**品优购电商系统开发**

**第4章**

**安全框架与商家入驻审核**

传智播客.黑马程序员

# 课程目标

目标1：实现SpringSecurity入门小Demo (认证 和 授权)

目标2：完成运营商登陆与安全控制功能

目标3：完成商家入驻

目标4：完成商家审核

目标5：完成商家系统登陆与安全控制功能

# 1.Spring Security框架入门

## 1.1 Spring Security简介

Spring Security是一个能够为基于Spring的企业应用系统提供声明式的安全访问控制解决方案的安全框架。它提供了一组可以在Spring应用上下文中配置的Bean，充分利用了Spring IoC，DI（控制反转Inversion of Control ,DI:Dependency Injection 依赖注入）和AOP（面向切面编程）功能，为应用系统提供声明式的安全访问控制功能，减少了为企业系统安全控制编写大量重复代码的工作。

spring security安全框架：授权（角色）和认证（登录）。防止攻击。

## 1.2 Spring Security入门Demo

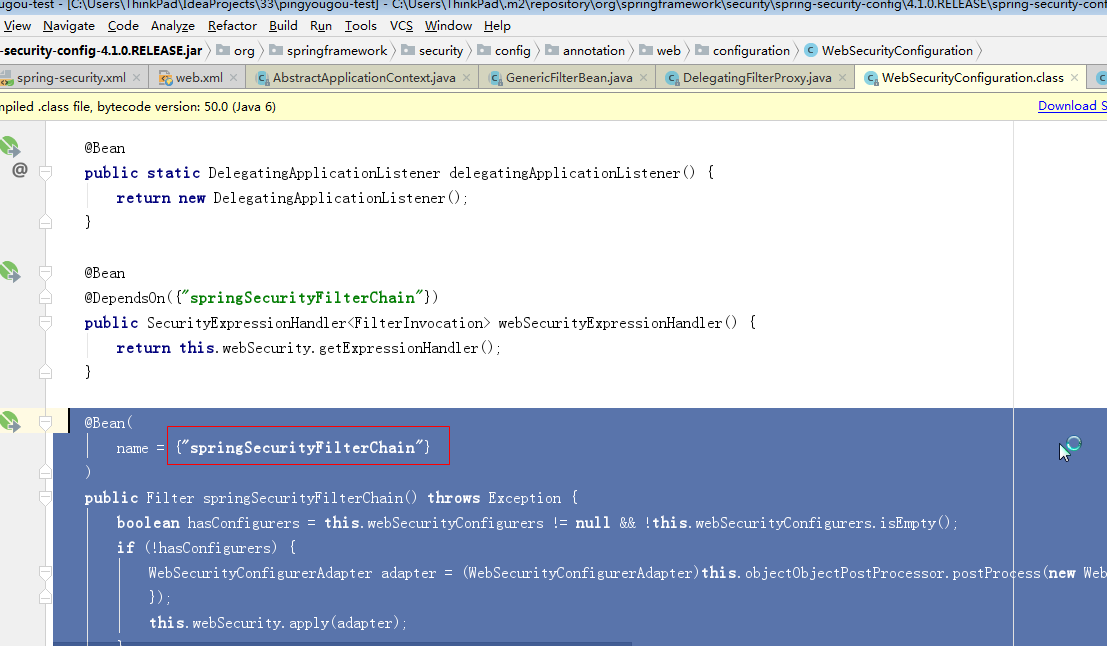
### 1.2.1最简单Demo

（1）创建工程spring-security-demo ,pom.xml内容

|  |
| --- |
| <projectxmlns=*"http://maven.apache.org/POM/4.0.0"*xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>cn.itcast.demo</groupId>  <artifactId>spring-security-demo</artifactId>  <packaging>war</packaging>  <version>1.0-SNAPSHOT</version>  <properties>  <spring.version>4.3.10.RELEASE</spring.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-web</artifactId>  <version>4.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-config</artifactId>  <version>4.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  <scope>provided</scope>  </dependency>  </dependencies>  <build>  <plugins>  <!-- java编译插件 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.2</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <!-- 指定端口 -->  <port>9090</port>  <!-- 请求路径 -->  <path>/</path>  </configuration>  </plugin>  </plugins>  </build>  </project> |

