1. Shiro同SSH集成综合案例

本章主要介绍Shiro同Struts、Spring、Hibernate框架的集成。重点把握：

* 基于Spring动态初始化Shiro安全对象
* 自定义Realm同Spring管理的Service对象进行数据交互
* Web中配置Shiro拦截Struts请求
* 基于Shiro实现加密密码的保存和验证

目录

[12 Shiro同Struts2和Spring集成 1](#_Toc495660159)

[12.1 集成思路 2](#_Toc495660160)

[12.2 实现Shiro Struts和Spring的集成 2](#_Toc495660161)

[12.2.1 创建基于Maven的web工程 2](#_Toc495660162)

[12.2.2 编写基于Struts+Spring的测试应用 6](#_Toc495660163)

[12.2.3 编写Shiro安全框架的自定义组件 11](#_Toc495660164)

[12.2.4 编写初始化Shiro安全框架对象的代码 14](#_Toc495660165)

[12.2.5 在Struts+Spring的测试应用上添加Shiro 20](#_Toc495660166)

[12.2.6 发布测试应用 21](#_Toc495660167)

[12.3 总结 21](#_Toc495660168)

前面章节我们已经介绍了Shiro常用的基本组件和Shiro同Jsp/Servlet、Shiro同Struts、Shiro同Spring以及Shiro同Struts+Spring的集成，接下来我们介绍Shiro同Struts+Spring+Hibernate的集成综合案例，进一步接近实战的方式使用Shiro实现应用中的安全控制。**特别强调，要顺利理解本章的内容，强烈需要读者熟悉Struts+Spring+Hibernate的框架开发**，因为本章是基于已有的Struts+Spring+Hibernate案例基础上，添加的Shiro相关的内容。（Struts+Spring+Hibernate开发相关的资料，请查看<http://cyedu.ke.qq.com>或<http://study.163.com/provider/1018463678/course.htm>视频等资料）。

本章分两部分进行，第一部分简单介绍SSH案例的基本功能和关键代码，第二部分详细介绍在SSH案例集成上集成shiro。

* 1. SSH案例简介

为了方便我们专注Shiro同SSH的集成研究和理解，我们这里的SSH案例，只是提供了《在线考试系统》添加用户，添加、修改和查询所有权限资源功能。接下来先给大家介绍一下这些功能的演示、表结构和关键接口代码，以便后面的Shiro集成和测试。在继续后面的研究前，请做好如下准备工作：

* 安装配置好开发环境

开发环境是Maven+Eclipse

运行环境是Tomca7

JDK为1.7以上

* 导入ShiroPOM和ShiroTP13工程

ShiroPOM是Maven的父工程，需要导入并且install到本地仓库，如果在前面章节有做过，可以忽略。

* 初始化数据库

安装好MySQL数据库，在MySQL数据库软件软件中添加demo\_shiro数据库，不建议使用其他数据库名称，如果自定义了其他数据库名称，请同步修改每个hibernate实体映射文件中的catalog属性。同时请在创建数据库时，请注意设置utf-8编码。

数据库用户名和密码是root和root，如果我们有自定义，请同步修改applicationContex.xml中的对应配置。

* 发布ShiroTP13工程应用

基于pom .xml中配置好的tomcat7插件，发布ShiroTP13应用。

* + 1. SSH案例功能演示

1. 添加用户

在浏览器地址栏中输入<http://localhost:9090/ShiroTP13/user/addUser.jsp，出现如下图13-1添加用户> 页面。



图13-1 添加用户

输入用户名和密码（其他项暂时没有意义，可以不用输入），点击提交按钮，弹出提示如下图13-2添加成功。

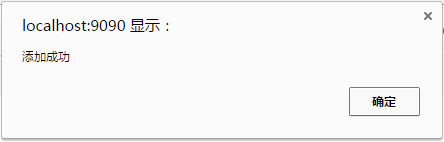


图13-2 添加成功

1. 查询所有资源

在浏览器地址栏中输入http://localhost:9090/ShiroTP13/resourceAction!findAll.  
action，出现如下 图13-3资源列表 页面。



