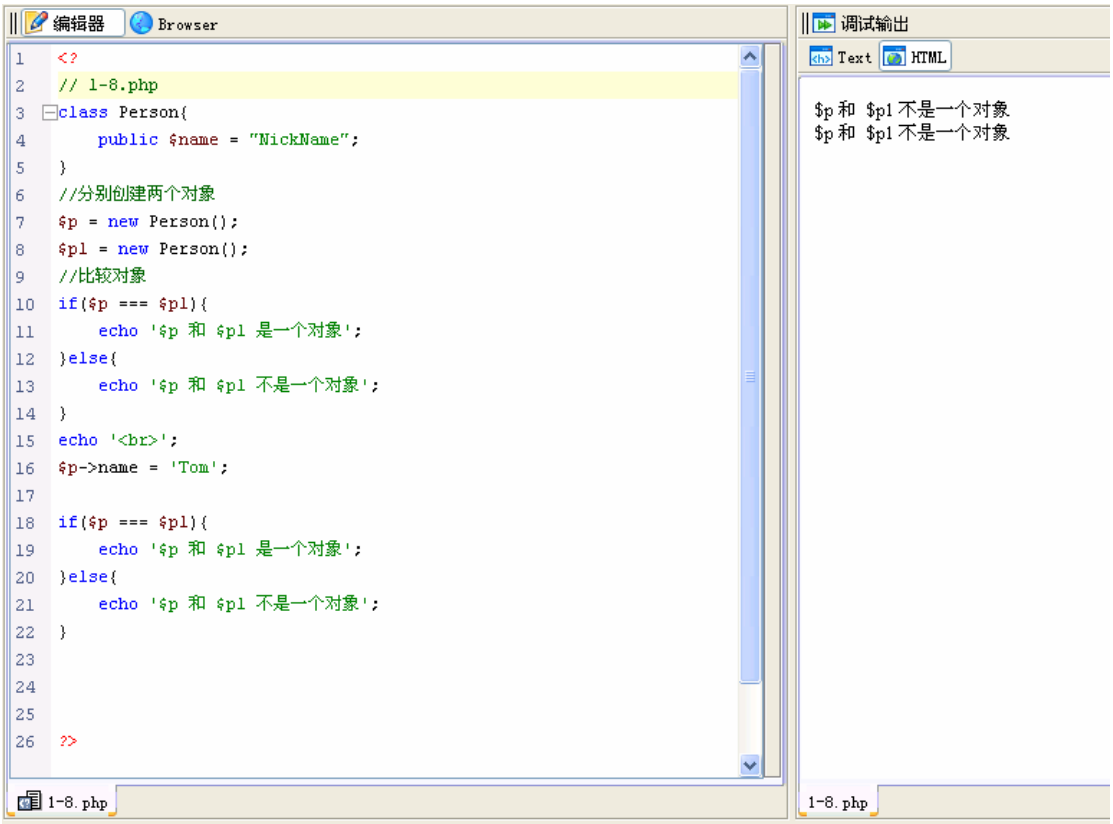
例：1-7：



使用 == 符号比较两个对象 ，比较的仅仅是两个对象的内容是否一致。

- 当使用全等符（===）时，当且仅当两个对象指向相同类（在某一特定的命名空间中）的同一个对象时才相等。
  是否在是同一个对象，两边指向的对象是否有同样的内存地址。

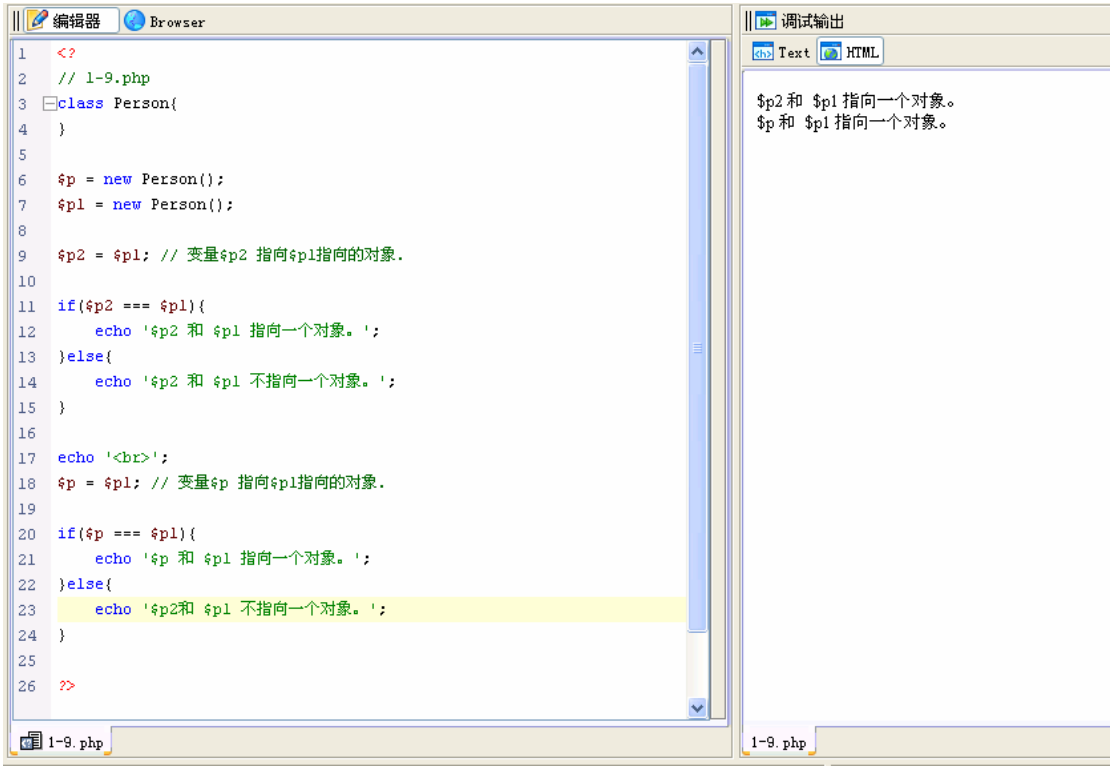
例 1-8.PHP



结果=== 比较的是两个变量是否一个对象。

- 一个等于符号（=）表示赋值，是赋值计算。
  如果将对象赋予变量，是指变量将指向这个对象。

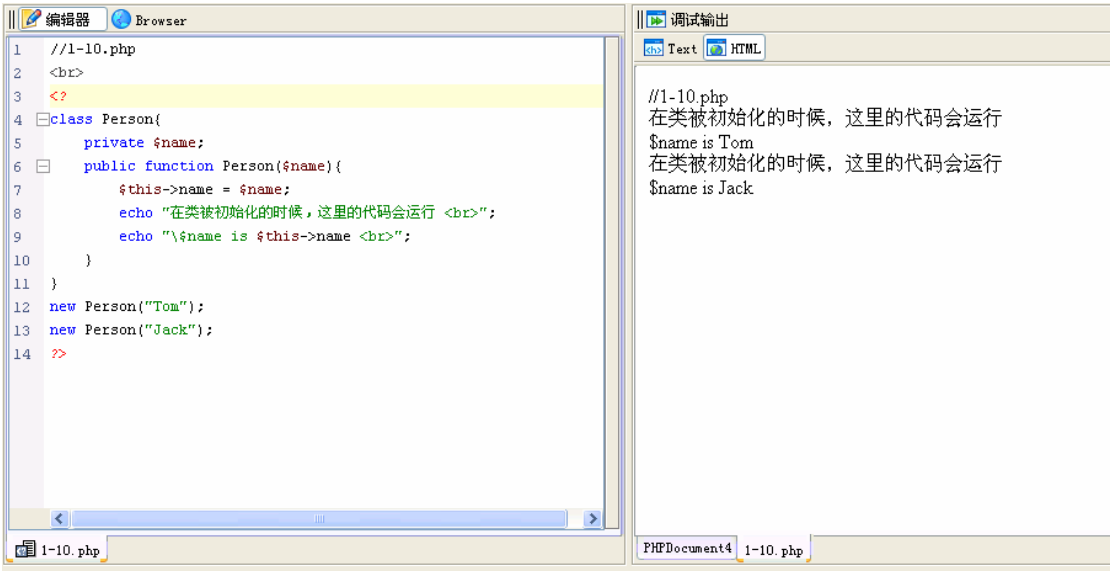
例：1-9



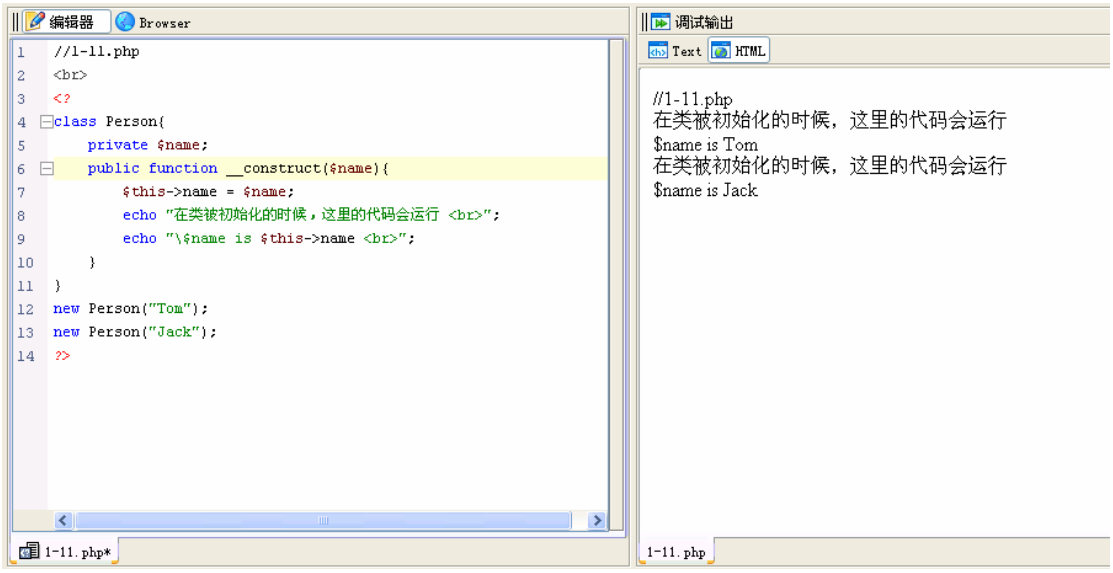
1.6 构造函数

构造方法又称为构造函数，是对象被创建时自动调用的方法，用来完成类初始化的工作。
构造函数和其它函数一样，可以传递参数，可以设定参数默认值。
构造函数可以调用属性，可以调用方法。
构造函数可以被其它方法显式调用。

在 PHP4 中使用与类名同名的方法为构造函数。在 PHP5 中依然支持了这种方式，但不建议再使用这种方式。
例 1-10.php



在 PHP5 中规定构造函数使用_ _construct() 函数表示构造函数，注意是两个 _ 。
例 1-11.php



1.7 析构函数与 PHP 的垃圾回收机制

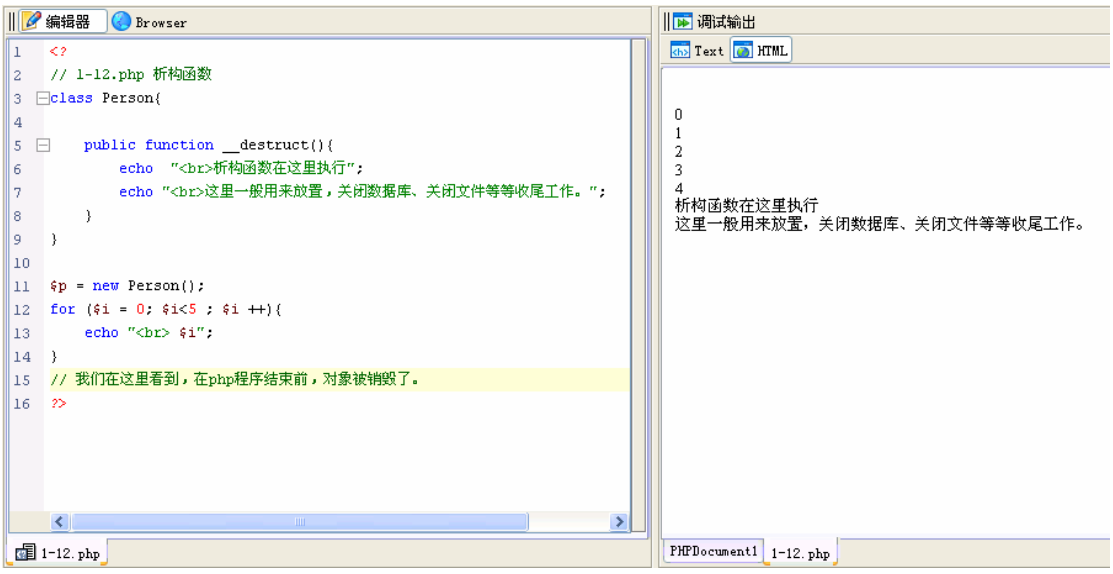
析构函数：当某个对象成为垃圾或者当对象被显式销毁时执行。

GC(Garbage Collector)
在 PHP 中，没有任何变量指向这个对象时，这个对象就成为垃圾。PHP 会将其在内存中销毁。
这是 PHP 的 GC(Garbage Collector)垃圾处理机制,防止内存溢出。
当一个 PHP 线程结束时，当前占用的所有内存空间都会被销毁，当前程序中的所有对象同样被销毁。

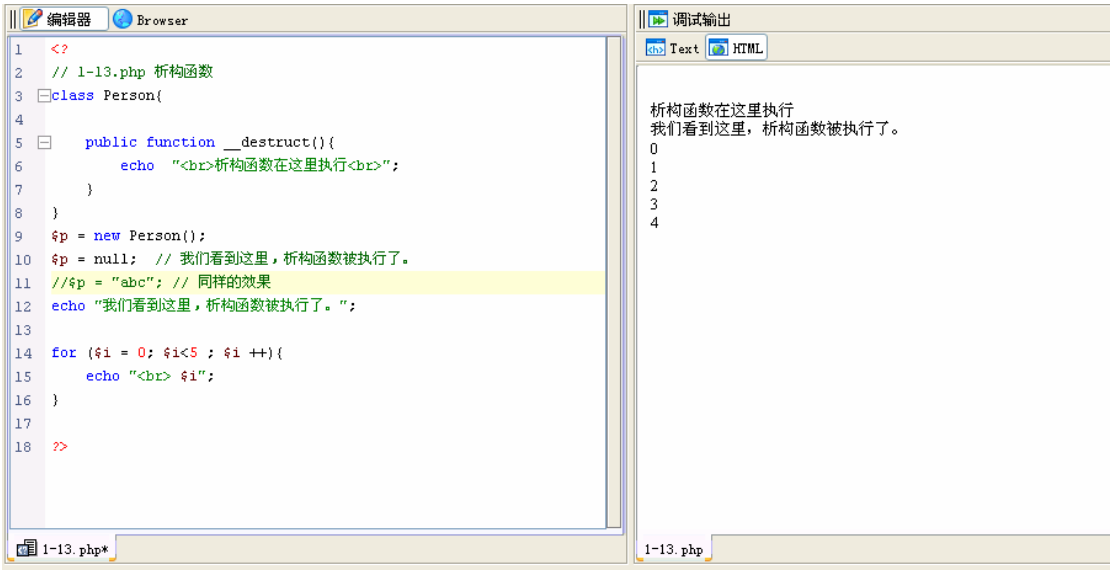
__destruct() 析构函数，是在垃圾对象被回收时执行。

析构函数也可以被显式调用,但不要这样去做。
析构函数是由系统自动调用的，不要在程序中调用一个对象的虚构函数。
析构函数不能带有参数。

例 1-12.php 。程序结束前，所有对象被销毁。
析构函数被调用了。

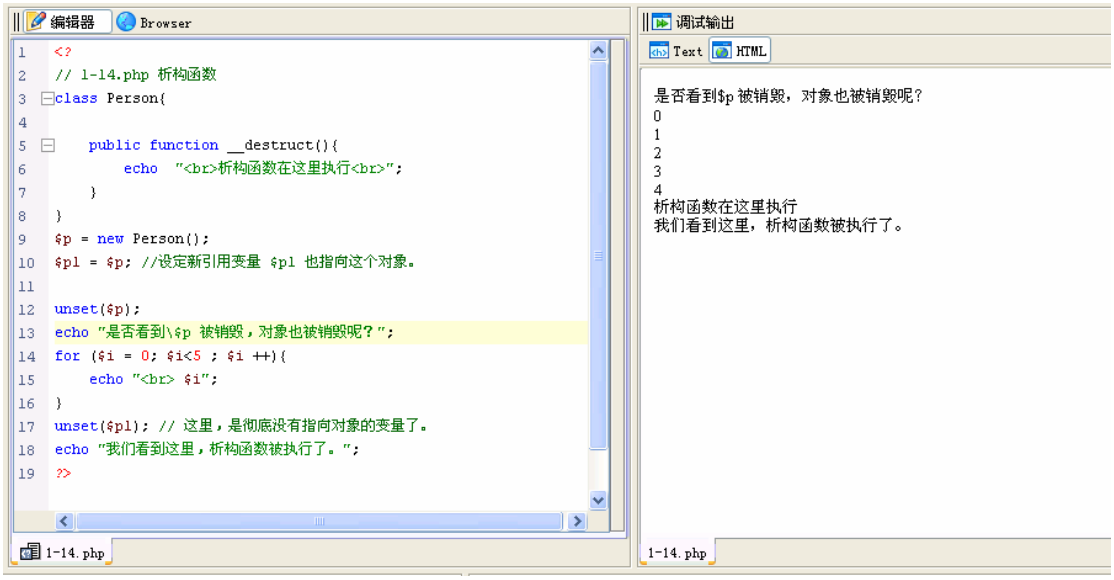


例 1-13.php 当对象没有指向时，对象被销毁。



上面的例子第 10 行，我们将$p 设置为空或者第 11 行赋予$p 一个字符串，这样$p 之前指向的对象就成为了垃圾对象。PHP 将这个对象垃圾销毁。

例 1-14.php unset 变量

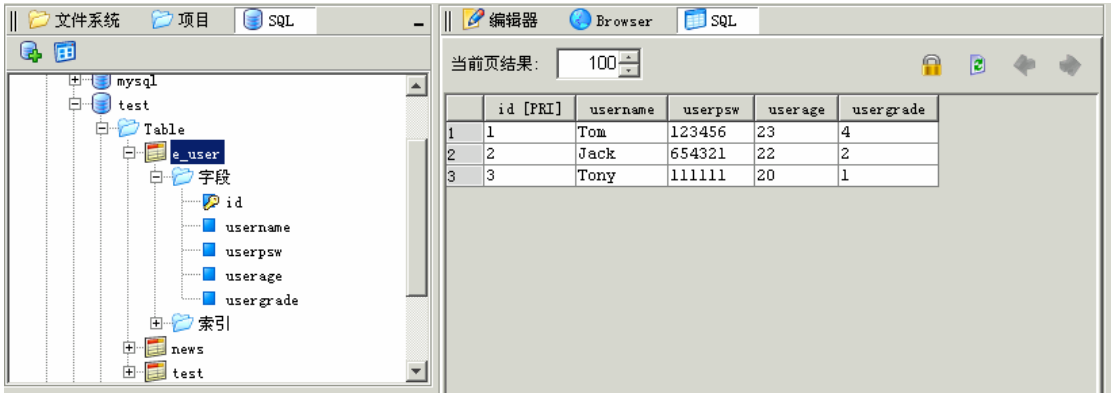


unset 一个引用变量时。
unset 销毁的是指向对象的变量，而不是这个对象。

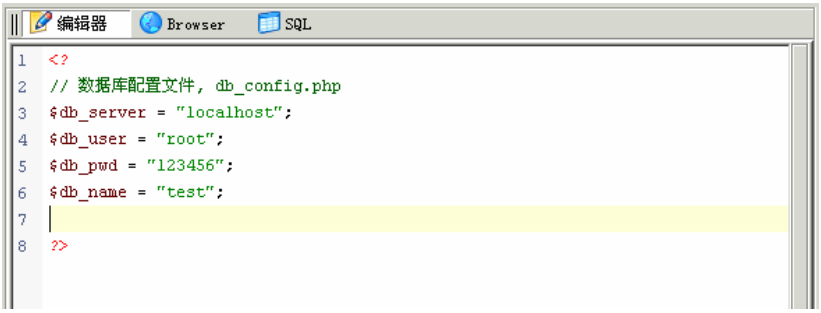
1-8 面向对象实例

我们使用面向过程的方式和面向对象的方式分别写几个程序，理解面向对象编程带来的优势。

数据库使用 mysql 数据库， 数据库结构和数据如下图所示。

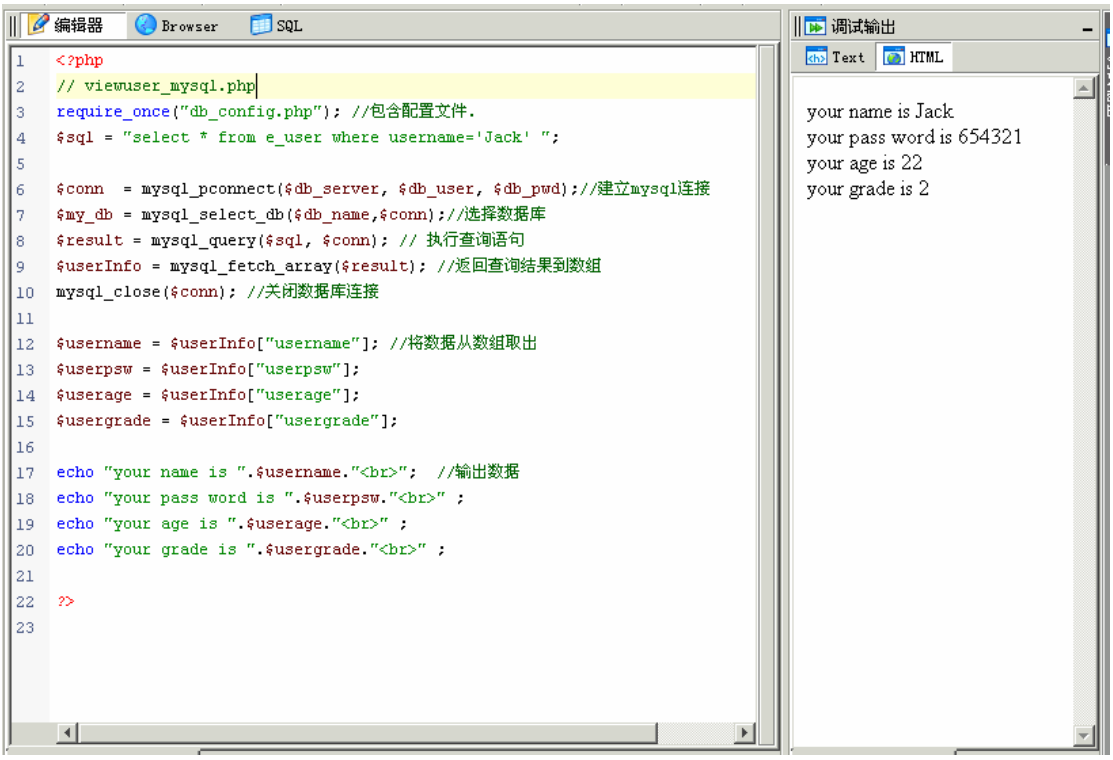


先写一个数据库配置文件如下：



我们先写一个纯粹面向过程的方式来读取数据库中的用户信息。

例： example



这个思维模式我们太熟悉不过了。

1. 读取配置文件中的数据库参数。
2. 建立数据库连接。
3. 选择数据库。
4. 执行 sql 语句。
5. 将数据返回给数组。
6. 将每个数据内容取出。
7. 将信息显示。

现在我们写一个面向对象的取数据库信息的内容。

例：example1

编辑器 Browser SQL

```php
<?
// class_user.php
class UserInfo{
    private $userName;  //属性,用户名
    private $userPSW ;  //属性,用户密码
    private $userAge ;  //属性,用户年龄
    private $userGrade ; //属性,用户级别
    private $userInfo; //存储数据库返回信息的数组变量.

    public function __construct($name){
        require_once("db_config.php"); //包含配置信息.
        $sql = "select * from e_user where username='$name' "; //书写sql
        $conn  = mysql_pconnect($db_server, $db_user, $db_pwd);//建立mysql连接
        $my_db = mysql_select_db($db_name,$conn);//选择数据库
        $result = mysql_query($sql, $conn); // 执行查询语句
        $this->userInfo = mysql_fetch_array($result); //返回查询结果到数组
        mysql_close($conn); //关闭数据库连接
        $this->getInfo(); //调用传递信息的方法.
    }
    // 获取信息传递给属性的方法
    private function getInfo(){
        $this->userName = $this->userInfo["username"];
        $this->userPSW = $this->userInfo["userpsw"];
        $this->userAge = $this->userInfo["userage"];
        $this->userGrade = $this->userInfo["usergrade"];
    }

    //返回每个属性的public 方法.
    public function getUserName(){
        return $this->userName;
    }

    public function getUserPSW(){
        return $this->userPSW;
    }

    public function getUserAge(){
        return $this->userAge;
    }

    public function getUserGrade(){
        return $this->userGrade;
    }
}
?>
```

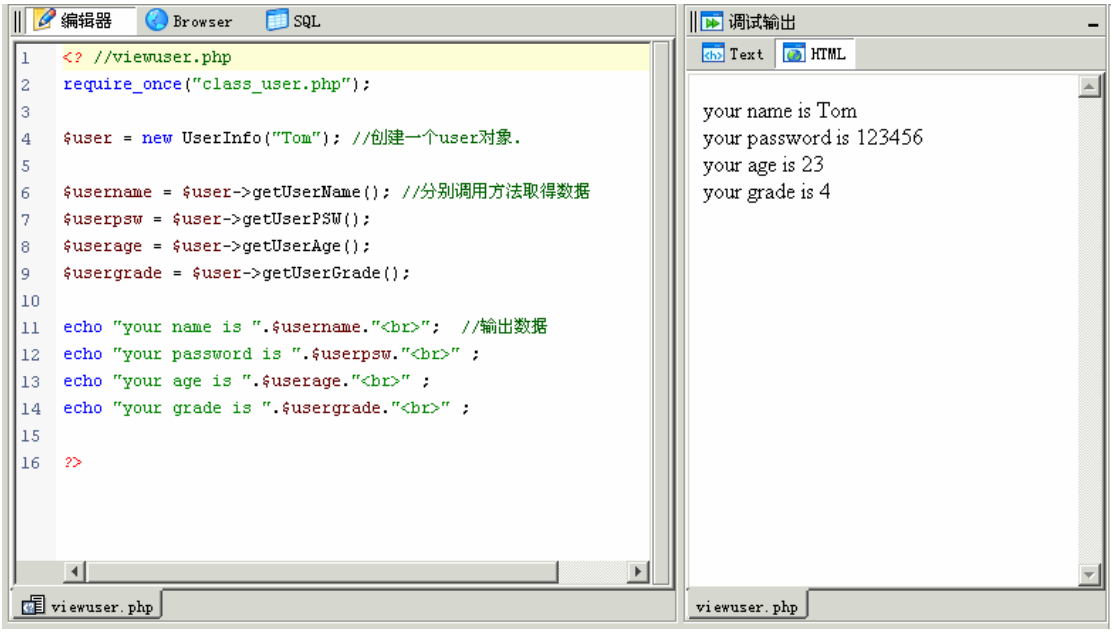
写 class 好像麻烦了些，但优点是结构清晰、扩展、重用和维护方便。

使用这个类非常复合我们看世界的眼光。
现在我们再做一次上帝，follow me。

一个典型的面向对象的案例。
大象放进冰箱里需要几步？
1. 打开冰箱门。
2. 大象放进去。
3. 关上冰箱门。

显示用户 Tom 的信息需要几步？
1. 创建 Tom 出来。
2. 让这个 Tom 告诉我们关于他的信息内容。
3. 显示这些信息。

例：



Tom 这个对象是如何创建的？创建时候做了什么？从那个服务器读取的？
从那个数据库读取的？从那个表单读取的？Tom 的信息是如何读取的？
这些问题，在这里我们不需要再考虑。写刚才那个 user 类的时候，已经考虑过了。
使用这个对象，就像使用家里的冰箱和微波炉一样方便、自然。
把 Tom 换成换成 jack 试试？

那现在，你愿意使用流水账一样的方式写代码，还是更自然的面向对象呢？

第二章 PHP5 面向对象进阶

10 月 27 日，家里添了一个男孩，生活变的忙碌起来。照顾小宝宝是件辛苦的事情，也更明白父母的辛苦。工作之余，继续写起来时间越来越紧张。

祝老婆早日康复，祝我的小宝健康成长，祝父母身体健康。

如果看到代码有什么问题。可以到 phpchina.com
我的博客留言给我。http://www.phpchina.com/?10 。

再次感谢 www.phpchina.com

gaoxiangming@hotmail.com
刀客羽朋
石家庄 2006-11-19

目录

2.1 类的继承

什么是继承

前面说过，面向对象的思想和我们的生活是息息相通的。
我们先分析一个生活中的例子：
自行车、折叠车、电动车的关系。
例 1：

自行车有什么特征（属性）？
- 两个轱辘
- 一个车座
- 两个脚蹬子
- 有颜色

自行车有什么动作（方法）？
- 骑行
- 刹车

折叠自行车有什么特征（属性）？
- 两个轱辘
- 一个车座
- 两个脚蹬子
- 有颜色

折叠自行车有什么动作（方法）？
- 骑行
- 刹车
- **折叠**

电动自行车有什么特征（属性）？
- 两个轱辘
- 一个车座
- 两个脚蹬子
- 有颜色
- 电池一块

电动自行车有什么动作（方法）？
- 骑行
- 刹车
- **电动行驶**

上面的三个表格，说明了自行车、折叠自行车、电动自行车特性。
我们描述折叠自行车和电动自行车时，除红色标注的部分，都和自行车一样。

我们尝试用另外一种方式，建立模型的方式来描述一次。

例 2：

自行车有什么特征（属性）？

- 两个轱辘
- 一个车座
- 两个脚蹬子
- 有颜色

自行车有什么动作（方法）？

- 骑行
- 刹车

折叠自行车有什么特征（属性）

- **折叠自行车和自行车有相同的属性**

折叠自行车有什么动作（方法）

- **折叠自行车具有自行车的所有方法。**
- **增加了折叠方法。**

电动自行车和自行车有相同的属性和方法。

- 增加了电池一块
- **增加了电动行驶的方法。**

这次的描述变简单了，只需要将增加的内容填写上去。关于自行车的描述被**复用**了。
仔细再观察对自行车的描述，我们发现上面三个自行车都缺少了一个重要的属性”车主架”。
在例 1 中，我们要在三个描述中分别添加”车铃铛”，这个属性。
在例 2 中，我们只要在自行车的描述中加入属性“车铃铛”，另外两个描述不用变化就完成内容的添加。同样，动作（方法）的变化也很容易。

感觉到了什么了么？**它让我们的描述更容易“扩充和维护”。**

PHP5 中的继承

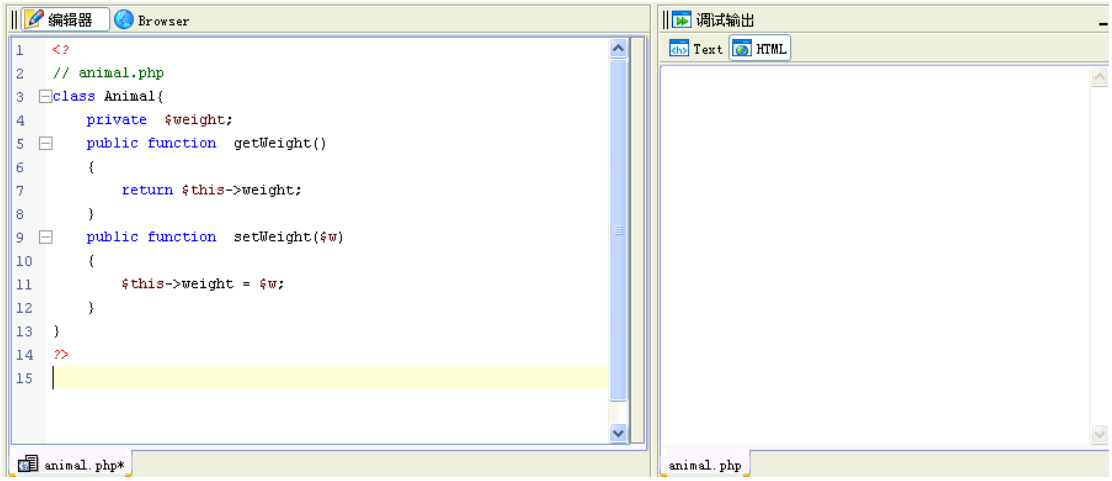
- 继承是面向对象最重要的特点之一，就是可以实现对类的复用。
- 通过“继承”一个现有的类，可以使用已经定义的类中的方法和属性。
- 继承而产生的类叫做子类。
- 被继承的类，叫做父类，也被成为超类。

- PHP 是单继承的，一个类只可以继承一个父类，但一个父类却可以被多个子类所继承。
- 从子类的角度看，它“ 继承（inherit ， extends）”自父类；而从父类的角度看，它“派生（derive）”子类。它们指的都是同一个动作，只是角度不同而已。
- 子类不能继承父类的私有属性和私有方法。
- 在 PHP5 中类的方法可以被继承，类的构造函数也能被继承。

继承的简单例子

我们分析自然界中的关系，动物类与犬类的关系。

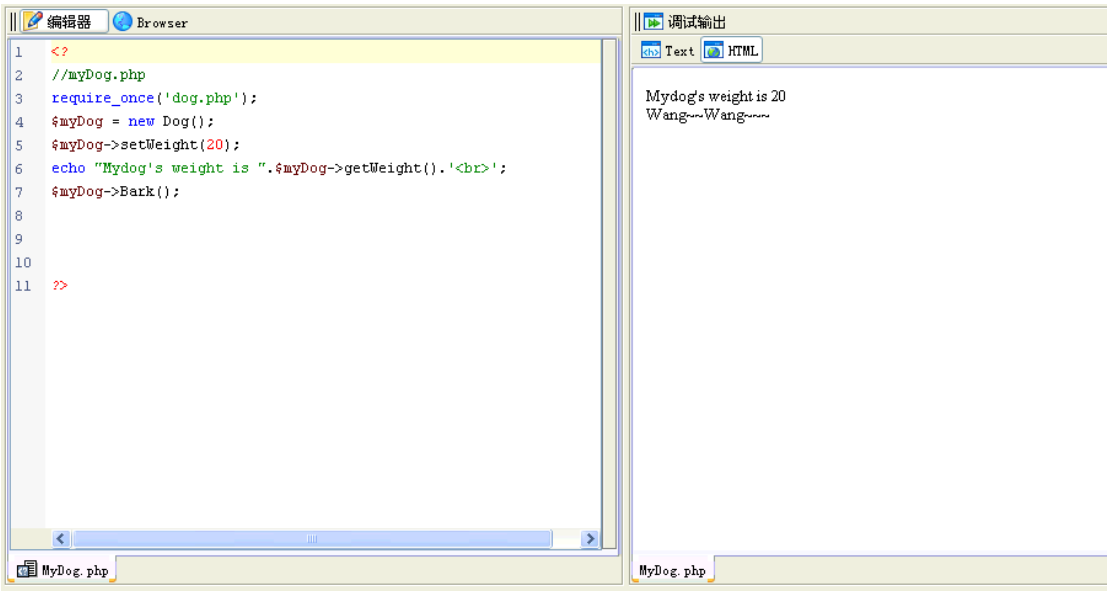
例 2-1　animal.php



Dog 类继承自 animal 类。
Dog.php

当我们实例化 animal 类的子类 Dog 类时， 父类的方法 setWeight（） 和 getWeight() 被继承。
我们可以直接调用父类的方法设置其属性$weight，取得其属性$weight 。

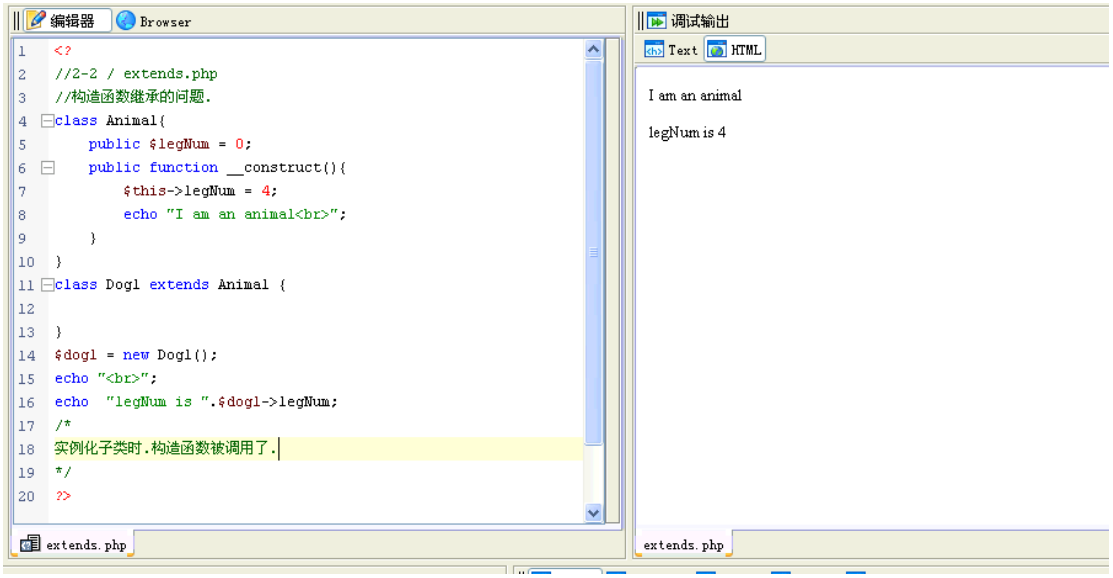
dog 类的实例。



构造函数的继承

有些资料上说 PHP5 的构造函数不被继承。
演示的结果证明，PHP5 的构造函数被继承了。

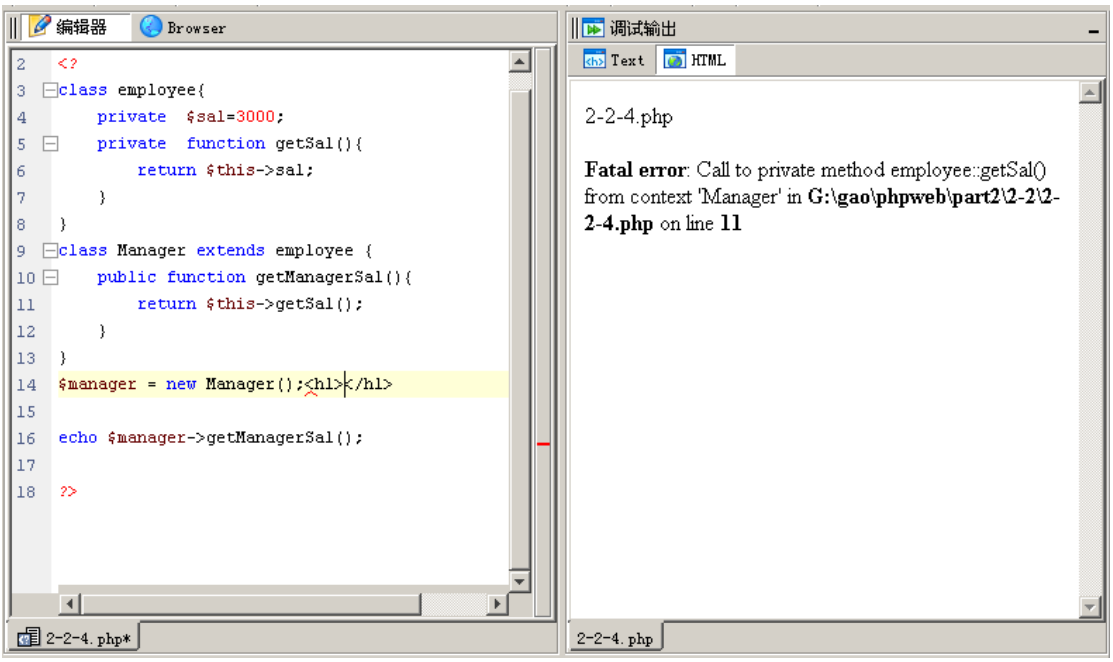
当子类 Dog1 被实例化时，继承的构造函数被调用了。
屏幕上显示了一句　" I am an Animal. " .



私有变量和方法不被继承

- 私有变量不被继承，私有方法也不被继承。
- 另外一种说法，私有变量和属性被继承了，但不能被调用。
- 无论怎么说，都不能调用父类的私有属性和私有方法。

例 2-2-4



2.2 访问控制

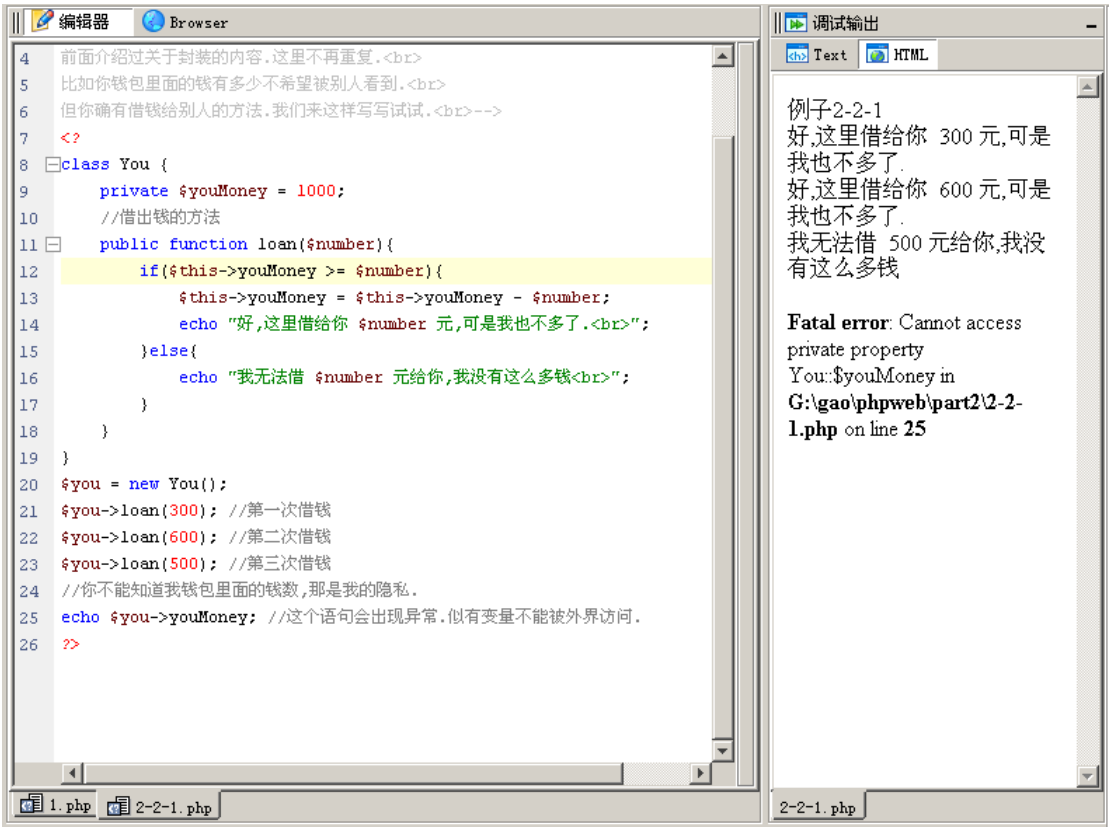
在 PHP5 中，可以在类的属性和方法前面加上一个修饰符（modifier），来对类进行一些访问上的控制。

下面表格显示了访问的权限。

修饰符	同一个类中	子类中	全局
private	Yes		
protected	Yes	Yes	
public	Yes	Yes	Yes(默认)

Private 的访问权限

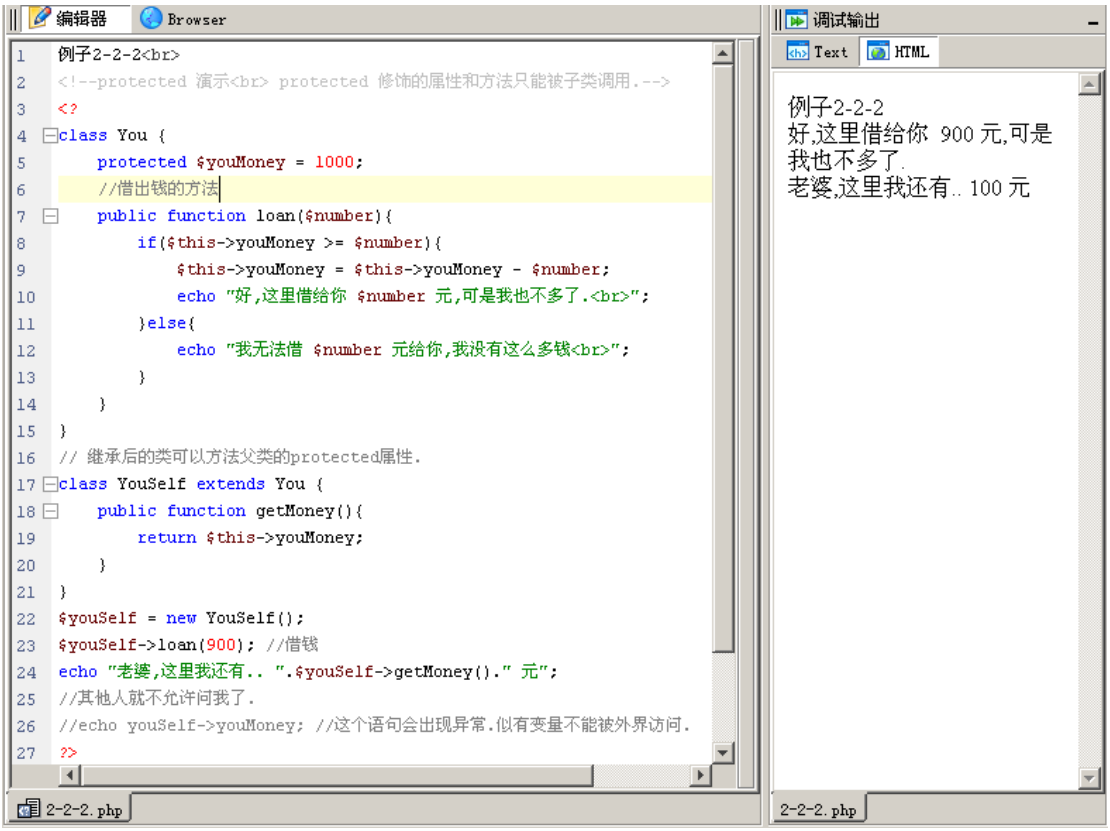
例 2-2-1
private 不能直接被外部调用，只能由当前对象调用。
前面介绍过关于封装的内容.这里不再重复。
比如你可以借钱给别人，但不希望别人知道你钱包里面有多少钱。
我们把它用 private 隐藏起来。



protected 的访问权限

例 2-2-2 演示

protected 修饰的属性和方法只能被子类调用。外界无法调用。



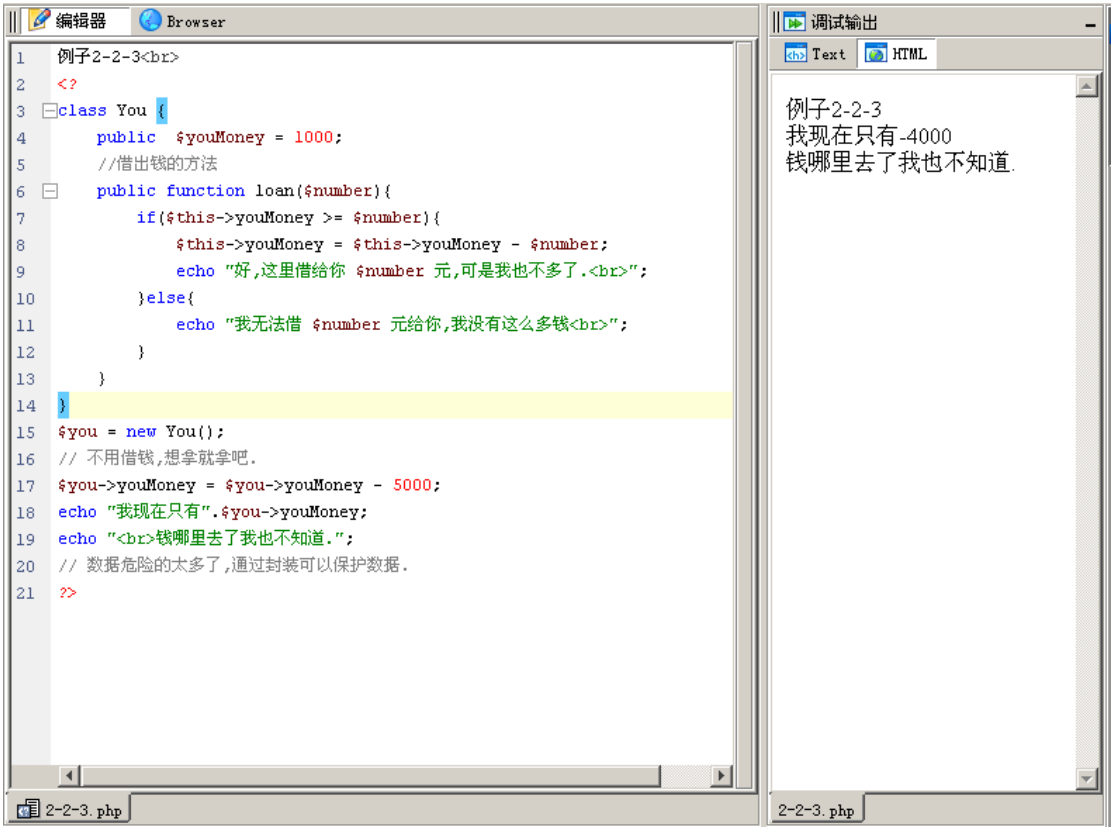
public 的访问权限

例子 2-2-3

数据的隐藏和封装是能够帮助我们保护数据的安全性。

Public 修饰的属性和方法，可以被无限制的调用。

嘿。。你的钱，不安全了。



2.3 重写（override）

•如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（**override**），也称为方法的重写。

•当对父类的方法进行重写时，子类中的方法必须和父类中对应的方法具有相同的方法名称，在 PHP5 中不限制输入参数类型、参数数量和返回值类型。（这点和 JAVA 不同）

