# 复星项目前台开发指南v0.1

[复星项目前台开发指南v0.1 1](#_Toc25525)

[开发环境安装 1](#_Toc16170)

[1、 安装Node.js 2](#_Toc31052)

[2、Webstorm安装及配置 4](#_Toc29975)

[3、 下载工程文件 8](#_Toc5143)

[目录结构 13](#_Toc3362)

[1、 modules业务模块 14](#_Toc14414)

[2、 styles 样式 16](#_Toc27282)

[命名规范 16](#_Toc5219)

[1、 文件命名 16](#_Toc328)

[2、变量命名 17](#_Toc14081)

[3、 RESTFul API设计规范 19](#_Toc11854)

[4、 Angular Style Guide 19](#_Toc5613)

[开发我的第一个页面 19](#_Toc29104)

[1、 创建业务目录 19](#_Toc29939)

[2、 创建业务模块定义文件 19](#_Toc31794)

[3、 将模块添加到工程中 19](#_Toc30029)

[4、 创建路由文件 20](#_Toc22435)

[5、 创建用户列表控制器 20](#_Toc6432)

[6、 创建用户列表 21](#_Toc26330)

[7、 执行gulp 21](#_Toc9221)

[CRUD 增删改查 22](#_Toc8874)

[1、创建API文件： 23](#_Toc18617)

[2、 查询列表 23](#_Toc25413)

[3、 查询 24](#_Toc21861)

[4、 编辑 27](#_Toc16873)

[5、 新增 30](#_Toc23332)

[6、 删除 31](#_Toc15566)

[在弹出窗口中进行操作 32](#_Toc29356)

[数据校验 32](#_Toc22446)

[其他场景 32](#_Toc3908)

[常用组件 32](#_Toc1374)

# 开发环境安装

复星项目采取前、后台分离的方式进行开发。前台开发环境如下：

|  |  |
| --- | --- |
| IDE | Webstrom10 |
| 工具 | Git、Node.js、Gulp、Bower等 |
| 文件服务器 | http-server |
|  |  |

下面将从0开始，搭建我们的开发环境

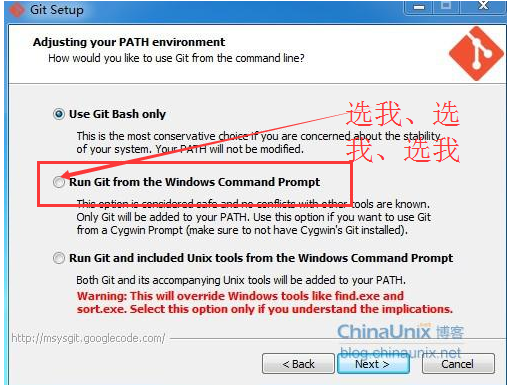
## 安装Node.js

由于项目通过Node.js插件进行构建、下载第三方js插件等工作，所以第一步要安装Node.js环境。

a、在[官网](https://nodejs.org/en/)选择最新的长期支持版本，一路next安装即可：

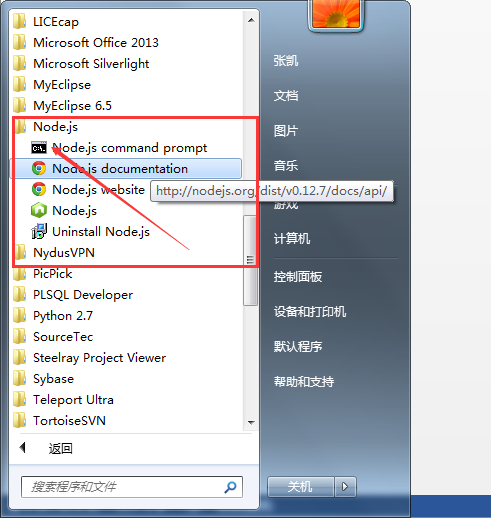


b、接下来我们安装 Git，直接百度搜索Git就有[下载地址](https://www.baidu.com/s?ie=utf-8&f=8&rsv_bp=1&rsv_idx=1&tn=baidu&wd=git&oq=nrm&rsv_pq=c35202bd0001eec9&rsv_t=b03cEgV1rjBAVI%2B0nm%2FI6YV9dUTidf0jeUq49H4VP0UJK06N5kmU2Dm4tPQ&rsv_enter=1&inputT=1012&rsv_sug3=10&rsv_sug1=10&rsv_sug7=100&rsv_sug2=0&rsv_sug4=1012)。安装时需要**特别注意：**环境变量选择“Run Git from the Windows Command Prompt”



安装完后需要**重启**电脑

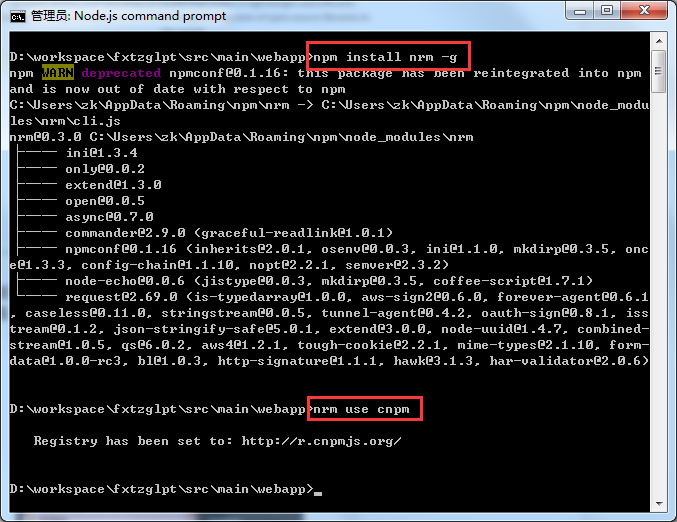
c、重启成后，在开始菜单打开Node.js命令行（**注意：不是windows的cmd！**），以后我们大部分操作都在Node.js命令行下进行：



d、（此步非必须）Node.js通过npm管理包下载插件，其默认服务器在国内访问比较卡顿，可以通过下载 [nrm](http://www.tuicool.com/articles/nYjqeu) 切换到较快的服务器

