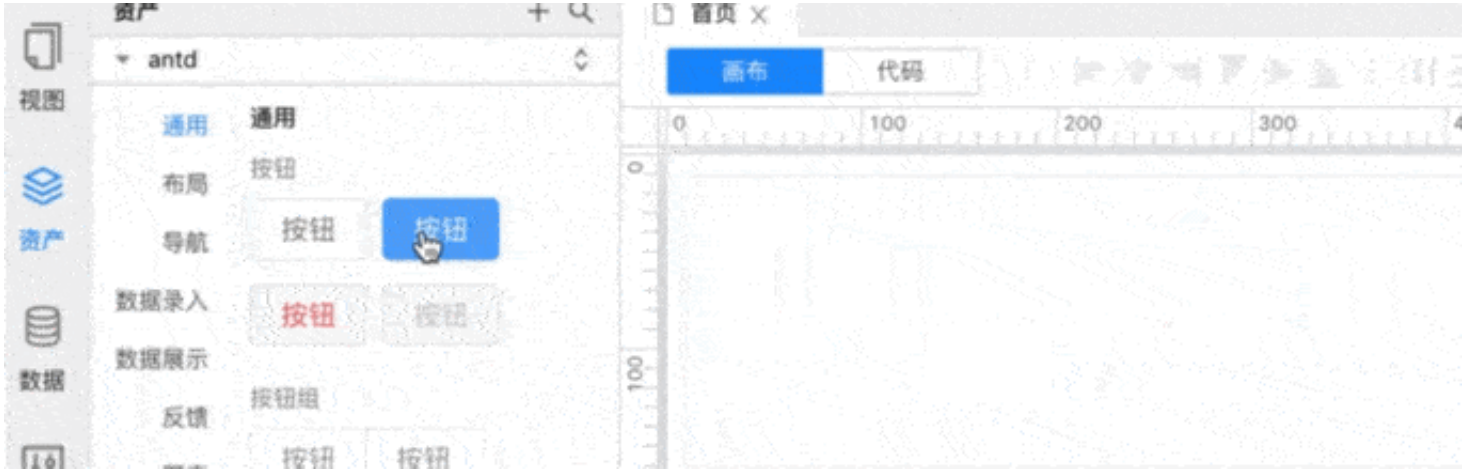
canvas 实现自动对齐布局功能（auto layout）

最近因为要做一个工具，查资料的过程中发现了这个效果，感觉还是挺有意思的，记录一下实践过程。



图片地址

首先看一下 Apple 开发者文档 有关于这个效果的实现标准和介绍，虽然我们是做网页和 Apple 也不着边，但是既然 Apple 有这个标准，也可以大致看一下有什么关键点。