•子类中的覆盖方法不能使用比父类中被覆盖方法更严格的访问权限。

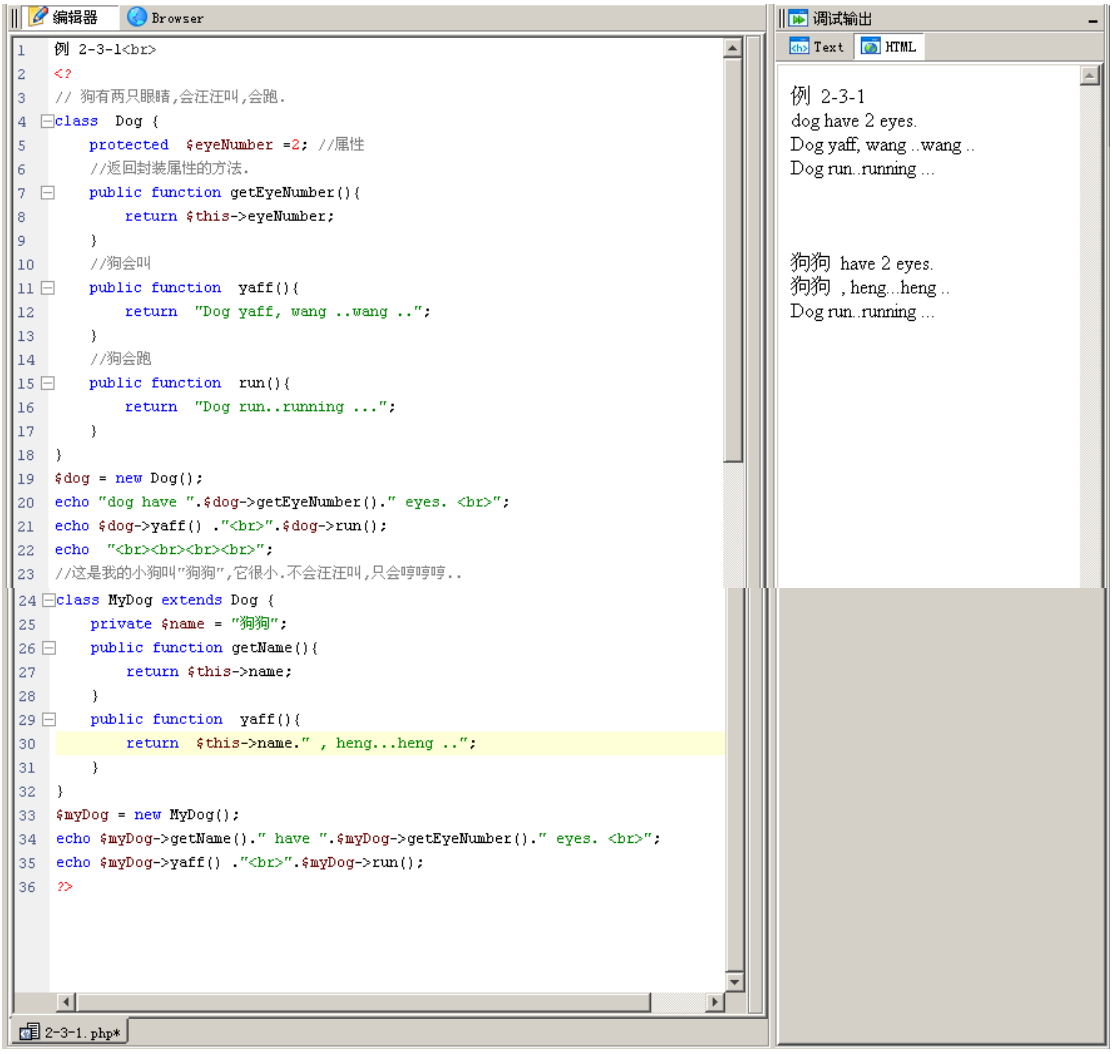
•声明方法时，如果不定义访问权限。默认权限为 public。

PHP5 重写方法

先设置一个父类，这个父类是 “Dog”类，这个类描述了 dog 的特性。
Dog 有 2 个眼睛，会跑，会叫。就这样描述先。

我养了一直狗，是只小狗，符合 Dog 类的特性，但有所不同。
我的小狗有名字，我的小狗太小了，不会大声的叫，只会哼哼。

我们用继承的概念去实现这个设计。
例子 2-3-1

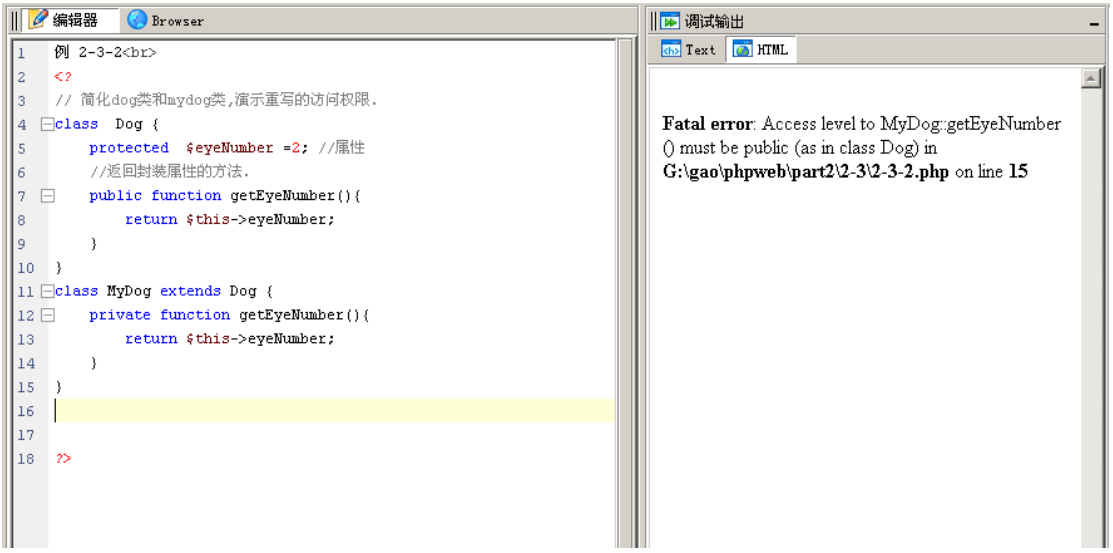


重写方法与访问权限

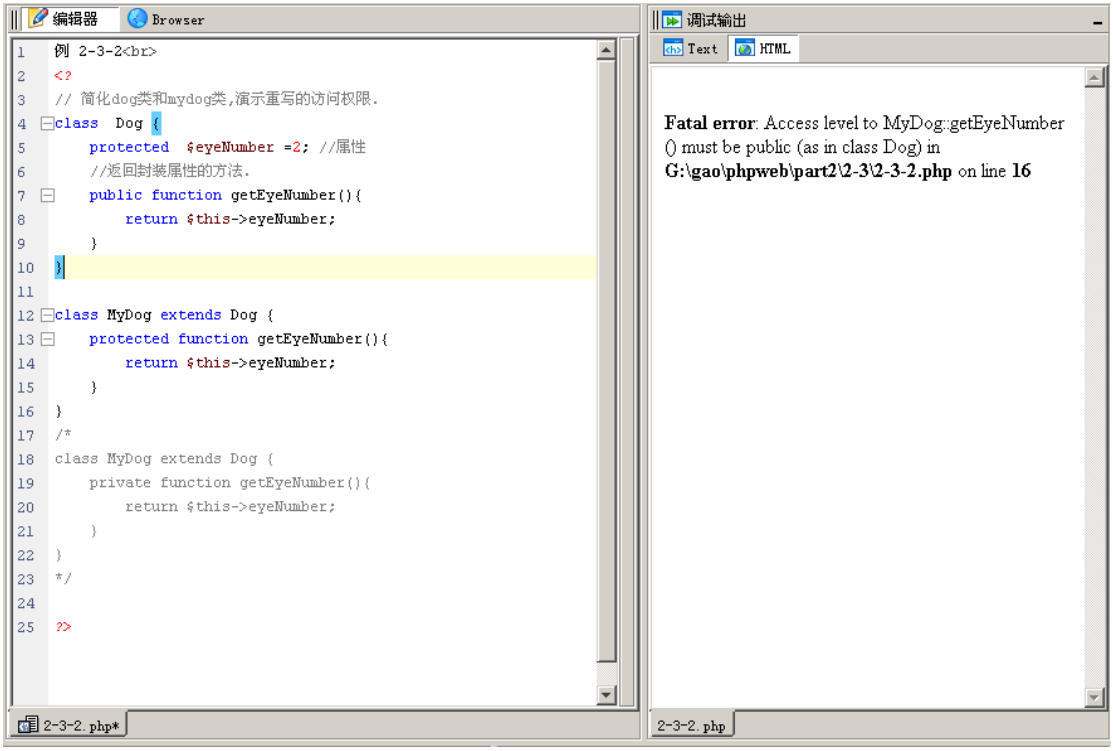
子类中的覆盖方法不能使用比父类中被覆盖方法更严格的访问权限。

例 2-3-2

父类为 public 子类为 private 时。



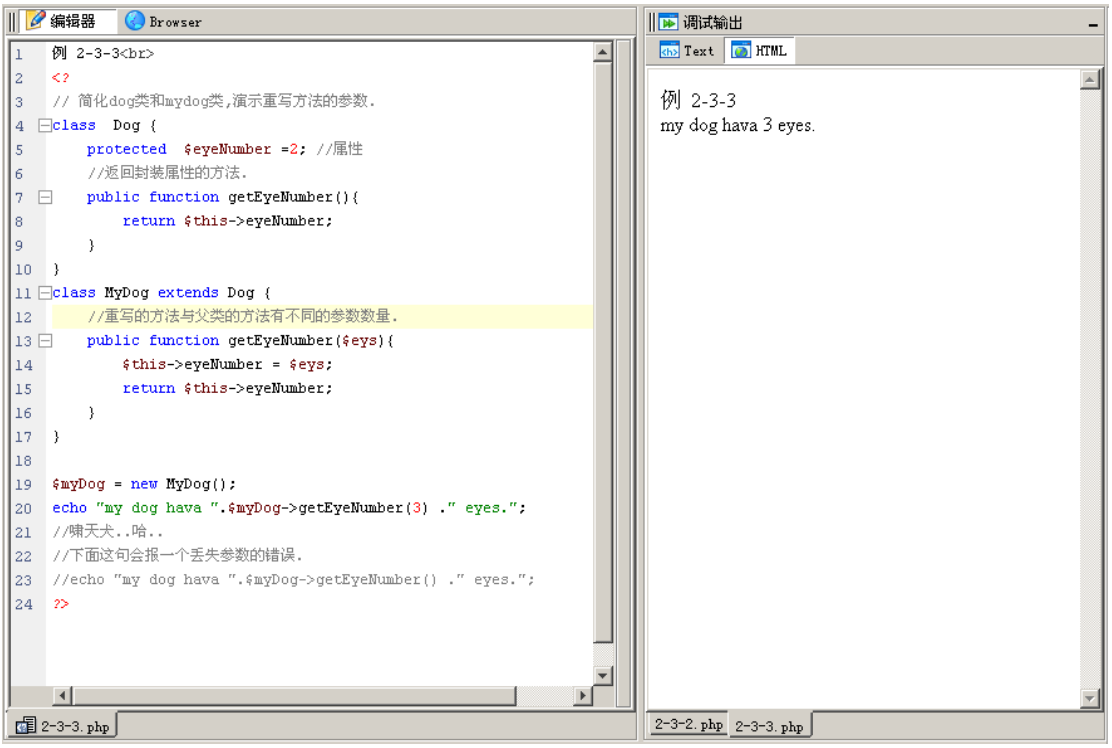
父类为 public 子类为 protected 时。



重写时的参数数量

例 2-3-3

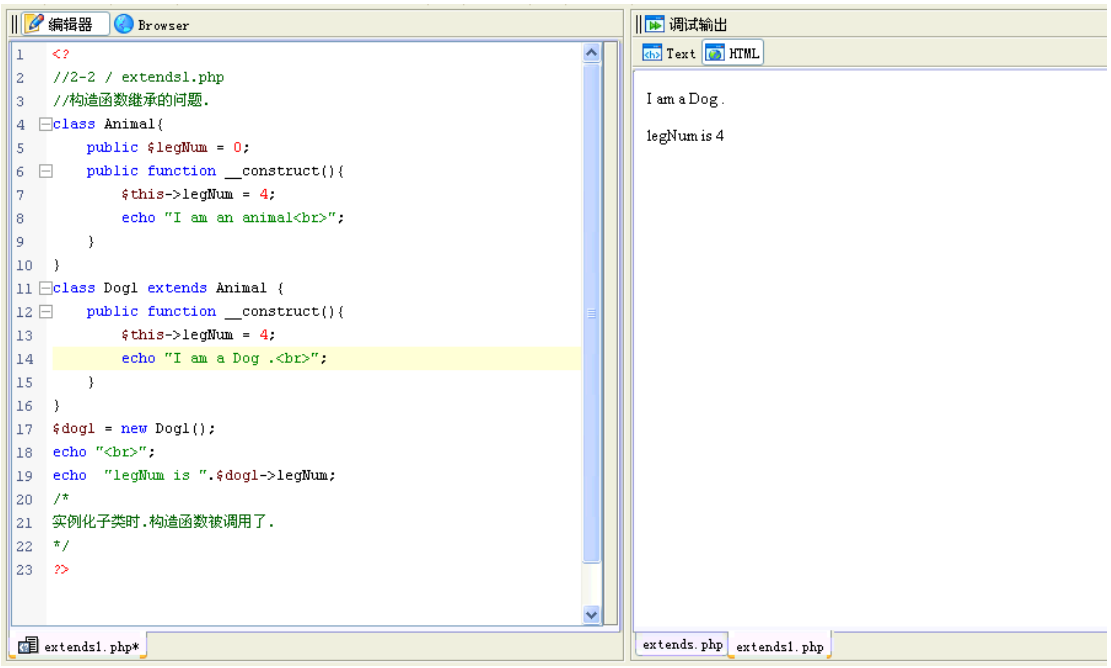
子类可以拥有与父类不同的参数数量。(这点与 java 不同，PHP 是弱类型语言。)



构造函数重写

下面这个例子中，父类和子类都有自己的构造函数，当子类被实例化时，子类的构造函数被调用，而父类的构造函数没有被调用，请对比第一节的构造函数继承。

例子：2-2/ extends1.php



（注：这点和 Java 不同，在 java 中构造函数是不能被继承的，而且子类实例化时，子类的构造函数被调用，父类的构造函数也会调用。）

2.4 this 关键字

- PHP5 中为解决变量的命名冲突和不确定性问题，引入关键字“$this”代表其所在当前对象。
- $this 在构造函数中指该构造函数所创建的新对象
- 在类中使用当前对象的属性和方法，必须使用$this->取值。
- 方法内的局部变量，不属于对象，不使用$this 关键字取值。

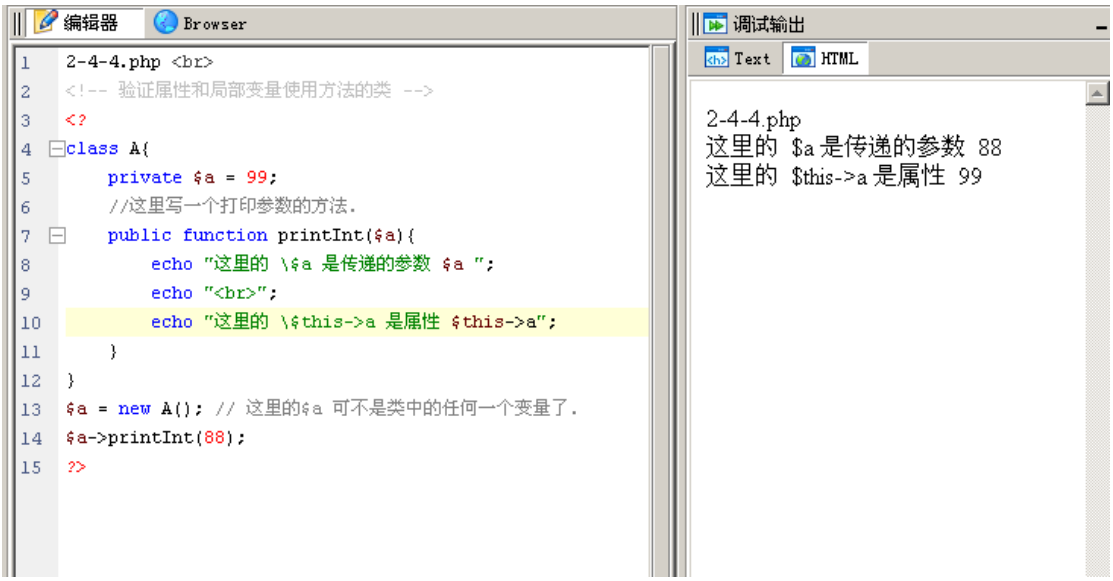
局部变量和全局变量与 $this 关键字

例 2-4-4.php

使用当前对象的属性必须使用$this 关键字。

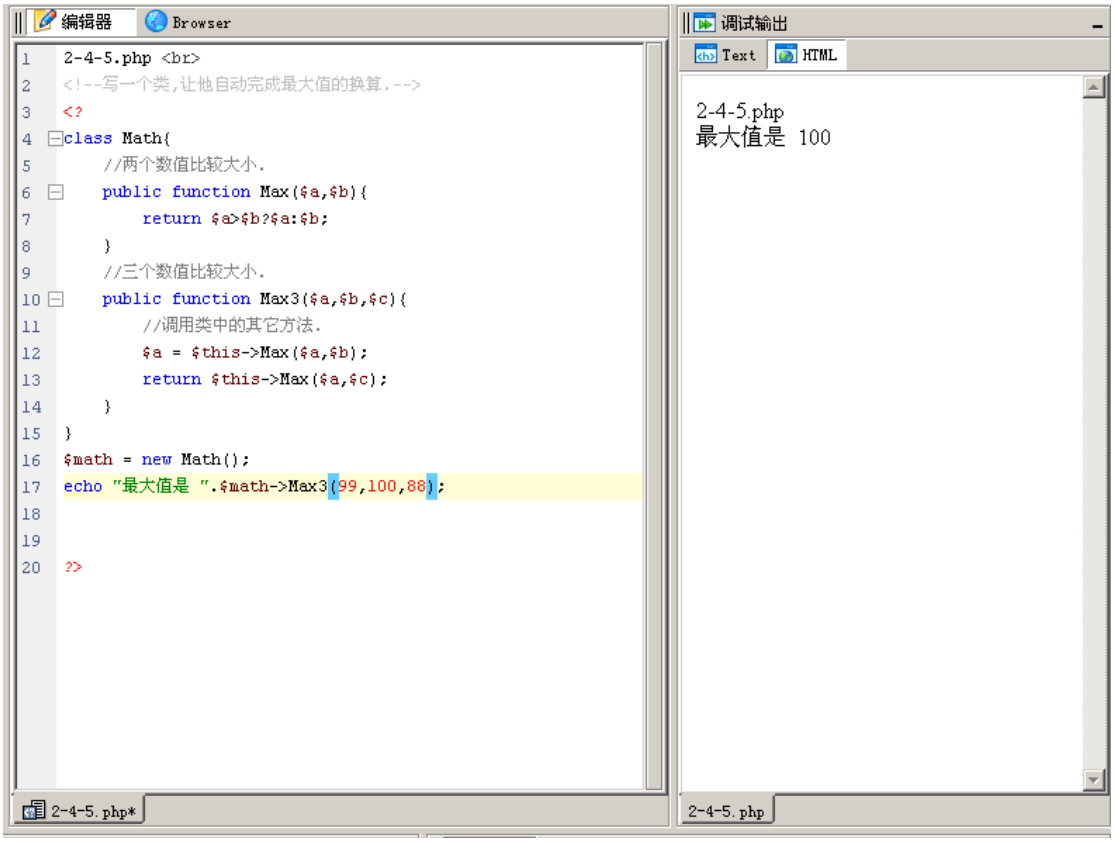
局部变量的只在当前对象的方法内有效，所以直接使用。

注意：局部变量和属性可以同名，但用法不一样。在使用中，要尽量避免这样使用，以免混淆。



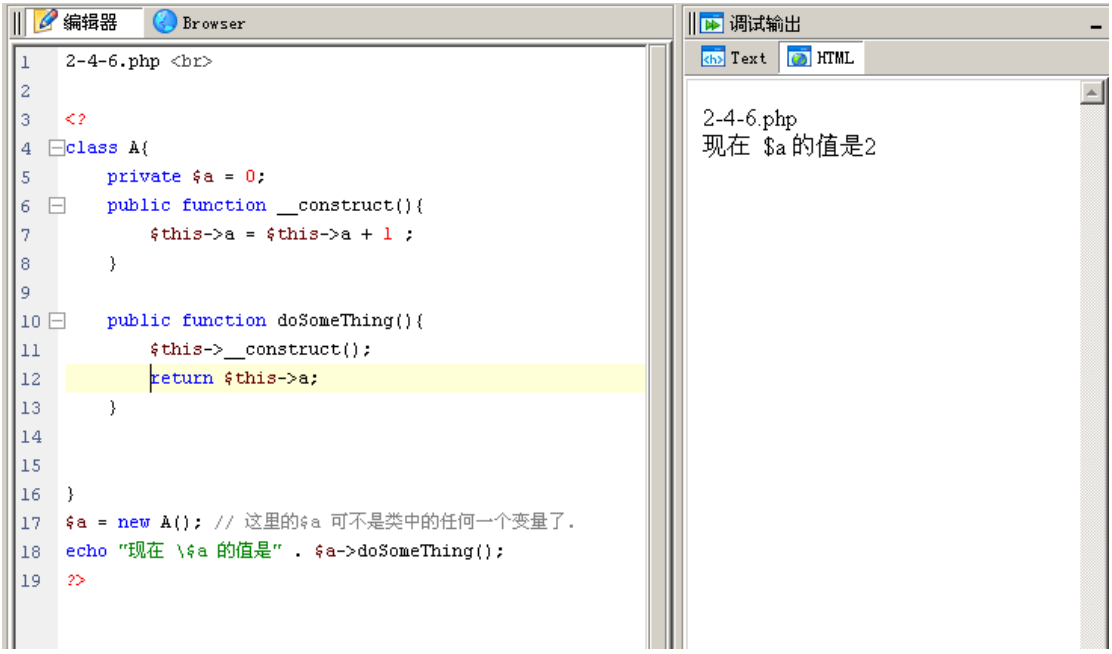
用$this 调用对象中的其它方法

例：2-4-5



使用$this 调用构造函数

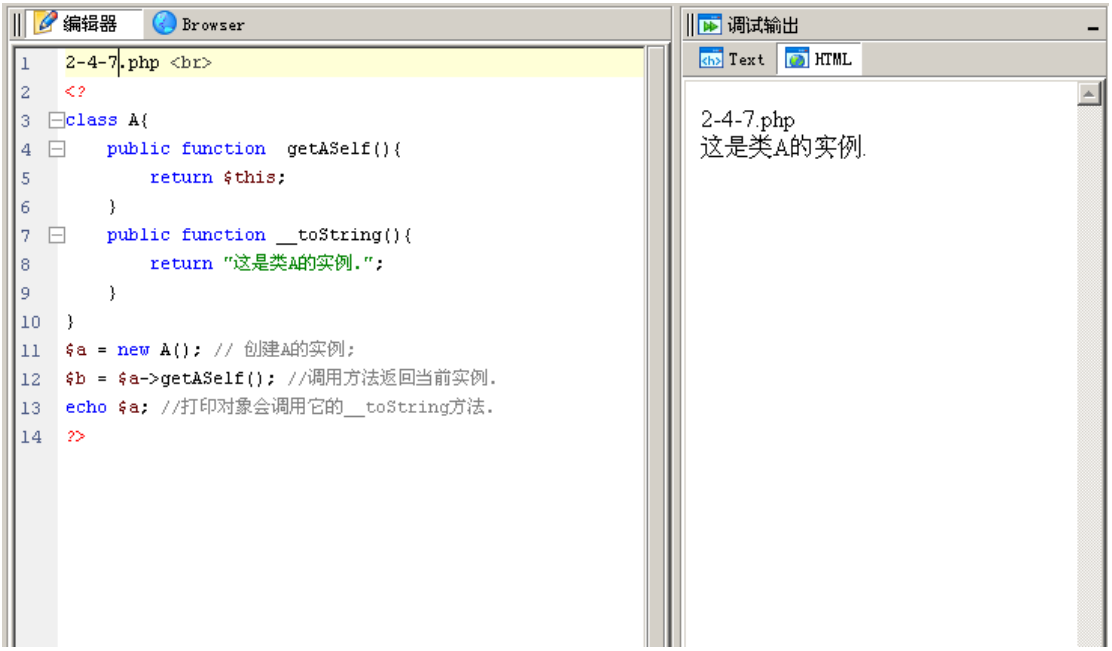
例：2-4-6　调用构造函数和析构函数的方法一致。



$this 到底指的什么？

$this 就是指当前对象，我们甚至可以返回这个对象使用 $this

例 2-4-7



通过 $this 传递对象

编辑器 Browser

```
2-4-8.php <br>
<!-- 通过$this 传递对象
在这个例子中,我们写一个根据不同的年龄发不同工资的类.
我们设置处理年龄和工资的业务模型为一个独立的类.
-->
<?
class User{
    private $age ;
    private $sal ;
    private $payoff ; //声明全局属性.

    //构造函数,中创建Payoff的对象.
    public function __construct(){
        $this->payoff = new Payoff();
    }
    public function getAge(){
        return $this->age;
    }
    public function setAge($age){
        $this->age = $age;
    }
    // 获得工资.
    public  function getSal(){
        $this->sal =  $this->payoff->figure($this);
        return $this->sal;
    }
//这是对应工资与年龄关系的类.
class Payoff{
    public function figure($a){
        $sal =0;
        $age = $a->getAge();
        if($age >80 || $age <16 ){
            $sal = 0;
        }elseif ($age > 50){
            $sal = 1000;
        }else{
            $sal = 800;
        }
        return $sal;
    }
}
//实例化User
$user = new User();

$user->setAge(55);
echo $user->getAge()."age ,his sal is " . $user->getSal();
echo "<br>";

$user->setAge(20);
echo $user->getAge()."age , his sal is " . $user->getSal();
echo "<br>";

$user->setAge(-20);
echo $user->getAge()."age , his sal is " . $user->getSal();
echo "<br>";

$user->setAge(150);
echo $user->getAge()."age , his sal is " . $user->getSal();


?>
```

调试输出

Text HTML

```
2-4-8.php
55age ,his sal is 1000
20age , his sal is 800
-20age , his sal is 0
150age , his sal is 0
```

2.5 parent::关键字

•PHP5 中使用 parent::来引用父类的方法。

parent:: 可用于调用父类中定义的成员方法。

parent::的追溯不仅于直接父类。

通过 parent::调用父类方法

例 2-5-1

编辑器 Browser

```
1  2-5-1.php<br>
2  <!-- 声明一个员工类,经理类继承自员工类 -->
3  <?
4  class employee{
5      protected  $sal=3000;
6      public function getSal(){
7          $this->sal = $this->sal + 1200;
8          return $this->sal ;
9      }
10 }
11
12 class Manager extends employee {
13     //如果想让经理在员工工资的基础上多发1500元.
14     //必须先调用父类的getSal()方法.
15     public function getSal(){
16         parent::getSal();// 这里调用了父类的方法.
17         $this->sal = $this->sal + 1500;
18         return $this->sal ;
19     }
20 }
21 $emp = new employee();
22 echo "普通员工的工资是 " . $emp->getSal();
23 echo "<br>";
24
25 $manager = new Manager();
26 echo "经理的工资是: " . $manager->getSal();
27 ?>
```

2-5-1.php

调试输出

Text HTML

```
2-5-1.php
普通员工的工资是 4200
经理的工资是: 5700
```

2-5-1.php

父类的 private 属性

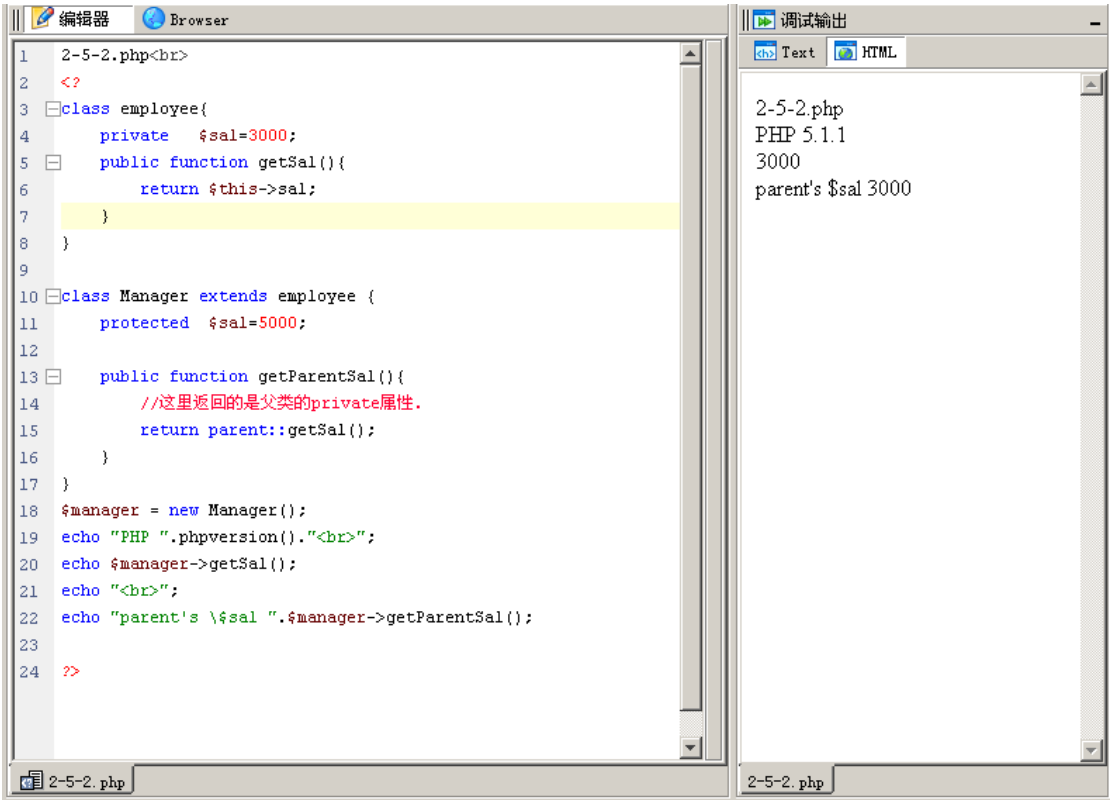
这个东西解释起来十分的不爽。

Private 属性是不能被继承的，如果父类有私有的属性。那么父类的方法只为父类的私有属性服务。

例：2-5-2

下面的例子看起来很奇怪，在子类中重新定义了一个属性$sal

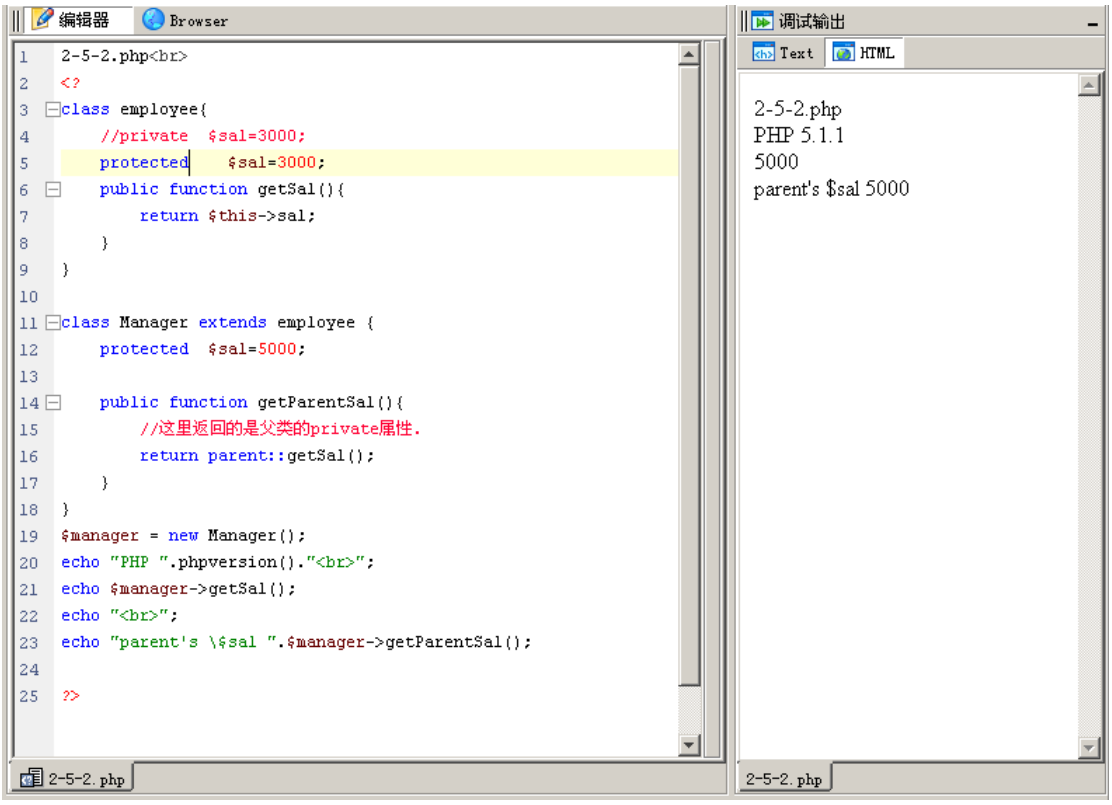
系统却返回了父类的属性。




```
1  2-5-2.php<br>
2  <?
3  class employee{
4      private   $sal=3000;
5      public function getSal(){
6          return $this->sal;
7      }
8  }
9
10 class Manager extends employee {
11     protected  $sal=5000;
12
13     public function getParentSal(){
14         //这里返回的是父类的private属性.
15         return parent::getSal();
16     }
17 }
18 $manager = new Manager();
19 echo "PHP ".phpversion()."<br>";
20 echo $manager->getSal();
21 echo "<br>";
22 echo "parent's \$sal ".$manager->getParentSal();
23
24 ?>
```

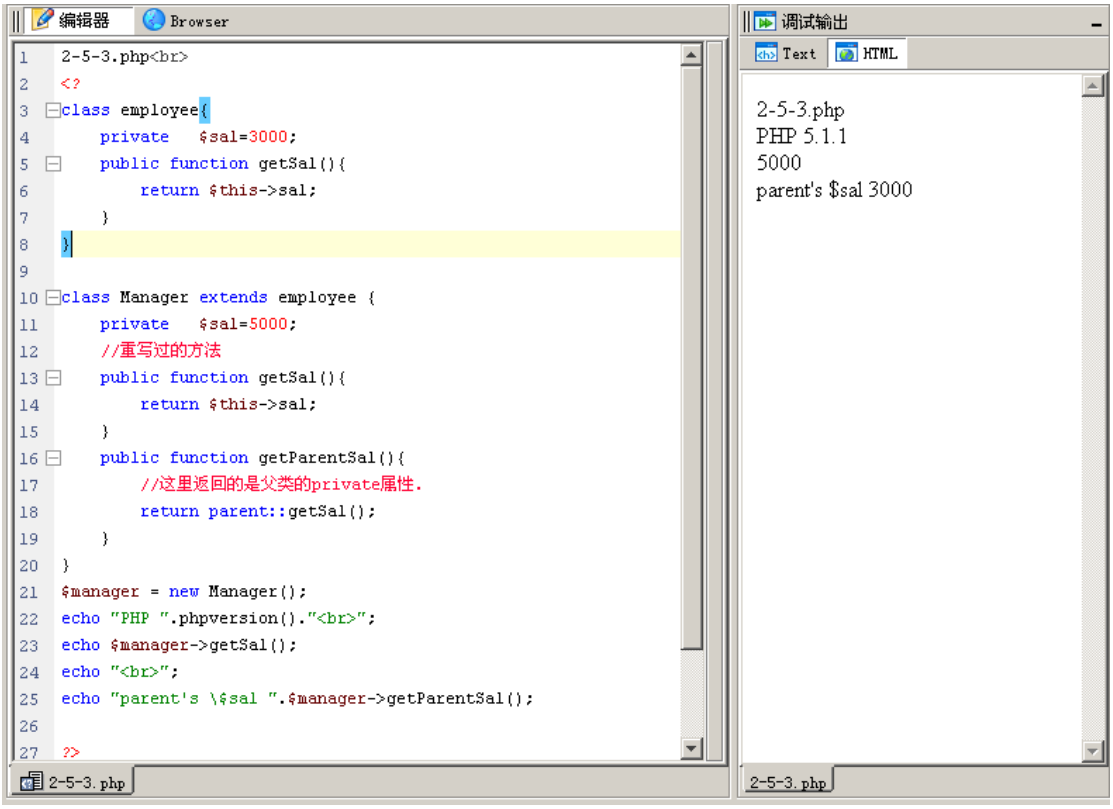
如果父类中的属性被子类重写了。结果是这样的。

注意 第 5 行的属性定义变成了 protected。



子类中重写的方法对当前 private 有效。

例：2-5-3.php



打开 zend 调试状态看看，内存中的情况。注意最下面，有两个$sal 。分别是 3000 和 5000 。
例：2-5-4.php

将父类的属性$sal 改成 protected ,子类重写了父类的属性。
在内存中只有一个 $sal .
例：2-5-5.php

如果你学过 java，你会觉得这一切都是很难理解的。
在 Java 中当子类被创建时，父类的属性和方法在内存中都被创建，甚至构造函数也要被调用。
PHP5 不是这样，PHP5 调用父类用的是 parent:: 而不是 parent-> ，这足以说明 PHP5 不想在内存中让父类也被创建。PHP5 想让继承变的比 Java 更简单。
适应下就好。

这样调用会让 PHP5.1.1 溢出。新版不知道有没有问题。
例：2-5-10.php

第 13 行改成这样就好了。注意比较。

```
return parent:: getSal();
```

这样的代码引起了递归操作，子类调用父类的方法，父类又调用子类方法。

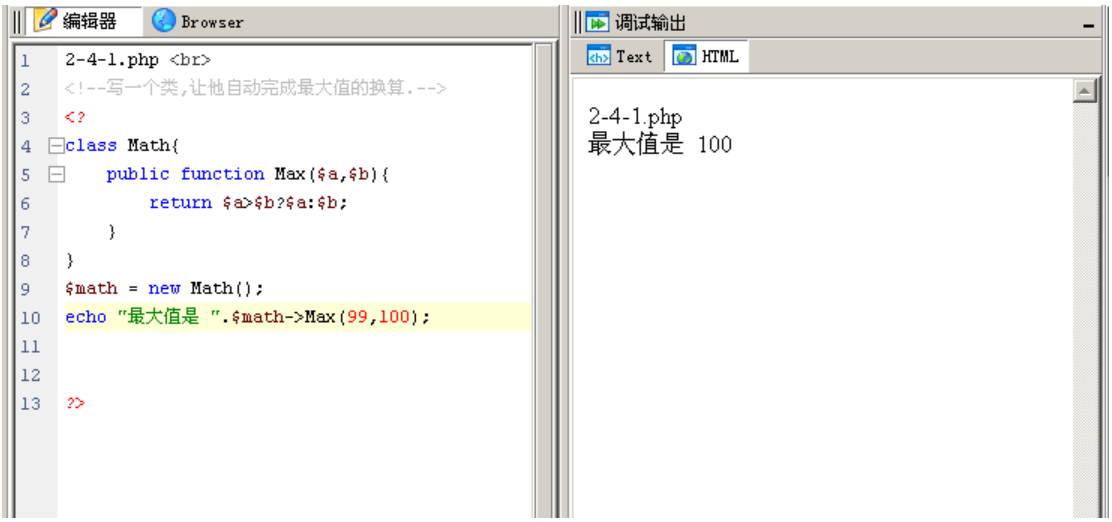
```
return parent::$this->getSal();
```

2.5 重载 Overload

- 当类中的方法名相同时，称为方法的重载(overload)
- 重载是 Java 等面向对象语言中重要的一部分。
- 在 PHP5 中不支持重载。

在 PHP5 中不支持重载。

例 2-4-1
先写一个取最大值的类。

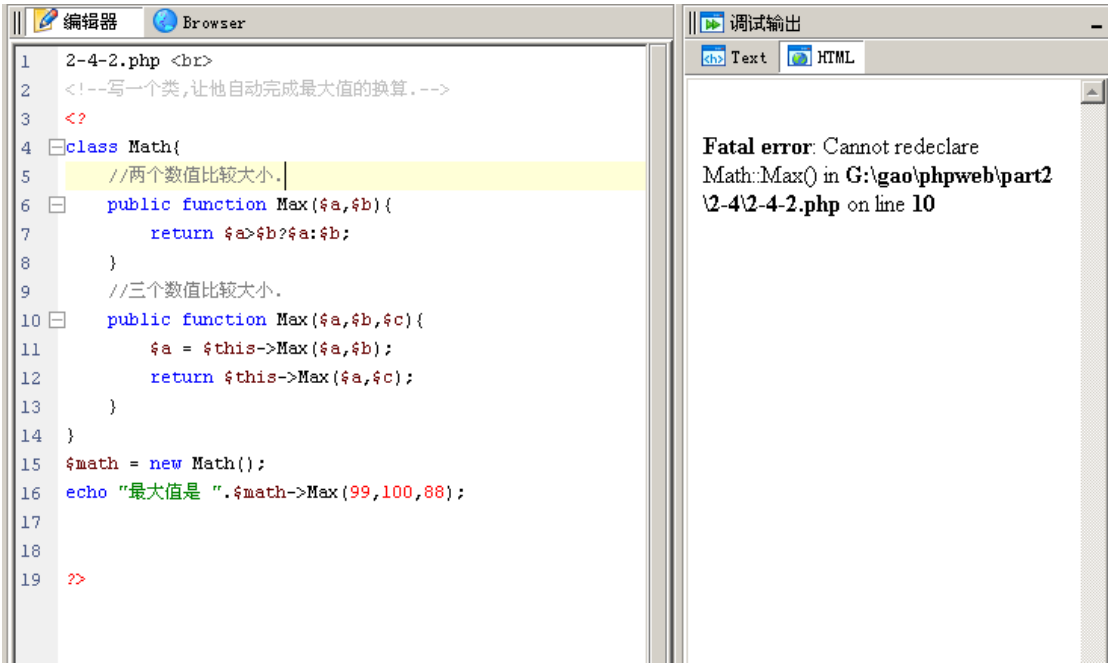


例：2-4-2

如果传递过来三个数值。如何计算？

下面的写法，在 Java 中是太平常不过了。但是在 PHP5 中，不能这样。

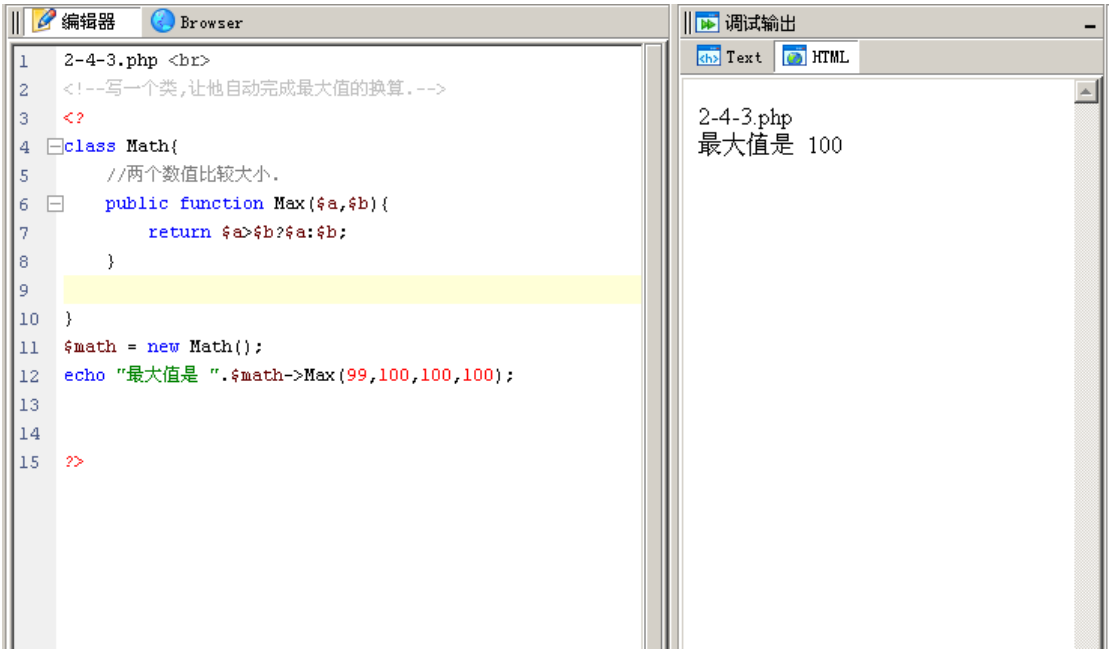
PHP5 不支持有多个相同名称的方法，也就是不支持重载。



例 2-4-3

对于一个方法，缺少参数时候会报错。

当参数太多的时候，PHP 就当什么都没有看到。程序可以正常运行。



2.7 实例

简单写一个实例，这个例子只是本章内容的表达。在了解更多面向对象内容后，相信你能写出更合适的表达代码。

在上一章，我们写了一个建立用户 user 类，直接使用 user 类读取用户信息的类。假设我们又有了新的需求。

- 任何用户都可以查看别的用户的信息，当然不能看到别人的密码。
- 任何用户都可以修改自己的密码。

于是我们对第一章的类做些改动，首先我们在 userInfo 类中，将获得密码的方法隐藏。

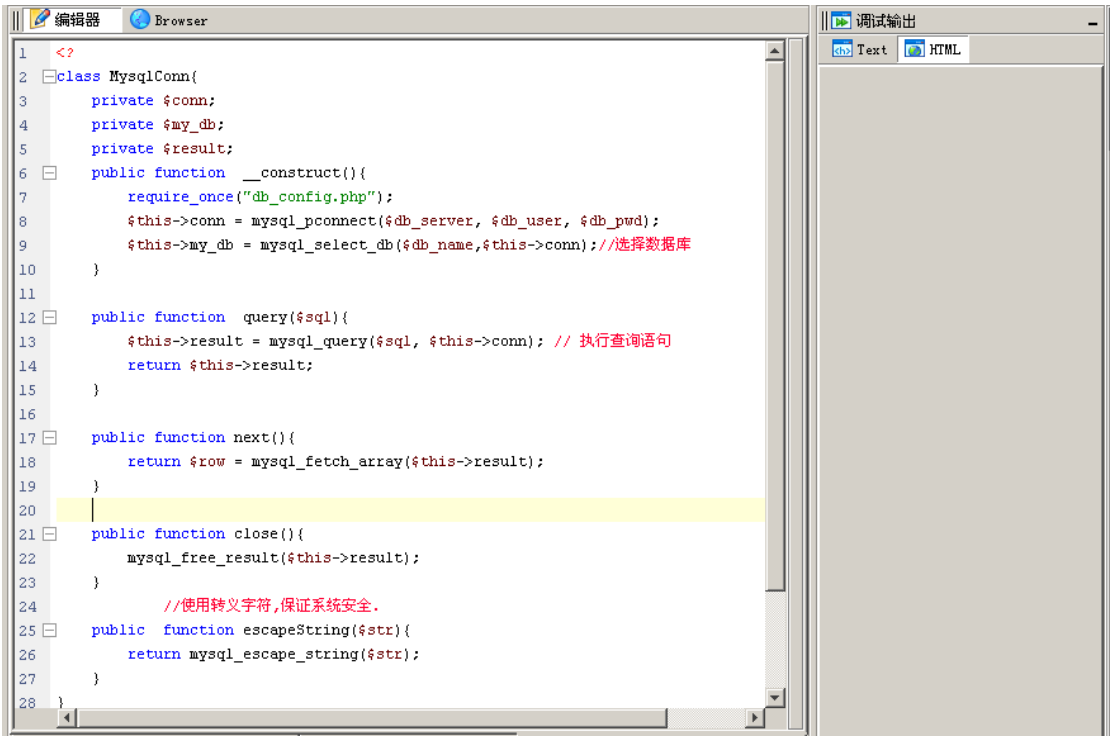
我们再写一个 UserChange 类继承自 userInfo，在 UserChange 中增加修改密码的方法。并将获取密码的方法重写为 public 权限。

这样，在你的页面中，就可以创建两种 user。一种是只能看到信息不能看到密码、不能修改密码的 userInfo 的实例。另外一种是比 userInfo 功能更强的 UserChange 类，这个实例可以修改密码，可以获得密码。

在合适的位置创建不同的 user，就是你的业务逻辑的内容了。

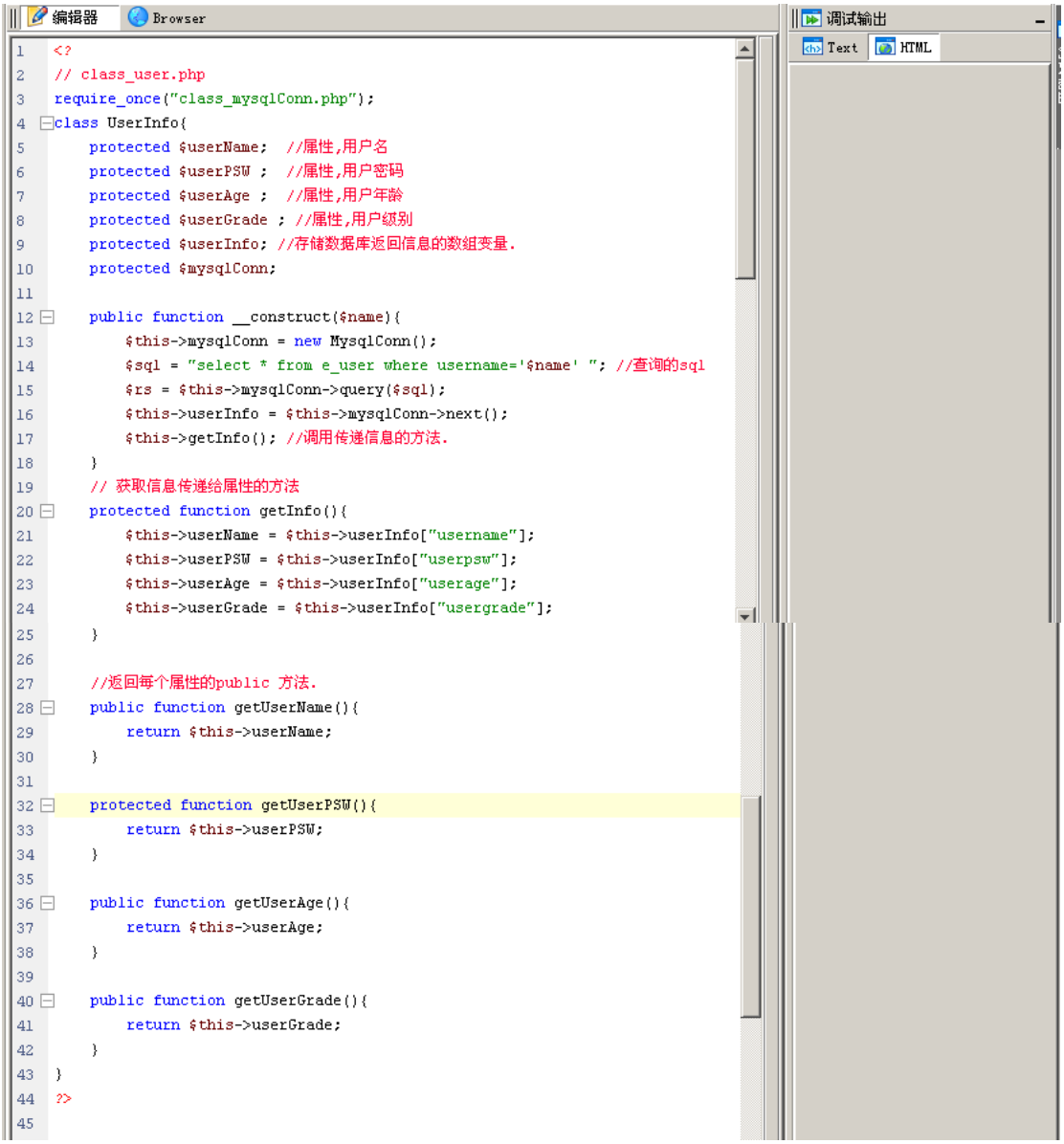
同时，我们独立出一个数据库连接类，数据库连接类比较完善的网上有很多，大家学习完毕面向对象后，自己也可以写出更完善的数据库类。

