# 引出webpack

## 在网面中会引入哪些资源

**CSS**

.css .less .sass .scss

**JS**

.js .jsx .coffee .ts

**Img**

.png .jgp .svg .gif .bmp

**字体文件**

.svg .ttf .eot ....

**模板文件**

.ejs .jade .vue

## 网页中引入静态资源多了以后会有什么样的问题？

1. 网页加载速度过慢，因为我们要发起二次请求。
2. 要处理错综复杂的依赖关系

## 如何解决如上的两个问题？

1. 合并，压缩，精灵图
2. 模块化编程RequireJS
3. 使用Webpack解决

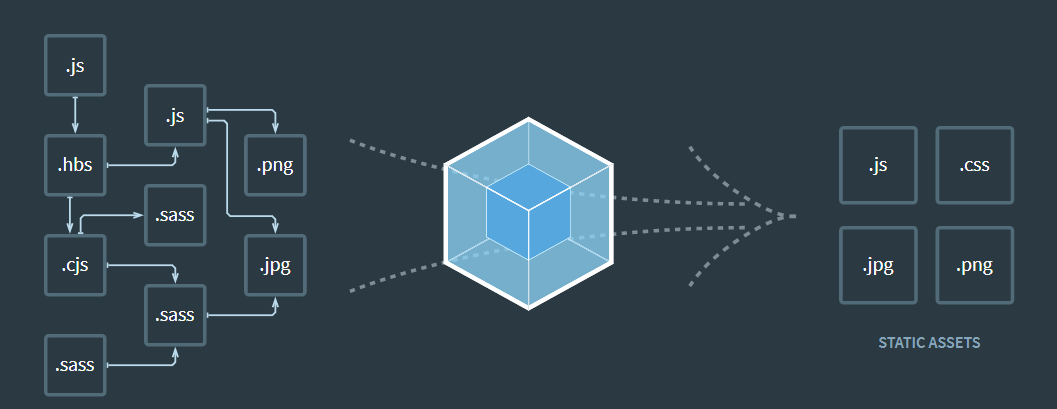
## 引出webpack

Webpack是一个资源的打包工具，是前端一个项目构建工具，它是基于nodejs的。目前有1.x，2.x 3.x这三个版本，学习时，学习3.x这个版本。

# 初识webpack

## 什么是webpack

Webpack的官网是：<https://webpack.js.org/>



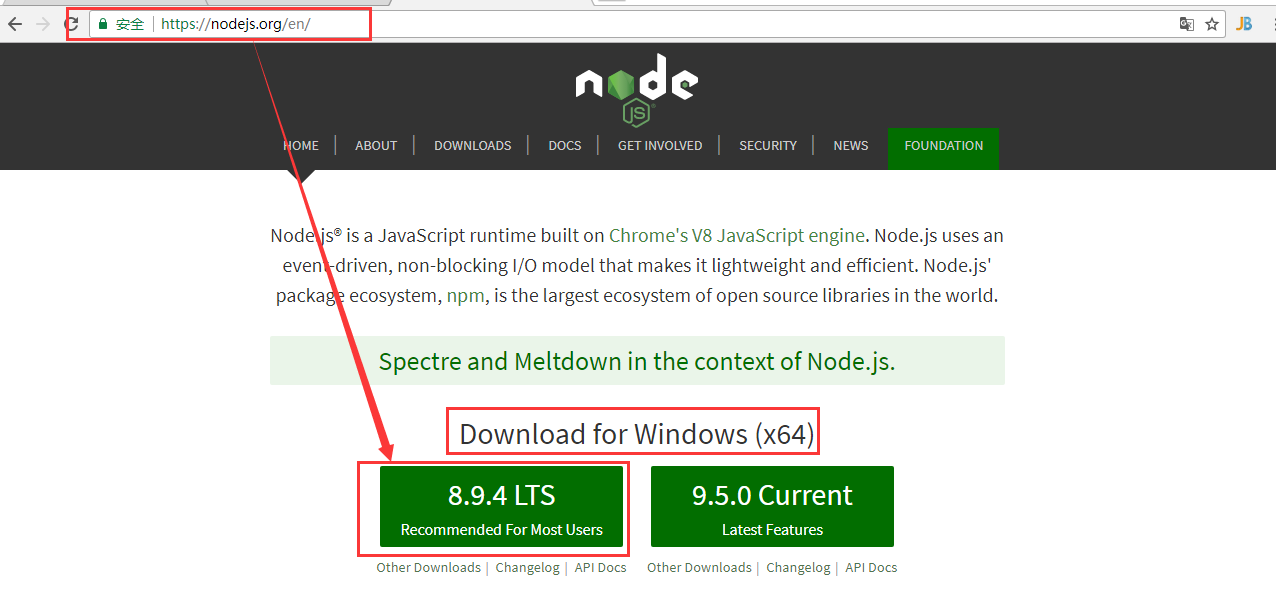
Webpack是一个资源打包工具，有三个版本，1.0 2.0 3.0 学习时学习3.0的。可以将.js .css .img等静态资源当做一个模块来进行打包处理。

## node环境安装

由于webpack是基于nodejs的，所以需要把node复习一下。

安装nodejs。

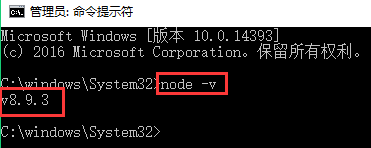
**第一步：去官网下载nodejs**



下载后，会得到一个安装包，拿到这个安装包后，双击安装，一路next就OK了。、

说明：安装完nodejs后，会自动地安装一个工具，叫npm工具，可以通过npm这个工具安装各种依赖了。

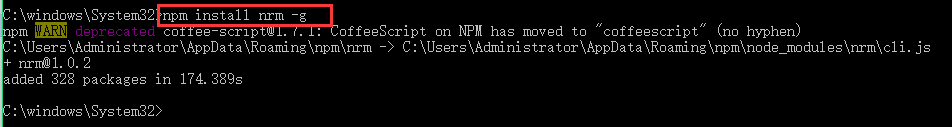
测试nodejs是否安装OK了，如下：



## 学习一个各种命令

通过npm install安装的各种包是去国外的网站上下载，有可能下载失败或被墙。所以我们有时候会去切换下载源，如何做？

**第一步：安装nrm这个工具，如下：**



**nrm的作用：**提供了一些最常用的npm包镜像地址，能够让我们快速的切换安装包的服务器地址。

安装完后，可以通过**nrm ls**查看都有哪些镜像源，如下：

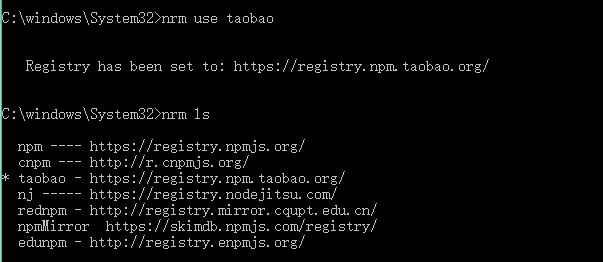


\*号表示当前我们的下载源是哪一个，我们当前的下载源是taobao。

把下载源切换到npm，如下：



我们一般都使用taobao的下载源，因为是在国内，速度会快一点，如下：



## 安装webpack

**有两种方式：**

1. 使用npm i webpack -g全局安装webpack，这样就能在全局使用webpack的命令了
2. 在项目根目录运行npm i webpakck --save-dev安装到项目依赖中