图13-3 资源列表

1. 添加资源

点击图13-3 资源列表 图中的新增资源按钮，出现如下图13-4 新增资源 页面。

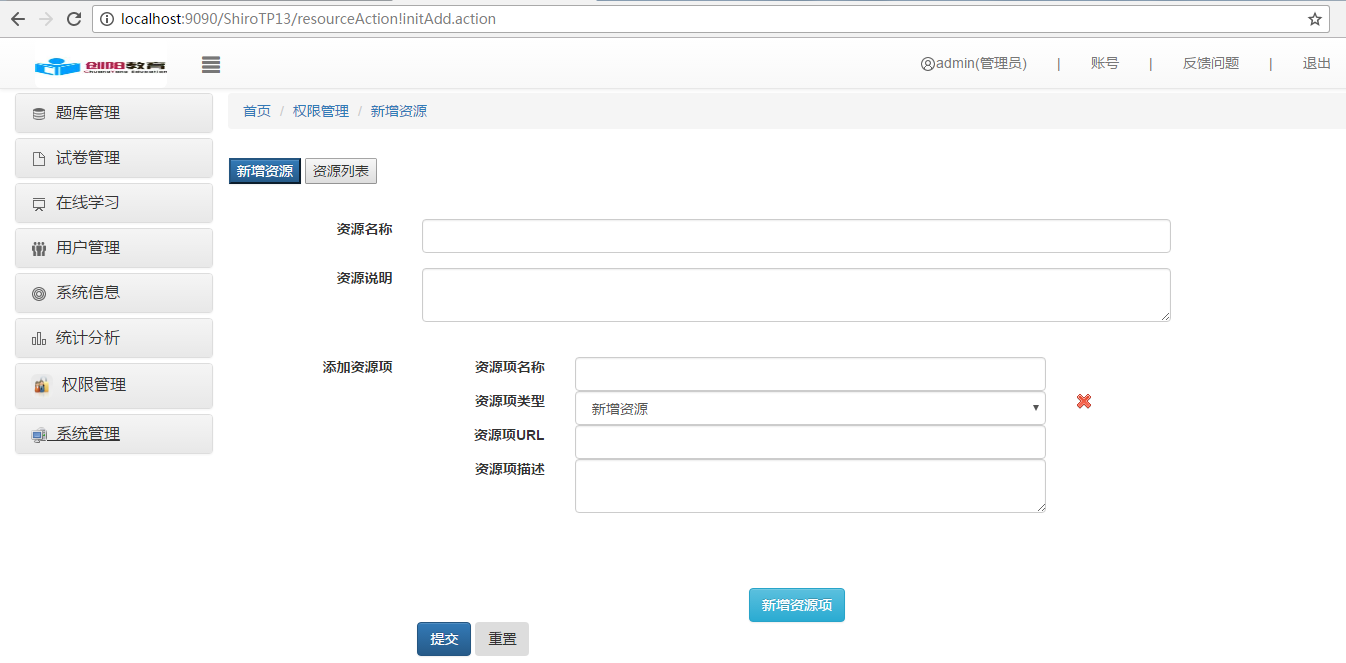


图13-4 新增资源

通过该界面可以添加新的资源和资源项。

1. 修改资源

点击图13-3资源列表 中的某个资源，也可进入如图13-5修改资源 界面。

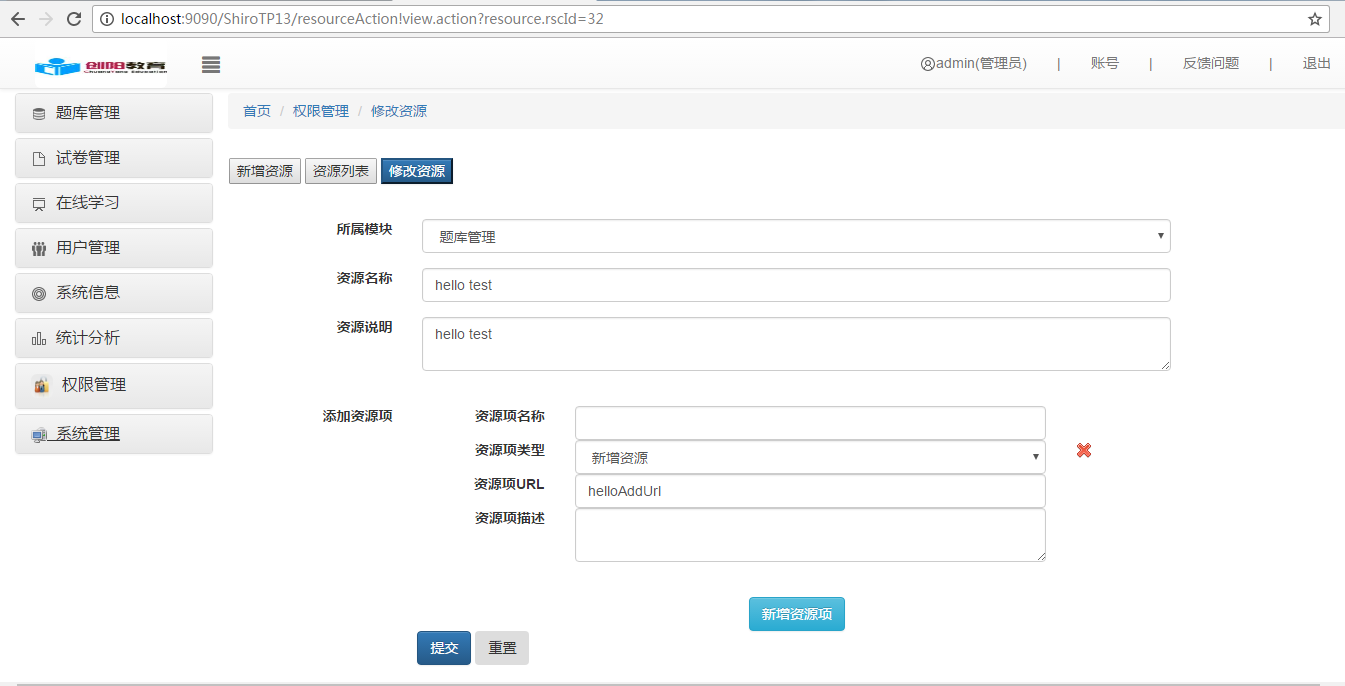


图13-5 修改资源

通过该界面可以修改资源和资源项信息。

* + 1. SSH案例表简介

1. Shr\_resource

资源表，保存资源名称、资源说明信息

1. Shr\_resource\_item

资源项表，保存对资源的操作项信息：操作类型（1,2,4,8分别表示添加，修改，删除，查询），url和说明

1. Shr\_role

角色表，保存角色名称和角色说明

1. Shr\_role\_permission

角色权限表，保存每个角色对每个资源的操作权限，以二进制的形式同时记录各种操作。共四位，对应的是 查询、删除、修改、添加。如1111表示CRUD都可以；1000表示可以查询，删除、修改和添加不允许；1010表示查询，修改可以，删除和添加不允许。

1. Shr\_session

Shiro会话表，保存shiro的session对象，暂时没用到

1. Shr\_ur\_rl\_map

用户角色映射表，保存用户和角色的对应关系（中间表）

1. Shr\_user

用户表，保存用户信息

1. Shr\_user\_permission

用户权限表，保存每个用户对每个资源的操作权限，记录形式同shr\_role\_permission。

* + 1. SSH案例关键代码

1. Action类

cn.com.shiro.book.tp12.action包下的类为Struts的Action类，当前案例里面用到了ResourceAction和UserAction。

ResourceAction里面包含维护资源相关的方法

1. view()方法查询当个资源
2. initAdd()方法初始化添加资源页面数据
3. findAll()方法查询所有资源
4. add()方法添加资源
5. edit()方法修改资源

