第9章 权限控制、图形报表

# 第一章. 在项目中使用Spring Security

前面我们已经学习了Spring Security框架的使用方法，本章节我们就需要将Spring Security框架应用到后台系统中进行权限控制，其本质就是认证和授权。

要进行认证和授权需要前面课程中提到的权限模型涉及的7张表支撑，因为用户信息、权限信息、菜单信息、角色信息、关联信息等都保存在这7张表中，也就是这些表中的数据是我们进行认证和授权的依据。所以在真正进行认证和授权之前需要对这些数据进行管理，即我们需要开发如下一些功能：

1、用户数据管理（增删改查、用户关联角色）

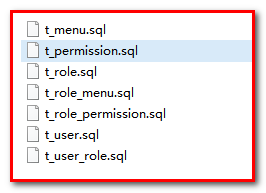
2、角色数据管理（增删改查、角色关联权限、角色关联菜单）

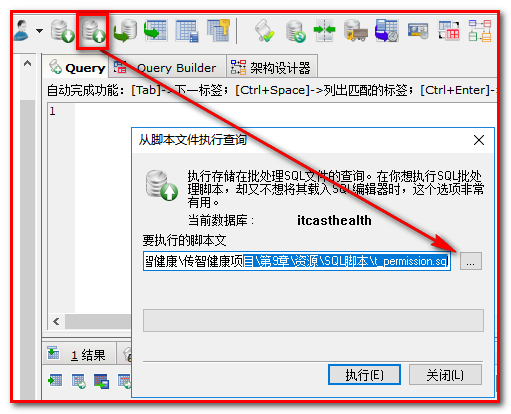
3、权限数据管理（增删改查）

4、菜单数据管理（增删改查）

鉴于时间关系，我们不再实现这些数据管理的代码开发。我们可以直接将数据导入到数据库中即可。

## 导入用户、角色、权限、菜单的初始数据





## 导入Spring Security环境

### 第一步：pom.xml导入坐标

在health\_parent父工程的pom.xml中导入Spring Security的maven坐标（已经引入）

<dependency>  
  <groupId>org.springframework.security</groupId>  
  <artifactId>spring-security-web</artifactId>  
  <version>${spring.security.version}</version>  
</dependency>  
<dependency>  
  <groupId>org.springframework.security</groupId>  
  <artifactId>spring-security-config</artifactId>  
  <version>${spring.security.version}</version>  
</dependency>

### 第二步：web.xml添加代理过滤器

在health\_web工程的web.xml文件中配置用于整合Spring Security框架的过滤器DelegatingFilterProxy

<**filter**>  
 *<!--  
 DelegatingFilterProxy用于整合第三方框架（代理过滤器，非真正的过滤器，真正的过滤器需要在spring的配置文件）  
 整合Spring Security时过滤器的名称必须为springSecurityFilterChain，  
 否则会抛出NoSuchBeanDefinitionException异常  
 -->* <**filter-name**>springSecurityFilterChain</**filter-name**>  
 <**filter-class**>org.springframework.web.filter.DelegatingFilterProxy</**filter-class**>  
</**filter**>  
<**filter-mapping**>  
 <**filter-name**>springSecurityFilterChain</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
</**filter-mapping**>

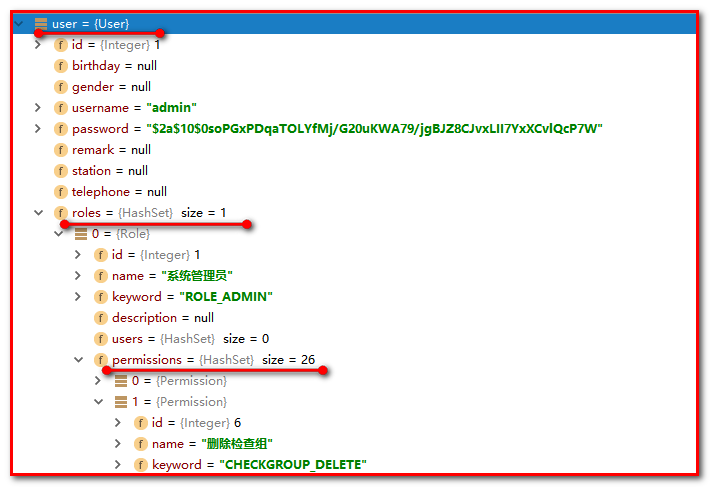
## 实现认证和授权

### 第一步：SpringSecurityUserService.java

在health\_web工程中按照Spring Security框架要求提供SpringSecurityUserService，并且实现UserDetailsService接口

**package** com.itheima.health.security;  
  
**import** com.alibaba.dubbo.config.annotation.Reference;  
**import** com.itheima.health.pojo.Permission;  
**import** com.itheima.health.pojo.Role;  
**import** com.itheima.health.service.UserService;  
**import** org.springframework.security.core.GrantedAuthority;  
**import** org.springframework.security.core.authority.SimpleGrantedAuthority;  
**import** org.springframework.security.core.userdetails.User;  
**import** org.springframework.security.core.userdetails.UserDetails;  
**import** org.springframework.security.core.userdetails.UserDetailsService;  
**import** org.springframework.security.core.userdetails.UsernameNotFoundException;  
**import** org.springframework.stereotype.Component;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Set;  
  
  
@Component  
**public class** SpringSecurityUserService **implements** UserDetailsService {  
  
 @Reference *//注意：此处要通过dubbo远程调用用户服务* **private** UserService **userService**;  
  
 *//根据用户名查询用户信息* **public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {  
 *//远程调用用户服务，根据用户名查询用户信息* com.itheima.health.pojo.User user = **userService**.findUserByUsername(username);  
 **if**(user == **null**){  
 *//用户名不存在，抛出异常UsernameNotFoundException* **return null**;  
 }  
 List<GrantedAuthority> list = **new** ArrayList<GrantedAuthority>();  
 Set<Role> roles = user.getRoles();  
 **for**(Role role : roles){  
 Set<Permission> permissions = role.getPermissions();  
 **for**(Permission permission : permissions){  
 *//授权* list.add(**new** SimpleGrantedAuthority(permission.getKeyword()));  
 }  
 }  
 */\*\*  
 \* User()  
 \* 1：指定用户名  
 \* 2：指定密码（SpringSecurity会自动对密码进行校验）  
 \* 3：传递授予的角色和权限  
 \*/* UserDetails userDetails = **new** User(username,user.getPassword(),list);  
 **return** userDetails;  
 }  
}

使用debug跟踪调试，查看user。



### 第二步：Service、Dao接口、Mapper映射文件

创建UserService服务接口、服务实现类、Dao接口、Mapper映射文件等

1：服务接口

**package** com.itheima.health.service;  
  
**import** com.itheima.health.pojo.User;  
  
*/\*\*  
 \* 用户服务接口  
 \*/***public interface** UserService {  
  
 User findUserByUsername(String username);  
}

2：服务实现类

**package** com.itheima.health.service;  
  
**import** com.alibaba.dubbo.config.annotation.Service;  
**import** com.itheima.health.dao.UserDao;  
**import** com.itheima.health.pojo.User;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.transaction.annotation.Transactional;  
  
  
@Service(interfaceClass = UserService.**class**)  
@Transactional  
**public class** UserServiceImpl **implements** UserService {  
  
 @Autowired  
 **private** UserDao **userDao**;  
  
 **public** User findUserByUsername(String username) {  
 User user = **userDao**.findUserByUsername(username);  
 **return** user;  
 }  
}

