目录

[基础知识 3](#_Toc501466766)

[常用工具 3](#_Toc501466767)

[React Developer tool 3](#_Toc501466768)

[Windows常用对象 3](#_Toc501466769)

[相对路径和绝对路径 4](#_Toc501466770)

[常用命令 4](#_Toc501466771)

[导入项目 4](#_Toc501466772)

[Html5+js+css 4](#_Toc501466773)

[Html5 4](#_Toc501466774)

[常用 5](#_Toc501466775)

[Js 5](#_Toc501466776)

[Css 5](#_Toc501466777)

[Emmet语法 5](#_Toc501466778)

[基本语法 5](#_Toc501466779)

[Css 7](#_Toc501466780)

[<a> 7](#_Toc501466781)

[背景图片 8](#_Toc501466782)

[背景颜色 8](#_Toc501466783)

[快捷键 8](#_Toc501466784)

[day01 经典网址 8](#_Toc501466785)

[无解的依赖失败 不影响使用！！！ 9](#_Toc501466786)

[git 11](#_Toc501466787)

[clone 11](#_Toc501466788)

[day02 11](#_Toc501466789)

[Cmd指令 12](#_Toc501466790)

[创建文件夹 12](#_Toc501466791)

[Echo 13](#_Toc501466792)

[终止程序 ctrl+c 13](#_Toc501466793)

[jspm(有问题) 13](#_Toc501466794)

[12.9 14](#_Toc501466795)

[12.10 14](#_Toc501466796)

[Webpack 15](#_Toc501466797)

[配置文件 15](#_Toc501466798)

[创建 16](#_Toc501466799)

[Init 16](#_Toc501466800)

[在项目中再安装一次webpack 16](#_Toc501466801)

[创建配置文件webpack.config.js 17](#_Toc501466802)

[Bundle 17](#_Toc501466803)

[Loader 18](#_Toc501466804)

[新建webpack.config.js 配置文件 18](#_Toc501466805)

[Sourcemap 代码调试工具 19](#_Toc501466806)

[Babel 19](#_Toc501466807)

[Webpack-dev-server 20](#_Toc501466808)

[结合react 21](#_Toc501466809)

[重新打开后不需要重新安装依赖 21](#_Toc501466810)

[常见错误 22](#_Toc501466811)

[没有.babelrc文件 22](#_Toc501466812)

[找不到节点 23](#_Toc501466813)

[getInitalState 23](#_Toc501466814)

[事件 23](#_Toc501466815)

[通用属性 23](#_Toc501466816)

[不同事件的属性 24](#_Toc501466817)

[鼠标滚轮滚动实例 27](#_Toc501466818)

[.bind 27](#_Toc501466819)

[React创建 28](#_Toc501466820)

[静态react 不要用360浏览器!!! 28](#_Toc501466821)

[Router 29](#_Toc501466822)

[配置 29](#_Toc501466823)

[React代码 31](#_Toc501466824)

[组件三种写法 31](#_Toc501466825)

[无状态函数式组件 31](#_Toc501466826)

[浏览器写法 33](#_Toc501466827)

[Yeoman 34](#_Toc501466828)

[安装构建 34](#_Toc501466829)

[总结 34](#_Toc501466830)

[数据传递 34](#_Toc501466831)

[prop传递 34](#_Toc501466832)

[next相关 35](#_Toc501466833)

[redux 35](#_Toc501466834)

[常见错误 35](#_Toc501466835)

[NODE\_ENV 35](#_Toc501466836)

[toSubmit={this.toSubmit1()}立即执行 35](#_Toc501466837)

[组件名首字母必须大写！！！ 35](#_Toc501466838)

react学习笔记

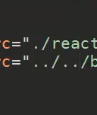
网址[https://facebook.github.io/react/docs/installation.html#trying-out-react](https://facebook.github.io/react/docs/installation.html" \l "trying-out-react)

# 基础知识

## 常用工具

### React Developer tool

在谷歌浏览器里面添加插件



当前路径和上一级路径

Windows常用对象

<body><SCRIPT LANGUAGE="JavaScript">

var  s = "";

s += "\r\n网页可见区域宽："+ document.body.clientWidth;

s += "\r\n网页可见区域高："+ document.body.clientHeight;

s += "\r\n网页可见区域宽："+ document.body.offsetWidth  +" (包括边线和滚动条的宽)";

s += "\r\n网页可见区域高："+ document.body.offsetHeight +" (包括边线的宽)";

s += "\r\n网页正文全文宽："+ document.body.scrollWidth;

s += "\r\n网页正文全文高："+ document.body.scrollHeight;

s += "\r\n网页被卷去的高："+ document.body.scrollTop;

s += "\r\n网页被卷去的左："+ document.body.scrollLeft;

s += "\r\n网页正文部分上："+ window.screenTop;

s += "\r\n网页正文部分左："+ window.screenLeft;

s += "\r\n屏幕分辨率的高："+ window.screen.height;

s += "\r\n屏幕分辨率的宽："+ window.screen.width;

s += "\r\n屏幕可用工作区高度："+ window.screen.availHeight;

s += "\r\n屏幕可用工作区宽度："+ window.screen.availWidth;

s += "\r\n你的屏幕设置是 "+ window.screen.colorDepth +" 位彩色";

s += "\r\n你的屏幕设置 "+ window.screen.deviceXDPI +" 像素/英寸";

alert(s);