UserAction里面实现添加用户的方法

1. add()方法添加用户
2. Service类

cn.com.shiro.book.tp12.service包下定义了相关service的接口，每个方法都有详细的文档说明，请查看随书文档。

cn.com.shiro.book.tp12.service.impl包下是每个service接口的实现类。

1. Dao类

cn.com.shiro.book.tp12.dao包下定义了所有dao的接口

cn.com.shiro.book.tp12.hibernate.dao是对每个实体基于Hibernate的持久化方法

cn.com.shiro.book.tp12.hibernate.dao.impl包下是dao接口的实现类，里面的封装了hibernate.dao的持久化方法。

1. Entity类

cn.com.shiro.book.tp12.entity包下是所有实体类。

* 1. 集成Shiro

为了方便区分理解SSH和Shiro的代码，我们单独创建一个新的工程ShiroTP130，将ShiroTP13里面的代码全部移植到ShiroTP130中，在ShiroTP130中添加Shiro相关的代码，并且实现SSH同Shiro的集成。

* + 1. 添加Shiro依赖

在pom.xml中添加Shiro相关的依赖，样例代码块如下，详细内容请查看随书代码code/ShiroTP130/pom.xml。

<!-- shiro begin -->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>${shiro.version}</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-web</artifactId>

<version>${shiro.version}</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-cas</artifactId>

<version>${shiro.version}</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>${shiro.version}</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-ehcache</artifactId>

<version>${shiro.version}</version>

</dependency>

<!-- ehcache -->

<dependency>

<groupId>net.sf.ehcache</groupId>

<artifactId>ehcache-core</artifactId>

<version>2.5.3</version>

</dependency>

* + 1. 自定义Permission和PermissionResolver

1. 自定义Permission

在cn.com.shiro.book.tp12.shiro.authc.permission包下创建BitPermission类，实现shiro的Permission接口，实现基于二进制的权限封装和匹配。样例代码块如下，详细内容请查看随书代码[code/ShiroTP130/src/main/java/cn/com/shiro/book/tp12/shiro/authz/  
permission/BitPermission.java](../code/ShiroTP130/src/main/java/cn/com/shiro/book/tp12/shiro/authz/permission/BitPermission.java)。

package cn.com.shiro.book.tp12.shiro.authz.permission;

… …

/\*\*

\* 权限对象，实现比较逻辑

\* \*/

public class BitPermission implements Permission {

private int resourceId = -1;

private int operateCode = -1;

private int instanceId = -1;

/\*\*

\* +资源id+操作码+对象id

\* \*/

public BitPermission(String perm) {

String[] arrays = perm.split("\\+");

if (arrays.length > 1) {

this.resourceId = Integer.parseInt(arrays[1]);

}

if (arrays.length > 2) {

this.operateCode = Integer.parseInt(arrays[2]);

}

if (arrays.length > 3) {

this.instanceId = Integer.parseInt(arrays[3]);

}

}

public BitPermission(int resourceId, int operateCode) {

this.resourceId = resourceId;

this.operateCode = operateCode;

}

public BitPermission(int resourceId, int operateCode, int instanceId) {

this.resourceId = resourceId;

this.operateCode = operateCode;

this.instanceId = instanceId;

}

@Override

// 实现两个权限对象的比较

public boolean implies(Permission p) {

boolean bool = false;

if (p != null && p instanceof BitPermission) {

BitPermission bitPrm = (BitPermission) p;

if (this.hasAll() || bitPrm.hasAll()) {

bool = true;

} else {

System.out.println("target->" + bitPrm.resourceId + ":"

+ bitPrm.operateCode);

bool = (this.resourceId == bitPrm.resourceId)

&& ((this.operateCode & bitPrm.operateCode) == this.operateCode || (this.operateCode & bitPrm.operateCode) == bitPrm.operateCode);

}

}

return bool;

}

private boolean hasAll() {

return this.resourceId == -1 && this.operateCode == -1

&& this.instanceId == -1;

}

… …

}

1. 自定义PermissionResolver

在cn.com.shiro.book.tp12.shiro.authc.permission包下创建BitPermissionResolver类，实现shiro的PermissionResolver接口，实现 +资源id+操作code+实例id 格式字符串同BitPermission的解析。样例代码块如下，详细内容请查看随书代码[code/ShiroTP130/src/main/java/cn/com/shiro/book/tp12/shiro/authz/permission/BitPermissionResolver.java](../code/ShiroTP130/src/main/java/cn/com/shiro/book/tp12/shiro/authz/permission/BitPermissionResolver.java)。

package cn.com.shiro.book.tp12.shiro.authz.permission;