3：Dao接口

1. UserDao

**package** com.itheima.health.dao;  
  
**import** com.itheima.health.pojo.User;  
**import** org.springframework.stereotype.Repository;  
  
*/\*\*  
 \* 持久层Dao接口  
 \*/*@Repository  
**public interface** UserDao {  
  
 User findUserByUsername(String username);  
}

1. RoleDao

**package** com.itheima.health.dao;  
  
**import** com.itheima.health.pojo.Role;  
**import** org.springframework.stereotype.Repository;  
  
**import** java.util.Set;  
  
*/\*\*  
 \* 持久层Dao接口  
 \*/*@Repository  
**public interface** RoleDao {  
  
 Set<Role> findRolesByUserId(Integer userId);

}

1. PermissionDao

**package** com.itheima.health.dao;  
  
**import** com.itheima.health.pojo.Permission;  
**import** org.springframework.stereotype.Repository;  
  
**import** java.util.Set;  
  
*/\*\*  
 \* 持久层Dao接口  
 \*/*@Repository  
**public interface** PermissionDao {  
  
 Set<Permission> findPermissionsByRoleId(Integer roleId);

}

4：Mapper映射文件

1. UserDao.xml

*<?***xml version="1.0" encoding="UTF-8"** *?>***<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
 "http://mybatis.org/dtd/mybatis-3-mapper.dtd" *>***<**mapper namespace="com.itheima.health.dao.UserDao"**>  
  
 <**resultMap id="userMap" type="com.itheima.health.pojo.User"**>  
 <**id column="id" property="id"**></**id**>  
 <**collection property="roles" column="id" select="com.itheima.health.dao.RoleDao.findRolesByUserId"**></**collection**>  
 </**resultMap**>  
 <**select id="findUserByUsername" parameterType="string" resultMap="userMap"**>  
 select u.*\** from t\_user u where username = #{username}  
 </**select**>  
   
</**mapper**>

1. RoleDao.xml

*<?***xml version="1.0" encoding="UTF-8"** *?>***<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
 "http://mybatis.org/dtd/mybatis-3-mapper.dtd" *>***<**mapper namespace="com.itheima.health.dao.RoleDao"**>  
  
 <**resultMap id="roleMap" type="com.itheima.health.pojo.Role"**>  
 <**id column="id" property="id"**></**id**>  
 <**collection property="permissions" column="id" select="com.itheima.health.dao.PermissionDao.findPermissionsByRoleId"**></**collection**>  
 </**resultMap**>  
 <**select id="findRolesByUserId" parameterType="int" resultMap="roleMap"**>  
 SELECT r.*\** FROM t\_role r,t\_user\_role ur WHERE r.id = ur.role\_id AND ur.user\_id = #{userId}  
 </**select**>  
</**mapper**>

1. PermissionDao.xml

*<?***xml version="1.0" encoding="UTF-8"** *?>***<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
 "http://mybatis.org/dtd/mybatis-3-mapper.dtd" *>***<**mapper namespace="com.itheima.health.dao.PermissionDao"**>  
  
 <**select id="findPermissionsByRoleId" parameterType="int" resultType="com.itheima.health.pojo.Permission"**>  
 SELECT p.*\** FROM t\_permission p,t\_role\_permission rp WHERE p.id = rp.permission\_id AND rp.role\_id = #{roleId}  
 </**select**>  
</**mapper**>

### 第三步：springmvc.xml

修改health\_web工程中的springmvc.xml文件，修改dubbo批量扫描的包路径

之前的包扫描

*<!--批量扫描-->*<**dubbo:annotation package="com.itheima.health.controller"** />

**现在的包扫描**

*<!--批量扫描-->*<**dubbo:annotation package="com.itheima.health"** />

**注意：**此处原来扫描的包为com.itheima.controller，现在改为com.itheima包的目的是需要将我们上面定义的SpringSecurityUserService也扫描到，因为在SpringSecurityUserService的loadUserByUsername方法中需要通过dubbo远程调用名称为UserService的服务。

### 第四步：spring-security.xml

在health\_web工程中提供spring-security.xml配置文件

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xmlns:security="http://www.springframework.org/schema/security"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/mvc  
 http://www.springframework.org/schema/mvc/spring-mvc.xsd  
 http://code.alibabatech.com/schema/dubbo  
 http://code.alibabatech.com/schema/dubbo/dubbo.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context.xsd  
 http://www.springframework.org/schema/security  
 http://www.springframework.org/schema/security/spring-security.xsd"**>  
 *<!--一：定义哪些链接可以放行-->  
 <!--  
 http：用于定义相关权限控制  
 指定哪些资源不需要进行权限校验，可以使用通配符  
 -->* <**security:http security="none" pattern="/js/\*\*"** />  
 <**security:http security="none" pattern="/css/\*\*"** />  
 <**security:http security="none" pattern="/img/\*\*"** />  
 <**security:http security="none" pattern="/plugins/\*\*"** />  
  
 *<!--开启注解方式权限控制-->* <**security:global-method-security pre-post-annotations="enabled"** />  
 *<!--  
 二：定义哪些链接不可以放行，即需要有角色、权限才可以放行  
 http：用于定义相关权限控制  
 auto-config：是否自动配置  
 设置为true时框架会提供默认的一些配置，例如提供默认的登录页面、登出处理等  
 设置为false时需要显示提供登录表单配置，否则会报错  
 use-expressions：用于指定intercept-url中的access属性是否使用表达式  
 -->* <**security:http auto-config="true" use-expressions="true"**>  
  
  
 *<!--只要认证通过就可以访问-->  
 <!--  
 intercept-url：定义一个拦截规则  
 pattern：对哪些url进行权限控制  
 access：在请求对应的URL时需要什么权限，默认配置时它应该是一个以逗号分隔的角色列表，  
 请求的用户只需拥有其中的一个角色就能成功访问对应的URL  
 isAuthenticated()：需要经过认证后才能访问（不是匿名用户）  
 -->* <**security:intercept-url pattern="/pages/\*\*" access="isAuthenticated()"** />  
  
 *<!--  
 form-login：定义表单登录信息  
 -->* <**security:form-login login-page="/login.html"  
 username-parameter="username"  
 password-parameter="password"  
 login-processing-url="/login.do"  
 default-target-url="/pages/main.html"  
 authentication-failure-url="/login.html"  
 always-use-default-target="true"** />  
  
 *<!--  
 csrf：对应CsrfFilter过滤器  
 disabled：是否启用CsrfFilter过滤器，如果使用自定义登录页面需要关闭此项，否则登录操作会被禁用（403）  
 -->* <**security:csrf disabled="true"**></**security:csrf**>  
  
 *<!--  
 logout：退出登录  
 logout-url：退出登录操作对应的请求路径  
 logout-success-url：退出登录后的跳转页面  
 -->* <**security:logout logout-url="/logout.do"  
 logout-success-url="/login.html" invalidate-session="true"**/>  
  
  
  
 </**security:http**>  
  
  
 *<!--配置密码加密对象-->* <**bean id="passwordEncoder"  
 class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"** />  
  
  
 *<!--  
 三：认证管理，定义登录账号名和密码，并授予访问的角色、权限  
 authentication-manager：认证管理器，用于处理认证操作  
 -->* <**security:authentication-manager**>  
 *<!--  
 authentication-provider：认证提供者，执行具体的认证逻辑  
 -->* <**security:authentication-provider user-service-ref="springSecurityUserService"**>  
 *<!--指定密码加密策略-->* <**security:password-encoder ref="passwordEncoder"**></**security:password-encoder**>  
 </**security:authentication-provider**>  
 </**security:authentication-manager**>  
  