（2）创建web.xml

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <web-appxmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  version=*"2.5"*>  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring-security.xml</param-value>  </context-param>  <listener>  <listener-class>  org.springframework.web.context.ContextLoaderListener  </listener-class>  </listener>  <filter>  <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>  </web-app> |

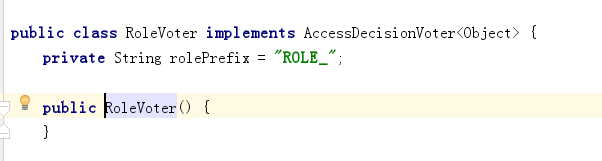


如图，必须filter名字要叫做springSecurityFilterChain不能修改。

（3）创建index.html 内容略

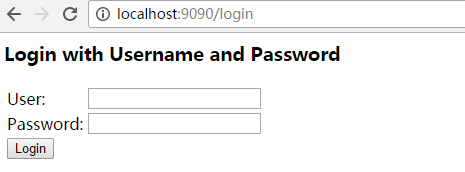
（4）创建spring 配置文件spring-security.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans:beans xmlns="http://www.springframework.org/schema/security"  xmlns:beans="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security.xsd"**>  </**beans:beans**>  *<!-- 页面拦截规则 -->*  *<http use-expressions="false">*  *<intercept-urlpattern="/\*\*"access="ROLE\_USER"/>*  *<form-login/>*  *</http>*  *<!-- 认证管理器 -->*  *<authentication-manager>*  *<authentication-provider>*  *<user-service>*  *<user name="admin" password="123456" authorities="ROLE\_USER"/>*  *</user-service>*  *</authentication-provider>*  *</authentication-manager>*  *</beans:beans>* |



角色一定是ROLE\_开头

此案例我们没有登录页，而是使用了系统自动生成的登陆页，效果如下：



配置说明：

intercept-url表示拦截页面

/\* 表示的是该目录下的资源，只包括本级目录不包括下级目录

/\*\* 表示的是该目录以及该目录下所有级别子目录的资源

form-login为开启表单登陆

use-expressions为是否使用使用 Spring 表达式语言（SpEL），默认为true ,如果开启，则拦截的配置应该写成以下形式

|  |
| --- |
| <intercept-url pattern=*"/\*\*"* access=*"hasRole('ROLE\_USER')"*/> |

### 1.2.2用户自定义登录页

实际开发中，我们不可能使用系统生成的登录页，而是使用我们自己的登录页。

（1）构建登陆页：

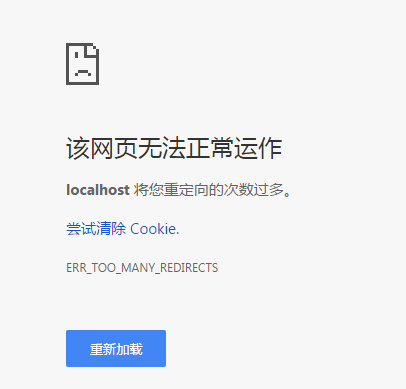
|  |
| --- |
| <**html**> <**head**>  <**meta http-equiv="Content-Type" content="text/html; charset=UTF-8"**>  <**title**>登陆</**title**> </**head**> <**body**> <**form action='/login' method='POST'**>  <**table**>  <**tr**>  <**td**>用户名:</**td**>  <**td**><**input type='text' name='username' value=''**></**td**>  </**tr**>  <**tr**>  <**td**>密码:</**td**>  <**td**><**input type='password' name='password'**/></**td**>  </**tr**>  <**tr**>  <**td colspan='2'**><**input name="submit" type="submit" value="登陆"**/></**td**>  </**tr**>  </**table**> </**form**> </**body**> </**html**> |