</SCRIPT>

相对路径和绝对路径

<h3>1.html的相对路径</h3>

<a href=*"./demo5"*>demo5</a>

<a href=*"demo5"*>demo5</a>

<h3>1.html的绝对路径</h3>

<a href=*"http://localhost/day09/demo5"*>demo5</a>

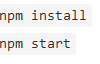
<a href=*"/day09/demo5"*>demo5</a>

## 常用命令

**查看所有安装过的模块**

**npm list -g --depth 0**

## 导入项目



# Html5+js+css

Html5



第一行：html5才有的头

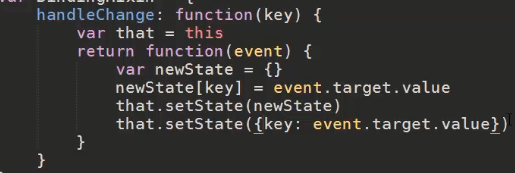
第二行：语言 zh-cn表示中文

常用

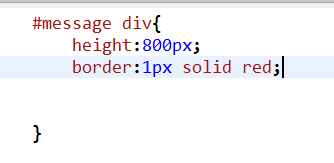
**&nbsp； <br><br>换行**

Js

Es5的缺陷



# Css



不是并列，而是message下面的div

# Emmet语法

基本语法



然后按tab键生成









# Css

<a>

链接失效





背景图片

首先在Body中加入下面的代码：

代码如下：

<div id="div1"><img src="img.jpg" /></div>

然后加入CSS代码：

div#div1{

position:fixed;

top:0;

left:0;

bottom:0;

right:0;

z-index:-1;

}

div#div1 > img {

height:100%;

width:100%;

border:0;

}

背景颜色

错误



正确

**<body style="background:#ff0000;">**

快捷键

**html:5 生成html页面**

day01 经典网址

先安装node.js，里面有npm工具，使用cmd，npm是专门用来安装各种包的

npm install -g create-react-app 安装react

create-react-app hello-world 创建项目

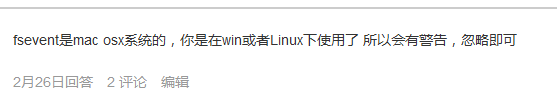
cd hello-world 进入项目目录

npm start 运行 <http://localhost:3000/>

npm install --save react react-dom 安装react-dom --save 命令用于将包添加至 package.json 文件

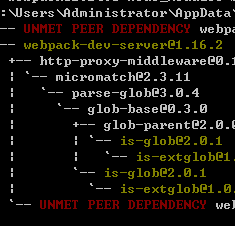
$ npm install babel -g

$ npm install webpack –g



$ npm install webpack-dev-server –g 安装各种工具

# 无解的依赖失败 不影响使用！！！



day02

margin 外边距

padding 内边距

h1 {

border-bottom: 1px solid #ddd;

font-size: 2.5em;

font-weight: bold;

margin: 30px 0 15px ;上右下左

padding: 0;

}

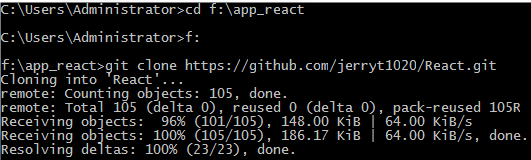
em：

任意浏览器的默认字体高都是16px。所有未经调整的浏览器都符合: 1em=16px。那么12px=0.75em,10px=0.625em。为了简化font-size的换算，需要在css中的body选择器中声明 Font-size=62.5%，这就使em值变为 16px\*62.5%=10px, 这样12px=1.2em, 10px=1em, 也就是说只需要将你的原来的px数值除以10，然后换上em作为单位就行了。  
　　**em有如下特点：**  
　　1. em的值并不是固定的;  
　　2. em会继承父级元素的字体大小。  
　　**所以我们在写CSS的时候，需要注意两点：**  
　　1. body选择器中声明Font-size=62.5%;  
　　2. 将你的原来的px数值除以10，然后换上em作为单位;  
　　3. 重新计算那些被放大的字体的em数值。避免字体大小的重复声明。  
　　也就是避免1.2 \* 1.2= 1.44的现象。比如说你在#content中声明了字体大小为1.2em，那么在声明p的字体大小时就只能是1em，而不是1.2em, 因为此em非彼em，它因继承#content的字体高而变为了1em=12px。  
　　但是12px汉字例外，就是由以上方法得到的12px(1.2em)大小的汉字在IE中并不等于直接用12px定义的字体大小，而是稍大一点。这个问 题 Jorux已经解决，只需在body选择器中把62.5%换成63%就能正常显示了。原因可能是IE处理汉字时，对于浮点的取值精确度有限。不知道有没有 其他的解释。

