**组件**

[一、浅谈组件 1](#_Toc518059669)

[二、创建组件的两种方式（两种） 1](#_Toc518059670)

[三、基于继承component来创建组件 2](#_Toc518059671)

[四、“受控组件”和“非受控组件” 3](#_Toc518059672)

[五、vue【mvvm】和react【mvc】 3](#_Toc518059673)

[六、复合组件和平行组件 3](#_Toc518059674)

### 一、浅谈组件

不管是vue还是react框架，设计之初都是期望我们按照“组件、模块管理”的方式来构建程序的，也就是把一个程序划分为一个个的组件来单独处理

**【优势】**

1. 有助于多人开发

2. 我们开发的组件可以被复用

…

### 二、创建组件的两种方式（两种）

函数声明式组件、基于继承component类来创建组件

**【函数式】**

1. 操作简单

2. 能实现的功能也很简单，只是简单的调取和返回JSX而已

**【创建类式】**

1. 操作相对复杂一些，但是也可以实现更为复杂的业务功能

2. 能够使用生命周期函数操作业务

3. 函数式可以理解为静态组件（组件中的内容调取的时候就已经固定，很难修改，而这种方式，可以基于组件内部的状态来动态更新渲染内容）

4….

### 三、基于继承component来创建组件

基于create-element把jsx转换为一个对象，当render渲染这个对象的时候，遇到type是一个函数或者类，不是直接创建元素，而是先把方法执行：

-》如果是函数式声明的组件，就会把它当做普通方法执行（方法中的this是undefined），把函数返回的jsx元素（也就是解析后的对象）进行渲染

-》如果是类声明式的组件，会把当前类new执行，创建一个类的实例（当前本次调取的组件就是他的实例），执行constructor之后，会执行this.render();把render中返回的jsx拿过来渲染，所以“类声明式组件，必须有一个render的方法，方法中需要返回一个jsx元素”

但是不管哪种方式，最后都会把解析出来的props属性对象作为实参传递给对应的函数或者类

**知识点：**

1. create-element在处理的时候，遇到一个组件，返回的对象中，type就不是字符串标签名了，而是一个函数（类），但是属性还存在props

{  
 type:Dialog,  
 props:{  
 lx:1,  
 con:'xxx',  
 children:一个值或者一个数组  
 }  
}

2. render渲染的时候，我们需要做处理，首先判断type的类型，如果是字符串，就创建一个元素标签，如果函数或者是类，就把函数实行，把props中的每一项（包含children）传递给函数

3. 在执行函数的时候，把函数中return中jsx转换为新的对象（通过create-element），把这个对象返回；紧接着render按照以往的渲染方式，创建dom元素，插入到指定的容器中即可

### 四、“受控组件”和“非受控组件”

1. 基于数据驱动（修改状态数据，react帮我们重新渲染视图）完成的组件叫做“受控组件”

2. 基于ref操作dom实现视图更新的，叫做“非受控组件”

真实项目中，建议多使用“受控组件”

### 五、vue【mvvm】和react【mvc】

【vue】【MVVM】数据更改视图跟着更改，视图更改数据也跟着更改（双向数据绑定）

【react】【MVC】数据更改视图跟着更改（原本单向数据绑定，但是我们可以自己构建双向的效果）

### 六、复合组件和平行组件

##### 一、复合组件：父组件嵌套子组件

**1、【传递信息的方式】**

**A：父组件需要把信息传递给子组件**

**1.“属性传递”**： 调取子组件的时候，把信息基于属性的方式传递给子组件（子组件PROPS中存储传递的信息）;这种方式只能父组件把信息传递子组件，子组件无法直接的把信息传递给父组件，也就是属性传递信息是单向传递的；

export default class Vote extends React.Component {  
 render() {  
 let {title, count} = this.props;  
 return <section className={'panel panel-default'} style={{width: '50%', margin: '20px auto'}}>  
 <VoteHead title={title}/>  
 ...  
 </section>;  
 }  
}

**2. “上下文传递”**：父组件先把需要给后代元素（包括孙子元素）使用都设置好（设置在上下文中），后代组件需要用到父组件的信息，主动去父组件中调取使用即可

**父组件设置信息**

static childContextTypes = {  
 //=>设置上下文中信息值的类型  
 n: PropTypes.number,  
 m: PropTypes.number  
};  
getChildContext() {  
 //->RETURN的是啥，相当相当于往上下文中放了啥  
 let {count: {n = 0, m = 0}} = this.props;  
 return {  
 n,  
 m  
 };  
}

**子组件主动获取需要的信息**

static contextTypes = {  
 //=>首先类型需要和设置时候类型一样，否则报错；并且你需要用啥，就写啥即可；  
 n: PropTypes.number,  
 m: PropTypes.number  
};

**属性VS上下文**

1. 属性操作起来简单，子组件是被动接收传递的值（组件内的属性是只读的），只能父传子（子传父不行，父传孙也需要处理：父传字，再传孙）；

2. 上下文操作起来相对复杂一些，子组件是主动获取信息使用的（子组件是可以修改的获取到上下文信息的，但是不会影响到父组件中的信息，其他组件不受影响）

##### 二、 平行组件：兄弟组件或者毫无关系的两个组件

1、方案一：让两个平行组件拥有一个共同的父组件

父：parent

子：A/B

父组件中有一些信息，父组件把一个方法传递给A，A中把方法执行（方法执行修改父组件信息值），父组件再把最新的信息传递给B即可，等价于A操作，影响了B

2.、 方案二：基于redux来进行状态管理，实现组件之间的信息传输（常用方案）

redux可以应用在任何的项目中（vue/jq,react的都可以），react-redux才是专门给react提供的方案