shiro第二天 shiro授权 shiro和企业项目整合开发

# 复习

什么是权限管理？

权限管理是系统的安全范畴，要求必须是合法的用户才可以访问系统（用户认证），且必须具有该 资源的访问权限才可以访问该 资源（授权）。

认证：对用户合法身份的校验，要求必须是合法的用户才可以访问系统。

授权：访问控制，必须具有该 资源的访问权限才可以访问该 资源。

权限模型：标准权限数据模型包括 ：用户、角色、权限（包括资源和权限）、用户角色关系、角色权限关系。

权限分配：通过UI界面方便给用户分配权限，对上边权限模型进行增、删、改、查操作。

权限控制：

基于角色的权限控制：根据角色判断是否有操作权限，因为角色的变化 性较高，如果角色修改需要修改控制代码，系统可扩展性不强。

基于资源的权限控制：根据资源权限判断是否有操作权限，因为资源较为固定，如果角色修改或角色中权限修改不需要修改控制代码，使用此方法系统可维护性很强。建议使用。

权限管理的解决方案：

* 对于粗颗粒权限管理，建议在系统架构层面去解决，写系统架构级别统一代码（基础代码）。

粗颗粒权限：比如对系统的url、菜单、jsp页面、页面上按钮、类方法进行权限管理，即对资源类型进行权限管理。

* 对于细颗粒权限管理：

粗颗粒权限：比如用户id为001的用户信息（资源实例）、类型为t01的商品信息（资源实例），对资源实例进行权限管理，理解对数据级别的权限管理。

细颗粒权限管理是系统的业务逻辑，业务逻辑代码不方便抽取统一代码，建议在系统业务层进行处理。

基于url的权限管理（掌握）：

企业开发常用的方法，使用web应用中filter来实现，用户请求url，通过filter拦截，判断用户身份是否合法（用户认证），判断请求的地址是否是用户权限范围内的url(授权)。

shiro：

shiro是一个权限管理框架，是apache下的开源项目。相比spring security框架更简单灵活，spring security对spring依赖较强。shiro可以实现web系统、c/s、分布式等系统 权限管理。

shiro认证流程：（掌握）

1、subject(主体)请求认证，调用subject.login(token)

2、SecurityManager (安全管理器)执行认证

3、SecurityManager通过ModularRealmAuthenticator进行认证。

4、ModularRealmAuthenticator将token传给realm，realm根据token中用户信息从数据库查询用户信息（包括身份和凭证）

5、realm如果查询不到用户给ModularRealmAuthenticator返回null，ModularRealmAuthenticator抛出异常（用户不存在）

6、realm如果查询到用户给ModularRealmAuthenticator返回AuthenticationInfo(认证信息)

7、ModularRealmAuthenticator拿着AuthenticationInfo(认证信息)去进行凭证（密码 ）比对。如果一致则认证通过，如果不致抛出异常（凭证错误）。

subject：主体

Authenticator：认证器（ shiro提供）

realm（一般需要自定义）：相当于数据源，认证器需要realm从数据源查询用户身份信息及权限信息。

# 课程安排

1、shiro授权

通过测试代码讲解基于角色权限控制，基于资源的权限控制（掌握）

2、shiro与spring进行整合

项目框架springmvc+mybatis+shiro

3、在整合工程下学习：

认证

授权

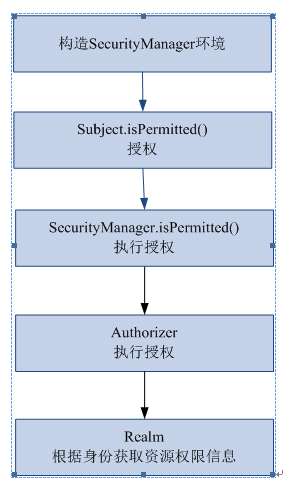
sessionManager

缓存管理

....