… …

/\*\*

\* 基于权限字符串，解析出权限对象

\* \*/

public class BitPermissionResolver implements PermissionResolver {

// 只能解析+开头的权限字符串

@Override

public Permission resolvePermission(String permissionString) {

if (permissionString.startsWith("+")) {

return new BitPermission(permissionString);

}

throw new AuthorizationException("权限描述格式不支持["+permissionString+"]");

}

}

* + 1. 自定义Realm

在cn.com.shiro.book.tp12.shiro.realm包下创建ShiroRealm类，继承AuthorizingRealm类，实现doGetAuthorizationInfo和doGetAuthenticationInfo方法，获取授权控制信息和身份认证信息，样例代码如下，详细内容请查看随书代码[code/ShiroTP130/src/  
main/java/cn/com/shiro/book/tp12/shiro/realm/ShiroRealm.java](../code/ShiroTP130/src/main/java/cn/com/shiro/book/tp12/shiro/realm/ShiroRealm.java)。

package cn.com.shiro.book.tp12.shiro.realm;

… …

public class ShiroRealm extends AuthorizingRealm {

private IUserService userService;

/\*\*

\* 获取当前用户在数据库中记录的权限

\* \*/

@Override

protected AuthorizationInfo doGetAuthorizationInfo(

PrincipalCollection principals) {

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

// 获取当前用户

String userName = principals.getPrimaryPrincipal().toString();

// 如果是admin

if ("admin".equals(userName)) {

// 拥有所有的权限

info.addObjectPermission(new BitPermission("+"));

} else {

// 调用userService的方法，获取数据库中当前用户的所有权限实体

List<ShrPermission> shrPermList = userService

.findPermissions(userName);

// 将数据库中的权限实体信息，转换成BitPermission对象，添加到info

if (shrPermList != null && shrPermList.size() > 0) {

// 创建一个存放BitPermission对象的集合

Set<Permission> permissions = new HashSet<Permission>();

BitPermission bitPerm = null;

// 循环获取数据库中的权限实体

for (ShrPermission shrPerm : shrPermList) {

// 将权限实体转变成BitPermission对象

bitPerm = new BitPermission(shrPerm.getShrResource()

.getRscId(), shrPerm.getPrmOperationCode());

// 添加到BitPermission对象集合

permissions.add(bitPerm);

}

// 将所有的BitPermission对象，添加info

info.setObjectPermissions(permissions);

}

}

return info;

}

// shiro验证 token(userName,password) info(userName,password)

@Override

protected AuthenticationInfo doGetAuthenticationInfo(

AuthenticationToken token) throws AuthenticationException {

// 获取身份标识(用户名)

// 根据用户找到密码和盐

// 获取私盐，拼凑新的密码（加密后验证）

// 封装到Info中返回

SimpleAuthenticationInfo info = null;

// 获取身份标识(用户名)

UsernamePasswordToken pswToken = (UsernamePasswordToken) token;

String userName = pswToken.getUsername();

// subject的密码

String password = new String(pswToken.getPassword());

// 根据用户找到密码和盐

String[] result = this.userService

.findPasswordAndSaltByUserName(userName);

// 获取私盐，拼凑新的密码（加密后验证）

String data = result[1].substring(result[1].length() - 1);

int index = Integer.parseInt(data);

String privateSalt = result[1].substring(index, 5);

// 在subject的密码上加上私盐，更新token

pswToken.setPassword((privateSalt + password).toCharArray());

// 封装到Info中返回

info = new SimpleAuthenticationInfo(userName, result[0], this.getName());

return info;

}

public void setUserService(IUserService userService) {

this.userService = userService;