# git

cd  /f/dss

clone

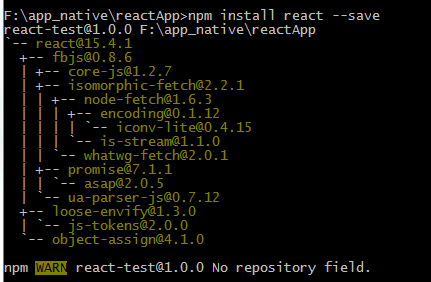


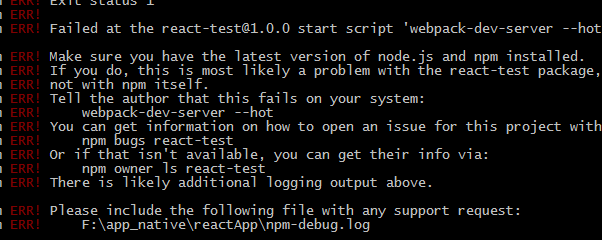
# day02

创建项目

file:///F:/编程/react学习笔记/React%20安装%20\_%20菜鸟教程.htm

从创建项目开始





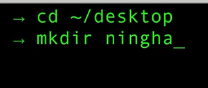
问题：nodejs x64位自带的bug，使用32可以解决？

注解：$ npm install webpack-dev-server --save-dev 最后两个没有空格

注解：cmd 返回上一级菜单

# Cmd指令

创建文件夹



cd ..是返回上一级目录，cd ...是返回上上级目录 cd \ 是返回根目录  
这些都是用的相对路径  
也可以用绝对路径  
例如当前路径是 c:\abc\bcd\efg  
你输入cd .. 则当前路径是c:\abc\bcd  
你输入cd c:\abc\bcd 也可以

**如果是f盘，还要点f：进入f盘**



Echo

## 终止程序 ctrl+c

# jspm(有问题)

安装 npm install jspm –g

配置 在建好的项目下（可以只见一个package.json）

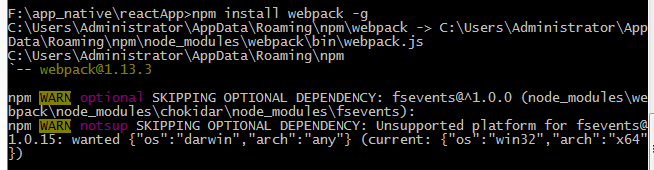
npm install jspm –save –dev

**注释semantic ui**

npm install -g gulp

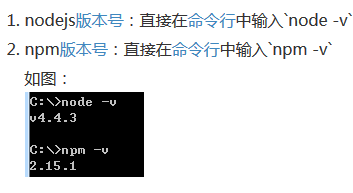
npm install semantic-ui –save

webpack默认版本



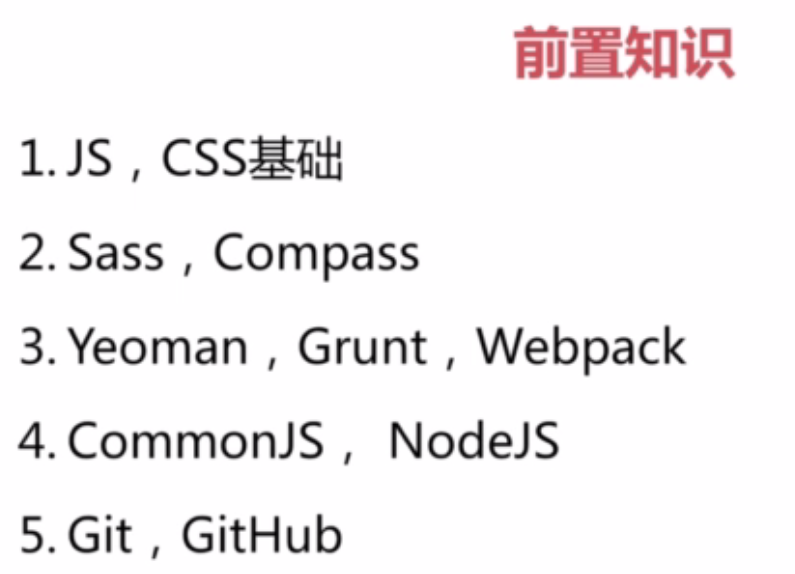


# 12.9



# 12.10

前置知识



Webpack

## 配置文件

entry:打包的入口文件，一个字符串或者一个对象

　　　　output:配置打包的结果，一个对象

　　　　　　fileName：定义输出文件名，一个字符串

　　　　　　path：定义输出文件路径，一个字符串

　　　　module:定义对模块的处理逻辑，一个对象

　　　　　　loaders：定义一系列的加载器，一个数组

　　　　　　　　[

　　　　　　　　　　{

　　　　　　　　　　　　test:正则表达式，用于匹配到的文件

　　　　　　　　　　　　loader/loaders：字符串或者数组，处理匹配到的文件。如果只需要用到一个模块加载器则使用　　　　　　　　　　　　　　　　　loader：string，如果要使用多个模块加载器，则使用loaders：array