如下：

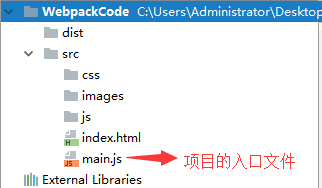


如何查看webpack是否安装成功，如下：

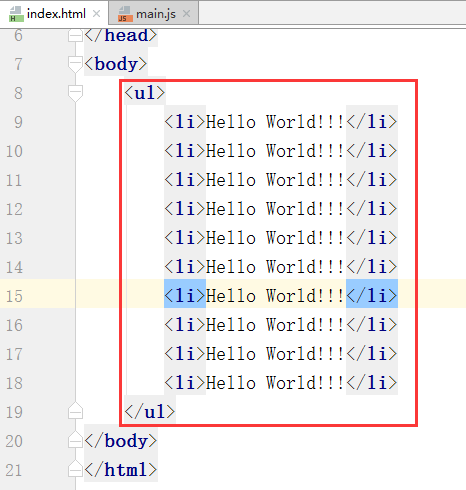


## webpack的快速入门

第一步：搭建项目的结构如下：



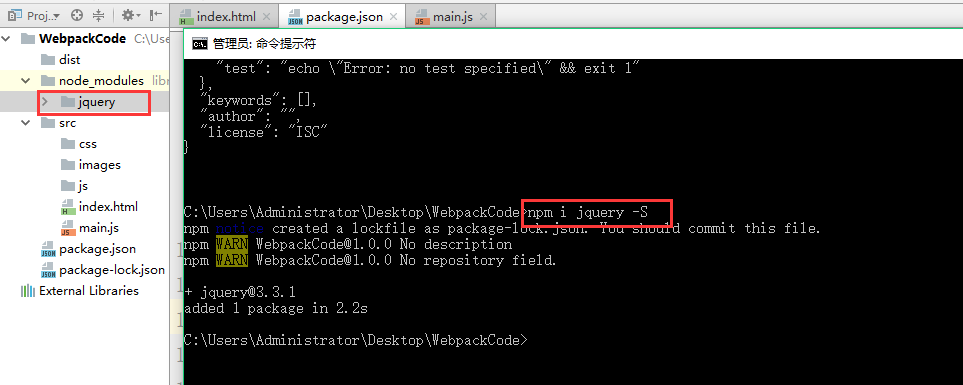
第二步，写几个li，如下：



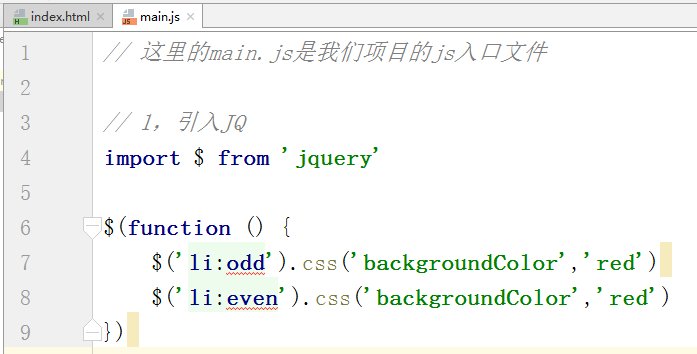
第三步，初识化项目，如下：



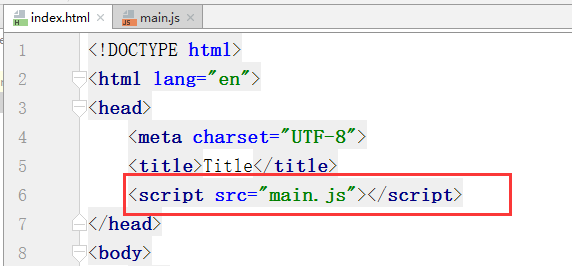
第四步，安装JQ，如下：



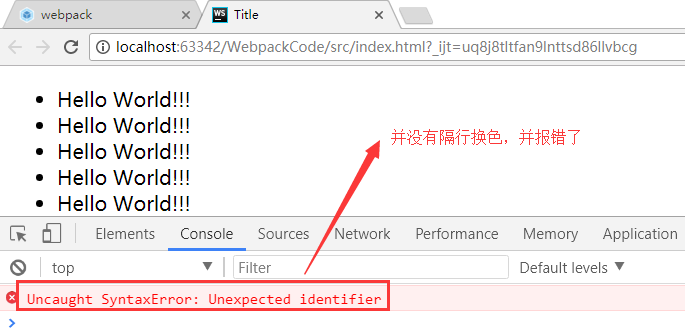
第五步，引入JQ，并写入如下的代码：



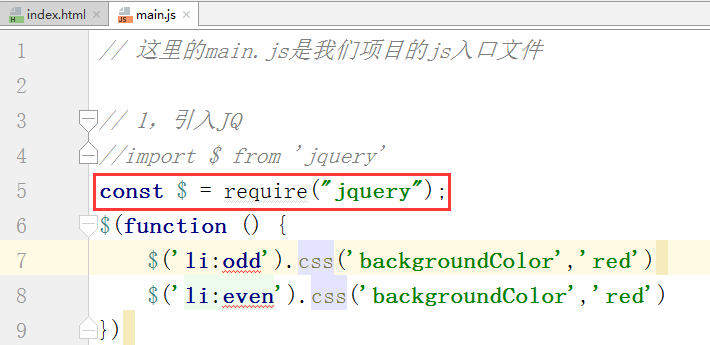
第六步，在index.html中引入main.js如下：



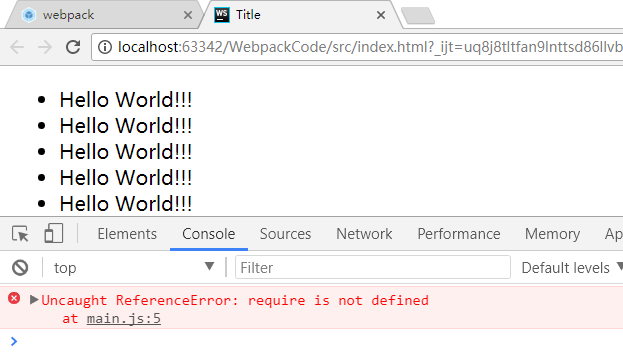
第七步，运行如下：



我们换一个require的写法，如下：

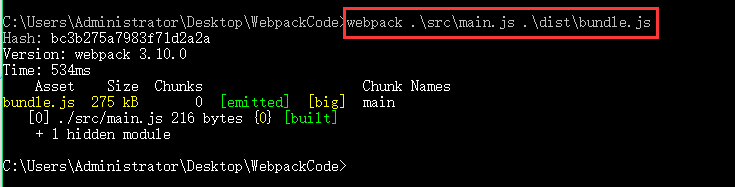


同样，也报错了，如下：



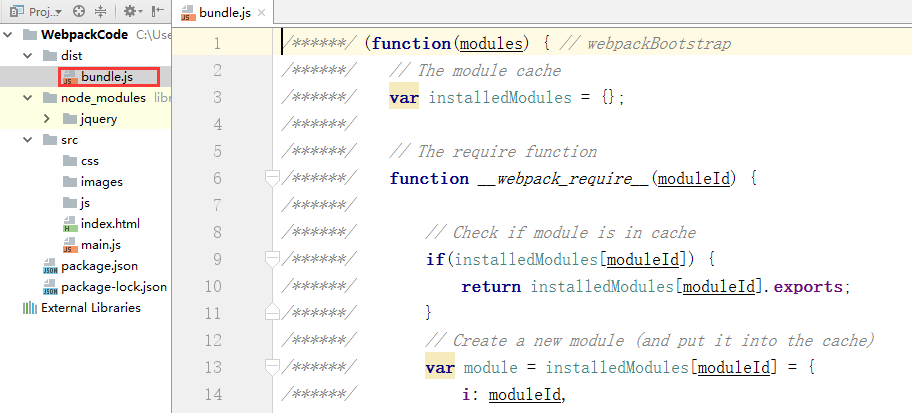
错误原因是：在mina中涉及了es6的写法，浏览器不识别。

现在我们通过webpack这么一个前端构建工具，把main.js做一个处理，生成一个bundle.js的文件，如下：



命令格式：webpack 要打包的文件路径 打包好的输出文件路径

此时就生成了bundle.js这个文件，如下：



在index.html中引入bundle.js这个文件，如下：



运行，看效果，如下：



此时就是webpack的快速入门案例。

**小结：**

**使用webpack之后，究竟做了什么？**

* webapck能够处理JS文件的相互依赖。
* Webpack能够处理JS的兼容性问题，把高级的，浏览器不识别的语法，转换成低级的，浏览器能够识别的语法。

# 深入webpack

上面打包时是通过如下命令进行的，如下：

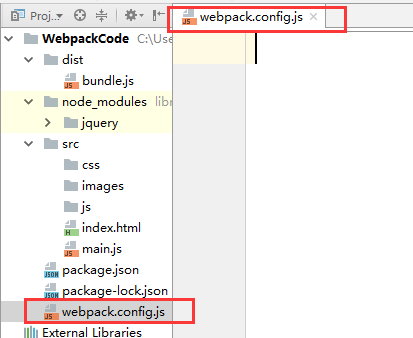


在每一次修改代码后，都要通过这个命令重新打包，否则就不能生效。

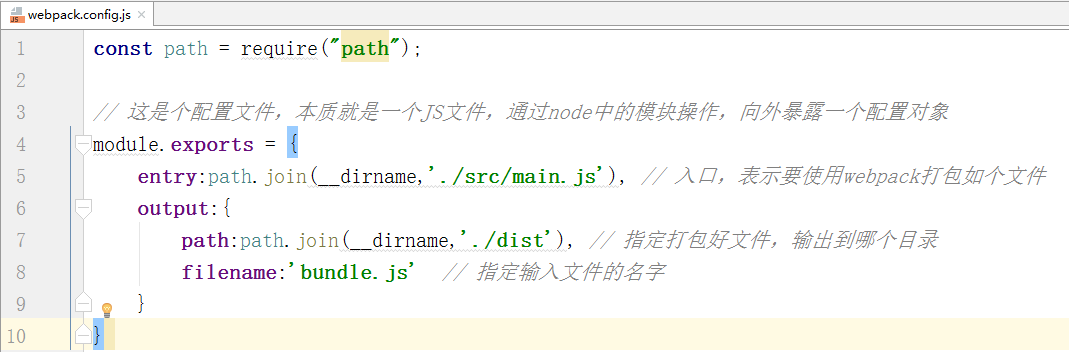
可能通过webpack配置，直接使用webpack这个命令，就可以实现打包。

## webpack的配置

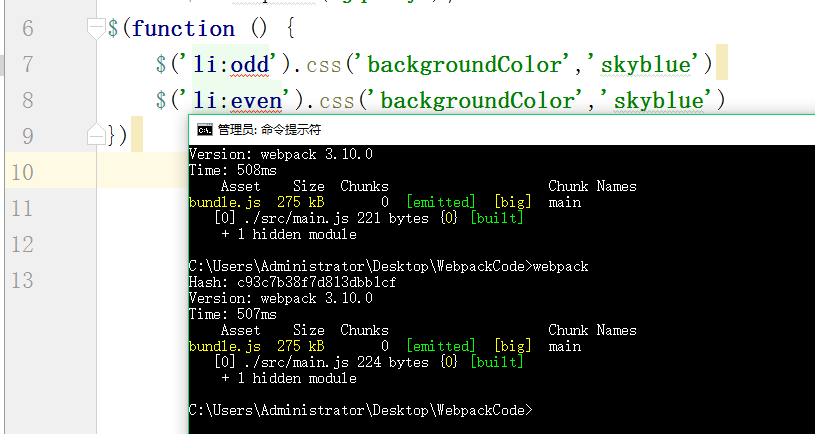
第一步：新建一个webpack.config.js这个文件，如下：



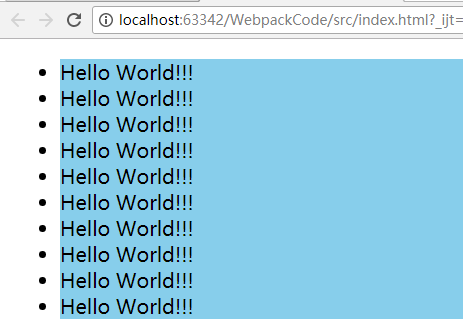
第二步，写入如下的代码，如下：



第三步，我们就可以直接通过webpack的命令实现打包了，如下：



第三步，看效果，如下：



## webpack-dev-server

能不能在修改完代码后，不需要自己手动的打包编译，让webpack帮我们做呢？

答：可以的，使用webpack-dev-server这个工具。

Webpack-dev-server这个工具是用来实现自动打包编译的。

使用步骤如下：

第一步：把这个工具安装到项目的本地开发依赖中。

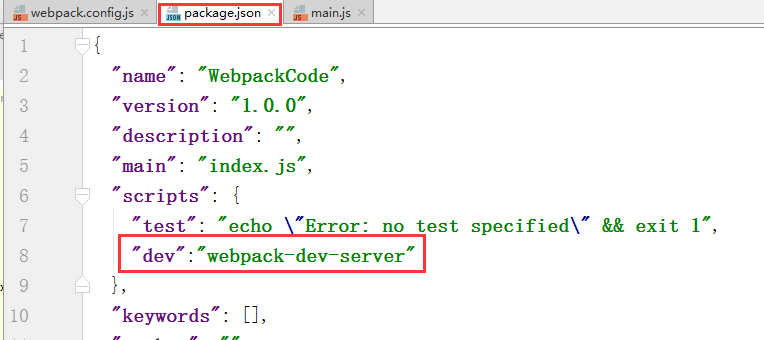


第二步，使用这个工具，如下：

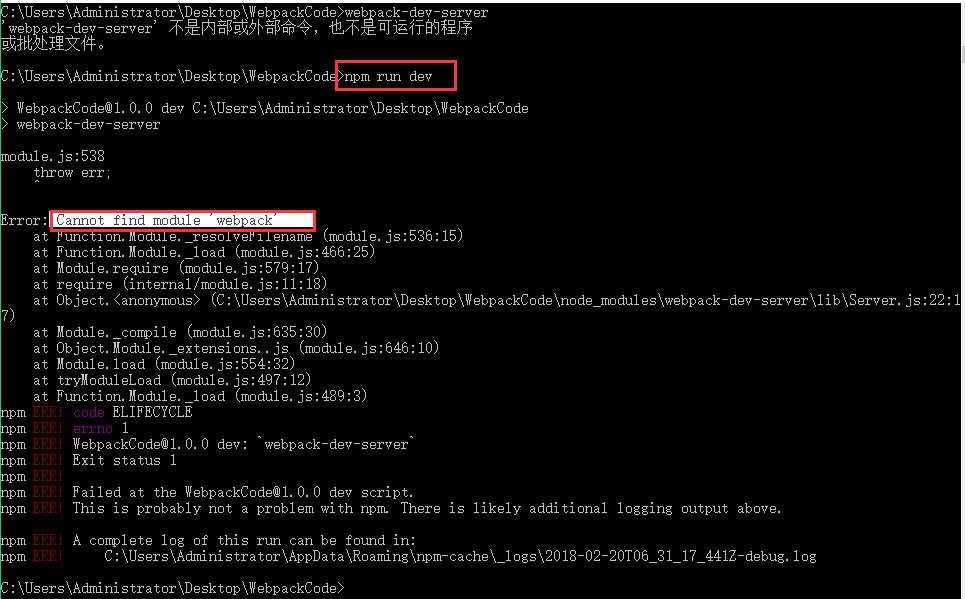


上面错误的原因：我们在项目中本地安装了webpack-dev-server，所以无法把它当作脚本命令在cmd窗口中直接使用。只有那些安装到全局中的工具，才能在终端中使用。

第三步，由于上面的不能直接使用，所以配置到package.json中，如下：



第四步，使用npm run dev 这个命令，使用这个工具，如下：

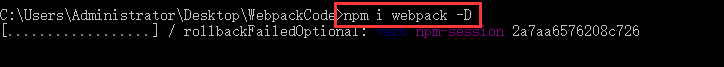


解决办法：

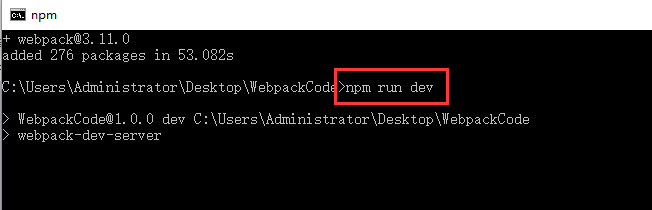
把所有的包全部删除，重装，如下：



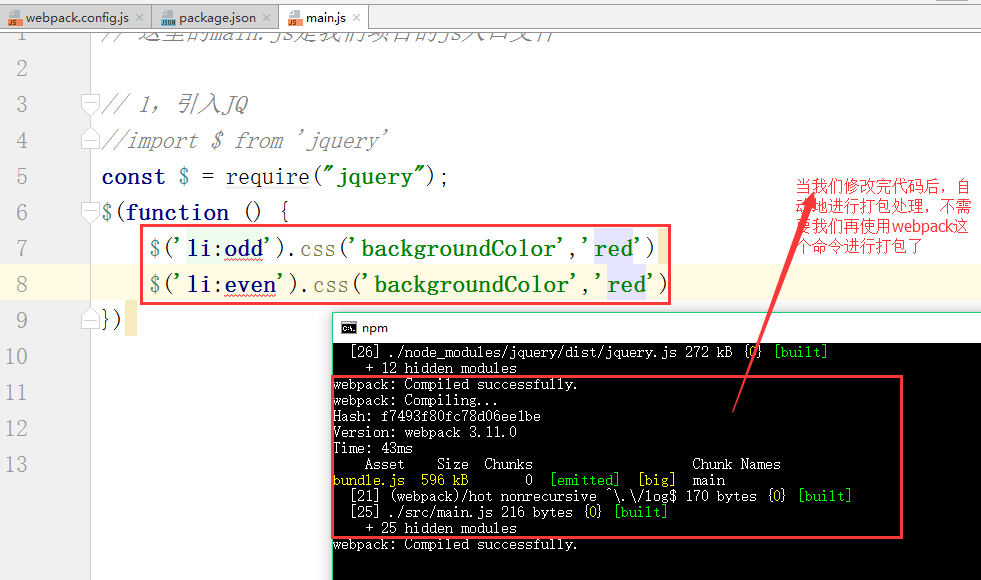
注意：在使用webpack-dev-server这个工具时，如果想要正常运行，需要本地安装webpack这个工具。



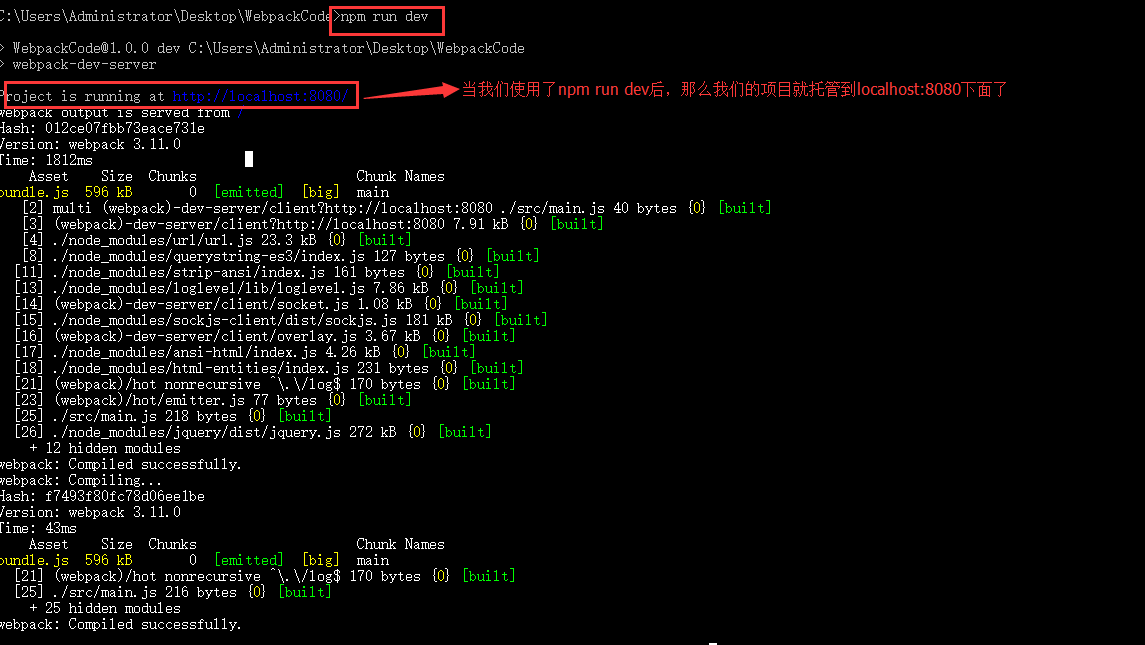
上面的安装完后，就可以使用了，如下：



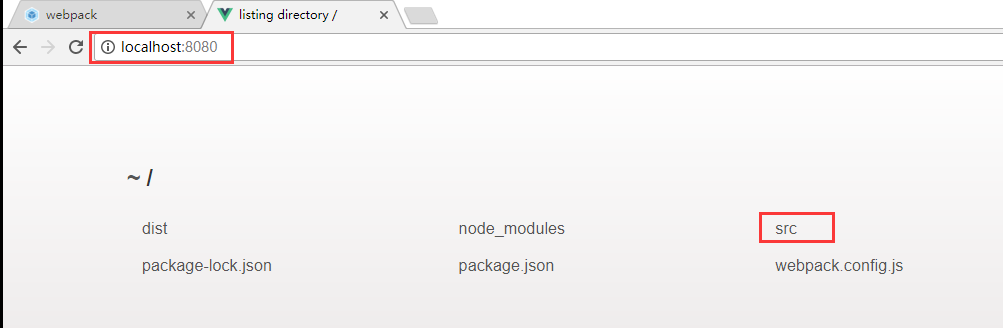
修改代码，测试如下：



使用了npm run dev后，我们就不能通过本地打开项目了，需要通过localhost打开，如下：



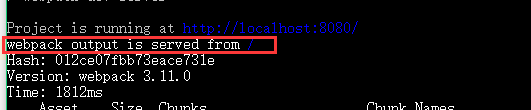
打开浏览器，如下：



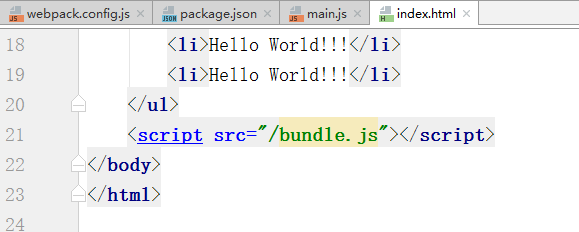
我们的index.html是放在src下面的，如下：



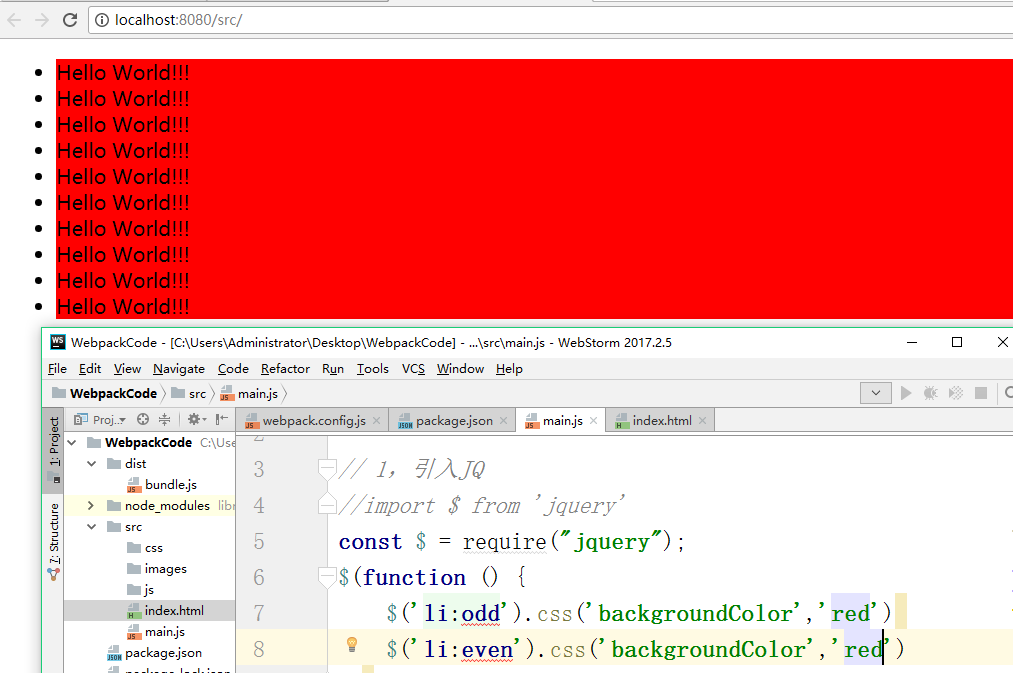
现在遇到了一个问题，我们把颜色变成了红色，但是现在的颜色是蓝色，为什么？



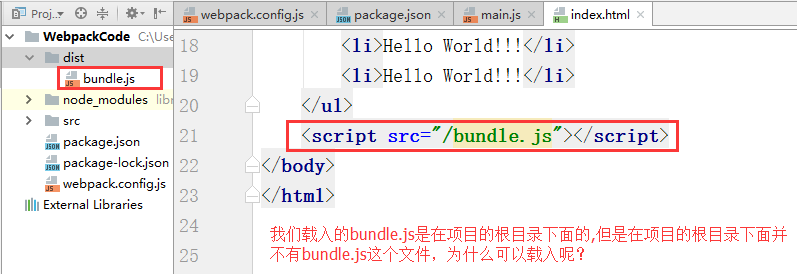
上面说了，输出是在项目的根目录下面，所以说，我们引入bundle.js时，需要从项目的根目录下面的引入，如下：