Mysql 连接类。




```php
<?
class MysqlConn{
    private $conn;
    private $my_db;
    private $result;
    public function  __construct(){
        require_once("db_config.php");
        $this->conn = mysql_pconnect($db_server, $db_user, $db_pwd);
        $this->my_db = mysql_select_db($db_name,$this->conn);//选择数据库
    }

    public function  query($sql){
        $this->result = mysql_query($sql, $this->conn); // 执行查询语句
        return $this->result;
    }

    public function next(){
        return $row = mysql_fetch_array($this->result);
    }

    public function close(){
        mysql_free_result($this->result);
    }
        //使用转义字符,保证系统安全.
    public  function escapeString($str){
        return mysql_escape_string($str);
    }
}
```

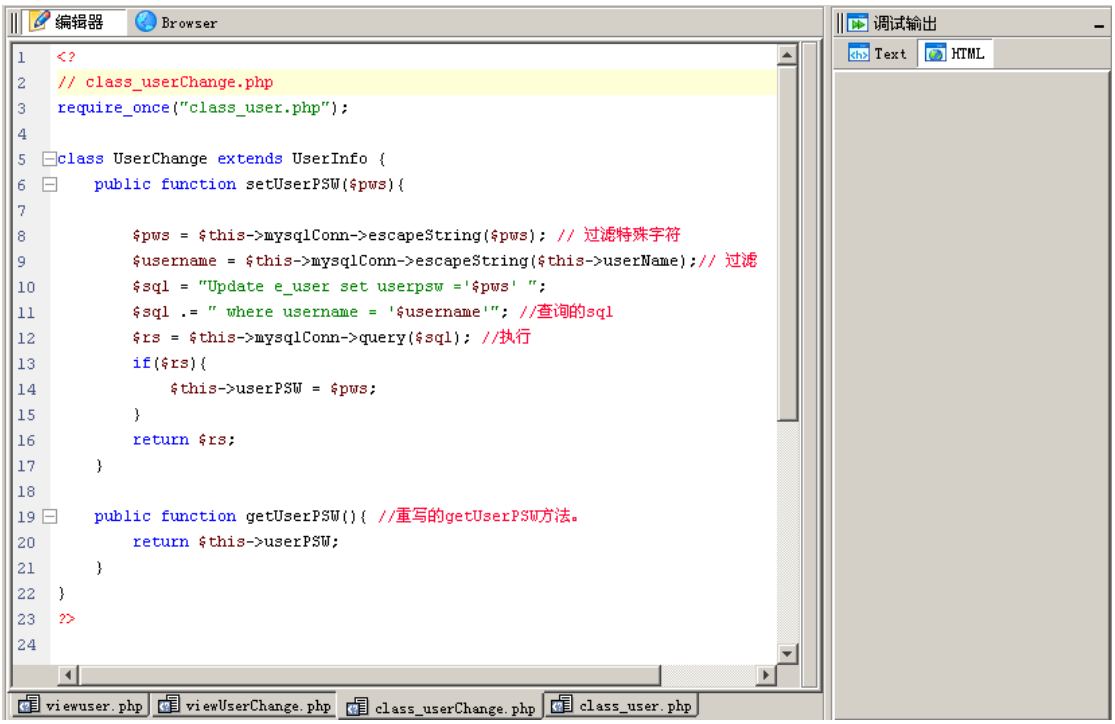
父类 User 类




```php
<?
// class_user.php
require_once("class_mysqlConn.php");
class UserInfo{
    protected $userName;  //属性,用户名
    protected $userPSW ;  //属性,用户密码
    protected $userAge ;  //属性,用户年龄
    protected $userGrade ; //属性,用户级别
    protected $userInfo; //存储数据库返回信息的数组变量.
    protected $mysqlConn;

    public function __construct($name){
        $this->mysqlConn = new MysqlConn();
        $sql = "select * from e_user where username='$name' "; //查询的sql
        $rs = $this->mysqlConn->query($sql);
        $this->userInfo = $this->mysqlConn->next();
        $this->getInfo(); //调用传递信息的方法.
    }
    // 获取信息传递给属性的方法
    protected function getInfo(){
        $this->userName = $this->userInfo["username"];
        $this->userPSW = $this->userInfo["userpsw"];
        $this->userAge = $this->userInfo["userage"];
        $this->userGrade = $this->userInfo["usergrade"];
    }

    //返回每个属性的public 方法.
    public function getUserName(){
        return $this->userName;
    }

    protected function getUserPSW(){
        return $this->userPSW;
    }

    public function getUserAge(){
        return $this->userAge;
    }

    public function getUserGrade(){
        return $this->userGrade;
    }
}
?>

```

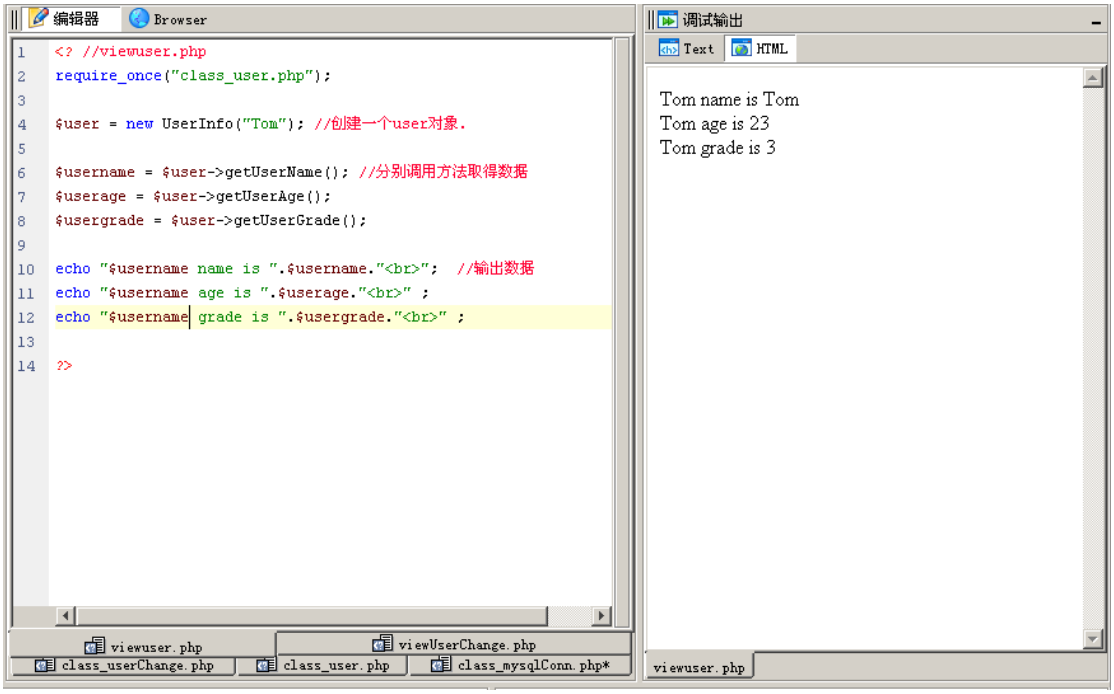
子类 UserInfo 类

添加了修改密码的方法，重写并公开了获取密码的方法。

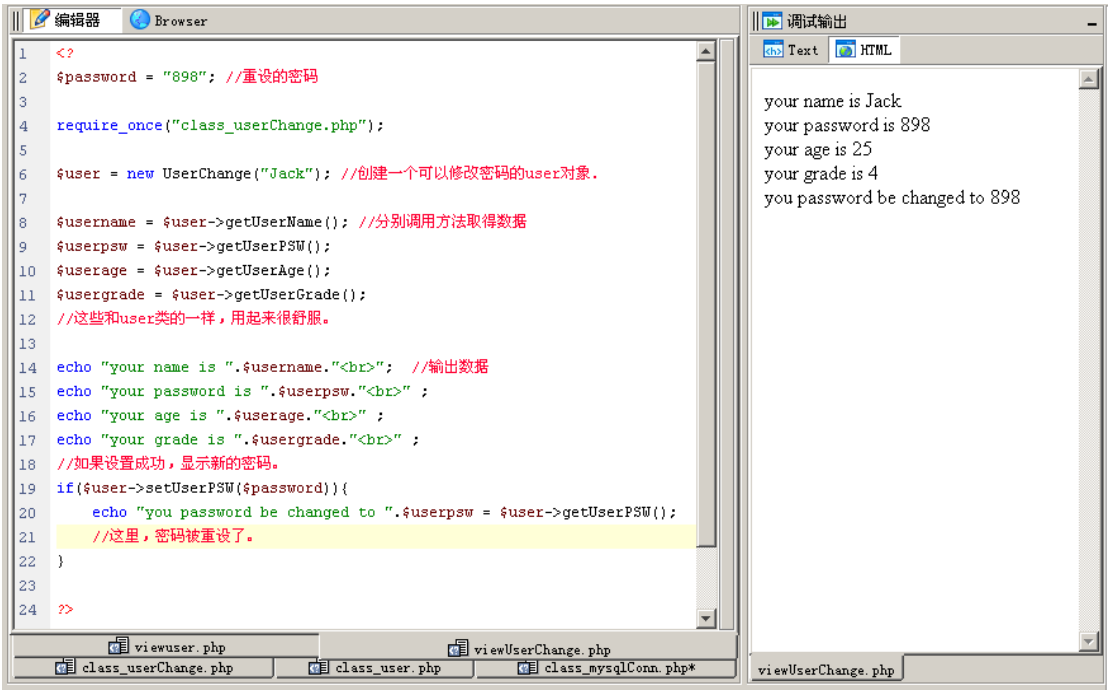



```php
<?
// class_userChange.php
require_once("class_user.php");

class UserChange extends UserInfo {
    public function setUserPSW($pws){

        $pws = $this->mysqlConn->escapeString($pws); // 过滤特殊字符
        $username = $this->mysqlConn->escapeString($this->userName);// 过滤
        $sql = "Update e_user set userpsw ='$pws' ";
        $sql .= " where username = '$username'"; //查询的sql
        $rs = $this->mysqlConn->query($sql); //执行
        if($rs){
            $this->userPSW = $pws;
        }
        return $rs;
    }

    public function getUserPSW(){ //重写的getUserPSW方法。
        return $this->userPSW;
    }
}
?>
```

在任何位置都可以放心使用 userInfo 类。



可以重设密码的 Userchange 类的实例。



本章介绍了，PHP5 面向对象关于继承、重载、重写、$this 关键字，通过 parent 调用父类方法等 PHP5 面向对象初步的知识。

下一章，面向对象设计进阶。希望大家支持。

第三章 PHP5 面向对象高级类特性

这一周来正好白天没课，赶出来这章。

宝宝出生将近一个月，快 10 斤重了。
宝宝快张大，你想编程么？宝宝妈说，做这个行业太累了。

刀客羽朋
于石家庄 2006-11-23

目录

3.1 static 变量、方法

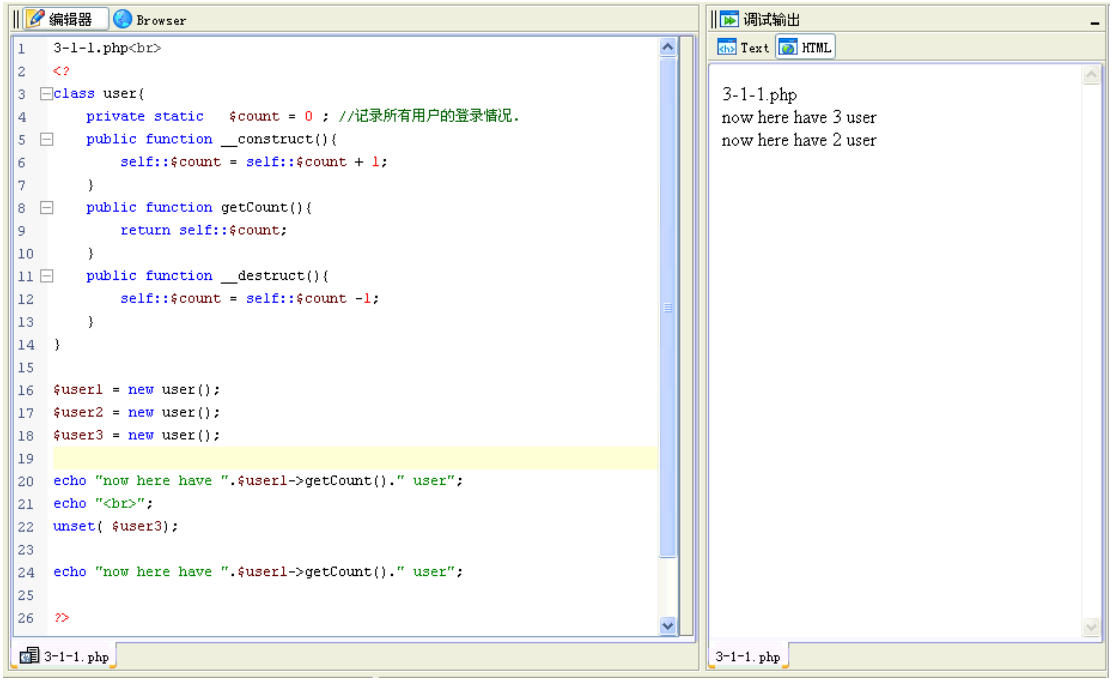
- static 关键字用来修饰属性、方法，称这些属性、方法为静态属性、静态方法。
- static 关键字声明一个属性或方法是和类相关的，而不是和类的某个特定的实例相关，因此，这类属性或方法也称为“类属性”或“类方法”
- 如果访问控制权限允许，可不必创建该类对象而直接使用类名加两个冒号“::”调用。

- **static** 关键字可以用来修饰变量、方法。
- 不经过实例化，就可以直接访问类中 static 的属性和 static 的方法。
- static 的属性和方法，只能访问 static 的属性和方法，不能类访问非静态的属性和方法。因为静态属性和方法被创建时，可能还没有任何这个类的实例可以被调用。
- static 的属性，在内存中只有一份，为所有的实例共用。
- 使用 self:: 关键字访问当前类的静态成员。

3.1.1 静态属性公用特性

- 一个类的所有实例，共用类中的静态属性。
- 也就是说，在内存中即使有多个实例，静态的属性也只有一份。

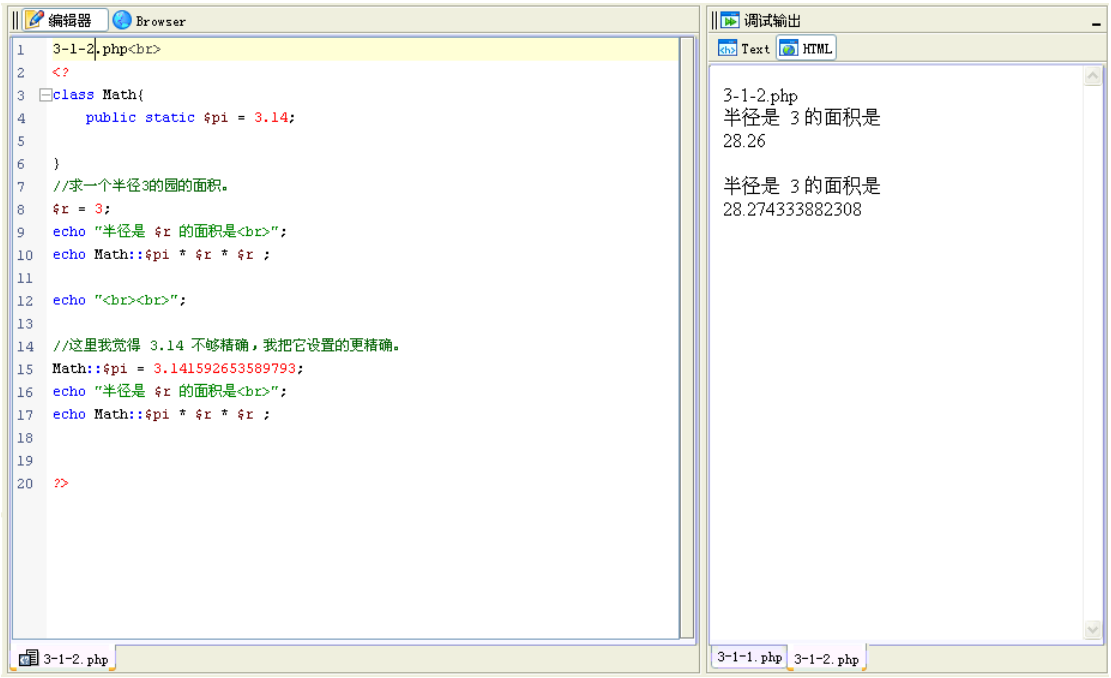
下面例子中的设置了一个计数器$count 属性，设置 private 和 static 修饰。
这样，外界并不能直接访问$count 属性。而程序运行的结果我们也看到多个实例在使用同一个静态的$count 属性。
例 3-1-1.php



3.1.2 静态属性直接调用

- 静态属性不需要实例化就可以直接使用，在类还没有创建时就可以直接使用。
- 使用的方式是 类名::静态属性名

例 3-1-2.php



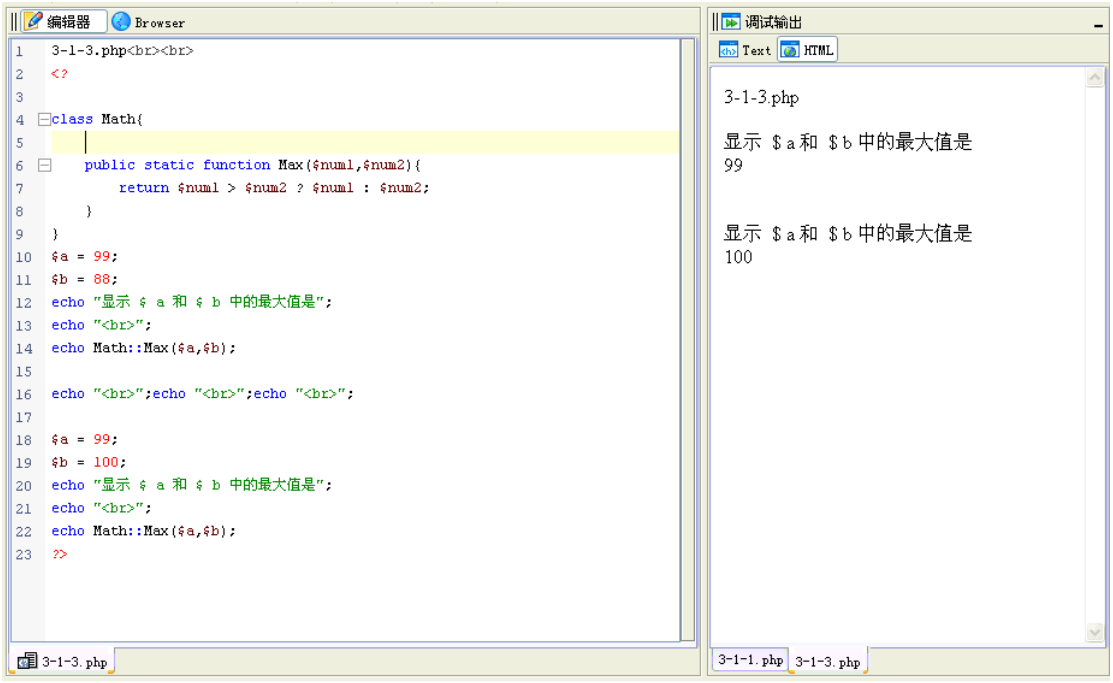
类没有创建，静态属性就可以直接使用。
那静态属性在什么时候在内存中被创建？
在 PHP 中没有看到相关的资料。引用 Java 中的概念，来解释应该也具有通用性。
静态属性和方法，在类被调用时创建。

类没有创建，静态属性就可以直接使用。 那静态属性在什么时候在内存中被创建？
在 PHP 中没有看到相关的资料。我们引用 Java 中的概念，来解释 PHP5 的静态修饰符，应该也具有通用性。
静态属性和方法，在类被调用时创建。
类被调用，是指类被创建或者类中的任何静态成员被调用。

3.1.3 静态方法

- 静态方法不需要所在类被实例化就可以直接使用。
- 使用的方式是 类名:: 静态方法名

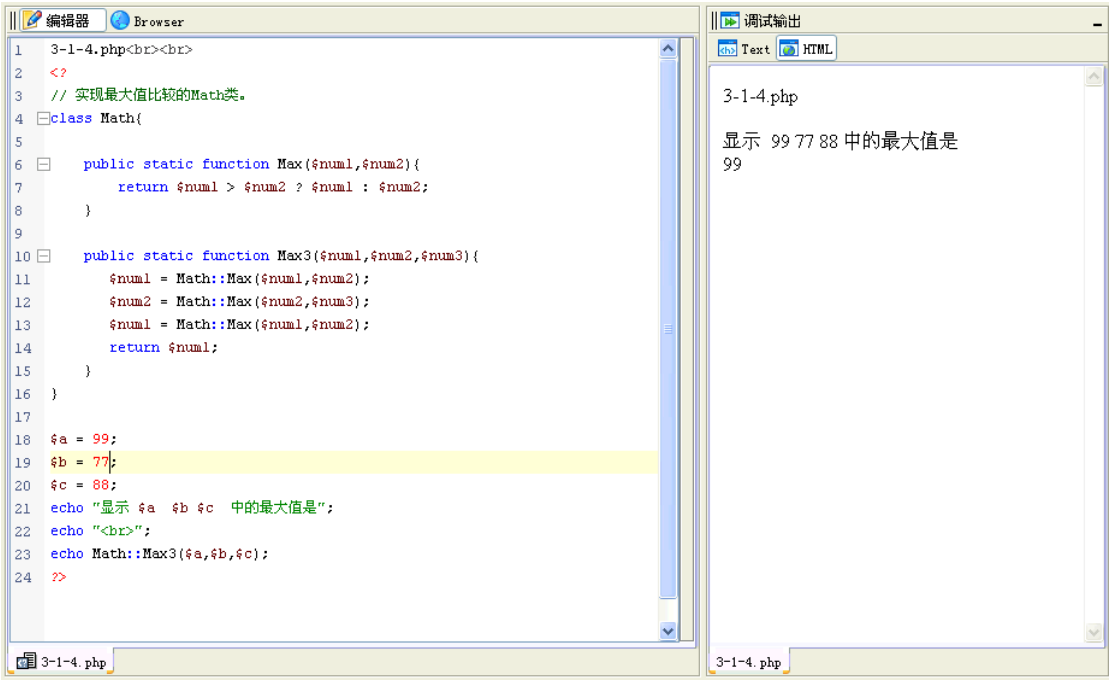
下面我们继续写这个 Math 类，用来进行数学计算。我们设计一个方法用来算出其中的最大值。
既然是数学运算，我们也没有必要去实例化这个类，如果这个方法可以拿过来就用就方便多了。
我们这只是为了演示 static 方法而设计的这个类。在 PHP 提供了 max() 函数比较数值。
例 3-2-3.php



3.1.4 静态方法如何调用静态方法

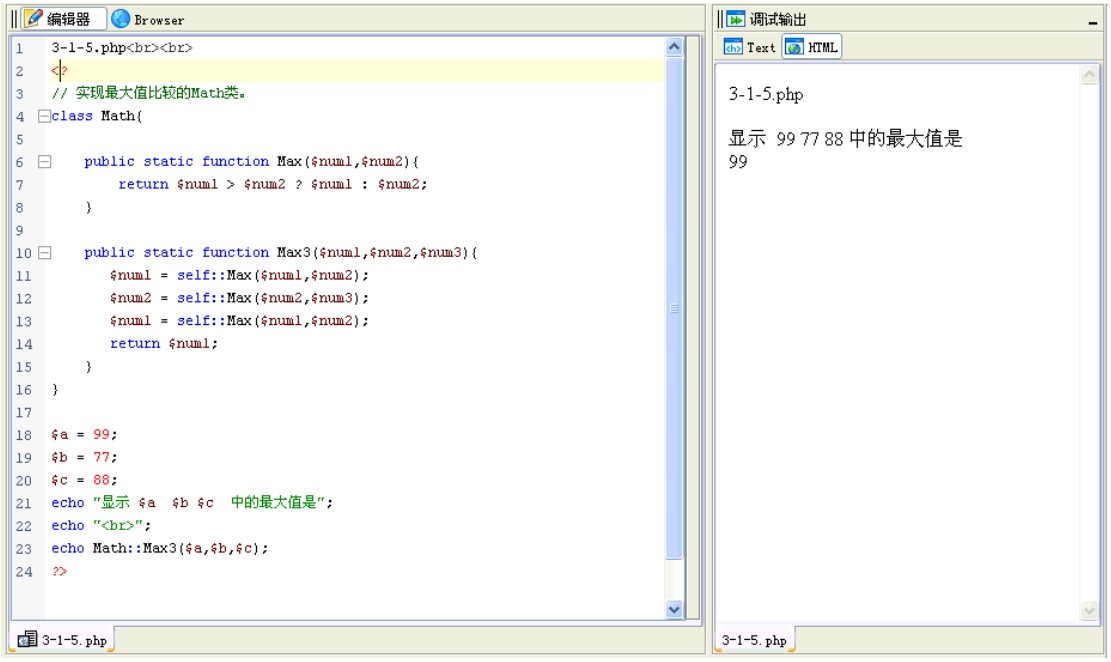
第一个例子，一个静态方法调用其它静态方法时，直接使用 类名

例 3-1-4.php



也可以使用 **self::** 调用当前类中的其它静态方法。（建议）

例 3-1-5.php

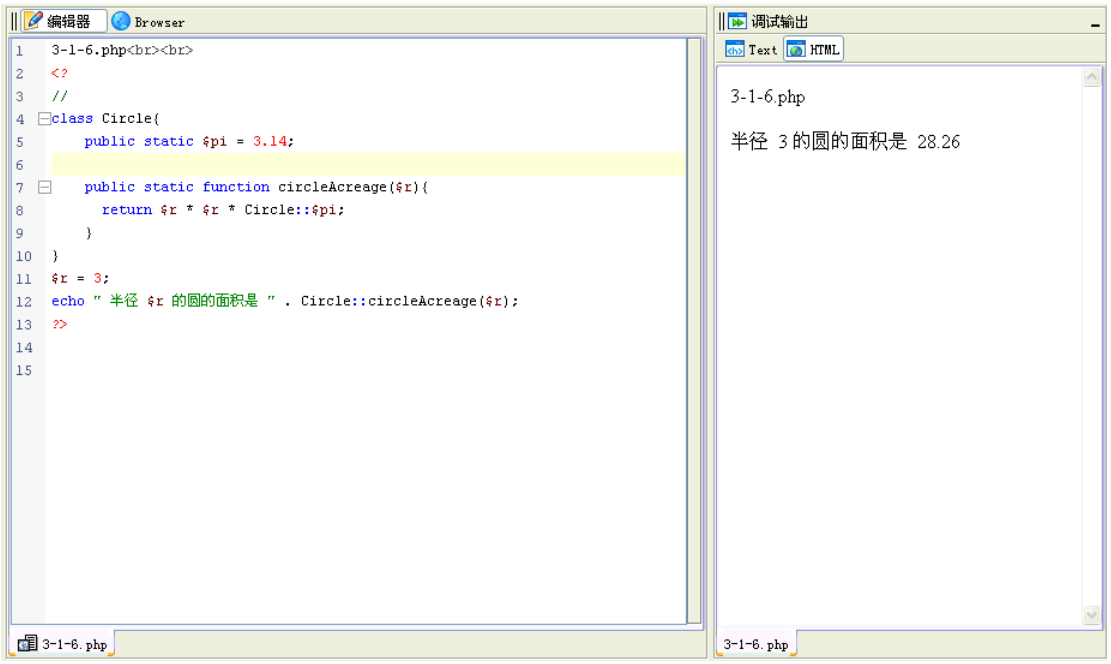


记住这个 self:: 它表示当前类的静态成员，与 $this 不同，$this 指当前对象。

3.1.5 静态方法调用静态属性

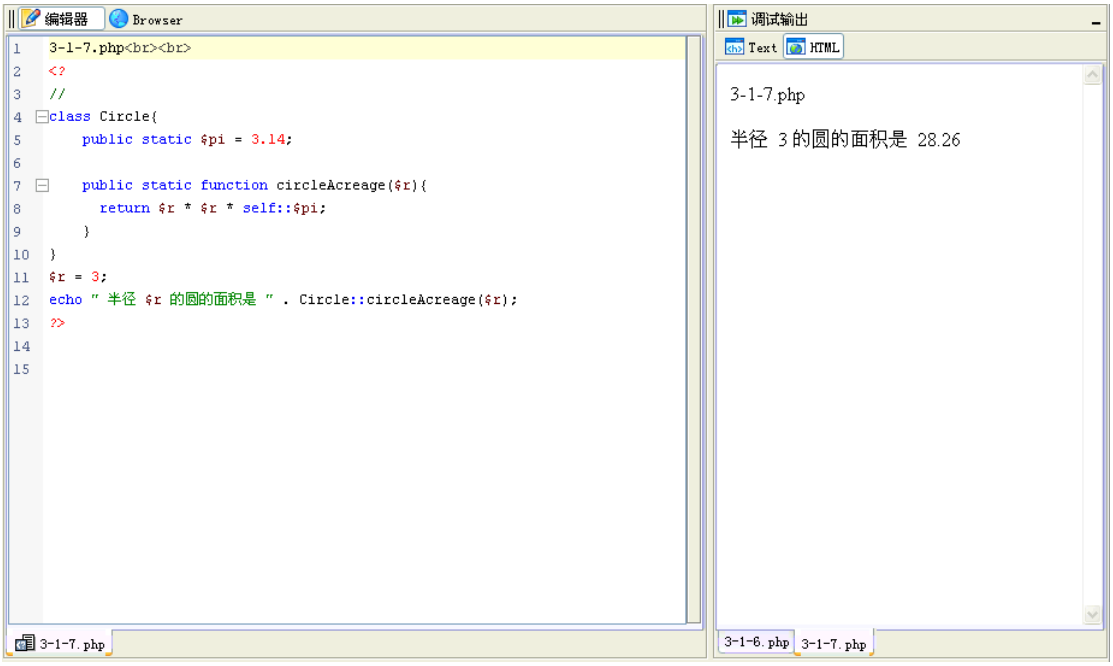
使用 **类名::静态属性名** 调用本类中的静态属性。

例 3-1-6.php



使用 **self::** 调用本类的静态属性。（建议）

例 3-1-7.php

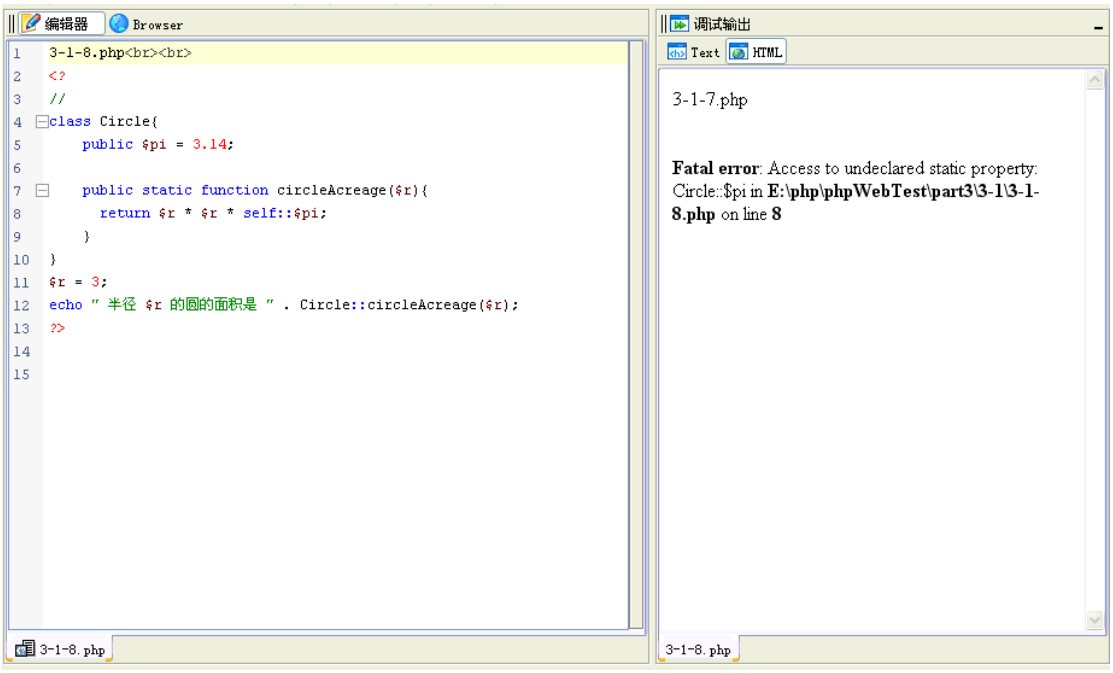


3.1.6 静态方法不能调用非静态属性

- 静态方法不能调用非静态的属性。

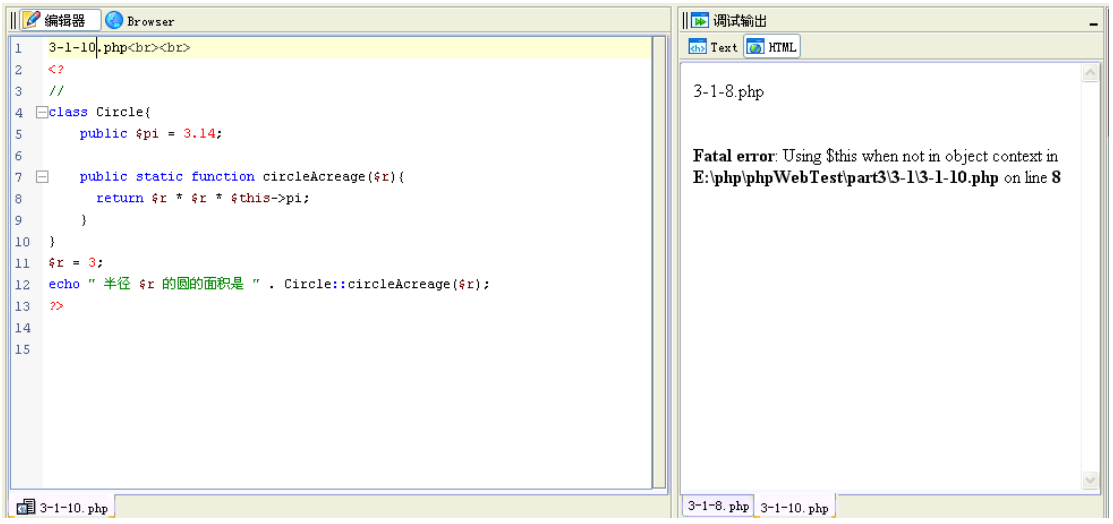
不能使用 self::调用非静态属性。

例 3-1-8.php



也不能使用 $this 获取非静态属性的值。

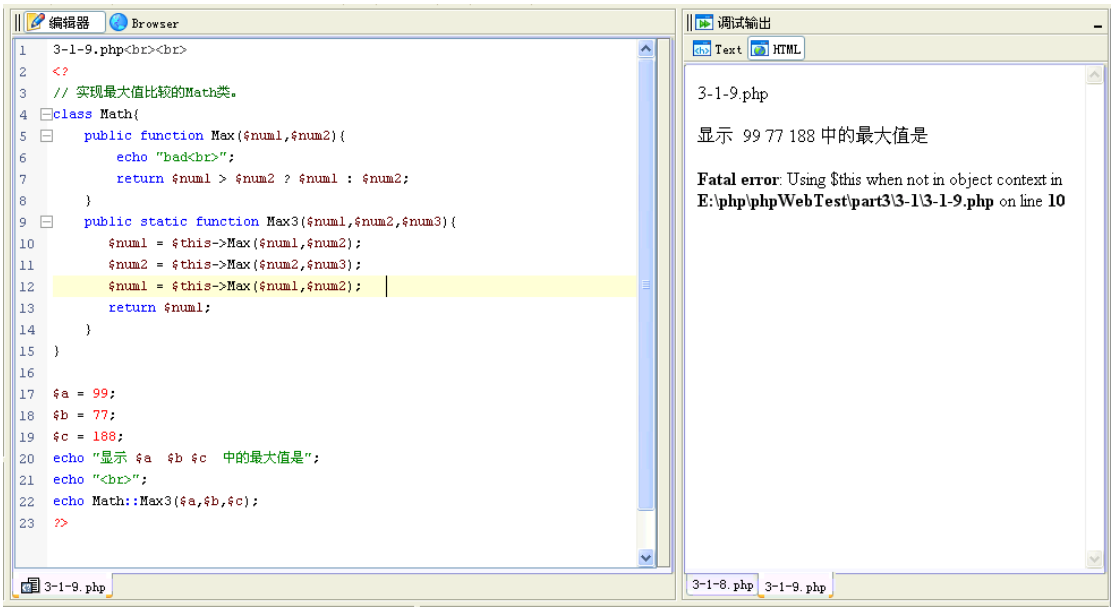
例 3-1-10



3.1.6 静态方法调用非静态方法

PHP5 中，在静态方法中不能使用 $this 标识调用非静态方法。

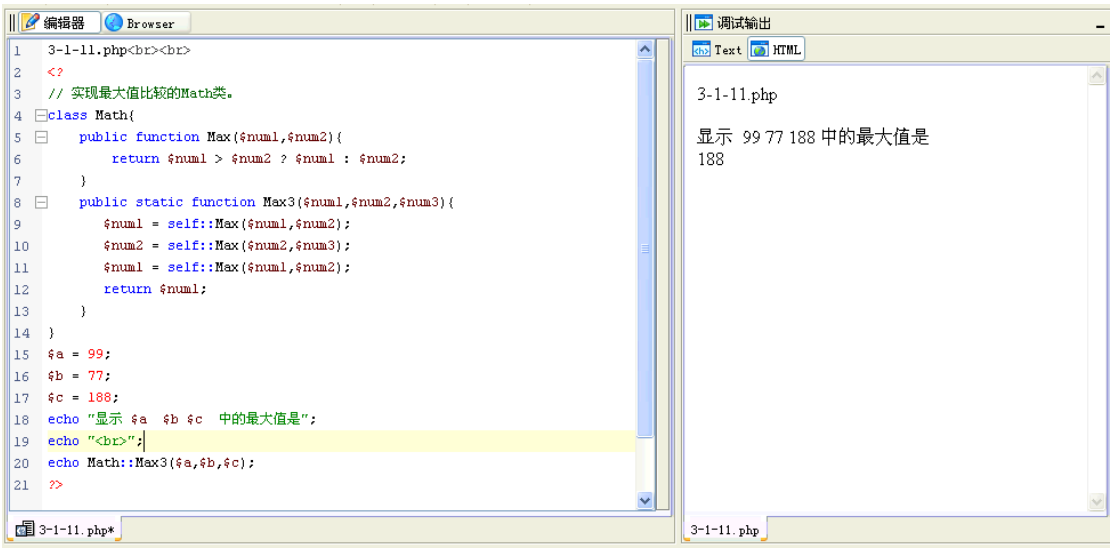
例 3-1-9



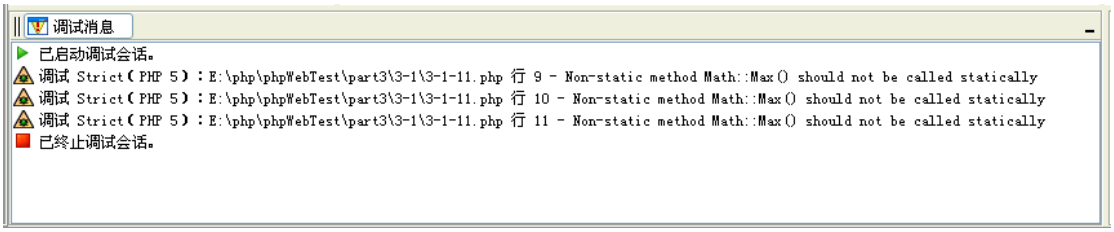
- 当一个类中有非静态方法被 self:: 调用时，系统会自动将这个方法转换为静态方法。

下面的代码被执行了，而且有结果。因为 Max 方法被系统转换为静态方法了。

例：3-1-11.php



在 Zend Studio 的调试部分显示出了这些错误的信息。

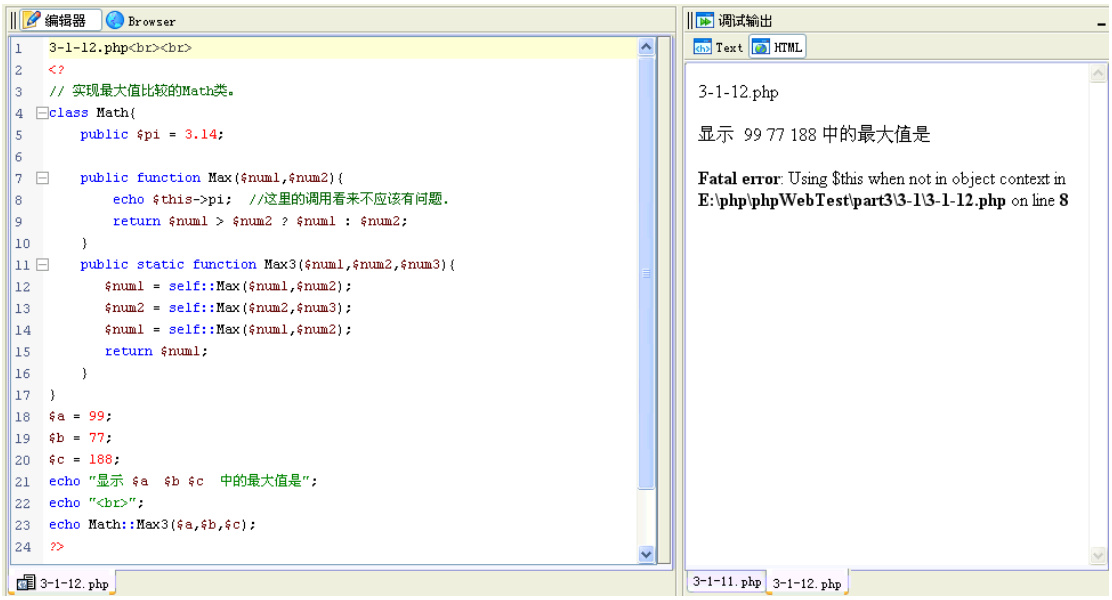


下面的例子中，我们让静态方法 Max3 用过 self::调用了非静态方法 Max，有让非静态方法 Max 通过$this 调用非静态属性 $pi 。

在运行是报出了错误，这个错误和前一个例子 3-1-9.php 一样，这次倒是非静态方法 Max 报出了静态方法调用非静态属性的错误。

这倒是证明了一点，在这里我们定义的非静态方法 Max 被系统自动转换成静态方法了。

例：3-1-12.php



3.1.7 设计模式之单件模式

单件模式要解决的问题就是“如何让这个类只有一个实例”。

我们的 web 应用中，大量使用了数据库连接，如果反复建立与数据库的连接必然消耗更多的系统资源。

我们如何解决这个问题，建立唯一的数据库连接是必要的方式。

我们又如何知道与这个数据库的连接是否已经建立？ 还是需要现在建立？

单件模式可以解决这个问题。

先假设我们需要一个类完成在内存中只有一份的功能，我们该如何做呢？
我们一步一步的使用前面学过的知识来写一个单件的例子。

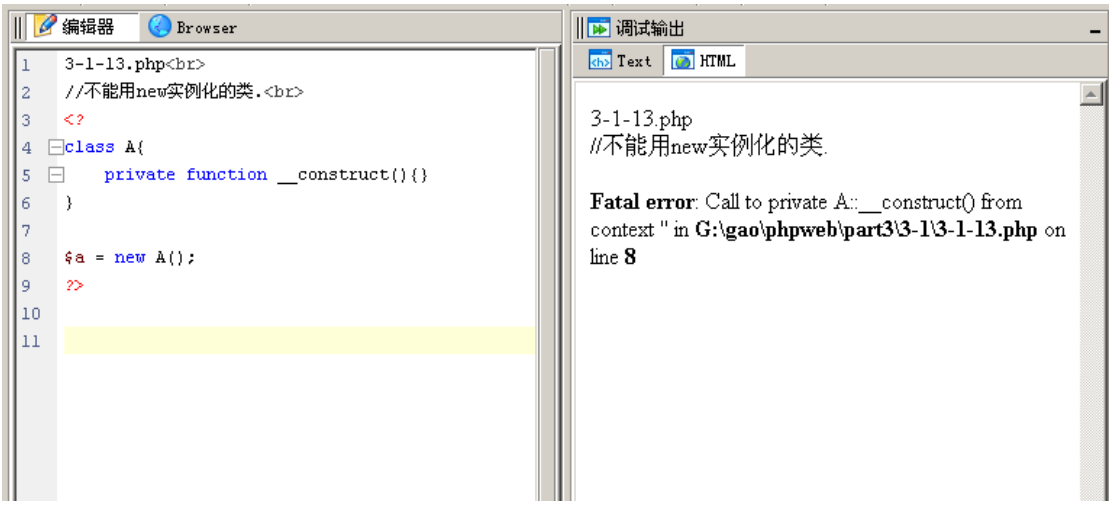
问题 1：

- 前面学过，每次用 new 类名 的方式，就可以创建一个对象。
- 我们必须禁止外部程序用 new 类名的方式来创建多个实例。

解决办法：

- 我们将构造函数设置成 private ，让构造函数只能在内部被调用，而外部不能调用。
- 这样，这个类就不能被外部用 new 的方式建立多个实例了。

例：3-1-13.php 不能被外部用 new 实例化的类。



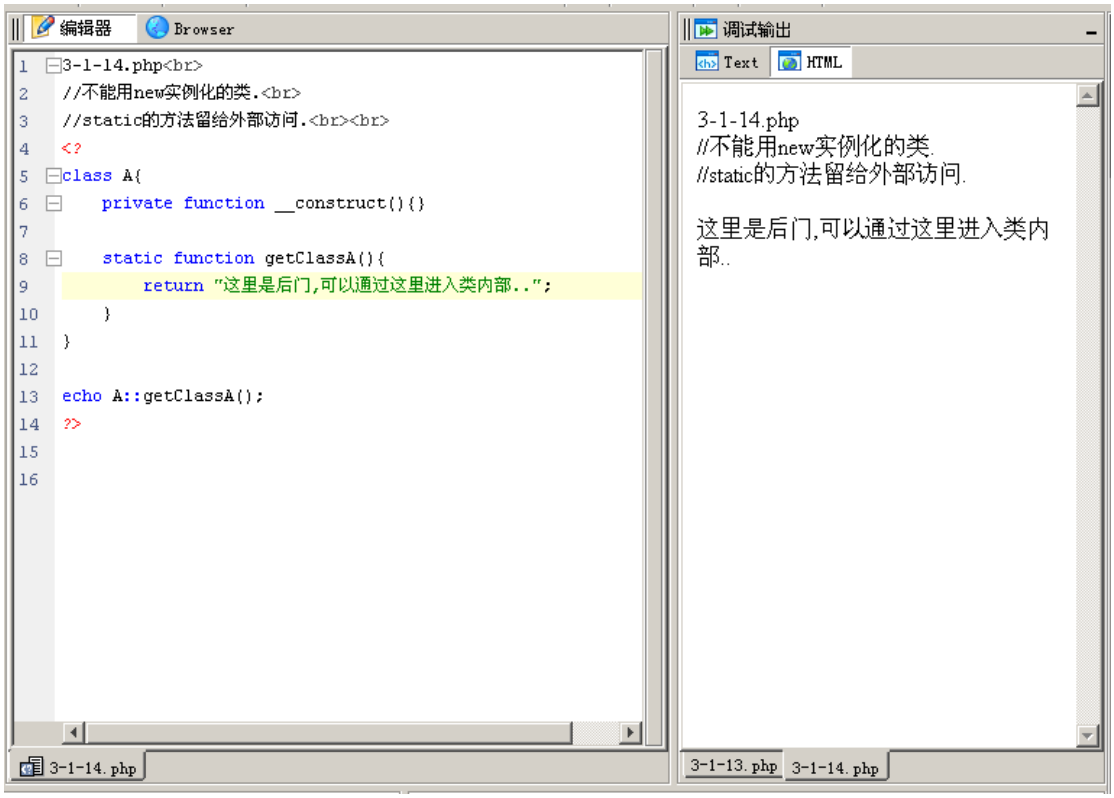
问题 2：

- 我们已经禁止外部用 new 实例化这个类，我们改如何让用户访问这个类呢?
- 前门堵了，我们需要给用户留个后门。

解决办法：

- static 修饰的方法，可以不经实例化一个类就可以直接访问这个方法。
- 后门就在这里。

例：3-1-14.php



问题 3：

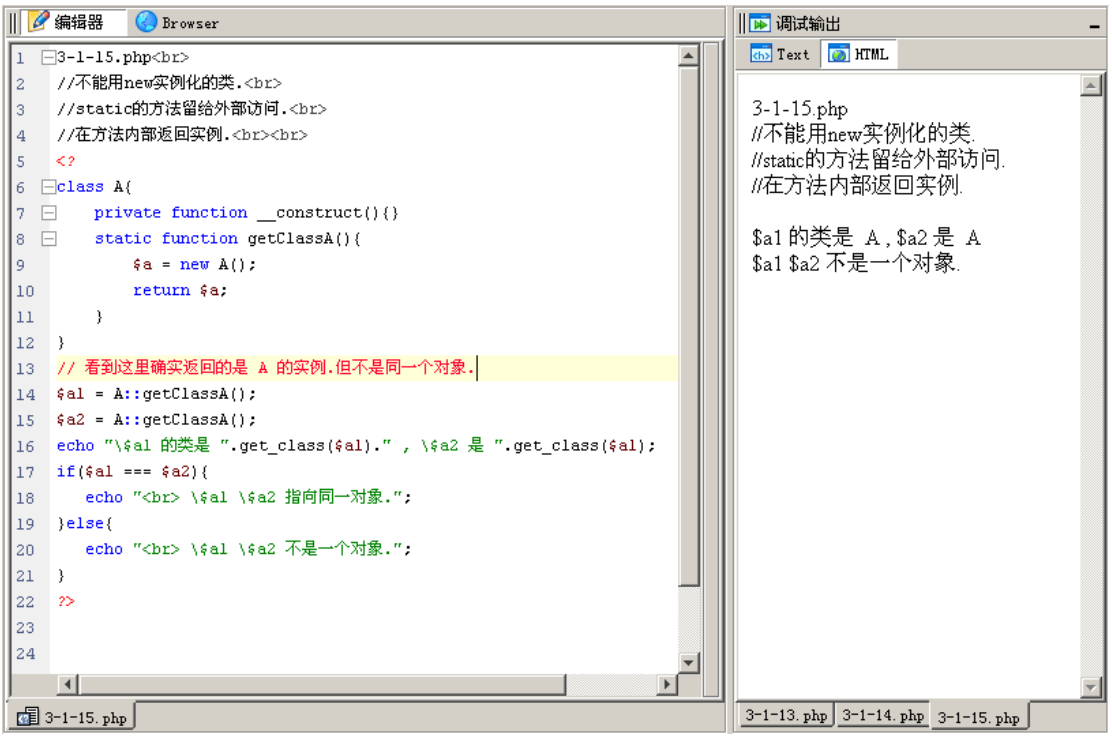
- 虽然我们已经进入类内部，但我们要的是这个类的唯一实例？
- 先不管别的，我们先需要一个实例。
- 通过这个 static 的方法返回这个实例，如何做呢？