# shiro授权

## 授权流程



## 三种授权方法

Shiro 支持三种方式的授权：

* 编程式：通过写if/else 授权代码块完成：

Subject subject = SecurityUtils.getSubject();

if(subject.hasRole(“admin”)) {

//有权限

} else {

//无权限

}

* 注解式：通过在执行的Java方法上放置相应的注解完成：

@RequiresRoles("admin")

public void hello() {

//有权限

}

* JSP/GSP 标签：在JSP/GSP 页面通过相应的标签完成：

<shiro:hasRole name="admin">

<!— 有权限—>

</shiro:hasRole>

## shiro-permission.ini

shiro-permission.ini里边的内容相当于在数据库。

#用户

[users]

#用户zhang的密码是123，此用户具有role1和role2两个角色

zhang=123,role1,role2

wang=123,role2

#权限

[roles]

#角色role1对资源user拥有create、update权限

role1=user:create,user:update

#角色role2对资源user拥有create、delete权限

role2=user:create,user:delete

#角色role3对资源user拥有create权限

role3=user:create

权限标识符号规则：资源:操作:实例(中间使用半角:分隔)

user：create:01 表示对用户资源的01实例进行create操作。

user:create：表示对用户资源进行create操作，相当于user:create:\*，对所有用户资源实例进行create操作。

user：\*：01 表示对用户资源实例01进行所有操作。

## 程序编写



# 自定义realm进行授权

## 需求

上边的程序通过shiro-permission.ini对权限信息进行静态配置，实际开发中从数据库中获取权限数据。就需要自定义realm，由realm从数据库查询权限数据。

realm根据用户身份查询权限数据，将权限数据返回给authorizer（授权器）。

## 自定义realm

在原来自定义的realm中，修改doGetAuthorizationInfo方法。



## shiro-realm.ini

在shiro-realm.ini中配置自定义的realm，将realm设置到securityManager中。

[main]

#自定义 realm

customRealm=cn.itcast.shiro.realm.CustomRealm

#将realm设置到securityManager，相当 于spring中注入

securityManager.realms=$customRealm

## 测试程序

// 自定义realm进行资源授权测试

@Test

**public** **void** testAuthorizationCustomRealm() {

// 创建SecurityManager工厂

Factory<SecurityManager> factory = **new** IniSecurityManagerFactory(

"classpath:shiro-realm.ini");

// 创建SecurityManager

SecurityManager securityManager = factory.getInstance();

// 将SecurityManager设置到系统运行环境，和spring后将SecurityManager配置spring容器中，一般单例管理

SecurityUtils.*setSecurityManager*(securityManager);

// 创建subject

Subject subject = SecurityUtils.*getSubject*();

// 创建token令牌

UsernamePasswordToken token = **new** UsernamePasswordToken("zhangsan",

"111111");

// 执行认证

**try** {

subject.login(token);

} **catch** (AuthenticationException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.*out*.println("认证状态：" + subject.isAuthenticated());

// 认证通过后执行授权

// 基于资源的授权，调用isPermitted方法会调用CustomRealm从数据库查询正确权限数据

// isPermitted传入权限标识符，判断user:create:1是否在CustomRealm查询到权限数据之内

**boolean** isPermitted = subject.isPermitted("user:create:1");

System.*out*.println("单个权限判断" + isPermitted);

**boolean** isPermittedAll = subject.isPermittedAll("user:create:1",

"user:create");

System.*out*.println("多个权限判断" + isPermittedAll);

// 使用check方法进行授权，如果授权不通过会抛出异常

subject.checkPermission("items:add:1");

}

## 授权流程

1、对subject进行授权，调用方法isPermitted（"permission串"）

2、SecurityManager执行授权，通过ModularRealmAuthorizer执行授权

3、ModularRealmAuthorizer执行realm（自定义的CustomRealm）从数据库查询权限数据

调用realm的授权方法：doGetAuthorizationInfo

4、realm从数据库查询权限数据，返回ModularRealmAuthorizer

5、ModularRealmAuthorizer调用PermissionResolver进行权限串比对

6、如果比对后，isPermitted中"permission串"在realm查询到权限数据中，说明用户访问permission串有权限，否则 没有权限，抛出异常。

# shiro与项目整合

## 需求

将原来基于url的工程改成使用shiro实现。

## 创建新工程

创建web工程 ：permission\_shiro1110

## 去除原工程 的认证和授权的拦截

删除springmvc.xml中：

<mvc:interceptors>

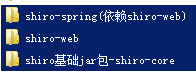
## jar包

包括 ：

shiro-web的jar、

shiro-spring的jar

shiro-code的jar



## web.xml中配置shiro的filter

在web系统中，shiro也通过filter进行拦截。filter拦截后将操作权交给spring中配置的filterChain（过虑链儿）

shiro提供很多filter。

在web.xml中配置filter



## applicationContext-shiro.xml

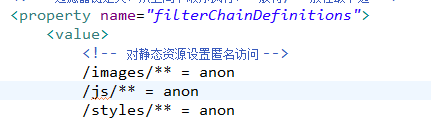
在applicationContext-shiro.xml 中配置web.xml中fitler对应spring容器中的bean。