npm install nrm –g

nrm use cnpm



e、接下来我们需要通过npm 安装 gulp、http-server、bower（如果之前已经通过nrm改变npm源，速度会快很多）

npm install gulp –g

npm install http-server –g

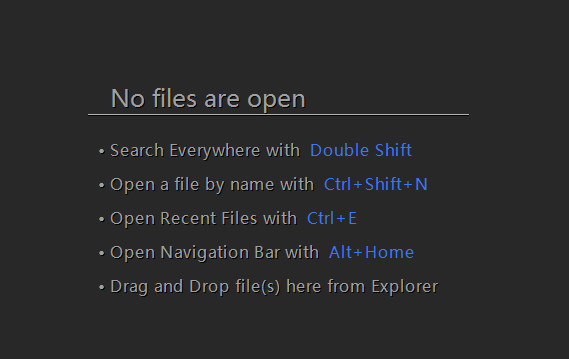
npm install bower –g

所有安装命令执行完后，Node.js安装部分就基本上完成了。

## 2、Webstorm安装及配置

a、百度下载webstrom10破解版，一路next安装。

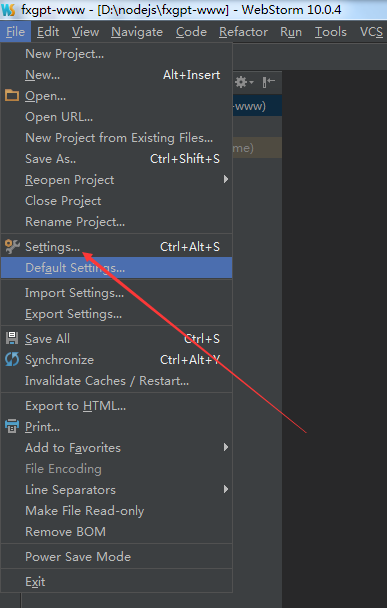
b、webstorm的快捷键和以前使用的Eclipse有很大不同。安装完成后在空白首页已经显示了常用的快捷键：