解决办法：

- private 的构造函数，不能被外部实例化。
- 但是我们已经成功潜入类的内部了（间谍？007？），我们在内部当然可以调用 private 的方法创建对象。
- 我们这样做看看。

例 3-1-15.php

下面的例子我们确实返回了 A 类的实例，但注意两次执行返回的不是同一个实例。



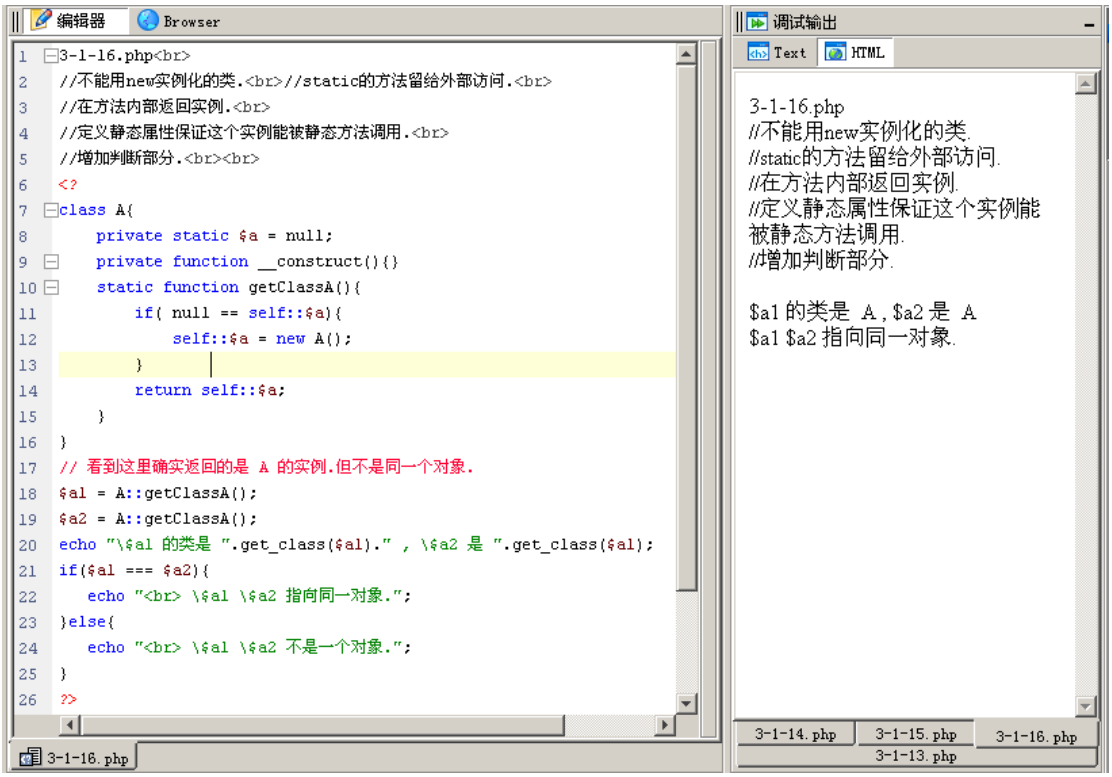
问题 4：

- 我们已经通过 static 方法返回了 A 的实例。但还有问题。
- 我们如何保证我们多次操作获得的是同一个实例的呢？

解决办法：

- static 的属性在内部也只有一个。
- static 属性能有效的被静态方法调用。
- 将这个属性也设置成 private，以防止外部调用。
- 先将这个属性设置成 null。
- 每次返回对象前，先判断这个属性是否为 null 。
- 如果为 null 就创建这个类的新实例，并赋值给这个 static 属性。
- 如果不为空，就返回这个指向实例的 static 属性。

例：3-1-16.php



- 到此，我们写了一个最简单的 单件模式 。
- 现在，你可以尝试写一个应用单件设计模式的数据库连接类。
- 要记住单件模式的使用效果和书写方式。

3.2final 类、final 方法和常量

- final---用于类、方法前。
- final 类---不可被继承。
- final 方法---不可被覆盖。

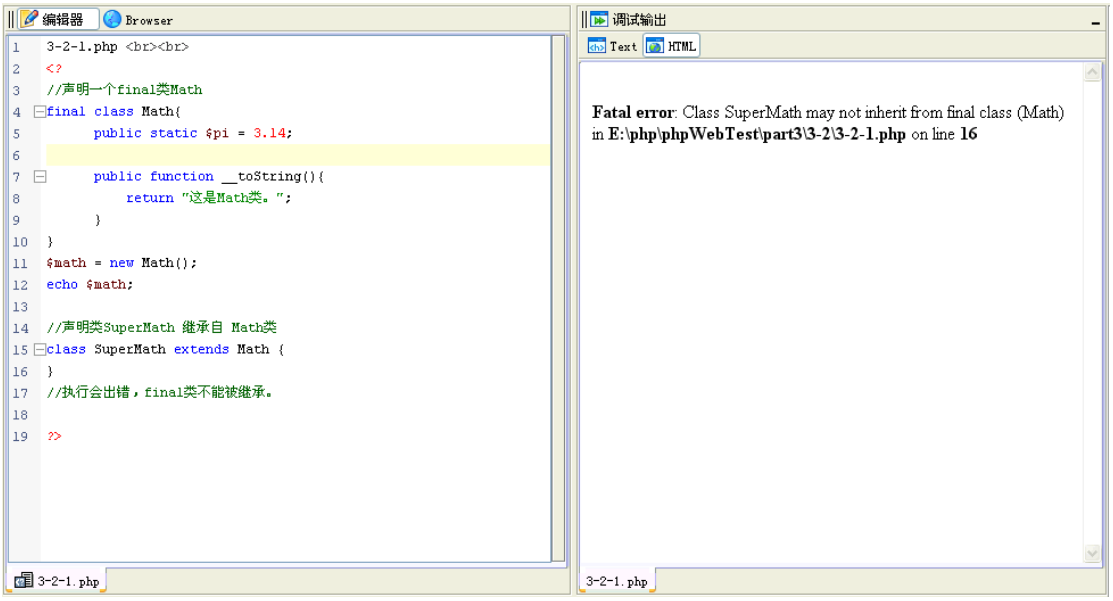
3.2.1 final 类的不能被继承

如果我们不希望一个类被继承，我们使用 final 来修饰这个类。

于是这个将无法被继承。

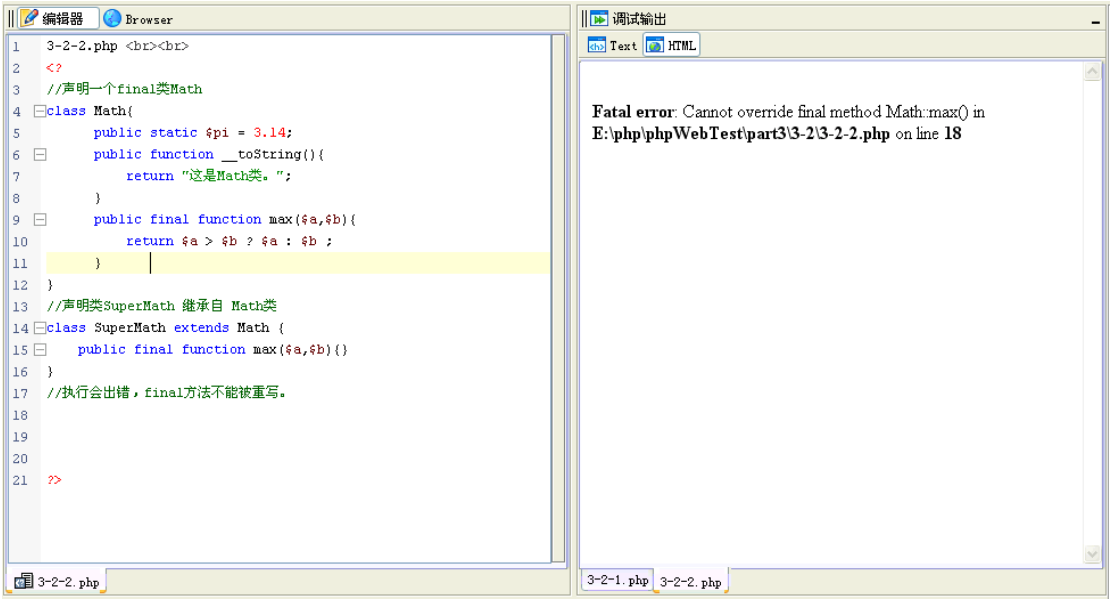
比如我们设定的 Math 类，涉及了我们要做的数学计算方法，这些算法也没有必要修改，，也没有必要被继承，我们把它设置成 final 类型。

例 3-2-1.php



3.2.2 final 方法不能被重写

如果不希望类中的某个方法被子类重写，我们可以设置这个方法为 final 方法，只需要在这个方法前加上 final 修饰符。
如果这个方法被子类重写，将会出现错误。
例：3-2-1.php

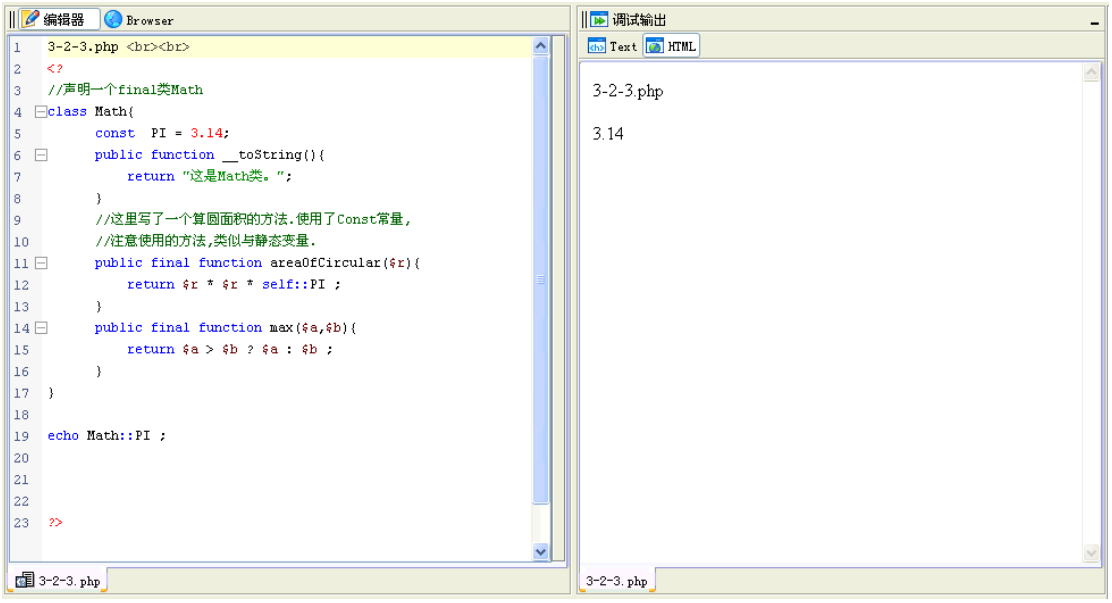


3.2.3 PHP5 中的常量

- 在 PHP5 类中继续使用 cons 修饰常量。
- 我们使用 const 定义一个常量，定义的这个常量不能被改变。
- 在 PHP5 中 const 定义的常量与定义变量的方法不同，不需要加 $ 修饰符。
  const PI = 3.14； 这样就可以。

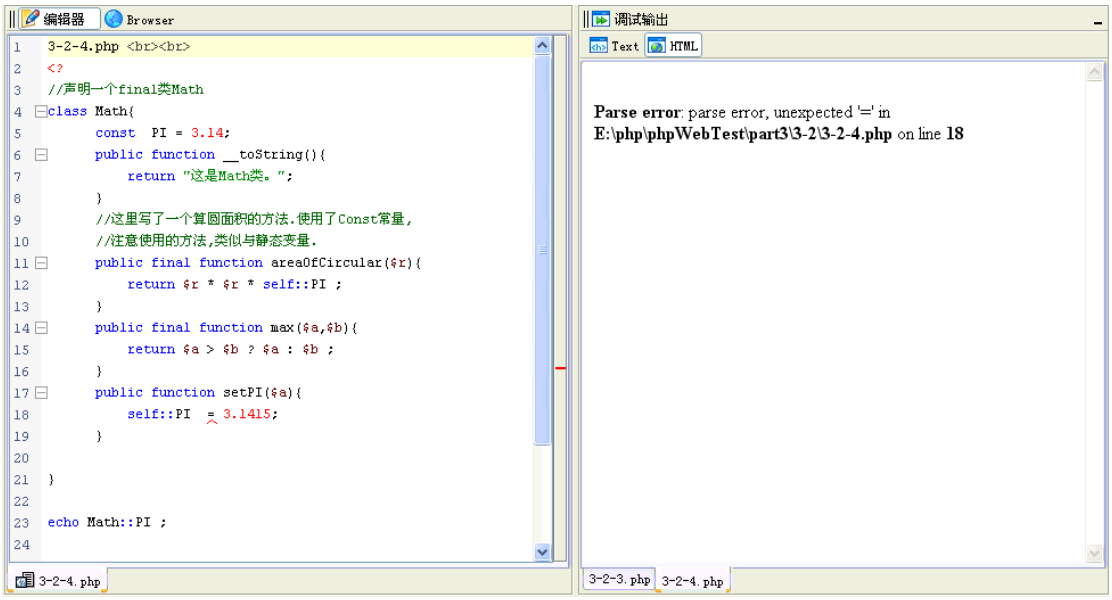
- 而使用 const 定义的常量名称一般都大写，这是一个约定，在任何语言中都是这样。
- 如果定义的常量由多个单词组成，使用 _ 连接，这也是约定。
- 比如， MAX_MUMBER 这样的命名方式。一个良好的命名方式，是程序员必须注意的。
- 类中的常量使用起来类似静态变量，不同点只是它的值不能被改变。我们使用 类名::常量名 来调用这个常量。

例 3-2-3.php



尝试为 const 定义的常量赋值，将会出现错误。

例 3-2-4.php



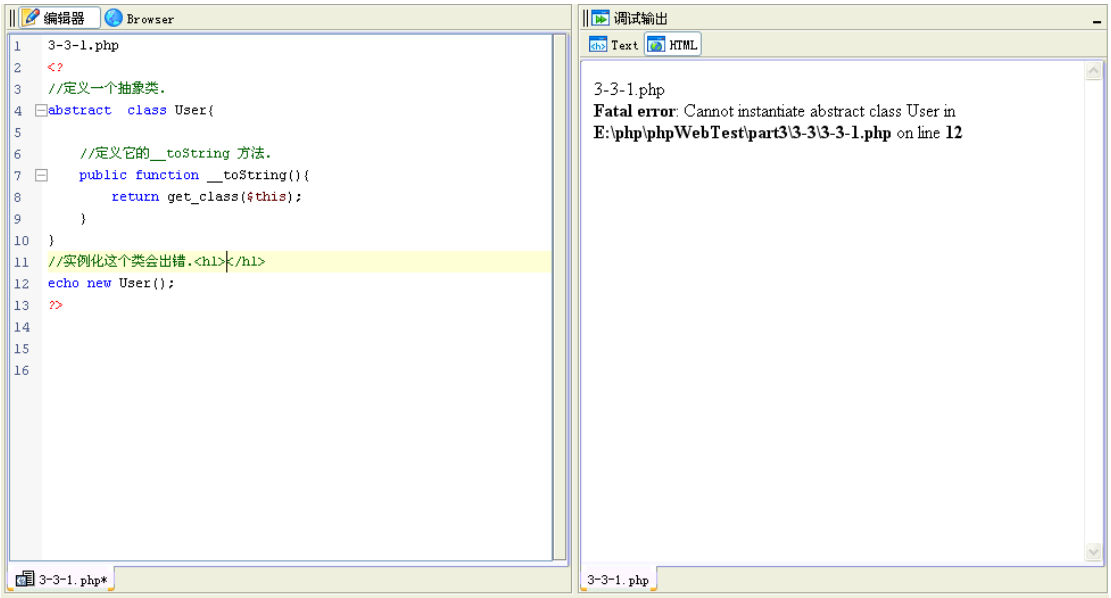
3.3 abstract 类和 abstract 方法

- 可以使用 abstract 来修饰一个类或者方法。
- 用 abstract 修饰的类表示这个类是一个抽象类，用 abstract 修饰的方法表示这个方法是一个抽象方法。
- 抽象类不能被实例化。
- 抽象方法是只有方法声明，而没有方法的实现内容。

3.3.1 abstract 抽象类

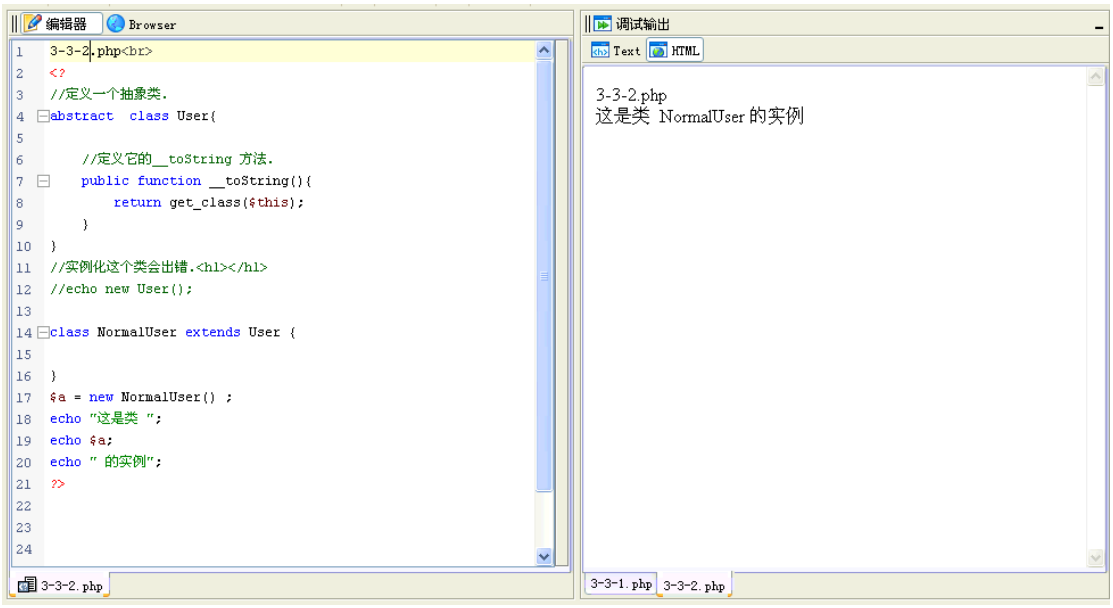
- 可以使用 abstract 来修饰一个类。
- 用 abstract 修饰的类表示这个类是一个抽象类。
- 抽象类不能被实例化。

这是一个简单抽象的方法，如果它被直接实例化，系统会报错。
例：3-3-1.php



例：3-3-2.php

下面例子的 NormalUser 继承自 User 类，就可以被实例化了。



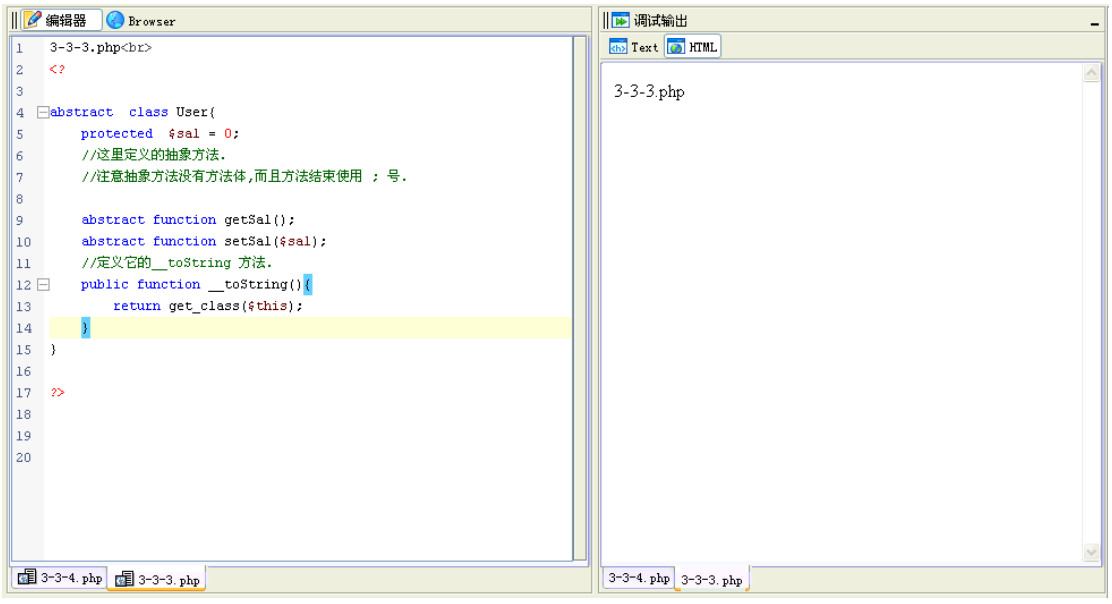
单独设置一个抽象类是没有意义的，只有有了抽象方法，抽象类才有了血肉。
下面介绍抽象方法。

3.3.2 abstract 抽象方法

- 用 abstract 修饰的类表示这个方法是一个抽象方法。
- 抽象方法，只有方法的声明部分，没有方法体。
- 抽象方法没有 {} ，而采用 ； 结束。
- 一个类中，只要有一个抽象方法，这个类必须被声明为抽象类。
- 抽象方法在子类中必须被重写。

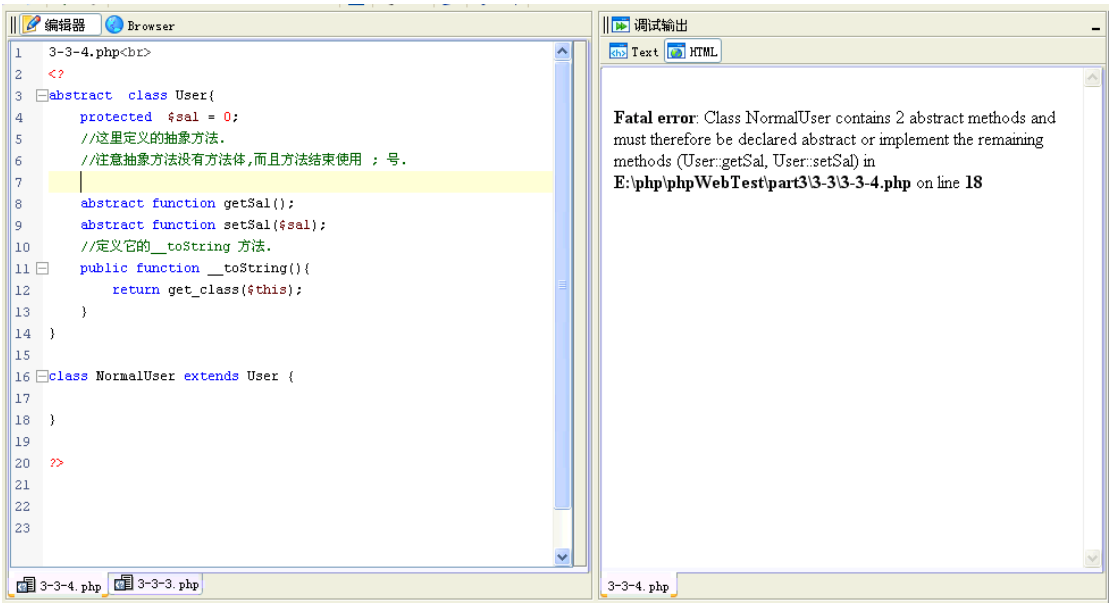
下面是一个抽象类，其中有两个抽象方法，分别是 setSal() 和 getSal()。用来取回$sal 员工的工资。

例：3-3-3.php



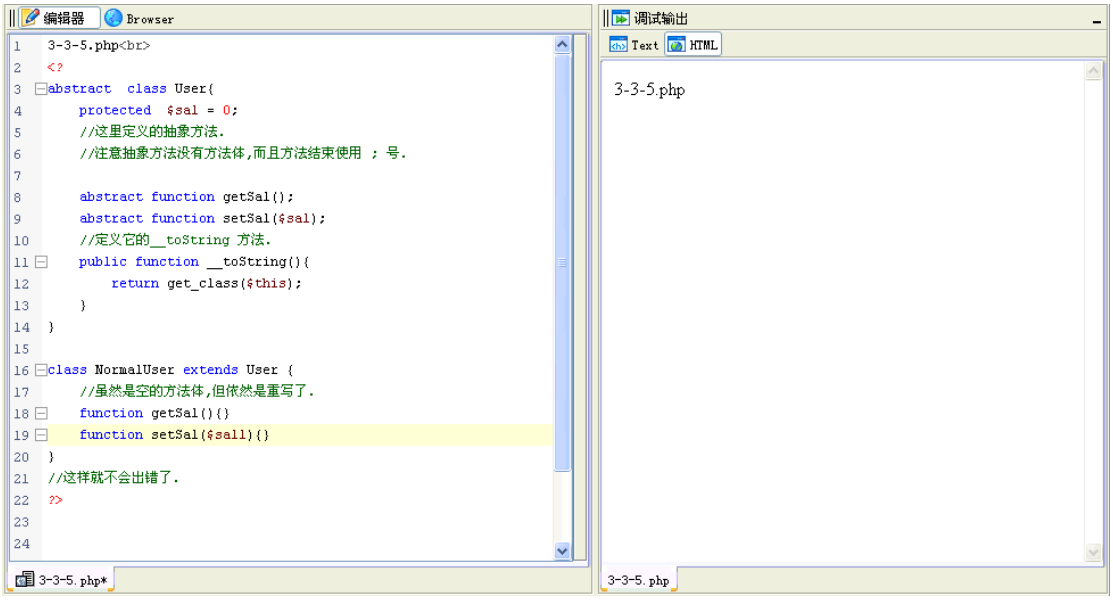
既然 User 类不能被直接继承，我们写一个 NormalUser 类继承自 User 类。
当我们写成如下代码时，系统会报错。
这个错误告诉我们，在 User 类中有两个抽象方法，我们必须在子类中重写这两个方法。

例：3-3-4.php



例：3-3-5.php

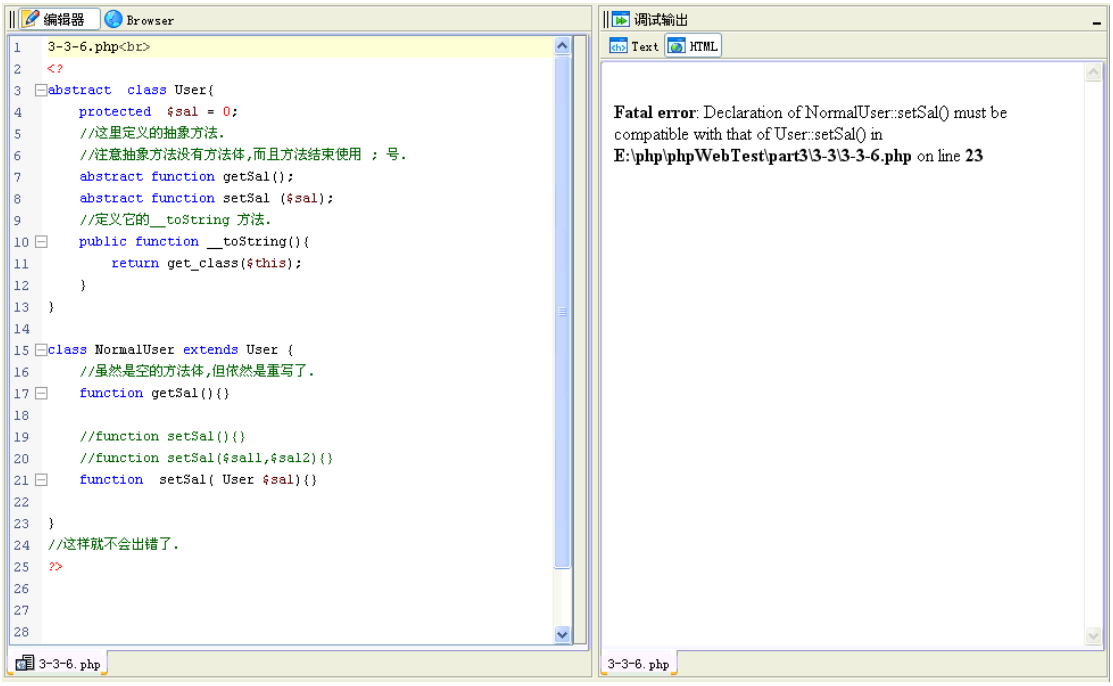
下面例子，重写了这两个方法，虽然方法体里面 {} 的内容是空的，也算重写了这个方法。
注意看重写方法的参数名称，这里只要参数数量一致就可以，不要求参数的名称必须一致。



下面 19-21 行，三种写重写的方式都会报错。

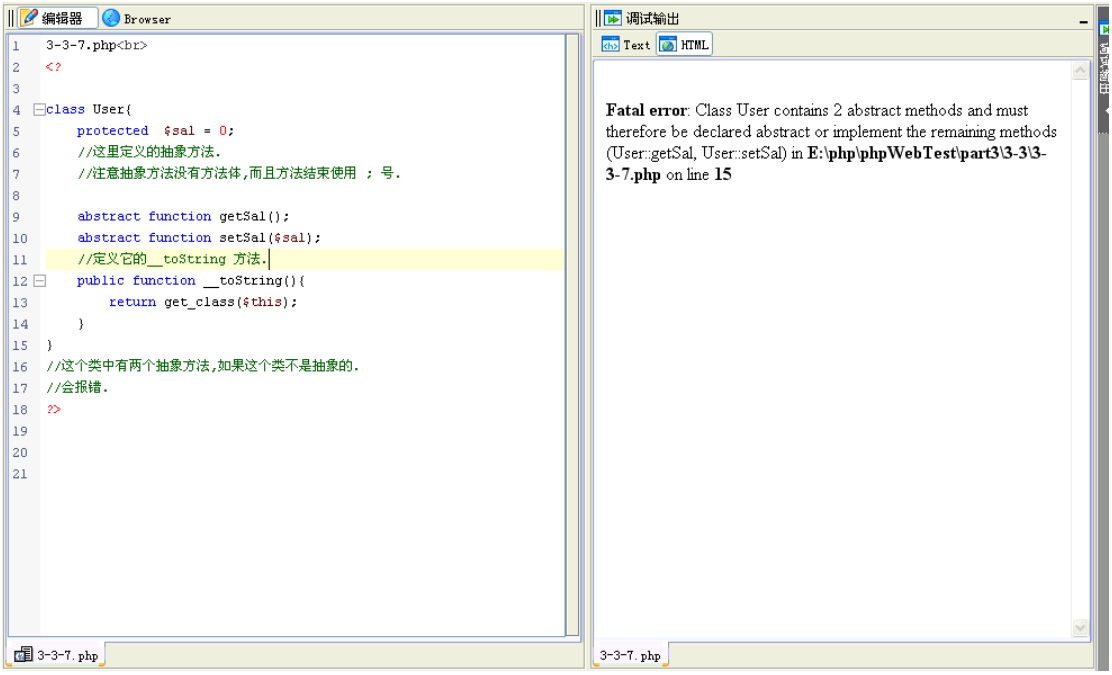
- 19 行，缺少参数。
- 20 行，参数又多了。
- 21 行，参数类型不对。（这种写法在以后章节介绍）

例:3-3-3.php



- 一个类中，如果有一个抽象方法，这个类必须被声明为抽象类。

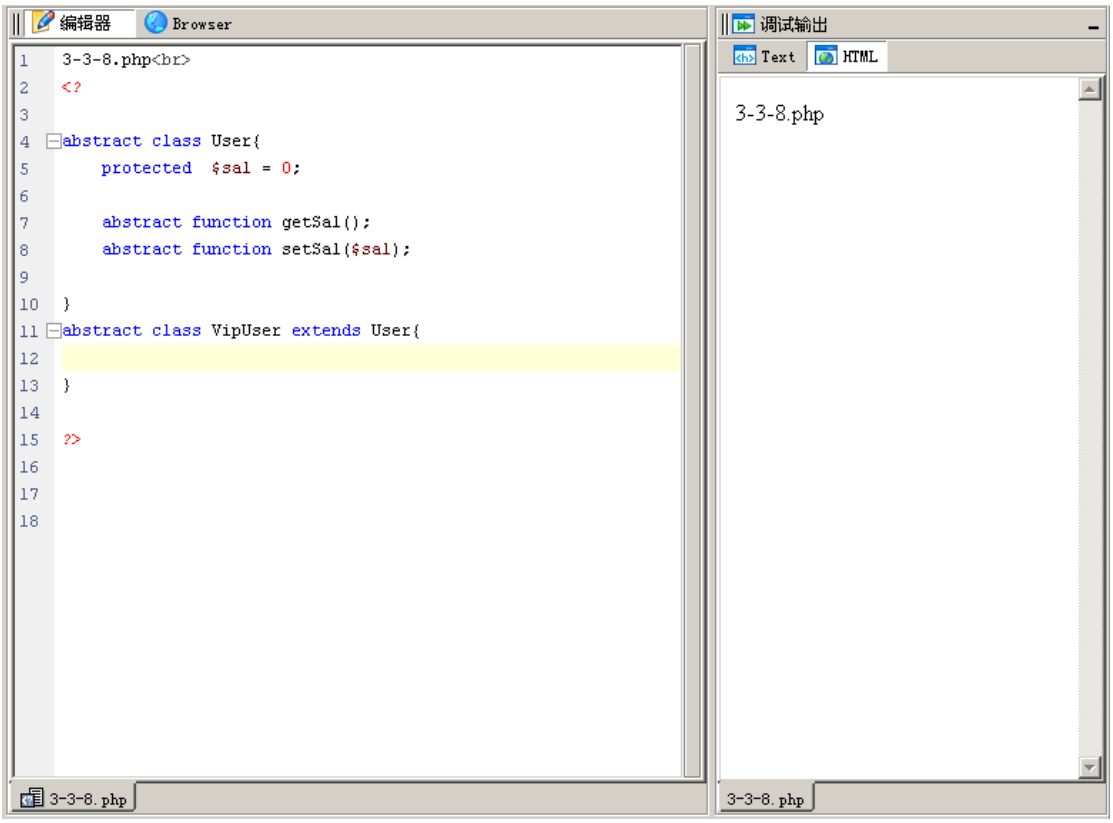
下面这个类不是抽象类，其中定义了一个抽象方法，会报错。
例：3-3-7.php



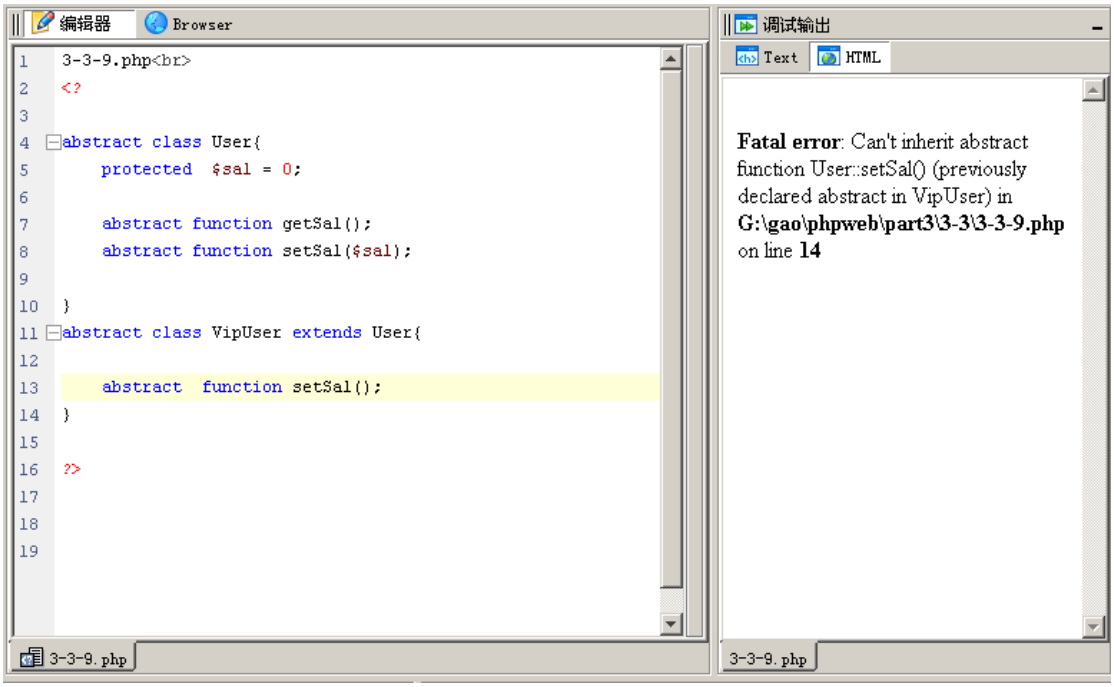
3.3.3 抽象类继承抽象类

- 抽象类继承另外一个抽象类时，不用重写其中的抽象方法。
- 抽象类中，不能重写抽象父类的抽象方法。
- 这样的用法，可以理解为对抽象类的扩展。

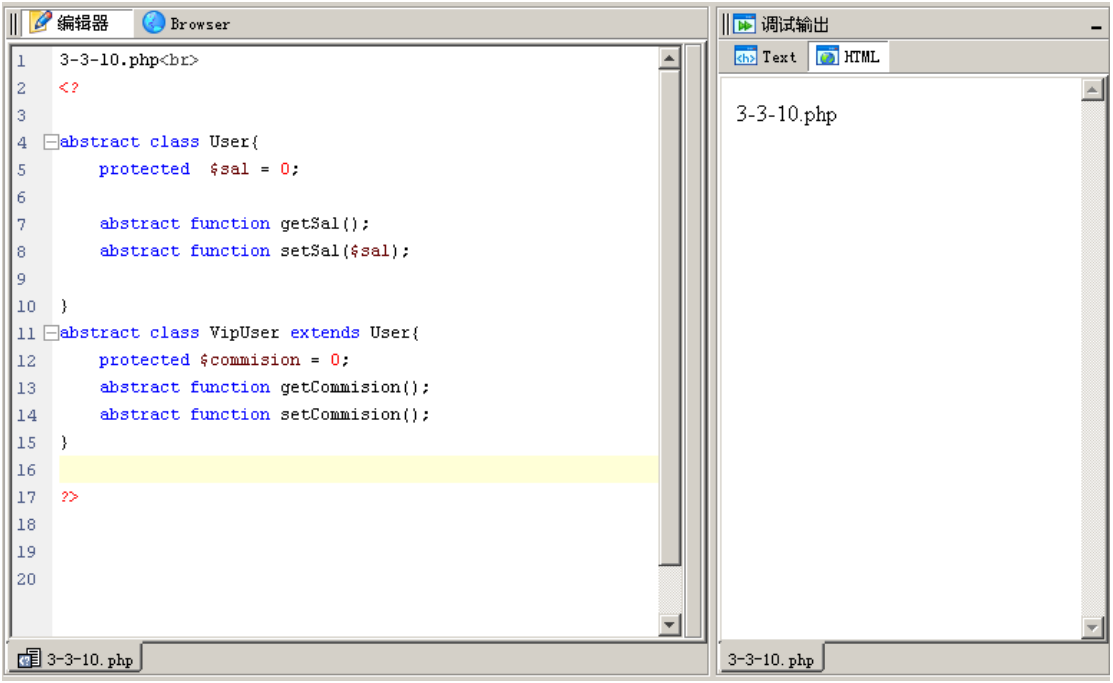
下面的例子，演示了一个抽象类继承自另外一个抽象类时，不需要重写其中的抽象方法。
例：3-3-8.php



抽象类在被继承后，其中的抽象方法不能被重写。
如果发生重写，系统会报错。
例 3-3-9.php



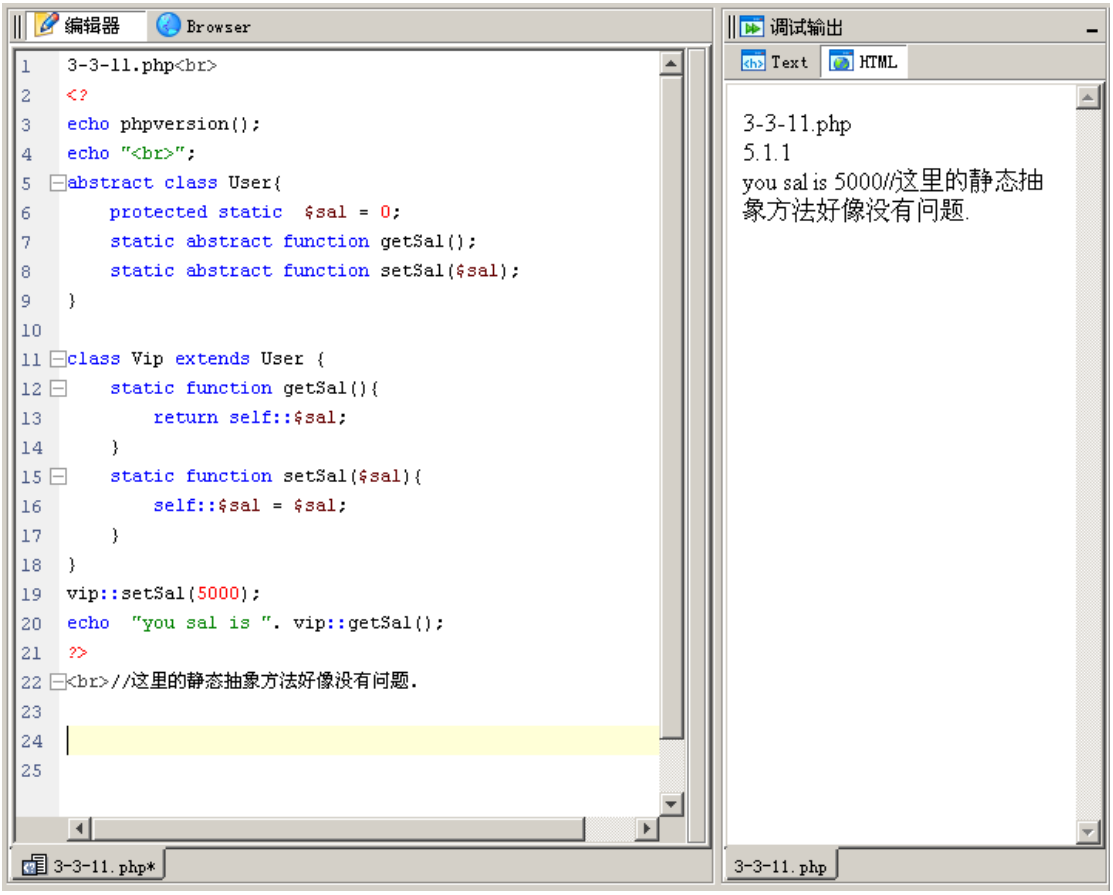
抽象类继承抽象类，目的对抽象类的扩展。
例 3-3-10.php



3.3.4 静态抽象方法

在 PHP5.1 中，抽象类中支持静态抽象方法。
下面这个例子，看到静态抽象方法可以声明。实现这个方法时，必须是静态的方法。

例：3-3-11.php



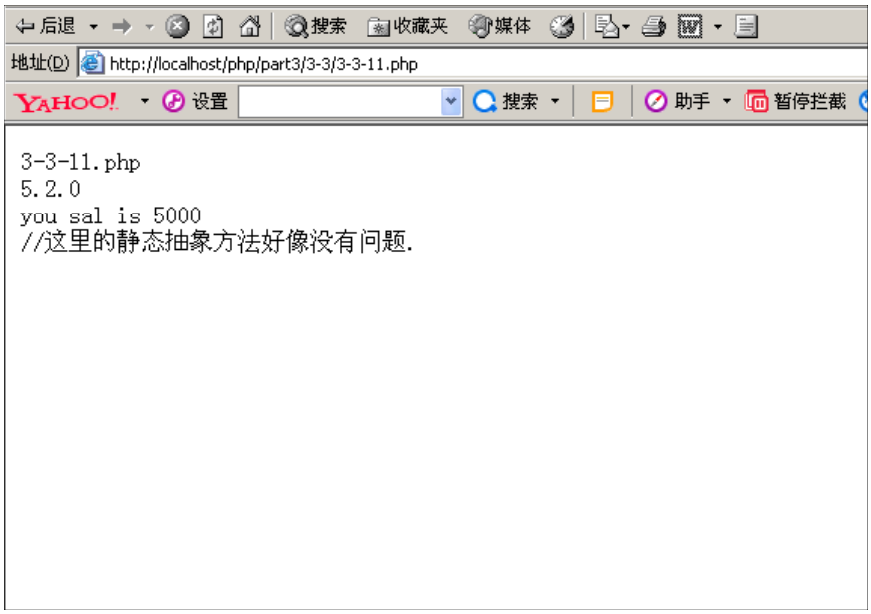
3.3.5 PHP5.2.0 中的静态抽象方法

在 PHP5.2.0 的 更新信息中有这样一段话：

```
- Dropped abstract static class functions. (Marcus)

  Due to an oversight, PHP 5.0 and 5.1 allowed abstract static functions in
  classes. In PHP 5.2, only interfaces can have them.
```

译文：(因为疏漏，在 PHP5.0 和 PHP5.1 的类中允许静态抽象方法。
在 PHP5.2 中，只有接口可以拥有静态抽象方法。)

在 PHP5.2.0 版本中测试，刚才的代码执行正常。在 PHP5.2.0 中是依然兼容。



3.4 设计模式之模版模式

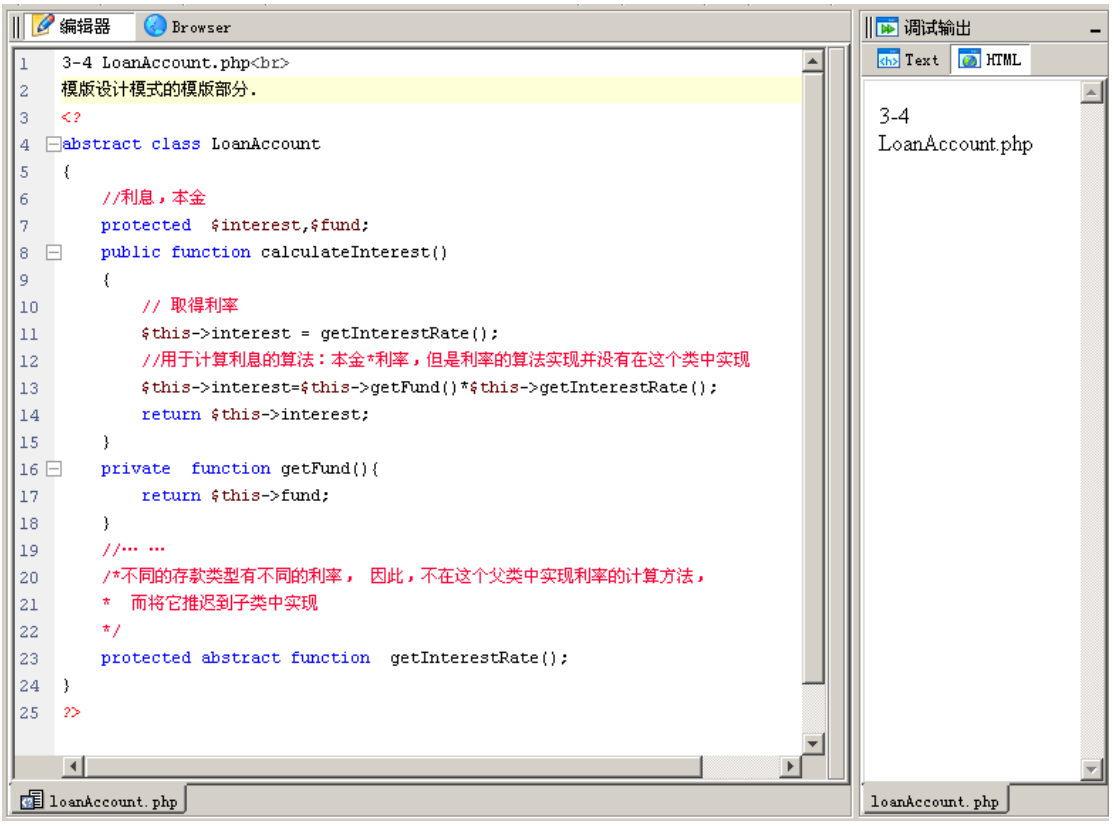
抽象类的应用就是典型的模版模式，先声明一个不能被实例化的模版，在子类中去依照模版实现具体的应用。

3.4.1 模版模式实例

我们写这样一个应用：

> 银行计算利息，都是利率乘以本金和存款时间，但各种存款方式计算利率的方式不同，所以，在账户这个类的相关方法里，只搭出算法的骨架，但不具体实现。具体实现由各个子类来完成。

例：LoanAccount.php




```
3-4 LoanAccount.php<br>
模版设计模式的模版部分.
<?
abstract class LoanAccount
{
    //利息，本金
    protected  $interest,$fund;
    public function calculateInterest()
    {
        // 取得利率
        $this->interest = getInterestRate();
        //用于计算利息的算法：本金*利率，但是利率的算法实现并没有在这个类中实现
        $this->interest=$this->getFund()*$this->getInterestRate();
        return $this->interest;
    }
    private  function getFund(){
        return $this->fund;
    }
    //… …
    /*不同的存款类型有不同的利率， 因此，不在这个父类中实现利率的计算方法，
    *  而将它推迟到子类中实现
    */
    protected abstract function  getInterestRate();
}
?>
```