## 静态资源

对静态资源设置逆名访问：

修改applicationContext-shiro.xml:



## 登陆

### 原理

使用FormAuthenticationFilter过虑器实现 ，原理如下：

将用户没有认证时，请求loginurl进行认证，用户身份和用户密码提交数据到loginurl

FormAuthenticationFilter拦截住取出request中的username和password（两个参数名称是可以配置的）

FormAuthenticationFilter调用realm传入一个token（username和password）

realm认证时根据username查询用户信息（在Activeuser中存储，包括 userid、usercode、username、menus）。

如果查询不到，realm返回null，FormAuthenticationFilter向request域中填充一个参数（记录了异常信息）

### 登陆页面

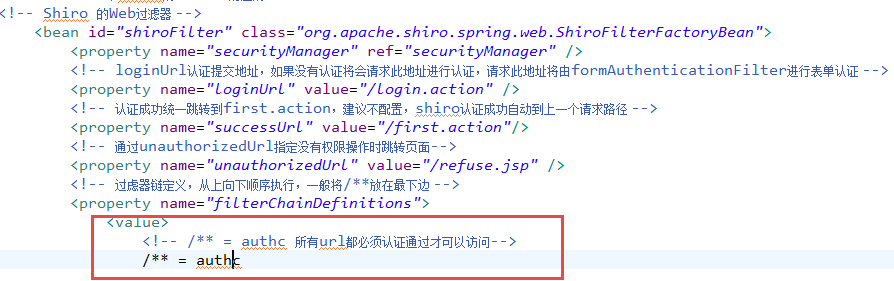
由于FormAuthenticationFilter的用户身份和密码的input的默认值（username和password），修改页面的账号和密码 的input的名称为username和password

### 登陆代码实现



### 认证拦截过虑器

在applicationContext-shiro.xml中配置：

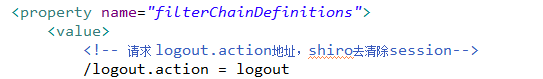


## 退出

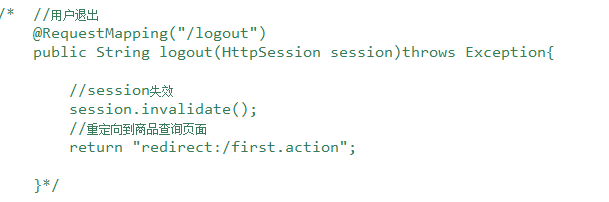
### 使用LogoutFilter

不用我们去实现退出，只要去访问一个退出的url（该 url是可以不存在），由LogoutFilter拦截住，清除session。

在applicationContext-shiro.xml配置LogoutFilter：



可以删除原来的logout的controller方法代码。



## 认证信息在页面显示

1、认证后用户菜单在首页显示

2、认证后用户的信息在页头显示

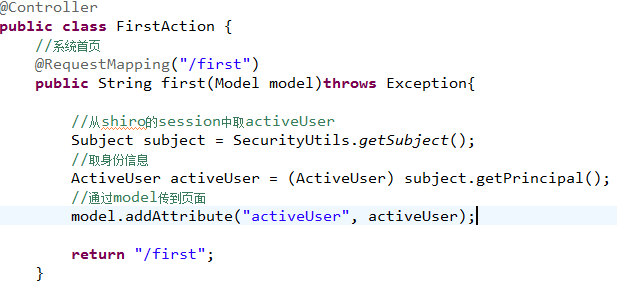
### 修改realm设置完整认证信息

realm从数据库查询用户信息，将用户菜单、usercode、username等设置在SimpleAuthenticationInfo中。

先使用静态代码实现：



### 修改first.action将认证信息在页面显示



## 授权过虑器测试

### 使用PermissionsAuthorizationFilter

在applicationContext-shiro.xml中配置url所对应的权限。

测试流程：

1、在applicationContext-shiro.xml中配置filter规则

<!--商品查询需要商品查询权限 -->

/items/queryItems.action = perms[item:query]

2、用户在认证通过后，请求/items/queryItems.action

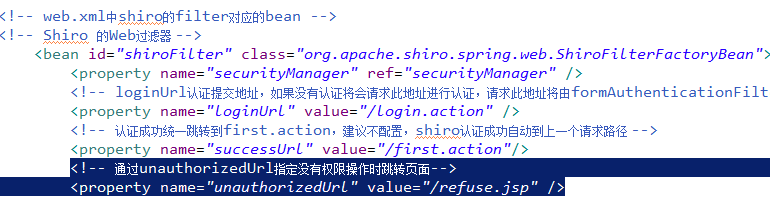
3、被PermissionsAuthorizationFilter拦截，发现需要“item:query”权限