其他快捷键还有：

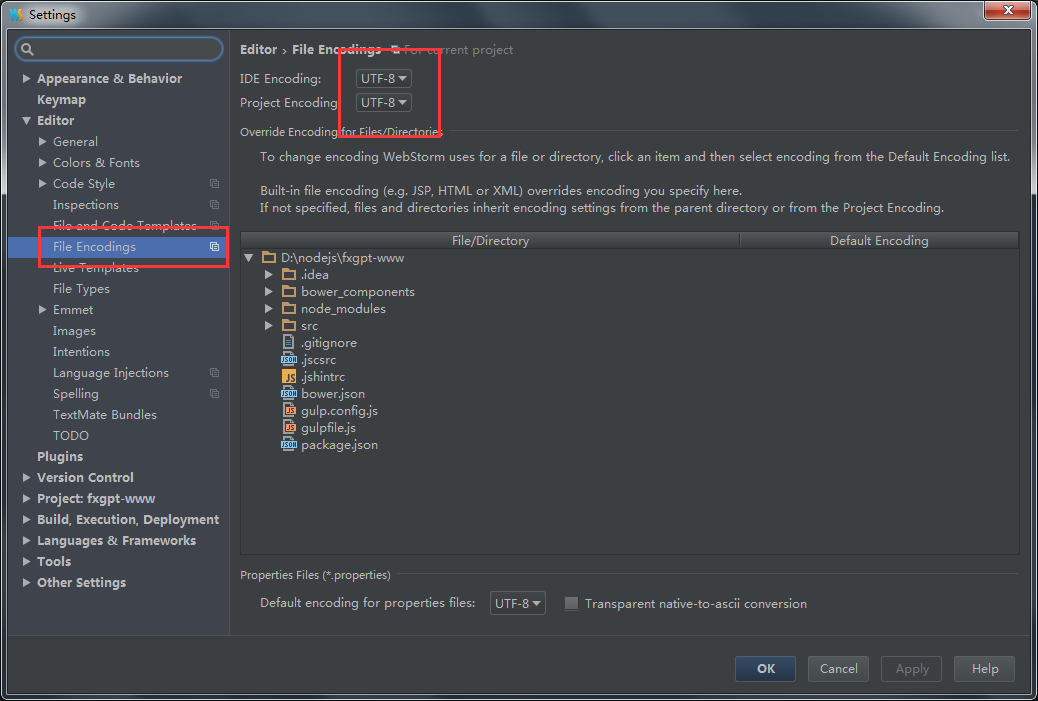
重命名 -> Shift + F6

c、安装完后需要对webstorm进行配置

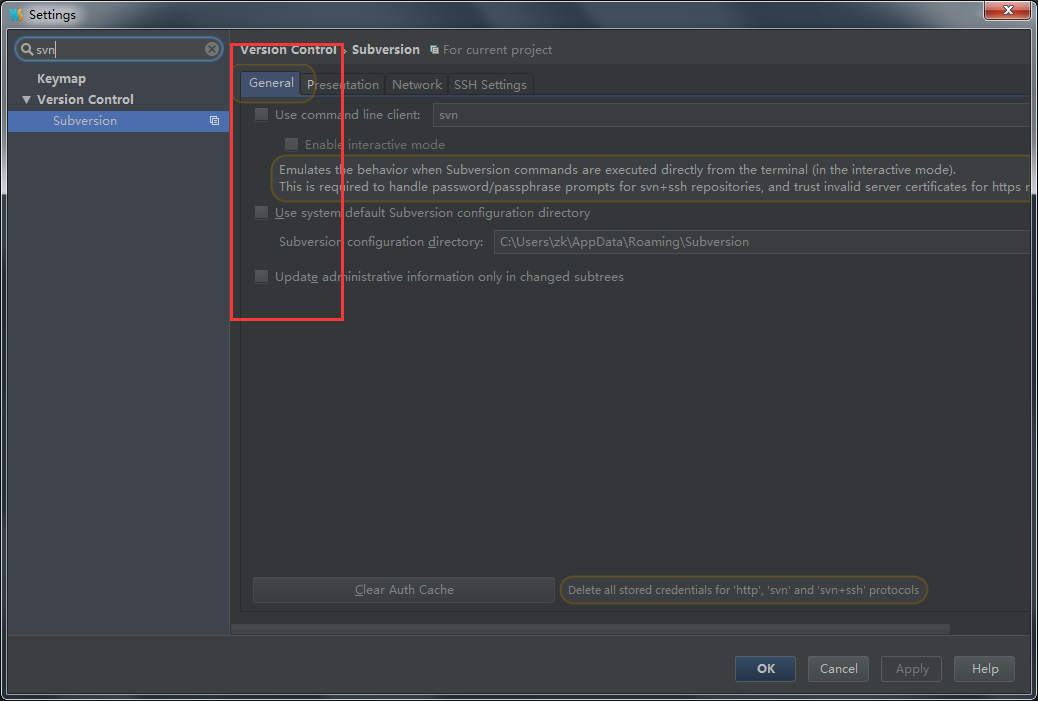


webstrom的设置分为Settings和Default Settings，最好2个都设置一下，不然新建工程后又会恢复到默认设置。

d、设置文件编码。webstom默认文件编码为：GBK，需要设置成 UTF-8



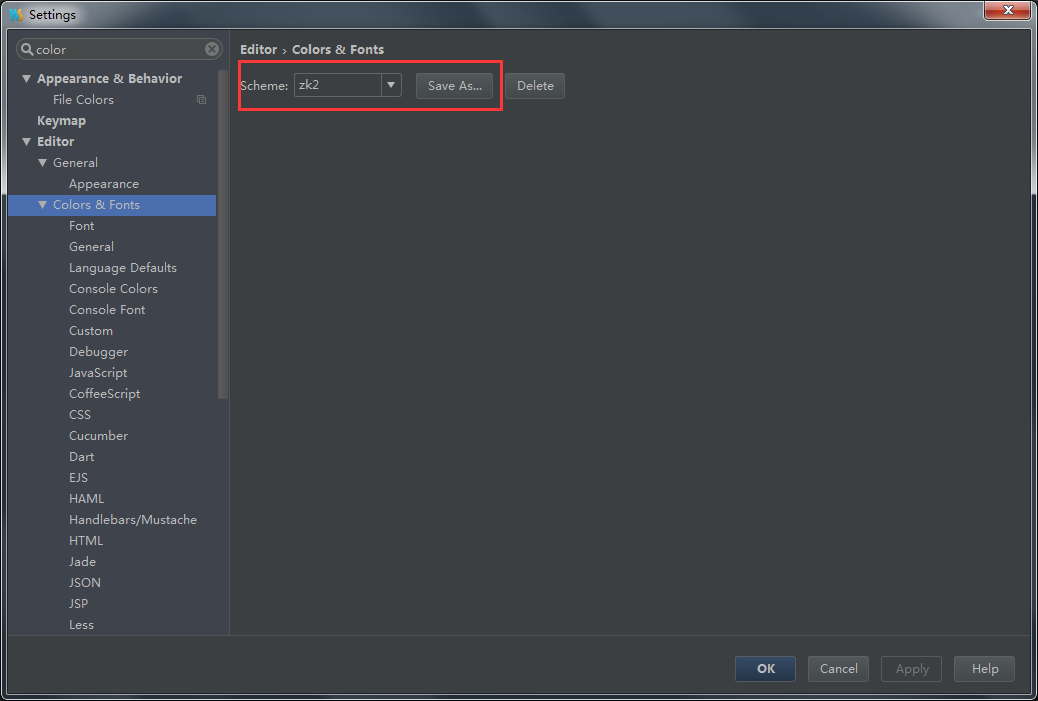
e、设置svn

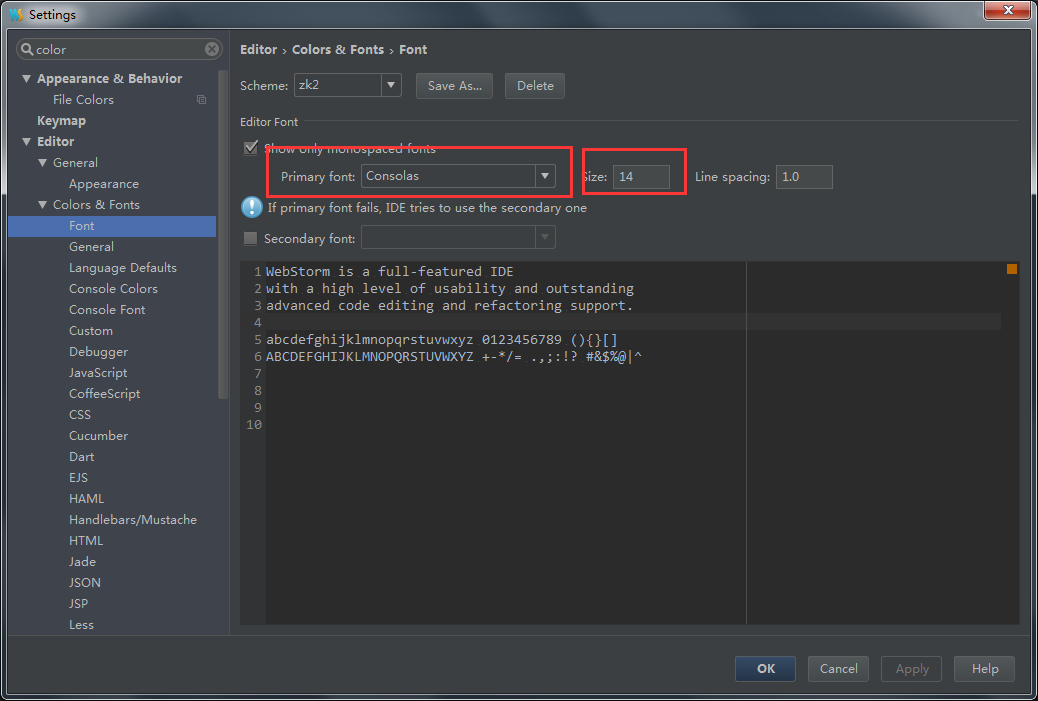


把钩都去掉，不然以后通过svn下载会报错

f、设置字体大小。

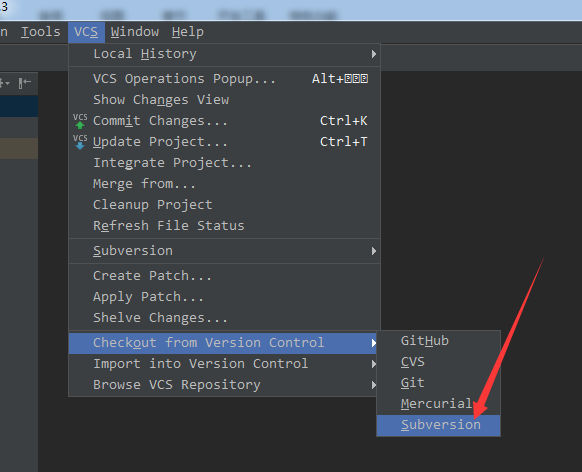
先另存为当前主题，起一个新名字之后就可以设置字体大小了。



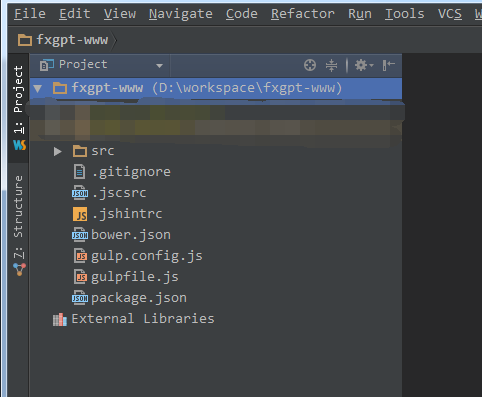


## 下载工程文件

a、Svn地址: <https://fileserver.centit.com/svn/centit/otherindustries/tzglfx/fxgpt-www>