修改代码，再次打开浏览器，如下：



现在，我们需要思考一个问题？



**原因：**webpack-dev-server帮我们打包生成了bundle.js文件，并没有存放到实际的物理磁盘上，而是直接托管到电脑的内存中，所以在项目的根目录下面，根本找到不到这个打包好的bundle.js这个文件。

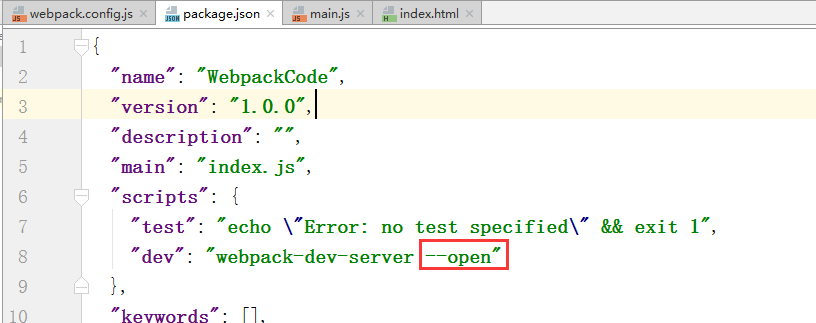
**为什么要把bundle.js托管到内存中？**

答：快。

## webpack-dev-server中常用的命令参数

### open

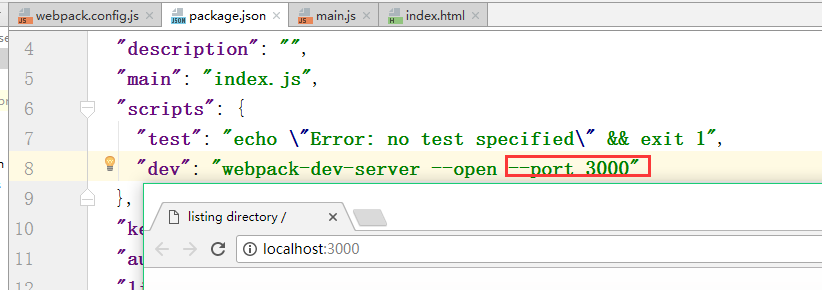
当修改完代码后，自动打开浏览器，就可以添加open这个参数，如下：



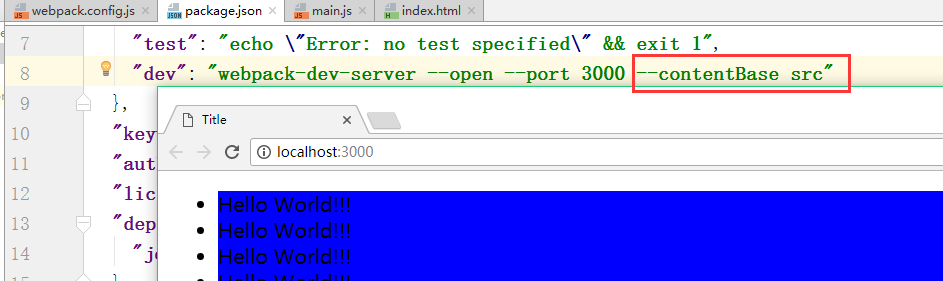
此时，浏览器会自动打开，不需要我们手动的打开了。

### port

这个参数可以用来修改端口。

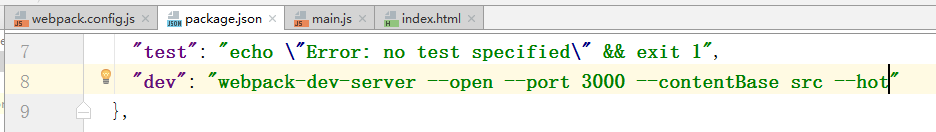


### contentBase src



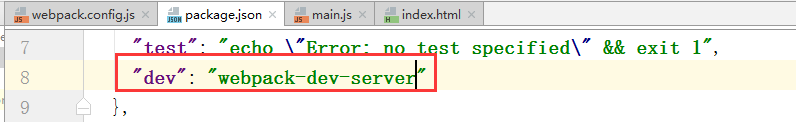
### hot

热更新可以减少不必要的代码更新，不会重新生成bundle.js，还可以使浏览器无刷新。针对样式的，对JS无效。



## 常用命令参数的第二种写法

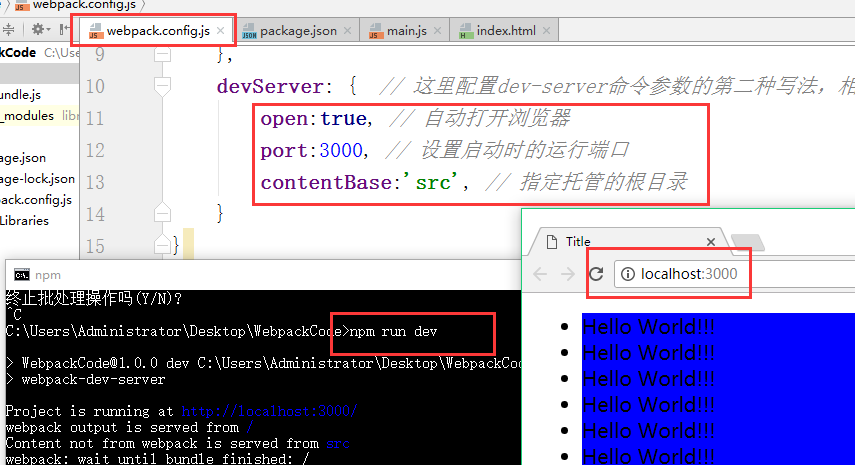
除了上面的写法外，还有一种写法，比上面的写法复杂一点。



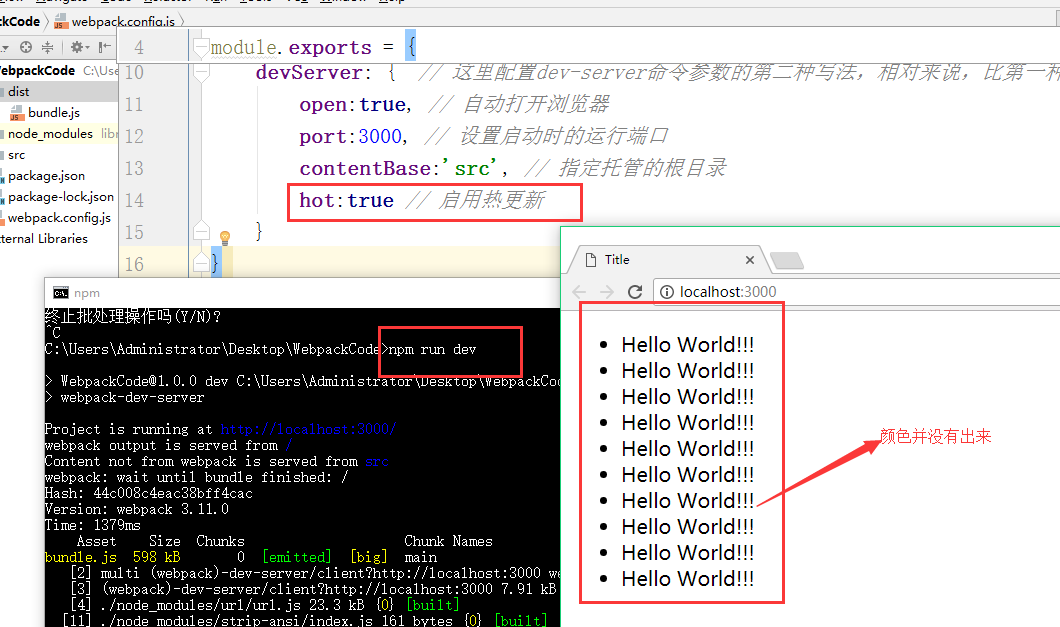
在写在webpack.config.js中，如下：



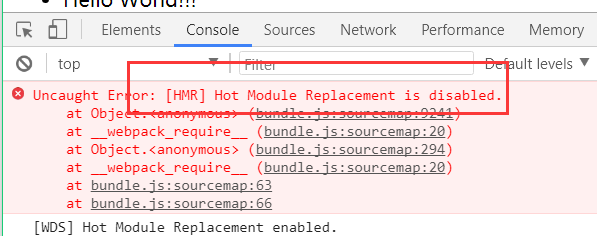
还有其它几个参数，如下：



还有一个热更新，也按照上面的写法，如下：

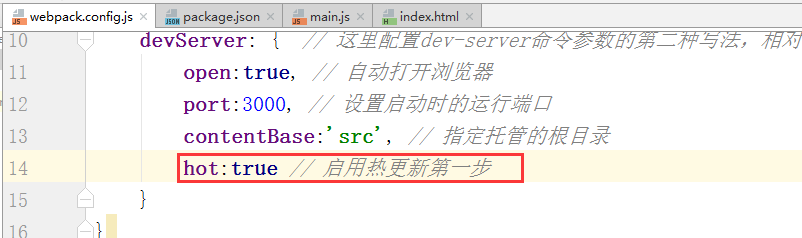


并且在浏览器中报错了，如下：

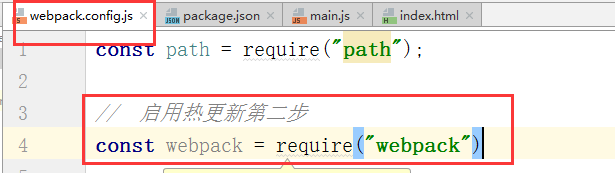


要启动热更新，需要3步，如下：

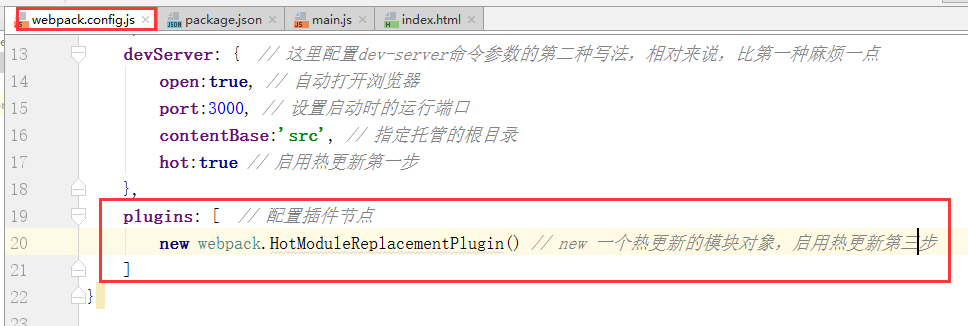
第一步：



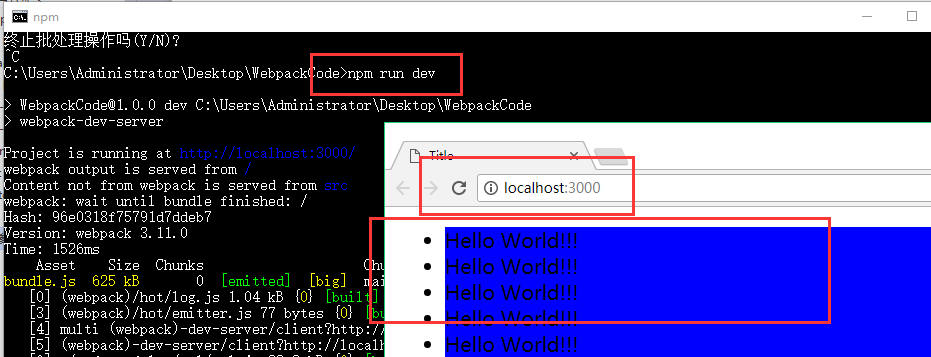
第二步：



第三步：



运行如下：



## 将index.html载入内存

我们也可以把index.html载入内存中， 使用html-webpack-plugin插件来配置。

第一步：就是安装这个插件，如下：



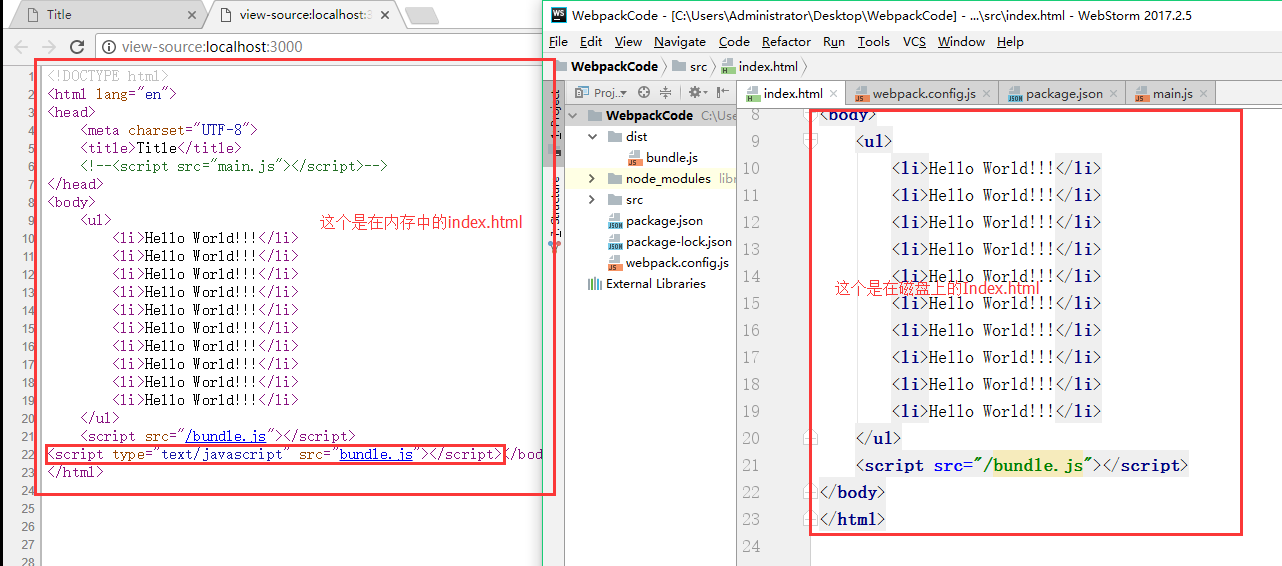
第二步，载入，如下：



第三步，new，如下：



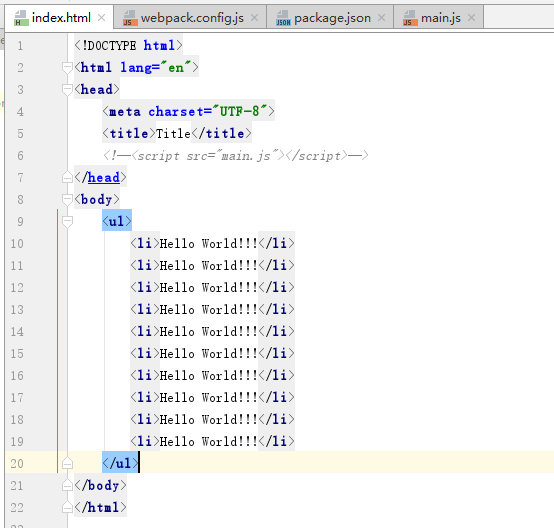
运行npm run dev，如下：



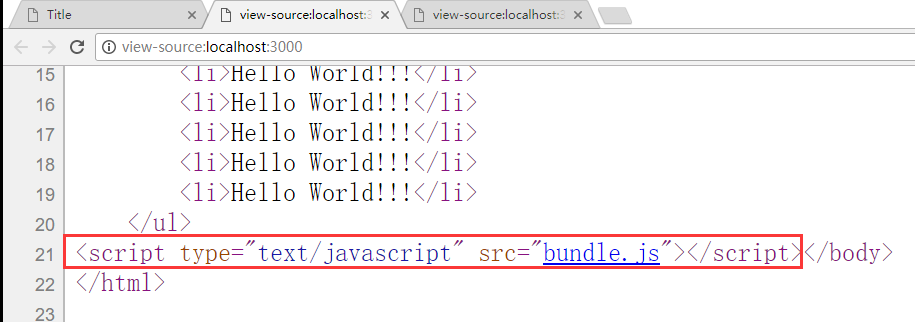
小结：

使用html-webpack-plugin，自动在内存中根据指定页面生成一个内存页面，还会自动把打包好的bundle.js追加到页面中去。所以说，在我们的页面中就不需要引入bundlejs了

在模板文件中，就不需要引入bundle.js了，如下：



但在内存生成的index.html中，会自动载入bundle.js这个文件，如下：

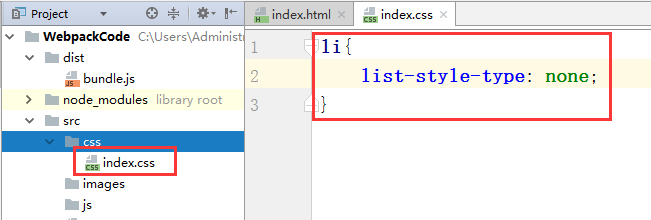


# 各种loader

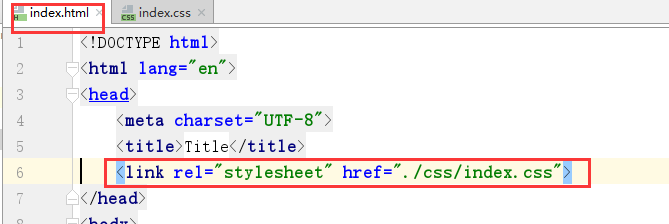
## CSS的loader

需求：把li的小圆点去掉。

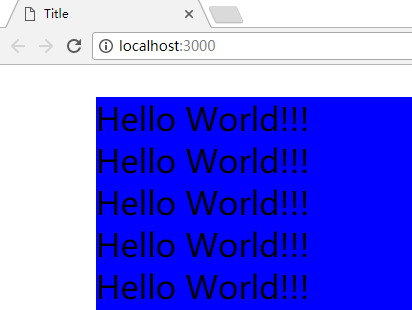
第一步：新建一个index.css文件，写入如下代码：



第二步：引入CSS，如下：



第三步，看效果，如下：

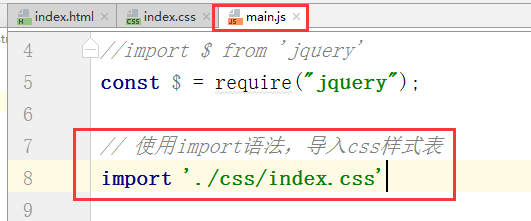


由于第二步，中引入外部了css文件，会发生二次请求，所以不能这做做，怎么办？

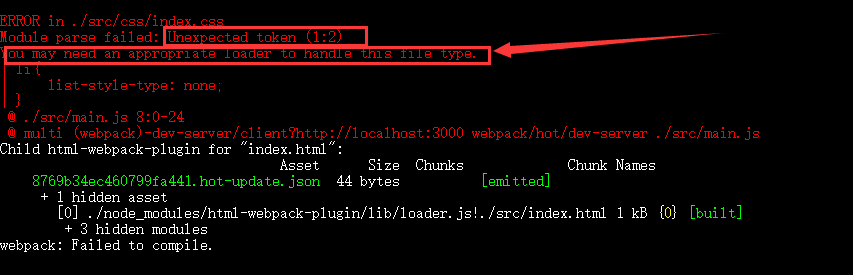
在main.js中导入css文件。

**新的做法如下：**

第一步：在main.js中使用import语法，导入css样式表，如下：



第二步：运行，如下：



**为什么导入JQ时没有报错，而导入css时报错了？**

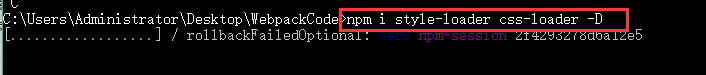
原因：webpack，默认只能打包处理js类型的文件，无法处理其它非js类型的文件。

**不能处理，怎么办？**

如果要处理非js类型的文件，需要使用第三方的loader加载器来处理。

现在，我们要处理index.css，那么我们需要安装两个包：style-loader css-loader

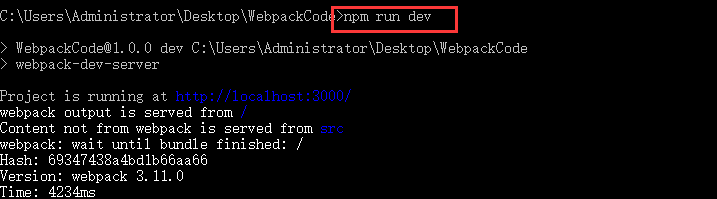
安装如下：

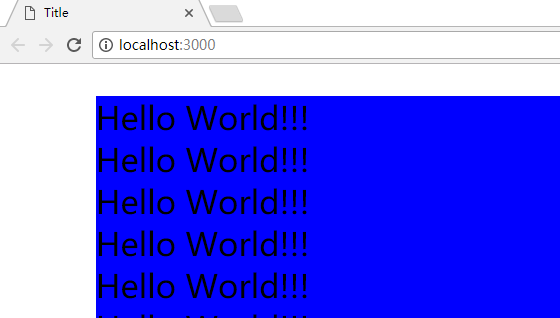


安装完毕后，打开，webpack.config.js这个配置文件，在里面新增一个配置节点，叫module，它是个对象，在个对象上，有个rules属性，这个属性是一个数组，在这个数组上，存放了所有的第三方文件的匹配和处理规则。如下：



**接着，运行如下：**





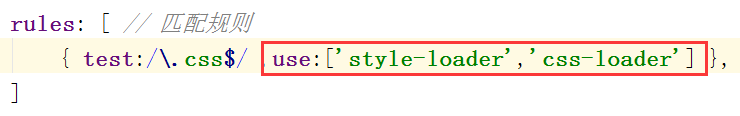
**小结：**

处理webpack不能识别的文件时，是通过loader来实现的。使用loader分步：

* 1、导入文件， import ‘./css/index.css’
* 2：安装loader npm I style-loader css-loader -d 具体安什么包不确定
* 第二步：配置 打开，webpack.config.js这个配置文件，在里面新增一个配置节点，叫module，它是个对象，在个对象上，有个rules属性，这个属性是一个数组，在这个数组上，存放了所有的第三方文件的匹配和处理规则。

## webpack处理第三方文件过程

下面的两个loader中先调用哪一个呢？

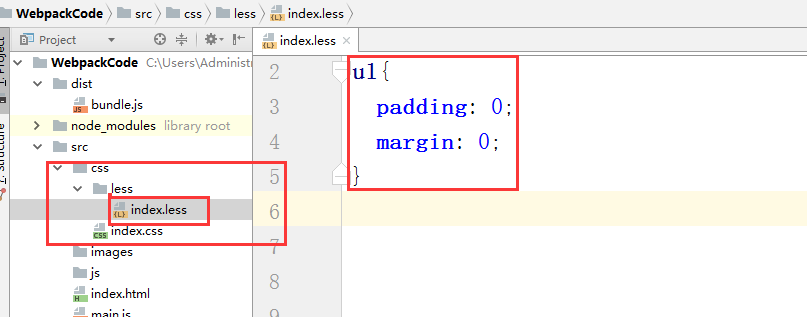


**Webpack处理第三方文件类型的过程：**

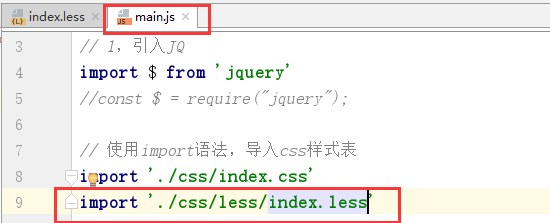
* 第一步：看一个要处理的文件是不是js类型文件，如果不是js类型的文件，就去配置文件中找，有没有对应的第三方loader的规则。
* 第二步：如果找到loader，就会调用对应的loader来处理这个文件类型。
* 第三步：这种调用顺序是从后向前的。调用完最后的一个loader后，将结果交给前一个loalder，.....
* 第三步：当第一个loader调用完毕后，会把结果直接交给webpack进行打包处理，最终输出到bundle.js中。