}

}

其中依赖passwordService属性，基于passwordService属性对象，实现加密密码的验证。

* + 1. 自定义ShiroFilterFactoryBean

在cn.com.shiro.book.tp12.shiro.config包下创建MyShiroFilterFactoryBean，继承Shiro安全框架中的ShiroFilterFactoryBean，实现Spring中的InitialzingBean接口。

继承ShiroFilterFactoryBean类扩展初始化Shiro安全对象的功能

实现InitialzingBean接口，目的是在afterPropertiesSet方法中编写初始化对象，并且注入好所有属性后需要执行的业务逻辑。

样例代码如下，详细内容请好看随书代码[code/ShiroTP130/src/main/java/cn/com/shiro/  
book/tp12/shiro/config/MyShiroFilterFactoryBean.java](../code/ShiroTP130/src/main/java/cn/com/shiro/book/tp12/shiro/config/MyShiroFilterFactoryBean.java)。

package cn.com.shiro.book.tp12.shiro.config;

… …

public class MyShiroFilterFactoryBean extends ShiroFilterFactoryBean implements

InitializingBean {

private IResourceItemService itemService;

// 初始化好对象后再执行

@Override

public void afterPropertiesSet() throws Exception {

this.setFilterChainDefinitions(this.getChainDefinitions());

this.setFilterAttributes(this);

// 注册全局SecurityManager对象

SecurityUtils.setSecurityManager(this.getSecurityManager());

}

// 初始化authc、logout、roles、perms和authcBasic拦截器

private void setFilterAttributes(ShiroFilterFactoryBean shiroFilter) {

Map<String, Filter> filters = shiroFilter.getFilters();

FormAuthenticationFilter authc = (FormAuthenticationFilter) filters

.get("authc");

if (authc == null) {

authc = new FormAuthenticationFilter();

// 注册authc拦截器

filters.put("authc", authc);

}

// 设置authc的属性

authc.setSuccessUrl("/resourceAction!findAll.action");

authc.setLoginUrl("/login.jsp");

authc.setUsernameParam("userName");

authc.setPasswordParam("password");

LogoutFilter logout = (LogoutFilter) filters.get("logout");

if (logout == null) {

logout = new LogoutFilter();

filters.put("logout", logout);

}

logout.setRedirectUrl("/login.jsp");

RolesAuthorizationFilter roles = (RolesAuthorizationFilter) filters

.get("roles");

if (roles == null) {

roles = new RolesAuthorizationFilter();

filters.put("roles", roles);

}

roles.setUnauthorizedUrl("/unauthorized.jsp");

PermissionsAuthorizationFilter perms = (PermissionsAuthorizationFilter) filters

.get("perms");

if (perms == null) {

perms = new PermissionsAuthorizationFilter();

filters.put("perms", perms);

}

perms.setUnauthorizedUrl("/unauthorized.jsp");

BasicHttpAuthenticationFilter authcBasic = (BasicHttpAuthenticationFilter) filters

.get("authcBasic");

if (authcBasic == null) {

authcBasic = new BasicHttpAuthenticationFilter();

filters.put("authcBasic", authcBasic);

}

// authcBasic.setLoginUrl("/login.jsp");

authcBasic.setSuccessUrl("/resourceAction!findAll.action");

authcBasic.setApplicationName("ShiroBook 登录");

}

private String getChainDefinitions() {

StringBuilder builder = new StringBuilder();

builder.append("/login.jsp=authc\r\n");

builder.append("/logout=logout\r\n");

// 获取所有的资源项对象

List<ShrResourceItem> itemList = this.itemService

.searchAllResourceItems();

if (itemList != null) {

// 循环每个资源项

for (ShrResourceItem item : itemList) {

// 判断一定要有url /user/addUser.jsp=perms[+3+5]\r\n

if (StringUtils.hasLength(item.getItmUrl())) {

// url

builder.append(item.getItmUrl());

builder.append("=perms[+");

builder.append(item.getShrResource().getRscId());

builder.append("+");

builder.append(item.getItmType());

builder.append("]\r\n");

}

}

}

builder.append("/\*\*=anon\r\n");

System.out.println(builder.toString());

return builder.toString();