　　　　　　　　　　　　include:字符串或者数组，指包含的文件夹

　　　　　　　　　　　　exclude：字符串或者数组，指排除的文件夹

　　　　　　　　　　}

　　　　　　　　]

　　　　resolve:影响对模块的解析，一个对象

　　　　　　extensions：自动补全识别后缀，是一个数组

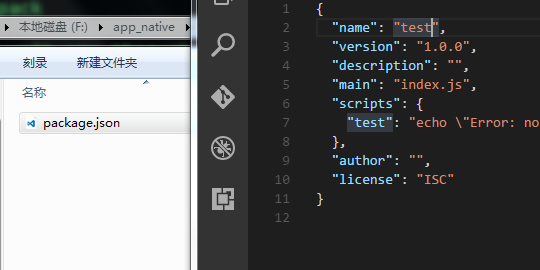
　　　　plugins:定义插件，一个数组

**创建**

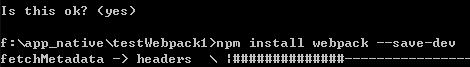
### Init



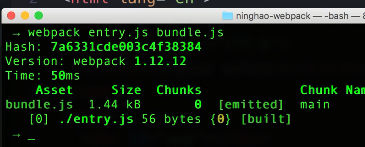
效果



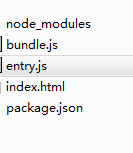
### 在项目中再安装一次webpack



打包（手动打包到指定文件中）（实际不需要手动）



生成效果



### 创建配置文件webpack.config.js

**注意安装对应的文件，babel，loader,react,redux,根据实际情况**

var webpack = require('webpack');

var path = require('path');

module.exports = {

entry: [

'./app/index.js'

],

output: {

path: path.join(\_\_dirname, 'dist'),

publicPath: '/',

filename: './bundle.js'

},

module: {

loaders:[

**{ test: /\.css$/, include: \_\_dirname, loader: 'style-loader!css-loader' },**

**{test:[/\.js$/,/\.jsx?$/],exclude:/(node\_modules)/,loader:'react-hot!babel'},**

]

},

resolve: {

extensions: ['', '.js', '.jsx'],

}

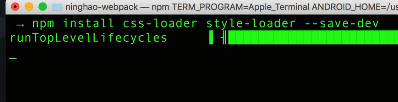
};

Bundle

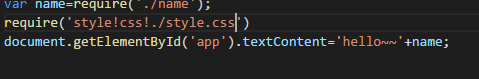
将项目分成不同部分

## Loader

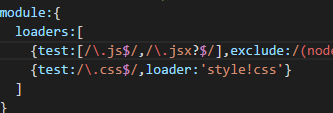
Webpack默认只会处理js相关的内容，使用其他内容需要对应的loader，一种转换器