## 打包less文件

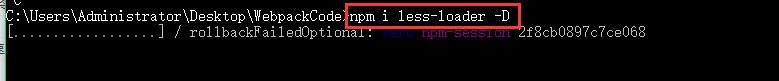
第一步：新建一个less文件，如下：



第二步：导入less文件，如下：



第三步：安装less-loader这个loader，如下：



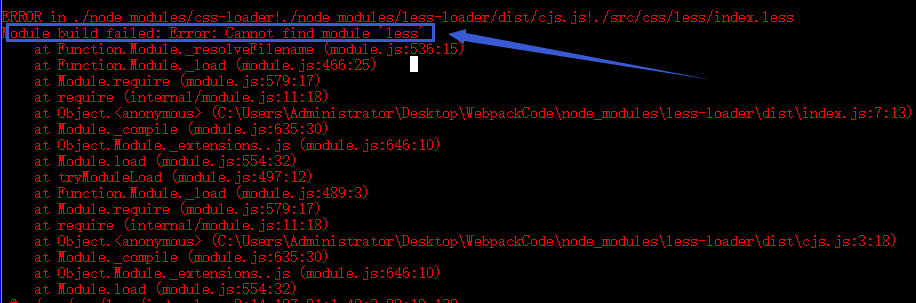
第四步：配置，如下：



第五步，运行，看效果，如下：



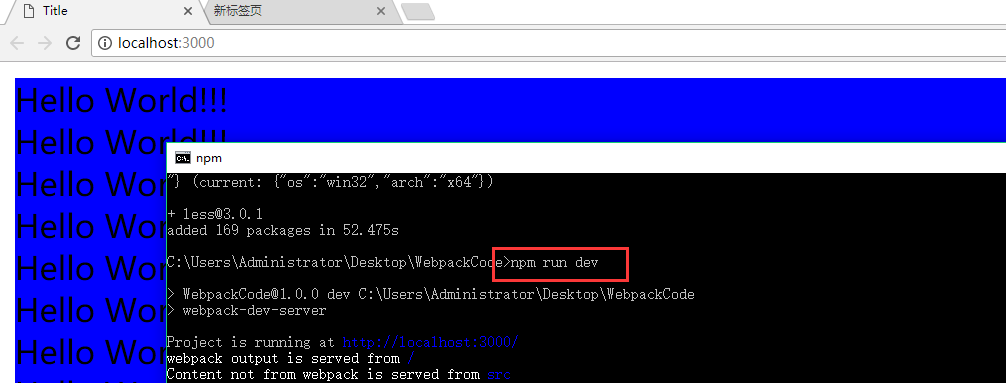
效果如下：



第六步，那么我们需要安装less这个包，如下：



第七步，再次运行一下，看效果，如下：



注意，要打包less文件，需要安装less这个包。

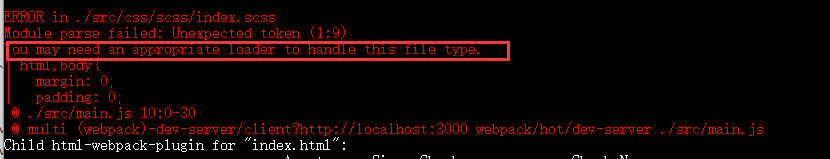
## 打包scss文件

第一步：新建一个scss文件，并写入如下代码，如下：



第二步，导入scss文件，如下：

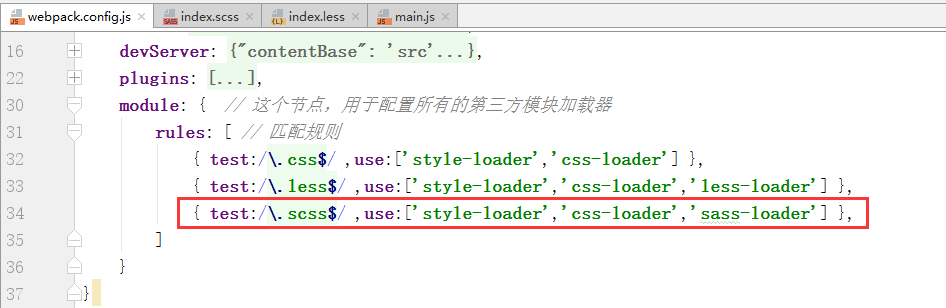




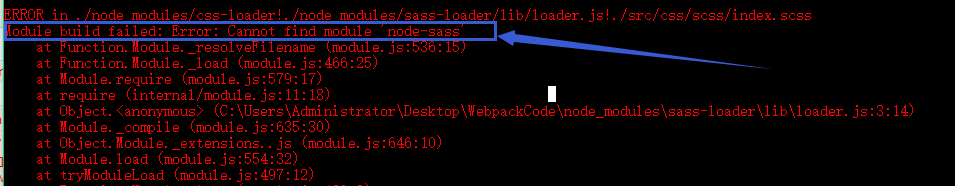
第三步，安装loader，如下：



第三步：配置，如下：



第四步，运行，看效果，如下：



原因：是少了一个包，node-sass这个包。

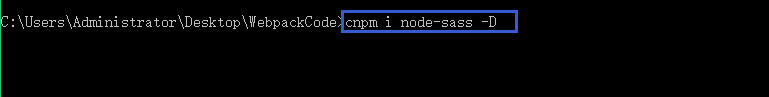
需要安装node-sass这个包，说明：这个包尽量使用cnpm 去安装，不要使用npm 去安装。

因为npm 容易安装失败。

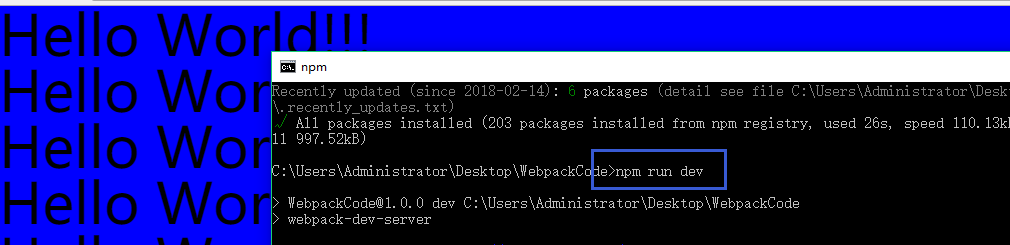
首先，需要全局安装cnpm 这个工具，如下：



然后，通过cnpm i node-sass -D，安装node-sass，如下：



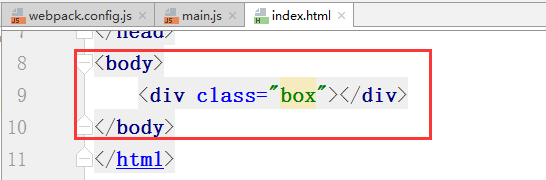
最后，运行，如下：



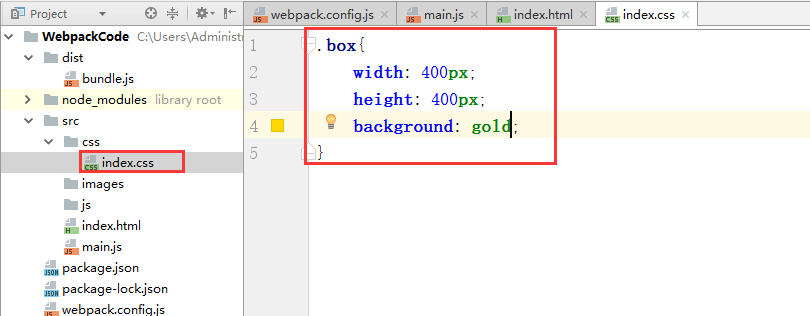
## 打包url

**在之前的代码基本上，修改或删除代码。**

先在index.html写入如下的代码，如下：



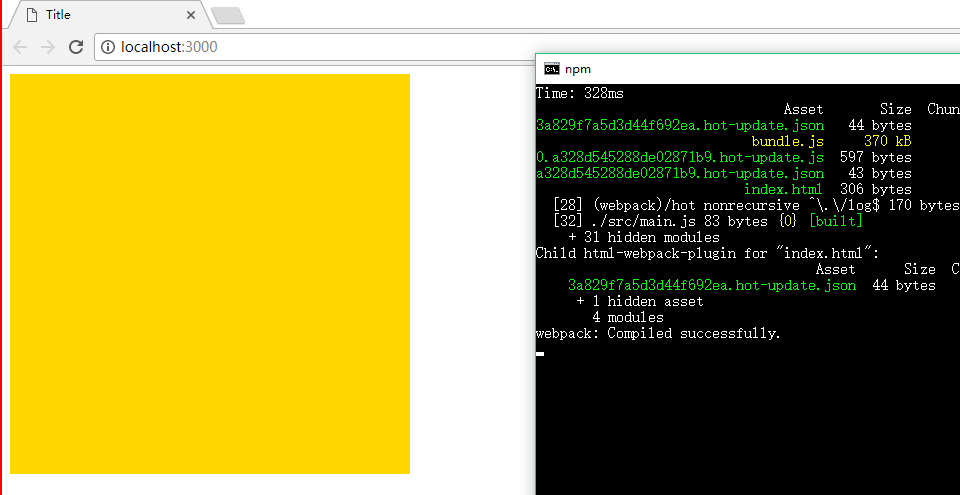
然后给div写一些样式，如下：



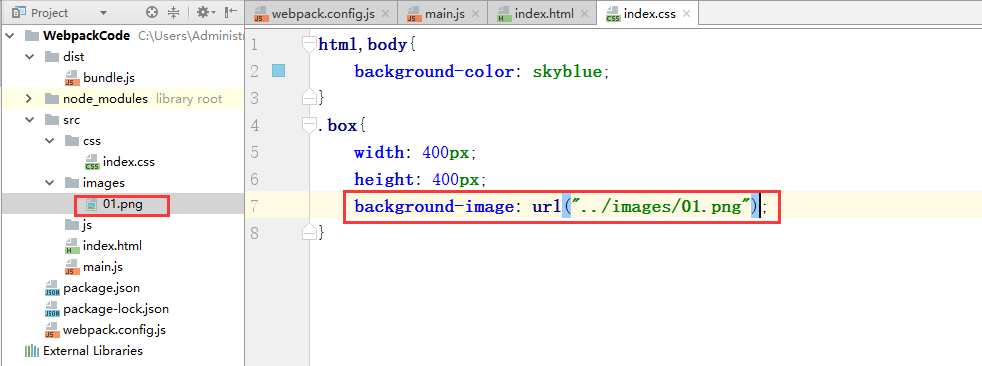
在main.js中，导入index.css文件，如下：



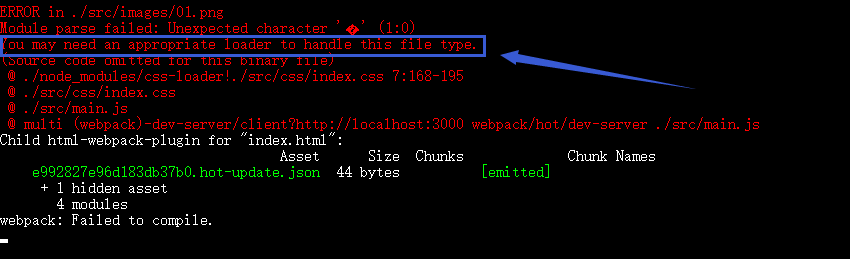
看效果，如下：



现在我要在div上，放一张小图片，如下：



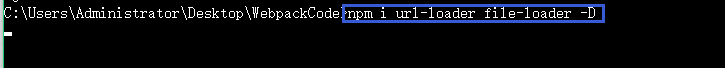
效果如下：



也就是说，需要一个loader来处理url。默认情况下，webpack无法处理url地址的，不管是这个地址是图片还是字体。

答：url-loader。

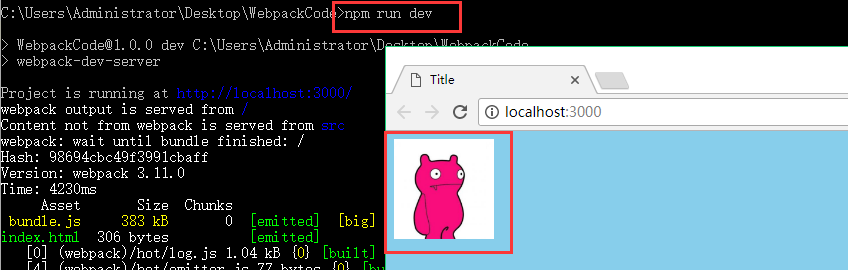
安装url-loader和file-loader，如下：



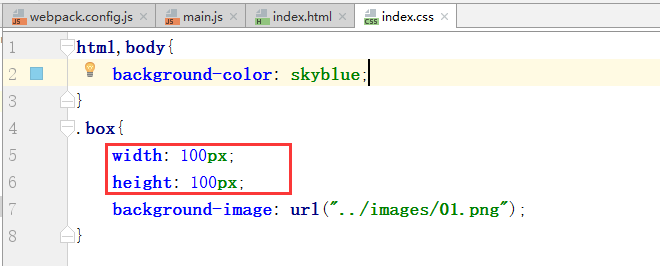
安装完毕后，需要配置，如下：



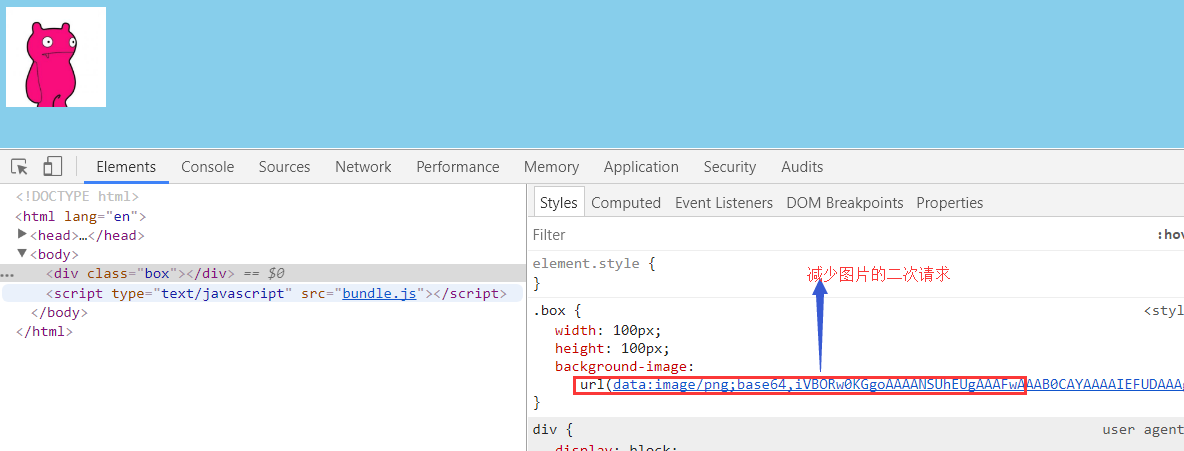
运行如下：



修改了代码如下：

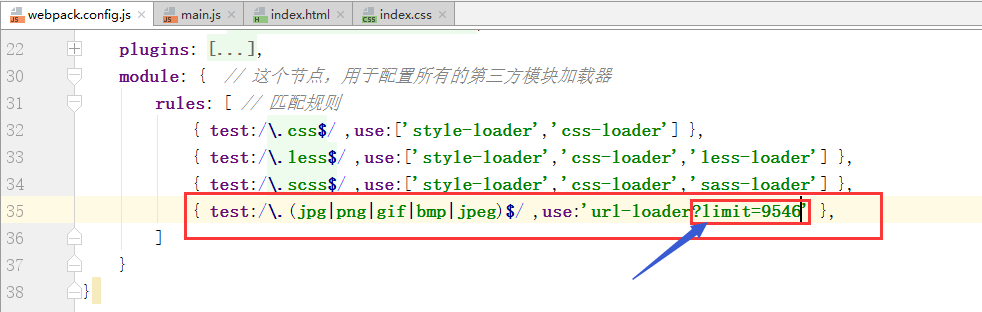


会把小图打包成base64，这样就可以减小二次请求了。如下：



现在，不想打包成base64，怎么做？

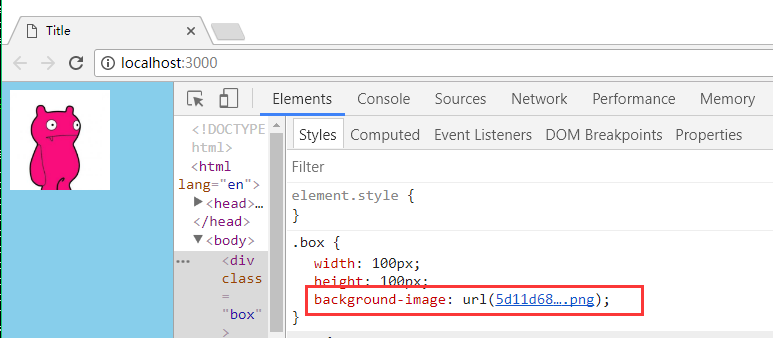
可以通过参数传递来实现。 如下：



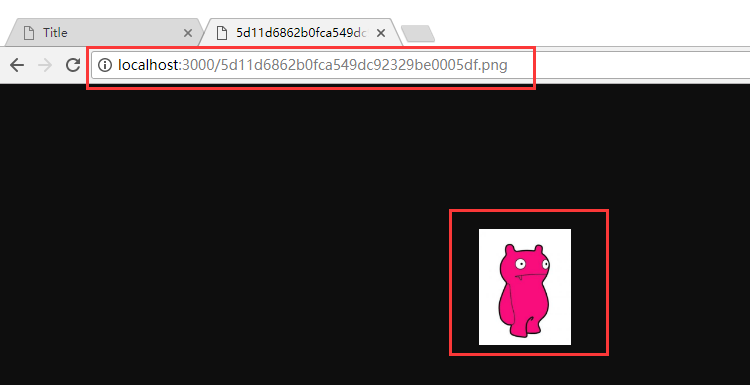
limit给定的值，是图片的大小，单位是byte，如果我们引用的图片，大于或等于给定的limit值，则不会被转为base64格式的字符串，如果图片小于给定的limti的值，则会被转成base64字符串。

