#### 第1章

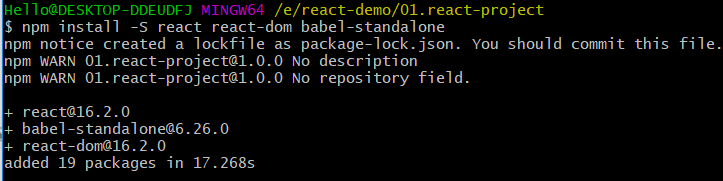
1. React安装

官网是：<https://reactjs.org/>

初始化项目：npm init //cpnm init

1. 安装依赖包

npm install react react-dom babel-standalone -S



注意使用JSX语法，因此需要安装这些依赖包。这三个都要安装才行。

react--核心库

react-dom 提供与DOM相关的功能

babel-standalone 将JSX语法转化为浏览器可以识别的ES5语法。

1. 使用JSX语法时，



注意属性的区别：class要写成className。模板后面一定要有个逗号。遇到<和>会解释成HTML，遇到大括号{和}，解释成JS。

1. 使用数组存储模板，示例如下：

**var *arr*** = [  
 <**div key="1"**>111</**div**>,  
 <**div key="2"**>222</**div**>,  
 <**div key="3"**>333</**div**>  
];  
**ReactDOM**.render(  
 <**div**>{***arr***}</**div**>,  
 **document**.getElementById(**'example'**)  
)

注意：每个实例必须有唯一一个根元素，将数组变量放在大括号中展开。

#### 第2章

1. JSX之函数

定义函数，然后调用，示例：

**function** *action*(item) {**return** item.name + **""** + item.**age**;  
}**const *user*** = {  
 **name**: **'lili'**,  
 **age**: 22  
};

**var *element*** = <**h1**>hello,{*action*(***user***)}</**h1**>;  
**ReactDOM**.render(  
 ***element***,  
 **document**.getElementById(**'example'**)  
);

1. 组件

组件定义方法：通过class继承的方式；自定义，函数。class方式，简单示例和传入属性：

**class** HelloMessage **extends React**.*Component* {  
 render() {  
 **return** <**h1**>hello, {**this**.props.name}</**h1**>  
 }  
}  
**ReactDOM**.render(  
 <**HelloMessage name="lili"**/>,  
 **document**.getElementById(**'example'**)  
);

**class** HelloMessage **extends React**.*Component* {  
 render() {  
 **return** <**h1**>hello, world</**h1**>  
 }  
}  
**ReactDOM**.render(  
 *//<HelloMessage></HelloMessage>,* <**HelloMessage**/>,  
 **document**.getElementById(**'example'**)  
);

自定义函数方式，加不上生命周期钩子的。

注意调用组件时，要关闭组件的两种方式。this表示此组件。通过自定义函数方式定义组件：



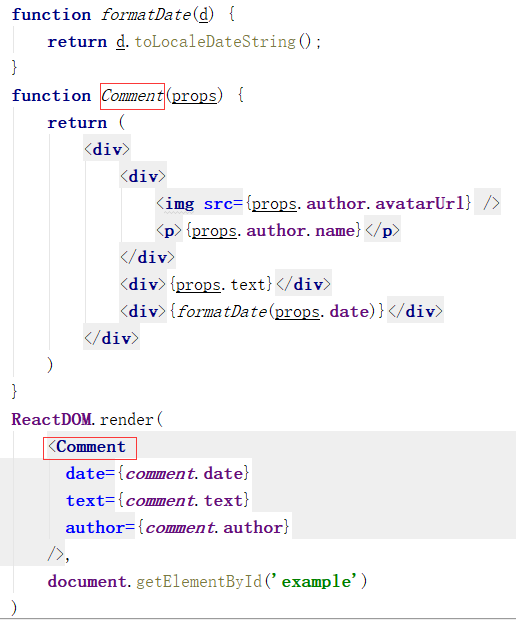
以上是两种方式定义的两个组件的混合使用。或者组件赋值给一个变量，右边示例。

组件传参，通过以下方式传参：

*//自定义组件传参，需要显式传递props(属性对象)，形参名字可以自定义***function** *Welcome*(props) {  
 **return** <**h1**>hello,{props.name},{props.**age**}</**h1**>;  
}  
**ReactDOM**.render(  
 <**Welcome name="mile" age="22"**/>,  
 **document**.getElementById(**'example'**)  
)