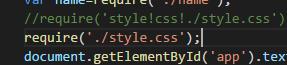


手动添加



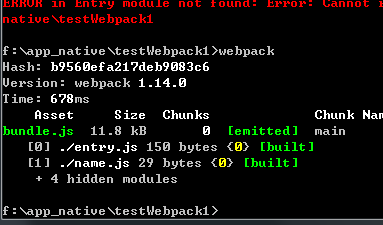
配置文件中添加





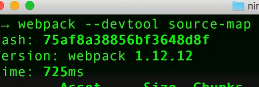
新建webpack.config.js 配置文件

添加配置文件后只需要webpack命令，不需要指定打包文件



Sourcemap 代码调试工具

不用这个工具打包后不能看见原代码（不需要安装）



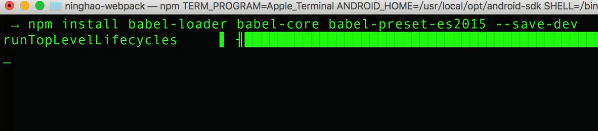
直接执行webpack即可

也可以添加到webpack配置文件中

devtool: 'source-map',

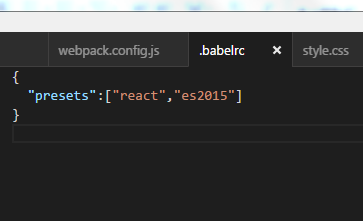


Babel



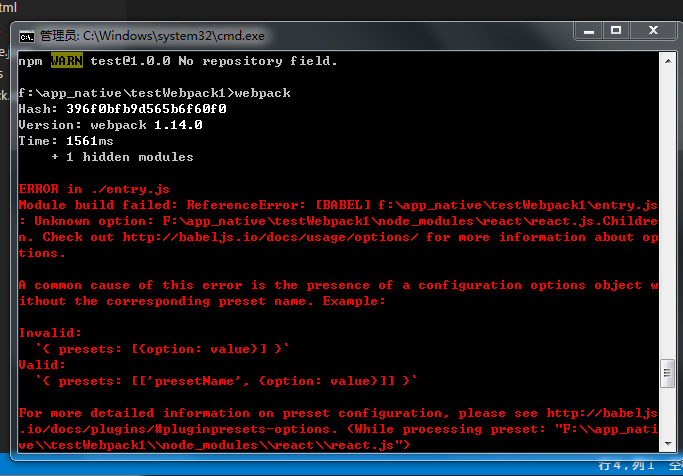
(没有安装过babel才安装babel-core)

**创建配置文件.babelrc**



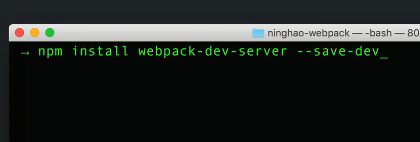
使用

**{test:[/\.js$/,/\.jsx?$/],exclude:/(node\_modules)/,loader:'babel'},**

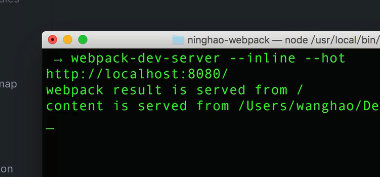
**没有安装babel-preset-react就在babelrc中使用了react**