1. 构建登陆失败页 login\_error.html（内容略）

（3）修改 spring 配置文件spring-security.xml

|  |
| --- |
| <!-- 以下页面不被拦截 -->  <httppattern=*"/login.html"* security=*"none"*></http>  <httppattern=*"/login\_error.html"* security=*"none"*></http>  <!-- 页面拦截规则 -->  <httpuse-expressions=*"false"*>  <intercept-urlpattern=*"/\*"*access=*"ROLE\_USER"*/>  <form-loginlogin-page=*"/login.html"*default-target-url=*"/index.html"*authentication-failure-url=*"/login\_error.html"*/>  <csrfdisabled=*"true"*/>  </http> |

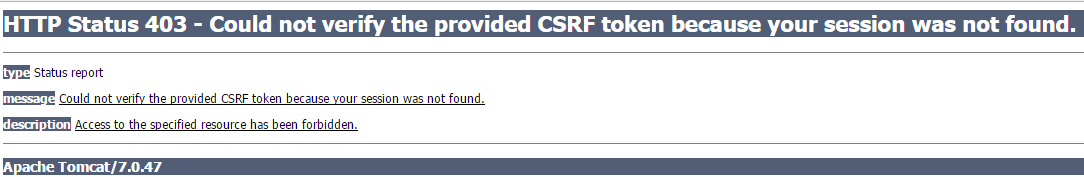
security="none"设置此资源不被拦截.   
如果你没有设置登录页security="none"，将会出现以下错误



因为登录页会被反复重定向。

login-page：指定登录页面。  
authentication-failure-url：指定了身份验证失败时跳转到的页面。  
default-target-url：指定了成功进行身份验证和授权后默认呈现给用户的页面。

csrf disabled="true"关闭csrf ,如果不加会出现错误



CSRF（Cross-site request forgery）跨站请求伪造，也被称为“One Click Attack”或者Session Riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。

# 2.运营商系统登录与安全控制

## 2.1需求分析

完成运营商登陆功能



## 2.2登陆功能的实现

### 2.2.1配置文件

（1）修改pinyougou-manager-web的pom.xml ，添加依赖

|  |
| --- |
| <!-- 身份验证 -->  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-config</artifactId>  </dependency> |

（2）修改web.xml

|  |
| --- |
| <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/spring-security.xml</param-value>  </context-param>  <listener>  <listener-class>  org.springframework.web.context.ContextLoaderListener  </listener-class>  </listener>  <filter>  <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> |

（3）pinyougou-manager-web的spring目录下添加配置文件spring-security.xml

|  |
| --- |
| <!-- 以下页面不被拦截 -->  <httppattern=*"/login.html"*security=*"none"*></http>  <httppattern=*"/css/\*\*"*security=*"none"*></http>  <httppattern=*"/img/\*\*"*security=*"none"*></http>  <httppattern=*"/js/\*\*"*security=*"none"*></http>  <httppattern=*"/plugins/\*\*"*security=*"none"*></http>    <!-- 页面拦截规则 -->  <http use-expressions=*"false"*>  <intercept-url pattern=*"/\*"*access=*"ROLE\_ADMIN"*/>  <form-loginlogin-page=*"/login.html"* default-target-url=*"/admin/index.html"*authentication-failure-url=*"/login.html"*always-use-default-target=*"true"*/>  <csrf disabled=*"true"*/>  <headers>  <frame-options policy=*"SAMEORIGIN"*/>  </headers>  </http>  <!-- 认证管理器 -->  <authentication-manager>  <authentication-provider>  <user-service>  <username=*"admin"*password=*"123456"*authorities=*"ROLE\_ADMIN"*/>  <username=*"sunwukong"*password=*"dasheng"*authorities=*"ROLE\_ADMIN"*/>  </user-service>  </authentication-provider>  </authentication-manager> |