下载完成后后，工程文件结构如下：

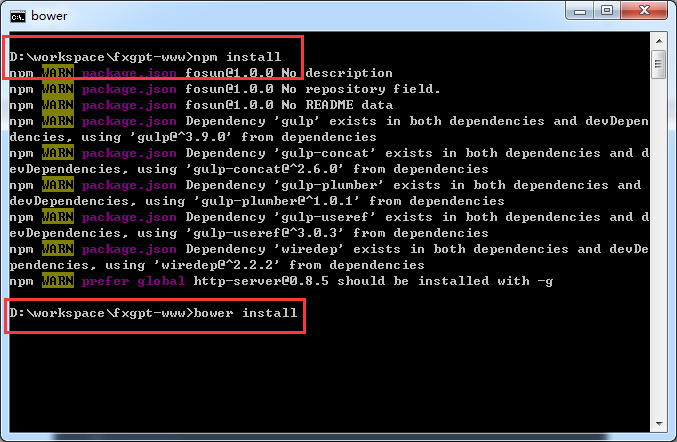


b、如果是首次下载工程需要安装工程依赖的第三方依赖，打开Node.js命令行，**将当前路径移动到工程路径下**（本例中为 D:\workspace\fxgpt-www），分别执行

npm install

这个过程根据网络条件不同，耗时会比较长，请耐心等待

bower install



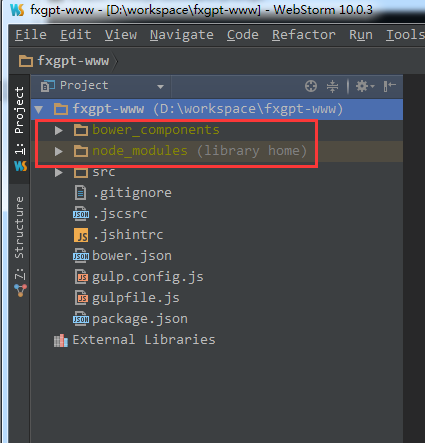
在bower安装时，会提示选择一些js插件的版本需要注意：

**Bootstrap选择2.3.2**

**AngularJS选择1.4.8或者1.4.9**

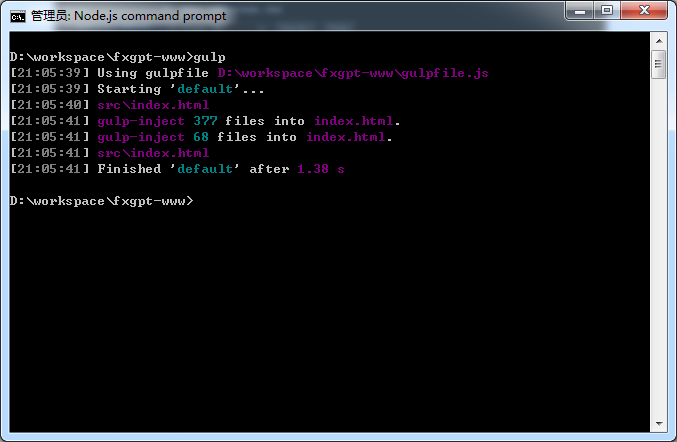
其他选择最新版本即可

安装完后，工程中会多出2个文件夹：



c、步骤b只在**首次**下载工程需要执行，实际上每次更新完代码，只需要执行以下步骤：

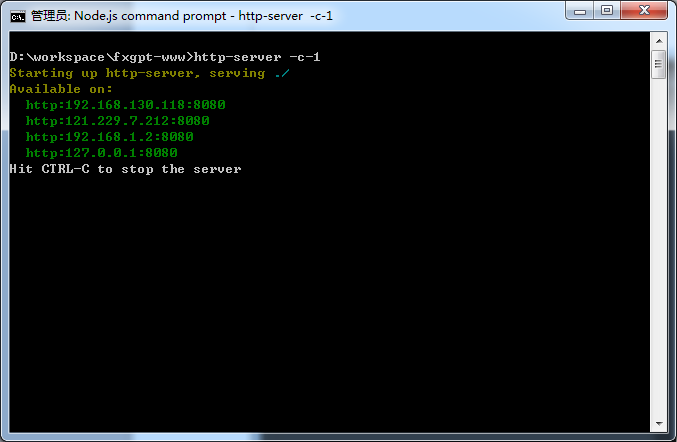
gulp



gulp的目的是将工程中的js和css以及所依赖的第三方插件加入到首页index.html中。

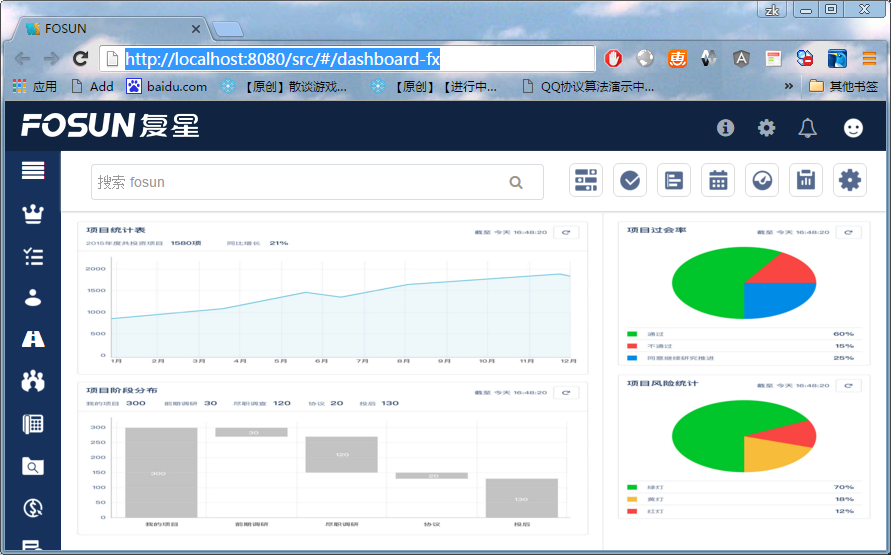
加入后就可以启动服务器

http-server -c-1

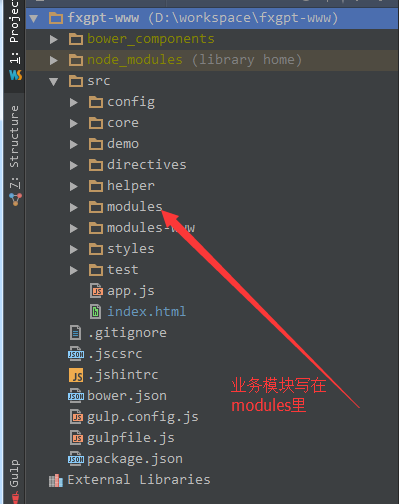


默认8080端口

打开浏览器访问地址：<http://localhost:8080/src/> 就能看到首页了



# 目录结构

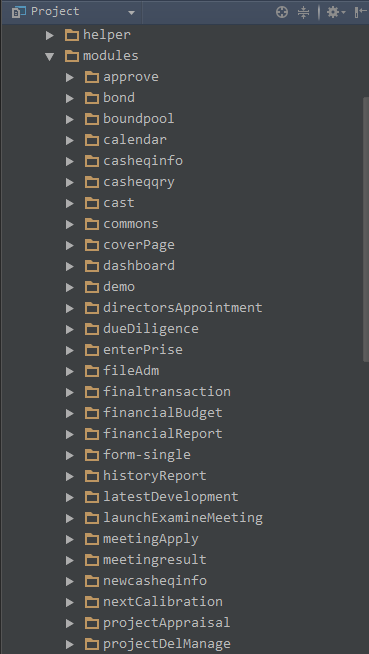


|  |  |
| --- | --- |
| Config | 插件配置\* |
| Core | 核心模块\* |
| Demo | 一些页面示例\* |
| Directives | 第三方插件\* |
| Helper | 通用配置\* |
| Modules | 业务文件 |
| Styles | 样式文件 |
| Test | 测试文件\* |
| App.js | 工程模块定义\* |
| Index.html | 首页\* |

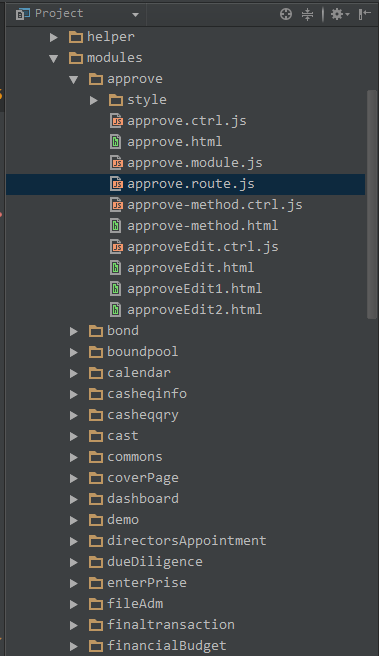
开发者只需要在modules和styles添加业务文件和样式，其他文件夹一般情况下无需修改。

## modules业务模块

里面每一个业务对应一个文件夹，例如：



在每一个业务文件夹里，虽然根据业务不同文件也不同，但有几个文件是必须有的，以appprove（审批）为例：

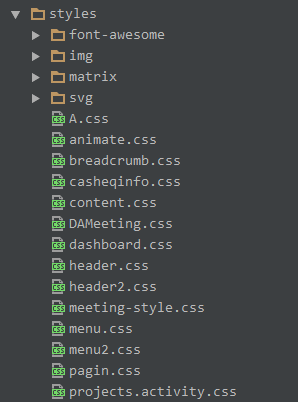


|  |  |
| --- | --- |
| approve.module.js | 模块定义\* |
| approve.route.js | 路由 |
| approve.service.js | API |
| approve.list.html | 页面 |
| approve.list.ctrl.js | 控制器 |

其中“模块定义”是必须要有的，“路由”和“API”有且各有一个，“页面”和“控制器”根据实际需求会有多对。