Webpack-dev-server

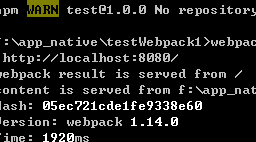
在全局中安装了以后还要在项目中安装依赖



使用

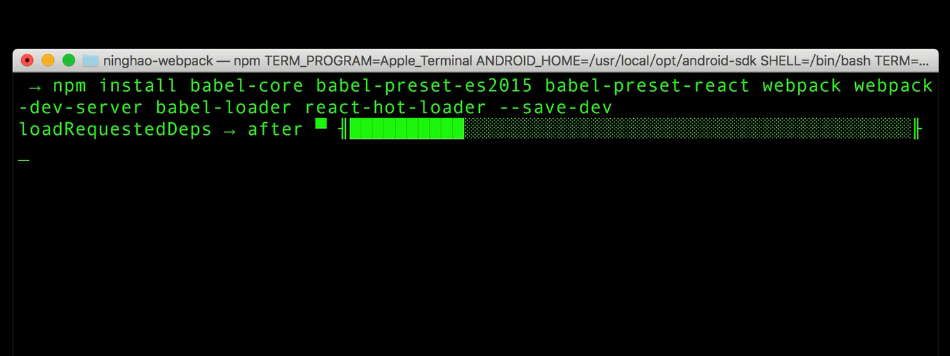


服务器地址

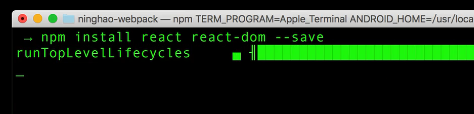


结合react

添加依赖，应该是重新打开cmd后才这样全部重复添加一遍



安装react和react dom（没有-dev）

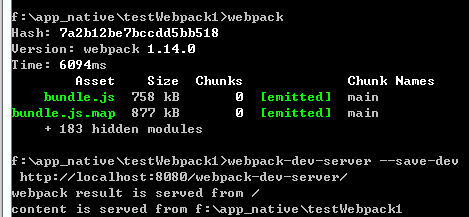


配置

**{test:[/\.js$/,/\.jsx?$/],exclude:/(node\_modules)/,loader:'react-hot!babel'},**

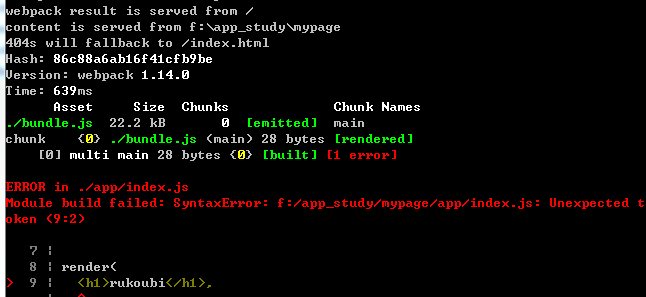
然后就可以使用了

重新打开后不需要重新安装依赖



# 常见错误

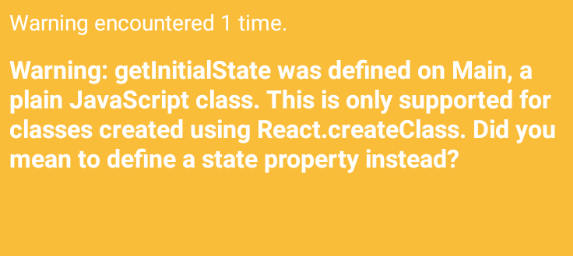
### 没有.babelrc文件



## 找不到节点



## getInitalState



事件 event（应该是一个对象）

通用属性



1事件是否可以冒泡

2事件是否可以取消

3

4是否禁止默认行为，比如submit默认为跳转页面

5事件所处的阶段

6是否可信（使用代码触发的事件就是不可信事件）

7拿到原生的即浏览器发出的event对象

8

9使用后，不会冒泡

10

11时间戳

12事件类型

不同事件的属性

剪切



键盘







45 零点浏览器窗口左上角

Page html页面左上角

Screen 显示器左上角



Changedtouches

Targettouches

Touches

滚动

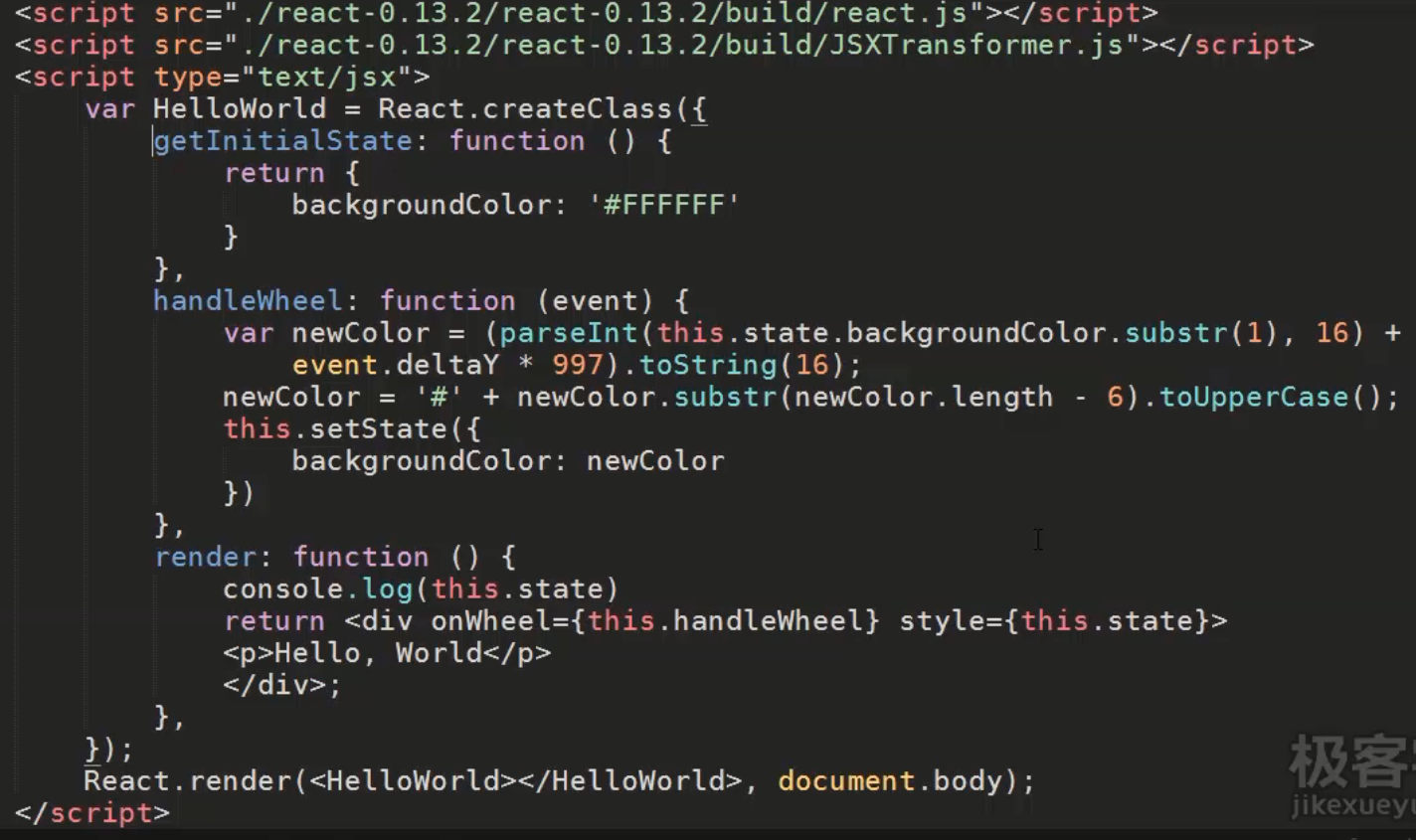


鼠标滚轮滚动



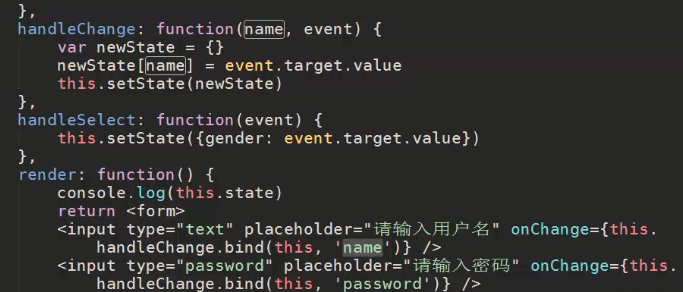
有些特殊设备可以在三个轴上移动

鼠标滚轮滚动实例



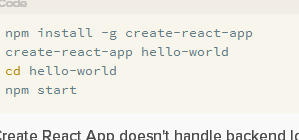
.bind