重新运行，再看一个效果，如下：





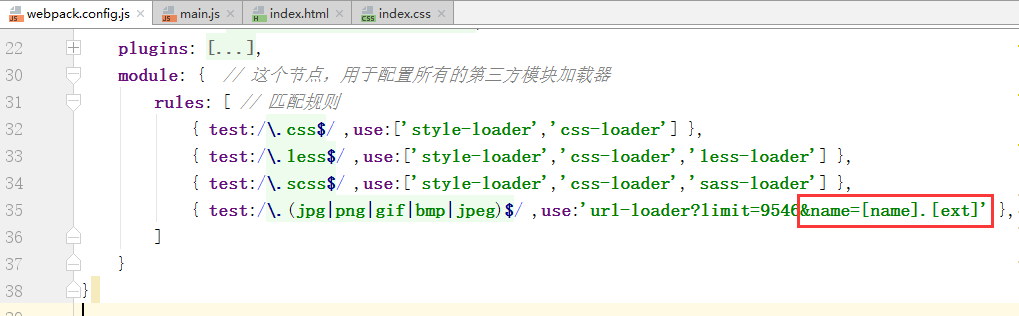
此时，图片就不是base64编码了。



为什么会把图片的名字给改了？

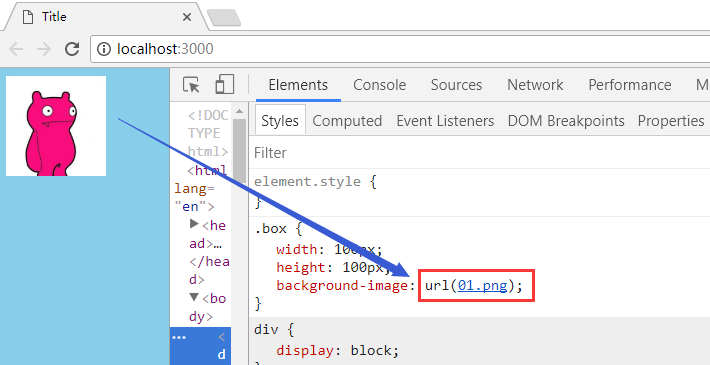
答：为了防止文件重名。

现在不想让它改名，做法如下：



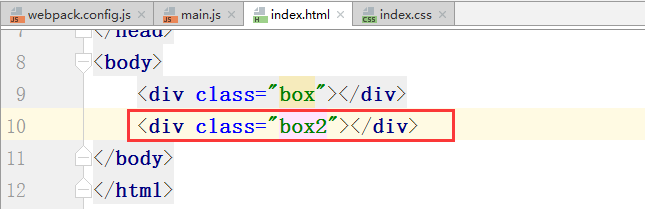
重新运行，再看一个效果，如下：





不改名有不好地方分析如下：

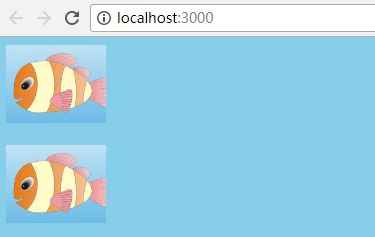
在div中在放一些图片，修改代码如下：



修改样式如下：



重新运行一下，按理说，两个盒子图片是不一样，但效果如下：



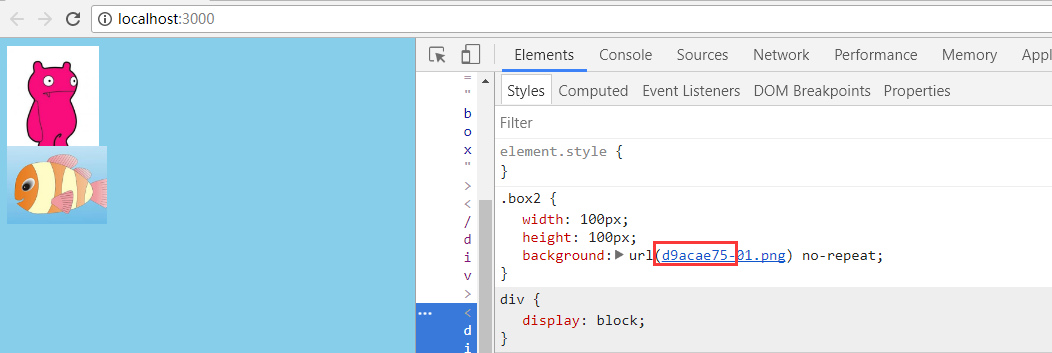
由于我们没有改名字，下面的图片会把上面的图片给覆盖了，所以，我们需要改名。

可以给图片名称前面加一个hash值，如下：



再次运行之，看效果，如下：



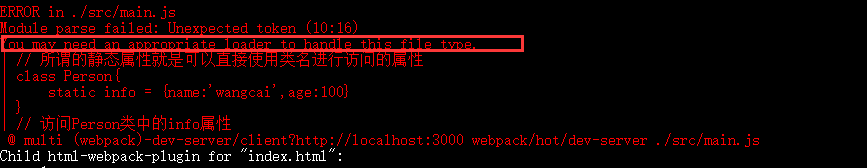


## webpack中的babel配置

第一步：在main.js中写如下的代码：



第二步：编译如下：



**分析错误的原因：**

在webpack中，默认只能处理一部分的es6的新语法，一些更加高级的es6或es7中的语法 ，webpack是处理不了的，这个时候，就需要借助第三方的loader来处理了，当第三方的loader把高级的语法转换为低级的语法后，会把结果交给webpack去打包到bundle.js中

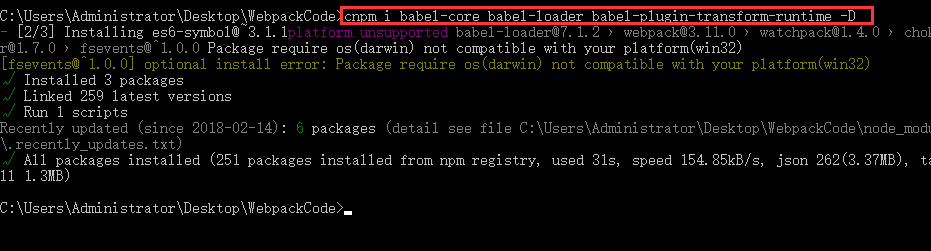
安装步骤如下：

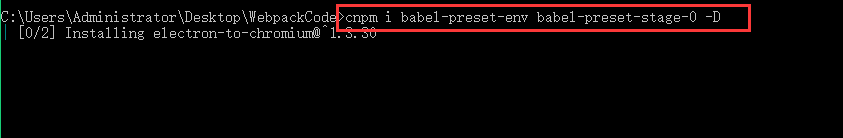
* **第一步：**

通过babel，可以帮我们将高级的语法转换成低级的语法。在webpack中，可以运行如下两套包，去安装babel相关的loader功能：

第一套：babel-core babel-loader babel-plugin-transform-runtime

第二套：babel-preset-env babel-preset-stage-0



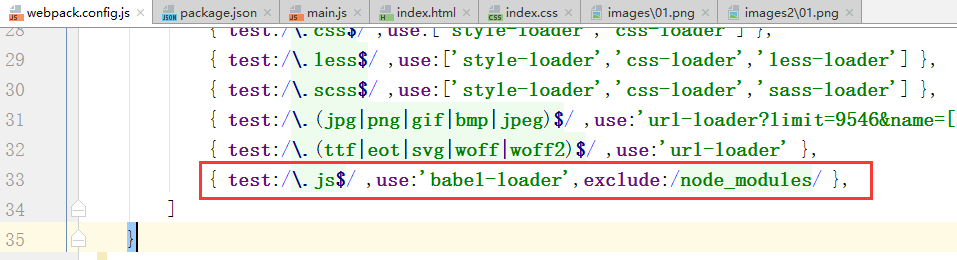


* **第二步：**

配置babel的loader规则，配置时，需要通过exclude选项排除node\_modules这个选项。

原因：如果不排除，则babel会把node\_modules中所有的第三方js文件都进行打包处理，这样会非常消耗cpu，同时打包速度非常慢。

有时候，打包了node\_modules，会导致项目不能运行。



* **第三步：**

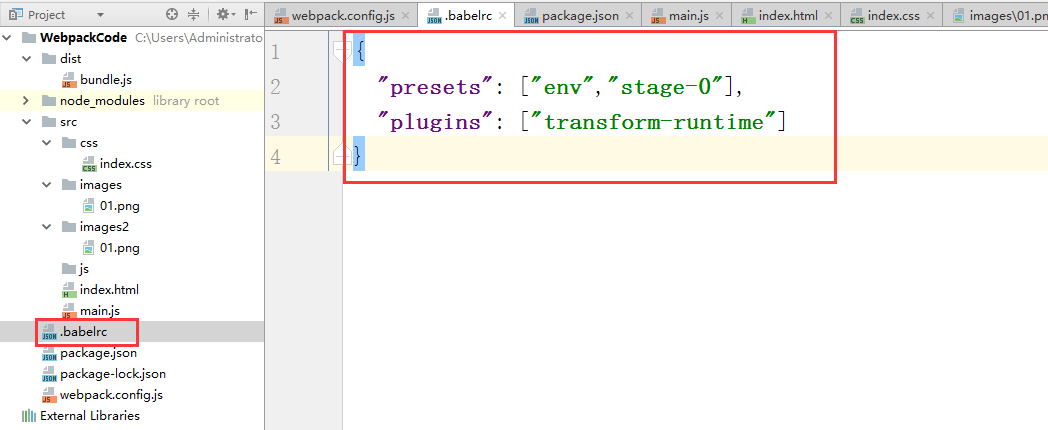
在项目的根目录下，新建一个.babelrc的babel的配置文件，这个配置文件，属性json的格式。并写入如下的内容： preset是语法的意思

{

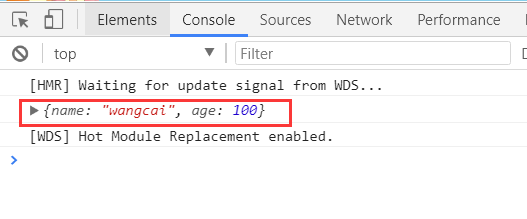
“presets”:[“env”,”stage-0”],

“plugins”:[“transform-runtime”]

}



接着，重新启动，如下：



# Vue中使用webpack

## 使用vue实例中的render方法渲染组件

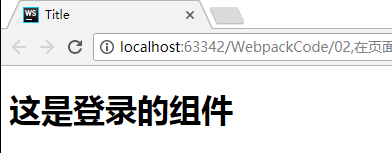
### 在页面中渲染基本的组件



### 在页面中使用render函数渲染组件

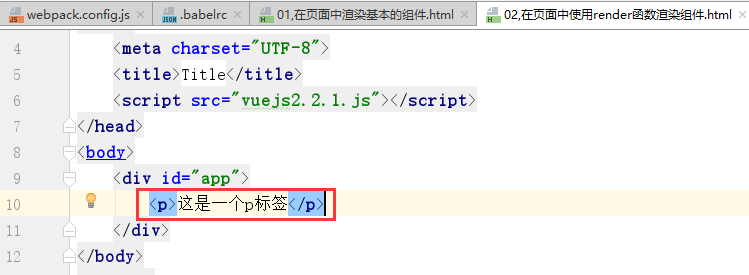


结果如下：



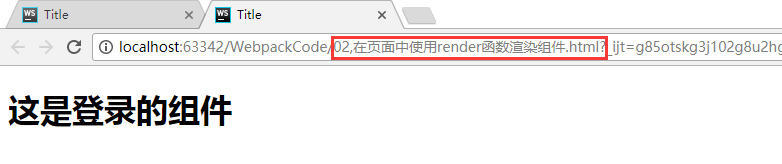
在div中添加代码如下：





现在结果如下：





Render函数是替换

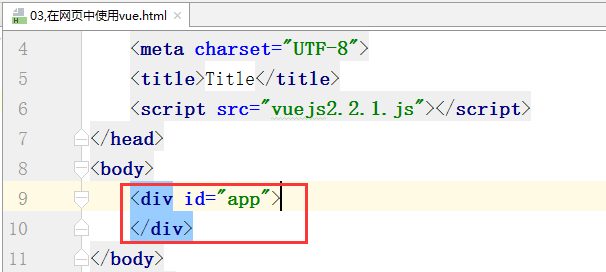
## webpack导入vue和普通网页使用scirpt标签导入vue的区别

### 在网页中使用vue

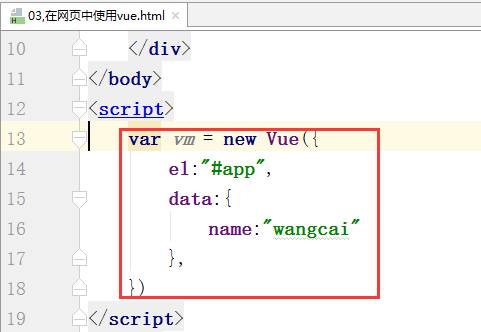
第一步：使用scirpt标签，引入vue包



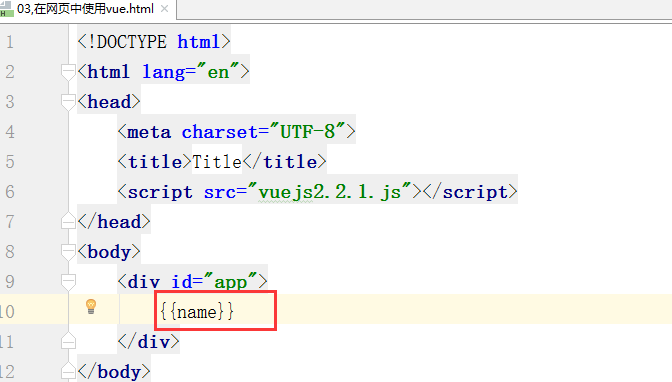
第二步：在页面中，创建一个id为app的div容器



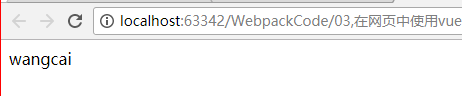
第三步：new一个vue实例，如下：



第四步，使用数据，如下：

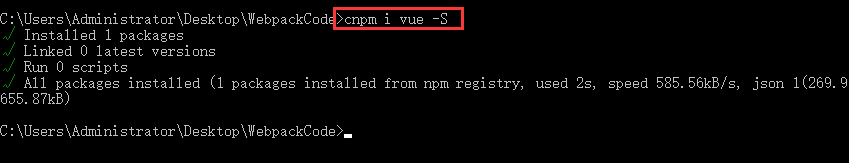


第五步，效果如下：

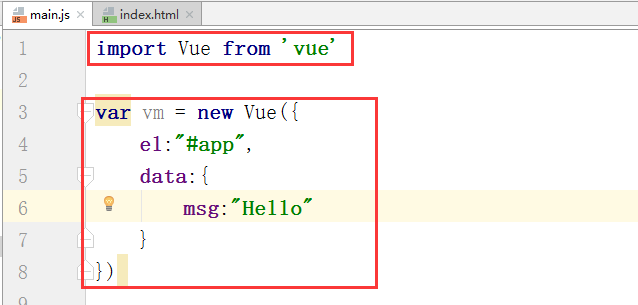


### 如何在webpack构建的项目中，使用vue

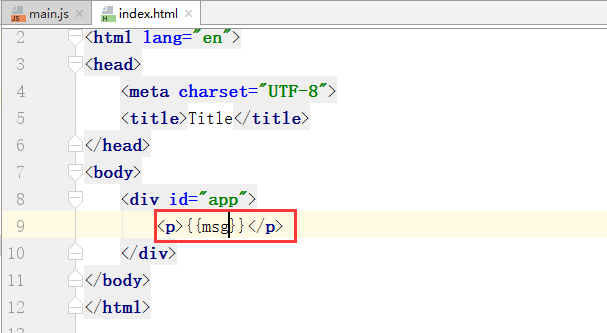
第一步：安装vue



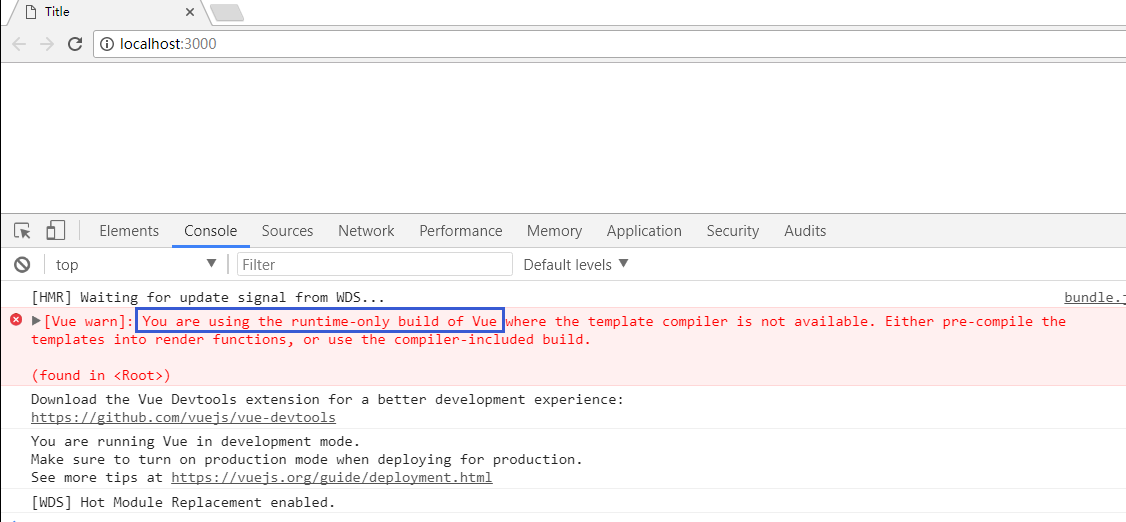
第二步：导入vue，如下：



第三步，使用：



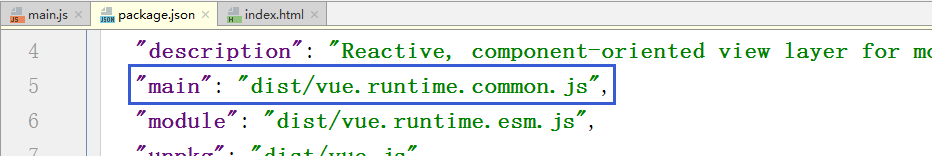
第四步，结果如下：



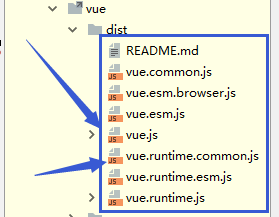
**错误的原因是：**我们在这里使用的vue和网页中通过script标签使用的vue不是同一个，也就是说这里使用的vue不是一个完整的vue。或者说，我们通过import Vue from ‘vue’导入的vue功能不是特别完善，只提供了runtime-only的方式，并没有提供像网页中那样的使用方式。

复习一下，包的查找规则：

1. 找项目根目录中没有没node\_modules的文件夹
2. 如果找到，在node\_modules中根据包名找对应有的vue文件夹
3. 如果找到，在vue文件夹中，找一个叫package.json的包配置文件
4. 在package.json文件中，查找一个main的属性，它指定了这个包在加载时，加载哪个文件。如下：



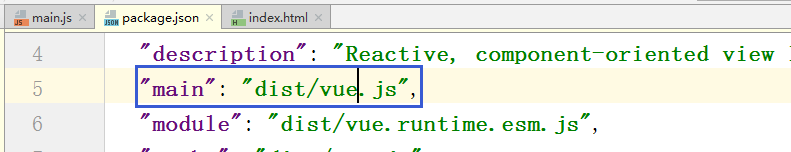
发现，它加载的不是vue.js，而是vue.runtime.common.js这个文件，在vue这个包下面有如下的文件：