配置说明：

always-use-default-target：指定了是否在身份验证通过后总是跳转到default-target-url属性指定的URL。

如果你在系统中使用了框架页，需要设置框架页的策略为SAMEORIGIN

|  |
| --- |
| <headers>  <frame-options policy=*"SAMEORIGIN"*/>  </headers> |

### 2.2.2登录页面

修改pinyougou-manager-web的 login.html

|  |
| --- |
| <formid=*"loginform"* action=*"/login"* method=*"post"* class=*"sui-form"*>  <div class=*"input-prepend"*><span class=*"add-on loginname"*></span>  <input id=*"prependedInput"* name=*"username"* type=*"text"* placeholder=*"邮箱/用户名/手机号"* class=*"span2 input-xfat"*>  </div>  <div class=*"input-prepend"*><span class=*"add-on loginpwd"*></span>  <input id=*"prependedInput"* name=*"password"* type=*"password"* placeholder=*"请输入密码"* class=*"span2 input-xfat"*>  </div>  <div class=*"setting"*>  <div id=*"slider"*>  <div id=*"slider\_bg"*></div>  <span id=*"label"*>>></span><span id=*"labelTip"*>拖动滑块验证</span>  </div>  </div>  <div class=*"logined"*>  <a class=*"sui-btnbtn-block btn-xlargebtn-danger"*onclick="document:loginform.submit()" target=*"\_blank"*>登&nbsp;&nbsp;录</a>  </div>  </form> |

## 2.3主界面显示登录用户名

目前登录功能已经实现，登录之后需要在主界面显示登录用户的用户名。

需要通过spring security的方式来获取。

### 2.3.1后端代码

在pinyougou-manager-web新建LoginController.java

|  |
| --- |
| **package**com.pinyougou.manager.controller;  **import**java.util.HashMap;  **import**java.util.Map;  **import** org.springframework.security.core.context.SecurityContextHolder;  **import**org.springframework.security.core.userdetails.UserDetails;  **import**org.springframework.web.bind.annotation.RequestMapping;  **import**org.springframework.web.bind.annotation.RestController;  @RestController  @RequestMapping("/login")  **public class** LoginController {  @RequestMapping("name")  **Public** Map name(){  String name=SecurityContextHolder.*getContext*()  .getAuthentication().getName();  Mapmap=**new**HashMap();  map.put("loginName", name);  **return** map ;  }  } |

### 2.3.2前端代码

1. 新建loginService.js

|  |
| --- |
| //登陆服务层  app.service('loginService',**function**($http){  //读取登录人名称  **this**.loginName=**function**(){  **return** $http.get('../login/name.do');  }  }); |

1. 新建indexController.js

|  |
| --- |
| app.controller('indexController' ,**function**($scope,$controller ,loginService){  //读取当前登录人  $scope.showLoginName=**function**(){  loginService.loginName().success(  **function**(response){  $scope.loginName=response.loginName;  }  );  }  }); |

index.html页面上引入JS

|  |
| --- |
| <scripttype=*"text/javascript"*src=*"../plugins/angularjs/angular.min.js"*></script>  <scripttype=*"text/javascript"*src=*"../js/base.js"*></script>  <scripttype=*"text/javascript"*src=*"../js/service/loginService.js"*></script>  <scripttype=*"text/javascript"*src=*"../js/controller/indexController.js"*></script> |

指令

|  |
| --- |
| <body class=*"hold-transition skin-green sidebar-mini"*ng-app=*"pinyougou"*ng-controller=*"indexController"ng-init="*showLoginName*()">* |

将页面上的测试用户替换成 {{loginName}}

## 2.4退出登录

在pinyougou-manager-web的spring-security.xml的http节点中添加配置

|  |
| --- |
| <logout/> |

加此配置后，会自动的产生退出登录的地址/logout,如果你不想用这个地址，你也可以定义生成的退出地址以及跳转的页面，配置如下

|  |
| --- |
| <**logout logout-success-url="/login.html" logout-url="/mylogout"**/> |

logout-url:退出的地址，会自动生成

logout-success-url:退出后跳转的地址 。

注销之后跳转到登录的页面。

修改注销的链接

|  |
| --- |
| <div class=*"pull-right"*>  <a href=*"../logout"*class=*"btnbtn-default btn-flat"*>注销</a>  </div> |