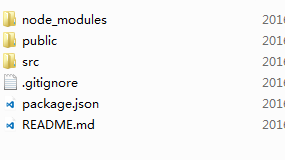
逻辑相同来源不同的函数不需要写两个



# React创建

简单方法





# 静态react 不要用360浏览器!!!

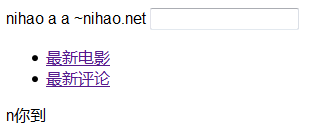
<div id="app"></div>

<div id="navbar"></div>

n你到

<script type="text/babel" src="nav.js"></script>

<script type="text/babel" src="entry.js"></script>



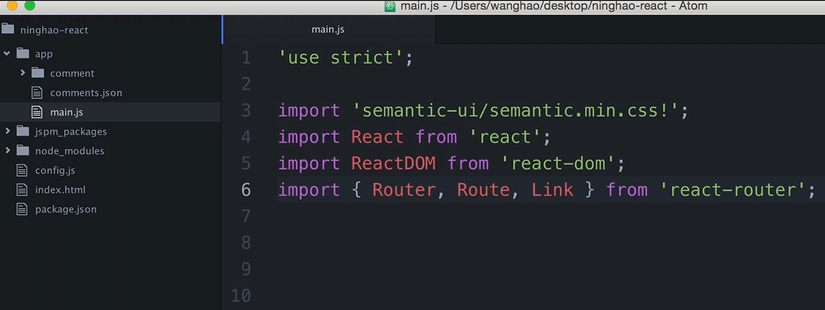
# Router

配置

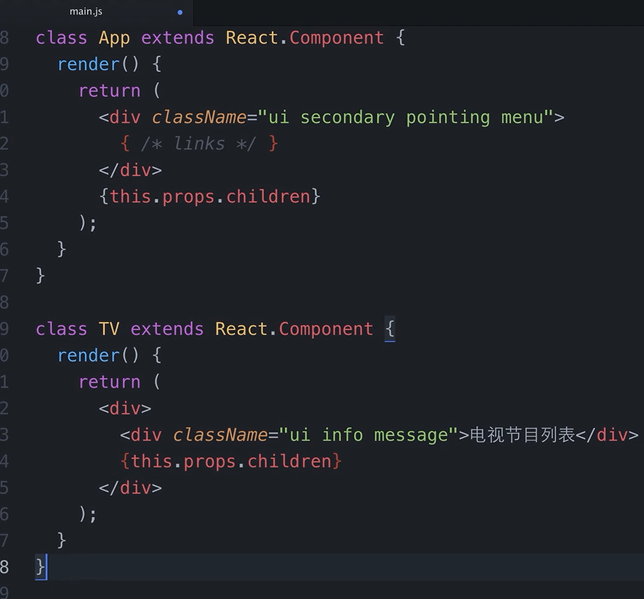
安装

$ npm install -S react-router

**导入**



**使用**



# React代码

## 组件三种写法



## 无状态函数式组件

创建[无状态函数式组件形式](https://facebook.github.io/react/blog/2015/10/07/react-v0.14.html" \l "stateless-functional-components" \t "_blank)是从React 0.14版本开始出现的。它是为了创建纯展示组件，这种组件只负责根据传入的props来展示，不涉及到要state状态的操作。具体的无状态函数式组件，其官方指出：

在大部分React代码中，大多数组件被写成无状态的组件，通过简单组合可以构建成其他的组件等；这种通过多个简单然后合并成一个大应用的设计模式被提倡。

无状态函数式组件形式上表现为一个只带有一个render方法的组件类，通过函数形式或者ES6 arrow function的形式在创建，并且该组件是无state状态的。具体的创建形式如下：

**function HelloComponent(props, /\* context \*/) {**

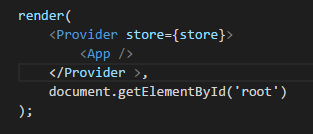
**return <div>Hello {props.name}</div>**