我们实际上需要的是vue.js，但是它加载的不是vue.js。所以说，在上面使用vue时，就报错了。

如何让它不报错？

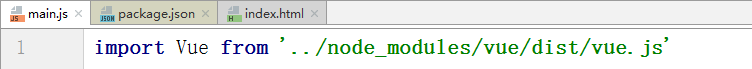
**方式一：**



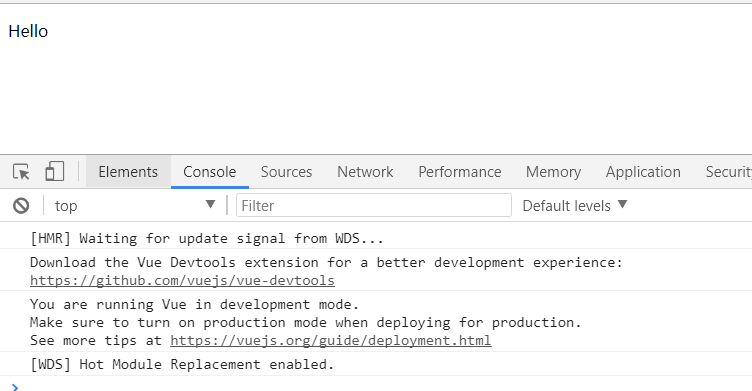
这种方式不好，因为，我们直接修改人家的代码。

**方式二：**

导入真正的vue.js，如下：

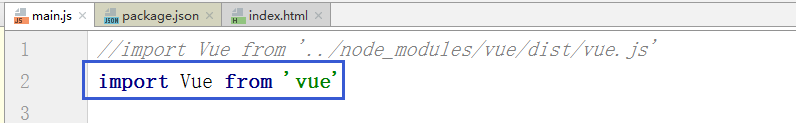


运行看一下：



**方式三：**

还是导入vue，如下：

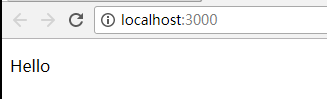


接着在webpack.config.js中添加一个节点，如下：



重新运行，看效果：

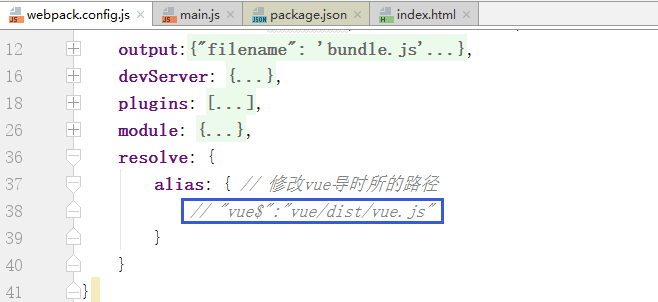


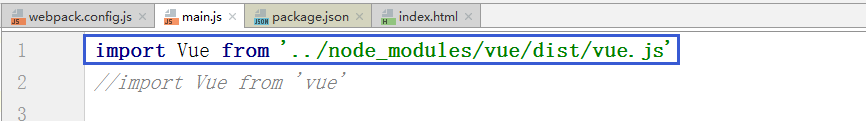


通常我们是使用方式3

## 在vue中结合render函数渲染指定的组件到容器中

修改代码如下：

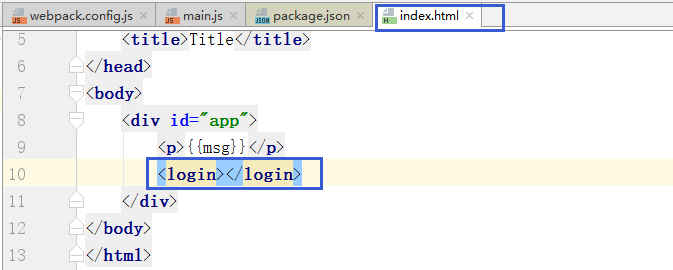




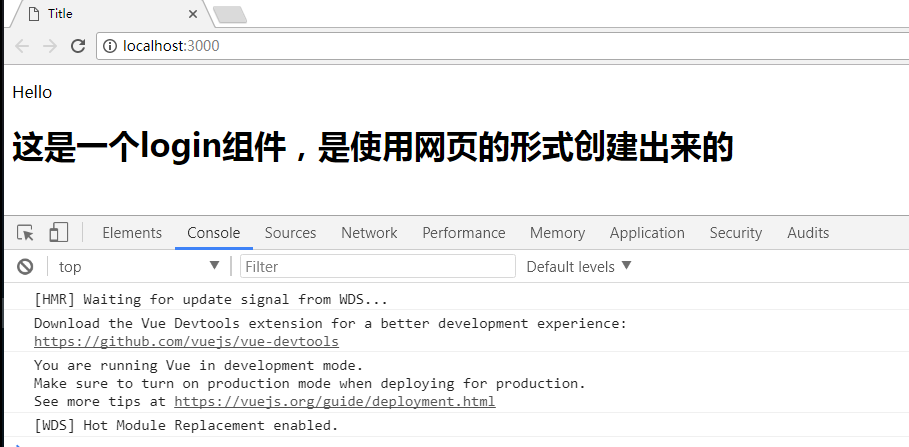
定义组件：



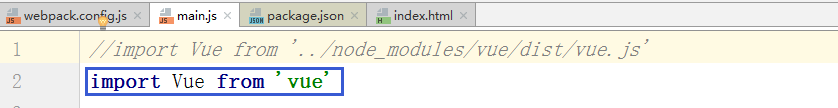
在index.html中使用组件，如下：



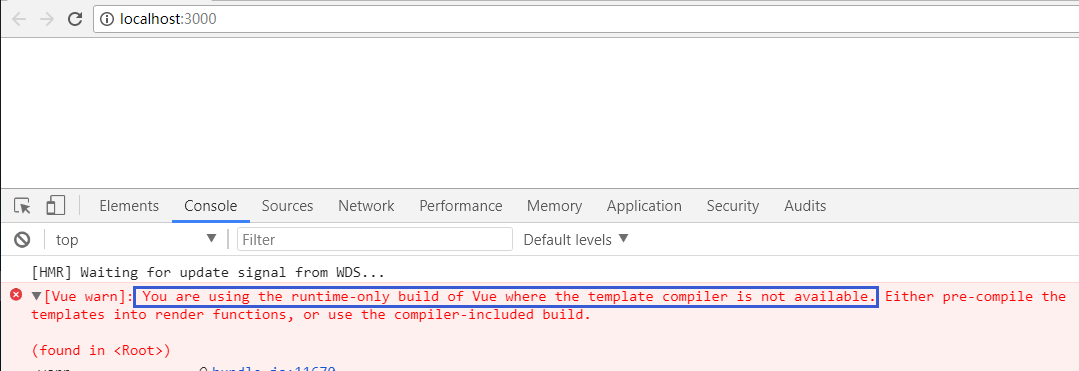
效果如下：



如果使用import Vue from ‘vue’，如下：



效果：

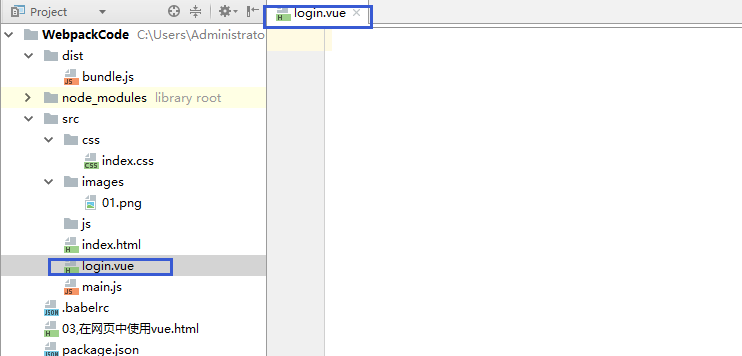


解决办法：

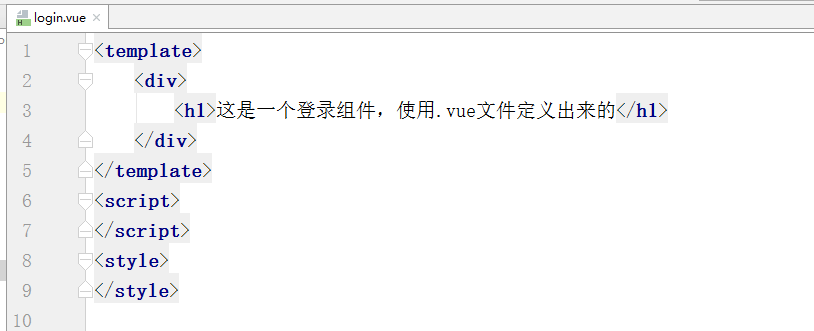
第一步：把之前定义vue的组件方式去掉：



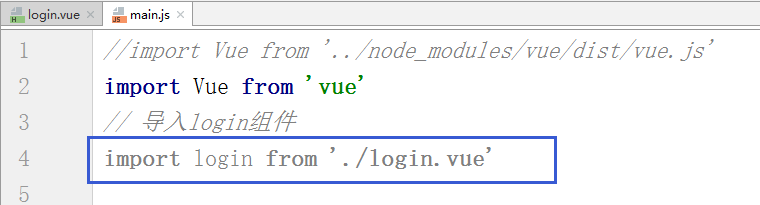
第二步，创建一个login.vue这个文件



写入如下的代码：

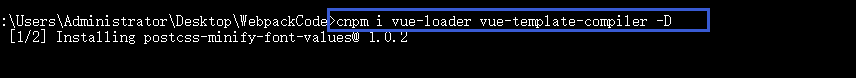


第三步：导入login组件，如下：

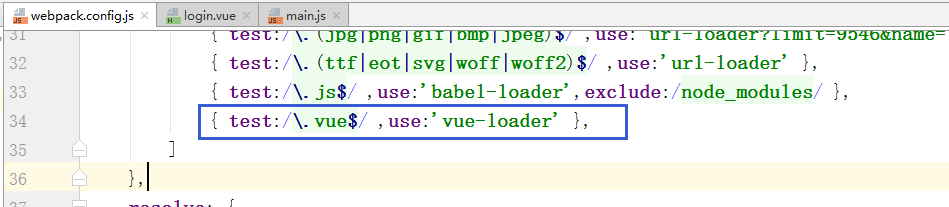


默认，webpack无法打包.vue文件，需要安装相关的loader：vue-loader vue-template-compiler

如下：



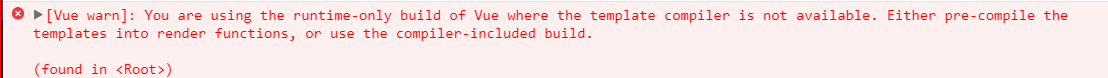
当然，还需要在配置文件中配置：



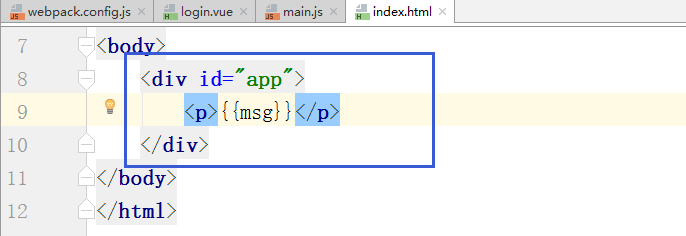
重新npm run dev，如下：

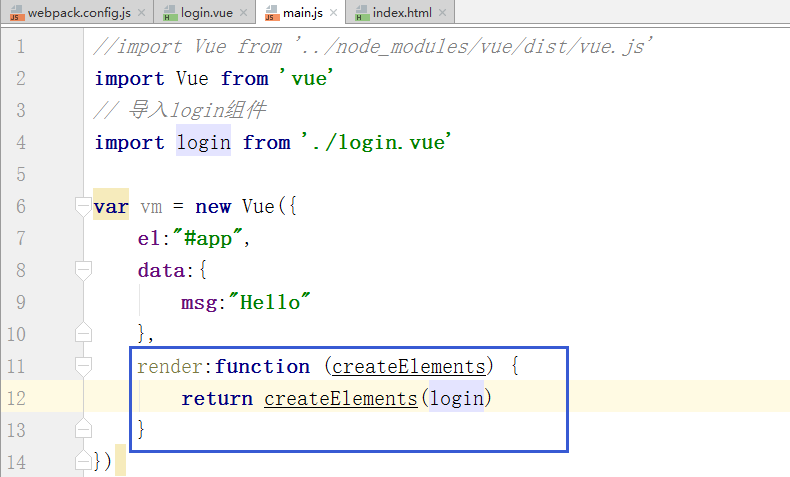


报错了：

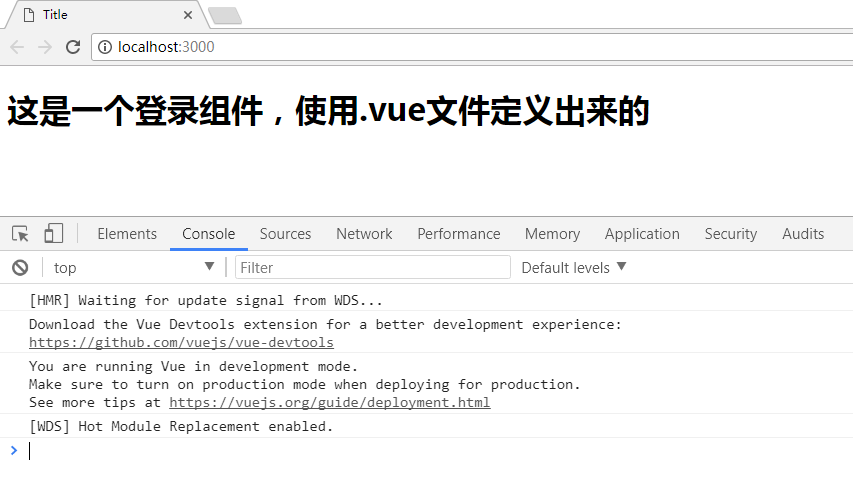


解决，如下：

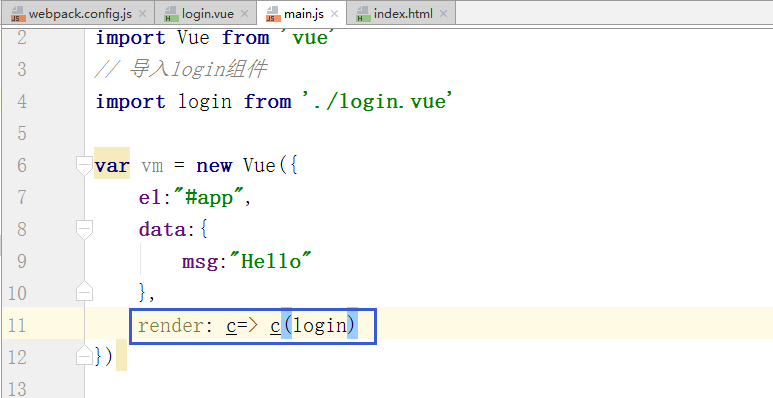




再次运行之，如下：



上面的render是可以简写的，如下：



再看一下效果，如下：



**总结webpack使用vue的步骤：(不用非按照不走写，最后一样就行)**

* 1，安装vue包 npm i vue -S
* 2，由于在webpack中，推荐使用.vue这个组件模板文件定义组件，所以，需要安装能够解析这种文件的loader vue-loader vue-template-complier
* 3，在main.js中，导入vue模块 import Vue from ‘vue’
* 4，定义一个.vue结尾的组件，包含三个标签：template script style
* 5，导入组件 import login from ‘./login.vue’ 并在配置文件中定义{text:/.\vue$/,ues:’vue-loader’}
* 6，创建vue实例， var vm = new Vue({ el:”#app”, render:c=>c(login) })
* 7，在页面中创建一个id为app的div元素，作为我们vm实例要控制的区域

## export default 和 export

### （1）export default

能不能在组件中使用数据？

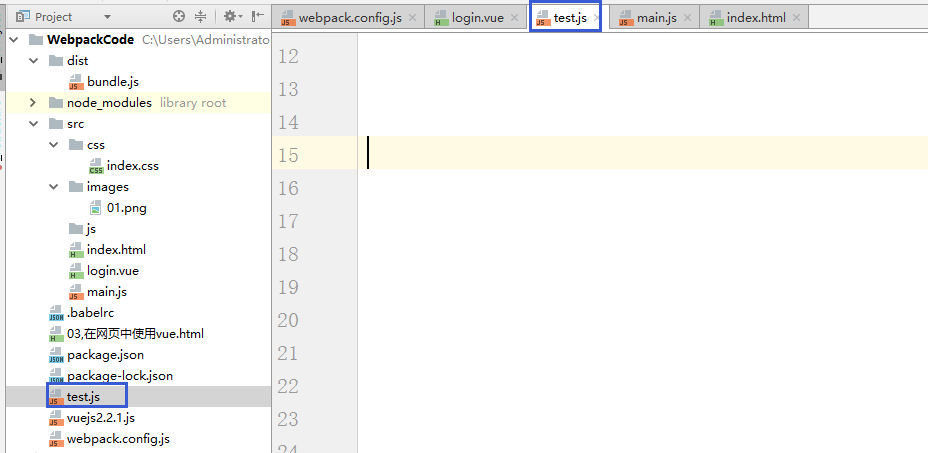
答：肯定可以的。

使用如下：



创建一个test.js进行演示exports default。如下：

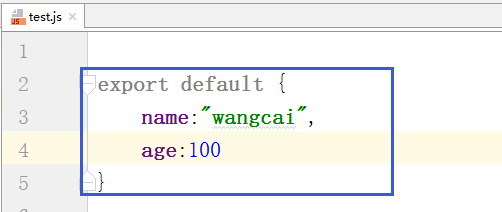
第一步：在项目的根目录下创建一个test.js文件，如下：



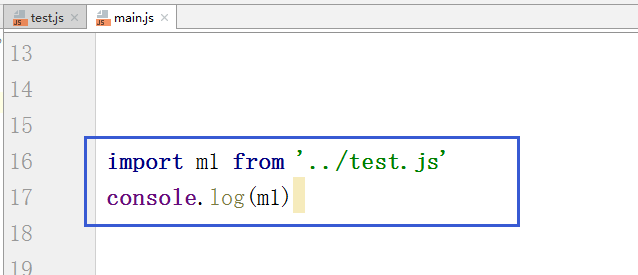
**在node中使用var 名称 = require(模块名) 载入模块，使用module.exports和exports来暴露成员。**