以后，所有和计算利息的类都继承自这个类，而且必须实现其中的 getInterestRate() 方法，这种用法就是模版模式。

> 下一章将介绍面向对象语言的其它重要组成 接口、多态。

第四章 PHP5 接口与多态

目录

4.1 接口的定义与规范

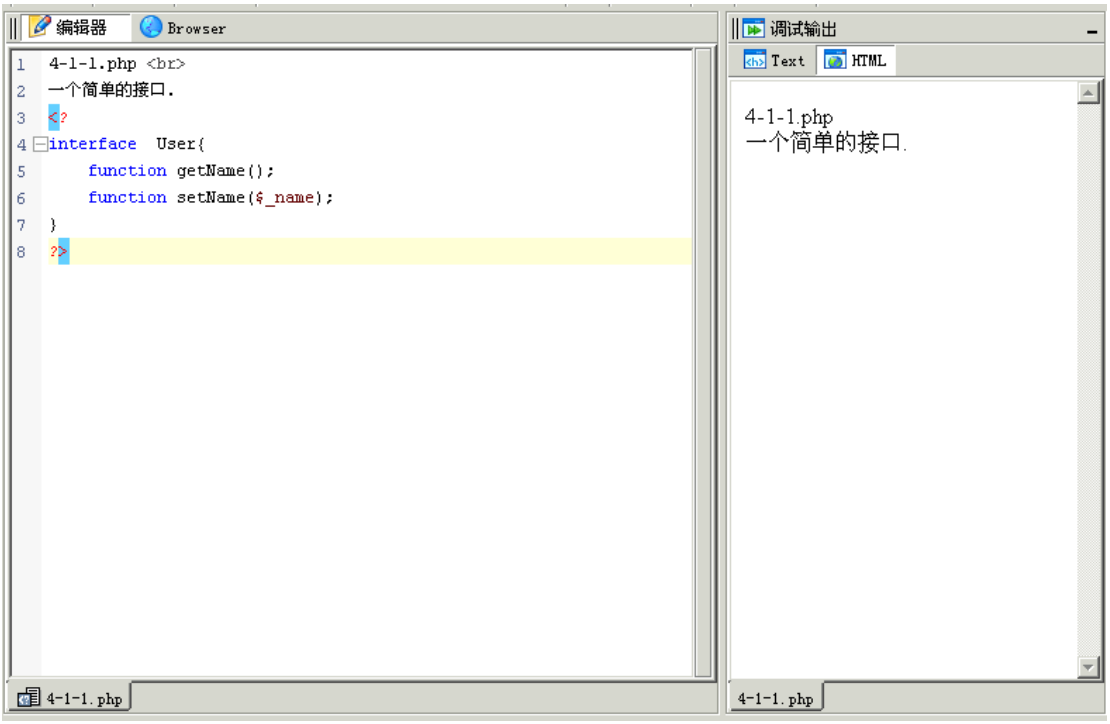
- 接口(interface)是**抽象方法**和**静态常量**定义的集合。
- 接口是一种特殊的抽象类，这种抽象类中只包含抽象方法和静态常量。
- 接口中没有其它类型的内容。

4.1.1 接口的定义

我们先写接口的定义，后面几节再介绍接口的意义。
下面的例子是接口的一个简单写法。

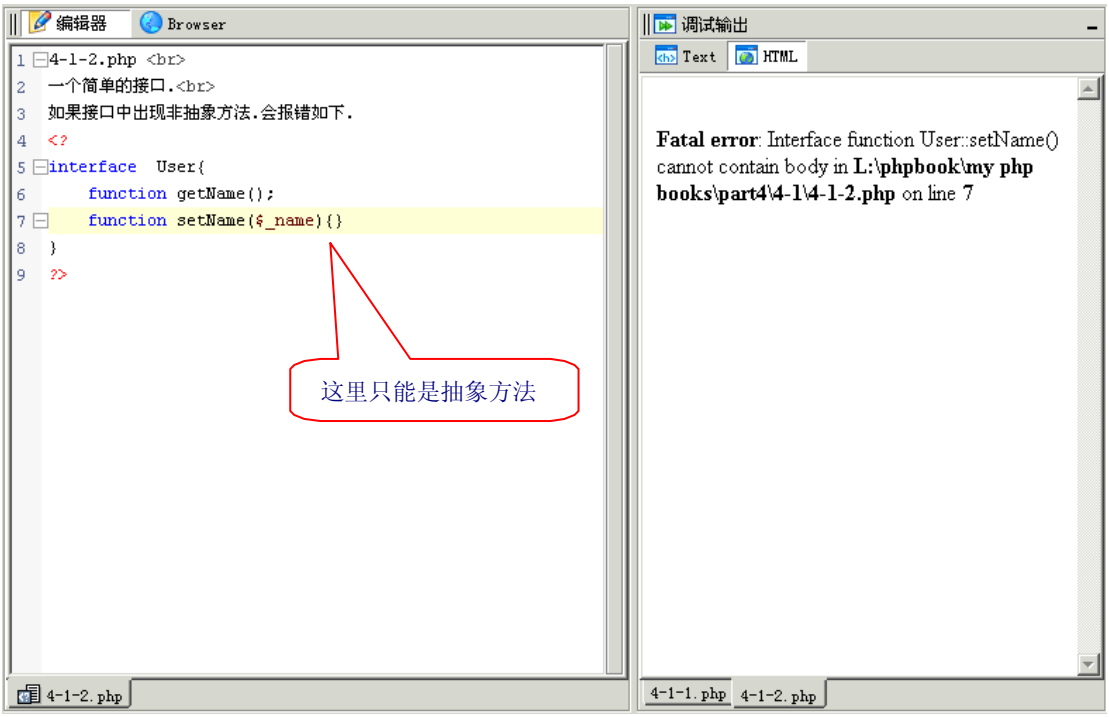
```
interface 接口名{
}
```

下面的例子定义了一个接口 User ，这个接口中有两个抽象方法，getName() 和 setName()。能看到接口的写法和类很相似。
例：4-1-1.php


编辑器
Browser
4-1-1.php
一个简单的接口.
<?
interface User{
function getName();
function setName($_name);
}
?>
4-1-1.php
调试输出
Text
HTML
4-1-1.php
一个简单的接口.
4-1-1.php

4.1.2 接口中的抽象方法

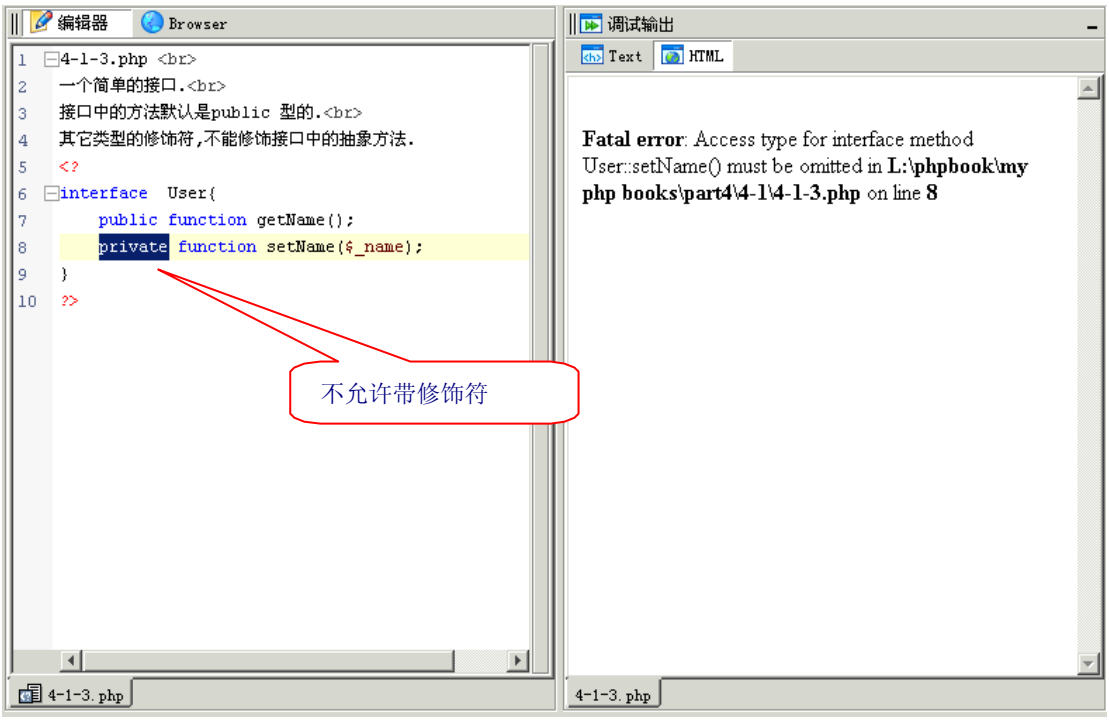
注意，在接口中只能有抽象方法。如果在接口中出现了非抽象方法，会报错如下：
Interface function User::setName() cannot contain body in …….
例 4-1-2.php



4.2.3 接口中抽象方法的修饰和访问权限

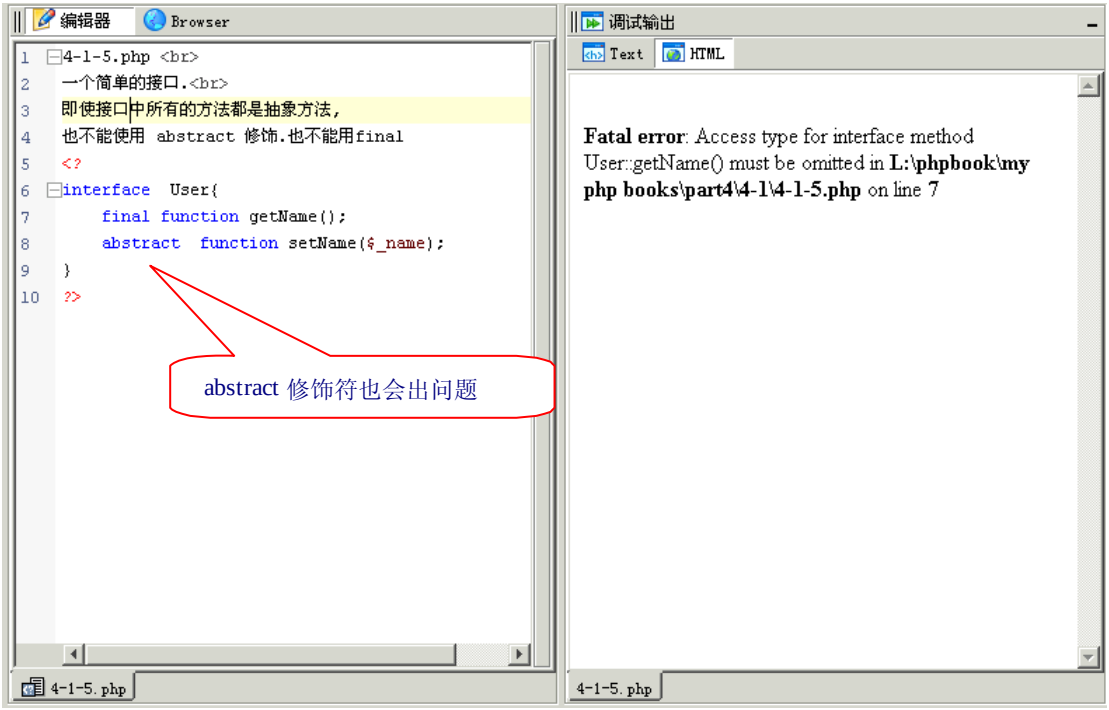
在接口中的抽象方法只能是 public 的，默认也是 public 权限。
并且不能设置成 private 或者 protected 类型。
否则会报错如下：
Access type for interface method User::setName() must be omitted in —on line —
（在接口中，访问类型必须忽略。）

例：4-1-3.php


编辑器
Browser
1 4-1-3.php
2 一个简单的接口.
3 接口中的方法默认是public 型的.
4 其它类型的修饰符,不能修饰接口中的抽象方法.
5 <?
6 interface User{
7 public function getName();
8 private function setName($_name);
9 }
10 ?>
不允许带修饰符
4-1-3.php
调试输出
Text
HTML
Fatal error: Access type for interface method User::setName() must be omitted in L:\phpbook\my php books\part4\4-1\4-1-3.php on line 8
4-1-3.php

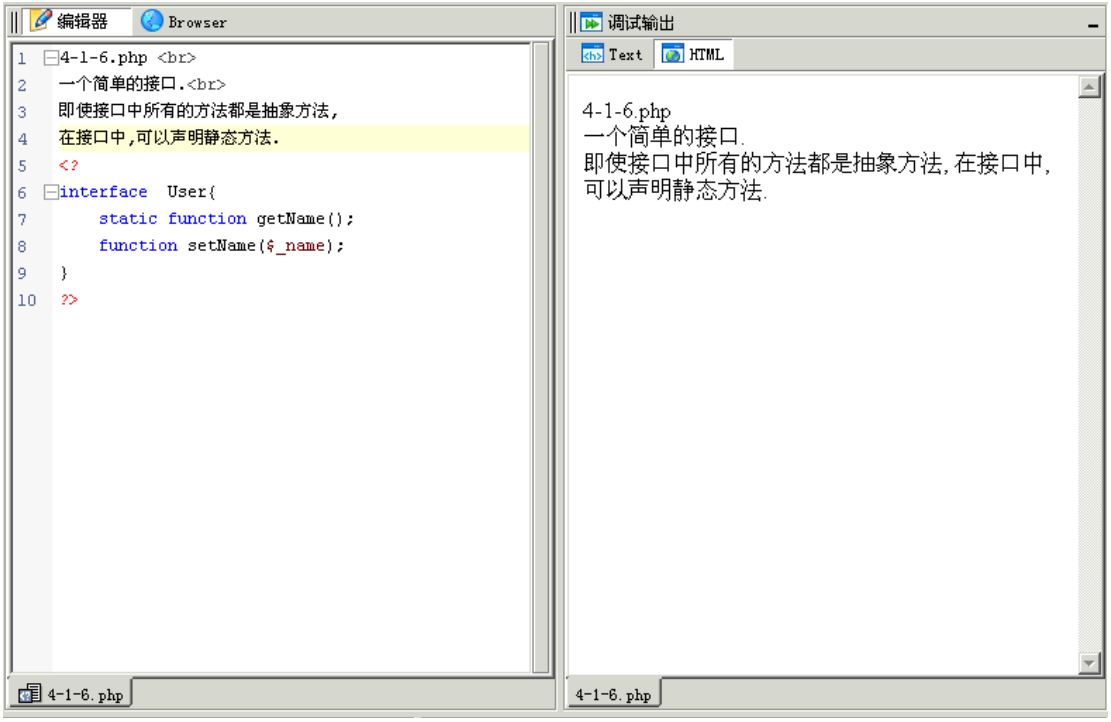
例：4-1-4.php
protected 访问权限也会有问题

例：4-1-5.php
即使 abstract 和 final 修饰符不能修饰接口中的抽象方法。



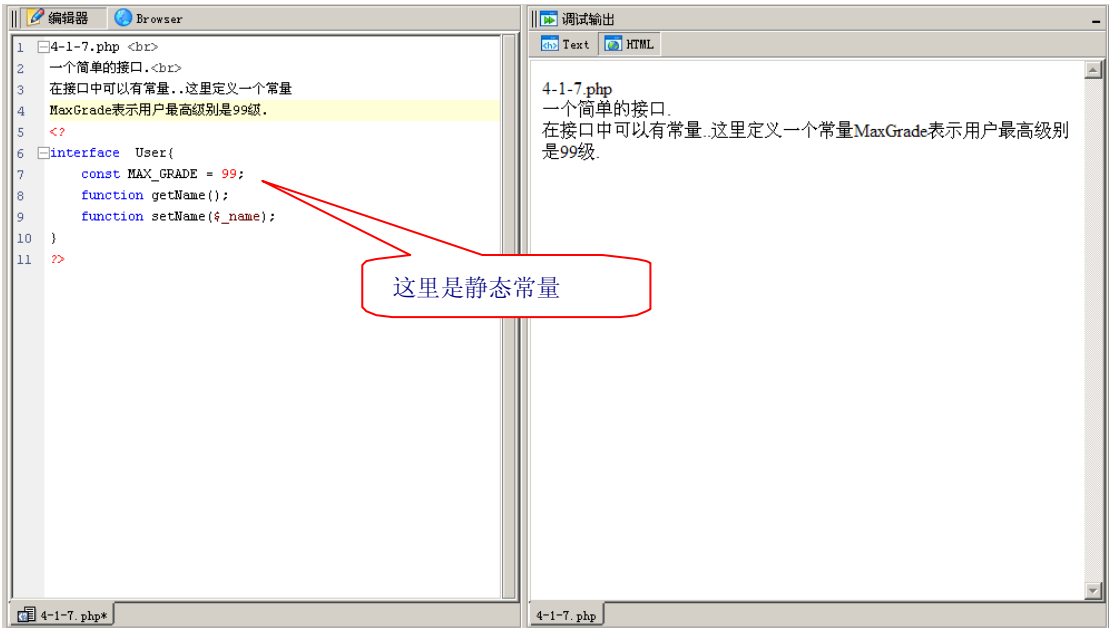
4.1.4 接口中的静态抽象方法

例：4-1-6.php
在接口中可以使用静态抽象方法。在 PHP5.2 中，不建议在抽象类中使用静态抽象方法。而接口中依然保留了静态抽象方法。



4.1.5 接口中的静态常量

在接口中可以定义静态常量。而且不用 static 修饰就是静态的常量。
例 4-1-7.php


编辑器
Browser
1 4-1-7.php
2 一个简单的接口.
3 在接口中可以有常量..这里定义一个常量
4 MaxGrade表示用户最高级别是99级.
5 <?
6 interface User{
7 const MAX_GRADE = 99;
8 function getName();
9 function setName($_name);
10 }
11 ?>
这里是静态常量
4-1-7.php*
调试输出
Text
HTML
4-1-7.php
一个简单的接口.
在接口中可以有常量..这里定义一个常量MaxGrade表示用户最高级别是99级.
4-1-7.php

4.2 实现接口

- 类实现接口要使用 implements 。
- 类实现接口要实现其中的抽象方法。
- 一个类可以实现多个接口。
-

一个类可以使用 implements 实现接口，甚至可以实现多个接口。

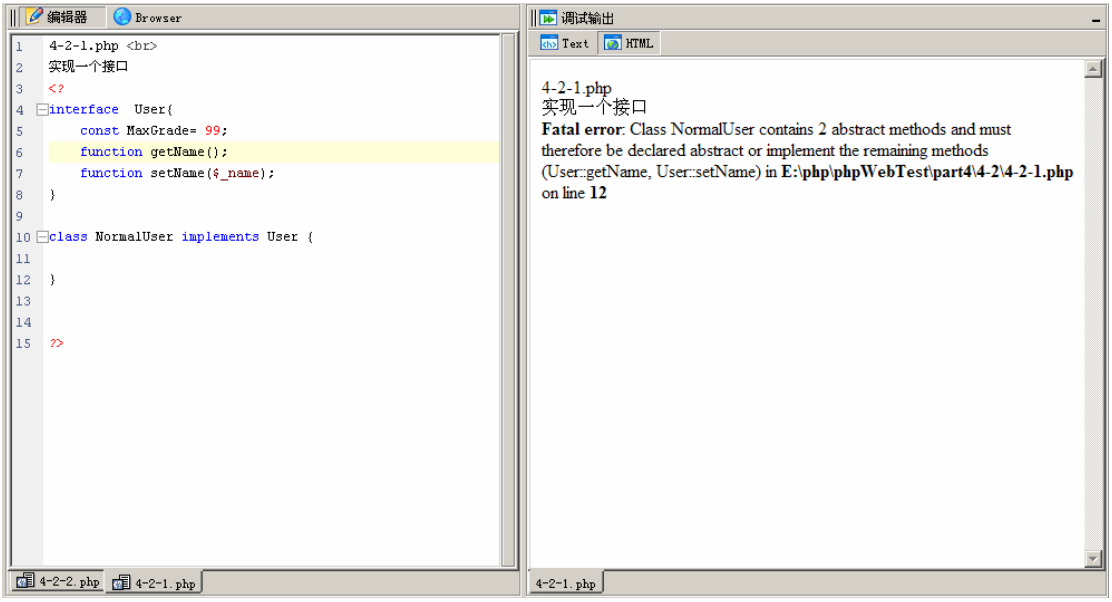
大部分的书说，这样是为了实现 PHP 的多继承。为什么呢？ PHP5 是单继承的，一个类只可以继承自一个父类。接口可以实现多个，这样就是多继承了。

这样说有些道理。但，既然接口里面的方法没有方法体，所谓的多继承又有什么意义？接口的意义在于后面一节继续说的多态。

4.2.1 使用 implements 实现接口

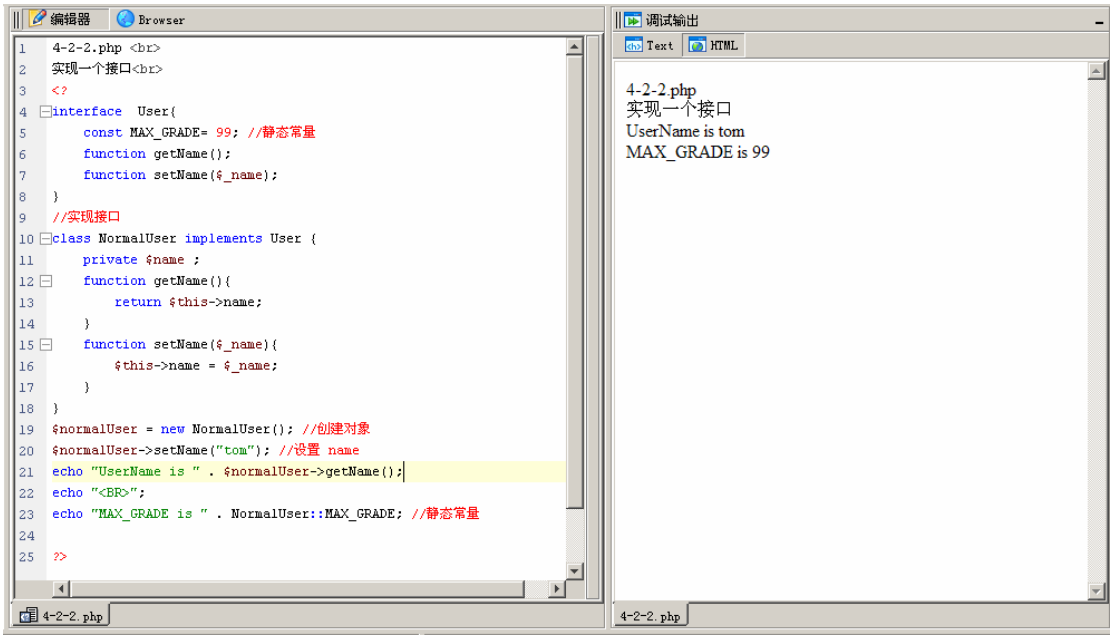
使用 implements 来实现一个接口。如果实现接口而没有实现其中的抽象方法，会报错如下。

例：4-2-1.php



例 4-2-2.php

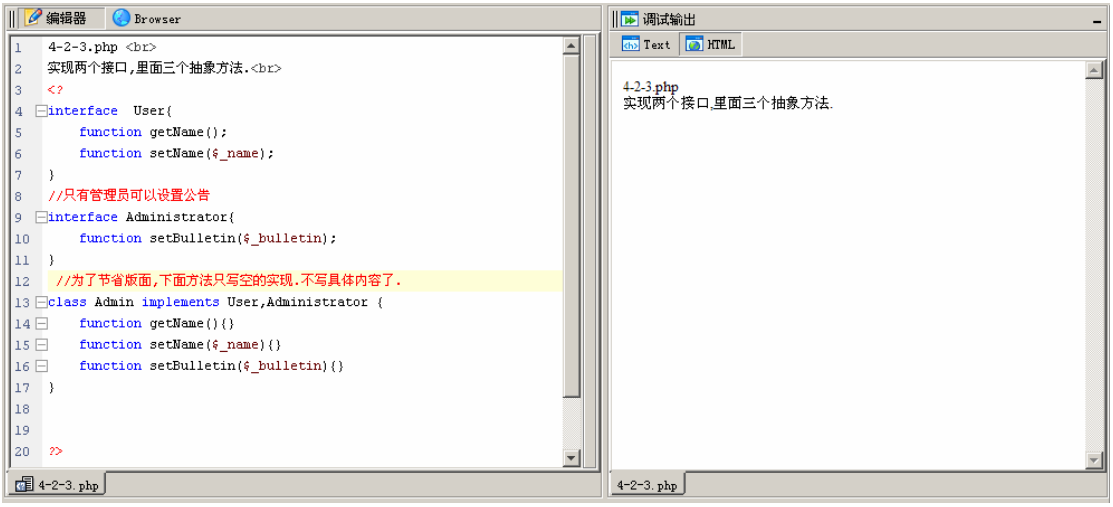
实现接口要实现方法。注意静态变量的使用。



4.2.2 实现多个接口

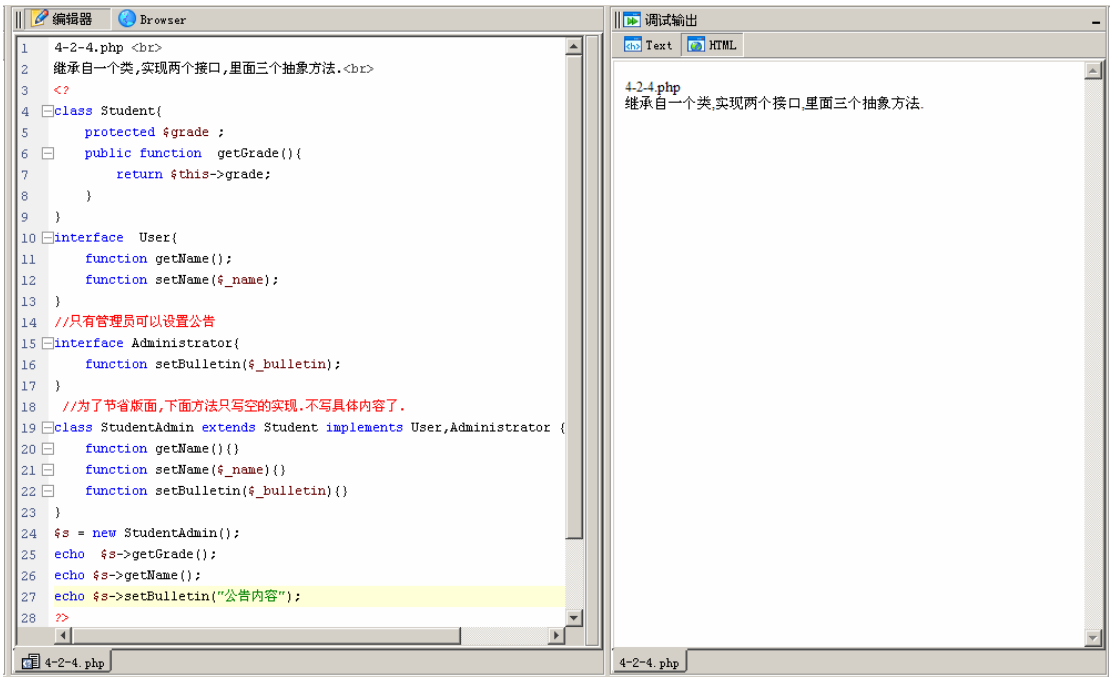
一个类可以实现多个接口。只要使用 ，号将多个接口链接起来就可以。

例： 4-2-3.php



4.2.3 继承并实现接口

这个例子，让 StudentAdmin 继承自 Student 类，并实现了 User 和 Administrator 接口。
例： 4-2-4.php



4.3 接口的继承

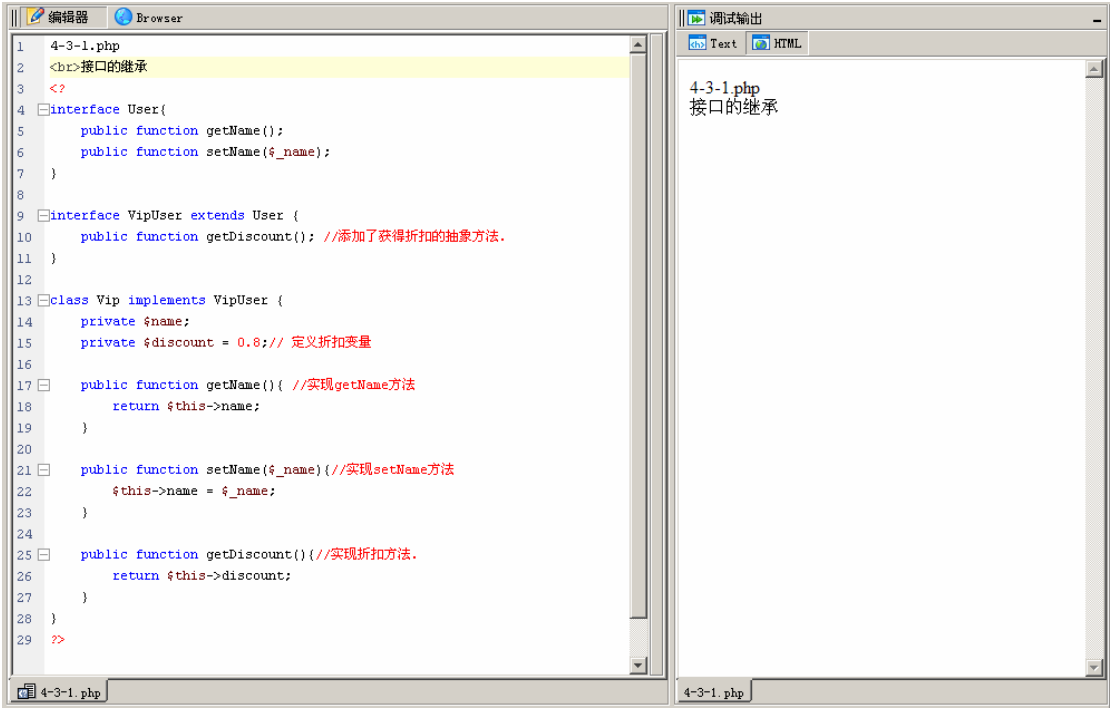
- 一个接口可以继承自另外的接口。
- PHP5 中的类是单继承，但是接口很特殊。一个接口可以继承自多个接口。
- 一个接口继承其它接口时候，直接继承父接口的静态常量属性和抽象方法。

在 PHP5 中，接口是可以继承自另外一个接口的。这样代码的重用更有效了。
要注意只有接口和接口之间使用 继承关键字 extends。
类实现接口必须实现其抽象方法，使用实现关键字 implements。

4.3.1 接口实现继承

要注意只有接口和接口之间使用 继承关键字 extends。
类实现接口必须实现其抽象方法，使用实现关键字 implements。

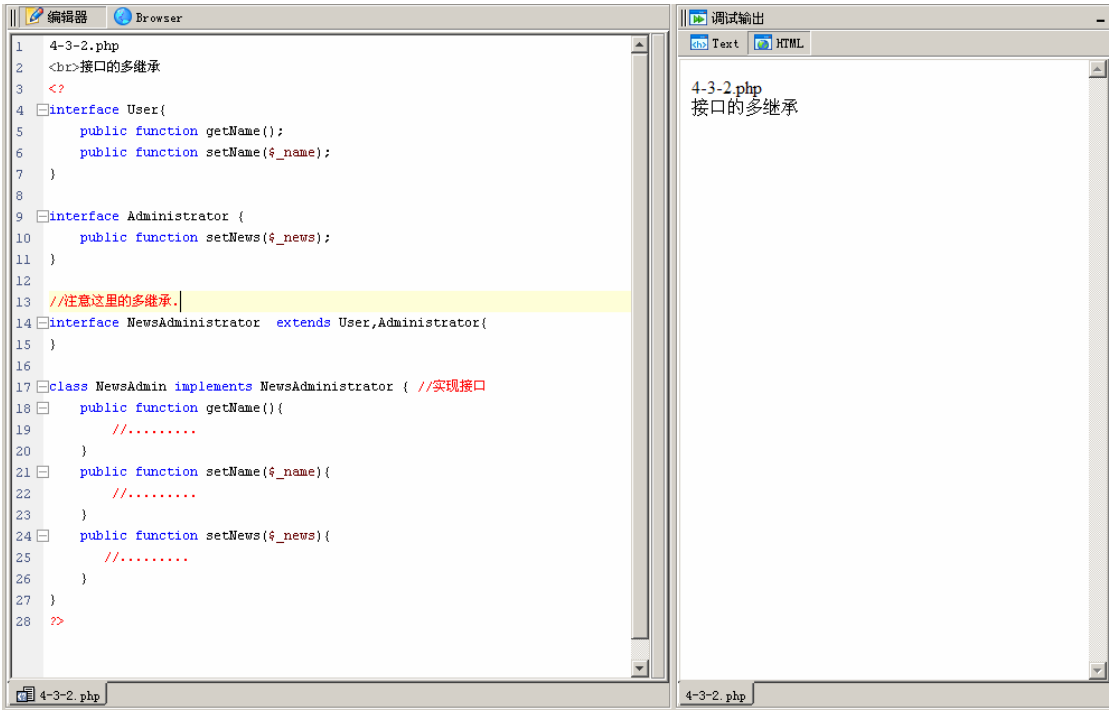
例 4-3-1.php
这个例子定义接口 User，User 有两个抽象方法 getName 和 setName。又定义了接口 VipUser，继承自 User 接口，并增加了和折扣相关的方法 getDiscount。
最后定义了类 Vip ，实现了 VipUser 接口。并实现了其中的三个方法。



4.3.2 接口可以实现多继承

接口可以实现多继承，这是接口很特殊的地方。注意下面的代码和用法。

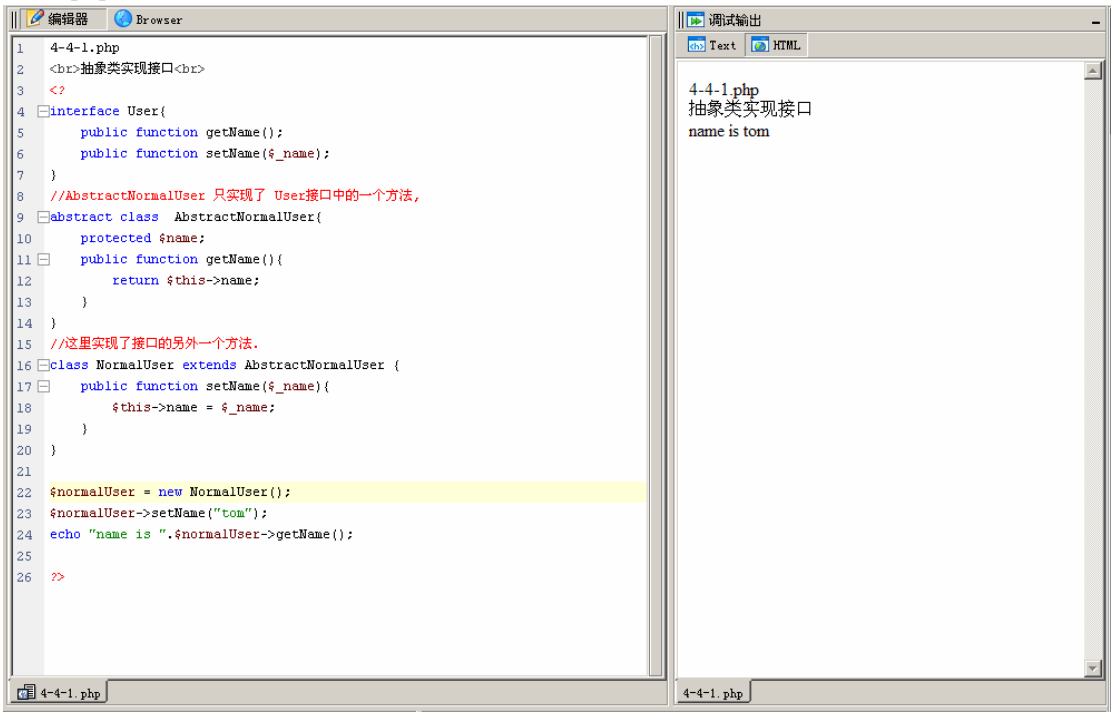
例：4-3-2.php



4.4 抽象类实现接口

- 抽象类实现接口，可以不实现其中的抽象方法，而将抽象方法的实现交付给具体能被实例化的类去处理。

4-4-1.php



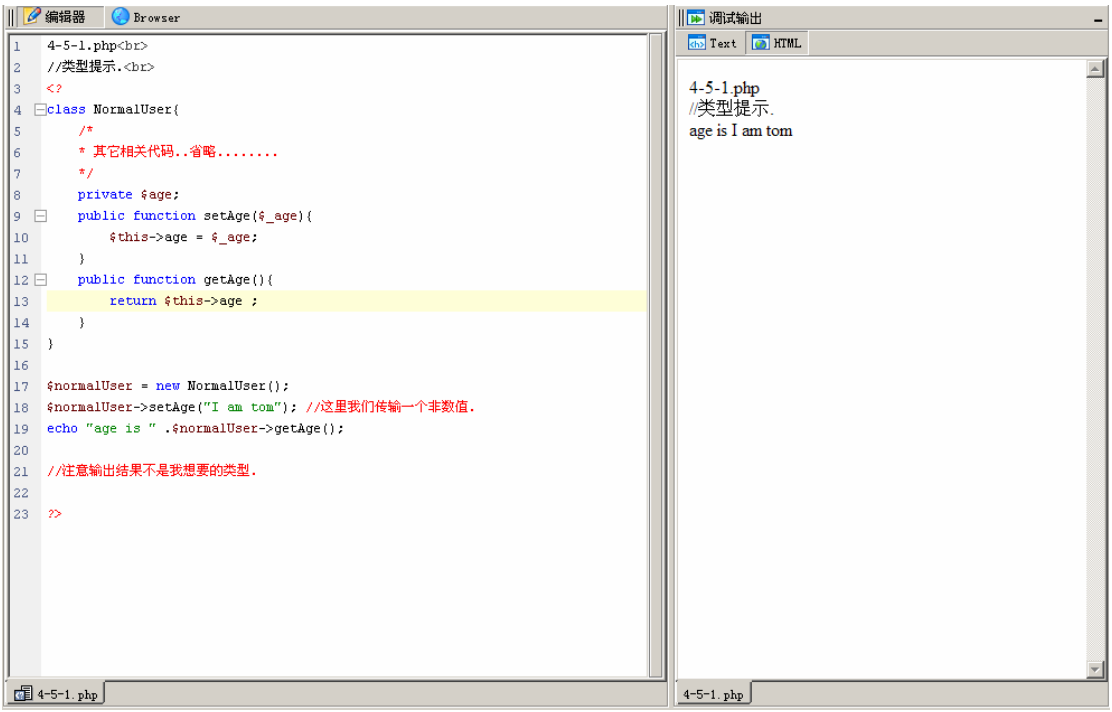
4.5 类型提示

PHP 是弱类型语言，向方法传递参数时候也不太区分类型。这样的使用会引起很多的问题，PHP 开发者认为，这些问题应该是由代码书写者在书写代码时进行检验以避免。

4.5.1 没有类型提示很危险

下面的代码可能会出现问题。

例： 4-5-1.php



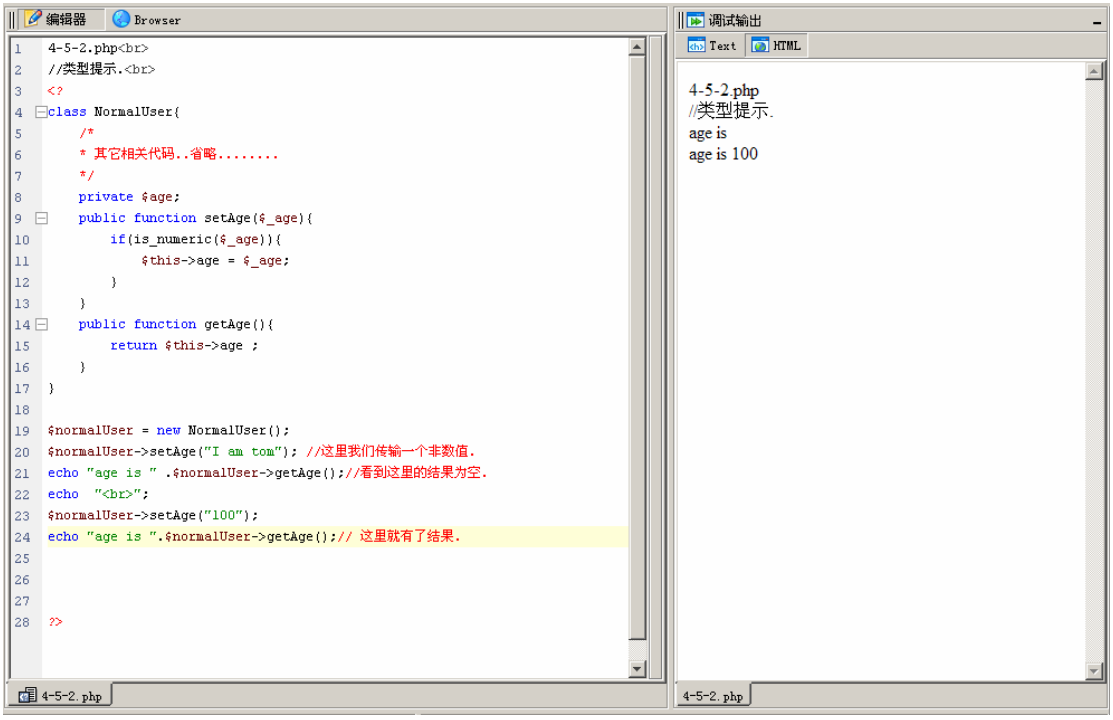
4.5.2 原始类型的类型判断

PHP 中提供了一些函数，来判断数值的类型。我们可使用 **is_numeric()**。判断是否是一个数值或者可转换为数值的字符串。

其它相关的还有 **is_bool()**、**is_int()**、**is_float()**、**is_integer()**、**is_numeric()**、**is_string()**、**is_array()** 和 **is_object()**。

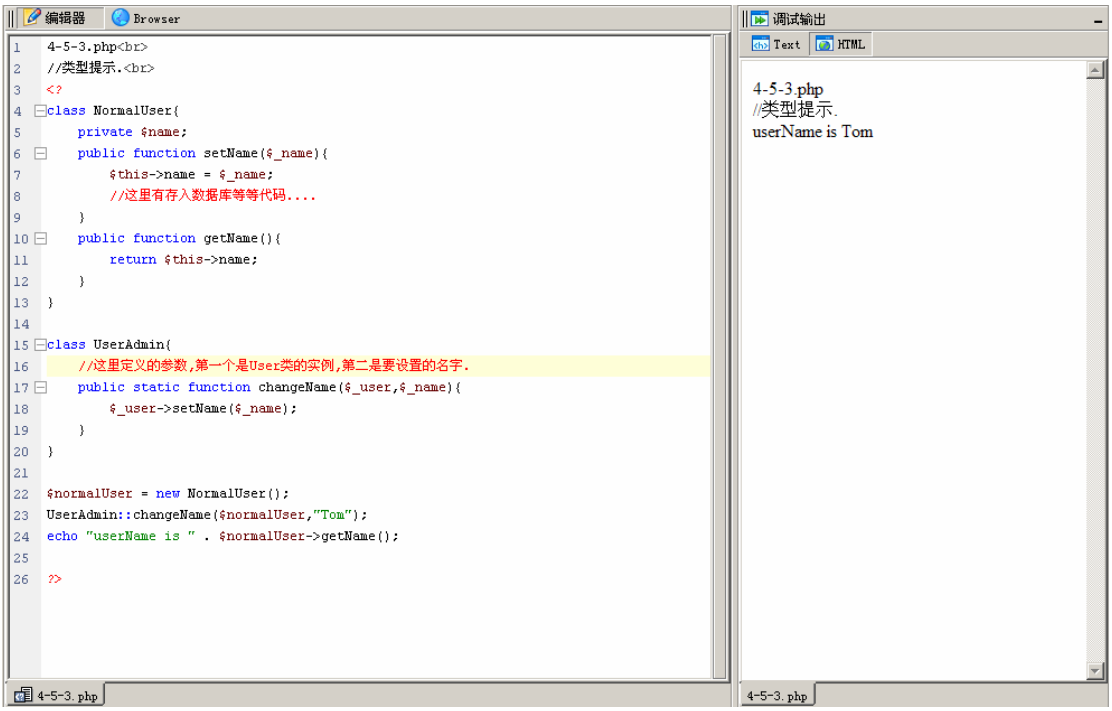
于是代码有了修改

例：4-5-2.php



4.5.3 向方法内传递对象

如果传递的参数是一个对象呢？
下面的代码用起来很正常。
例：4-5-3.php



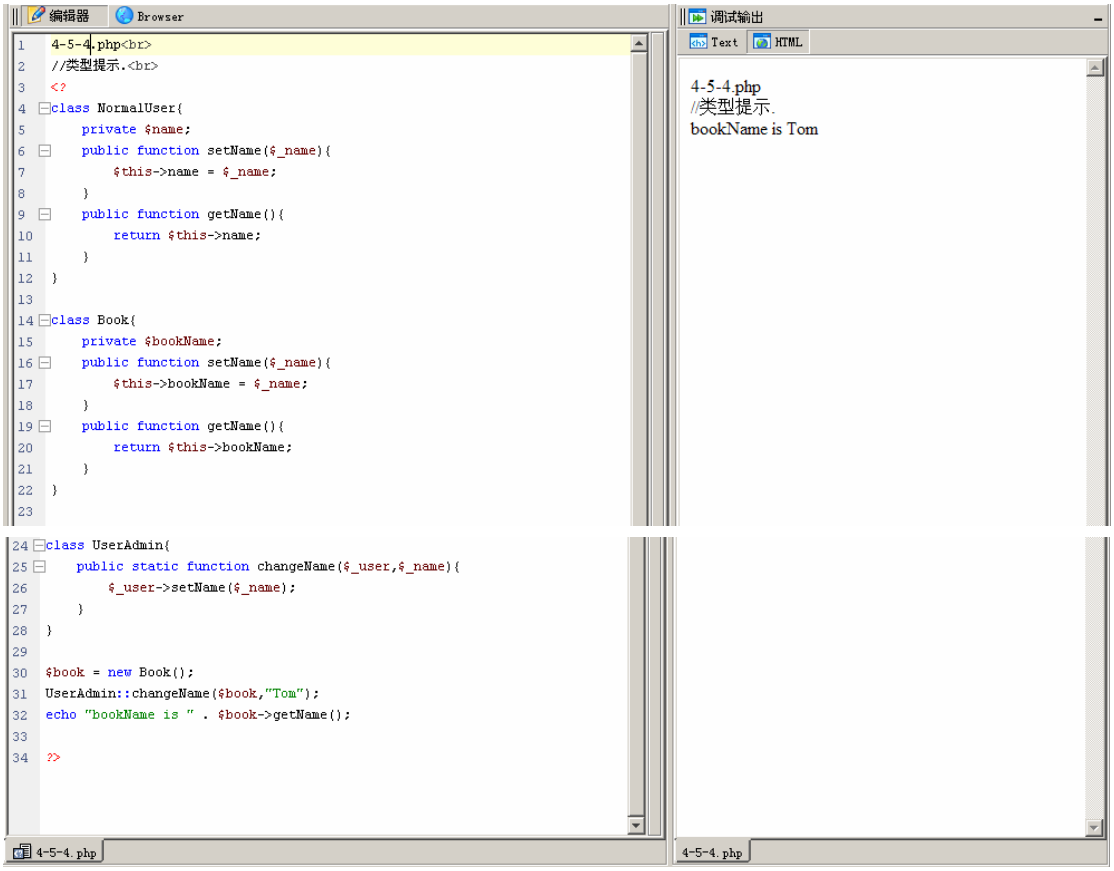
我们还有一个类，和图书相关的类，在图书类中也可以设置图书的书名 setName($name)。

如果我向，刚才代码中的方法 changeName() 中传入一个 Book 的实例，原定于改变人名的方法现在改变了书的书名。

这有什么风险？能把脏衣服扔到洗衣机里面去洗，同样的清洗，把盘子和碗都扔进洗衣机里面洗洗试试。

下面的代码演示我们不详看到的一幕。

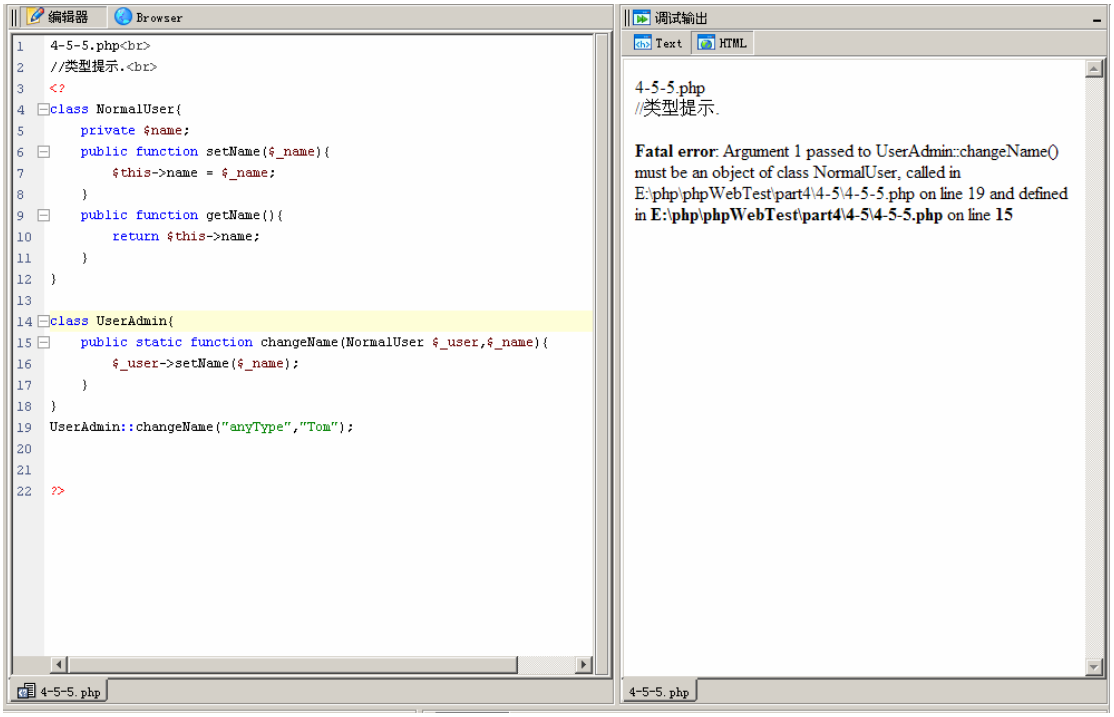
例：4-5-4.php




```
4-5-4.php<br>
//类型提示.<br>
<?
class NormalUser{
    private $name;
    public function setName($_name){
        $this->name = $_name;
    }
    public function getName(){
        return $this->name;
    }
}

class Book{
    private $bookName;
    public function setName($_name){
        $this->bookName = $_name;
    }
    public function getName(){
        return $this->bookName;
    }
}

class UserAdmin{
    public static function changeName($_user,$_name){
        $_user->setName($_name);
    }
}

$book = new Book();
UserAdmin::changeName($book,"Tom");
echo "bookName is " . $book->getName();

?>
```

4.5.4 类型提示保障数据安全

为了避免对象类型不规范引起的问题，PHP5 中引入了类型提示这个概念。
在定义方法参数时，同时定义参数的对象类型。
如果在调用的时候，传入参数的类型不对会报错。这样保证了数据的安全性。
例 4-5-5.php