其中我摘抄了一些关键点，基本是按照 web 开发的标准，放弃掉一些 web 可能用不上的东西，斜体、黑体为我的注释信息和需要注意的信息。

介绍

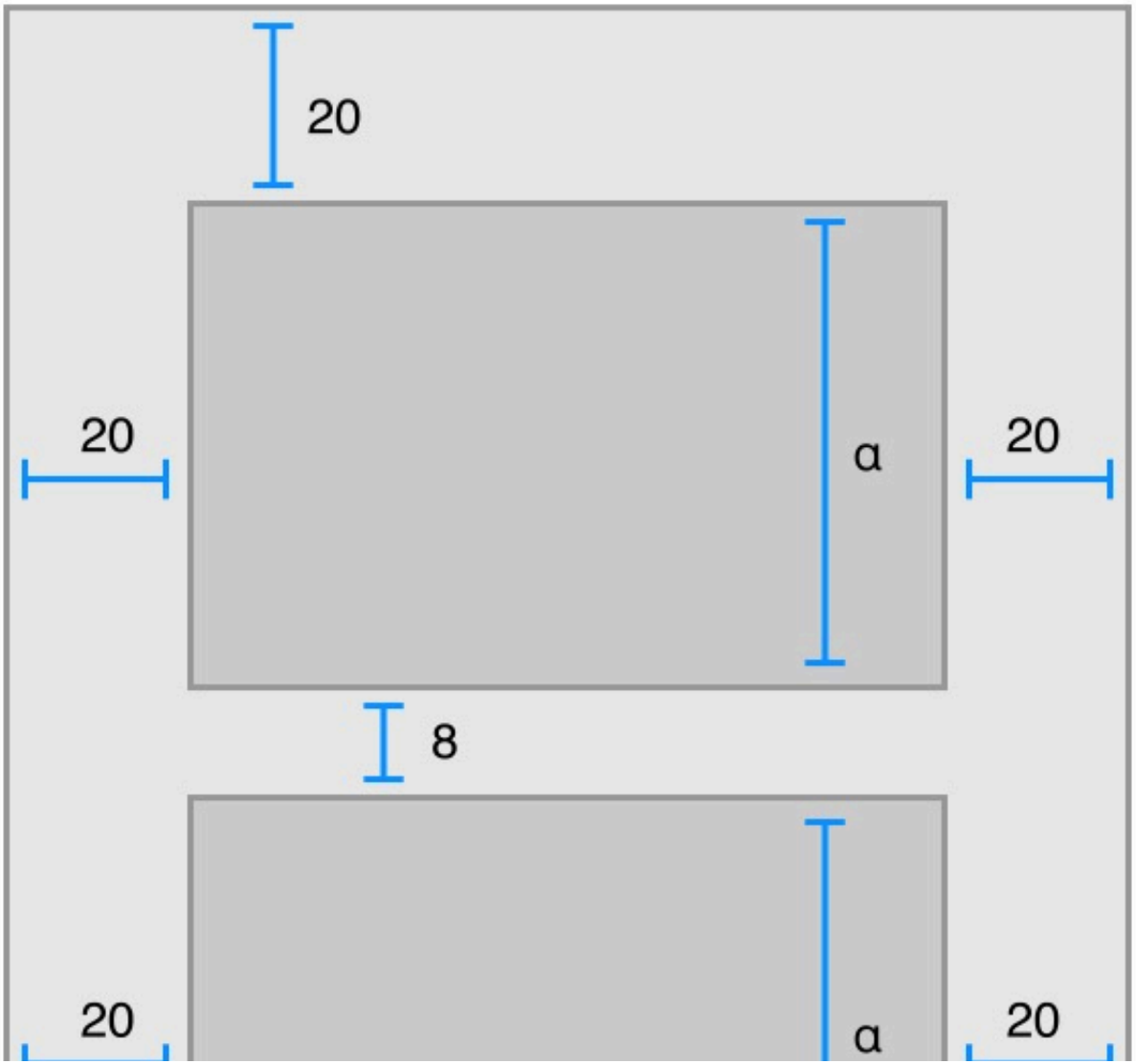
`Auto Layout` （* 自动布局 ）*会根据放置在视图上的约束条件，动态计算视图层次结构中所有视图的大小和位置。例如，您可以限制按钮，使其与图像视图水平居中，* 并使按钮的顶部边缘始终保持在图像底部下方 8 点 **。如果图像视图的大小或位置发生变化，按钮的位置就会自动调整以匹配。

