🡪

React 有一个特殊的属性(prop) children。主要用于组件需要渲染内容，但它并不知道将来具体要渲染什么内容的情况。怎么会有这种使用场景？确实比较少，但并不是没有，比如弹出框。当你写一个弹出框组件的时候，你知道它要弹出什么吗？肯定不知道，只有使用的时候才知道。那为什么要写弹出框组件？虽然内容不一样，但框是一致的，居中啊，阴影啊，宽度啊，高度啊，每一个弹出框都一样，所以有必要形成一个组件，代码复用。框写好了，那到时候用的时候，具体的内容怎么放到框里面？那就要在框中占一个位置，如果有内容候就放到这个地方，占位置使用的就是this.props.children.

　　根据描述，children 最简单的使用场景就是一个组件中直接写this.props.children,调用这个组件的时候，再具体写内容。这里要注意的是组件调用方式，.假设包含children的组件叫theme, 那么调用的时候，就要在theme组件两个标签之间写内容，<theme><message/></theme>, 只有这样<message />才会被theme组件中的children 获取到。为什么要这要写？想一想html 标签， 只有在两个标签之间内容才被称之为children, 这里的children 也是同样的道理，只不过标签换成了组件。

🡪

React中一般我们将组件标签写为<Component />. 但是这并不是必须的，实际规则是如果标签不需要包裹其他内容,便可以简洁的写为单闭合标签,就和<input />一样道理。如果需要包含其他内容，则其为包裹组件可以写为<Component>….. </Component>.

🡪

包裹组件中将使用props.children， 它将指代包裹组件开标签以及闭合标签之间的所有内容。主要的意义在于**设计包裹组件**，或者该包裹组件作为工具包让其他人使用时，我们不知道未来此包裹组件将具体包含什么内容，所以引入props.children进行统一的指代。

一般在包裹组件的render方法中将使用{this.props.children}。然后生成组件的时候所有包含在包裹组件开标签以及闭合标签之间的内容，都将放置在包裹组件render方法中的{this.props.children}位置。

🡪

包裹组件中具体渲染出来的内容由包裹组件以及被包裹组件共同决定。例如包裹组件selector，子组件ActionButton以及Message

假设在Apps.js中render方法返回结构为

<selector>

<ActionButton theme={this.state.theme} />

<Message theme={this.state.theme} />

</selector> //从此在selector这个包裹组件中，this.props.children指代<ActionButton>以及<Message>组件。但具体渲染内容还要看<selector>组件的render方法以及<ActionButton>,<Message>组件的渲染方法。

那么包裹组件selector的render方法中可以写为：

return <div>{this.props.children}</div>

意思是将直接普通的渲染出所有子组件ActionButton以及Message. 而ActionButton以及Message子组件具体渲染出什么内容由子组件决定。

包裹组件selector的render方法也可以写为：

return <div>Hello</div>

那么虽然包裹组件中含有子组件<ActionButton>以及<Message>， 但是包裹组件的render方法返回的渲染内容不包含子组件，所以子组件ActionButton以及Message组件不会被渲染。

一般情况是对包裹组件中的子组件进行一些处理，例如map + clone处理后进行渲染。

🡪

children 可以是任何内容， 这些方法正是为了children 属性量身定做的。重新认识一下children， 它可以是字符串，可以是函数，可以是数组，可以是undefined ， 可以是null， 可以是Boolean，几乎，你想到的都可以是children的内容。

例如ThemeSelector作为包裹组件：

<ThemeSelector>123</ThemeSelector>

<ThemeSelector>{undefined}</ThemeSelector>

<ThemeSelector>{null}</ThemeSelector>

<ThemeSelector>{(() => 'helloWorld')()}</ThemeSelector>

<ThemeSelector>{[4,5,6]}</ThemeSelector>

<ThemeSelector>{false} {true}</ThemeSelector>

　　只不过 null ，undefined，false, true 并不会渲染到页面上，由于提供给children 内容的各种各样，this.props.children的返回值也是不同，如果children没有获取到内容，它返回的就是undefined, 如果只有一个内容， 它返回的就是一个object, 如果有多个，它返回的就是数组。正是由于children的复杂性，React才提从提供了以下几个方法来帮我们正确地使用children 属性。

React提供了几个自带的功能性函数用于处理props.children

|  |  |
| --- | --- |
| React.Children.map(children, function[(thisArg)] | React.Children.map函数需要2个参数。第一个参数是组件的children, 一般为this.props.children.第二个参数是一个函数，该函数接受一个参数为children中的每一个元素，React将对children中每一个元素调用该函数并且将结果放入数组中进行返回。 |
| React.Children.forEach | 和React.Children.map 的使用方法一致，只不过不返回内容 |
| React.Children.count | 接受children 作为参数，计算children的数量 |
| React.Children.only | 判断获取的children是不是只有一个，如果是，就返回这个children, 如果不是，则报错 |
| React.Children.toArray | React.children.toArray: 接受一个参数，通常为props.children.把children 转化为数组，可以使用数组中的方法来操作children， 比如反转或删除 |
| React.cloneElement | 用于复制一个React元素，第一个参数是原React元素，第二个参数是新加入的props对象。返回的结果为一个克隆出来的新的React元素，并且key和ref都保留。  克隆children,  为什么要克隆呢？因为获取到children 之后，它是只读的，并不能给它添加属性，如果想给children添加属性，那只能先复制一份到组件中，然后再添加属性，所以它接受两个参数，第一个是某个children对象, 即是要复制的children，第 二个是属性对象，代表将要要添加的给某个children对象的属性。  我们可以连用React.Children.map函数以及React.cloneElement函数，为组件的Children批量的新增props。[这样的原因是我们不能直接修改props.children] |