默认的配置项登出的url:/logout

# 3.商家申请入驻

## 3.1需求分析

商家申请入驻，需要填写商家相关的信息。待运营商平台审核通过后即可使用使用。

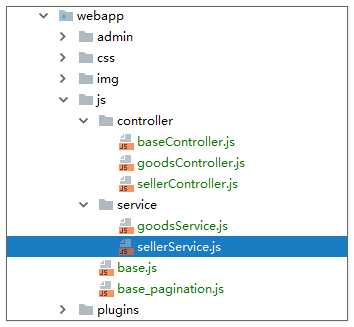
需要涉及到的表为tb\_seller

## 3.2准备工作

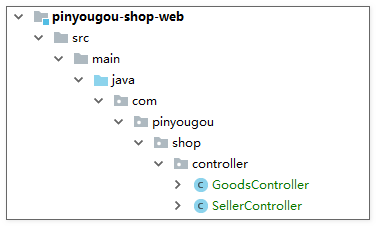
（1）拷贝资源：将“资源/静态原型/商家管理后台”下的页面拷贝到pinyougou-shop-web工程。如果没有，则创建pinyougou-shop-web工程。



1. 参照“运营商后台”构建js



1. 拷贝后端控制层代码



(4)参考pinyougou-manger-web工程搭建环境配置。（web.xml resource下的配置文件）

略

## 3.3前端代码

修改register.html 引入JS

|  |
| --- |
| <scripttype=*"text/javascript"*src=*"plugins/angularjs/angular.min.js"*></script>  <scripttype=*"text/javascript"*src=*"js/base.js"*></script>  <scripttype=*"text/javascript"*src=*"js/service/sellerService.js"*></script>  <scripttype=*"text/javascript"*src=*"js/controller/baseController.js"*></script>  <scripttype=*"text/javascript"*src=*"js/controller/sellerController.js"*></script> |

指令

|  |
| --- |
| <body ng-app=*"pinyougou"*ng-controller=*"sellerController"*> |

使用ng-model绑定变量 将来需要将此表单数据插入到数据库中（即：商家入驻）

绑定表单（部分代码）

|  |
| --- |
| <divclass=*"control-group"*>  <labelclass=*"control-label"*>登陆名（不可修改）：</label>  <divclass=*"controls"*>  <inputtype=*"text"*ng-model=*"entity.sellerId"*placeholder=*"登陆名"*class=*"input-xfat input-xlarge"*>  </div>  </div>  <divclass=*"control-group"*>  <labelclass=*"control-label"*>登陆密码：</label>  <divclass=*"controls"*>  <inputtype=*"password"*ng-model=*"entity.password"*placeholder=*"登陆密码"*class=*"input-xfat input-xlarge"*>  </div>  </div>  ..............略 |

修改sellerController.js ，在保存成功后跳转到登陆页

|  |
| --- |
| //添加  $scope.add=**function**(){  sellerService.add( $scope.entity ).success(  **function**(response){  **if**(response.success){  location.href='shoplogin.html';  }**else**{  alert(response.message);  }  }  );  } |

绑定“申请入驻”按钮

|  |
| --- |
| <a class=*"sui-btnbtn-block btn-xlargebtn-danger"*ng-click=*"add()"*target=*"\_blank"*>申请入驻</a> |

## 3.4后端代码

插入数据到商家表中，表示商家申请要入驻。

修改后端代码pinyougou-sellergoods-service的SellerServiceImpl类的add方法，设置默认状态为0

|  |
| --- |
| /\*\*  \* 增加  \*/  @Override  **public void** add(TbSeller seller) {  seller.setStatus("0");  seller.setCreateTime(new Date());  sellerMapper.insert(seller);  } |

# 4.商家审核(运营商管理后台)

## 4.1需求分析

商家申请入驻后，需要网站运营人员在运营商后台进行审核，审核后商家才可以登陆系统。

**状态值：** 0：未审核 1：已审核 2：审核未通过 3：已关闭

## 4.2商家待审核列表

修改seller\_1.html

引入JS

|  |
| --- |
| <scripttype=*"text/javascript"*src=*"../plugins/angularjs/angular.min.js"*></script>  <!-- 分页组件开始 -->  <scriptsrc=*"../plugins/angularjs/pagination.js"*></script>  <linkrel=*"stylesheet"*href=*"../plugins/angularjs/pagination.css"*>  <!-- 分页组件结束 -->  <scripttype=*"text/javascript"*src=*"../js/base\_pagination.js"*></script>  <scripttype=*"text/javascript"*src=*"../js/service/sellerService.js"*></script>  <scripttype=*"text/javascript"*src=*"../js/controller/baseController.js"*></script>  <scripttype=*"text/javascript"*src=*"../js/controller/sellerController.js"*></script> |