4、PermissionsAuthorizationFilter调用realm中的doGetAuthorizationInfo获取数据库中正确的权限

5、PermissionsAuthorizationFilter对item:query 和从realm中获取权限进行对比，如果“item:query”在realm返回的权限列表中，授权通过。

### 创建refuse.jsp

如果授权失败，跳转到refuse.jsp，需要在spring容器中配置：



### 问题总结

1、在applicationContext-shiro.xml中配置过虑器链接，需要将全部的url和权限对应起来进行配置，比较发麻不方便使用。

2、每次授权都需要调用realm查询数据库，对于系统性能有很大影响，可以通过shiro缓存来解决。

## shiro的过虑器

|  |  |
| --- | --- |
| 过滤器简称 | 对应的java类 |
| anon | org.apache.shiro.web.filter.authc.AnonymousFilter |
| authc | org.apache.shiro.web.filter.authc.FormAuthenticationFilter |
| authcBasic | org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter |
| perms | org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter |
| port | org.apache.shiro.web.filter.authz.PortFilter |
| rest | org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter |
| roles | org.apache.shiro.web.filter.authz.RolesAuthorizationFilter |
| ssl | org.apache.shiro.web.filter.authz.SslFilter |
| user | org.apache.shiro.web.filter.authc.UserFilter |
| logout | org.apache.shiro.web.filter.authc.LogoutFilter |

anon:例子/admins/\*\*=anon 没有参数，表示可以匿名使用。

authc:例如/admins/user/\*\*=authc表示需要认证(登录)才能使用，FormAuthenticationFilter是表单认证，没有参数

perms：例子/admins/user/\*\*=perms[user:add:\*],参数可以写多个，多个时必须加上引号，并且参数之间用逗号分割，例如/admins/user/\*\*=perms["user:add:\*,user:modify:\*"]，当有多个参数时必须每个参数都通过才通过，想当于isPermitedAll()方法。

user:例如/admins/user/\*\*=user没有参数表示必须存在用户, 身份认证通过或通过记住我认证通过的可以访问，当登入操作时不做检查

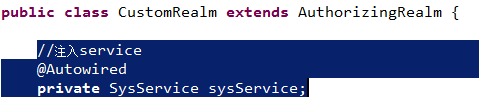
## 认证

### 需求

修改realm的doGetAuthenticationInfo，从数据库查询用户信息，realm返回的用户信息中包括 （md5加密后的串和salt），实现让shiro进行散列串的校验。

### 修改doGetAuthenticationInfo从数据库查询用户信息

1、将SysService注入到realm中。



//realm的认证方法，从数据库查询用户信息

@Override

**protected** AuthenticationInfo doGetAuthenticationInfo(

AuthenticationToken token) **throws** AuthenticationException {

// token是用户输入的用户名和密码

// 第一步从token中取出用户名

String userCode = (String) token.getPrincipal();

// 第二步：根据用户输入的userCode从数据库查询

SysUser sysUser = **null**;

**try** {

sysUser = sysService.findSysUserByUserCode(userCode);

} **catch** (Exception e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

// 如果查询不到返回null

**if**(sysUser==**null**){//

**return** **null**;

}

// 从数据库查询到密码

String password = sysUser.getPassword();

//盐

String salt = sysUser.getSalt();

// 如果查询到返回认证信息AuthenticationInfo

//activeUser就是用户身份信息

ActiveUser activeUser = **new** ActiveUser();

activeUser.setUserid(sysUser.getId());

activeUser.setUsercode(sysUser.getUsercode());

activeUser.setUsername(sysUser.getUsername());

//..