  
</**beans**>

这里注意：如果出现以下问题



使用下面的配置，在spring-security.xml中添加。

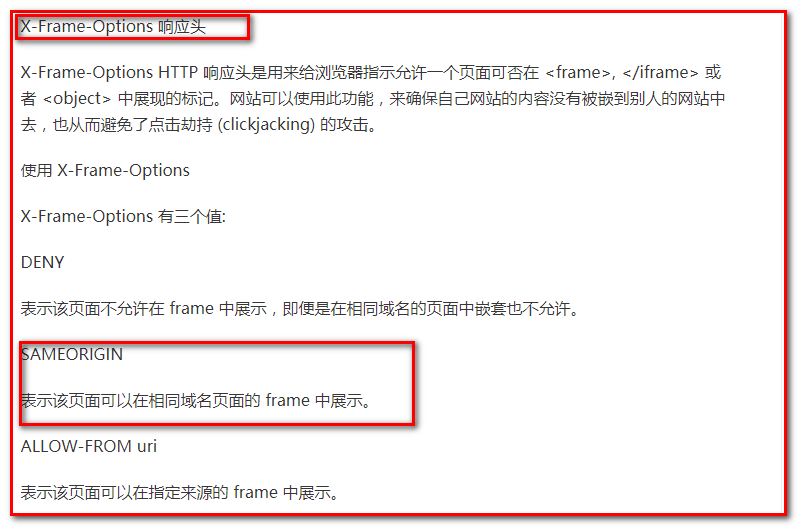
放置到<security:http auto-config="true" use-expressions="true">里面

<**security:headers**>  
 *<!--设置在页面可以通过iframe访问受保护的页面，默认为不允许访问-->* <**security:frame-options policy="SAMEORIGIN"**></**security:frame-options**>  
</**security:headers**>

是因为我们在main.html中定义：如果不配置springSecurity会认为iframe访问的html页面是收保护的页面，不允许访问。

<**el-container**>  
 <**iframe name="right" class="el-main" src="checkitem.html" width="100%" height="580px" frameborder="0"**></**iframe**>  
</**el-container**>

备注：



### 第五步：springmvc.xml

在springmvc.xml文件中引入spring-security.xml文件

<**import resource="classpath:spring-security.xml"**></**import**>

### 第六步：Controller类

在Controller的方法上加入权限控制注解，此处以CheckItemController为例

**package** com.itheima.health.controller;  
  
**import** com.alibaba.dubbo.config.annotation.Reference;  
**import** com.itheima.health.constant.MessageConstant;  
**import** com.itheima.health.entity.PageResult;  
**import** com.itheima.health.entity.QueryPageBean;  
**import** com.itheima.health.entity.Result;  
**import** com.itheima.health.pojo.CheckItem;  
**import** com.itheima.health.service.CheckItemService;  
**import** org.springframework.security.access.prepost.PreAuthorize;  
**import** org.springframework.web.bind.annotation.RequestBody;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
  
**import** java.util.List;  
  
*/\*\*  
 \* 体检检查项管理  
 \*/*@RestController  
@RequestMapping(**"/checkitem"**)  
**public class** CheckItemController {  
 @Reference  
 **private** CheckItemService **checkItemService**;  
  
 *//新增* @RequestMapping(**"/add"**)  
 @PreAuthorize(**"hasAuthority('CHECKITEM\_ADD')"**)*//权限校验* **public** Result add(@RequestBody CheckItem checkItem){  
 **try** {  
 **checkItemService**.add(checkItem);  
 }**catch** (Exception e){  
 **return new** Result(**false**, MessageConstant.***ADD\_CHECKITEM\_FAIL***);  
 }  
 **return new** Result(**true**,MessageConstant.***ADD\_CHECKITEM\_SUCCESS***);  
 }  
  
 *//分页查询* @RequestMapping(**"/findPage"**)  
 @PreAuthorize(**"hasAuthority('CHECKITEM\_QUERY')"**)*//权限校验* **public** PageResult findPage(@RequestBody QueryPageBean queryPageBean){  
 PageResult pageResult = **checkItemService**.pageQuery(  
 queryPageBean.getCurrentPage(),  
 queryPageBean.getPageSize(),  
 queryPageBean.getQueryString());  
 **return** pageResult;  
 }  
  
  
 *//删除* @RequestMapping(**"/delete"**)  
 @PreAuthorize(**"hasAuthority('CHECKITEM\_DELETE')"**)*//权限校验，使用***CHECKITEM\_DELETE123测试****public** Result delete(Integer id){  
 **try** {  
 **checkItemService**.delete(id);  
 }**catch** (RuntimeException e){  
 **return new** Result(**false**,e.getMessage());  
 }**catch** (Exception e){  
 **return new** Result(**false**, MessageConstant.***DELETE\_CHECKITEM\_FAIL***);  
 }  
 **return new** Result(**true**,MessageConstant.***DELETE\_CHECKITEM\_SUCCESS***);  
 }  
  
 *// 跳转到检查项编辑页面* @RequestMapping(**"/findById"**)  
 **public** Result findById(Integer id){  
 **try** {  
 CheckItem checkItem = **checkItemService**.findById(id);  
 **return new** Result(**true**,MessageConstant.***QUERY\_CHECKITEM\_SUCCESS***,checkItem);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return new** Result(**false**,MessageConstant.***QUERY\_CHECKITEM\_FAIL***);  
 }  
 }  
  
  
 *//编辑保存检查项* @RequestMapping(**"/edit"**)  
 @PreAuthorize(**"hasAuthority('CHECKITEM\_EDIT')"**)*//权限校验* **public** Result edit(@RequestBody CheckItem checkItem){  
 **try** {  
 **checkItemService**.edit(checkItem);  
 }**catch** (Exception e){  
 **return new** Result(**false**,MessageConstant.***EDIT\_CHECKITEM\_FAIL***);  
 }  
 **return new** Result(**true**,MessageConstant.***EDIT\_CHECKITEM\_SUCCESS***);  
 }  
  
 *//查询所有* @RequestMapping(**"/findAll"**)  
 **public** Result findAll(){  
 List<CheckItem> checkItemList = **checkItemService**.findAll();  
 **if**(checkItemList != **null** && checkItemList.size() > 0){  
 Result result = **new** Result(**true**, MessageConstant.***QUERY\_CHECKITEM\_SUCCESS***,checkItemList);  
 *//result.setData(checkItemList);* **return** result;  
 }  
 **return new** Result(**false**,MessageConstant.***QUERY\_CHECKITEM\_FAIL***);  
 }  
}