在后面会详细介绍这些文件的具体用途。

## styles 样式

 **图片不是最新的，以文字描述为准**

|  |  |
| --- | --- |
| images | 图片文件夹 |
| fonts | 字体图片 |
| svg | Svg图片 |
| style.css | 通用样式文件 |
| approve.css | 业务样式文件 |

一般不需要添加特殊的业务样式文件，如果真有需求，命名和modules中保持一致。

# 命名规范

## 文件命名

1. 要不要加s？

这是一个比较纠结的问题，比如：是logs还是log？是projects还是project？

考虑到一些英文单词复数形式比较特殊，我们统一使用**不加S的小写英文单词**

b、模块定义

|  |  |
| --- | --- |
| \*.module.js | \*为业务名词 |

c、路由

|  |  |
| --- | --- |
| \*.route.js | \*为业务名词 |

d、API

|  |  |
| --- | --- |
| \*.service.js | \*为业务名词 |

e、页面

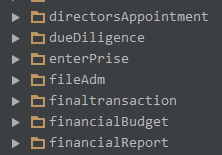
|  |  |
| --- | --- |
| \*.操作.html | \*为业务名词 |
| 操作根据实际需要可为动词或者名词，例如： | |
| \*.list.html | 列表页面 |
| \*.view.html | 详情页面 |
| \*.form.html | 编辑页面（新增、修改） |
| 如果新增、修改页面差异太大无法通用，也可以分开定义 | |
| \*.add.html | 新增页面 |
| \*.edit.html | 编辑页面 |

f、控制器

|  |  |
| --- | --- |
| \*.操作.ctrl.js | \*为业务名词 |
| 操作为动词： | |
| \*.list.ctrl.js | 列表控制器 |
| \*.add.ctrl.js | 新增控制器 |
| \*.edit.ctrl.js | 编辑控制器 |
| \*.view.ctrl.js | 查看详情控制器 |
| \*.remove.ctrl.js | 删除控制器 |

g、文件名中英文驼峰处理

对于变量名我们使用驼峰处理法，但是在文件命中驼峰处理法并不好看：



统一使用减号分割：

file-adm

due-diligence

## 2、变量命名

A、模块定义

总模块定义为：fosun

各业务模块定义为：fosun.\*

如果是主从关系子模块，父子关系间用.分割，例如：fosun.project.target

(**function**() {  
 'use strict';  
  
 angular.module('fosun.approve', []);  
})();

B、路由

|  |  |  |
| --- | --- | --- |
| state | root.\* | \*为业务名词 |
| url | /\* | \*为业务名词 |
|  |  |  |

如果返回的页面是列表页面，则url后的名词需要变成复数形式

state: 'root.approve',  
config: {  
 url: '/approves'

...