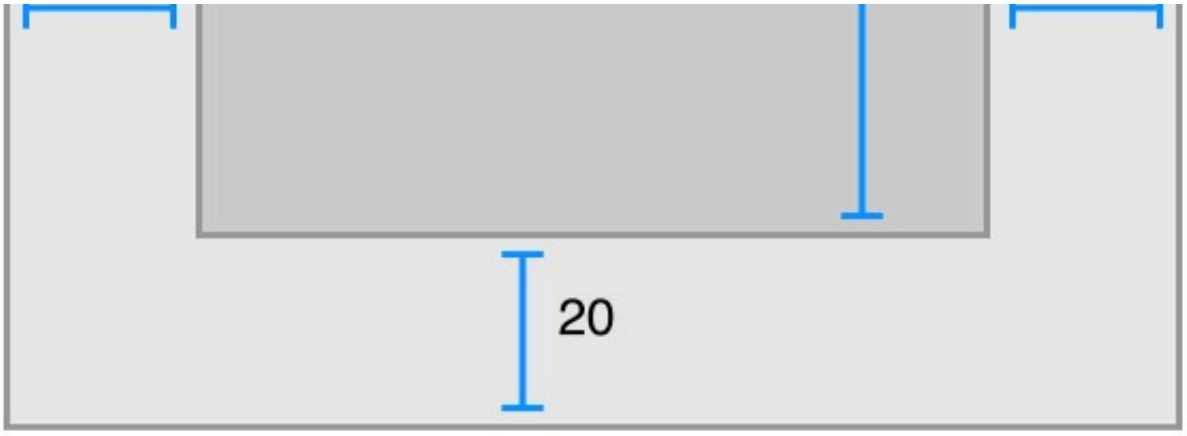
外部变化

当视图（* 窗口或者外部包裹 DOM*）的大小或形状发生变化时，会发生外部变化。每次更改时，您必须更新视图层次结构的布局，以最佳方式使用可用空间。

内部变化

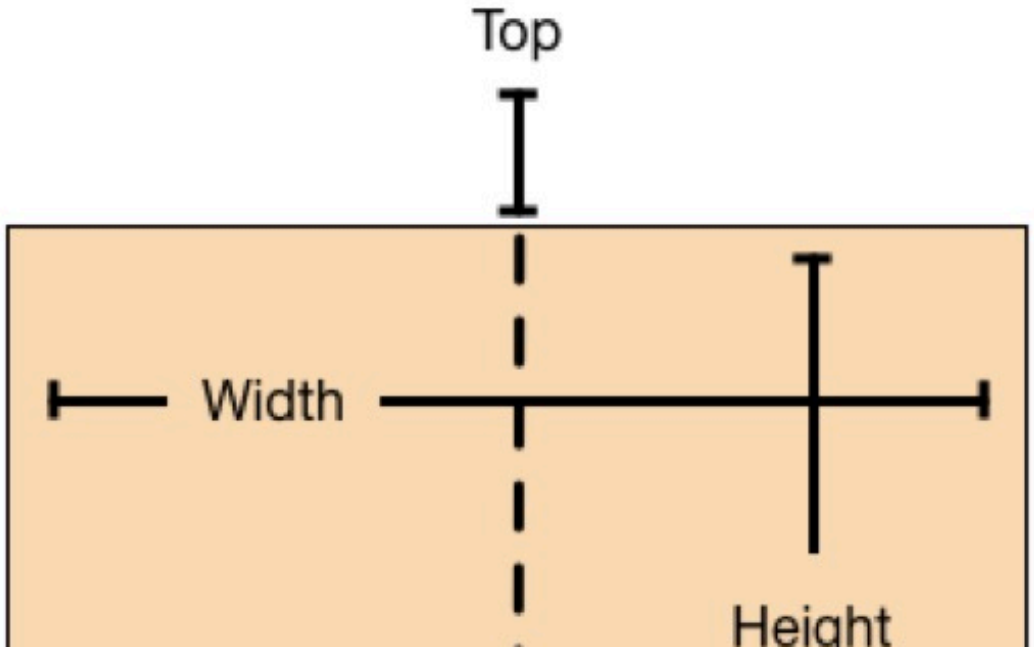