**那在es6中，使用import 模块名 from “模块的标识符” 来载入模块，使用export default和export向外暴露成员。**

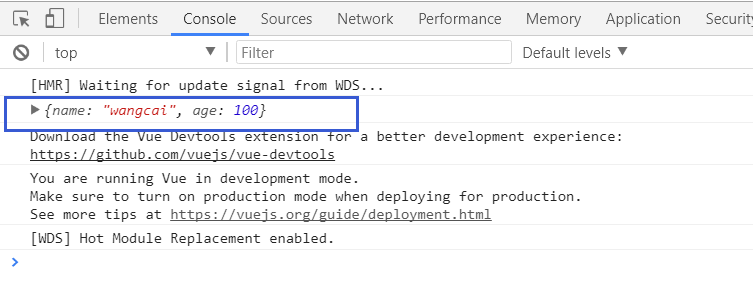
在test.js中写如下的代码：



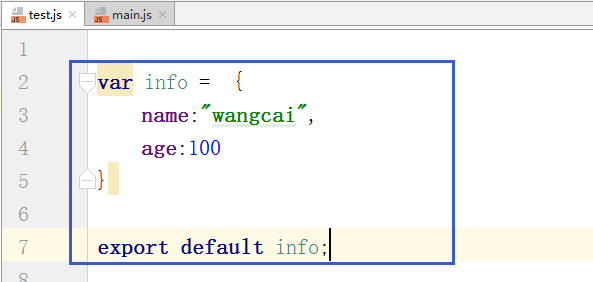
在main.js中使用：



结果：



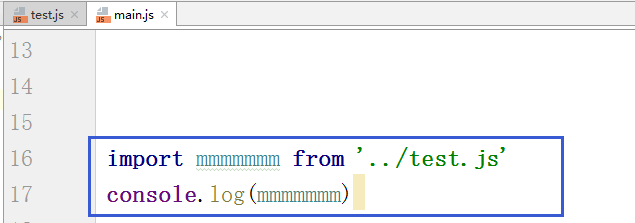
另一种写法：



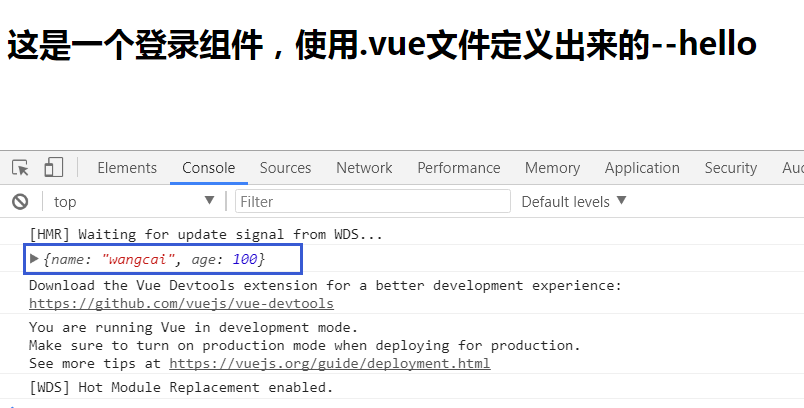
结果：



**通过export default向外暴露成员时，可以使用任意的变量来接收，如下**：

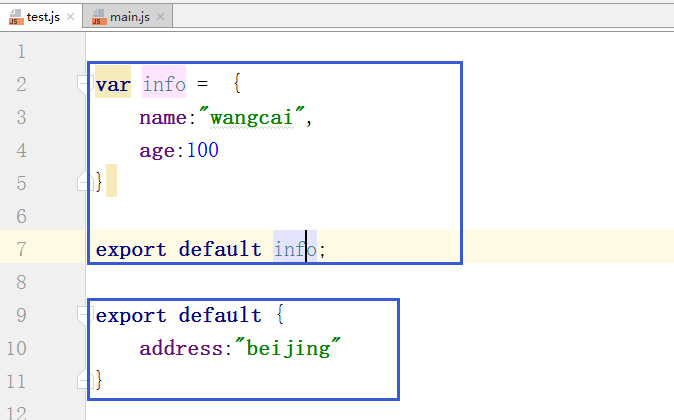


结果：



现在考虑一个问题，能不能使用export default来暴露多个对象呢？

答：不可以 如下：



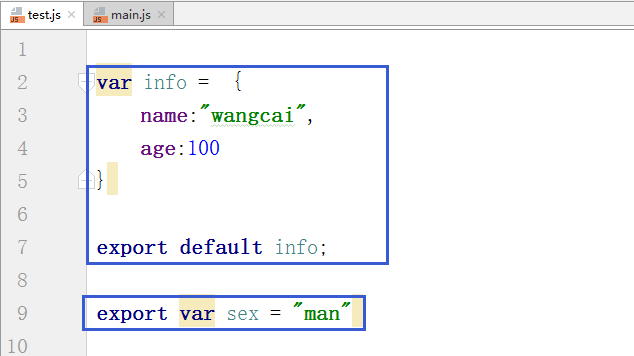
结果：



**在一个模块中，export default只能向外暴露一次，不能暴露多次。**

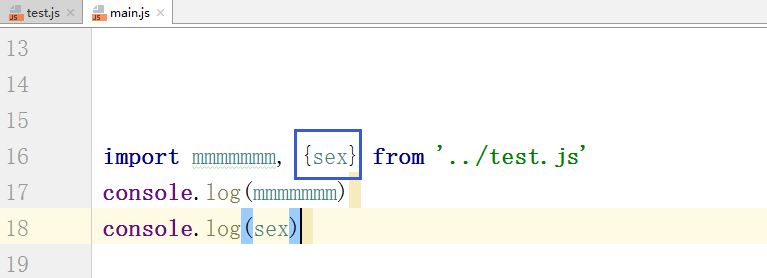
### （2）export

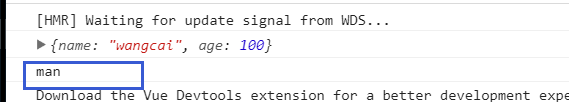
在test.js中使用export向外暴露数据如下：



**也就是说在同一个模块中，可以同时使用export default和export向外暴露成员。**

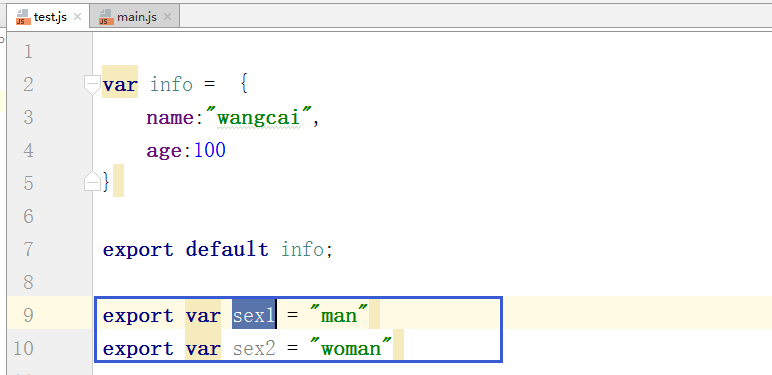
在main.js中进行接收成员，如下：



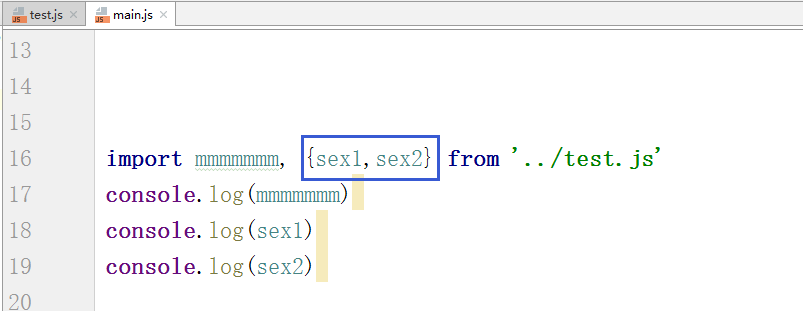


在使用export向外暴露成员时，只能使用{}的形式来接收，这种形式，叫做按需导出。

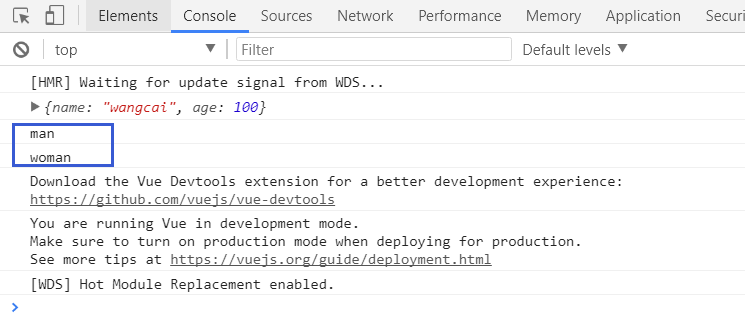
现在在test.js中导出sex1和sex2，如下：



在main.js中，可以接收两个，也可以接收一个，按需来，如下：

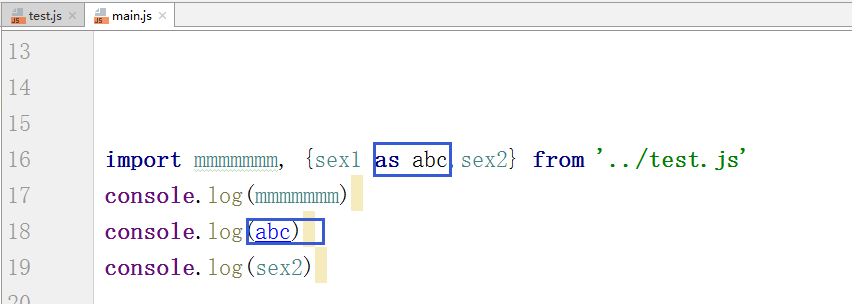


在浏览器中，如下：



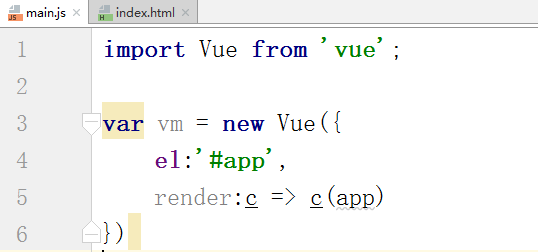
export可以向外暴露多个成员，export defalute只能向外暴露一个成员。

使用export导出成员时，必须严格按照导出时的名称使用{}来接收，如果想换个名子，可以使用如下的写法：

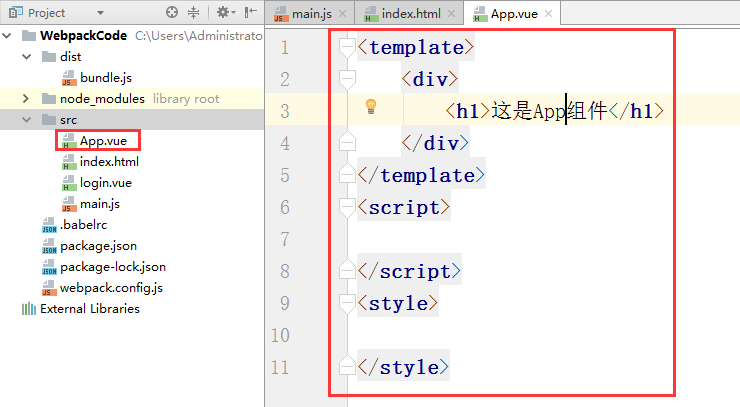


## 5，结合webpack使用vue-router

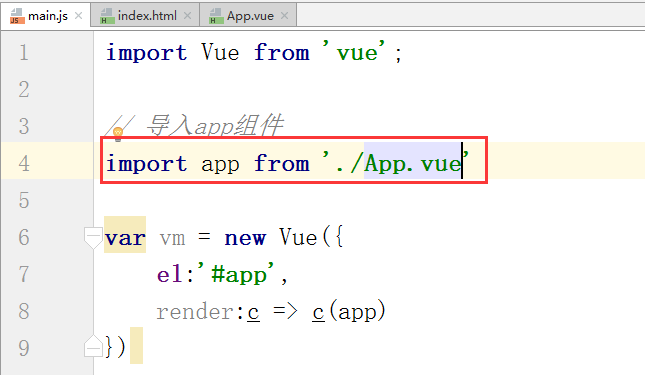
在main.js，创建一个vue实例：



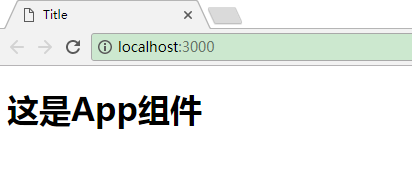
在scr目录下面创建一个App.vue的一个app组件，如下：



在main.js中导入app组件，如下：



运行之，看效果：



**接着，就要学习****vue-router了。**

官网如下：https://router.vuejs.org/zh-cn/installation.html

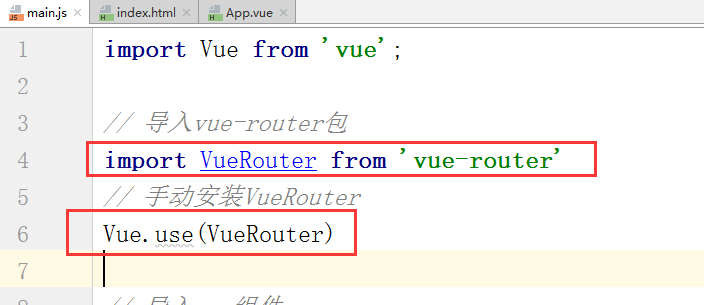


第一步：安装vue-router

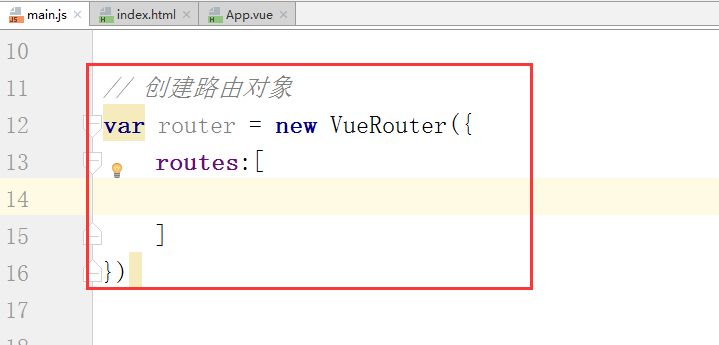


第二步：导入模块，并使用vue.use()，如下：

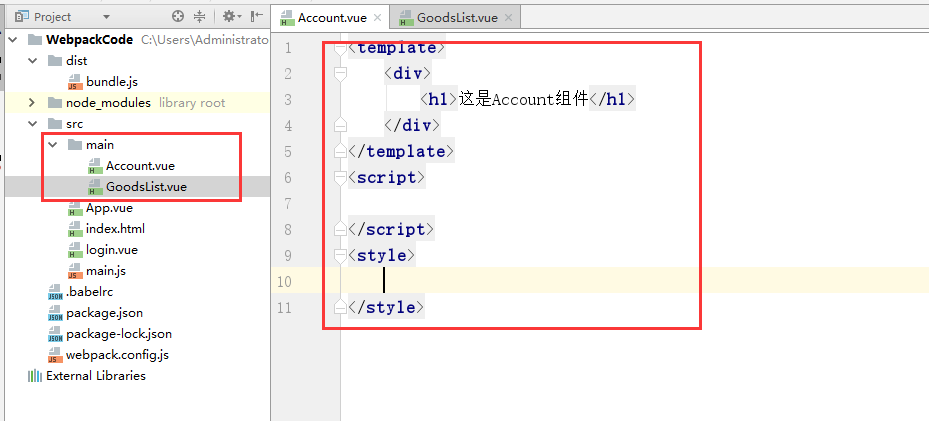


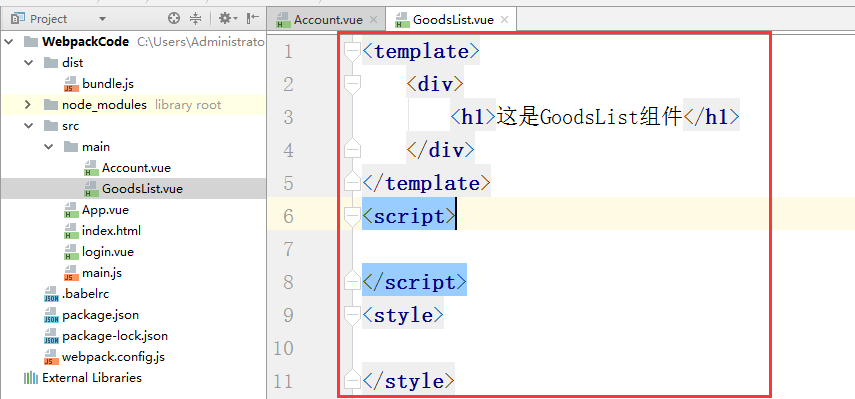


第三步：创建路由对象：

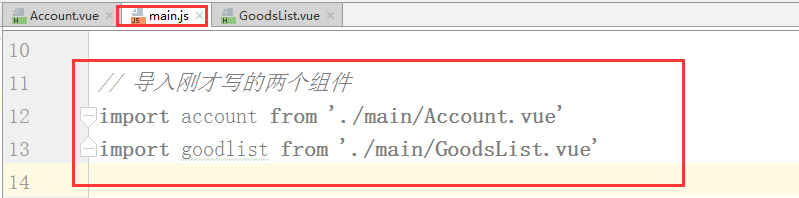


第四步：创建两个组件，一个是商品列表组件，另一个是账号组件，如下：





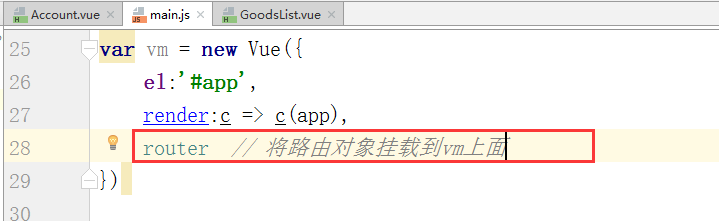
第五步：在main.js中导入这两个组件，如下：



第六步：匹配路由规则：



第七步：将路由对象挂载到vm上面



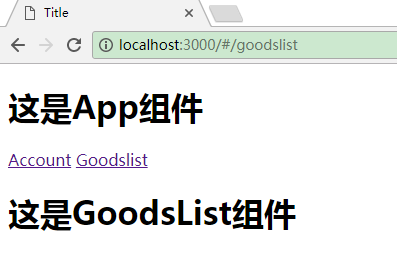
注意：render会把el指定的容器中，所有的内容都清空掉，所以不要把路由router-view和router-link直接写到el所指定的容器中。

