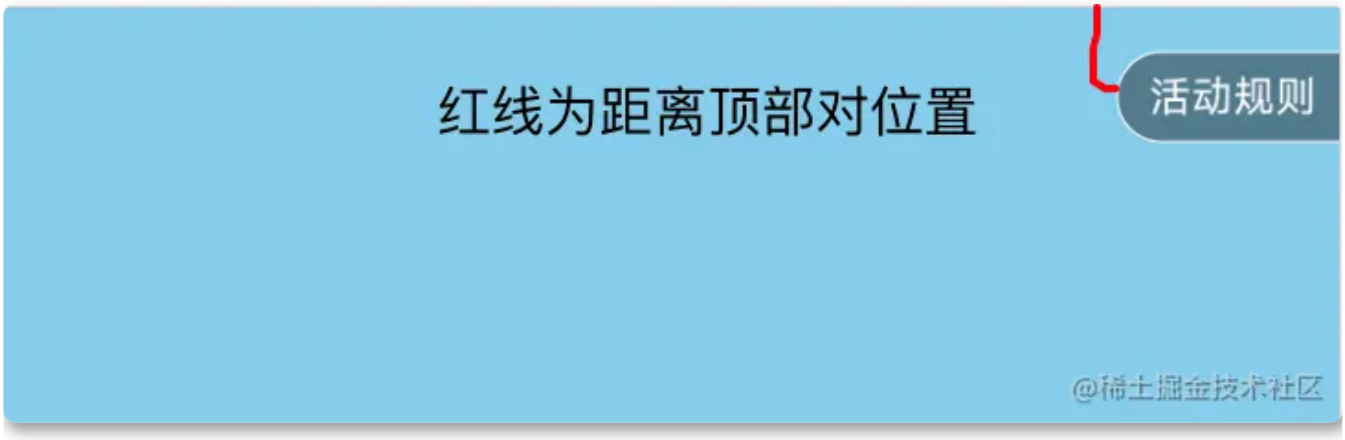
className 还能这么用，你学会了吗

抛出问题

`className` 大家都用过吧，用它在 `react` 项目中设置样式。它的用法很简单，除了可以设置一个样式外， `react` 中也可以使用 `className` 引入多个类样式。

这次在写项目的时候，碰到一个非常小但是当时却一直解决不了的问题。后面在复盘的时候将它解决了。问题大致是这样的：

有两个活动页，每个活动页上都有一个活动规则图标来弹出活动规则，活动规则图标距离顶部会有一个值。现在问题就是这个活动规则在这两个活动页距离顶部的这个值是不一样的，但是我已经将这个活动规则图标做成了组件，并在这两个活动页里都调用了它，从而导致两个页面的样式会相同。如下图所示：



解决问题

这个问题不算很大，但是属于细节问题。就和我的组长所说的一样，一个项目应该要做到先完成再完美。所以我当时的解决方法是再写一个活动规则组件，只是将距离顶部的值做出修改即可。效果确实是达到了，不过在最后复盘代码的时候，组长注意到了这两个组件，并开始询问我为什么这样做。

组长： `Rule_1` 和 `Rule_2` 这两个组件是什么意思，我看它们没有很大的区别呀。

我便简单说了一下缘由。

组长接着说：你忘了组件是什么吗？一个 `CSS` 样式值不同就大费周章地新增一个组件，这岂不是太浪费了。再去想想其他方案。

通过这一番谈话我想起了组件化思想的运用，发现之前解决的这个小问题解决的并不够好。于是，我就带着组件化思想又来重新完善它。

我重新写了一个 **demo** 代码，将主要内容和问题在 **demo** 代码中体现出来。下面是原版活动规则组件 **demo** 代码，之后的代码都是基于 **demo** 代码完成的

javascript复制代码

```
import React from "react";
import "./index.css";
const Header = ({ onClick }) => {
  return (
    <>
      <div className="container_hd">
        <div
          className='affix'
          onClick={onClick}
        ></div>
      </div>
    </>
  );
};
export default Header;
```