- 建议在定义方法参数时，要使用类型提示。
- 如果类型不是对象，要采用代码进行类型建议，以增强安全性。

4.6 PHP5 中的多态

多态这个概念，在 Java 中指的是变量可以指向的对象的类型，可是变量声明类型的子类。对象一旦创建，它的类型是不变的，多态的是变量。

在 PHP5 中，变量的类型是不确定的，一个变量可以指向任何类型的数值、字符串、对象、资源等。我们无法说 PHP5 中多态的是变量。

我们只能说在 PHP5 中，多态应用在方法参数的类型提示位置。

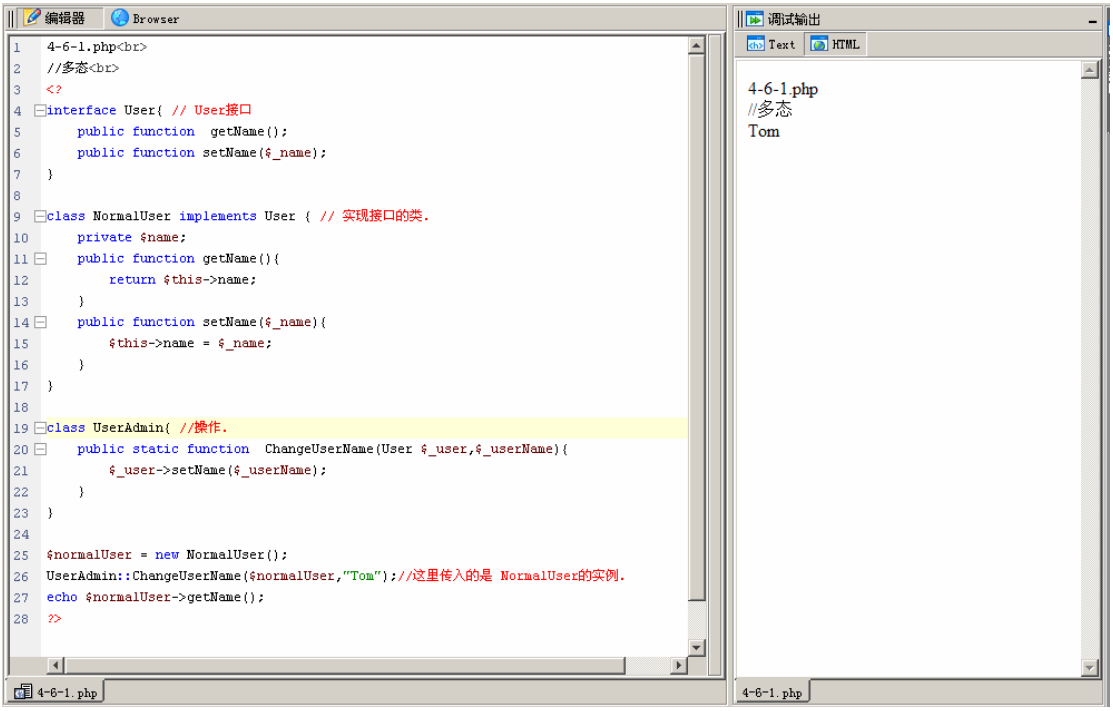
- 一个类的任何子类对象都可以满足以当前类型作为类型提示的类型要求。
- 所有实现这个接口的类，都可以满足以接口类型作为类型提示的方法参数要求。
- 简单的说，一个类拥有其父类、和已实现接口的身份。

4.6.1 通过实现接口实现多态

下面的例子中，UserAdmin 类的静态方法，要求一个 User 类型的参数。

在后面的使用中，传递了一个实现了 User 接口的类 NormalUser 的实例。代码成功运行。

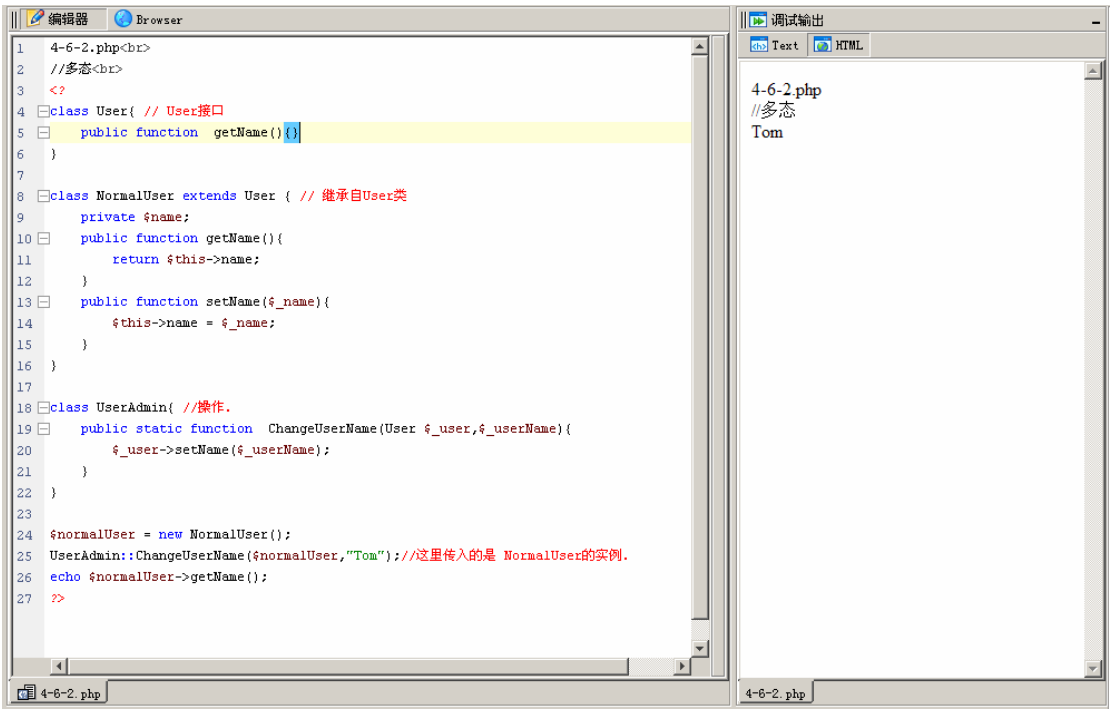
例 4-6-1.php



4.6.2 通过继承关系实现多态

下面是类和子类的关系。

例：4-6-2.php



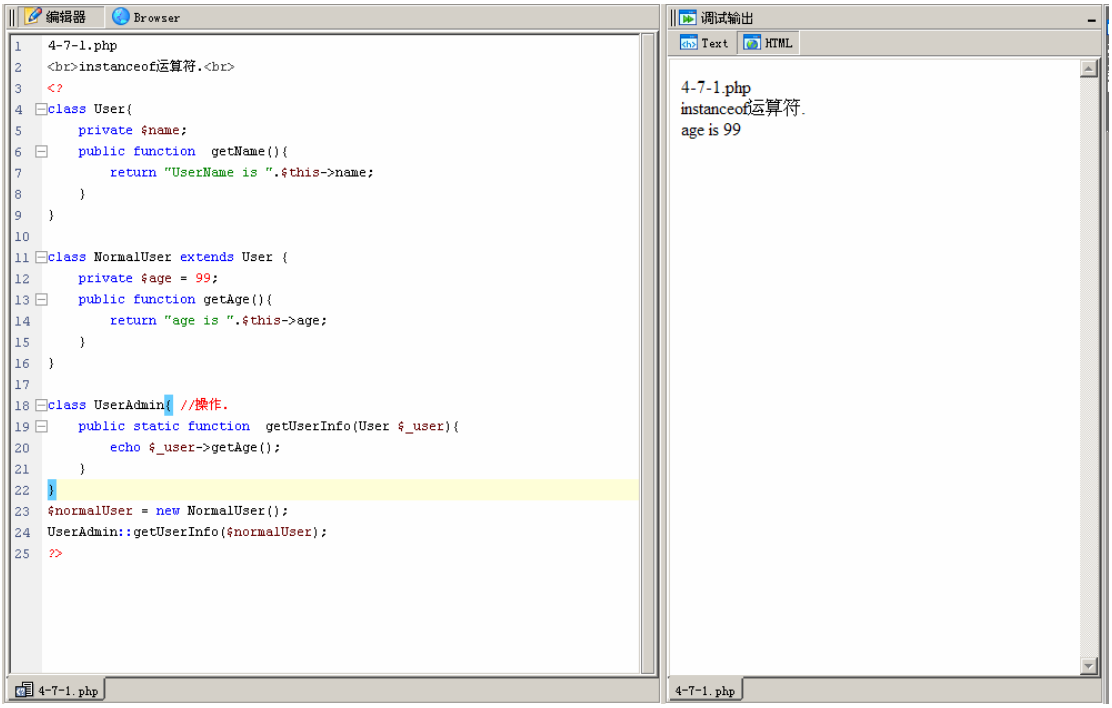
4.7 instanceof 运算符

在 PHP5 中，通过方法传递变量的类型有不确定性。
于是我们很难判断，一些操作是否可以运行。

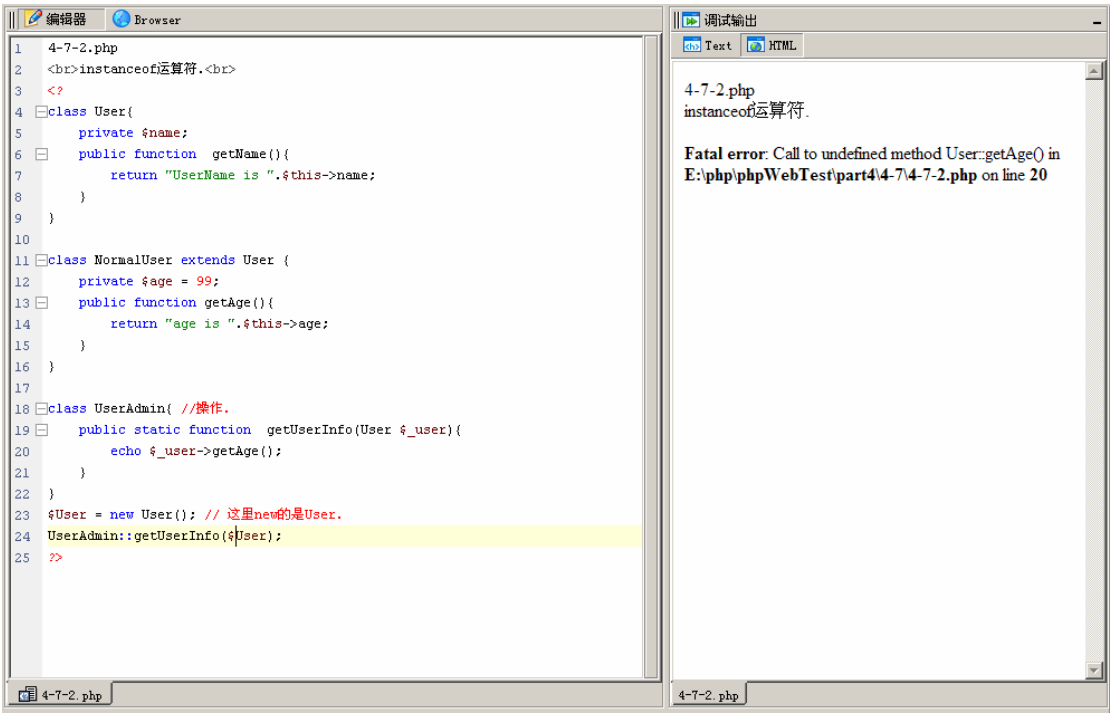
- 使用 instanceof 运算符，可以判断当前实例是否可以有这样的一个形态。
- 当前实例使用 instanceof 与 当前类，父类（向上无限追溯），已经实现的接口比较时，返回真。
- 代码格式　实例名 instanceof 类名

4.7.1 instanceof 运算符的运用

如下例子可以运行。
例 4-7-1.php



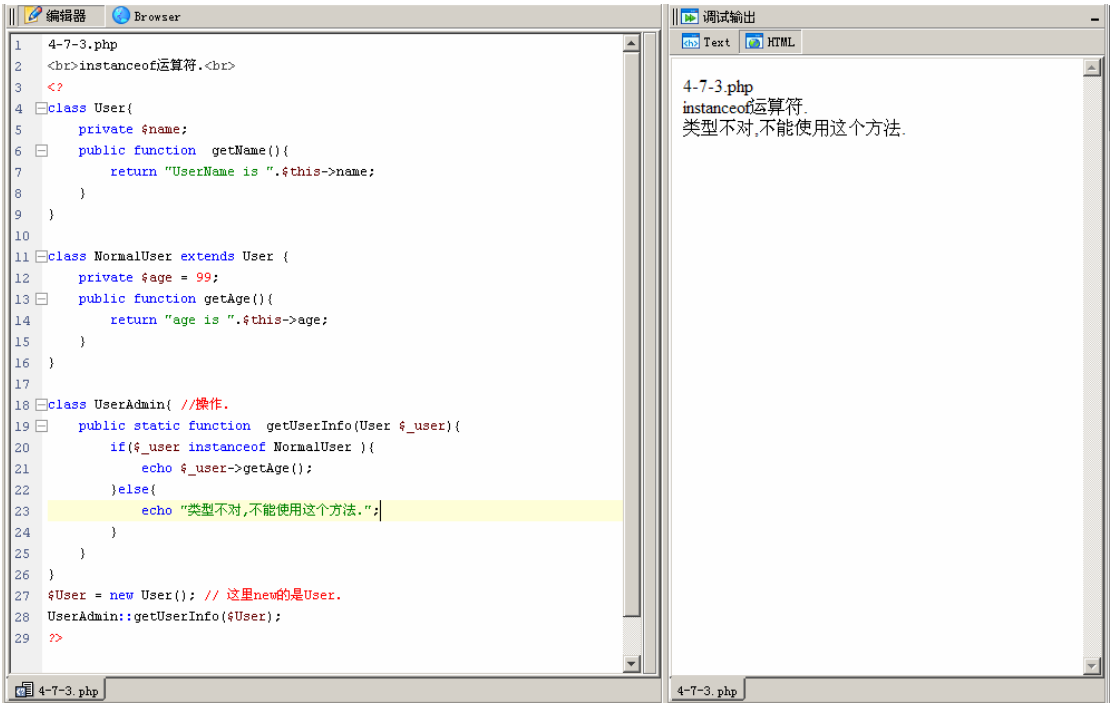
在 User 类中因为没有这个方法而报错，
例 4-7-2.php 就会出错。



4.7.2 使用 instatnceof 运算符保障代码安全

使用 instatnceof 运算符，在操作前先进行类型判断。以保障代码的安全性。

例：4-7-3.php

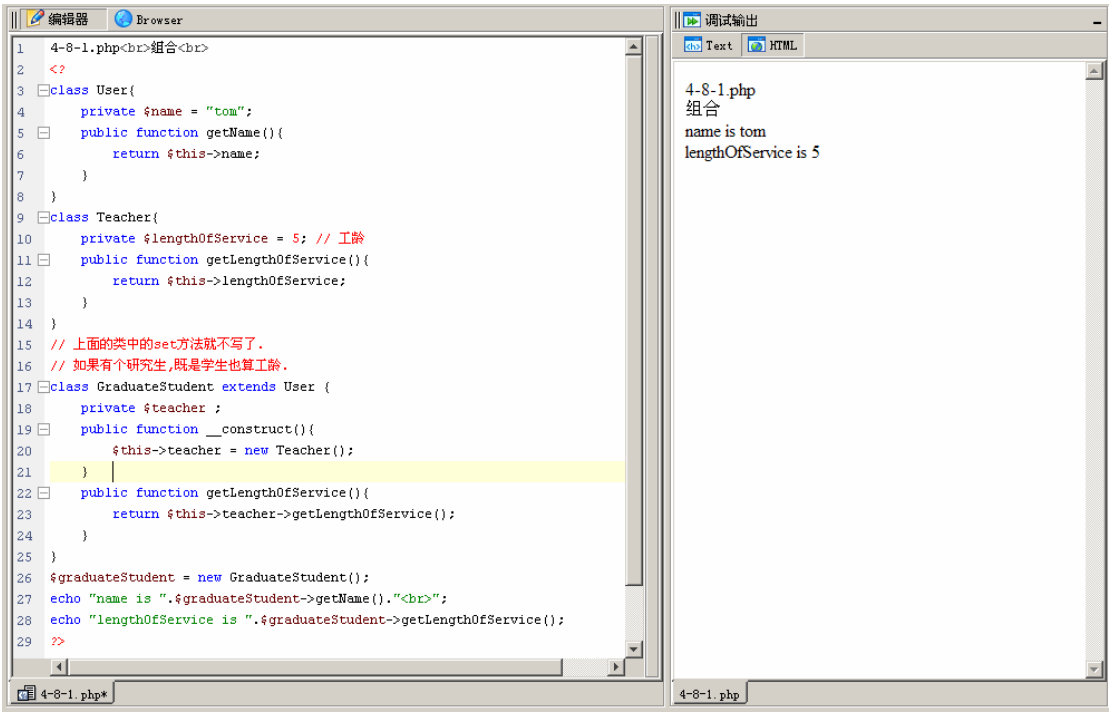


4.8 使用接口与组合模拟多继承

4.8.1 通过组合模拟多重继承

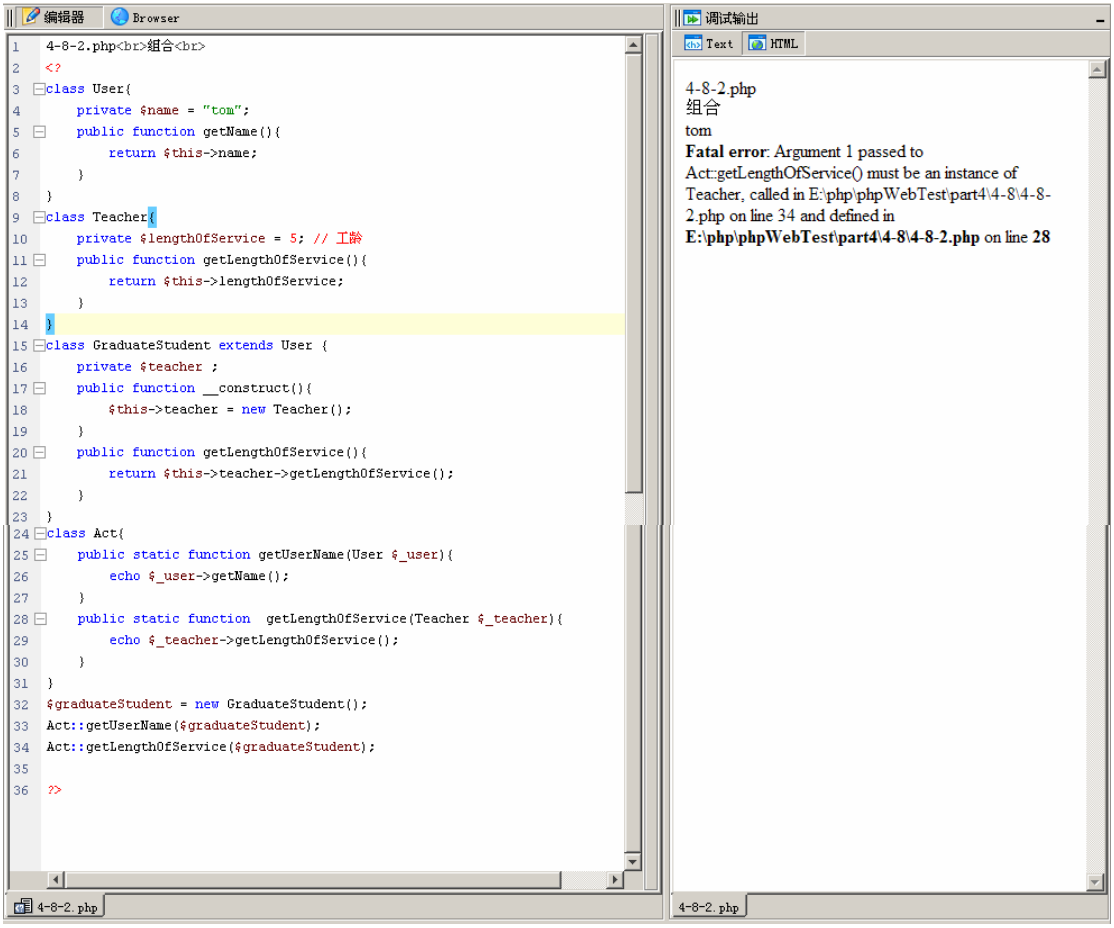
在 PHP 中不支持多重继承，如果我们向使用多个类的方法而实现代码重用有什么办法么？那就是组合。在一个类中去将另外一个类设置成属性。

下面的例子，模拟了多重继承。
例 4-8-1.php：



4.8.2 不完全的多重继承

上面的例子是多重继承么？
我们继续运行下面的代码。

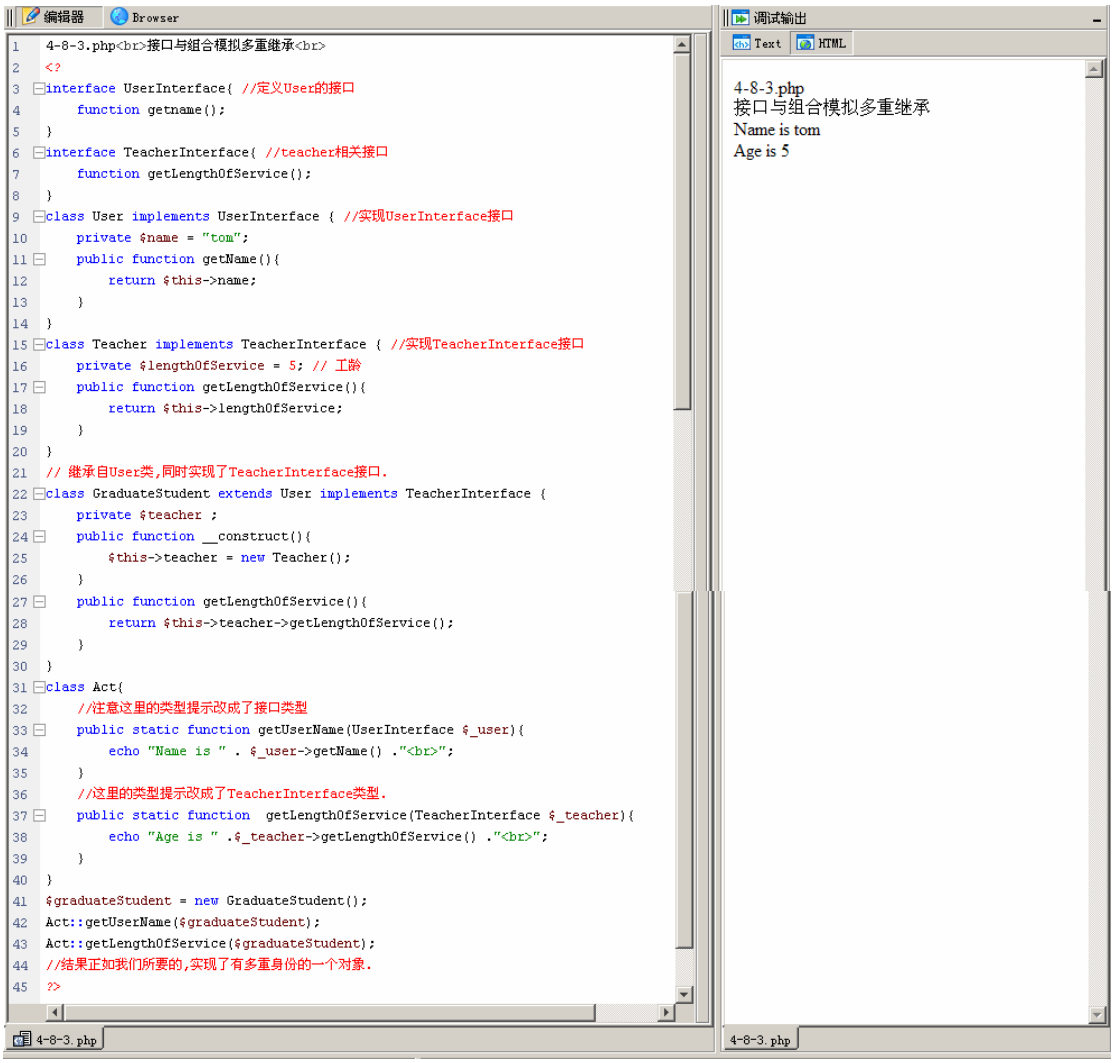


我们并不能将$graduateStudent 代表的实例当作 Teacher 类型传入。
这样的多重继承是失败的，没有多重身份，怎么能叫多重继承呢？

4.8.3 使用接口实现多重继承

通过把类的特征抽象为接口，并通过实现接口的方式让对象有多重身份，自然就可以完全模拟多重继承了。

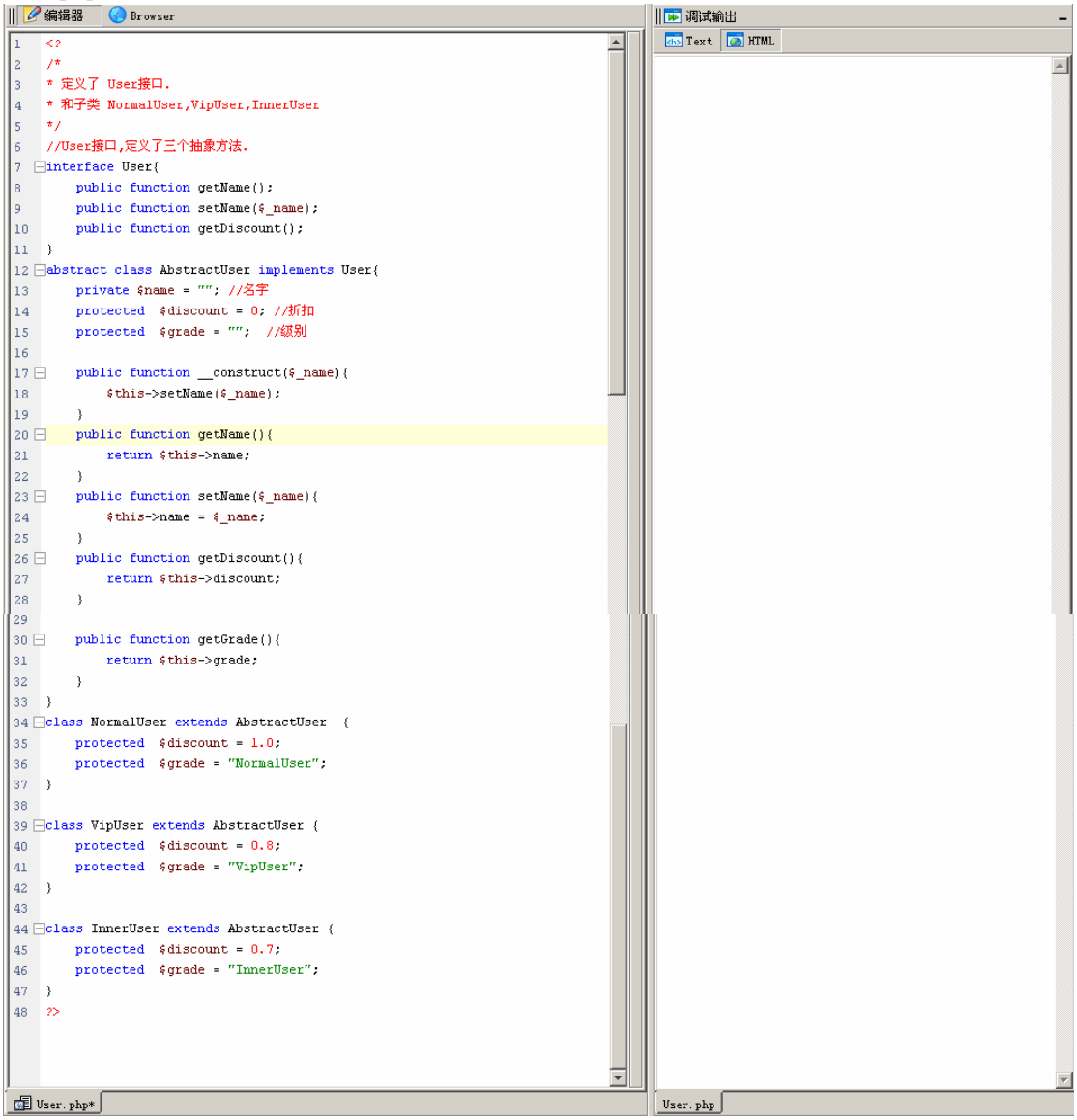
例：4-8-3.php



4.9 接口实例

写一个概念性的例子。
我们设计一个在线销售系统，用户部分设计如下：
将用户分为，NormalUser ,VipUser,InnerUser 三种。
要求根据用户的不同折扣计算用户购买产品的价格。
并要求为以后扩展和维护预留空间。

用户部分先声明了一个接口 User，用户都是 User 的实现。
User.php




```php
<?
/*
 * 定义了 User接口.
 * 和子类 NormalUser,VipUser,InnerUser
 */
//User接口,定义了三个抽象方法.
interface User{
    public function getName();
    public function setName($_name);
    public function getDiscount();
}
abstract class AbstractUser implements User{
    private $name = ""; //名字
    protected  $discount = 0; //折扣
    protected  $grade = "";  //级别

    public function __construct($_name){
        $this->setName($_name);
    }
    public function getName(){
        return $this->name;
    }
    public function setName($_name){
        $this->name = $_name;
    }
    public function getDiscount(){
        return $this->discount;
    }

    public function getGrade(){
        return $this->grade;
    }
}
class NormalUser extends AbstractUser  {
    protected  $discount = 1.0;
    protected  $grade = "NormalUser";
}

class VipUser extends AbstractUser {
    protected  $discount = 0.8;
    protected  $grade = "VipUser";
}

class InnerUser extends AbstractUser {
    protected  $discount = 0.7;
    protected  $grade = "InnerUser";
}
?>
```

关于产品，我们进行了如下设计。
声明一个接口 Product，然后从 Product 继承下 Book 接口。
在线销售的图书最后是实现了 Book 接口的 BookOnline 类。
Product.php

编辑器 Browser

```php
<?
/*与产品相关的类放.*/
Interface Product{ //定义产品接口
    public function getProductName();
    public function getProductPrice();
}

interface Book extends Product { // book是产品的一个分类
    public function getAuthor();
}

class BookOnline implements Book{ // 定义book类.
    private $productName;  // 产品名
    private $productPrice; // 产品价格
    private $author;  //作者

    public function __construct($_bookName){
        $this->productName = $_bookName;
        //这里放置相关初始化的代码.
        //与数据库关联的代码.
    }

    public function getProductName(){
        return $this->productName;
    }

    public function getProductPrice(){
        //这里从数据库读取价格.
        //假设价格是 100元.
        $this->productPrice = 100;
        return $this->productPrice;
    }

    public function getAuthor(){
        //从数据库里面取值.
        return $this->author;
    }
}


?>
```

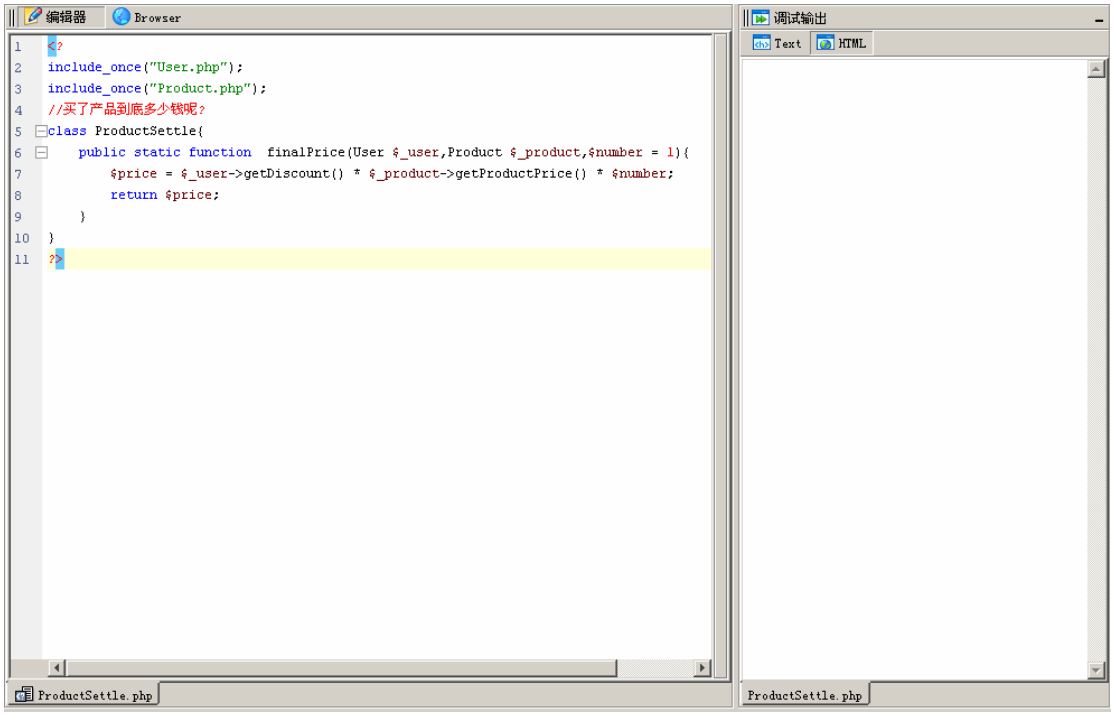
Product.php

调试输出

Text HTML

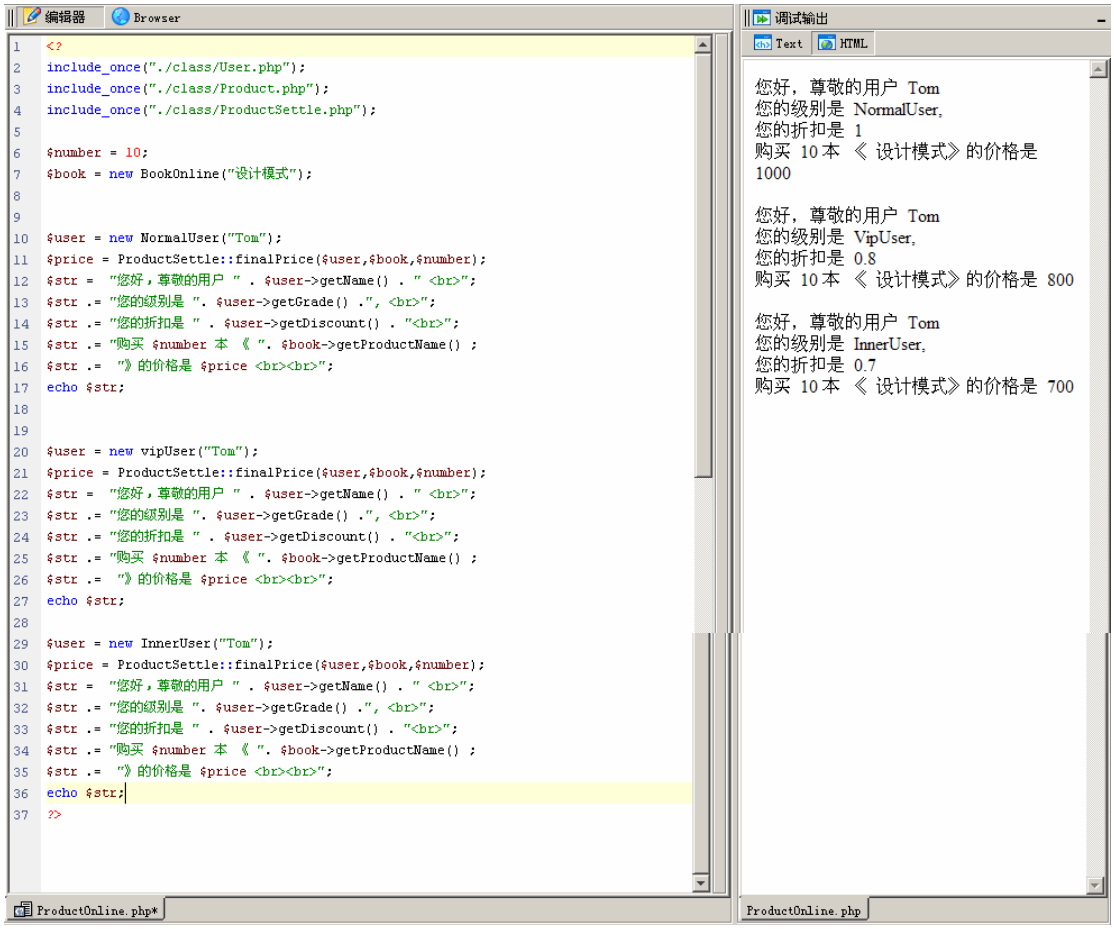
Product.php

关于结算，我们使用了独立的结算类，使用静态方法做计算。
产品结算。注意参数类型。




```php
<?
include_once("User.php");
include_once("Product.php");
//买了产品到底多少钱呢？
class ProductSettle{
    public static function  finalPrice(User $_user,Product $_product,$number = 1){
        $price = $_user->getDiscount() * $_product->getProductPrice() * $number;
        return $price;
    }
}
?>
```

下面的例子是实现。大家可以自己分析下。



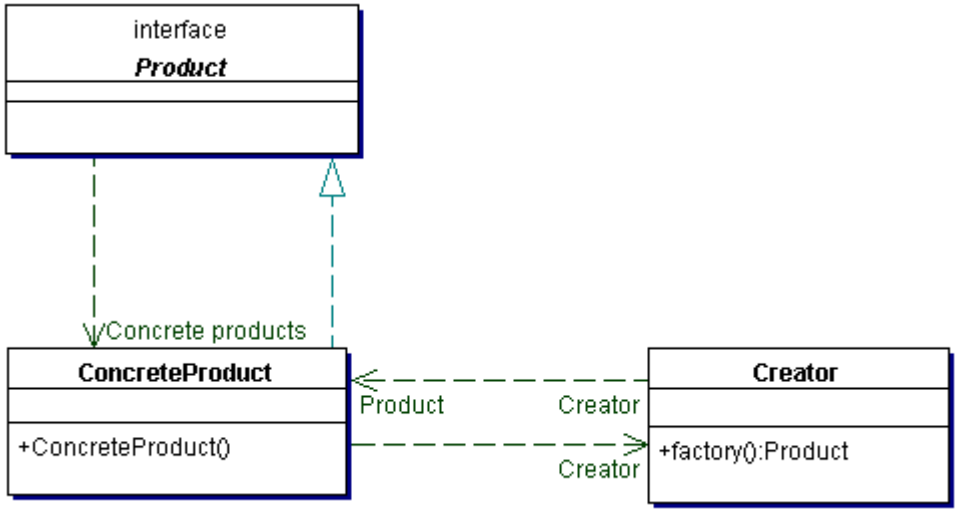

```php
<?
include_once("./class/User.php");
include_once("./class/Product.php");
include_once("./class/ProductSettle.php");

$number = 10;
$book = new BookOnline("设计模式");


$user = new NormalUser("Tom");
$price = ProductSettle::finalPrice($user,$book,$number);
$str =  "您好，尊敬的用户 " . $user->getName() . " <br>";
$str .= "您的级别是 ". $user->getGrade() .", <br>";
$str .= "您的折扣是 " . $user->getDiscount() . "<br>";
$str .= "购买 $number 本 《 ". $book->getProductName() ;
$str .=  "》的价格是 $price <br><br>";
echo $str;


$user = new vipUser("Tom");
$price = ProductSettle::finalPrice($user,$book,$number);
$str =  "您好，尊敬的用户 " . $user->getName() . " <br>";
$str .= "您的级别是 ". $user->getGrade() .", <br>";
$str .= "您的折扣是 " . $user->getDiscount() . "<br>";
$str .= "购买 $number 本 《 ". $book->getProductName() ;
$str .=  "》的价格是 $price <br><br>";
echo $str;

$user = new InnerUser("Tom");
$price = ProductSettle::finalPrice($user,$book,$number);
$str =  "您好，尊敬的用户 " . $user->getName() . " <br>";
$str .= "您的级别是 ". $user->getGrade() .", <br>";
$str .= "您的折扣是 " . $user->getDiscount() . "<br>";
$str .= "购买 $number 本 《 ". $book->getProductName() ;
$str .=  "》的价格是 $price <br><br>";
echo $str;
?>
```

4.10 简单工厂模式

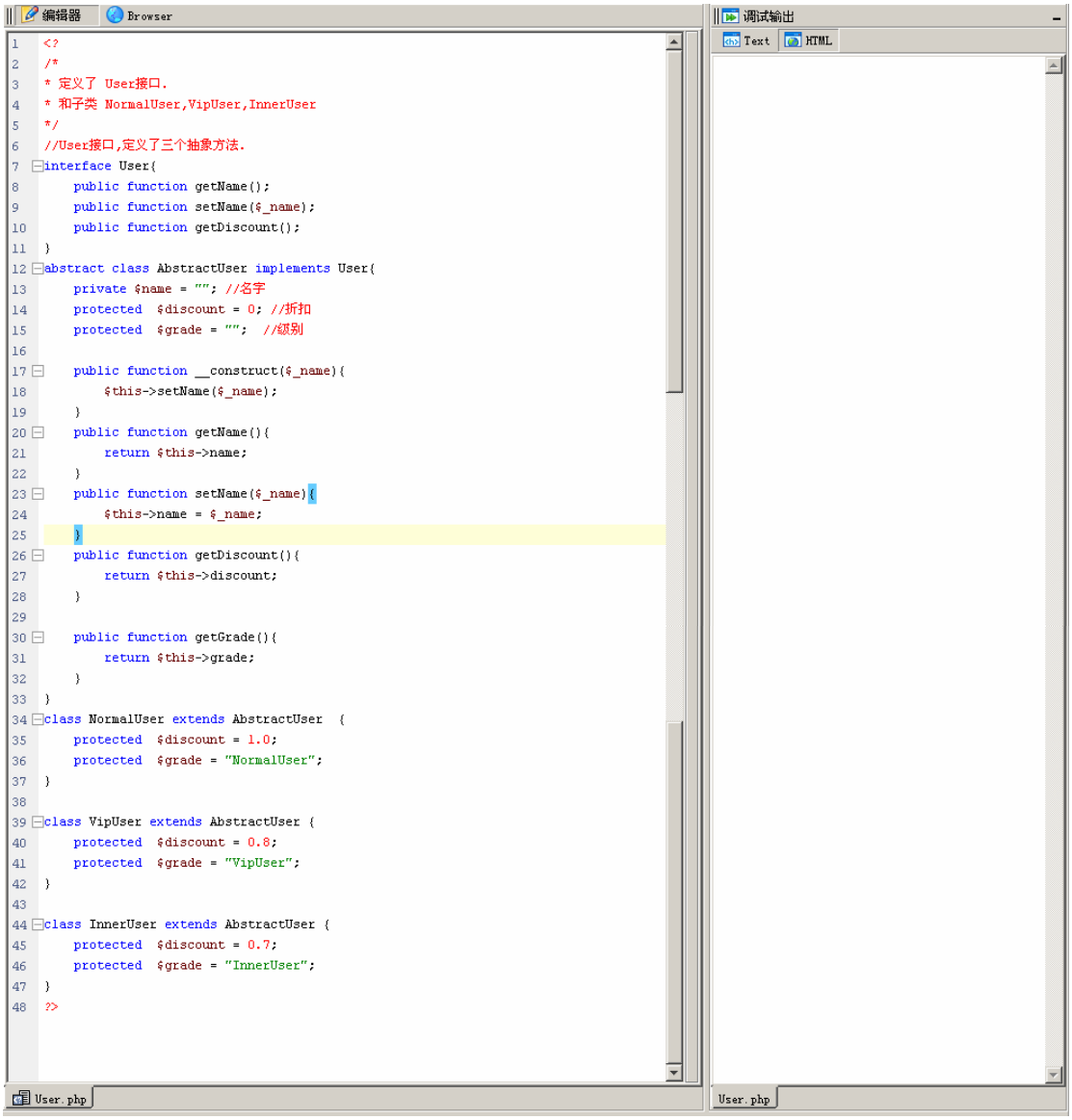
简单工厂模式是一种比较简单的设计模式，也是我们常用的设计模式。
使用简单工厂模式，能够根据不同的参数，使用不同的工厂，创建不同的对象。

简单工厂模式是类的创建模式,又叫做静态工厂方法(Static Factory Method)模式。

简单工厂模式是由一个工厂类根据传入的参量决定创建出哪一种产品类的实例。



说起来比较复杂，看一下就很简单了。
上一节的例子，在这一节我们做一些修改，我们用简单工厂模式建立。
大家注意比较改变前后代码的变化和使用。
User.php

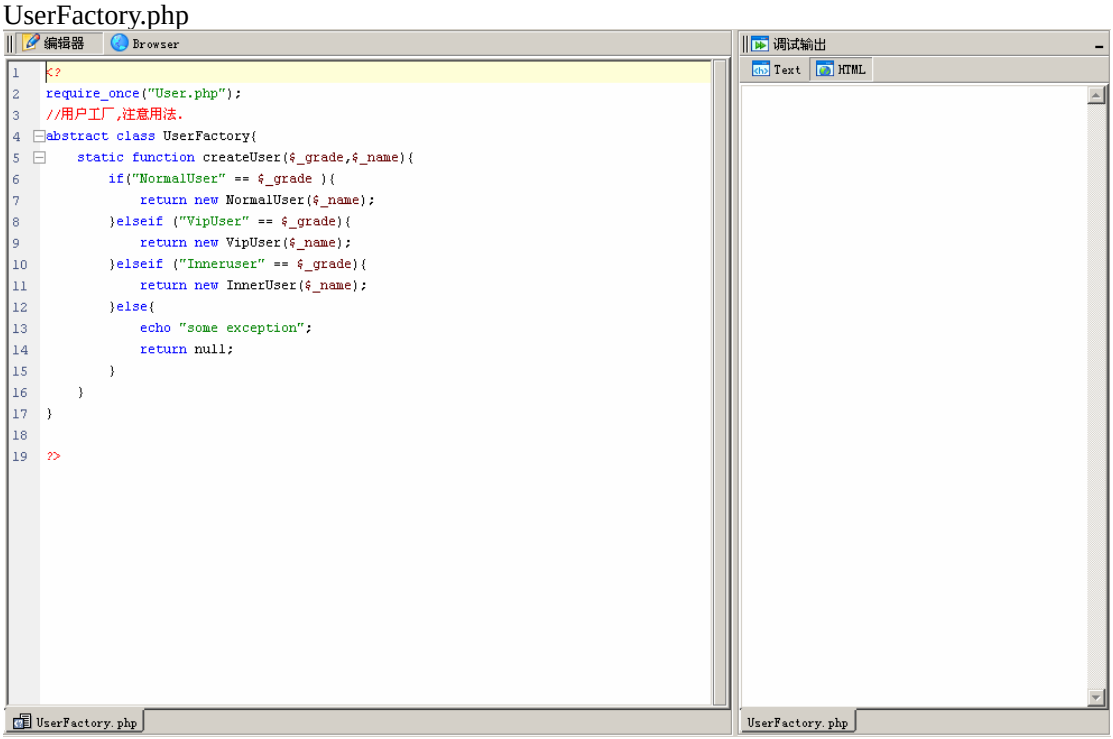



```php
<?
/*
* 定义了 User接口.
* 和子类 NormalUser,VipUser,InnerUser
*/
//User接口,定义了三个抽象方法.
interface User{
    public function getName();
    public function setName($_name);
    public function getDiscount();
}
abstract class AbstractUser implements User{
    private $name = ""; //名字
    protected  $discount = 0; //折扣
    protected  $grade = "";  //级别

    public function __construct($_name){
        $this->setName($_name);
    }
    public function getName(){
        return $this->name;
    }
    public function setName($_name){
        $this->name = $_name;
    }
    public function getDiscount(){
        return $this->discount;
    }

    public function getGrade(){
        return $this->grade;
    }
}
class NormalUser extends AbstractUser  {
    protected  $discount = 1.0;
    protected  $grade = "NormalUser";
}

class VipUser extends AbstractUser {
    protected  $discount = 0.8;
    protected  $grade = "VipUser";
}

class InnerUser extends AbstractUser {
    protected  $discount = 0.7;
    protected  $grade = "InnerUser";
}
?>
```

用户工厂

用户工厂根据传递来的参数，创建不同的对象。但他们拥有共同的形态 User

UserFactory.php




```php
<?
require_once("User.php");
//用户工厂,注意用法.
abstract class UserFactory{
    static function createUser($_grade,$_name){
        if("NormalUser" == $_grade ){
            return new NormalUser($_name);
        }elseif ("VipUser" == $_grade){
            return new VipUser($_name);
        }elseif ("Inneruser" == $_grade){
            return new InnerUser($_name);
        }else{
            echo "some exception";
            return null;
        }
    }
}

?>
```

产品的定义在这个类中实现

Product.php

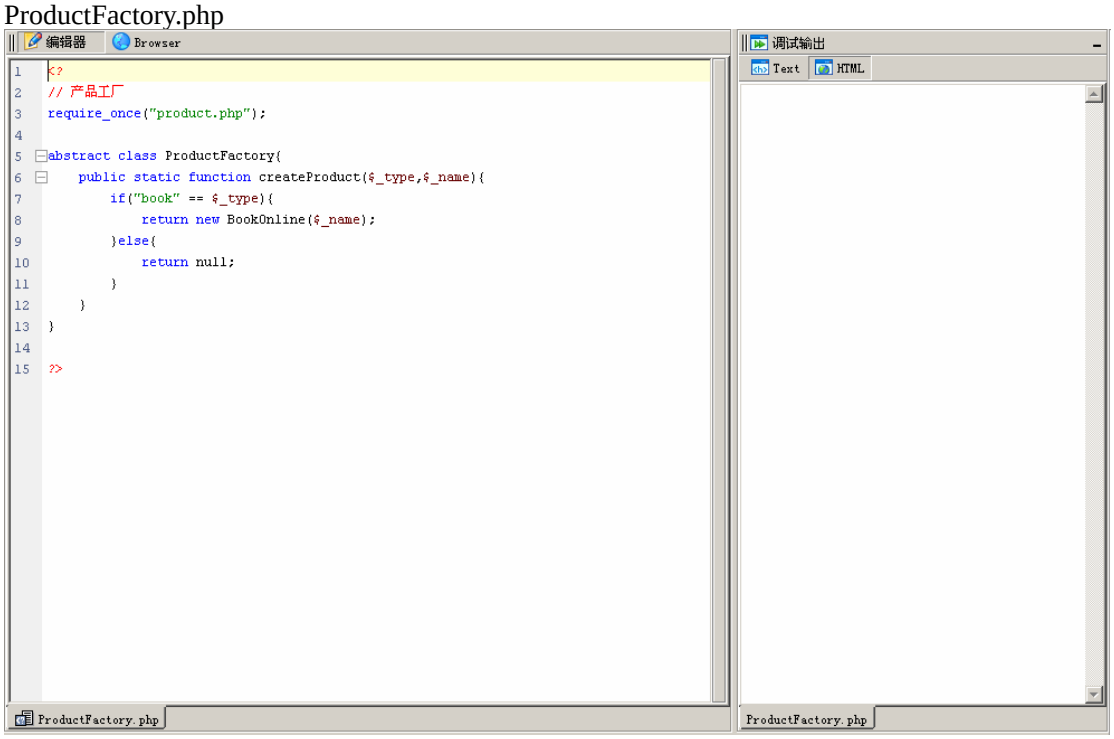
编辑器 Browser

```php
<?
/*与产品相关的类放.*/
Interface Product{ //定义产品接口
    public function getProductName();
    public function getProductPrice();
}

interface Book extends Product { // book是产品的一个分类
    public function getAuthor();
}

class BookOnline implements Book{ // 定义book类.
    private $productName;  // 产品名
    private $productPrice; // 产品价格
    private $author;  //作者

    public function __construct($_bookName){
        $this->productName = $_bookName;
        //这里放置相关初始化的代码.
        //与数据库关联的代码.
    }

    public function getProductName(){
        return $this->productName;
    }

    public function getProductPrice(){
        //这里从数据库读取价格.
        //假设价格是 100元.
        $this->productPrice = 100;
        return $this->productPrice;
    }

    public function getAuthor(){
        //从数据库里面取值.
        return $this->author;
    }
}


?>
```