当用户选择的语言、系统主题发生变化，元素也需要跟随变化。

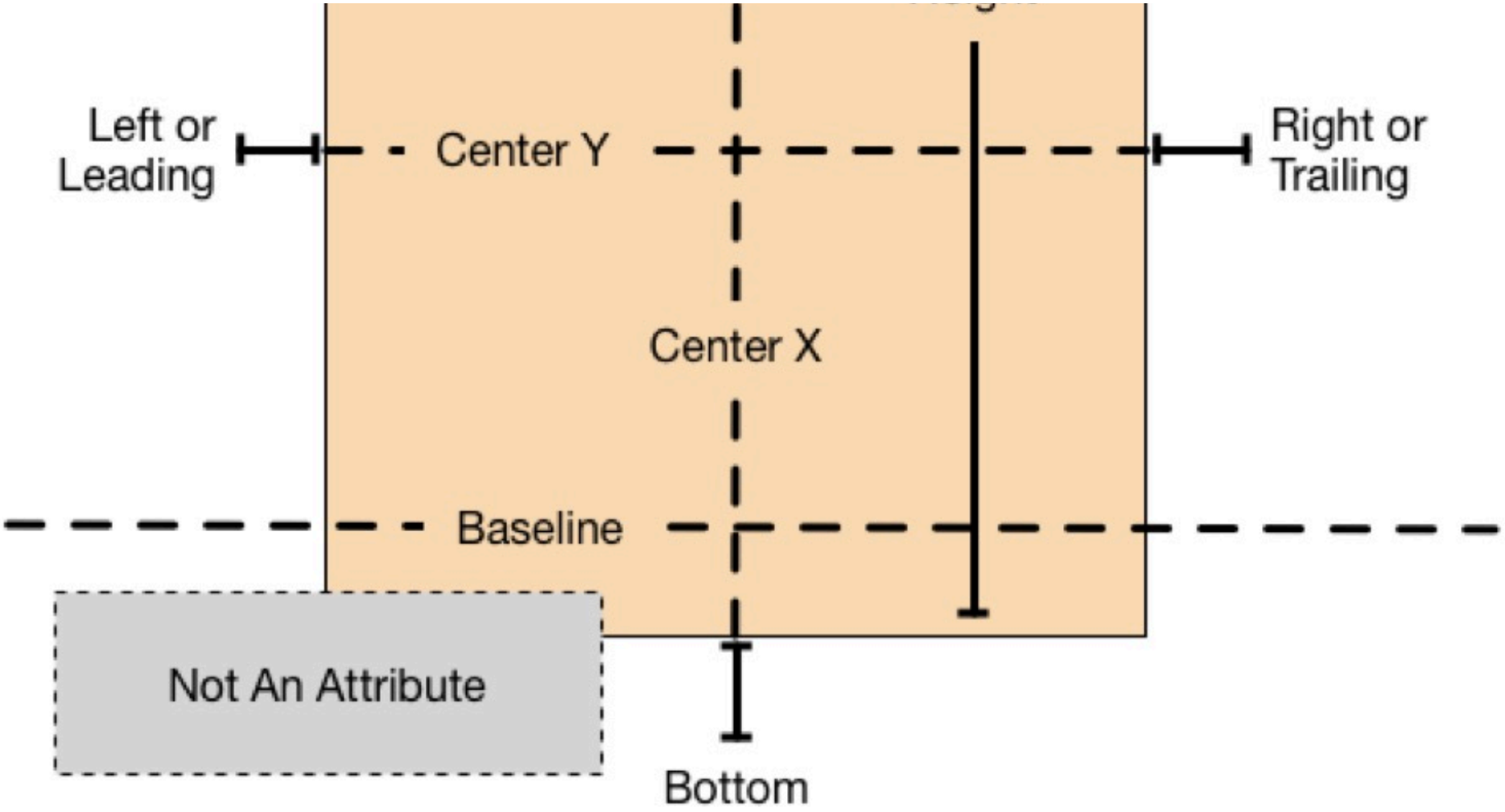




需要对齐的属性

在自动布局中，属性定义了一个可以限制的特征。一般来说，这包括四个边缘（前缘、后缘、顶部和底部），** 以及高度、宽度、垂直和水平中心 。 文本项也有一个或多个基线属性 **。