指令

|  |
| --- |
| <body class=*"hold-transition skin-red sidebar-mini"*ng-app=*"pinyougou"*ng-controller=*"sellerController"*ng-init="searchEntity={status:'0'}*"*> |

加入分页控件

|  |
| --- |
| <tm-paginationconf=*"paginationConf"*></tm-pagination> |

循环

|  |
| --- |
| <trng-repeat=*"entity in list"*>  <td><inputtype=*"checkbox"*></td>  <td>{{entity.sellerId}}</td>  <td>{{entity.name}}</td>  <td>{{entity.nickName}}</td>  <td>{{entity.linkmanName}}</td>  <td>{{entity.telephone}}</td>  <tdclass=*"text-center"*>  <buttontype=*"button"*class=*"btnbg-olive btn-xs"*data-toggle=*"modal"*data-target=*"#sellerModal"*>详情</button>  </td>  </tr> |

## 4.3商家详情



（1）绑定页面弹出窗口

|  |
| --- |
| <tableclass=*"table table-bordered table-striped"*width=*"800px"*>  <tr>  <td>公司名称</td>  <td>{{entity.name}}</td>  </tr>  <tr>  <td>公司手机</td>  <td>{{entity.mobile}}</td>  </tr>  <tr>  <td>公司电话</td>  <td>{{entity.telephone}}</td>  </tr>  <tr>  <td>公司详细地址</td>  <td>{{entity.addressDetail}}</td>  </tr>  </table> |

（2）列表的“详情”按钮 点击详情需要查询商家信息 根据商家ID获取商家信息。

|  |
| --- |
| <button type=*"button"* class=*"btnbg-olive btn-xs"* data-toggle=*"modal"* data-target=*"#sellerModal"*ng-click=*"findOne(entity.sellerId)"*>详情</button> |

## 4.4商家审核代码实现

商家的审核实际上就是通过将状态值修改即可。例如：将0 修改成1 表示将一个未审核的商家审核成已审核的状态。

### 4.4.1 后端代码

（1）在pinyougou-sellergoods-interface工程的SellerService.java服务接口新增方法定义

|  |
| --- |
| /\*\*  \* 更改状态 为了更好的通用性  \* **@param** id  \* **@param** status  \*/  **Public void** updateStatus(String sellerId,String status); |

1. 在pinyougou-sellergoods-service的SellerServiceImpl.java新增方法

|  |
| --- |
| @Override  **public void** updateStatus(String sellerId, String status) {  TbSeller seller = sellerMapper.selectByPrimaryKey(sellerId);  seller.setStatus(status);  sellerMapper.updateByPrimaryKey(seller);  } |

1. 在pinyougou-manager-web的SellerController.java新增方法

|  |
| --- |
| /\*\*  \* 更改状态  \* **@param**sellerId商家ID  \* **@param** status 状态  \*/  @RequestMapping("/updateStatus")  **public** Result updateStatus(String sellerId, String status){  **try** {  sellerService.updateStatus(sellerId, status);  **returnnew** Result(**true**, "成功");  } **catch** (Exception e) {  e.printStackTrace();  **returnnew** Result(**false**, "失败");  }  } |

### 4.4.2 前端代码

修改pinyougou-manager-web的sellerService.js

|  |
| --- |
| //更改状态  **this**.updateStatus=**function**(sellerId,status){  **return** $http.get('../seller/updateStatus.do?sellerId='+sellerId+'&status='+status);  } |

修改pinyougou-manager-web的sellerController.js

|  |
| --- |
| $scope.updateStatus=**function**(sellerId,status){  sellerService.updateStatus(sellerId,status).success(  **function**(response){  **if**(response.success){  $scope.reloadList();//刷新列表  }**else**{  alert("失败");  }  }  );  } |