第八步：在app.vue这个组件中写router-link 以及 router-view



第九步：看效果：



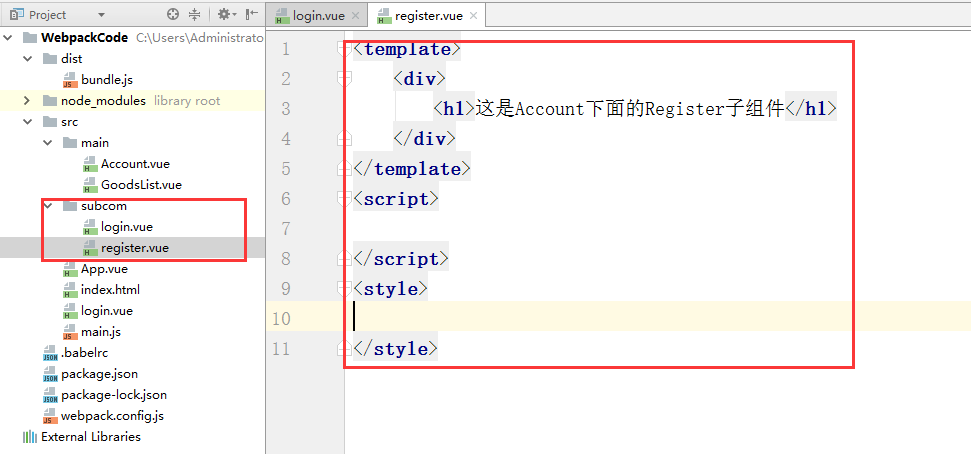


## 6，路由嵌套

现在，我要在账号组件下，创建两个子组件，一个是login组件，另一个是register组件。现在需要创建两个子组件，如下：

第一步：创建两个子组件：





第二步：在main.js中导入两个子组件：



第三步：匹配路由的规则：



第四步：在Account.vue中写router-link以及router-view，如下：



第五步：效果如下：



## 7，style标签lang属性和scoped属性介绍

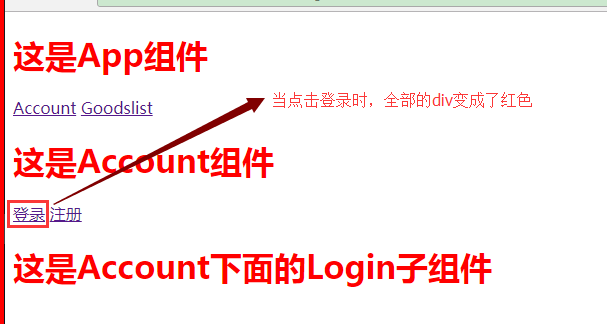
我在把login这个组件的颜色换成红色？

怎么做？

有人会这样做：在login.vue中写入如下的代码：



看效果如下：



怎么改进？

这个时候，就需要用到scoped，如下：



现在再看一个效果：



**注意：只在是在.vue的文件中，那么一般都是会使用scoped属性的。**

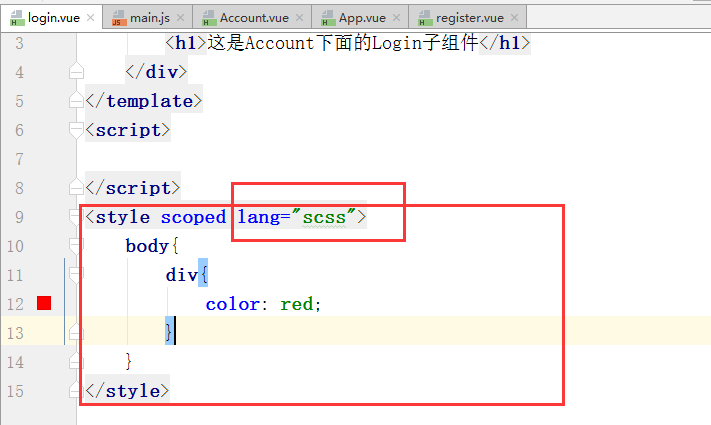
现在需要在style中写入scss语法格式的代码，能不能写？

答：不能，如下：



没有效果。

普通的style标签只支持普通的样式，如果要启用scss或less，需要为style标签添加lang属性，如下：



再看一个效果，如下：



有时候会报缺少各种模块，这个时候，你就可以把node\_modules全部删除，然后重新安装，一般情况下，安装完就没有问题。

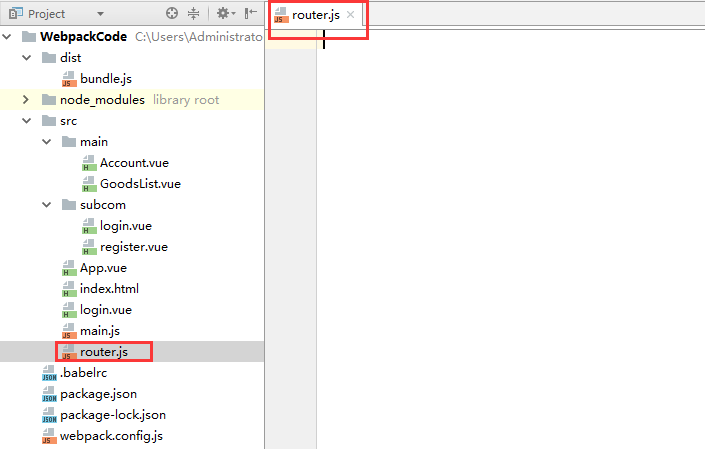


## 8，路由模块

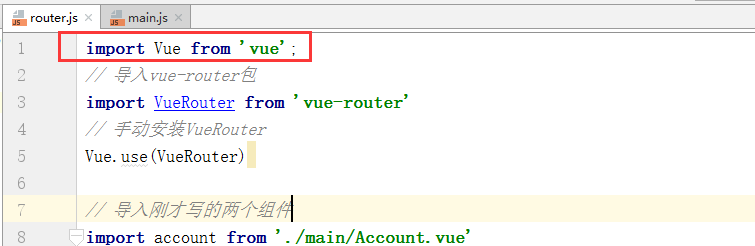
问题：



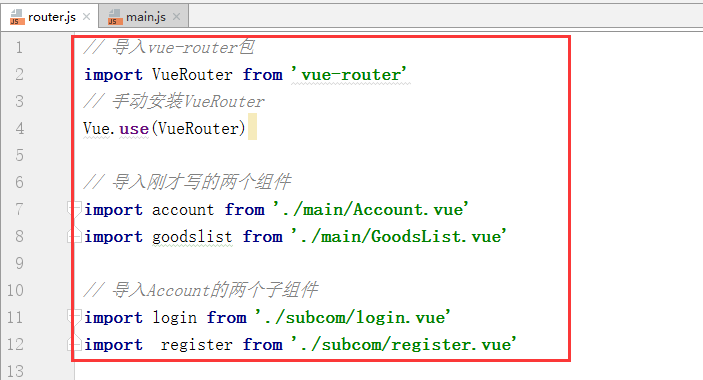
在src目录下面，新建一个router.js，作为路由模块，如下：



把相关的代码，copy到router.js中，如下：

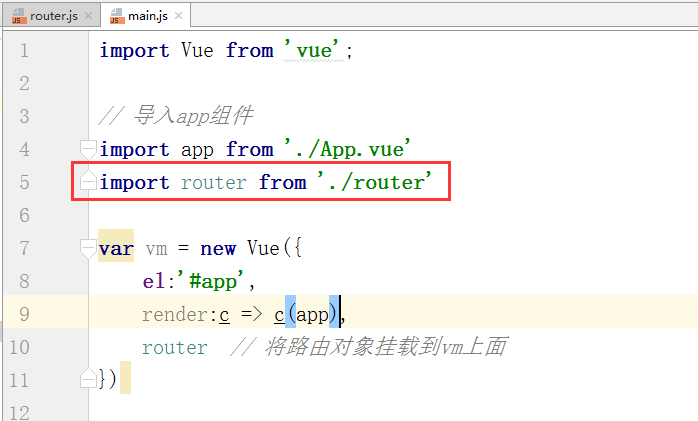


注意：同时也需要把Vue在router.js中导入。





接着，在main.js中就要导入router.js文件，如下：



## mintUI

官网如下：



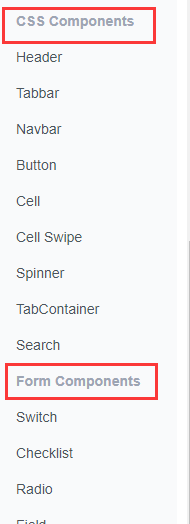


说明，MintUI是基于Vue.js的，它和前面学习的Bootstrap是不一样的，Bootstrap仅仅是代码片断，任意项目都可以使用Bootstrap，但是MintUI只能在VUE的项目中使用。因为它本身就是基于Vue的一个组件库，并且是移动端的。

**所以说，不能把它当用Bootstrap来使用。**

在MainUI中，把组件分成了三类，如下：





在使用mintUI时，可以引入全部组件，也可以引入部分的组件。

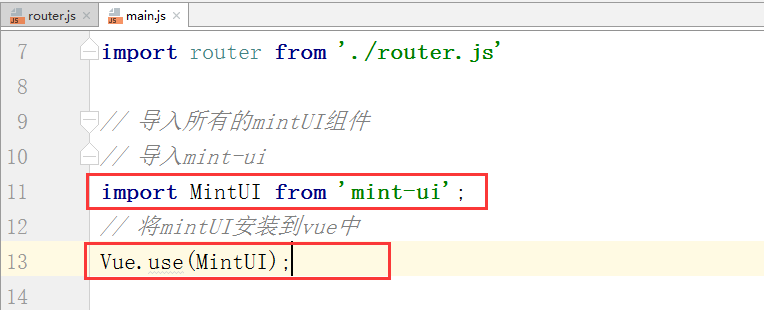
### 引入全部的组件

mintUI中有很多的组件，导入所有的组件，有时候我们并不能使用那么多。所以推荐按需导入。

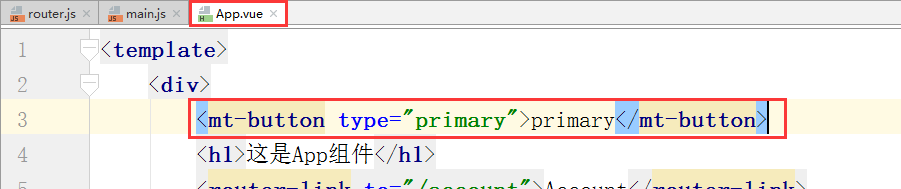
第一步：安装mint-ui



第二步：完整引入mint-ui



第三步：就可以在项目中，使用mintUI提供的组件了，如下：



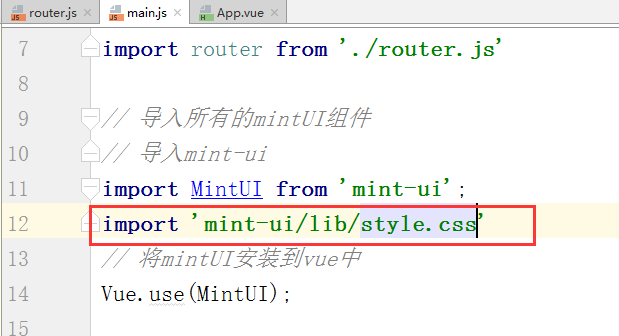
第四步，看效果，如下：



上面的按钮并不是我们所需要的，我们所需要的如下：



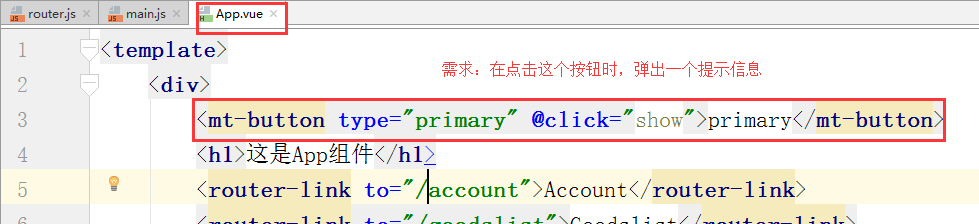
原因：我们没有把样式给导入进来，现在导入一下它的样式，如下：



现在再看一下效果，如下：



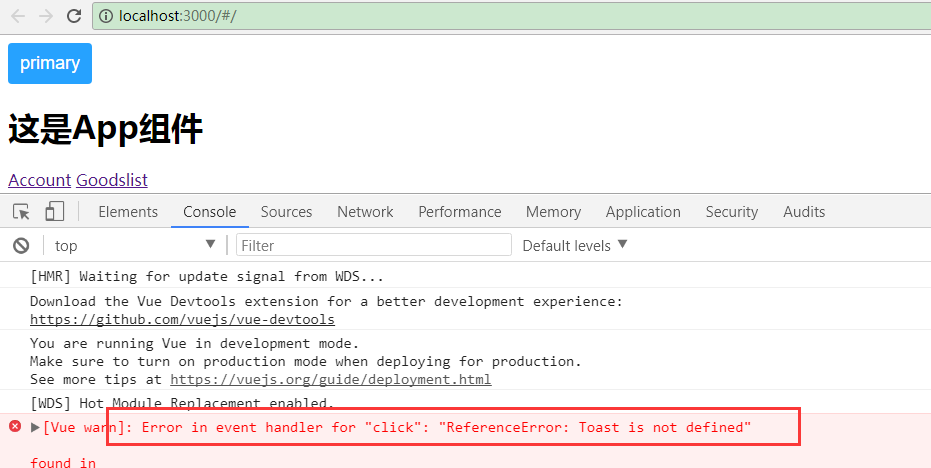
### Toast



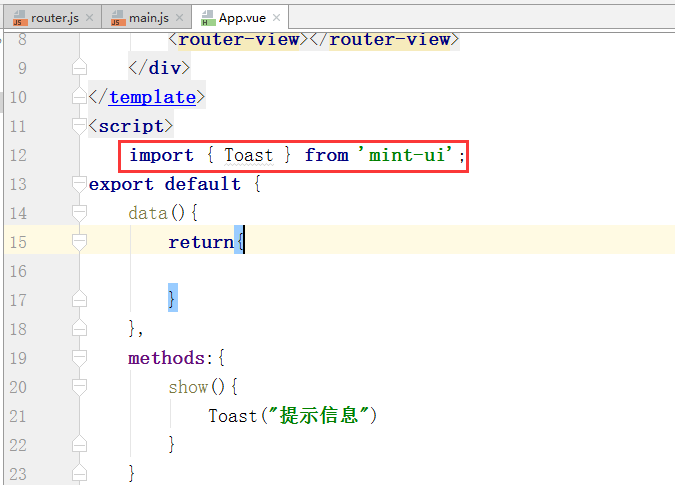
接着，在sciprt标签，写如下的代码：



看一个效果：



在这个组件中需要导入Toast组件，如下：

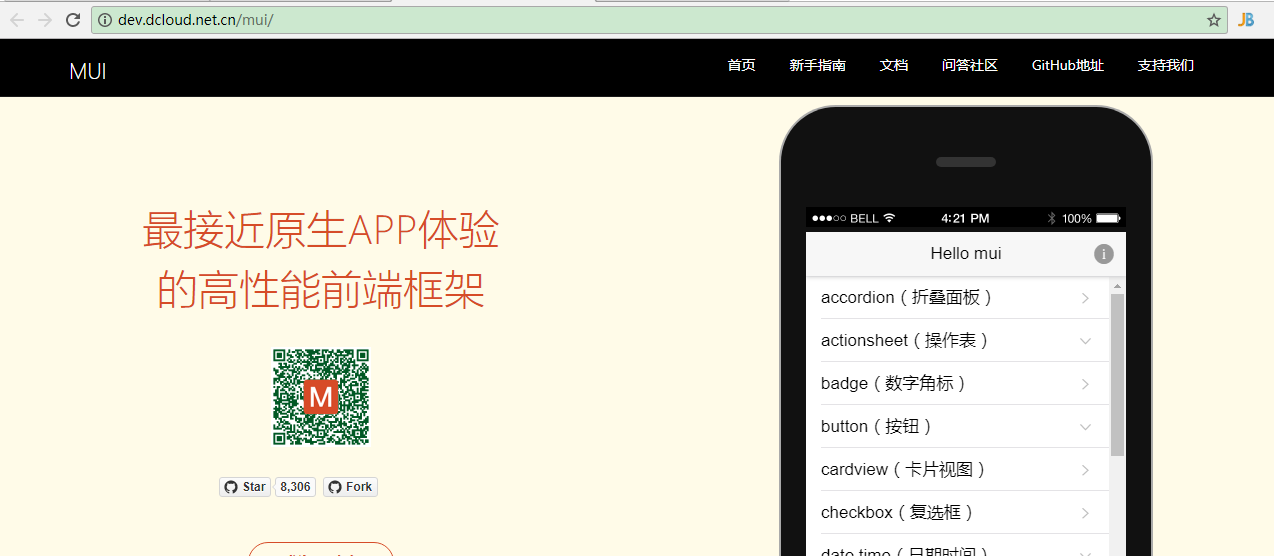


再次看一个效果，如下：



## MUI

官网：



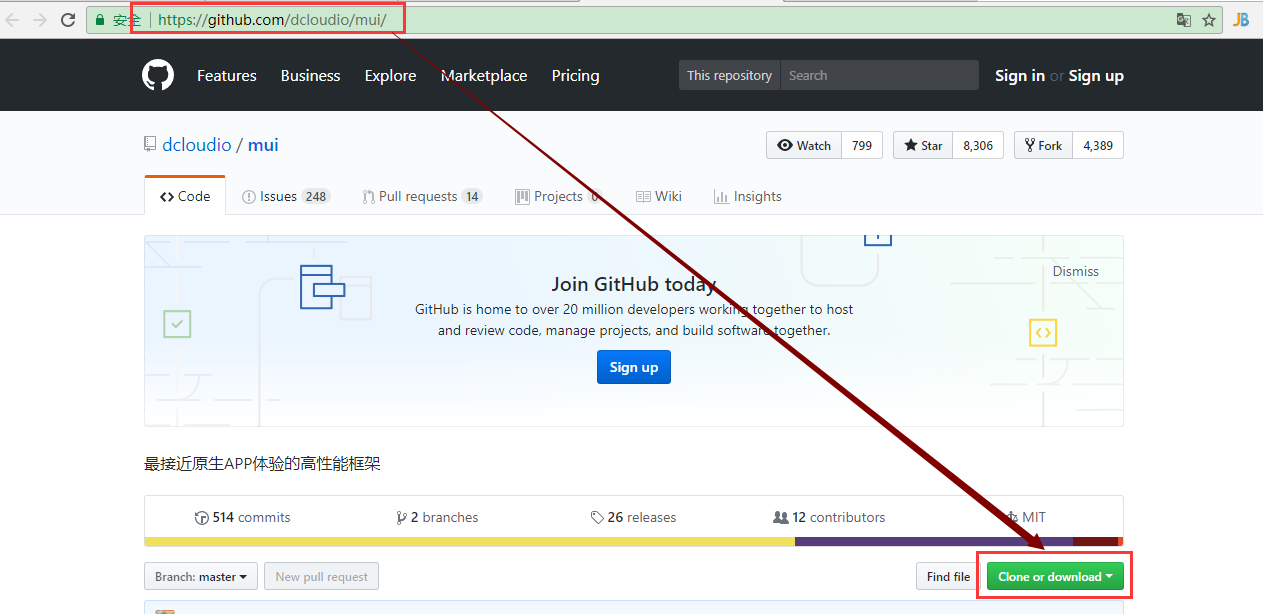
注意：MUI不同于MintUI，MUI只是一套好用的代码片段，类似于Bootstrap，而MintUI，是真正的组件库，是使用VUE技术封装出来的成套的组件，可以和VUE项目无缝集成。

因此，从体验上来说，MintUI体验更好，MUI和Bootstrap用法差不多。

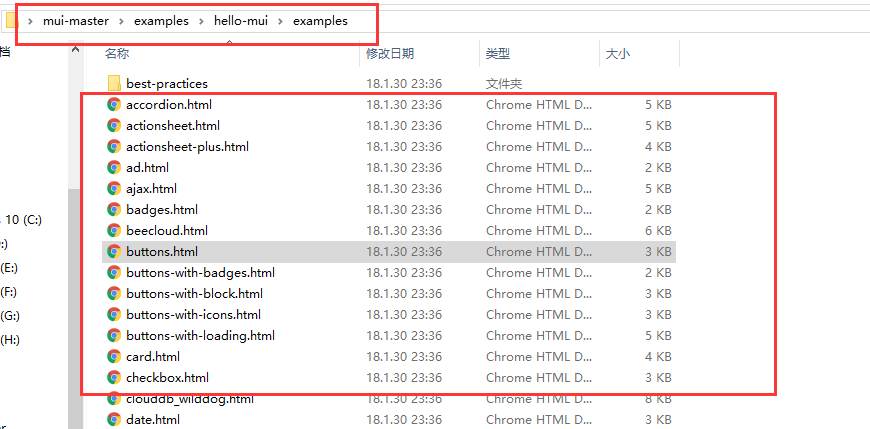
理论上来说，任何项目都可以使用MUI或Bootstrap，但是MintUI只适用于VUE项目。

学习MUI时，你就可以把它当用Bootstrap来学习。

第一步：下载MUI。



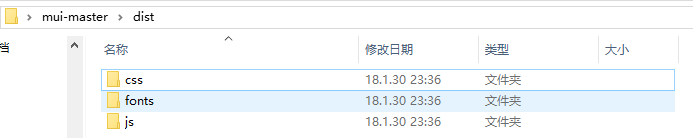
下载下来后，得到一个压缩包，解压，找到如下路径：



在这个examples中，有很多的hmtl文件，我们就可以运行html文件，找到我们想样的效果，copy代码就OK。

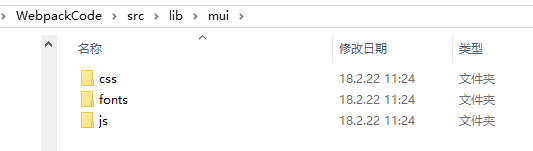
说明：MUI不能使用npm下去下，只能手动地下载。解压，放到项目中。

第二步：找到下载解压好的mui文件夹后，找到dist文件夹，如下：



类似bootstrap中的css，fonts，js。

第三步，在项目的src目录下，新建一个lib文件夹，这个文件夹就是用来放手动copy的库文件，如下：

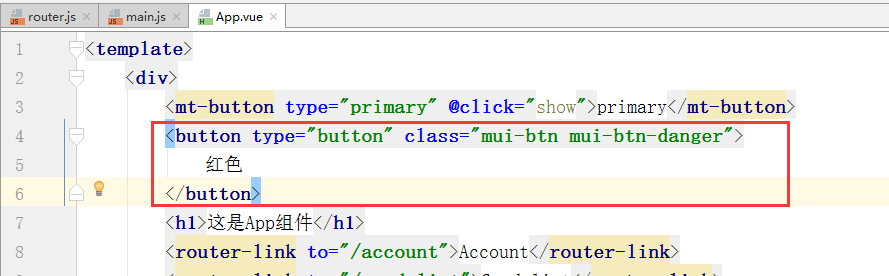


第四步：在项目中去使用MUI提供好代码，如下：

当然，首先是找到你所需要的代码，如下：



把代码copy到项目中，如下：



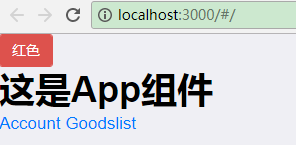
第五步：运行代码，如下：



发现，红色的按钮并没有出来，原因是没有引入样式 ，引入如下：



再看一下效果，如下：



和使用Bootstrap是有点类似的。

说明：webpack中的基本项目中的用到的所有的内容就说完了，有了这个基础，我们就可以做项目了。 app