注意自定义函数（组件）的形参名字props表示组件的所有属性的对象，形参名字可改变。

组件之间的嵌套：



注意这里面的App组件合成了<Welcome>组件。右边的例子使用了较复杂的传参。

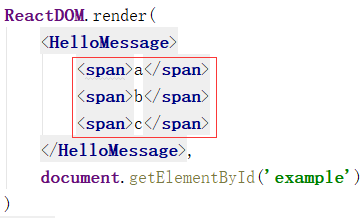
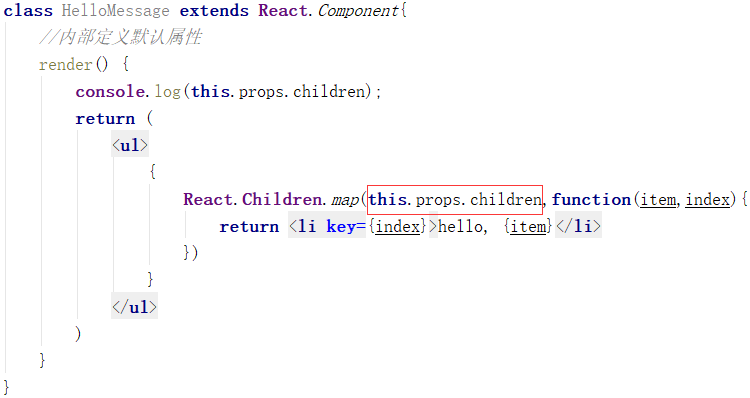
#### 第3章

1. 组件的属性默认值

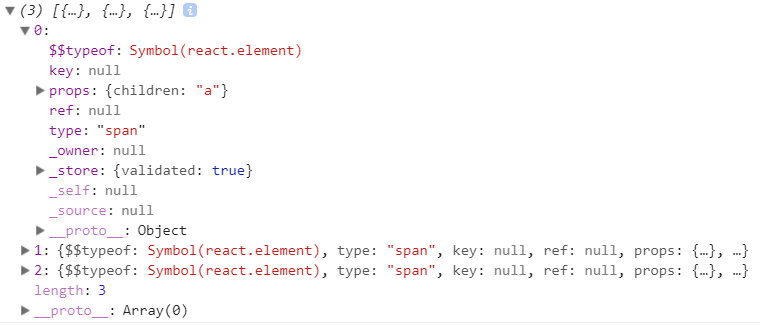


分为组件内部定义和组件外部定义两种方式。

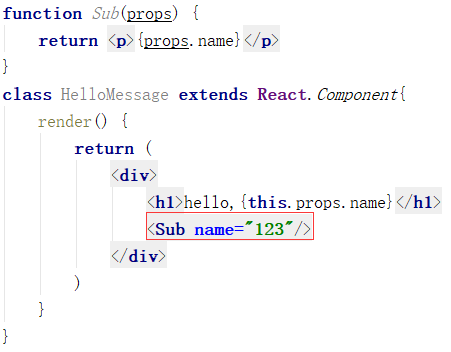
1. 获取组件的子节点，使用this.props.children:



渲染的结果和children值是：

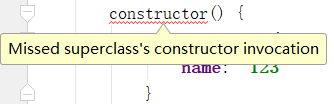
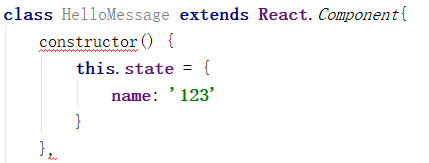


1. class语法的组件嵌套

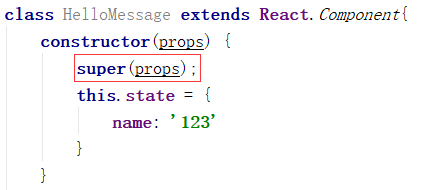


1. 状态state是可以改变的，是私有的，不能通过外部访问来修改，只能在组件内部修改，会导致组件重新render。而props主要用来传参，不直接改变它。

在构造函数中定义state，这样定义会报错：



提示必须调用父类的构造函数，正确定义如下：

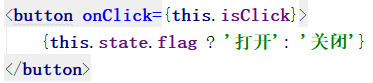
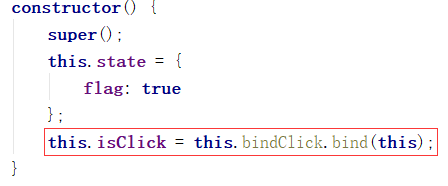