**}**

**ReactDOM.render(<HelloComponent name="Sebastian" />, mountNode)**

箭头函数写法，还是要用到render





无状态组件的创建形式使代码的可读性更好，并且减少了大量冗余的代码，精简至只有一个render方法，大大的增强了编写一个组件的便利，除此之外无状态组件还有以下几个显著的特点：

1. **组件不会被实例化，整体渲染性能得到提升**  
   因为组件被精简成一个render方法的函数来实现的，由于是无状态组件，所以无状态组件就不会在有组件实例化的过程，无实例化过程也就不需要分配多余的内存，从而性能得到一定的提升。
2. **组件不能访问this对象**  
   无状态组件由于没有实例化过程，所以无法访问组件this中的对象，例如：this.ref、this.state等均不能访问。若想访问就不能使用这种形式来创建组件
3. **组件无法访问生命周期的方法**  
   因为无状态组件是不需要组件生命周期管理和状态管理，所以底层实现这种形式的组件时是不会实现组件的生命周期方法。所以无状态组件是不能参与组件的各个生命周期管理的。
4. **无状态组件只能访问输入的props，同样的props会得到同样的渲染结果，不会有副作用**

无状态组件被鼓励在大型项目中尽可能以简单的写法来分割原本庞大的组件，未来React也会这种面向无状态组件在譬如无意义的检查和内存分配领域进行一系列优化，所以**只要有可能，尽量使用无状态组件**。

## 浏览器写法

<script src="https://unpkg.com/react@15.3.2/dist/react.js"></script>

<script src="https://unpkg.com/react-dom@15.3.2/dist/react-dom.js"></script>

<script src="https://unpkg.com/babel-core@5.8.38/browser.min.js"></script>

<script type="text/babel">

var Menu = React.createClass({

render : function() {

return () }

});

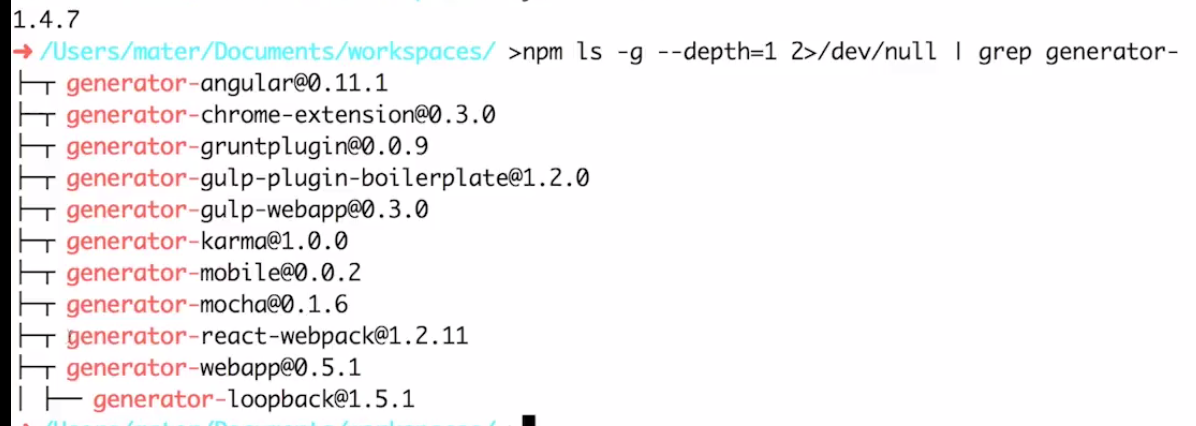
ReactDOM.render(<Menus data={data} />, document.getElementById("example"));

</script>

# Yeoman

## 安装构建





2>/dev/null

>重定向

2：错误消息

即不显示错误消息



# 总结

## 数据传递

### prop传递

传递过程只是正常赋值，所以子组件修改（赋值给state,修改state的值），可能导致父组件的属性改变。

### next相关

nextProps包括nextState只是一个对象，因为对比是浅对比，所以对比只对比到一层。

### redux

无论是挂载到组件里，还是对state进行更新，都是简单赋值。所以每次在reducer中都是返回一个新的state，但也只缓解一部分，嵌套太深依然没办法。

# 常见错误

## NODE\_ENV

Windows下不支持，添加cross-env即可

"scripts": {

"start": "cross-env NODE\_ENV=development node server.js",

"build": "cross-env NODE\_ENV=production webpack --config webpack.prod.config.js --progress --colors -p"

},

## toSubmit={this.toSubmit1()}立即执行

在render里面用时,每次渲染都会执行一遍

注意:一般是**toSubmit=**{**this**.toSubmit1},没有括号,

区分html里的监听，要括号。（和react里面不一样，react的jsx是类似js的，会编译成html）

## 组件名首字母必须大写！！！

不然渲染出来是空的