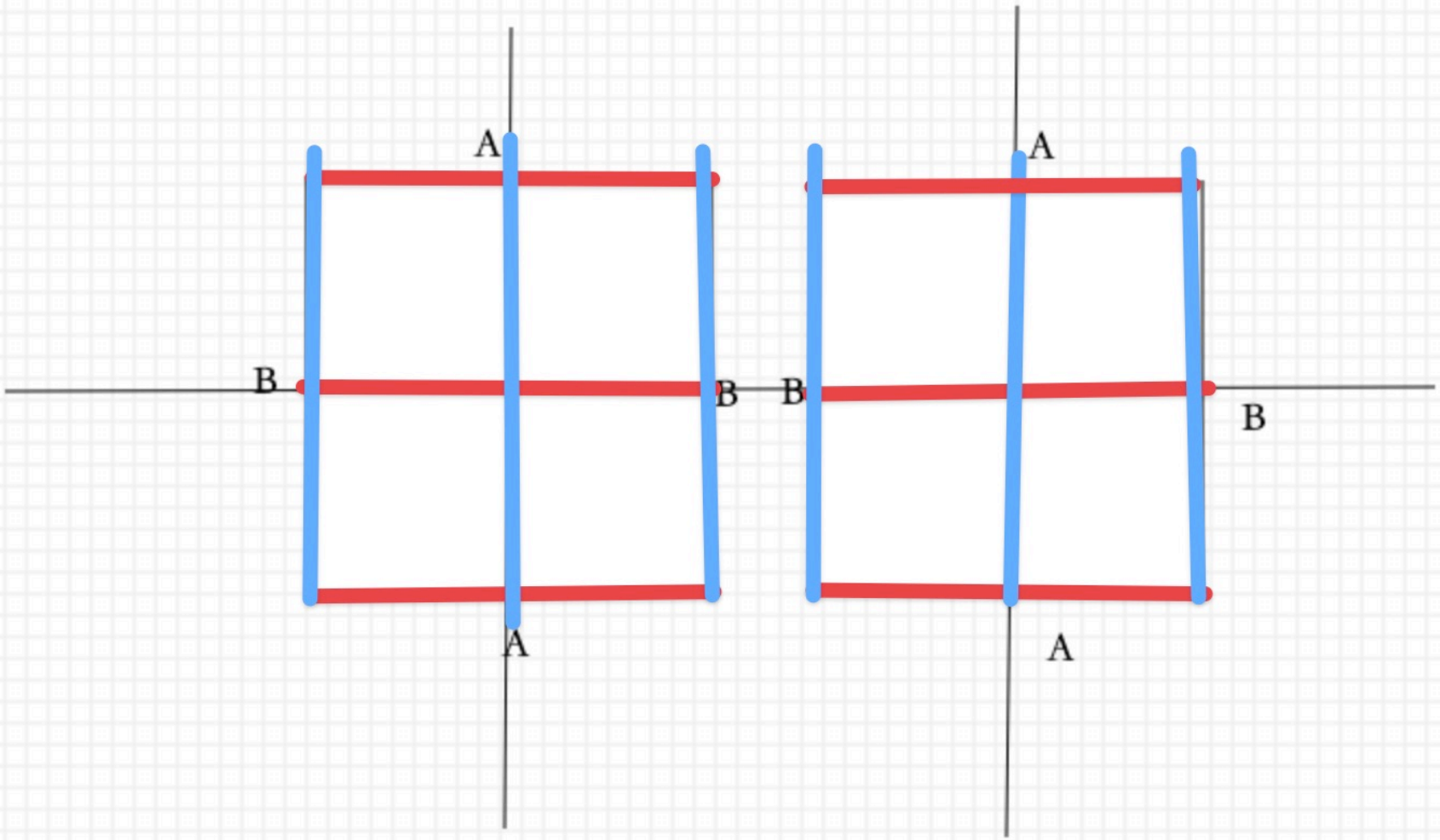
限制

对于位置属性，您不能将垂直属性约束为水平属性。

———————————————

其实文档写的非常详细，只是大部分都和 `web` 无关，那么只需要关注一些和 `web` 有关的即可，总结大概如下几点

1. 需要对其的属性除了元素的四个边以外，还有中间线
2. 对齐基于顶线底线互相对齐，左右线互相对齐的原则，不要把顶线和左右线对齐
3. 画布的边缘需要留一些边距，不要让元素贴进边缘
4. 可以做一个吸附效果，当对齐线互相靠近到达一个指定的像素内的时候，自动调整位置对齐


A
B
B
A
A
B
B
A

基本原理大概就如图片所画，红色的 A 线只能和 A 线互相对齐，蓝色的 B 线和 B 线互相对齐，不要 A、B 互相对齐。每次元素移动的时候，需要用元素的六条线分别于周围元素的六条线互相对齐比对，基于上面说的对齐方式，没有深入优化的情况下大概需要 18 次对比，当判断距离小于自动对齐线的一个偏移值的时候显示一条对齐线，当小于自动吸附偏移值的时候自动调整位置对齐，完成吸附效果。

基本原理和实现方案搞清楚，接下来就是实践过程了，方便实现，依旧使用 `konva` 来完成

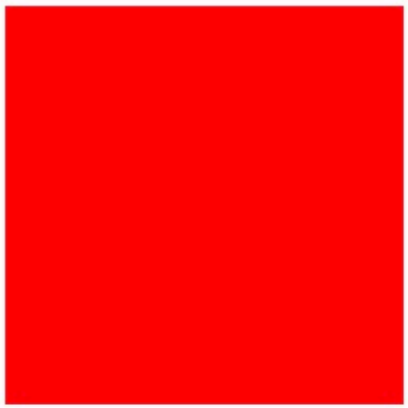
实践

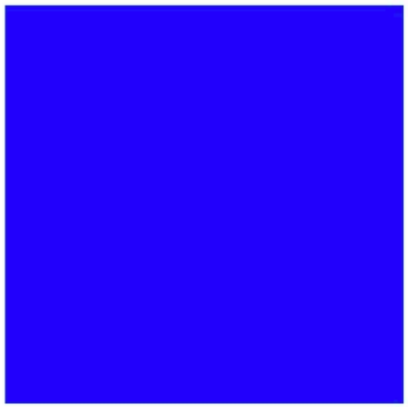
首先生成两个 shape 用作拖动使用

```
const layer:Layer = new Konva.Layer({});
const [redRect:Rect, blueRect:Rect] = ['red', 'blue'].map(
  (fill:string, index:number) =>
    new Konva.Rect({
      x: 100 + index * 150,
      y: 100 + index * 150,
      width: 100,
      height: 100,
      fill,
```

```
        draggable: true,
      }),
);
layer.add(redRect);
layer.add(blueRect);
stage.add(layer);
```