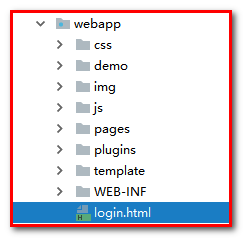
### 第七步：checkitem.html

修改页面，没有权限时提示信息设置，此处以checkitem.html中的handleDelete方法为例

*// 删除*handleDelete(row) {  
 *// alert(row.id);* **this**.**$confirm**(**"确认删除当前选中记录吗？"**,**"提示"**,{**type**:**'warning'**}).then(()=>{  
 *//点击确定按钮时只需此处代码  
 // alert('用户点击的是确定按钮');* **axios**.get(**"/checkitem/delete.do?id="** + row.**id**).then((res)=> {  
 **if**(res.**data**.flag){  
 *//删除成功* **this**.**$message**({  
 **message**: res.**data**.**message**,  
 **type**: **'success'** });  
 *//调用分页，获取最新分页数据* **this**.findPage();  
 }**else**{  
 *//删除失败* **this**.**$message**.error(res.**data**.**message**);  
 }  
 }).catch((error)=>{  
 **this**.showMessage(error);  
 });  
 }).catch(()=> {  
 *//点击取消按钮执行此代码* **this**.**$message**(**'操作已取消'**);  
 });  
},  
*//权限不足提示*showMessage(r){  
 **if**(r == **'Error: Request failed with status code 403'**){  
 *//权限不足* **this**.**$message**.error(**'无访问权限'**);  
 **return**;  
 }**else**{  
 **this**.**$message**.error(**'未知错误'**);  
 **return**;  
 }  
}

### 第八步：导入login.html页面





此时login.html是可放行的页面，而pages下的页面必须认证之后才能访问的页面

## 显示用户名

前面我们已经完成了认证和授权操作，如果用户认证成功后需要在页面展示当前用户的用户名。Spring Security在认证成功后会将用户信息保存到框架提供的上下文对象中，所以此处我们就可以调用Spring Security框架提供的API获取当前用户的username并展示到页面上。

实现步骤：

第一步：在main.html页面中修改，定义username模型数据基于VUE的数据绑定展示用户名，发送ajax请求获取username

1. ：引入js

<**script src="../js/axios-0.18.0.js"**></**script**>

1. ：定义username属性
2. ：使用钩子函数，调用ajax

<**script**>  
 **new** *Vue*({  
 **el**: **'#app'**,  
 **data**:{  
 **menuList**:[  
 {  
 **"path"**: **"1"**,  
 **"title"**: **"工作台"**,  
 **"icon"**:**"fa-dashboard"**,  
 **"children"**: []  
 },  
 {  
 **"path"**: **"2"**,  
 **"title"**: **"会员管理"**,  
 **"icon"**:**"fa-user-md"**,  
 **"children"**: [  
 {  
 **"path"**: **"/2-1"**,  
 **"title"**: **"会员档案"**,  
 **"linkUrl"**:**"member.html"**,  
 **"children"**:[]  
 },  
 {  
 **"path"**: **"/2-2"**,  
 **"title"**: **"体检上传"**,  
 **"children"**:[]  
 },  
 {  
 **"path"**: **"/2-3"**,  
 **"title"**: **"会员统计"**,  
 **"linkUrl"**:**"all-item-list.html"**,  
 **"children"**:[]  
 },  
 ]  
 },  
 {  
 **"path"**: **"3"**,  
 **"title"**: **"预约管理"**,  
 **"icon"**:**"fa-tty"**,  
 **"children"**: [  
 {  
 **"path"**: **"/3-1"**,  
 **"title"**: **"预约列表"**,  
 **"linkUrl"**:**"ordersettinglist.html"**,  
 **"children"**:[]  
 },  
 {  
 **"path"**: **"/3-2"**,  
 **"title"**: **"预约设置"**,  
 **"linkUrl"**:**"ordersetting.html"**,  
 **"children"**:[]  
 },  
 {  
 **"path"**: **"/3-3"**,  
 **"title"**: **"套餐管理"**,  
 **"linkUrl"**:**"setmeal.html"**,  
 **"children"**:[]  
 },  
 {  
 **"path"**: **"/3-4"**,  
 **"title"**: **"检查组管理"**,  
 **"linkUrl"**:**"checkgroup.html"**,  
 **"children"**:[]  
 },  
 {  
 **"path"**: **"/3-5"**,  
 **"title"**: **"检查项管理"**,  
 **"linkUrl"**:**"checkitem.html"**,  
 **"children"**:[]  
 },  
 ]  
 },  
 {  
 **"path"**: **"4"**,  
 **"title"**: **"健康评估"**,  
 **"icon"**:**"fa-stethoscope"**,  
 **"children"**:[  
 {  
 **"path"**: **"/4-1"**,  
 **"title"**: **"中医体质辨识"**,  
 **"linkUrl"**:**"all-medical-list.html"**,  
 **"children"**:[]  
 },  
 ]  
 },  
 {  
 **"path"**: **"5"**, *//菜单项所对应的路由路径* **"title"**: **"统计分析"**, *//菜单项名称* **"icon"**:**"fa-heartbeat"**,  
 **"children"**:[*//是否有子菜单，若没有，则为[]* {  
 **"path"**: **"/5-1"**,  
 **"title"**: **"工作量统计"**,  
 **"linkUrl"**:**"all-medical-list.html"**,  
 **"children"**:[]  
 }  
 ]  
 }  
 ],  
 **username**:**null** *// 用户名* },  
 created:**function** () {  
 *//发送请求获取当前登录用户的用户名* **axios**.get(**'/user/getUsername.do'**).then((response)=>{  
 **this**.**username** = response.**data**.**data**;  
 });  
 }  
 });  
 *$*(**function**() {  
 **var** wd = 200;  
 *$*(**".el-main"**).css(**'width'**, *$*(**'body'**).width() - wd + **'px'**);  
 });  
</**script**>

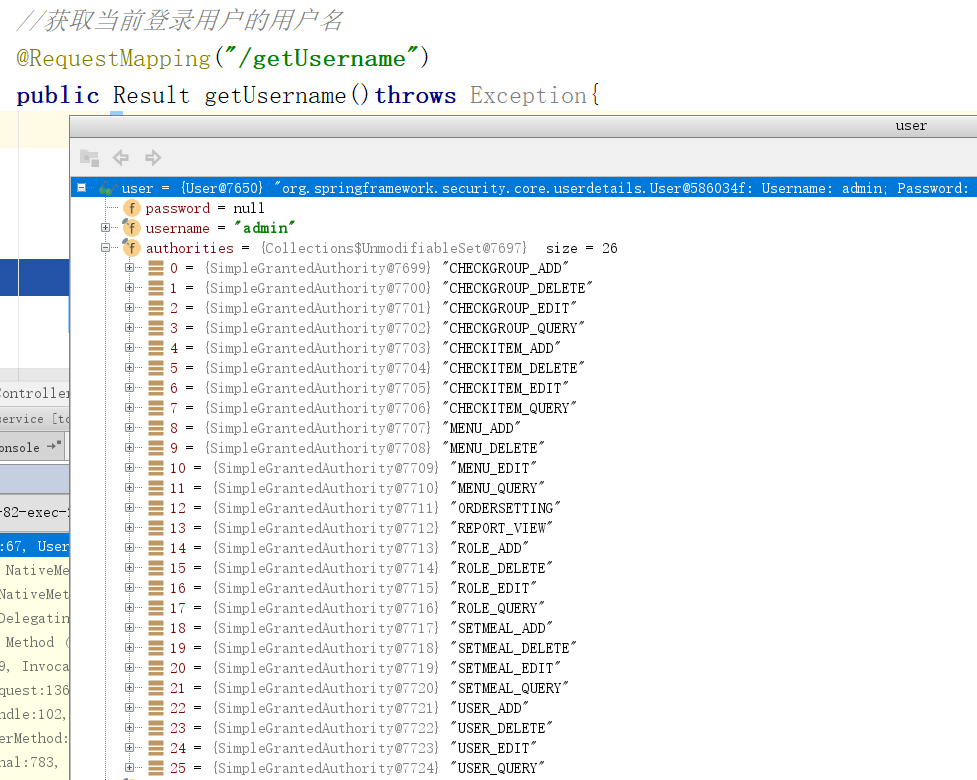
（4）：修改页面

<**div class="avatar-wrapper"**>  
 <**img src="../img/user2-160x160.jpg" class="user-avatar"**>  
 *<!--显示用户名-->* {{username}}  
</**div**>