相当于子类调用父类的构造器super。super是父类构造器。注意构造器后面不要跟逗号。默认状态只能写在构造函数中。

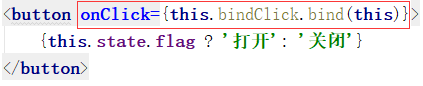
#### 事件绑定，state变化，条件渲染

1. 绑定事件有以下几种方式

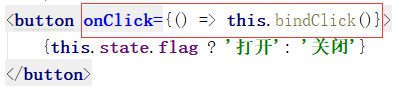
在构造器中声明一个变量，指向事件处理函数



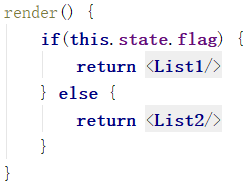
第2种是直接绑定指定的函数：



第3种方法是使用箭头函数：



1. 条件渲染



在进行条件判断时，需要写上全部的this.state.flag，不像vue那样直接写this.flag。react.js写法区分了state和props，需要显式指出，这点和vue不一样。

#### tab，event事件，生命周期

1. 组件的生命周期可分为三个状态：

**Mounting**：已插入真实DOM（挂载）

componentWillMount() 组件将要挂载

componentDidMount() 组件完成挂载

**Updating**：正在被重新渲染

componentWillReceiveProps() 组件将要接收props，这个方法在初始化render时不会被调用。

shouldComponentUpdate() 返回一个布尔值，在组件接收到新的props或者state时被调用。初始化时不被调用

componentWillUpdate() 组件将要更新

render() 渲染

componentDidUpdate() 组件完成更新立即调用。在初始化时不会调用

**Unmounting**

componentWillUnmount() 在组件从DOM中移除的时候立即被调用

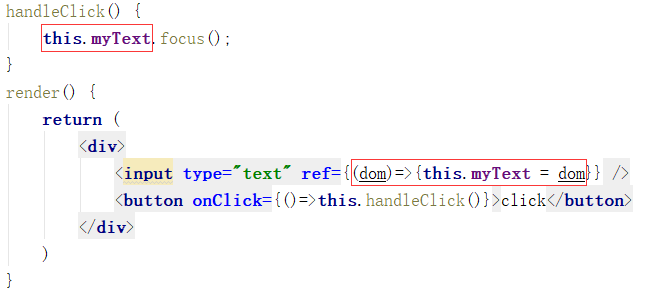
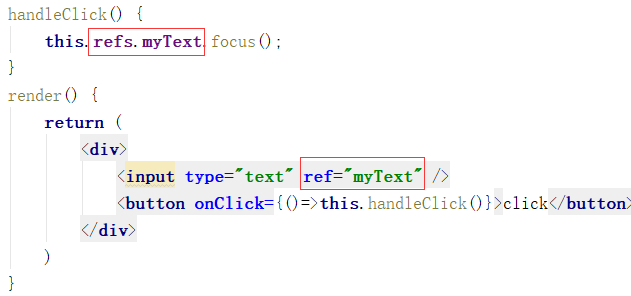
**Error** **Handling**

componentDidCatch() 错误处理

#### ref属性，表单，AJAX

1. ref属性

两种方式，一种是传字符串，用来引用这个元素；另一种是传入一个回调函数：



第二种方式，传入一个回调函数，参数表示这个元素的DOM对象，然后存储在this之中。

1. 表单（text/radio/checkbox/select）

约束性和非约束性组件（受限/非受限）

约束性组件由react管理了它的value，由状态来进行管理：

<**input type="text" value=**{**this**.state.value} **onChange=**{()=>**this**.handleChange()}/>