得到这样的两个图形





同时生成一根辅助线，需要先暂时隐藏起来

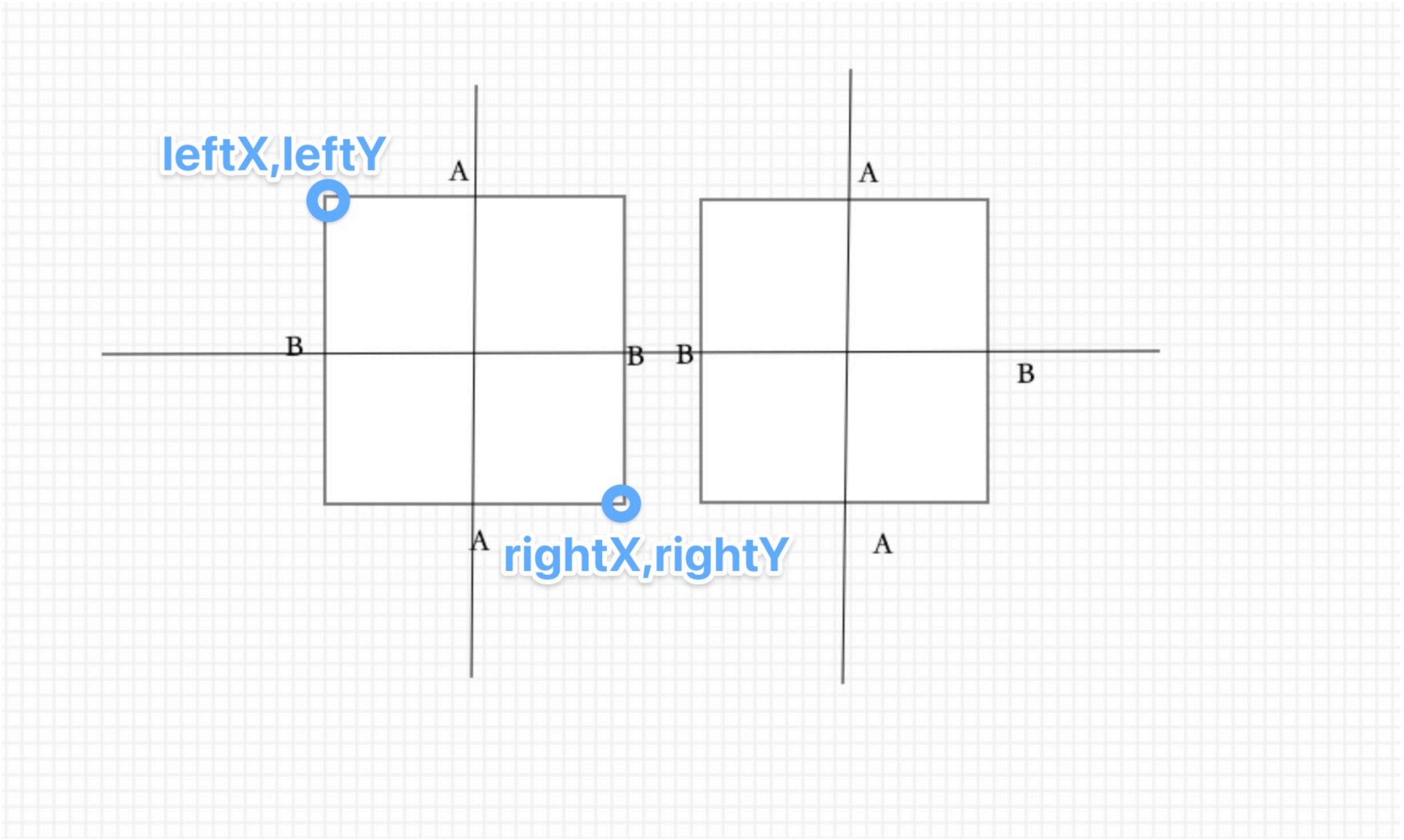
```
const AuxLine :Line<Config>  = new Konva.Line({
  points: [0, 0],
  opacity: 0,
  stroke: 'green',
  strokeWidth: 2,
  lineJoin: 'round',
  dash: [33, 10],
});
layer.add(AuxLine);
```

接着在拖动事件里监听位置

```
blueRect.on('dragmove', (e) => {
  // 获取当前的 xy，然后通过宽度来计算四个顶点以及中点的坐标
  const currentX = e.target.x();
  const currentY = e.target.y();
  const {x: targetX, y: targetY} = redRect.getPosition();
  const {width: currentWidth, height: currentHeight} = blueRect.getSize();
  const {width: targetWidth, height: targetHeight} = redRect.getSize();
  const current = {
    leftX: currentX,
    leftY: currentY,
    rightX: currentX + currentWidth,
    rightY: currentY + currentHeight,
    mediumX: Math.ceil(currentX + currentWidth / 2),
    mediumY: Math.ceil(currentY + currentHeight / 2),
  };
  const target = {
    leftX: targetX,
    leftY: targetY,
    rightX: targetX + targetWidth,
    rightY: targetY + targetHeight,
    mediumX: Math.ceil(targetX + targetWidth / 2),
    mediumY: Math.ceil(targetY + targetHeight / 2),
  };
});
```

当然这里需要注意性能问题，比如做一下节流，把 `target` 这种没有移动的元素位置以及尺寸的计算信息缓存起来，同时需要注意，`target` 可能不止一个，所以需要在 `dragStart` 的时候计算一次存一个数组，在 `dragMove` 的过程中对比整个数组，在 `dragEnd` 的时候清空数组，为了方便演示这里就不做这个过程了。