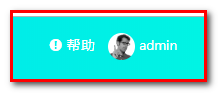
第二步：创建UserController并提供getUsername方法

**package** com.itheima.health.controller;  
  
**import** com.alibaba.dubbo.config.annotation.Reference;  
**import** com.itheima.health.constant.MessageConstant;  
**import** com.itheima.health.entity.Result;  
**import** com.itheima.health.service.UserService;  
**import** org.springframework.security.core.context.SecurityContextHolder;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
  
*/\*\*  
 \* 套餐管理  
 \*/*@RestController  
@RequestMapping(**"/user"**)  
**public class** UserController {  
 @Reference  
 **private** UserService **userService**;  
  
 *//获取当前登录用户的用户名* @RequestMapping(**"/getUsername"**)  
 **public** Result getUsername()**throws** Exception{  
 **try**{  
 org.springframework.security.core.userdetails.User user =  
 (org.springframework.security.core.userdetails.User)  
 SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();  
 **return new** Result(**true**, MessageConstant.***GET\_USERNAME\_SUCCESS***,user.getUsername());  
 }**catch** (Exception e){  
 **return new** Result(**false**, MessageConstant.***GET\_USERNAME\_FAIL***);  
 }  
 }  
}

通过debug调试可以看到Spring Security框架在其上下文中保存的用户相关信息：



显示当前登录人：



## 用户退出

第一步：在main.html中提供的退出菜单上加入超链接

<**el-dropdown-item divided**>  
 <**span style="display**:**block**;**"**><**a href="/logout.do"**>退出</**a**></**span**>  
</**el-dropdown-item**>

第二步：在spring-security.xml文件中配置

*<!--  
 logout：退出登录  
 logout-url：退出登录操作对应的请求路径  
 logout-success-url：退出登录后的跳转页面  
-->*<**security:logout logout-url="/logout.do"  
 logout-success-url="/login.html" invalidate-session="true"**/>

​

# 第二章. 图形报表ECharts

## ECharts简介

ECharts缩写来自Enterprise Charts，商业级数据图表，是百度的一个开源的使用JavaScript实现的数据可视化工具，可以流畅的运行在 PC 和移动设备上，兼容当前绝大部分浏览器（IE8/9/10/11，Chrome，Firefox，Safari等），底层依赖轻量级的矢量图形库 [ZRender](https://github.com/ecomfe/zrender)，提供直观、交互丰富、可高度个性化定制的数据可视化图表。

官网：<https://echarts.baidu.com/>

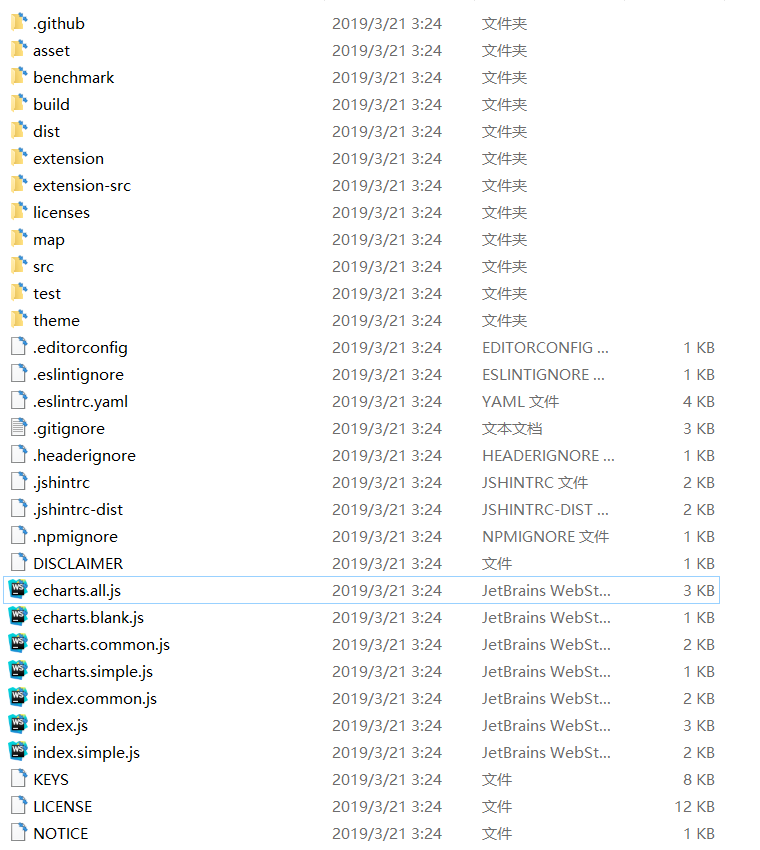
下载地址：<https://echarts.baidu.com/download.html>



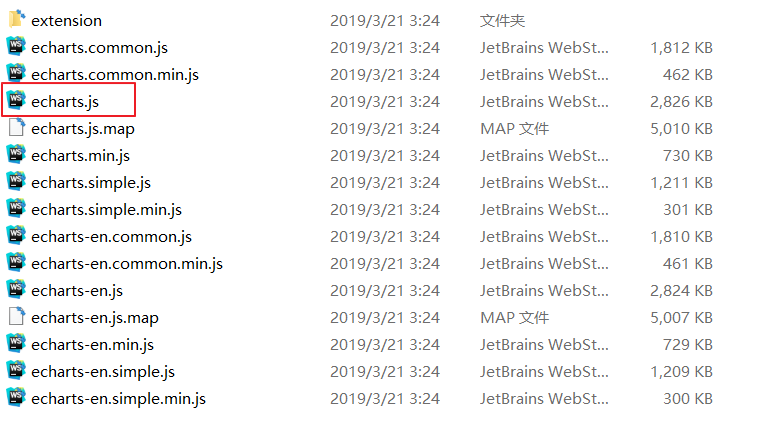
下载完成可以得到如下文件：

IMG_258

解压上面的zip文件：



我们只需要将dist目录下的echarts.js文件（已经压缩）引入到页面上就可以使用了

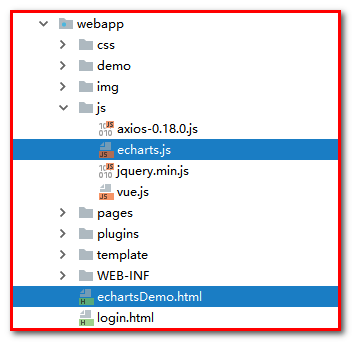


## 5分钟上手ECharts

我们可以参考官方提供的5分钟上手ECharts文档感受一下ECharts的使用方式，地址如下：

[https://www.echartsjs.com/tutorial.html#5%20%E5%88%86%E9%92%9F%E4%B8%8A%E6%89%8B%20ECharts](https://www.echartsjs.com/tutorial.html" \l "5 %E5%88%86%E9%92%9F%E4%B8%8A%E6%89%8B ECharts)

第一步：在health\_web中创建html页面并引入echarts.js文件



echartsDemo.html

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
</**head**>  
*<!-- 引入 ECharts 文件 -->*<**script src="js/echarts.js"**></**script**>  
<**body**>  
  