子路由：**切忌出现驼峰命名！（如： root.projectView 、 /projectView等）**

state中以.分割

url中以/分割

|  |  |  |  |
| --- | --- | --- | --- |
| state | url | 说明 | 浏览器显示url |
| root.project | /projects | 项目列表 | /projects |
| root.project.view | /:id/view | 项目详情页面 | /projects/1/view |
| root.project.target | /:id/targets | 项目标的列表 | /projects/1/targets |
| root.project.target.view | /:targetId/view | 项目标的详情 | /projects/1/targets/2/view |

其中参数 :targetId 由于不显示在浏览器上可以使用驼峰命名

C、API

业务名词首字母大写+API

angular  
 .module('fosun.user')  
 .factory('UserAPI', UserService);

D、控制器

控制器的名字和文件名相对应

|  |  |
| --- | --- |
| \*.list.ctrl.js | \*ListController |
| \*.add.ctrl.js | \*AddController |
| \*.edit.ctrl.js | \*EditController |
| \*.view.ctrl.js | \*ViewController |
| \*.remove.ctrl.js | \*RemoveController |

angular.module('fosun.users')  
 .controller('UserEditController', UserEditController);

## [RESTFul API设计规范](http://blog.jobbole.com/41233/)

## [Angular Style Guide](https://github.com/johnpapa/angular-styleguide)

# 开发我的第一个页面

我们以用户（user）为例，创建一个CRUD流程

## 创建业务目录

在/modules目录下创建/user目录

## 创建业务模块定义文件

在/modules/user目录下创建user.module.js

(**function**() {  
 'use strict';  
  
 angular.module('fosun.user', []);  
})();

## 将模块添加到工程中

在app.js中添加我们刚刚建好的模块名（Shift + Ctrl + N 可以快速查找文件）

angular.module('demo',  
 ['fosun.user', ... ]);

## 创建路由文件

在/modules/user目录下创建user.route.js

(**function**() {  
 'use strict';  
  
 angular.module('fosun.user')  
 .run(UserRun);  
  
 /\* @ngInject \*/  
 **function** UserRun(routerHelper) {  
 routerHelper.configureStates(getStates());  
 }  
  
 **function** getStates() {  
 **return** [  
 {  
 // 在此定义路由  
 }   
 ];  
 }  
})();

如果我们想在浏览器输入 /users 访问用户列表，在{}中定义路由

{  
 state: 'root.user',  
 config: {  
 url: '/users',  
 views: {  
 'main@': {  
 templateUrl: 'modules/user/user.list.html',  
 controller: 'UserListController as vm'  
 }  
 },  
 title: '用户列表'  
 }  
}

## 创建用户列表控制器

在/modules/user目录下创建user.list.ctrl.js

(**function**() {  
 'use strict';  
  
 angular.module('fosun.user')  
 .controller('UserListController', UserListController);  
  
 /\* @ngInject \*/  
 **function** UserListController() {  
 **var** vm = **this**;  
  
 vm.users = [  
 {id: 1, name: 'Tom', phone: '123', email: 'tom@centit.com'},  
 {id: 2, name: 'Rose', phone: '123', email: 'rose@centit.com'},  
 {id: 3, name: 'Jack', phone: '123', email: 'jack@centit.com'}  
 ];  
 }  
})();

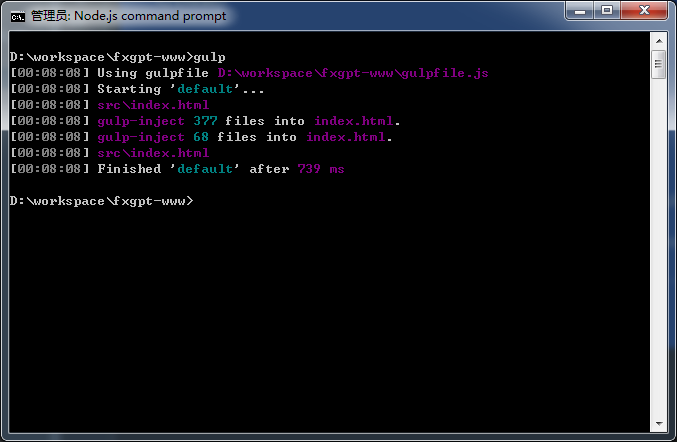
## 创建用户列表

在/modules/user目录下创建user.list.html

<table class="table">  
 <thead>  
 <tr>  
 <th>Name</th>  
 <th>Phone</th>  
 <th>Email</th>  
 </tr>  
 </thead>  
  
 <tbody>  
 <tr ng-repeat="user in vm.users">  
 <td>{{user.name}}</td>  
 <td>{{user.phone}}</td>  
 <td>{{user.email}}</td>  
 </tr>  
 </tbody>  
</table>

## 执行gulp

在Node.js命令行中执行gulp



会将我们刚刚创建的

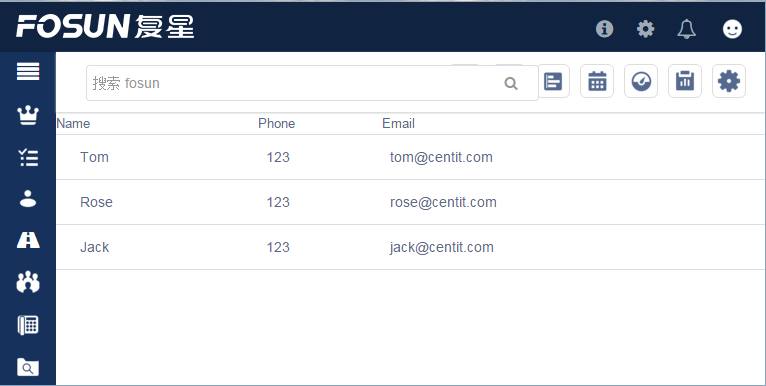
user.module.js

user.route.js

user.list.ctrl.js

自动加入到index.html中

此时在浏览器输入 <http://localhost:8080/src#/users> 就可看到数据已经显示到页面上了



# CRUD 增删改查

刚刚我们在页面上显示的用户数据是写死在控制器里的。实际中这些数据是需要通过发送ajax请求得到。在编写一个业务前，首先应该确定需要使用哪些请求API，在这里我们先假定有5个API

|  |  |  |
| --- | --- | --- |
|  | Api | Method |
| 获取用户列表 | /users | GET |
| 获取单个用户信息 | /users/:id | GET |
| 创建用户 | /users | POST |
| 修改用户信息 | /users/:id | PUT |
| 删除用户 | /users/:id | DELETE |