命名大概是这个定位，一个矩形只需要两个点就能确定位置，分别记为 left 和 right

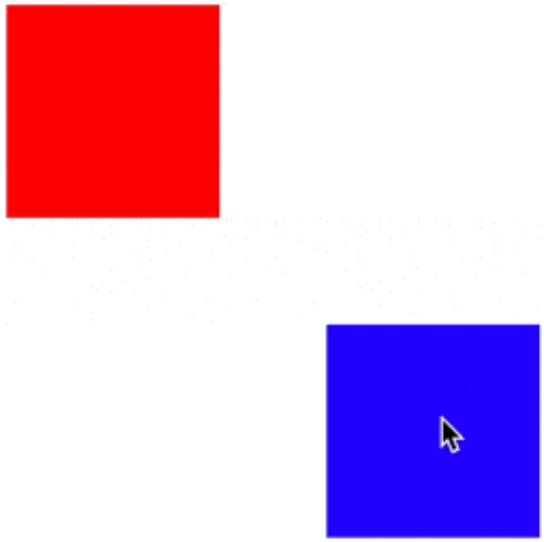


接着为了方便了解原理，实现一个最简单的，顶线互相对比

```
// 当判断顶线互相靠近距离小于等于 8，把辅助线设置为显示，辅助线起点 x 为目标的 x，终点 x 为当前元素的 x，起点 y 和重点 y 为目
```

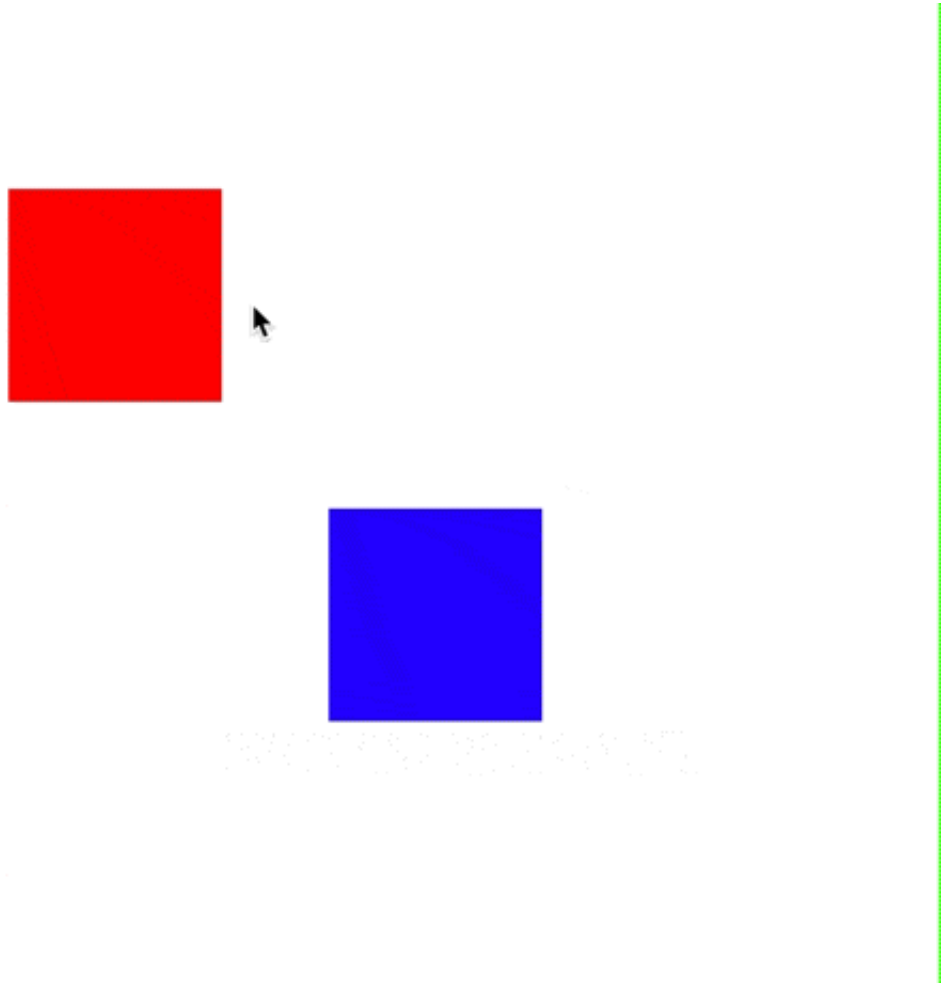
```
标图形的 y
if (Math.abs(current.leftY - target.leftY) <= 8) {console.log('贴近了');
  auxLine.setAttrs({points: [target.rightX, target.leftY, current.leftX, target.leftY],
    opacity: 1,
  });} else if (auxLine.getAttr('opacity') === 1) {
  auxLine.setAttrs({points: [0, 0],
    opacity: 0,
  });}
```