</**body**>  
</**html**>

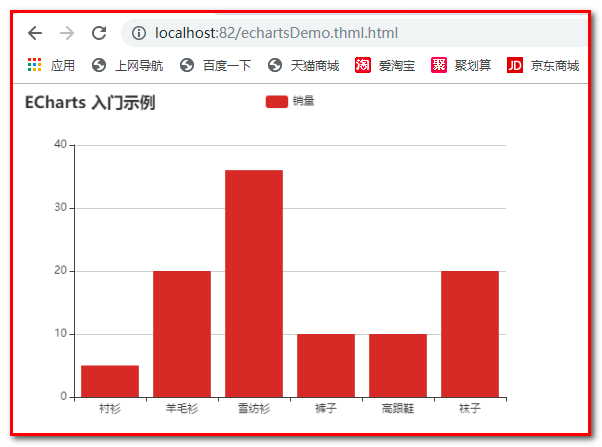
第二步：在页面中准备一个具备宽高的DOM容器。

<**body**>  
 *<!-- 为 ECharts 准备一个具备大小（宽高）的 DOM -->* <**div id="main" style="width**: 600**px**;**height**:400**px**;**"**></**div**>  
</**body**>

第三步：通过echarts.init方法初始化一个 echarts 实例并通过setOption方法生成一个简单的柱状图

<**script type="text/javascript"**>  
 *// 基于准备好的dom，初始化echarts实例* **var *myChart*** = **echarts**.init(***document***.getElementById(**'main'**));  
  
 *// 指定图表的配置项和数据* **var *option*** = {  
 **title**: {  
 **text**: **'ECharts 入门示例'** },  
 **tooltip**: {},  
 **legend**: {  
 **data**:[**'销量'**]  
 },  
 **xAxis**: {  
 **data**: [**"衬衫"**,**"羊毛衫"**,**"雪纺衫"**,**"裤子"**,**"高跟鞋"**,**"袜子"**]  
 },  
 **yAxis**: {},  
 **series**: [{  
 **name**: **'销量'**,  
 **type**: **'bar'**,  
 **data**: [5, 20, 36, 10, 10, 20]  
 }]  
 };  
 *// 使用刚指定的配置项和数据显示图表。* ***myChart***.setOption(***option***);  
</**script**>