调试输出 Text HTML

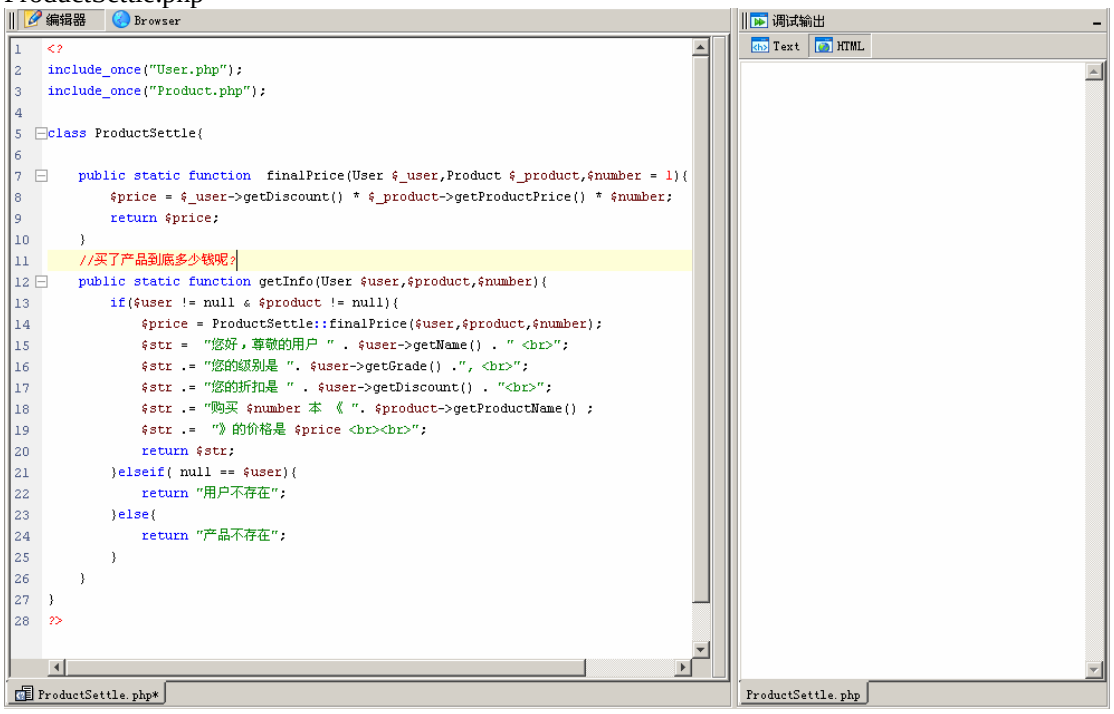
Product.php

产品工厂，根据参数创建不同的对象。
现在的工厂只能创建 book 的对象，以后有了新产品，来这里加上就好了。
ProductFactory.php



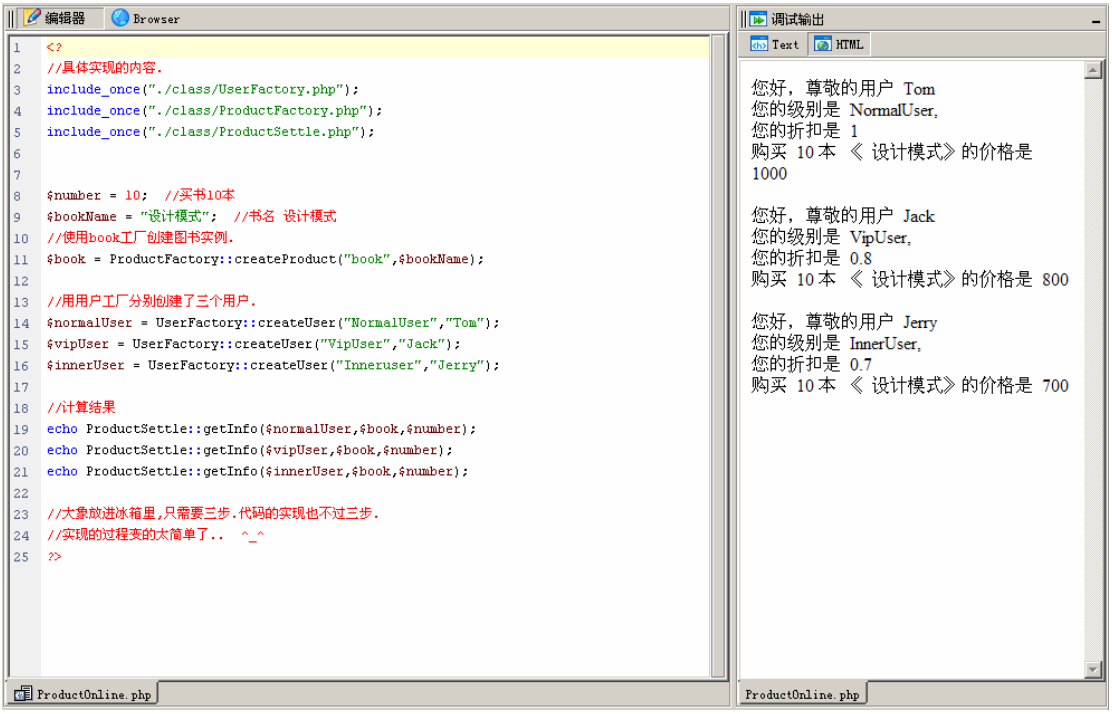
```php
<?
// 产品工厂
require_once("product.php");

abstract class ProductFactory{
    public static function createProduct($_type,$_name){
        if("book" == $_type){
            return new BookOnline($_name);
        }else{
            return null;
        }
    }
}

?>
```

定义一个操作类，里面的操作关联性不强，就用静态方法方便使用。
ProductSettle.php



```php
<?
include_once("User.php");
include_once("Product.php");

class ProductSettle{

    public static function  finalPrice(User $_user,Product $_product,$number = 1){
        $price = $_user->getDiscount() * $_product->getProductPrice() * $number;
        return $price;
    }
    //买了产品到底多少钱呢?
    public static function getInfo(User $user,$product,$number){
        if($user != null & $product != null){
            $price = ProductSettle::finalPrice($user,$product,$number);
            $str =  "您好，尊敬的用户 " . $user->getName() . " <br>";
            $str .= "您的级别是 ". $user->getGrade() .", <br>";
            $str .= "您的折扣是 " . $user->getDiscount() . "<br>";
            $str .= "购买 $number 本 《 ". $product->getProductName() ;
            $str .=  "》的价格是 $price <br><br>";
            return $str;
        }elseif( null == $user){
            return "用户不存在";
        }else{
            return "产品不存在";
        }
    }
}
?>
```

最后的实现。
上面所有的内容都定义完毕了，使用起来很方便。
只需要三步就可以了。



小结

这一章介绍了接口、多态和简单工厂模式。
希望大家对面向对象有了更多的了解。
有什么建议就到论坛给我留言好了。
http://www.phpchina.com/bbs/thread-10556-1-1.html

最近太忙了，有个孩子很累人啊。
这一章没有怎么太多的组织例子。对例子的描述也简单了些。希望大家包涵。
写到这里，对于面向对象的内容大体上应该有个认识了。
也应该从此开始，在写代码时候考虑如何面向对象了。
在 PHP 程序员中，这些概念少了些。而 Java 程序员，从任何一本书的第一节开始就是面向对象的。由此，很多偏见的观点只有 Java 和 .net 才能相提并论，PHP 总是和 Asp 放在一起。
也许在你了解了面向对象之后，可以看看 Java 的初级教程，会觉得一样简单。

下一章，将介绍 PHP5 中的异常处理。

刀客羽朋
于石家庄 2006-12-24

第五章 PHP5 异常处理

目录

5.1 PHP 中的错误信息

5.1.1 php.ini 中配置错误消息

在 PHP4 中，没有异常 **Exception** 这个概念，只有 错误 **Error**。我们可以通过修改 php.ini 文件来配置用户端输出的错误信息。

在 php.ini 中，一个分号 **；** 表示注释。

Php.ini 将能够显示的错误类型分为如下种类。

```
; E_ALL                -所有的错误和警告，（不包含 E_STRICT）.
; E_ERROR              -致命的运行时错误
; E_RECOVERABLE_ERROR  - 几乎致命的运行时错误
; E_WARNING            - 运行时的警告（非致命错误）
; E_PARSE              -编译时解析错误
; E_NOTICE             - 运行时的提示，这些提示常常是代码中的 bug 引起的，也许是故意的（如使用一
                         个未初始化的变量，事实上它被自动初始化成一个空字符串） 。
; E_STRICT             - 运行时提示，能够给于 PHP 建议，以改变你的代码，以获得最好的协同性，并完
善代码的兼容性。
; E_CORE_ERROR         - PHP 初始化启动过程中的致命错误。
; E_CORE_WARNING       - PHP 初始化启动过程中的非致命错误。
; E_COMPILE_ERROR      - 致命的编译错误。
; E_COMPILE_WARNING    - 编译错误（非致命的错误）。
; E_USER_ERROR         - 用户错误信息。
; E_USER_WARNING       - 用户警告信息。
; E_USER_NOTICE        -用户提示信息。；
```

在 php.ini 中 error_reporting 控制输出到用户端的消息种类。

以下几种是 php.ini 中推荐的几种配置。

```
error_reporting  = E_ALL
表示输出所有的信息。
```

```
error_reporting  = E_ALL & ~E_NOTICE 表示输出所有的错误，除了提示。
```

```
error_reporting  = E_COMPILE_ERROR|E_RECOVERABLE_ERROR|E_ERROR|E_CORE_ERROR
表示输出所有的 ERROR 信息。
```

在 php.ini 中，display_errors 可以设置是否将以上设置的错误信息输出到用户端。

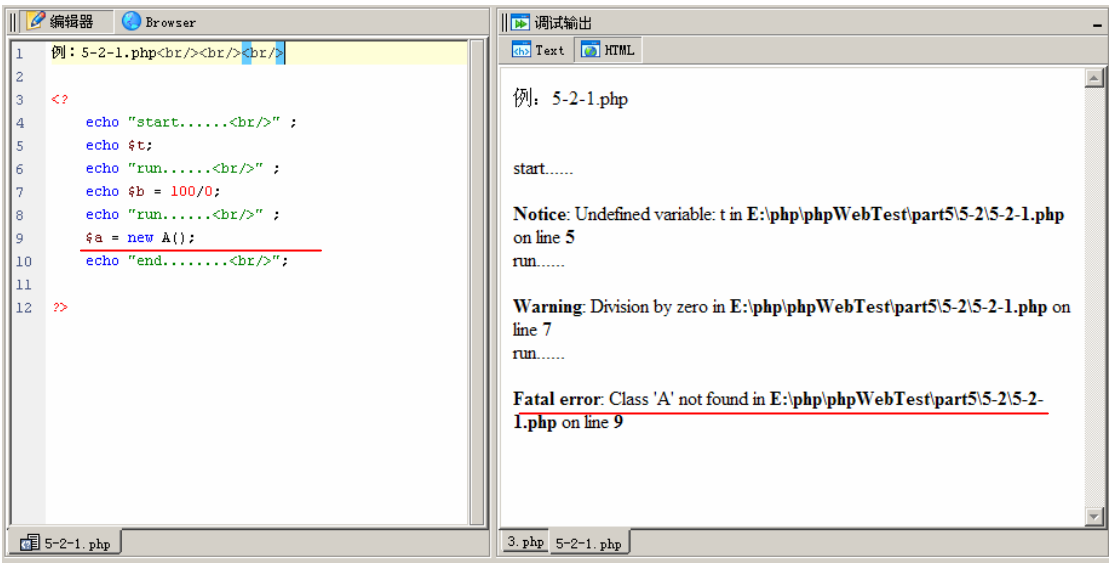
```
display_errors = On   输出到用户端（调试代码时候，打开这项更方便）
```

```
display_errors = OFF 消息将不会输出到用户端（最终发布给用户时记得改成 off）
```

5.1.2 php 中错误消息处理

在 php 中，对于错误处理非常的宽松。php 系统会尽量让程序运行下去，除非遇到致命错误。

例 5-2-1.php



第 5 行，直接打印一个未赋值变量$t 时候，系统报出一个 Notice，未定义变量。

第 7 行，做除以 0 的运算时，系统报出一个 Warning，提示有除以 0 这样的警告，程序依然在运行。

第 9 行，当实例化一个不存在的类的时候，发生致命错误，程序终止运行。

再次提示：如果不想显示错误信息给用户看到，设置 php.ini 中
display_errors = OFF

5.1.3 php 代码中调整错误级别

除了在 php.ini 文件中可以调整错误消息的显示级别外，在 php 代码中也可以自定义消息显示的级别。

PHP 提供了一个方便的调整函数。

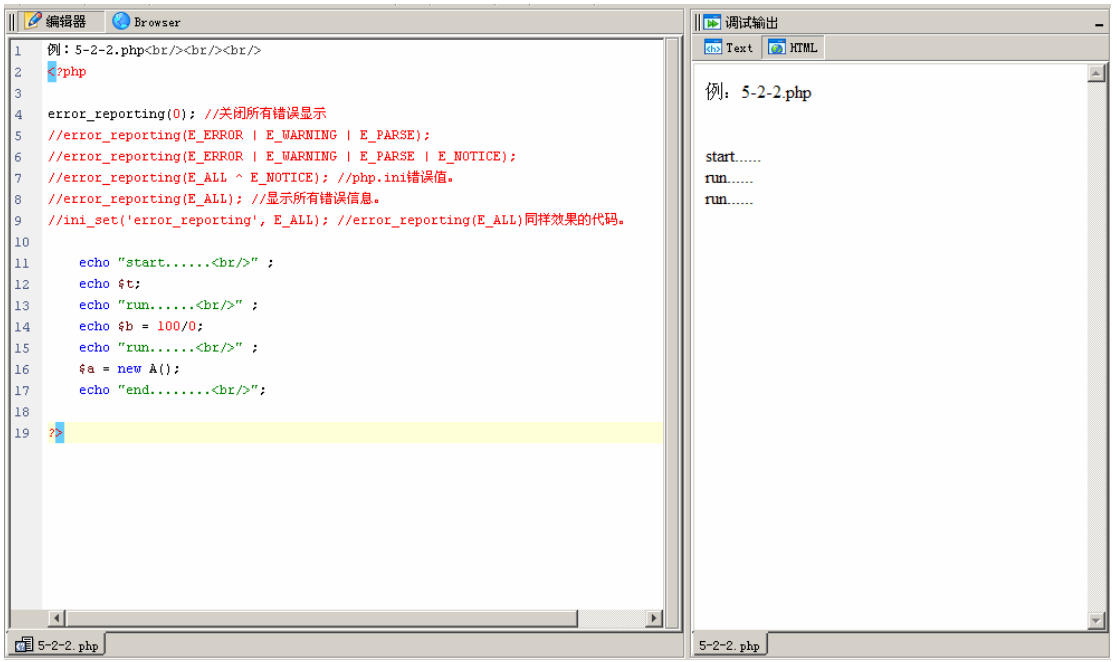
```
int error_reporting ( [int level] )
```

使用这个函数可以定义当前 php 页面中错误消息的显示级别。

参数 level 使用了二进制掩码组合的方式。

value	constant	value	constant
1	E_ERROR	2	E_WARNING
4	E_PARSE	8	E_NOTICE
16	E_CORE_ERROR	32	E_CORE_WARNING
64	E_COMPILE_ERROR	128	E_COMPILE_WARNING
256	E_USER_ERROR	512	E_USER_WARNING
1024	E_USER_NOTICE	2047	E_ALL
2048	E_STRICT	4096	E_RECOVERABLE_ERROR

例 5-2-2.php



5.1.4 自定义错误处理

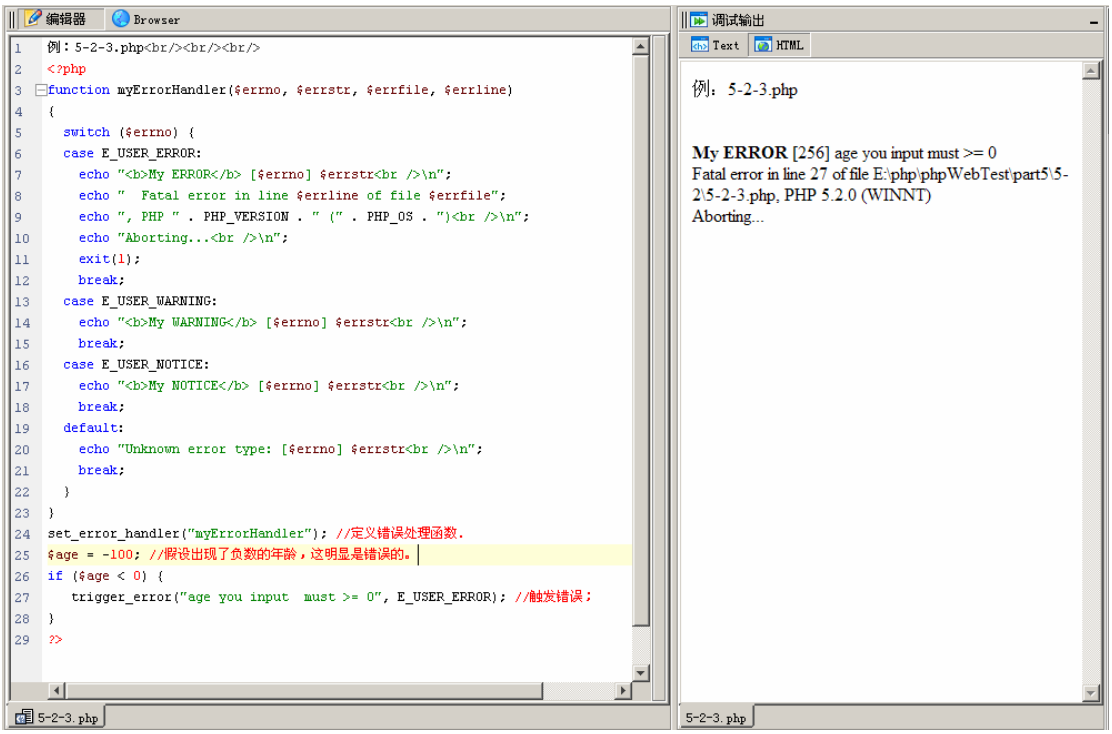
在 php 中，可以自定义对错误处理的方式。

首先要自定义一个错误处理函数，然后使用 **set_error_handler()**函数向系统声明错误处理函数。代码中产生的错误就会使用这个错误处理函数了。

使用 trigger_error()函数，可以触发一个 error。

例如 trigger_error("age you input　must >= 0", E_USER_ERROR)，触发自己的错误信息。

例：5-2-3.php



5.2 PHP5 中的 SPL 模块

5.2.1 什么是 SPL

在 PHP5 中有一个独特的部分 **SPL** - **StandardPHPLibrary Modules** （PHP 标准库）。

在 SPL 文档中这样说明的：

SPL – PHP 标准库是用来解决标准问题并实现一些高效数据访问的接口和类的集合。你会发现这些类用 PHP 代码编写在 sql.php 文件中，或在对应例子、内核的 .inc 文件中。基于这些内核的实现或在示例目录下的，也是一些 .php 文件。

在 SPL 中主要包含了以下内容：

1. **Iterators** : SPL offers some advanced iterator algorithms。
   迭代器: SPL 提供了一些高级的迭代器运算法则。
2. **Directories and Files** : SPL offers two advanced directory and file handling classes:
   目录和文件：SPL 提供了两个高级路径和文件处理类。
3. **XML** : SPL offers an advanced XML handling class:
   XML ： SPL 提供了一个高级 XML 处理类。
4. **Array Overloading** : SPL offers advanced Array overloading:
   数组重载 ：SPL 提供了高级数组重载。
5. **Counting** : interface Countable allows to hook into the standard array function count().
   计数：接口 Countable 允许勾住标准数组方法 count().
6. **Exceptions** : SPL provides a set of standard Exception classes each meant to indicate a certain problem type.
   异常：SPL 规定了一套标准异常类，每个类都标识了一类确定的问题。
7. **Observer**: SPL suggests a standard way of implementing the observer pattern.
   观察者：SPL 提出了实现观察者模式的标准。

关于SPL更多资料，请看 http://www.php.net/~helly/php/ext/spl 。

而后面我们讲只介绍 SPL 中的异常处理部分。

5.2.2 spl.php 中的异常处理类

从 PHP5.0 开始，在 SPL 中引入了异常处理类。

Notice：异常与错误在 PHP 中是两个完全不同的概念。

在 PHP 源码包中能找到这个文件 spl.php，在 spl.php 中定义了一个异常类 Exception。在这个类中，定义了一些属性如下：

protected $message ；
存储异常信息的变量。

private $string;
格式化过以后的异常信息。

protected $code;
通过构造函数传递的 异常代码。

protected $file;
产生异常的 php 文件的文件名。

protected $line;
引起异常的代码在 php 文件中所在的行数。

private $trace;
引起异常后，包含相关信息的一个数组。

构造函数如下：

```
function __construct($message = NULL, $code = 0) {
    if (func_num_args()) {    // func_num_args()返回参数数量
        $this->message = $message;
    }
    $this->code = $code; //错误代码默认是 0；
    $this->file = __FILE__; //  文件名
    $this->line = __LINE__; //  行号
    $this->trace = debug_backtrace(); //返回一个包含多个元素
    $this->string = StringFormat($this); //格式化字符串
}
```

其中还包含了__clone() 方法和对应这些属性的 geter 方法。

spl.php 源代码如下：

编辑器 Browser

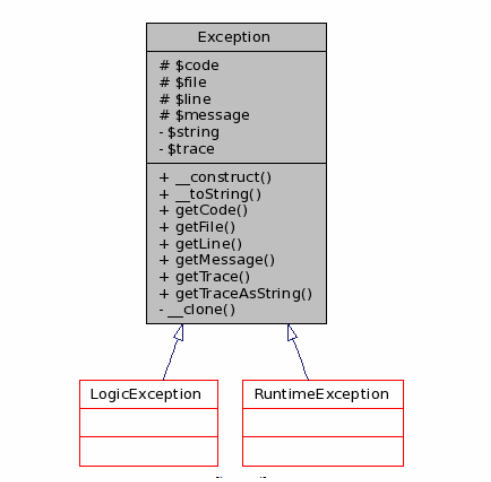
```php
 */
class Exception
{
    /** The exception message */
    protected $message;

    /** The string represenations as generated during construction */
    private $string;

    /** The code passed to the constructor */
    protected $code;

    /** The file name where the exception was instantiated */
    protected $file;

    /** The line number where the exception was instantiated */
    protected $line;

    /** The stack trace */
    private $trace;

    /** Prevent clone
     */
    final private function __clone() {}

    /** Construct an exception
     *
     * @param $message Some text describing the exception
     * @param $code    Some code describing the exception
     */
    function __construct($message = NULL, $code = 0) {
        if (func_num_args()) {
            $this->message = $message;
        }
        $this->code = $code;
        $this->file = __FILE__; // of throw clause
        $this->line = __LINE__; // of throw clause
        $this->trace = debug_backtrace();
        $this->string = StringFormat($this);
    }

    /** @return the message passed to the constructor
     */
    final public function getMessage()
    {
        return $this->message;
    }

    /** @return the code passed to the constructor
     */
    final public function getCode()
    {
        return $this->code;
    }

    /** @return the name of the file where the exception was thrown
     */
    final public function getFile()
    {
        return $this->file;
    }

    /** @return the line number where the exception was thrown
     */
    final public function getLine()
    {
        return $this->line;
    }
```

```
308     /** @return the stack trace as array
309      */
310     final public function getTrace()
311     {
312         return $this->trace;
313     }
314
315     /** @return the stack trace as string
316      */
317     final public function getTraceAsString()
318     {
319     }
320
321     /** @return string represenation of exception
322      */
323     public function __toString()
324     {
325         return $this->string;
326     }
327 }
328
```

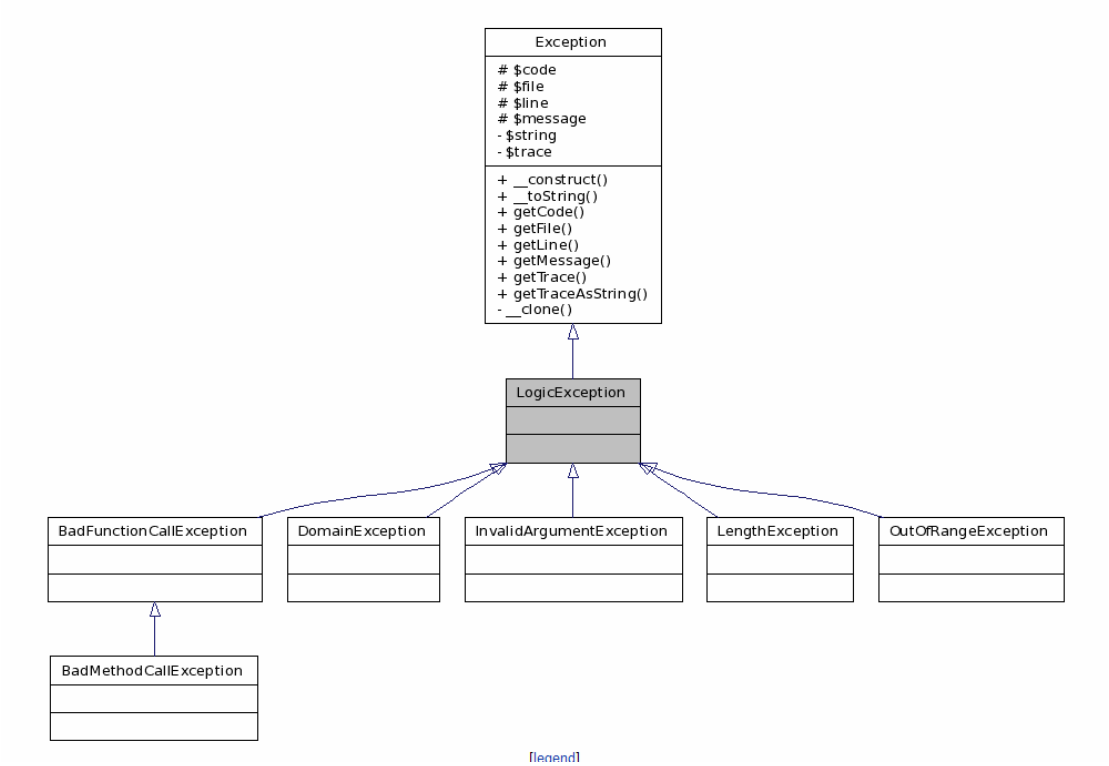
5.2.3 spl.php 中的其它异常类

在 SPL 中还定义了其它的异常类，以对应不同的异常类型。这些异常类都是 Exception 类的子类。

在 Exception 类有两个直接子类 LogicException 和 RuntimeException，分别表示逻辑异常和执行异常。



LogicException 又衍生出其它的逻辑异常子类。



class LogicException extends Exception
　　程序中的逻辑错误的异常类，它是 Exception 类的直接子类。

class BadFunctionCallException extends LogicException
　　当不合法的函数被调用产生的异常类。

class BadMethodCallException extends BadFunctionCallException
　　当不合法的方法被调用产生的异常类。

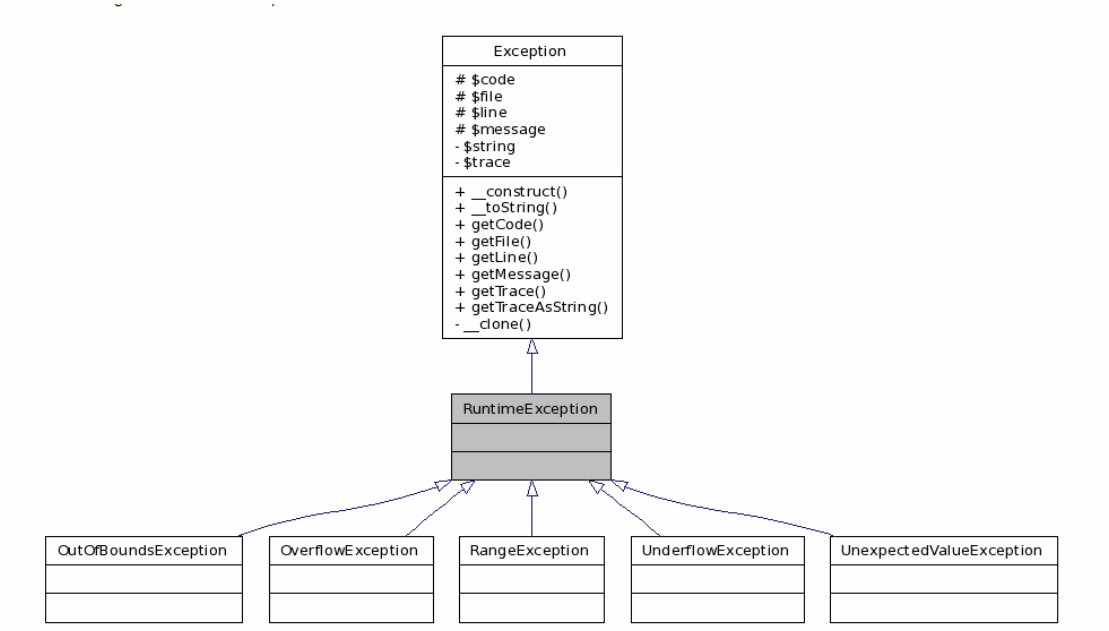
class DomainException extends LogicException
　　表示一个值不在有效范围内的异常。

class InvalidArgumentException extends LogicException
　　表示传递了无效的参数产生的异常。

class LengthException extends LogicException
　　表示一个参数超过了许可的长度的异常。

class OutOfRangeException extends LogicException
　　表示请求检索超越了数组等容器最大长度的异常。

RuntimeException 衍生出其它运行异常子类。



class RuntimeException extends Exception
只有在执行时才能发现的异常，是 Exception 的直接子类。

class OutOfBoundsException extends RuntimeException
表示请求检索超越了数组等容器最大长度的异常。

class OverflowException extends RuntimeException
表示算法/缓存溢出异常

class RangeException extends RuntimeException
运行期间的范围异常

class UnderflowException extends RuntimeException
运行期间的算法/缓存的向下溢出异常。

在 spl.php 中所有 Exception 子类的代码都仅仅是类的定义和简单的父类继承。
而方法内部没有任何扩展、重写。
如：LogicException 的定义。

```
329 /** @ingroup SPL
330  * @brief Exception that represents error in the program logic.
331  * @since PHP 5.1
332  *
333  * This kind of exceptions should directly leed to a fix in your code.
334  */
335 class LogicException extends Exception
336 {
337 }
```

又如：LengthException 的定义部分

```
378 /** @ingroup SPL
379  * @brief Exception thrown when a parameter exceeds the allowed length.
380  * @since PHP 5.1
381  *
382  * This can be used for strings length, array size, file size, number of
383  * elements read from an Iterator and so on.
384  */
385 class LengthException extends LogicException
386 {
387 }
```

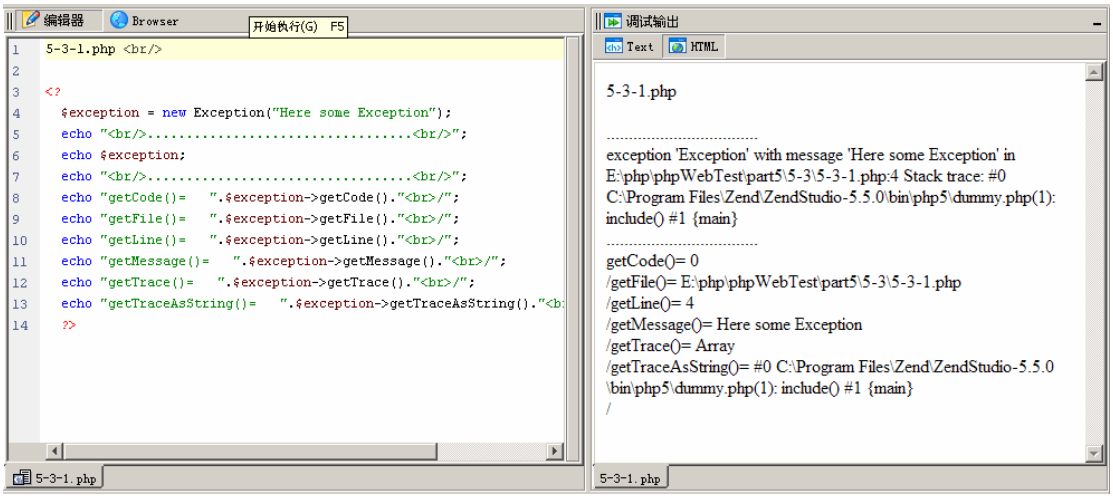
5.3 PHP5 异常捕获

5.3.1 异常实例

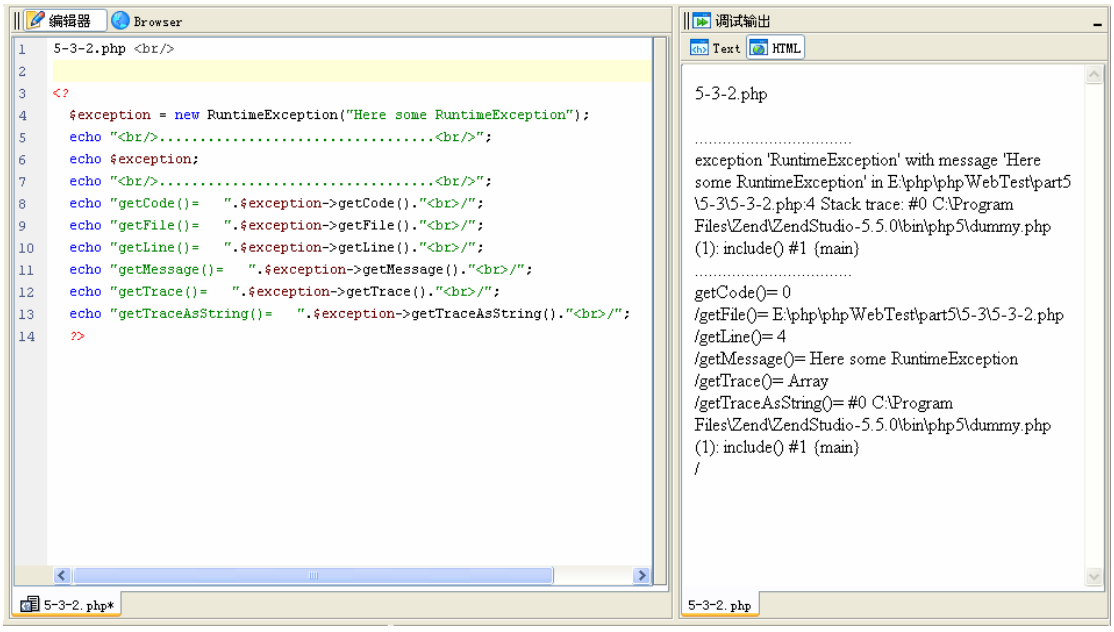
可以通过实例化 Exception 类或者它的子类来创建一个异常实例。

有了这个异常实例，就可以通过 Exception 中定义好的 getter 方法，获得相应的属性值。

例 5-3-1.php



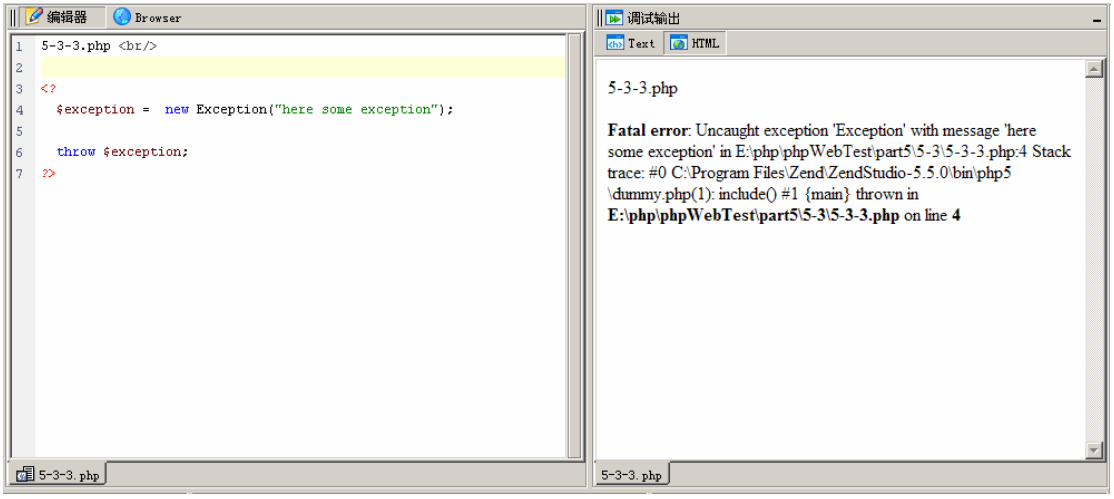
例 5-3-2.php



5.3.2 抛出异常

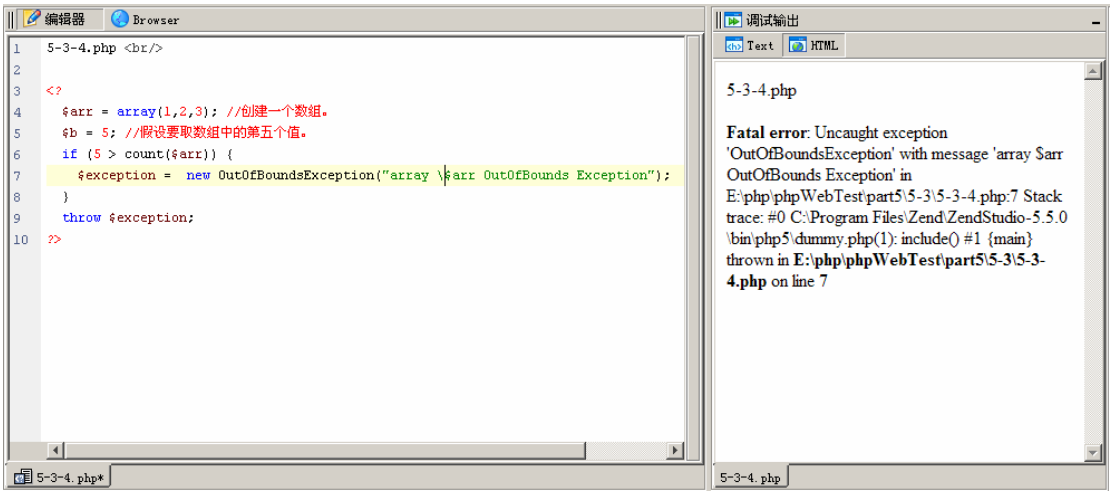
在 PHP5 中使用 throw 关键字，向外抛出一个异常实例。
如果这个异常如果未经处理，将会导致系统产生致命错误，而使代码终止。

例 5-3-3.php



根据需求，我们可以向系统抛出不同的异常。
在 php 中数组越界是不会产生知名错误的，而下面的代码抛出一个数组越界的异常，导致代码运行终止。

例 5-3-4.php



5.3.3 在代码中捕获异常

可以通过 PHP5 支持的 **try catch** 语句捕获并处理异常。

语法如下：

```
try{
    //可能引发异常的语句
}catch(异常类型  异常实例){
    //异常处理语句
}
```

在 try 块中，放置可能产生异常的代码片段。在 catch 块中放置对这个异常处理的代码。

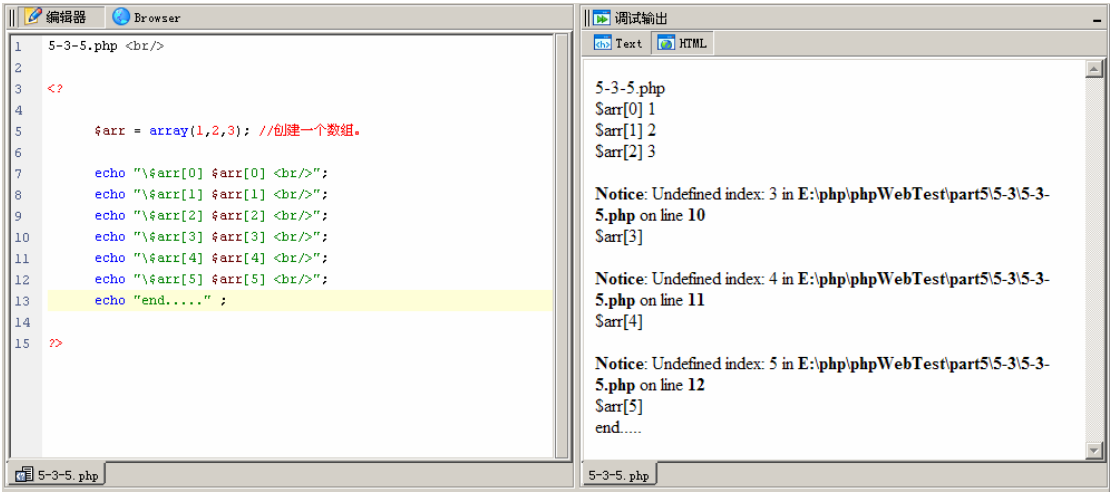
如：

```
try{
    throw new Exception("new Exception");   // 引发抛出异常
}catch(Exception  $ex){
    echo $ex; //打印这个异常对象
}
```

- 如果 try 块未产生任何异常，try 块将运行完毕，catch 块内容不会被执行。
- try 块如果抛出了异常，会立刻在 catch 中寻找可以捕获改异常的 catch 块，并运行相应的 catch 块代码，然后跳出 try catch 块继续运行。 而 try 块中抛出异常后面的代码将被跳过。
- 如果 try 块中的异常不能被 catch 块捕获，将抛向系统引发系统致命错误，代码终止运行。
- 在 catch 中，异常类型后面跟的是一个变量，这个变量将指向内存中被捕获的异常实例。

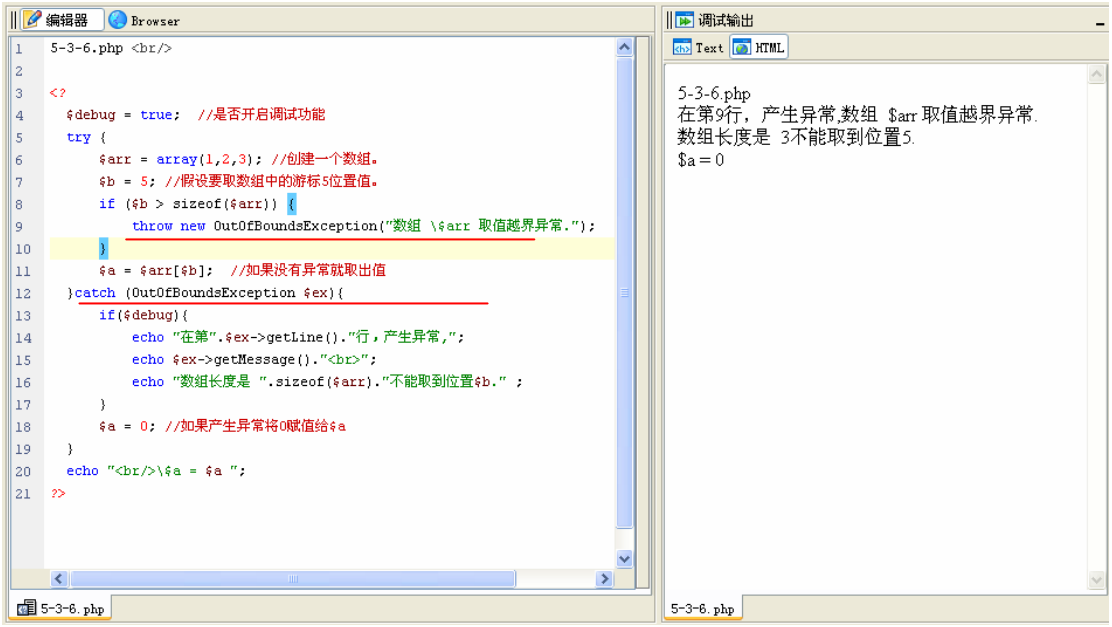
未使用异常处理时，我们从一个数组中取值，如果数组越界，PHP 只会报出一个错误 Notice，我们无法对这些错误做任何的处理。

例 5-3-5.php



在下面例子中，取值超过了数组边界，于是引发了自定义异常。在 catch 块做了处理。

例：5-3-6.php

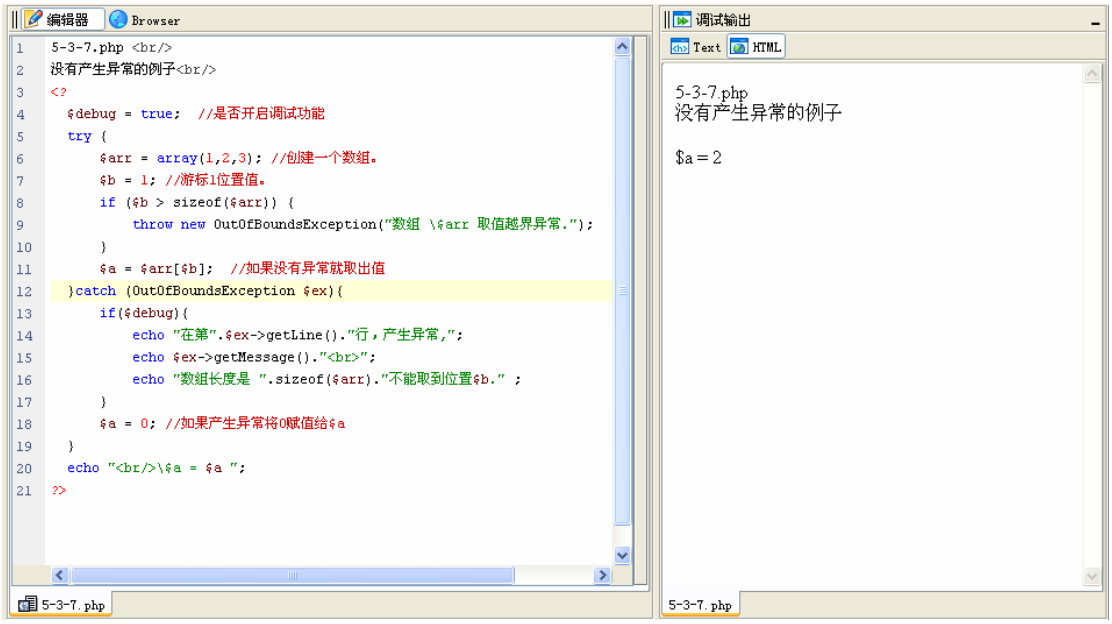


```
5-3-6.php <br/>

<?
  $debug = true;  //是否开启调试功能
  try {
      $arr = array(1,2,3); //创建一个数组。
      $b = 5; //假设要取数组中的游标5位置值。
      if ($b > sizeof($arr)) {
          throw new OutOfBoundsException("数组 \$arr 取值越界异常.");
      }
      $a = $arr[$b];  //如果没有异常就取出值
  }catch (OutOfBoundsException $ex){
      if($debug){
          echo "在第".$ex->getLine()."行，产生异常,";
          echo $ex->getMessage()."<br>";
          echo "数组长度是 ".sizeof($arr)."不能取到位置$b." ;
      }
      $a = 0; //如果产生异常将0赋值给$a
  }
  echo "<br/>\$a = $a ";
?>
```

```
5-3-6.php
在第9行，产生异常,数组 $arr 取值越界异常.
数组长度是 3不能取到位置5.
$a = 0
```

取值没有引发异常的代码。

例 5-3-7.php



```
5-3-7.php <br/>
没有产生异常的例子<br/>
<?
  $debug = true;  //是否开启调试功能
  try {
      $arr = array(1,2,3); //创建一个数组。
      $b = 1; //游标1位置值。
      if ($b > sizeof($arr)) {
          throw new OutOfBoundsException("数组 \$arr 取值越界异常.");
      }
      $a = $arr[$b];  //如果没有异常就取出值
  }catch (OutOfBoundsException $ex){
      if($debug){
          echo "在第".$ex->getLine()."行，产生异常,";
          echo $ex->getMessage()."<br>";
          echo "数组长度是 ".sizeof($arr)."不能取到位置$b." ;
      }
      $a = 0; //如果产生异常将0赋值给$a
  }
  echo "<br/>\$a = $a ";
?>
```

```
5-3-7.php
没有产生异常的例子

$a = 2
```

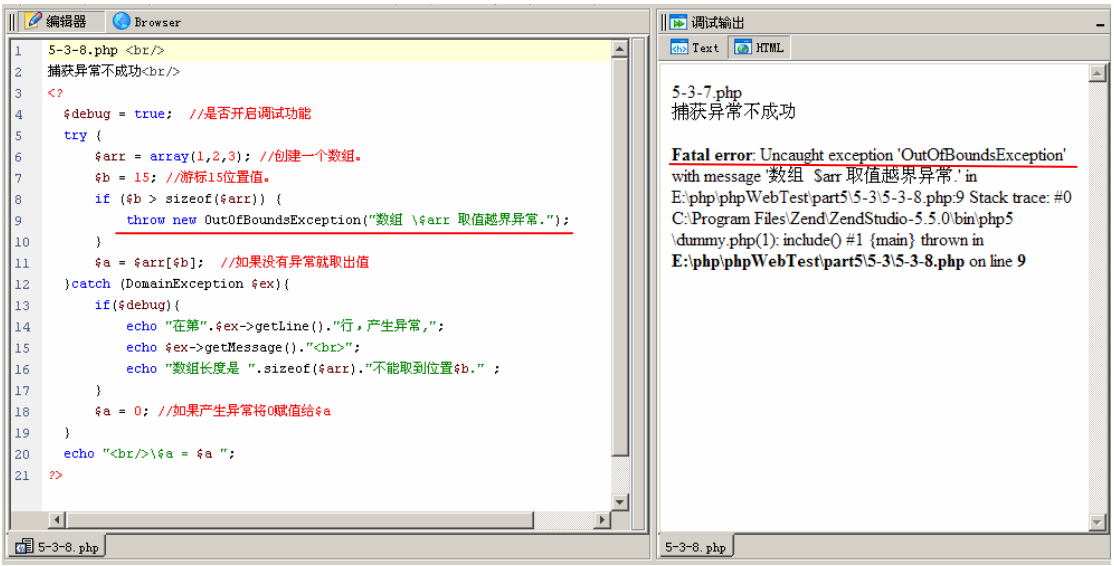
5.3.4 在代码中捕获异常(2)

大家注意到 catch(Exception $ex) 中 Exception 这个类名，下面解释它的具体意义。

- 在 catch 块中能捕获在 catch()块中声明的捕获的异常和其子类类型实例。

下面的例子，抛出一个 OutOfBoundsException 的异常，而 catch 语句捕获 DomainException 异常。这个异常不会被 catch 语句捕获，而直接抛向了系统，引发了一个致命错误，程序被终止了。