可以看到基本效果是实现了，但是依旧存在一些问题，例如图形移动到左边的时候显然辅助线应该从 `target` 的 `leftX` 开始，这里需要判断一下，辅助线永远取最短的路径，当然，基本效果实现之后优化还是比较简单的，这里就不继续写优化部分，开始实现自动吸附功能。



```
if (Math.abs(current.leftY - target.leftY) <= 8) {console.log('贴近了');
  auxLine.setAttrs({points: [target.rightX, target.leftY, current.leftX, target.leftY],
    opacity: 1,
  });

 // 设置吸附位置
  blueRect.setPosition({x: current.leftX, y: target.leftY});
} else if (auxLine.getAttr('opacity') === 1) {
  auxLine.setAttrs({points: [0, 0],
    opacity: 0,
  });}
```



可以看到，吸附效果已经很简单的实现了，但是需要优化一下，8 的距离还是太大，可以根据需要自行调整，同时还可以优化只吸附一次，吸附后如果移动不超过一定的距离，不再次吸附（否则的话其实是等于强制对齐了元素，仔细看视频，当吸附以后再移动其实是无效的，必须移动足够远才能脱离）。

结束语

总体而言实现这个功能还是不太难的，主要是优化的部分非常多，不止是性能方面，还有体验方面都需要做深度的优化，还是比较麻烦的，具体可以参照 `sketch`，但是实际效果使用起来倒是不错。

代码留存

```
export default function Index() {
  const ref = useRef<HTMLDivElement>(null);

  useEffect(() => {
    const dom = ref.current;
    if (!dom) return;

    const stage = new Konva.Stage({
      container: dom,
      width: dom.clientWidth,
      height: dom.clientHeight,
    });
    const layer = new Konva.Layer({});
    const [redRect, blueRect] = ['red', 'blue'].map(
      (fill, index) =>
        new Konva.Rect({
          x: 100 + index * 150,
          y: 100 + index * 150,
          width: 100,
          height: 100,
          fill,
          draggable: true,
        }),
    );
    const auxLine = new Konva.Line({
      points: [0, 0],
      opacity: 0,
      stroke: 'green',
      strokeWidth: 2,
```

```
    lineJoin: 'round',
    dash: [3],
  });
  layer.add(auxLine);
  layer.add(redRect);
  layer.add(blueRect);

  blueRect.on('dragmove', (e) => {
    // 获取当前的 xy，然后通过宽度来计算四个顶点以及中点的坐标
    const currentX = e.target.x();
    const currentY = e.target.y();
    const { x: targetX, y: targetY } = redRect.getPosition();
    const { width: currentWidth, height: currentHeight } = blueRect.getSize();
    const { width: targetWidth, height: targetHeight } = redRect.getSize();
    const current = {
      leftX: currentX,
      leftY: currentY,
      rightX: currentX + currentWidth,
      rightY: currentY + currentHeight,
      mediumX: Math.ceil(currentX + currentWidth / 2),
      mediumY: Math.ceil(currentY + currentHeight / 2),
    };
    const target = {
      leftX: targetX,
      leftY: targetY,
      rightX: targetX + targetWidth,
      rightY: targetY + targetHeight,
      mediumX: Math.ceil(targetX + targetWidth / 2),
      mediumY: Math.ceil(targetY + targetHeight / 2),
    };

    // 吸附效果
```

```
    if (Math.abs(current.leftY - target.leftY) <= 4) {
      blueRect.setPosition({ x: current.leftX, y: target.leftY });
    }

    // 当判断顶线互相靠近距离小于等于8，把辅助线设置为显示，辅助线起点x为目标的x，终点x为当前元素的x，起点y和重点y为目标图形的y
    if (Math.abs(current.leftY - target.leftY) <= 8) {
      console.log('贴近了');
      auxLine.setAttrs({
        points: [target.rightX, target.leftY, current.leftX, target.leftY],
        opacity: 1,
      });
    } else if (auxLine.getAttr('opacity') === 1) {
      auxLine.setAttrs({
        points: [0, 0],
        opacity: 0,
      });
    }
  });

  stage.add(layer);
}, []);

return (
  <div
    css={{
      display: 'flex',
      width: '100%',
      height: '100%',
    }}
  >
    <div>
      <Button draggable={true} type={'primary'}>
```

```
          按钮
        </Button>
      </div>
      <div
        ref={ref}
        css={{
          width: 900,
          height: 800,
        }}
      />
    </div>
  );
}
```