当内容发生变化时，调用onChange方法。

非约束性组件，value仍由原生的DOM来进行管理，不受react管理：

<**input type="text" defaultValue="123"** />

设置了默认值defaultValue，相当于原生value属性，value的值是用户输入的值。

1. AJAX

#### 校验，组件传参

1. 校验
2. 组件传参

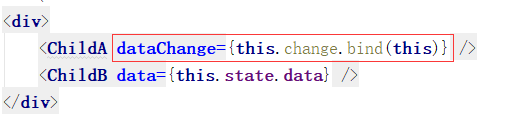
2.1 父组件-子组件

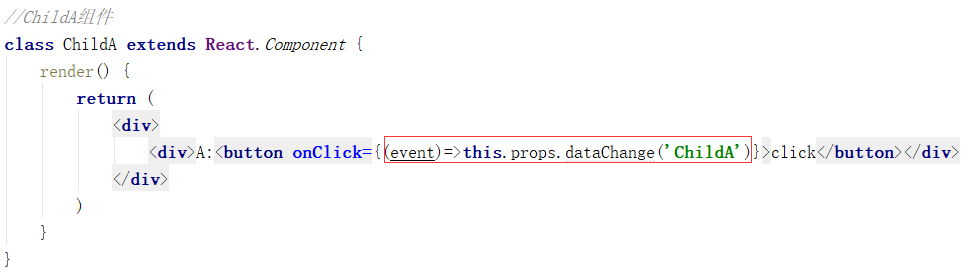
2.2 父组件-孙组件

2.3 子组件-父组件

2.4 兄弟组件

通过父组件的方法作为媒介，传给子组件的是一个方法。



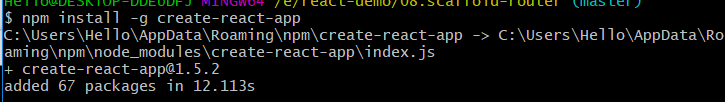


在子组件中调用的属性是一个方法dataChange，调用时传递参数。

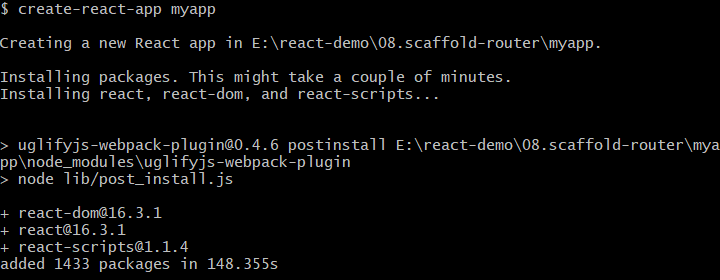
#### 第8章 react脚手架、路由

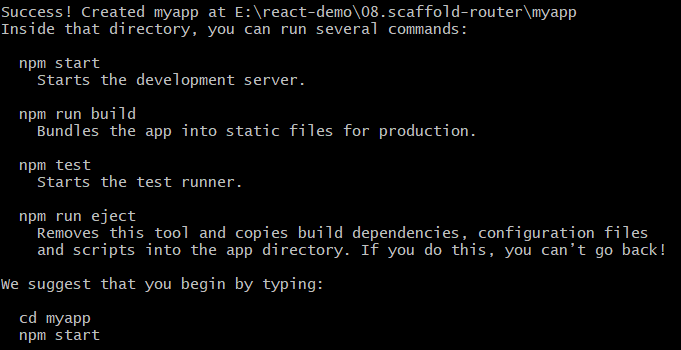
1.脚手架使用create-react-app这个npm包。

npm install -g create-react-app

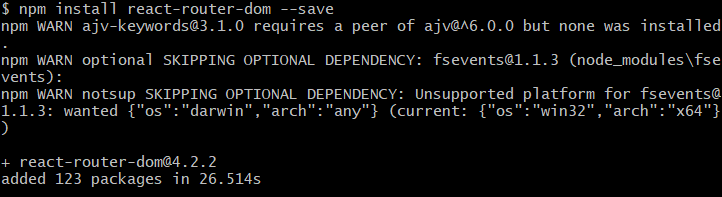


create-react-app myapp



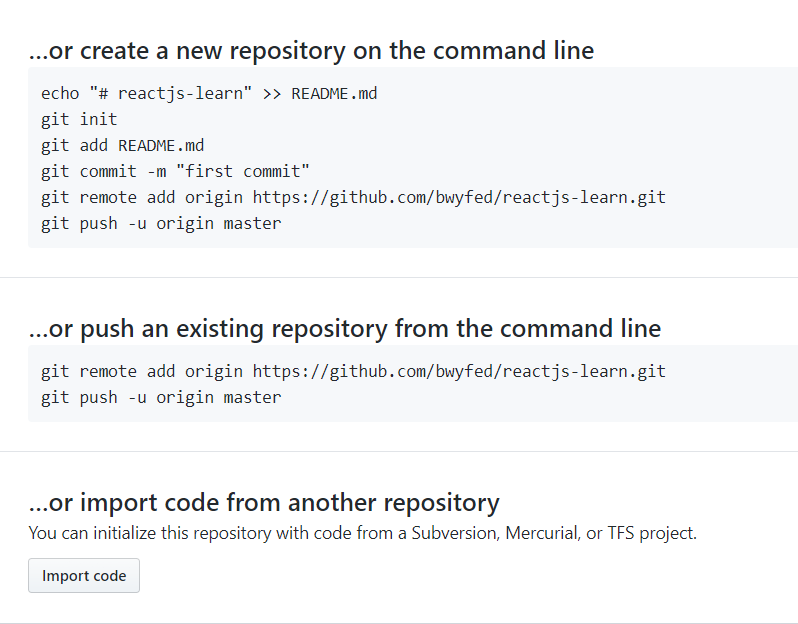


路由功能，需要安装react-router-dom：

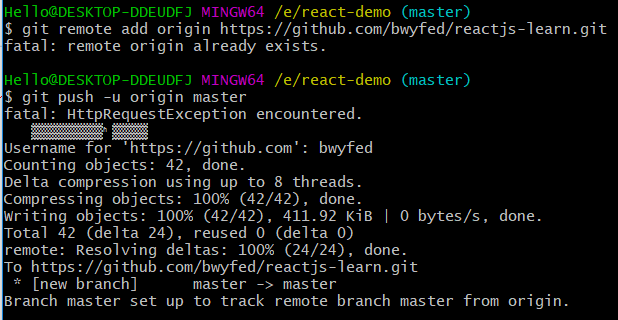


提交到github上：

github地址是：<https://github.com/bwyfed/reactjs-learn>



我在本地已经执行了git init初始化了git项目。因此使用第二种方法：

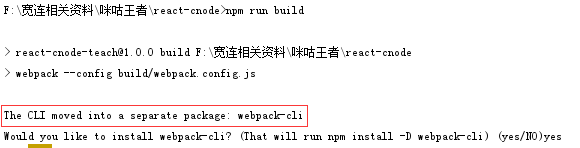


也就是这两条命令：

git remote add origin https://github.com/bwyfed/reactjs-learn.git

git push -u origin master

Webpack命令行现在已经是单独的命令行了：



因此，升级webpack-cli，从



升级为

为什么会有服务端渲染？

单页应用存在的问题：SEO不友好，首次请求等待时间较长，体验不好。

React中如何使用服务端渲染？

react-dom是React专门为web端开发的渲染工具。我们可以在客户端使用react-dom的render方法渲染组件，而在服务端，react-dom/server提供我们将react组件渲染成HTML的方法。

项目开发时常用的配置