所有的API都是以 /users开头。

## 1、创建API文件：

在/modules/user目录下创建user.service.js

(**function**() {  
 'use strict';  
  
 angular  
 .module('fosun.user')  
 .factory('UserAPI', UserService);  
  
 /\* @ngInject \*/  
 **function** UserService (Restangular) {  
 **var** service = Restangular.service('users');  
 **return** service;  
 }  
})();

## 查询列表

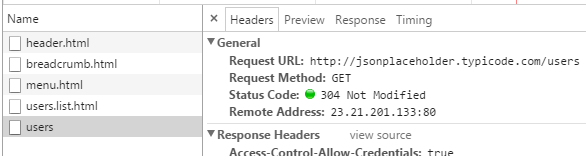
现在我们需要修改user.list.ctrl.js控制器，使用户数据通过UserAPI获取

**function** UserListController(UserAPI) {  
 **var** vm = **this**;  
  
 UserAPI  
 .getList(vm.$$params)  
 .then(**function**(data) {  
 vm.users = data;  
 });  
}

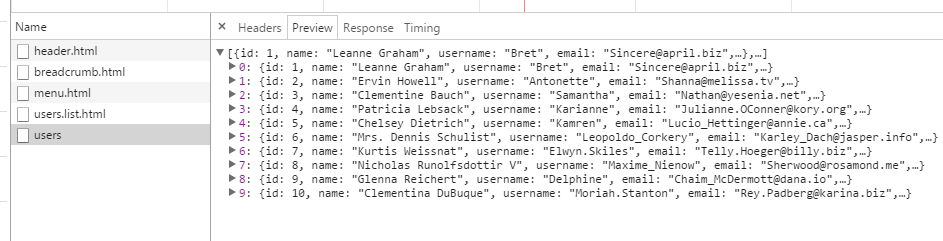
|  |  |
| --- | --- |
| getLIst() | 发送GET请求，返回结果为数组 |
| vm.$$params | 查询参数：分页、排序等 |
| then(function(data) {}) | 请求回调函数 |

**注意：当添加新的js后，需要再次执行gulp命令，后面不再重复**

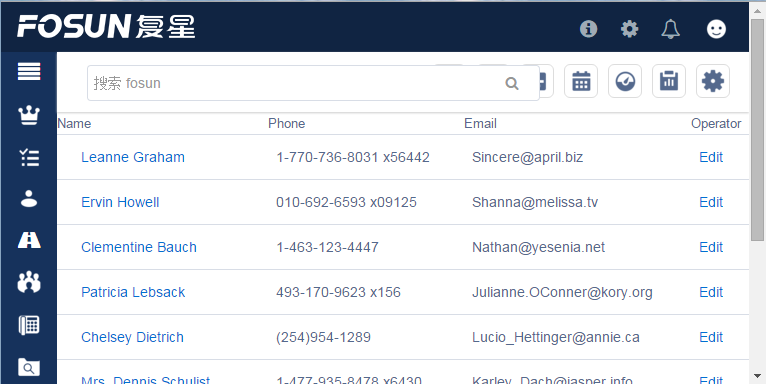
刷新页面后，浏览器发送请求 /users GET



响应



页面显示新的数据



## 查询

首先添加一个路由定义，当访问/users/1/view地址时，显示id=1的用户详情

user.route.js

{  
 state: 'root.user.view',  
 config: {  
 url: '/:id/view',  
 views: {  
 'main@': {  
 templateUrl: 'modules/user/user.view.html',  
 controller: 'UserViewController as vm'  
 }  
 },  
 title: '用户详情'  
 }  
}

同时添加详情控制器user.view.ctrl.js

(**function**() {  
 'use strict';  
  
 angular.module('fosun.user')  
 .controller('UserViewController', UserViewController);  
  
 /\* @ngInject \*/  
 **function** UserViewController($stateParams, UserAPI) {  
 **var** vm = **this**;  
  
 UserAPI.one($stateParams.id)  
 .get()  
 .then(**function**(data) {  
 vm.user = data;  
 });  
 }  
})();

|  |  |
| --- | --- |
| $stateParams | 根据实际url获取路由中定义的参数：  路由定义 /users/:id  实际URL /users/1  $stateParams.id = 1 |
| one(id) | 拼接url |
| get() | 发送GET请求，返回结果为对象 |
|  |  |

接下来将说明如何从列表页面跳转到详情页面。假设我们将这个操作添加到用户名上，点击用户名，跳转到对应的用户详情页面。

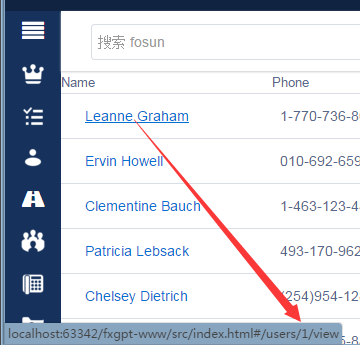
首先，我们要改动user.list.html对td中的用户名进行一点修改

<td>  
 <a ui-sref="root.users.view({id: user.id})">{{user.name}}</a>  
</td>

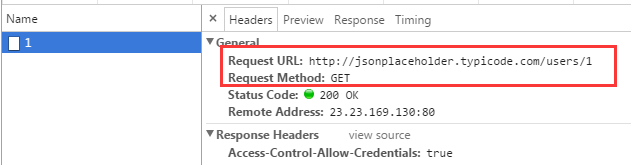
ui-sref是路由自带的一个指令，跳转到指定的路由。

在上面我们跳转到了用户详情路由的state，并传入用户id作为参数

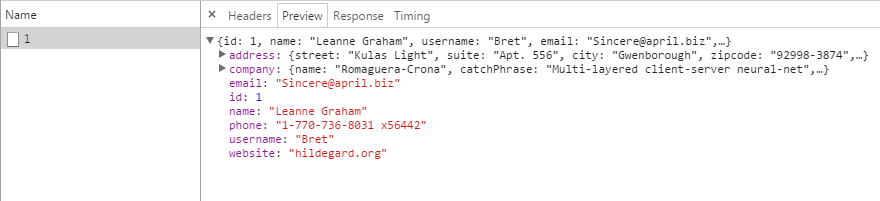
当我们把鼠标移动到用户名上，发现指令已经自动把正确的href加到了a标签上：



点击后发送请求 /users/1 GET



响应



让我们把这些值显示到页面上吧：

user.view.html

<div class="fluid-container fx-container">  
 <p>  
 <b>Name: </b> {{vm.user.name}}  
 </p>  
 <p>  
 <b>Phone: </b> {{vm.user.phone}}  
 </p>  
 <p>  
 <b>Email: </b> {{vm.user.email}}  
 </p>  
</div>