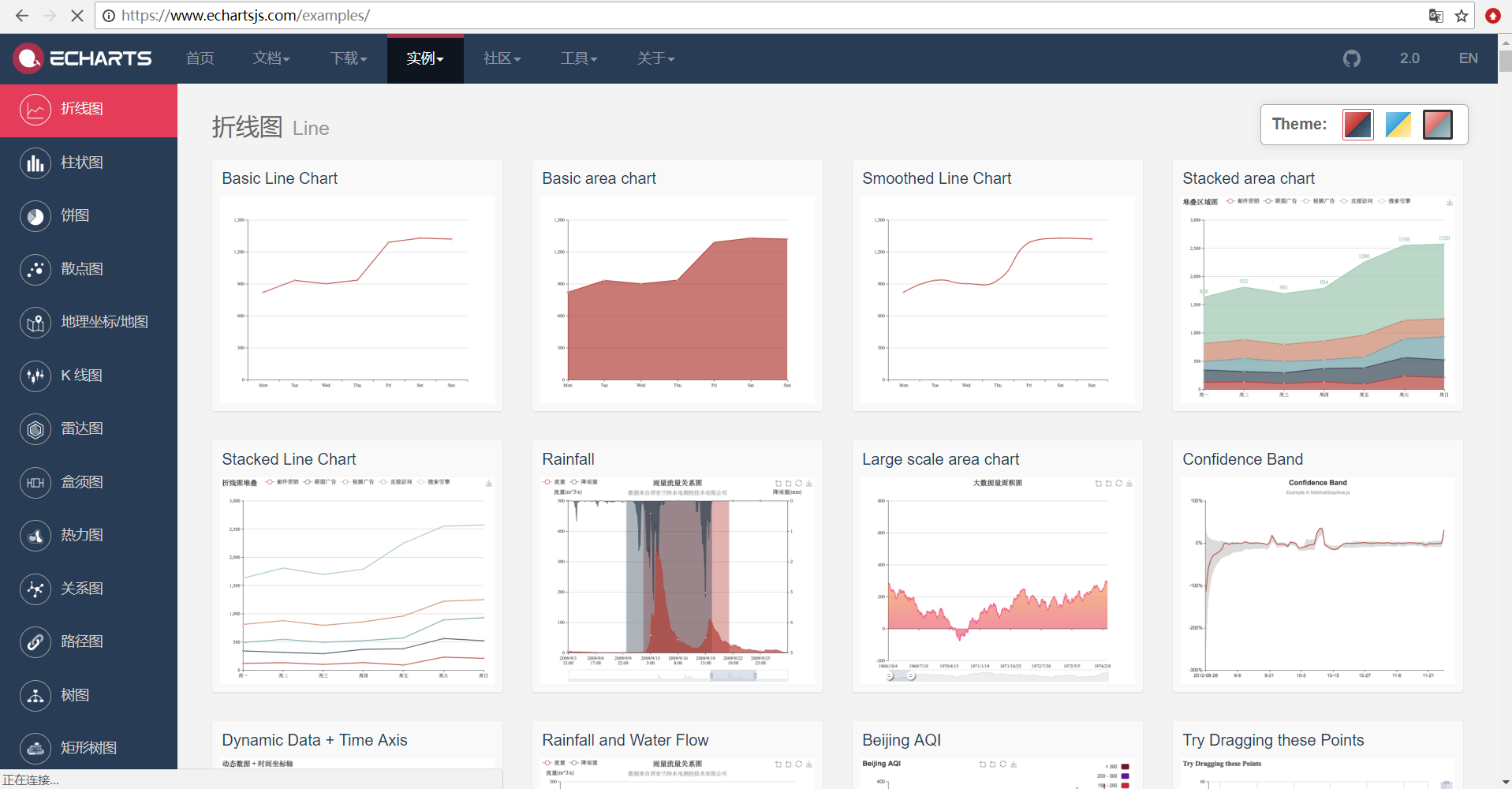
效果如下：



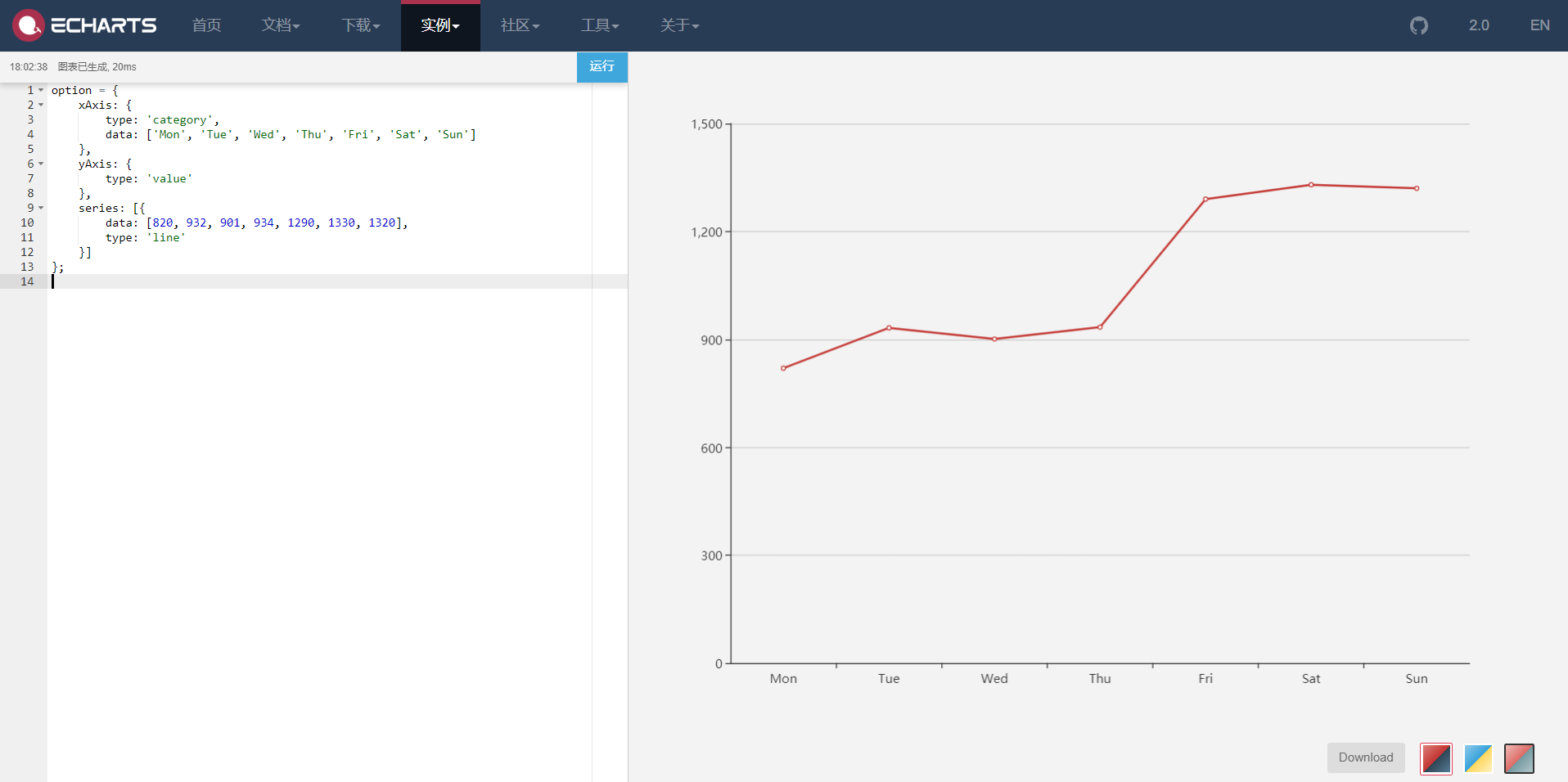
## 查看ECharts官方实例

ECharts提供了很多官方实例，我们可以通过这些官方实例来查看展示效果和使用方法。

官方实例地址：<https://www.echartsjs.com/examples/>



可以点击具体的一个图形会跳转到编辑页面，编辑页面左侧展示源码（js部分源码），右侧展示图表效果，如下：



要查看完整代码可以点击右下角的Download按钮将完整页面下载到本地。

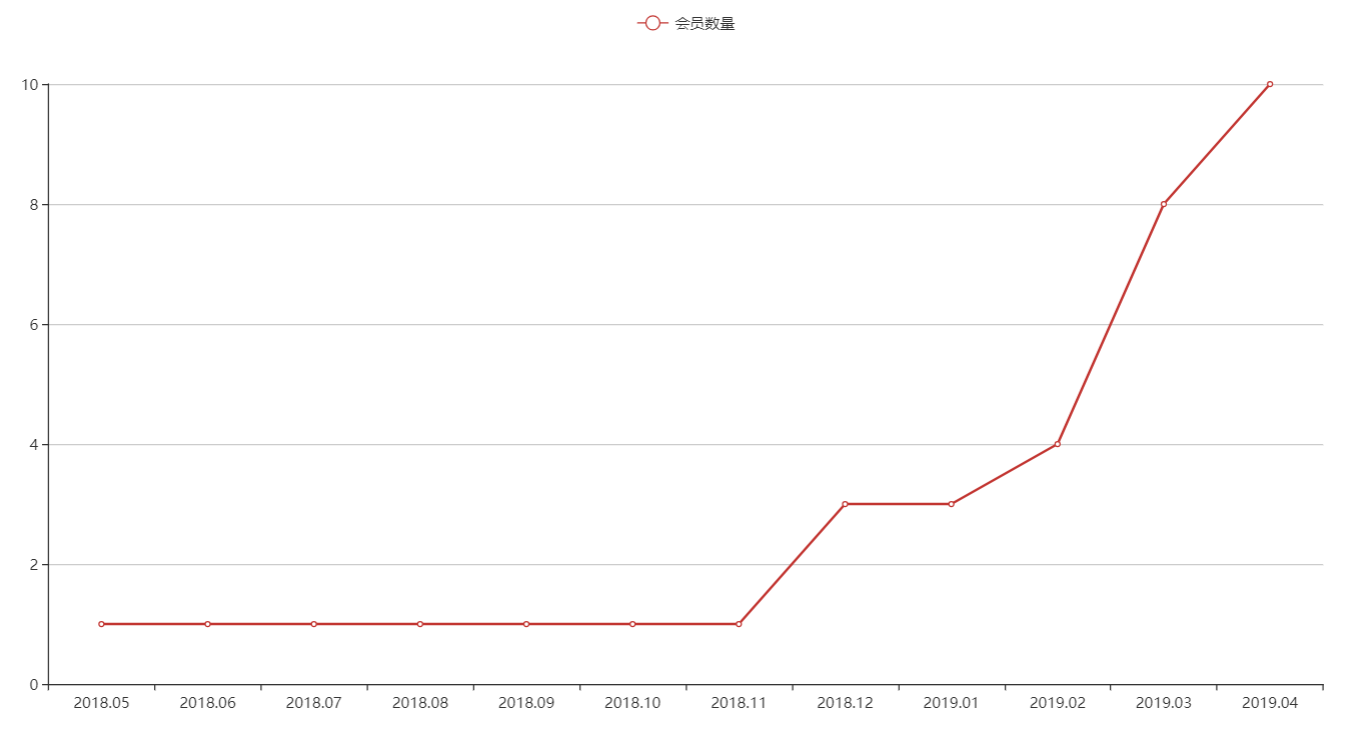
通过官方案例我们可以发现，使用ECharts展示图表效果，关键点在于确定此图表所需的数据格式，然后按照此数据格式提供数据就可以了，我们无须关注效果是如何渲染出来的。

在实际应用中，我们要展示的数据往往存储在数据库中，所以我们可以发送ajax请求获取数据库中的数据并转为图表所需的数据即可。

# 第三章. 会员数量折线图

## 需求分析

会员信息是体检机构的核心数据，其会员数量和增长数量可以反映出机构的部分运营情况。通过折线图可以直观的反映出会员数量的增长趋势。本章节我们需要展示过去一年时间内每个月的会员总数据量。展示效果如下图：



需要的sql：

#2019年05月31日之前入职的人数(11)

SELECT COUNT(id) FROM t\_member WHERE regTime <= '2019-05-31'

#2019年06月31日之前入职的人数(12)

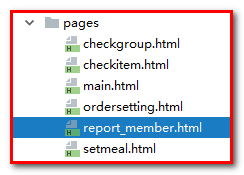
SELECT COUNT(id) FROM t\_member WHERE regTime <= '2019-06-31'

#2019年07月31日之前入职的人数(16)