修改按钮，调用方法

|  |
| --- |
| <divclass=*"modal-footer"*>  <buttonclass=*"btnbtn-success"*data-dismiss=*"modal"*aria-hidden=*"true"*ng-click=*"updateStatus(entity.sellerId,'1')"*>审核通过</button>  <buttonclass=*"btnbtn-danger"*data-dismiss=*"modal"*aria-hidden=*"true"*ng-click=*"updateStatus(entity.sellerId,'2')"*>审核未通过</button>  <buttonclass=*"btnbtn-danger"*data-dismiss=*"modal"*aria-hidden=*"true"*ng-click=*"updateStatus(entity.sellerId,'3')"*>关闭商家</button>  <buttonclass=*"btnbtn-default"*data-dismiss=*"modal"*aria-hidden=*"true"*>关闭</button>  </div> |

# 5.商家系统登录与安全控制

## 5.1需求分析

当商家申请入驻以后，由运营商审核，审核通过以后，该商家就需要登录到系统，进行商品的录入等基本功能。

商家账号来自数据库，并实现密码加密。

## 5.2自定义认证类

1. pom.xml、web.xml 、login.html 参照运营商管理后台
2. 在pinyougou-shop-web创建com.pinyougou.service包，包下创建类UserDetailsServiceImpl.java 实现UserDetailsService接口

|  |
| --- |
| **package** com.pinyougou.service;  **import** java.util.ArrayList;  **import** java.util.List;  **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;  /\*\*  \* 认证类  \* **@author** Administrator  \*  \*/  **publicclass**UserDetailsServiceImpl**implements**UserDetailsService {  @Override  **public**UserDetailsloadUserByUsername(String username) **throws**UsernameNotFoundException {  List<GrantedAuthority>grantedAuths = **new** ArrayList<GrantedAuthority>();  grantedAuths.add(**new** SimpleGrantedAuthority("ROLE\_SELLER"));  **returnnew** User(username,"123456", grantedAuths);  }  } |

1. 在pinyougou-shop-web的spring目录下创建spring-security.xml

|  |
| --- |
| <!-- 以下页面不被拦截 -->  <httppattern=*"/\*.html"*security=*"none"*></http>  <httppattern=*"/css/\*\*"*security=*"none"*></http>  <httppattern=*"/img/\*\*"*security=*"none"*></http>  <httppattern=*"/js/\*\*"*security=*"none"*></http>  <httppattern=*"/plugins/\*\*"*security=*"none"*></http>  <httppattern=*"/seller/add.do"*security=*"none"*></http>  <!-- 页面拦截规则 -->  <httpuse-expressions=*"false"*>  <intercept-urlpattern=*"/\*\*"*access=*"ROLE\_SELLER"*/>  <form-loginlogin-page=*"/shoplogin.html"*default-target-url=*"/admin/index.html"*authentication-failure-url=*"/shoplogin.html"*always-use-default-target=*"true"*/>  <csrfdisabled=*"true"*/>  <headers>  <frame-optionspolicy=*"SAMEORIGIN"*/>  </headers>  <logout/>  </http>  <!-- 认证管理器 -->  <authentication-manager>  <authentication-provider user-service-ref=*"userDetailService"*>  </authentication-provider>  </authentication-manager>  <beans:beanid=*"userDetailService"*  class=*"com.pinyougou.service.UserDetailServiceImpl"*></beans:bean> |

经过上述配置，用户在输入密码123456时就会通过（用户名随意）

## 5.3修改认证类 通过查询数据来认证

修改UserDetailsServiceImpl.java ，添加属性和setter方法，修改loadUserByUsername方法

|  |
| --- |
| /\*\*  \* 认证类  \* **@author** Administrator  \*  \*/  **Public class** UserDetailsServiceImpl **implements** UserDetailsService {  **private** SellerService sellerService;  **public void** setSellerService(SellerService sellerService) {  **this**.sellerService = sellerService;  }  @Override  **public** UserDetails loadUserByUsername(String username) **throws**UsernameNotFoundException {  System.***out***.println("经过了UserDetailsServiceImpl");  //构建角色列表  List<GrantedAuthority>grantAuths=**new**ArrayList();  grantAuths.add(**new**SimpleGrantedAuthority("ROLE\_SELLER"));  //得到商家对象  TbSellerseller = sellerService.findOne(username);  **if**(seller!=**null**){  **if**(seller.getStatus().equals("1")){  **return new** User(username,seller.getPassword(),grantAuths);  }**else**{  **return null**;  }  }**else**{  **returnnull**;  }  }  } |