## 编辑

编辑和查询用户信息的写法基本类似

添加路由定义 user.route.js

{  
 state: 'root.user.edit',  
 config: {  
 url: '/:id/edit',  
 views: {  
 'main@': {  
 templateUrl: 'modules/user/user.form.html',  
 controller: 'UserEditController as vm'  
 }  
 },  
 title: '用户编辑'  
 }  
}

控制器也很类似，只是添加了一个保存用户的方法

user.edit.ctrl.js

(**function**() {  
 'use strict';  
  
 angular.module('fosun.user')  
 .controller('UserEditController', UserEditController);  
  
 /\* @ngInject \*/  
 **function** UserEditController($state, $stateParams, UserAPI) {  
 **var** vm = **this**;  
  
 UserAPI.one($stateParams.id)  
 .get()  
 .then(**function**(data) {  
 vm.user = data;  
 });  
  
 // 保存用户  
 vm.saveUser = **function**() {  
 vm.user

.put()

.then(function() {

$state.go('root.user');

});  
 }  
 }  
})();

|  |  |
| --- | --- |
| .put() | 发送PUT请求 |
| $state.go() | 跳转到指定路由（比如编辑成功后返回列表页面） |

在编辑页面，添加一个保存按钮

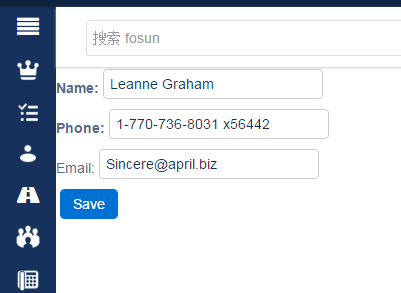
user.form.html

<div class="fluid-container fx-container">  
 <p>  
 <b>Name: </b>

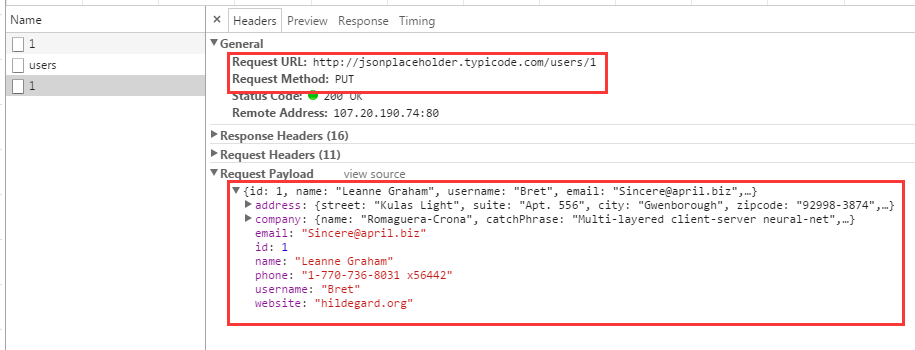
<input type="text" ng-model="vm.user.name"/>  
 </p>  
 <p>  
 <b>Phone: </b>

<input type="text" ng-model="vm.user.phone"/>  
 </p>  
 <p>  
 <bl>Email: </bl>

<input type="text" ng-model="vm.user.email"/>  
 </p>  
  
 <a class="btn btn-primary" ng-click="vm.saveUser()">Save</a>  
</div>



点击保存按钮，发送请求 /users POST



## 新增

在这个示例中新增和编辑可以用一个页面user.form.html

首先还是添加路由user.route.js

{  
 state: 'root.users.add',  
 config: {  
 url: '/add',  
 views: {  
 'main@': {  
 templateUrl: 'modules/user/user.form.html',  
 controller: 'UsersAddController as vm'  
 }  
 },  
 title: '用户新增'  
 }  
}

添加新增用户控制器user.add.ctrl.js

(**function**() {  
 'use strict';  
  
 angular.module('fosun.user')  
 .controller('UserAddController', UserAddController);  
  
 /\* @ngInject \*/  
 **function** UserAddController($state, UserAPI) {  
 **var** vm = **this**;  
  
 vm.user = {};  
  
 // 保存用户  
 vm.saveUser = **function**() {  
 UserAPI.post(vm.user)

.then(function() {

$state.go('root.user');

})  
 }  
 }  
})();

|  |  |
| --- | --- |
| .post(user) | 发送POST请求 |

## 删除

删除不需要跳转页面，所以不用再定义路由。

在列表页面添加“删除”操作按钮，user.list.html

<tr ng-repeat="user in vm.users">  
 <td>  
 <a ui-sref="root.users.view({id:user.id})">

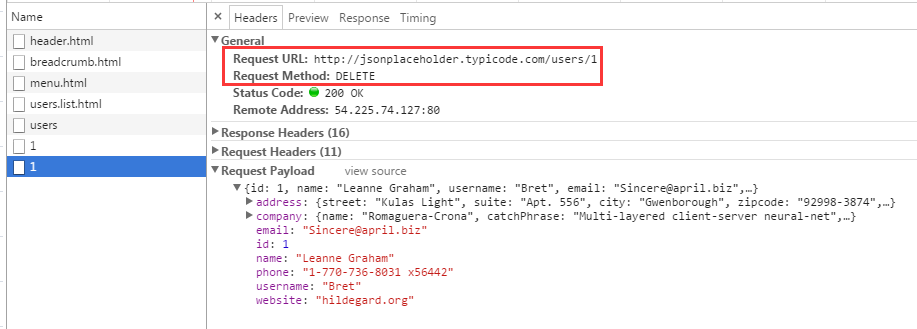
{{user.name}}

</a>  
 </td>  
 <td>{{user.phone}}</td>  
 <td>{{user.email}}</td>  
 <td>  
 <a ui-sref="root.users.edit({id: user.id})">Edit</a>  
  
 <a ng-click="vm.removeUser(user)">Delete</a>  
 </td>  
</tr>

列表控制器中添加removeUser方法，user.list.ctrl.js

// 删除用户  
vm.removeUser = **function**(user) {  
 user.remove()  
 .then(**function**() {  
 alert('删除成功')  
 });  
}

点击后发送 /users/1 DELETE 请求



# 在弹出窗口中进行操作

# 数据校验

# 其他场景

# 常用组件