}

public void setItemService(IResourceItemService itemService) {

this.itemService = itemService;

}

}

* + 1. 基于Spring管理Shiro对象

修改src/main/resources目录下的applicationContext.xml文件，在里面添加如下代码块，配置Shiro安全框架相关对象，样例代码块如下，详细内容请查看随书代码code/  
ShiroTP130/src/main/resources/applicationContext.xml。

… …

<bean id="passwordService" class="org.apache.shiro.authc.credential.DefaultPasswordService">

<property name="hashService"><!-- ddd -->

<bean class="org.apache.shiro.crypto.hash.DefaultHashService"></bean>

</property>

<property name="hashFormatFactory">

<bean class="org.apache.shiro.crypto.hash.format.DefaultHashFormatFactory"></bean>

</property>

<property name="hashFormat">

<bean class="org.apache.shiro.crypto.hash.format.Shiro1CryptFormat"></bean>

</property>

</bean>

<bean id="randomNumberGenerator" class="org.apache.shiro.crypto.SecureRandomNumberGenerator"></bean>

<bean id="passwordMatcher" class="org.apache.shiro.authc.credential.PasswordMatcher">

<property name="passwordService" ref="passwordService"></property>

</bean>

<bean id="shiroRealm" class="cn.com.shiro.book.tp12.shiro.realm.ShiroRealm">

<property name="credentialsMatcher" ref="passwordMatcher"></property>

<property name="userService" ref="userServiceImpl"></property>

</bean>

<!-- 权限字符串解析器 -->

<bean id="permResolver" class="cn.com.shiro.book.tp12.shiro.authz.permission.BitPermissionResolver"></bean>

<!-- 授权控制器 -->

<bean id="authorizer" class="org.apache.shiro.authz.ModularRealmAuthorizer">

<property name="permissionResolver" ref="permResolver"></property>

</bean>

<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

<!-- 一定要先设置授权器，再设置域，否则会抛出找不到域异常 -->

<property name="authorizer" ref="authorizer"></property>

<!-- <property name="realm" ref="shiroRealm"></property>

-->

<property name="realms">

<list>

<ref bean="shiroRealm"/>

</list>

</property>

</bean>

<bean id="lifecycleBeanPostProcessor" class="org.apache.shiro.spring.LifecycleBeanPostProcessor"></bean>

<bean name="shiroFilter" class="cn.com.shiro.book.tp12.shiro.config.MyShiroFilterFactoryBean">

<property name="securityManager" ref="securityManager"></property>

<!-- FilterChainDefinitions -->

<property name="itemService" ref="resourceItemServiceImpl"></property>

</bean>

… …

* + 1. 配置Shiro入口

修改src/main/webapp/WEB-INF/web.xml，在里面添加Shiro的配置，样例代码如下，详细内容请查看随书代码code/ShiroTP130/src/main/webapp/WEB-INF/web.xml。

<!-- The filter-name matches name of a 'shiroFilter' bean inside applicationContext.xml

-->

<filter>

<filter-name>shiroFilter</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

<init-param>

<param-name>targetFilterLifecycle</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<!-- Make sure any request you want accessible to Shiro is filtered. /\* catches all -->

<!-- requests. Usually this filter mapping is defined first (before all others) to -->

<!-- ensure that Shiro works in subsequent filters in the filter chain: -->

<filter-mapping>

<filter-name>shiroFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

**注意filter-mapping需要配置在struts的filter-mapping之前。**

* + 1. 测试

1. 发布应用

将ShiroTP130发布到Tomcat7服务器中。

1. 在数据库中初始化admin用户

在shr\_user表从插入一条记录，ur\_user\_name,ur\_password, ur\_salt的值分别是admin, $shiro1$SHA-512$1$$dZMkxWAiOBzSEjk+w0b2+z1/NX3omaw0boHcvMMHaLQ6b8+4E2X0OpF0ts4QWkEF1Npmr9P+mGk9f++pt+MUqQ==, yNlac+PHQr7PQ4Yu4cv3VQ==0。如shr\_user表中ur\_id为20的记录。用户名和密码都是admin。