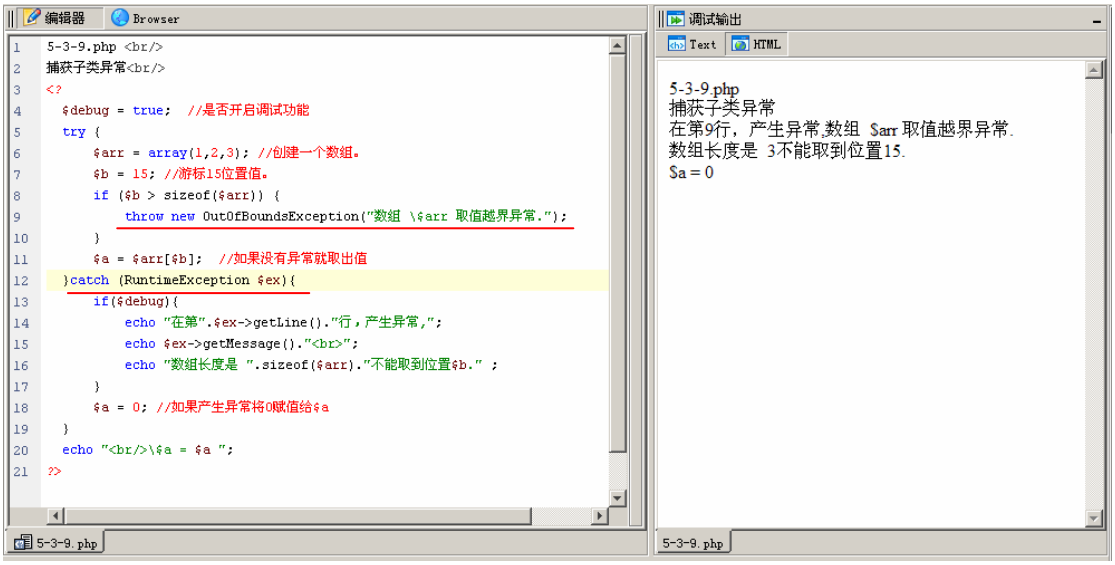
例 5-3-8.php



- 在 catch 块中能捕获 catch()块里声明的异常的子类异常。

下面的例子抛出了 OutOfBoundsException 异常，由它的父类 RuntimeException 捕获。

例 5-3-9.php

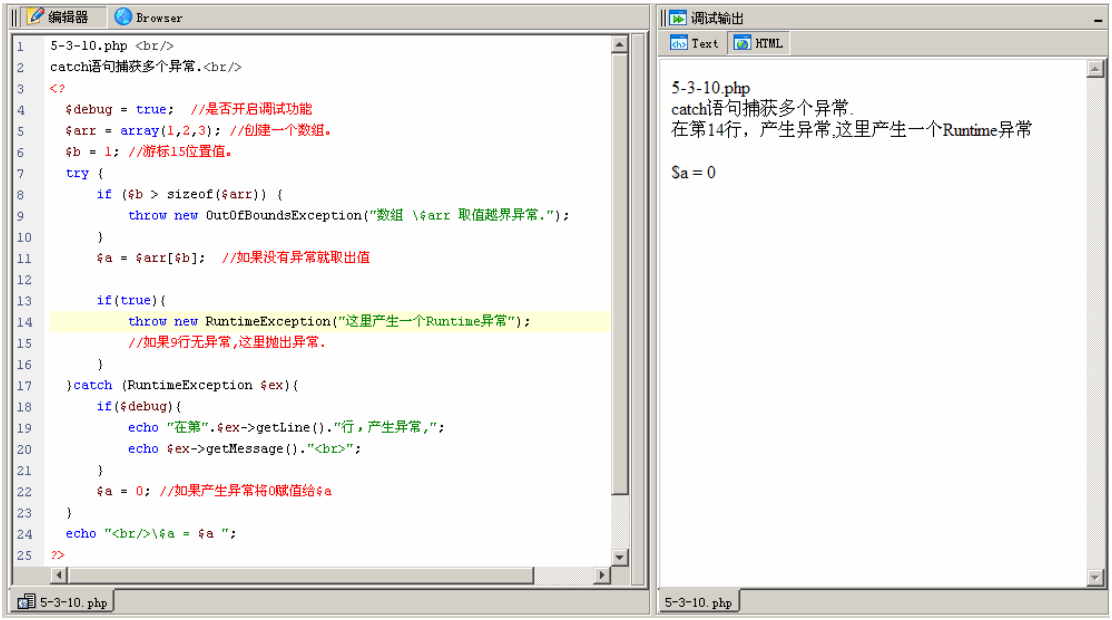


5.3.5 一个 catch 块处理多种异常

- 在 catch 块中能捕获在 catch()块中声明的捕获的异常和其子类类型实例。

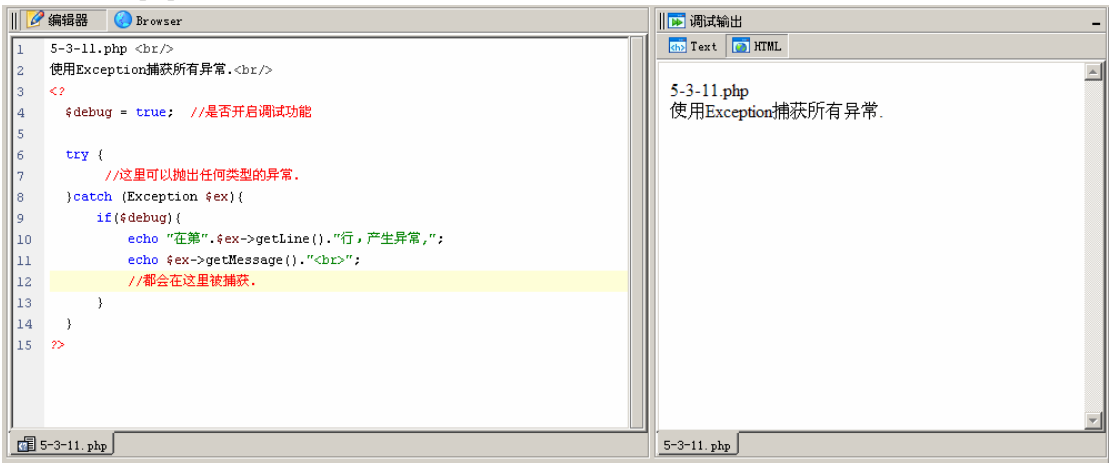
在下面的代码中，catch 块即可以捕获第 9 行的异常，也可以捕获第 14 行的异常。实现了一个异常处理块捕获多种异常。

例 5-3-10.php



Exception 是所有异常类的父类，catch(Exception $ex)可以捕获 try 块中的任何异常。

例：5-3-11.php



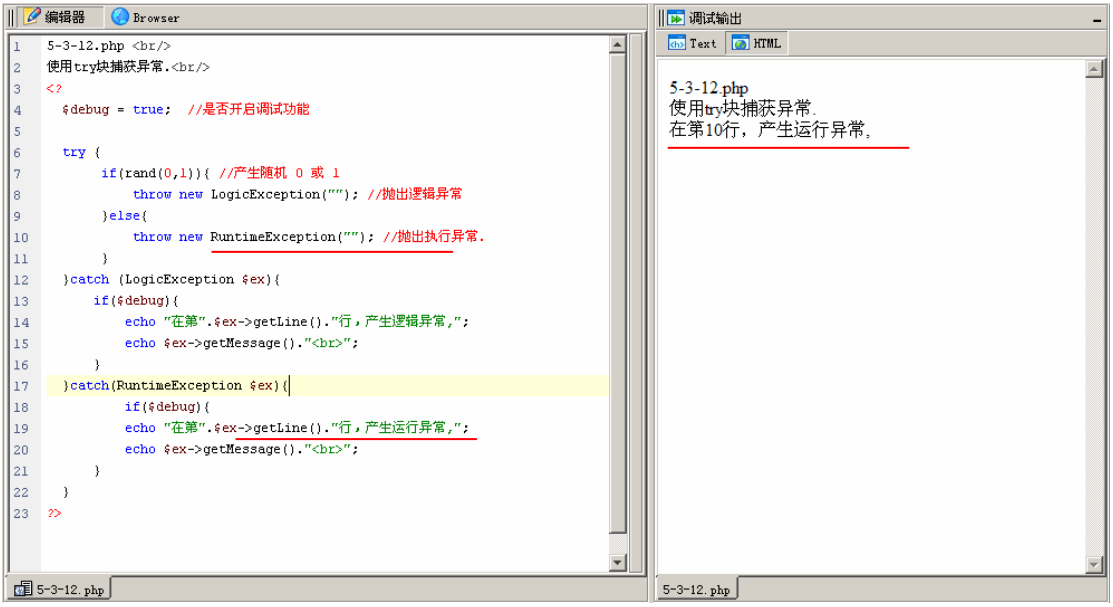
5.3.6 多个 catch 块处理异常

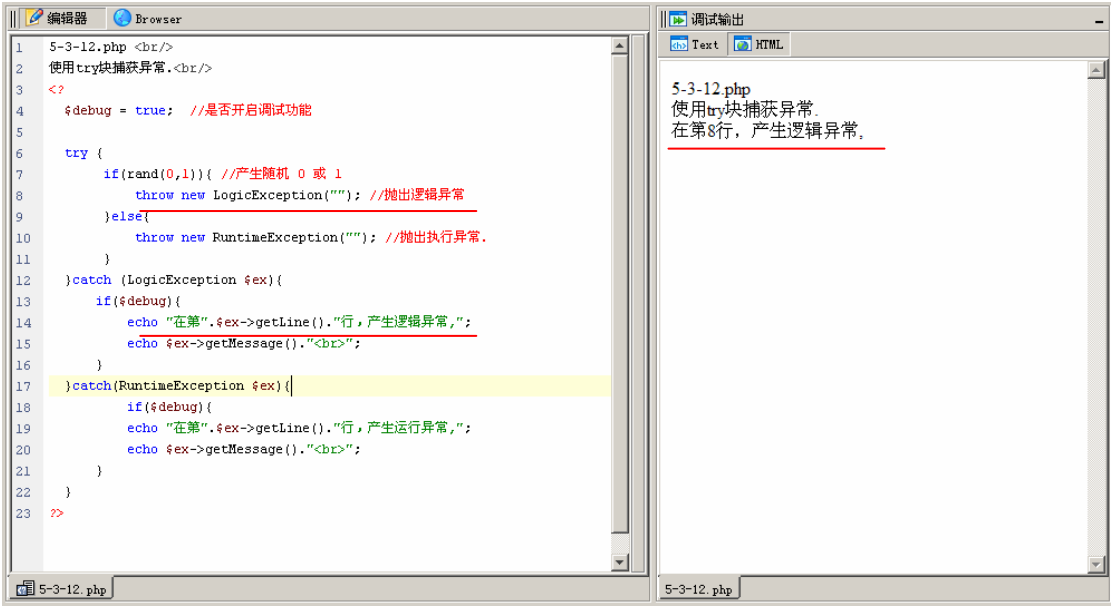
一个 try 块可以跟随多个 catch 块，每个 catch 块捕获不同的异常。

下面例子的第 7 行，使用 rand 函数产生了一个 0 或 1 的随机数，反复运行这个代码会随机抛出逻辑异常或执行异常。

在 try 块后，有两个 catch 块，分别捕获并处理对应的异常，注意查看不同。

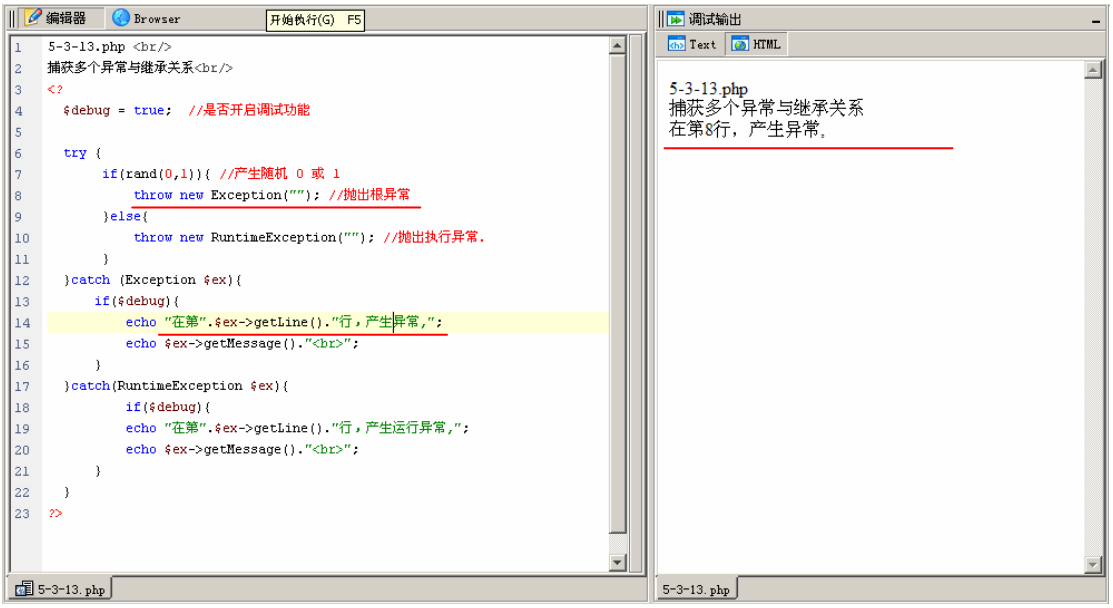
例 5-3-12.php

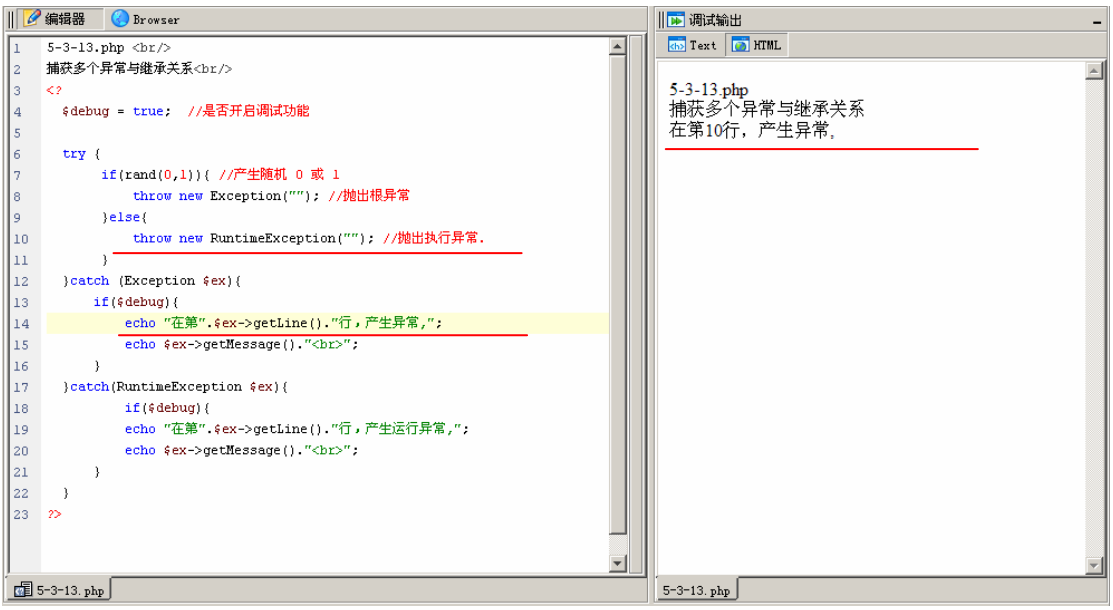




当有异常被抛出，系统从 catch 片段中寻找能处理这个异常的 catch 块，
前面说过一个父类的 catch 块可以捕获子类的异常实例。
在 catch 片段中，如果父类的 catch 块在前面，将屏蔽掉后面对应的子类 catch 块的功能。

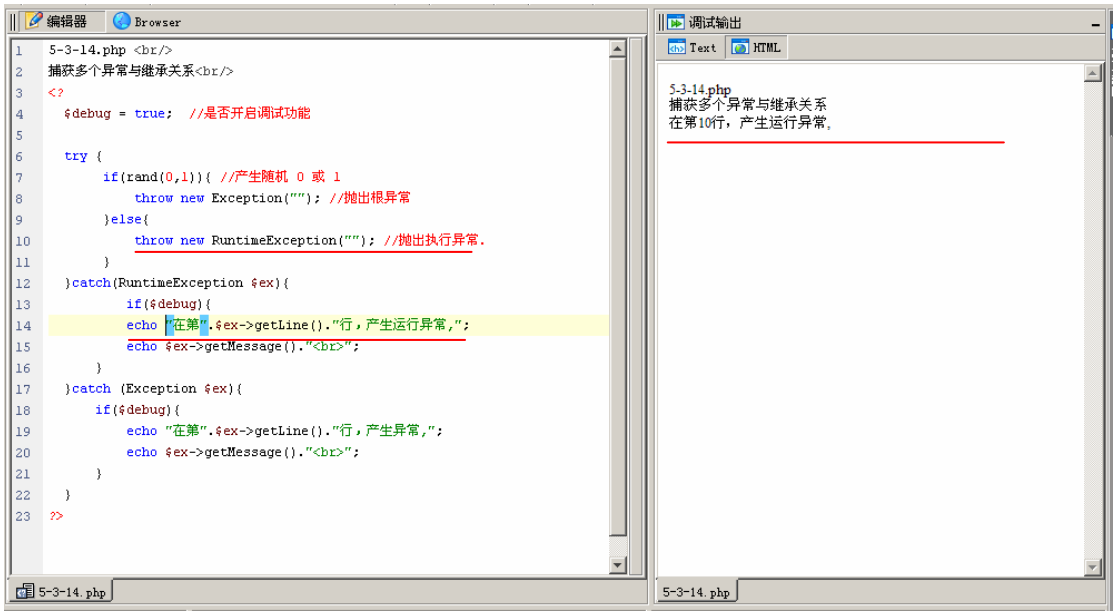
下面例子中引发的两种异常，都被第一个 Exception 类型的 catch 块捕获了。
例 5-3-13.php



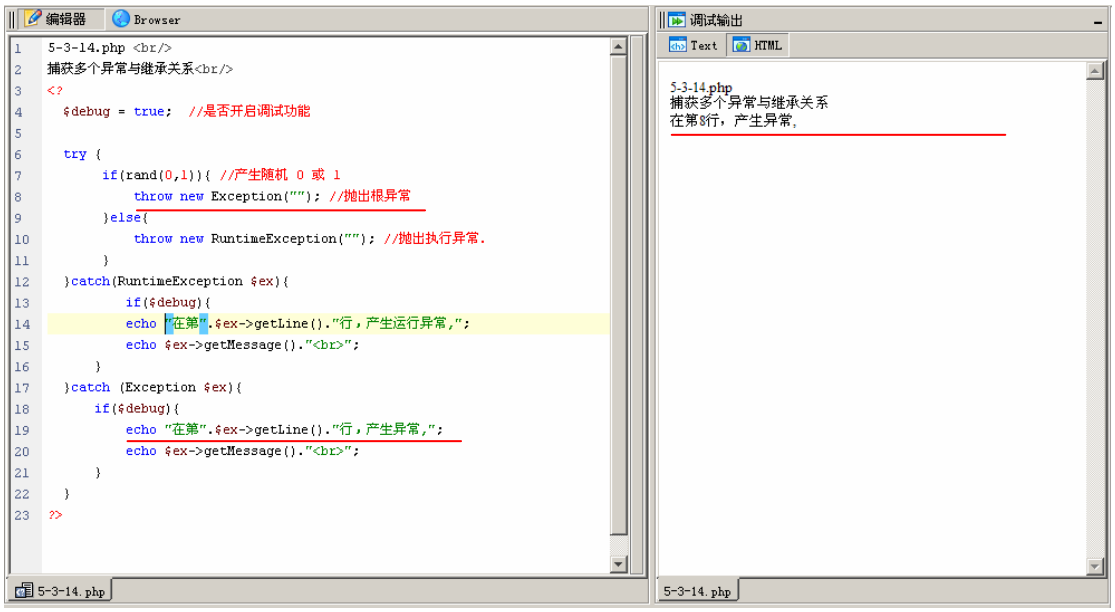


把两个 catch 块的顺序调换，将子类的异常捕获放在前面，父类的异常捕获放在后面。就会看到两个 catch 块能各司其职了。

例：5-3-14.php




```
5-3-14.php <br/>
捕获多个异常与继承关系<br/>
<?
  $debug = true;  //是否开启调试功能

  try {
       if(rand(0,1)){ //产生随机 0 或 1
           throw new Exception(""); //抛出根异常
       }else{
           throw new RuntimeException(""); //抛出执行异常.
       }
  }catch(RuntimeException $ex){
         if($debug){
         echo "在第".$ex->getLine()."行，产生运行异常，";
         echo $ex->getMessage()."<br>";
     }
  }catch (Exception $ex){
      if($debug){
          echo "在第".$ex->getLine()."行，产生异常，";
          echo $ex->getMessage()."<br>";
      }
  }
?>
```

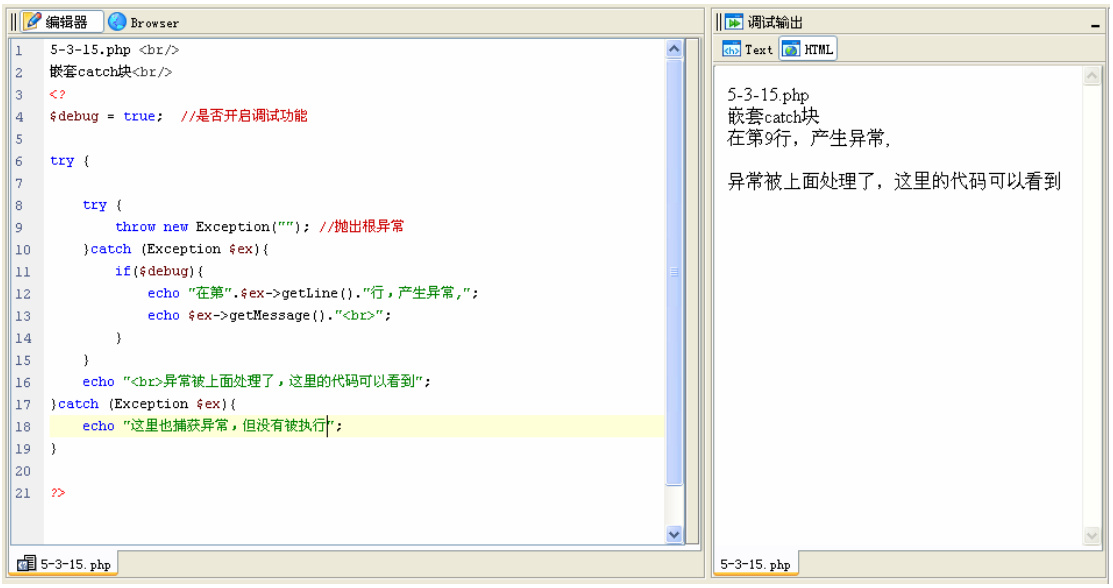



```
5-3-14.php <br/>
捕获多个异常与继承关系<br/>
<?
  $debug = true;  //是否开启调试功能

  try {
       if(rand(0,1)){ //产生随机 0 或 1
           throw new Exception(""); //抛出根异常
       }else{
           throw new RuntimeException(""); //抛出执行异常.
       }
  }catch(RuntimeException $ex){
         if($debug){
         echo "在第".$ex->getLine()."行，产生运行异常，";
         echo $ex->getMessage()."<br>";
     }
  }catch (Exception $ex){
      if($debug){
          echo "在第".$ex->getLine()."行，产生异常，";
          echo $ex->getMessage()."<br>";
      }
  }
?>
```

5.3.7 异常处理块嵌套

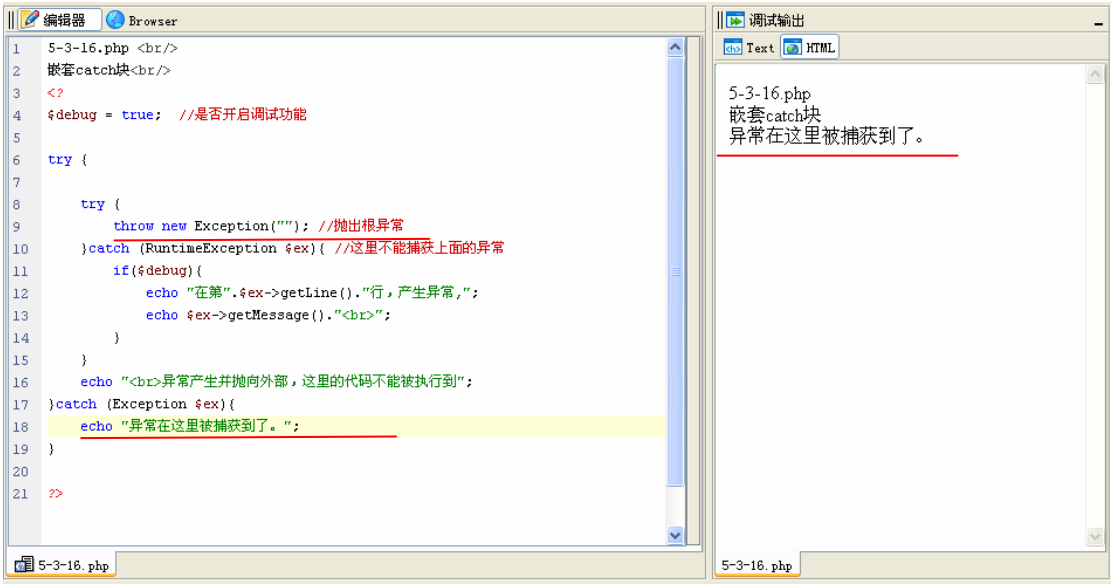
异常处理块只能处理自己 catch 块中的异常，已经处理过的异常将不会向外抛出。

例 5-3-15.php



如果内部块不能捕获异常，这个异常将继续向外抛出。直到能够被捕获为止。

例 5-3-16.php

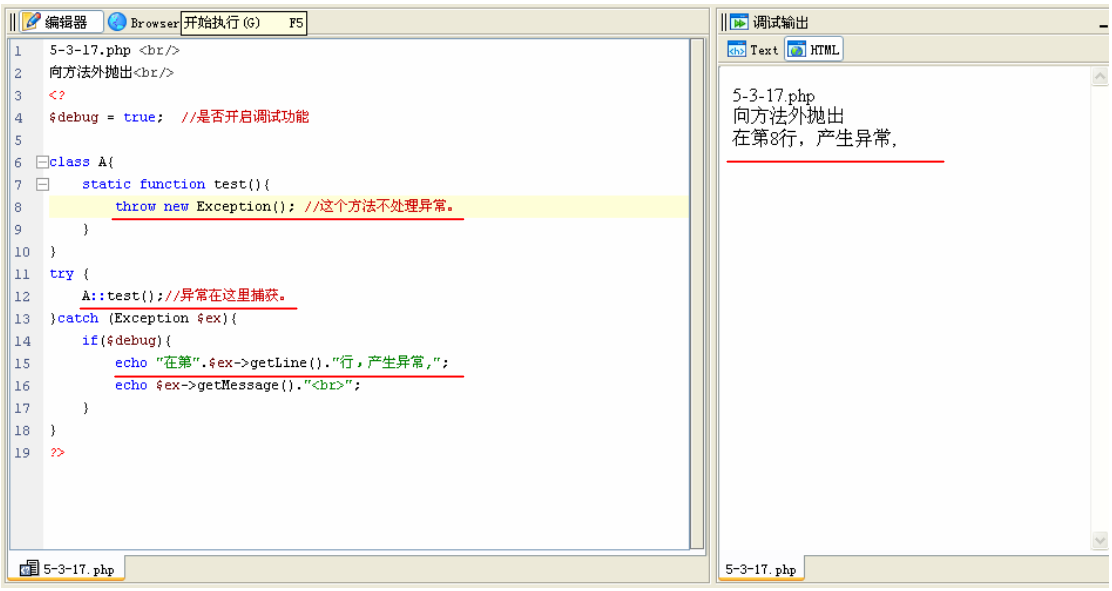


5.3.8 异常向外抛出

代码中一旦 throw 一个异常实例，系统会寻找能够处理这个异常的 try catch 块，如果当前方法不能够处理这个异常，就会向外抛出。抛向调用这个方法的代码，一直向外抛出，如果抛到最外层都无法处理这个异常，会引发致命错误，代码终止。

我们可以在方法引用的任何一个环节，根据业务需求决定捕获异常的位置。

例 5-3-17.php

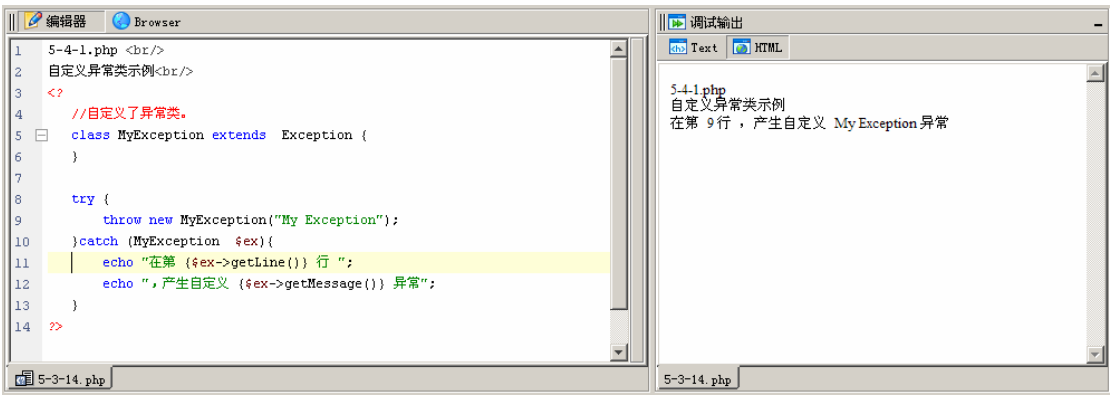


5.4 PHP5 自定义异常

5.4.1 自定义异常

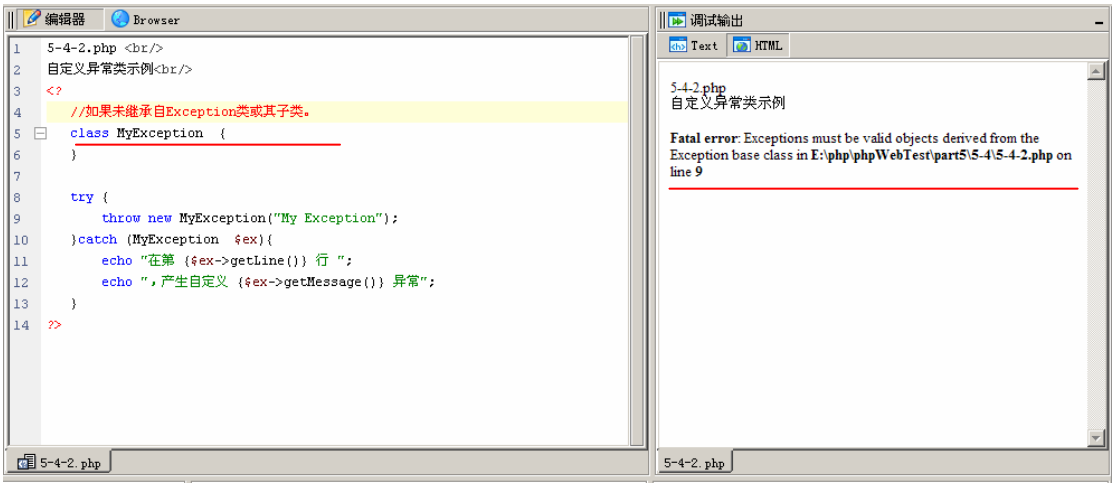
在 PHP5 中可以自定义异常，自定义的异常必须继承自 Exception 类或者它的子类。

例 5-4-1.php



如果自定义的类未继承自 Exception 或者其子类，就会出现不能运行的错误。

例 5-4-2.php



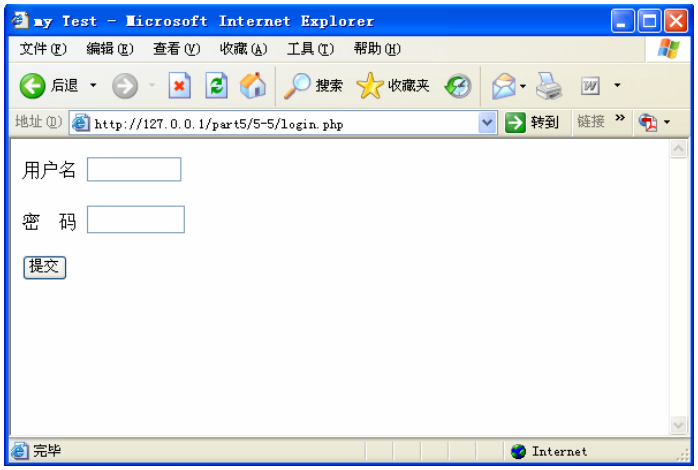
5.5 异常处理实例

5.5.1 验证实例

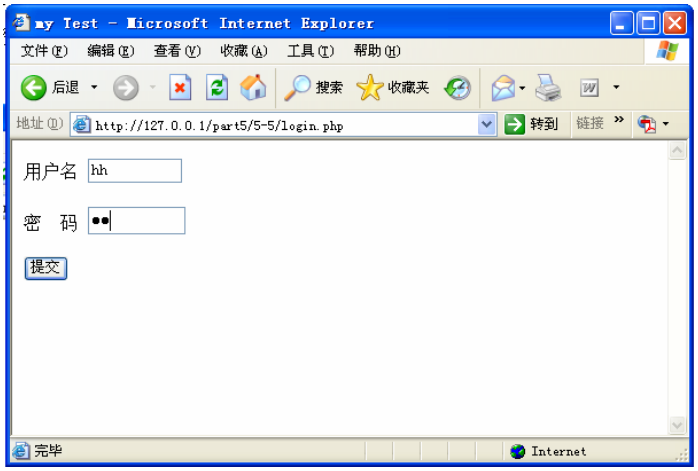
本例将写用户名和密码长度验证并对用户名和密码设定为只能是字符，配合正则表达式完成。

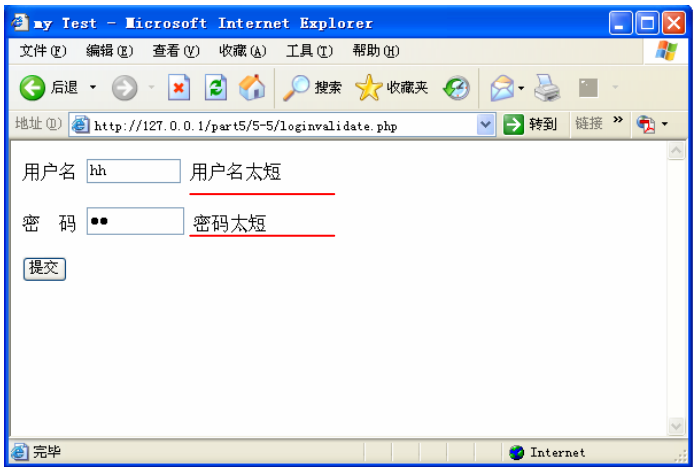
设定用户名和密码长度必须在 3-8 个之间，并且只能是字符与数字

如何与数据库链接，进行数据库验证，就不在本例表达。

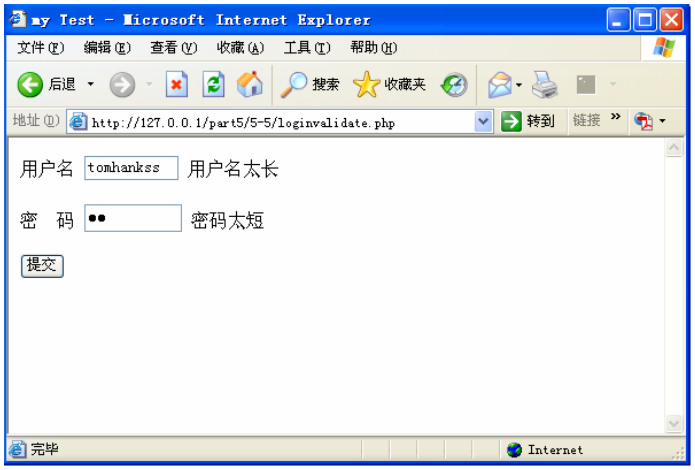


如果用户名和密码长度不够，提交后会提示。

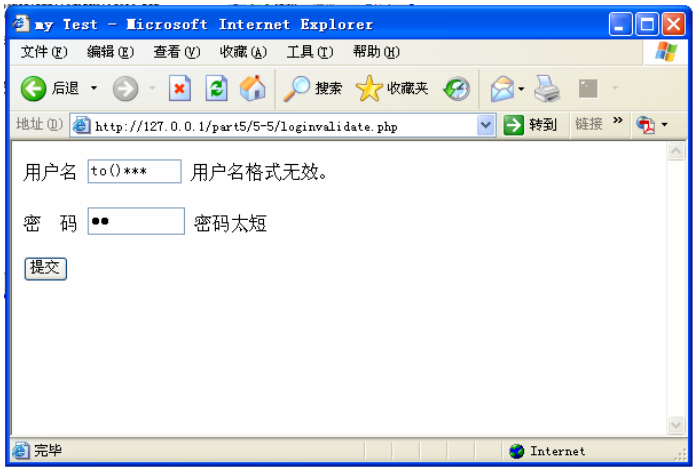




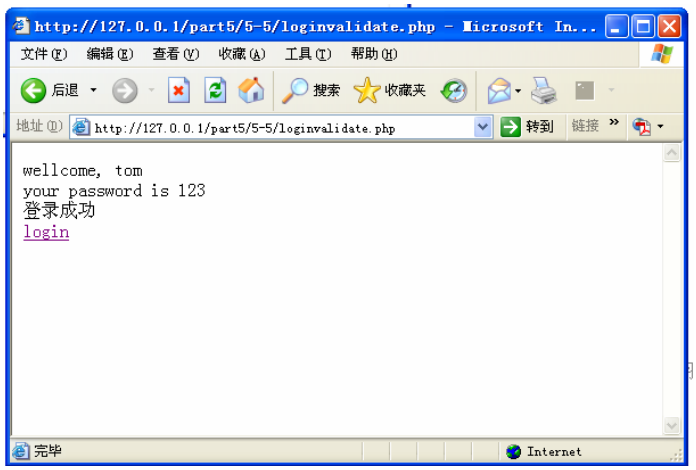
如果过长会提示。



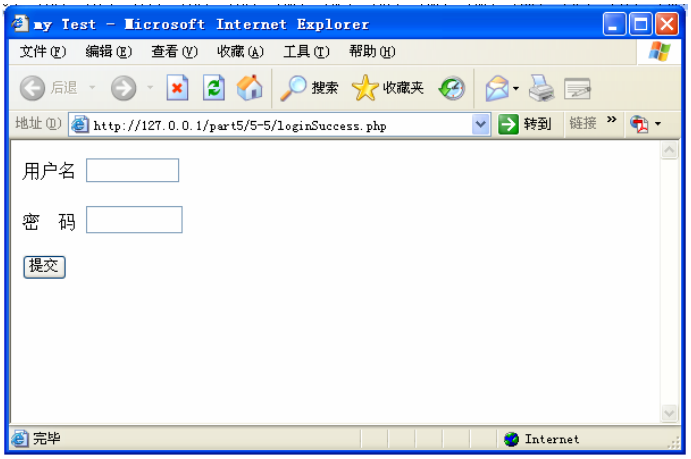
如果输入字符有 （*等符号不合法。



如果验证成功进入登录成功页面
用户输入的用户名和密码可以显示在用户界面。



如果用户直接输入 登录成功 loginSuccess.php 页面，会转到登录页面。



5.5.2 验证实例代码

User.php 实现功能：

- 定义 User 接口。
- 实现存放用户信息的类。
- 通过工厂模式返回用户类实例。

编辑器 Browser

```php
<?
//User.php
//这是一个简单工厂模式的实现,用来获得一个与表单对应的User实例.
interface User{ //定义的接口
    function setUsername($username);
    function getUsername();
    function setPassword($password);
    function getPassword();
}
abstract  class LoginUser implements User { //抽象类
    private $username;
    private $password;
    function setUsername($username){
        $this->username =  $username;
    }
    function getUsername(){
        return $this->username;
    }
    function setPassword($password){
        $this->password =  $password;
    }
    function getPassword(){
        return $this->password;
    }
    //返回子类的实例.
    static function getNormalUser($username,$password){
        $user = new NormalUser();
        $user->setUsername($username);
        $user->setPassword($password);
        return $user;
    }
}

class NormalUser extends LoginUser{
}
?>
```

定义一些参数的类，Properties.php
在这个类中集中定义了一些参数，方便进行后期一些与程序逻辑关系不大的维护。

编辑器 Browser

```php
<?
// Properties.php
//定义了一些参数,集中修改更方便,有些不会在程序中改变的变量可以设置为const类型.
class Properties{
    const User = "User";
    static $title = "my Test";

    static $UsernameLengthMin = 3;
    static $UsernameLengthMax = 8;
    static $username = "username";
    static $validate_Username = "erro_Username";
    static $validate_UsernameInfo = "用户名格式无效。";
    static $validate_UsernameLengthMin = "用户名太短";
    static $validate_UsernameLengthMax = "用户名太长";

    static $PasswordLengthMin = 3;
    static $PasswordLengthMax = 8;
    static $password="password";
    static $validate_Password = "erro_Password";
    static $validate_PasswordInfo="用户格式无效";
    static $validate_PasswordLengthMin="密码太短";
    static $validate_PasswordLengthMax="密码太长";

    static $loginFailes =  "login.php";
    static $loginSuccess = "loginSuccess.php";
}
?>
```

自定义类， MyException.php
自定义了用户名和密码两个独立的异常类，以及其它子类。

编辑器 Browser

```php
<?
//MyException.php
//这里定义了相关异常.为以后的扩展准备好了丰富的异常分类.
class UsernameException extends  Exception {}
class UsernameInvalidateException extends UsernameException {}
class UsernameLengthMinException extends UsernameException {}
class UsernameLengthMaxException extends UsernameException {}

class PasswordException extends Exception {}
class PasswordInvalidateException extends PasswordException {}
class PasswordLengthMinException extends PasswordException  {}
class PasswordLengthMaxException extends PasswordException  {}
?>
```

验证用户名和密码长度合法性的类 Validate.php
在 validateName 方法和 validatePassword 两个方法中，分别对长度做了判断和验证。并通过正则表达式，对用户输入的合法性做了验证。
虽然可以通过正则表达式一次验证用户字符合法性和长度， 本例没有这样写是为了向方法外抛出长度不同的异常，以及显示给用户的信息。

编辑器 Browser

```php
<?
//Validate.php 验证类
include_once("User.php");
include_once("Properties.php");
include_once("MyException.php");

class Validate{

    public static  function validateUser(User $user){

        try{
            self::validateName($user->getUsername()); //用户名验证
        }catch(UsernameException $ex){
            //错误信息放入session.
            $_SESSION[Properties::$validate_Username] = $ex->getMessage();
        }

        try{
            self::validatePassword($user->getPassword());
        }catch(PasswordException  $ex){
            //错误信息放入session.
            $_SESSION[Properties::$validate_Password] =$ex->getMessage();
        }
        if($ex){
            //将用户名和密码放入session
            $_SESSION[Properties::$username] = $user->getUsername();
            $_SESSION[Properties::$password] = $user->getPassword();
            throw $ex; //向外抛出异常实例.
        }//如果没有异常抛出,就天下太平. 没有问题.


    }

    //验证用户名长度和是否符合正则表达式.
    //没有完全使用正则表达式,而先使用了长度判断,是为了提供更确切的错误信息.
    private static function  validateName($username){

        if(strlen($username) < Properties::$UsernameLengthMin){
            throw new UsernameLengthMinException(Properties::$validate_UsernameLengthMin);
        }elseif (strlen($username) > Properties::$UsernameLengthMax){
            throw new UsernameLengthMaxException(Properties::$validate_UsernameLengthMax);
        }elseif(! ereg("^[a-zA-Z0-9]*$",$username)){
            throw new UsernameInvalidateException(Properties::$validate_UsernameInfo);
        }
    }
    //验证密码.
    private static function  validatePassword($password){
        if(strlen($password) < Properties::$PasswordLengthMin){
            throw new PasswordLengthMinException(Properties::$validate_PasswordLengthMin);
        }elseif (strlen($password) > Properties::$UsernameLengthMax){
            throw new PasswordLengthMaxException(Properties::$validate_PasswordLengthMax);
        }elseif(! ereg("^[a-zA-Z0-9]*$",$password)){
            throw new PasswordInvalidateException(Properties::$validate_PasswordInfo);
        }
    }
}
?>
```

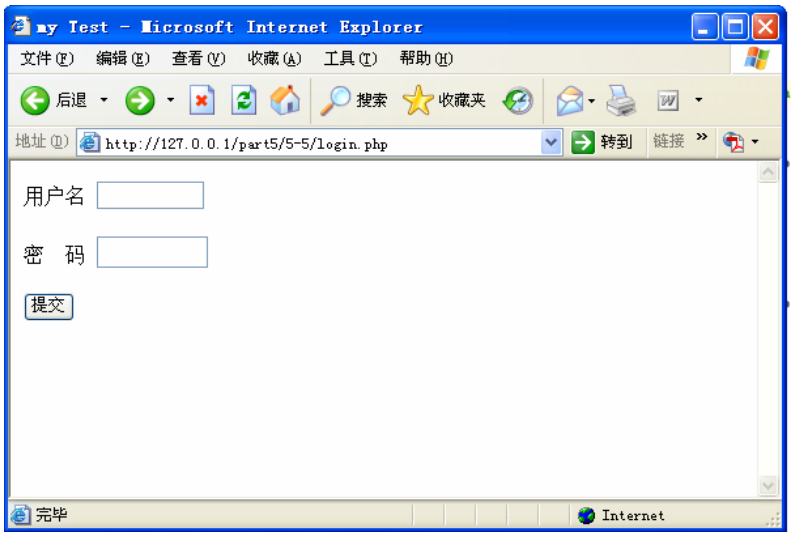
登录页面 login.php。

在第 12 行和第 18 行，分别取出 session 中的用户名密码，以及错误信息。

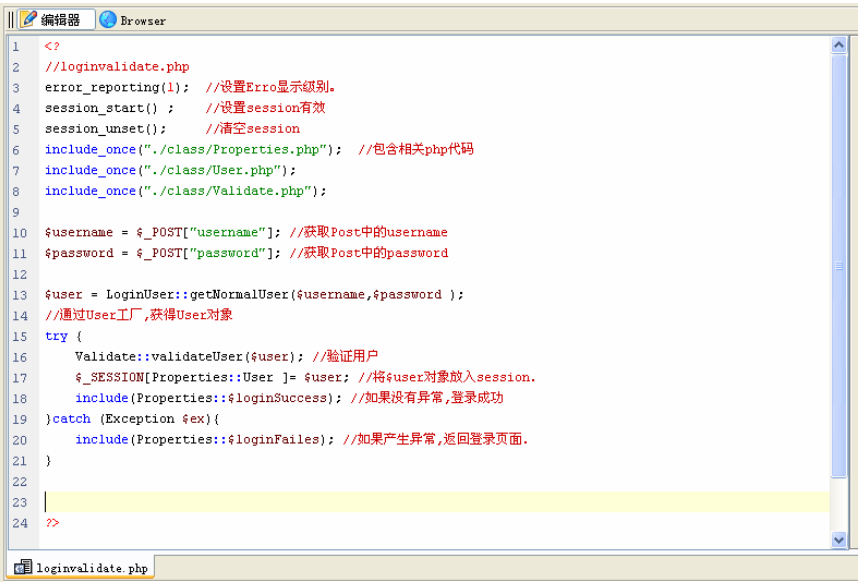
编辑器 Browser

```
1  <? require_once("./class/Properties.php"); ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml">
4  <head>
5  <meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
6  <title><?=Properties::$title?></title>
7  </head>
8  <body>
9  <form id="form1" name="form1" method="post" action="loginvalidate.php">
10   <label>用户名
11   <input name="username" type="text"
12 id="username" size="10" maxlength="10" value="<?=$_SESSION[Properties::$username]?>"/>
13   <?=$_SESSION[Properties::$validate_Username] ?>
14   </label>
15   <p>
16     <label>密&amp;nbsp;&amp;nbsp;码
17     <input name="password" type="password"
18 id="password" size="10" maxlength="10" value="<?=$_SESSION[Properties::$password] ?>" />
19      <?=$_SESSION[Properties::$validate_Password] ?>
20     </label>
21     <br />
22     <br />
23     <label>
24     <input type="submit" name="Submit" value="提交" />
25     </label>
26   </p>
27 </form>
28 </body>
```

界面如下

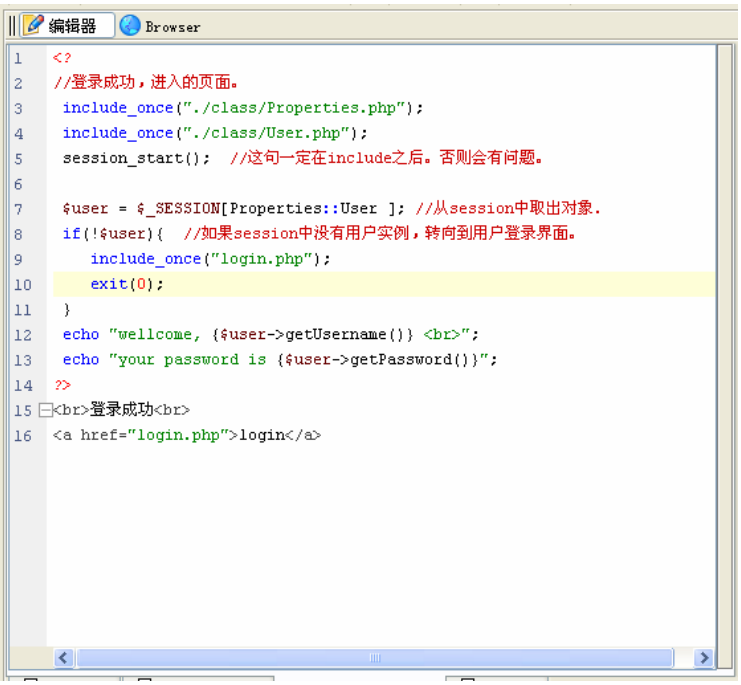


登录会提交给 loginvalidate.php
这段代码相当于 MVC 中的 controller 控制器。



```php
<?
//loginvalidate.php
error_reporting(1);  //设置Erro显示级别。
session_start() ;    //设置session有效
session_unset();     //清空session
include_once("./class/Properties.php");  //包含相关php代码
include_once("./class/User.php");
include_once("./class/Validate.php");

$username = $_POST["username"]; //获取Post中的username
$password = $_POST["password"]; //获取Post中的password

$user = LoginUser::getNormalUser($username,$password );
//通过User工厂,获得User对象
try {
    Validate::validateUser($user); //验证用户
    $_SESSION[Properties::User ]= $user; //将$user对象放入session.
    include(Properties::$loginSuccess); //如果没有异常,登录成功
}catch (Exception $ex){
    include(Properties::$loginFailes); //如果产生异常,返回登录页面.
}


?>
```

登录成功页面，loginSuccess.php
显示用户登录信息，判断 session。



```php
<?
//登录成功，进入的页面。
 include_once("./class/Properties.php");
 include_once("./class/User.php");
 session_start();  //这句一定在include之后。否则会有问题。

 $user = $_SESSION[Properties::User ]; //从session中取出对象.
 if(!$user){  //如果session中没有用户实例，转向到用户登录界面。
    include_once("login.php");
    exit(0);
 }
 echo "wellcome, {$user->getUsername()} <br>";
 echo "your password is {$user->getPassword()}";
?>
<br>登录成功<br>
<a href="login.php">login</a>
```

5.6 小结