常用配置

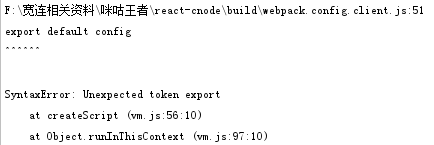
webpack-dev-server

Hot module replacement

注意在webpack配置文件中，要导出配置的话，不要使用export default:



启动时会出现报错：



应该使用module.exports = config才可以。

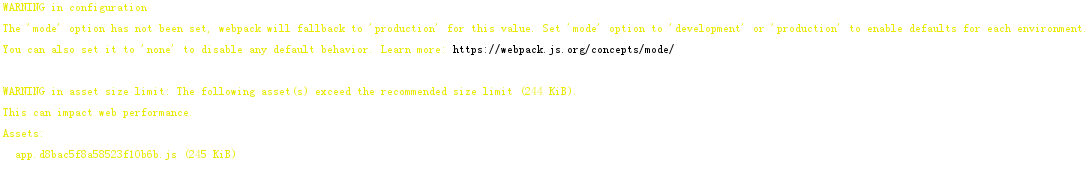
在进行如下设置：



启动脚本npm run dev:client

"dev:client": "cross-env NODE\_EVN=development webpack-dev-server --config build/webpack.config.client.js"

发现报错：



WARNING in configuration

The 'mode' option has not been set, webpack will fallback to 'production' for this value. Set 'mode' option to 'development' or 'production' to enable defaults for each environment.

You can also set it to 'none' to disable any default behavior. Learn more: <https://webpack.js.org/concepts/mode/>

后来发现即使设置了mode:’none’，还是启动不了。经过研究发现，在配置文件中：

**const** isDev = process.env.**NODE\_ENV** === **'development'**

这个isDev的值是false：



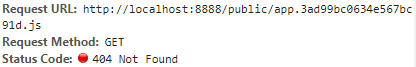
原来，是在npm脚本时输入错误了，导致无法进入了if分支：



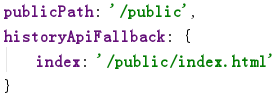
手误输错。

注意：这里要保留mode:’none’的配置。这个和视频里面的不一样。

启动之后，发现首页的index.html是可以正常访问的，但是因为js前面加了个前缀/public，访问不了：

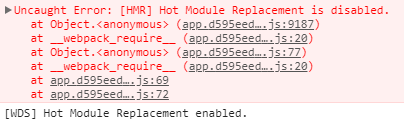


那么如何解决这个问题呢？



但是这里发现仍然是404 Not Found，如上所示。原因是首先检测硬盘上有没有这个目录，现在硬盘上的没有对应的js编译版本，所以会找不到，因此要将dist目录删掉。

删除后，发现js能正常访问了，但是js出现报错：



因为设置了hot:true，但是react并没有配置一些相关的模块，因此这里给hot:true去掉。重启后，发现有js报错：