组件化思想

我自己问自己：既然已经写好了一个活动规则组件，为什么仅仅因为一个样式值的不同而去新增一个功能一样的组件？很显然，这种方法是最笨的方案。既然是组件，那就应该要有复用性，或者说只需在原有的基础上稍加改动就可达到效果。

这是样式的问题，因此要从根本上解决问题。单纯地修改 **CSS** 样式肯定不行，因为两个页面两个不同的样式。

className 运用

className 就不用多介绍了，经常能使用，咱们直接来看如何解决问题。在这里我定义了一个 **Value** 值，用来区分是在哪个页面的，比如分别有提交页和成功页，我在成功页设置一个 **Value** 值，，然后将 **Value** 值传入到活动规则组件，那么在活动规则组件里只需要判断 **Value** 值是否等于成功页的 **Value** 值即可。在 **className** 处做一个三元判断，如下所示：

javascript复制代码

```
className={`affix_${Value === "0" ? "main" : "submit"}`}
```

相当于如果 **Value** 等于0的时候类名为「**affix_main**」，否则为「**affix_submit**」。最后再 **css** 将样式完善即可。完整代码可以参考如下：

- 成功页组件

```ini
import Header from "./components/Header";

const Success = () => {
  const Value = "0";
  return (
    <div style={{ backgroundColor: "purple", width: "375px", height: "670px" }}>
      <Header Value={Value}></Header>
    </div>
  );
};

export default Success;
```

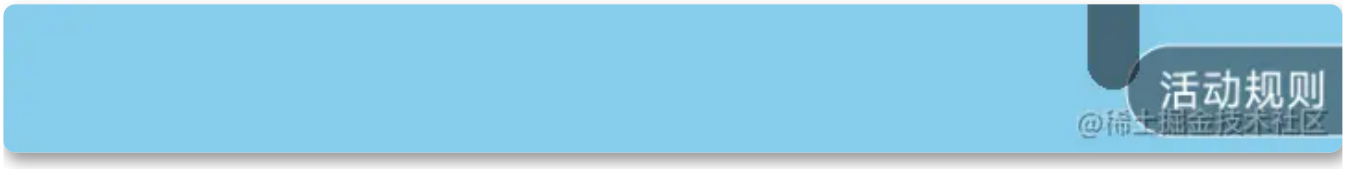
- 活动规则组件

```javascript
import React from "react";
import "./index.css";
const Header = ({ onClick, Value }) => {
  return (
    <>
      <div className="container_hd">
        <div
          className={`affix_${Value === "0" ? "main" : "submit"}`}
          onClick={onClick}
        ></div>
      </div>
    </>
  );
};
export default Header;
```

- 活动规则组件样式

```css
.container_hd {
  width: 100%;
}
```

```
.affix_main {
  position: absolute;
  top: 32px;
  right: -21px;
  z-index: 9;
  width: 84px;
  height: 26px;
  background: url('./assets/rule.png');
  background-size: contain;
  background-repeat: no-repeat;
}
.affix_submit {
  position: absolute;
  top: 12px;
  right: -21px;
  z-index: 9;
  width: 84px;
  height: 26px;
  background: url('./assets/rule.png');
  background-size: contain;
  background-repeat: no-repeat;
}
```



通过对比效果图可以看出，两者的效果确实发生变化。完成之后，我心里在想：为什么当时就没想出这个简单易行的方案呢？动态判断并设置类名，至少比最开始的新增一个组件的方法高级多了。

总结问题

对于这个问题的解决就这样告一段落了，虽然看起来比较简单（一个动态设置类名），但是通过这个 **className** 的灵活使用，让我对 **className** 的用法有了更进一步的掌

握，也不得不感叹组件化思想的广泛运用，这里最大程度地将组件化思想通过 `className` 发挥出来。

因此，希望通过这个问题，来学会 `className` 的灵活用法，并理解好组件化思想。当然如果大家还有更好的解决方案的话，欢迎在评论区告诉我。