1. 基于admin登录系统

在浏览器中输入<http://localhost:9090/ShiroTP130/login.jsp。出现如下图13-6>登录界面。



图13-6 登录界面

在用户名和密码后面输入框中，输入admin和admin，点击登录，进入如下图13-7资源列表 界面。



图13-7 资源列表

1. 添加用户资源和资源项

点击图13-7资源列表 中的新增资源按钮，进入如下图13-8添加资源 界面。在里面输入要添加的用户资源和用户资源项信息：

资源名称：维护用户

资源说明：实现用户的增删改查…

两个用户资源项：

显示添加用户页面的资源项

资源类型：新增资源

资源项URL：/user/addUser.jsp

资源项描述：添加用户的初始化页面URL

接收添加用户请求的资源项

资源类型：新增资源

资源项URL：/userAction!add.action

资源项描述：处理添加用户请求的URL



图13-8添加用户

点击提交，添加用户资源。

1. 添加权限资源和权限资源项

通过新增资源界面，添加 维护资源和维护资源项：

维护资源内容：

资源名称：维护资源

资源说明：实现资源的CRUD

资源项内容：

添加资源初始化页面项：

资源项类型：新增资源

资源项URL：/resourceAction!initAdd.action

资源项描述：初始化添加资源的页面URL

处理添加资源的请求项

资源项类型：新增资源

资源项URL：/resourceAction!add.action

资源项描述：处理添加资源的请求URL

修改资源项

资源项类型：修改资源

资源项URL：/resourceAction!edit.action

资源项描述：修改资源请求URL

查询资源列表项

资源项类型：查询资源

资源项URL：/resourceAction!findAll.action

资源项描述：查看所有资源列表URL

查看单个资源项

资源项类型：查询资源

资源项URL：/resourceAction!view.action

资源项描述：查看资源URL

1. 添加测试用户

在浏览器地址栏中输入<http://localhost:9090/ShiroTP130/user/addUser.jsp>。通过该界面添加两个用户，testUser1和testUser2，密码都是123456。

1. 给添加的testUser1和testUser2赋权限

因为我们暂时没有实现给用户分配权限的页面功能，需要在数据库表中手动给testUser1和testUser2两个用户分配不同的权限。请查看在shr\_user表中testUser1和testUser2的ur\_id，我们样例数据库记录分别是24和25。同时查看shr\_resource表中维护用户和维护资源两个资源的id，我们样例数据库记录分别是33和34。

为了简便，我们将直接在shr\_user\_permission表中，分配testUser1和testUser2的权限。（admin用户的权限默认是所有）

* 给testUser1赋予可以查看资源和添加用户权限

在shr\_user\_permission中新增一条记录，prm\_ur\_id, prm\_rsc\_id,prm\_operation\_code分别是24， 34， 8，表示testUser1用户可以操作 维护资源里面的查看资源操作。

在shr\_user\_permission表中新增一条记录，prm\_ur\_id, prm\_rsc\_id,prm\_operation\_code分别是24， 33， 1，表示testUser1用户可以操作 维护用户里面的新增用户操作。

* 给testUser2赋予可以查看资源、查看单个、新增资源、修改资源权限

在shr\_user\_permission表中新增一条记录，prm\_ur\_id, prm\_rsc\_id,prm\_operation\_code分别是25， 34， 11，表示testUser1用户可以操作 维护资源里面的新增、修改、查询操作。

1. 浏览器地址栏中输入<http://localhost:9090/ShiroTP130/logout>退出登录，返回到登录页面
2. 测试授权控制

使用testUser1用户，我们可以登录并且查看资源列表页面，不能操作添加资源请求，但是可以添加新用户；

使用testUser2用户，我们可以登录查看、添加、修改资源，但是不能添加新用户。

注意，如果不想麻烦，可以直接按随书代码[code/ShiroTP130/shr\_ssh.sql](../code/ShiroTP130/shr_ssh.sql)初始化数据库数据进行测试。

* 1. 总结

本章主要介绍了Shiro同SSH的集成步骤和注意事项，需要我们自己动手才能真正体会每个步骤。