修改pinyougou-shop-web的spring-security.xml ，添加如下配置

|  |
| --- |
| <!-- 引用dubbo服务 -->  <**dubbo:application name="pinyougou-shop-web"**/> <**dubbo:registry address="zookeeper://192.168.25.133:2181"**/> <**dubbo:reference id="sellerService" interface="com.pinyougou.sellergoods.service.SellerService"**></**dubbo:reference**>  <**beans:bean class="com.pinyougou.shop.service.ShopUserDetailService" id="shopUserDetailService"**>  <**beans:property name="sellerService" ref="sellerService"**></**beans:property**> </**beans:bean**> |

经过上述修改后，在登陆页输入用户名和密码与数据库一致即可登陆。

## 5.4密码加密

### 5.4.1 BCrypt加密算法

用户表的密码通常使用MD5等不可逆算法加密后存储，为防止彩虹表破解更会先使用一个特定的字符串（如域名）加密，然后再使用一个随机的salt（盐值）加密。特定字符串是程序代码中固定的，salt是每个密码单独随机，一般给用户表加一个字段单独存储，比较麻烦。BCrypt算法将salt随机并混入最终加密后的密码，验证时也无需单独提供之前的salt，从而无需单独处理salt问题。

### 5.4.2商家入驻密码加密存储

商家申请入驻的密码要使用BCrypt算法进行加密存储，修改SellerController.java的add方法

|  |
| --- |
| @Autowired **private** PasswordEncoder **passwordEncoder**; /\*\*  \* 增加  \* **@param *seller*** \* **@return** \*/ @RequestMapping(**"/add"**) **public** Result add(@RequestBody TbSeller seller){  **try** {  seller.setPassword(**passwordEncoder**.encode(seller.getPassword()));  **sellerService**.add(seller);  **return new** Result(**true**, **"增加成功"**);  } **catch** (Exception e) {  e.printStackTrace();  **return new** Result(**false**, **"增加失败"**);  } } |

### 5.4.3加密配置

修改pinyougou-shop-web的spring-security.xml ，添加如下配置

|  |
| --- |
| <beans:bean id=*"bcryptEncoder"*  class=*"org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"*/> |

修改认证管理器的配置

|  |
| --- |
| <!-- 认证管理器 -->  <authentication-manager alias=*"authenticationManager"*>  <authentication-provider user-service-ref=*'userDetailService'*>  <password-encoder ref=*"bcryptEncoder"*></password-encoder>  </authentication-provider>  </authentication-manager> |

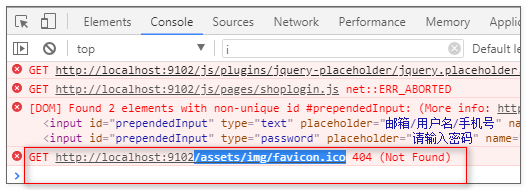
## 5.5显示登录名

参照运营商后台

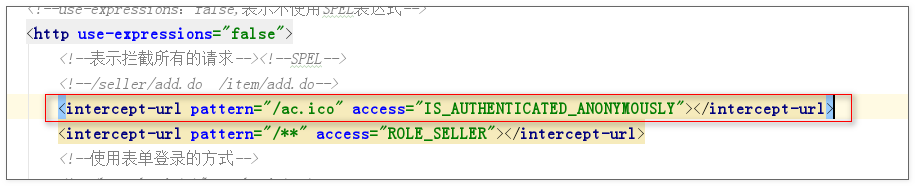
## 5.6退出登录

参照运营商后台。

如果出现 ico的错：



修改成：如图所示。



自定义认证类 ：

从数据库中查询相关的数据 根据密码是加密来进行匹配。

主要是有3个接口：

UserDetailsService ---->方法：去获取数据库中的用户信息（认证的信息 密码 是否被冻结）

UserDetails---->User :保存用户认证的信息（）

passwordEncoder --->bcrypt