SELECT COUNT(id) FROM t\_member WHERE regTime <= '2019-07-31'

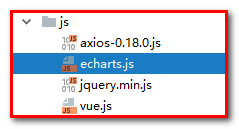
## 完善页面

会员数量折线图对应的页面为/pages/report\_member.html。



### 导入ECharts库

第一步：将echarts.js文件复制到health\_web工程中



第二步：在report\_member.html页面引入echarts.js文件

<**script src="../js/echarts.js"**></**script**>

### 参照官方实例导入折线图

1：定义<div>

<**div class="app-container"**>  
 <**div class="box"**>  
 *<!-- 为 ECharts 准备一个具备大小（宽高）的 DOM -->* <**div id="chart1" style="height**:600**px**;**"**></**div**>  
 </**div**>  
</**div**>

2：定义script

<**script type="text/javascript"**>  
 *// 基于准备好的dom，初始化echarts实例* **var *myChart1*** = **echarts**.init(***document***.getElementById(**'chart1'**));  
  
 *// 使用刚指定的配置项和数据显示图表。  
 //myChart.setOption(option);* **axios**.get(**"/report/getMemberReport.do"**).then((res)=>{  
 ***myChart1***.setOption(  
 {  
 **title**: {  
 **text**: **'会员数量'** },  
 **tooltip**: {},  
 **legend**: {  
 **data**:[**'会员数量'**]  
 },  
 **xAxis**: {  
 **data**: res.**data**.**data**.**months** },  
 **yAxis**: {  
 **type**:**'value'** },  
 **series**: [{  
 **name**: **'会员数量'**,  
 **type**: **'line'**,  
 **data**: res.**data**.**data**.memberCount  
 }]  
 });  
 });  
</**script**>

根据折现图对数据格式的要求，我们发送ajax请求，服务端需要返回如下格式的数据：

其中months和memberCount可以使用hashmap封装

返回Result：data、flag、message

其中data，是map集合

map集合的key：months； map集合的值：List<String>

map集合的key：memberCount； map集合的值：List<Integer>

{  
 **"data"**:{  
 **"months"**:[**"2019-01"**,**"2019-02"**,**"2019-03"**,**"2019-04"**],  
 **"memberCount"**:[3,4,8,10]  
 },  
 **"flag"**:**true**,  
 **"message"**:**"获取会员统计数据成功"**}

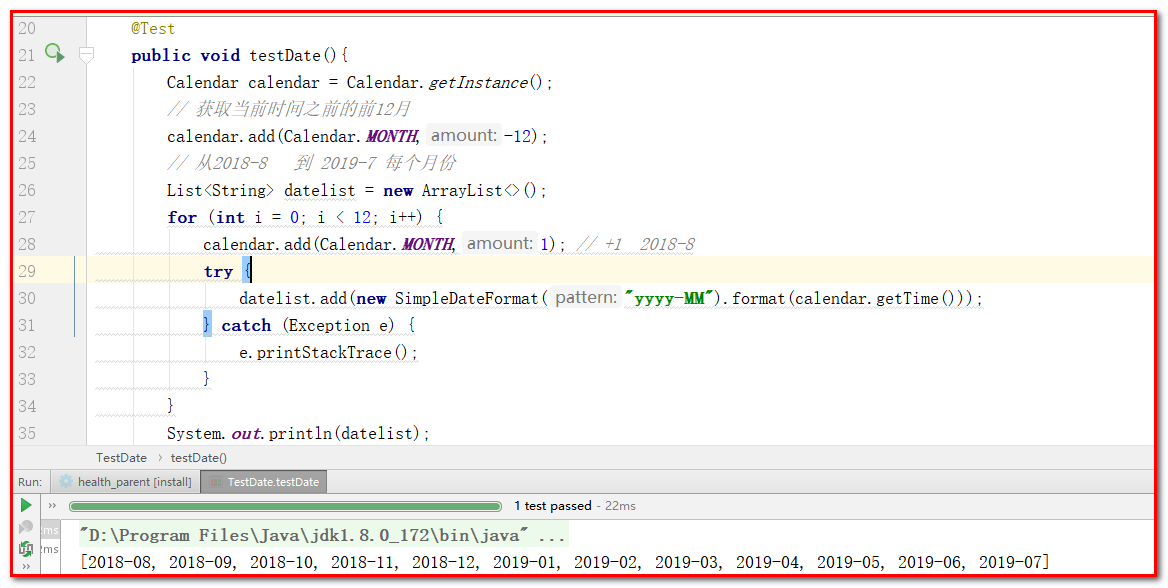
## 后台代码

### Controller

在health\_web工程中创建ReportController并提供getMemberReport方法

**package** com.itheima.health.controller;  
  
**import** com.alibaba.dubbo.config.annotation.Reference;  
**import** com.itheima.health.constant.MessageConstant;  
**import** com.itheima.health.entity.Result;  
**import** com.itheima.health.service.MemberService;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
  
**import** java.text.SimpleDateFormat;  
**import** java.util.\*;  
  
*/\*\*  
 \* 统计报表  
 \*/*@RestController  
@RequestMapping(**"/report"**)  
**public class** ReportController {  
 @Reference  
 **private** MemberService **memberService**;  
  
 */\*\*  
 \* 会员数量统计  
 \** ***@return*** *\*/* @RequestMapping(**"/getMemberReport"**)  
 **public** Result getMemberReport() {  
 Calendar calendar = Calendar.*getInstance*();  
 calendar.add(Calendar.***MONTH***,-12);*//获得当前日期之前12个月的日期* List<String> list = **new** ArrayList<String>();  
 **for**(**int** i=0;i<12;i++){  
 calendar.add(Calendar.***MONTH***,1);  
 list.add(**new** SimpleDateFormat(**"yyyy-MM"**).format(calendar.getTime()));  
 }  
  
 Map<String,Object> map = **new** HashMap<String,Object>();  
 map.put(**"months"**,list);  
  
 List<Integer> memberCount = **memberService**.findMemberCountByMonth(list);  
 map.put(**"memberCount"**,memberCount);  
  
 **return new** Result(**true**, MessageConstant.***GET\_MEMBER\_NUMBER\_REPORT\_SUCCESS***,map);  
 }  
}

计算日期，可以使用test方法测试一下



### 服务接口

在MemberService服务接口中扩展方法findMemberCountByMonth

List<Integer> findMemberCountByMonth(List<String> months);

### 服务实现类

在MemberServiceImpl服务实现类中实现findMemberCountByMonth方法

*// 根据月份统计会员数量***public** List<Integer> findMemberCountByMonth(List<String> months) {  
 List<Integer> list = **new** ArrayList<>();  
 **for**(String m : months){  
 m = m + **"-31"**;*//格式：2019-04-31* Integer count = **memberDao**.findMemberCountBeforeDate(m);  
 list.add(count);  
 }  
 **return** list;  
}

### Dao接口

在MemberDao接口中扩展方法findMemberCountBeforeDate

**public** Integer findMemberCountBeforeDate(String date);

### Mapper映射文件

在MemberDao.xml映射文件中提供SQL语句

*<!--根据日期统计会员数，统计指定日期之前的会员数-->*<**select id="findMemberCountBeforeDate" parameterType="string" resultType="int"**>  
 select count(id) from t\_member where regTime **&lt;**= #{value}  
</**select**>

注意：在xml文件中 ，<号需要转义”&lt;”

>号需要转义”&gt;”

测试

1：修改main.html



2：导入数据



3：测试效果