//根据用户id取出菜单

List<SysPermission> menus = **null**;

**try** {

//通过service取出菜单

menus = sysService.findMenuListByUserId(sysUser.getId());

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

//将用户菜单 设置到activeUser

activeUser.setMenus(menus);

//将activeUser设置simpleAuthenticationInfo

SimpleAuthenticationInfo simpleAuthenticationInfo = **new** SimpleAuthenticationInfo(

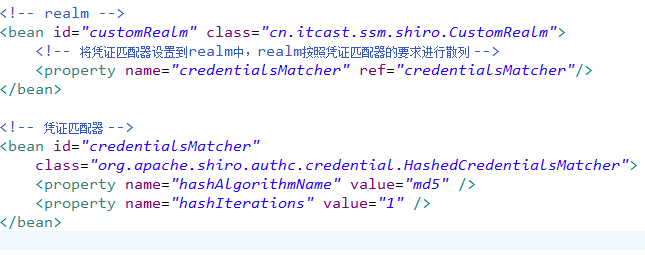
activeUser, password,ByteSource.Util.*bytes*(salt), **this**.getName());

**return** simpleAuthenticationInfo;

}

### 设置凭证匹配器

数据库中存储到的md5的散列值，在realm中需要设置数据库中的散列值它使用散列算法 及散列次数，让shiro进行散列对比时和原始数据库中的散列值使用的算法 一致。



## 授权

### 需求

修改realm的doGetAuthorizationInfo，从数据库查询权限信息。

使用注解式授权方法。

使用jsp标签授权方法。

### 修改doGetAuthorizationInfo从数据库查询权限

// 用于授权

@Override

**protected** AuthorizationInfo doGetAuthorizationInfo(

PrincipalCollection principals) {

//从 principals获取主身份信息

//将getPrimaryPrincipal方法返回值转为真实身份类型（在上边的doGetAuthenticationInfo认证通过填充到SimpleAuthenticationInfo中身份类型），

ActiveUser activeUser = (ActiveUser) principals.getPrimaryPrincipal();

//根据身份信息获取权限信息

//从数据库获取到权限数据

List<SysPermission> permissionList = **null**;

**try** {

permissionList = sysService.findPermissionListByUserId(activeUser.getUserid());

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

//单独定一个集合对象

List<String> permissions = **new** ArrayList<String>();

**if**(permissionList!=**null**){

**for**(SysPermission sysPermission:permissionList){

//将数据库中的权限标签 符放入集合

permissions.add(sysPermission.getPercode());

}

}

//查到权限数据，返回授权信息(要包括 上边的permissions)

SimpleAuthorizationInfo simpleAuthorizationInfo = **new** SimpleAuthorizationInfo();

//将上边查询到授权信息填充到simpleAuthorizationInfo对象中

simpleAuthorizationInfo.addStringPermissions(permissions);

**return** simpleAuthorizationInfo;

}

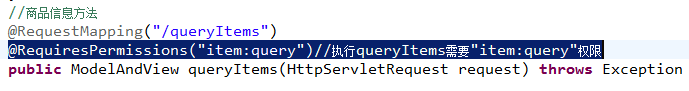
### 开启controller类aop支持

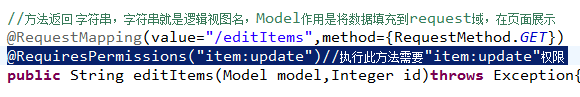
对系统中类的方法给用户授权，建议在controller层进行方法授权。

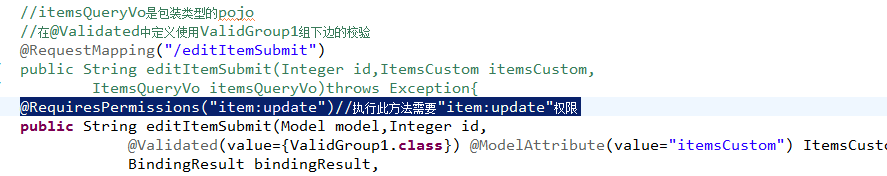
在springmvc.xml中配置：



### 在controller方法中添加注解







### jsp标签 授权

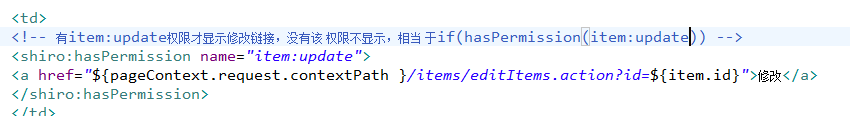
Jsp页面添加：

<%@ tagliburi="http://shiro.apache.org/tags" prefix="shiro" %>

|  |  |
| --- | --- |
| 标签名称 | 标签条件（均是显示标签内容） |
| <shiro:authenticated> | 登录之后 |
| <shiro:notAuthenticated> | 不在登录状态时 |
| <shiro:guest> | 用户在没有RememberMe时 |
| <shiro:user> | 用户在RememberMe时 |
| <shiro:hasAnyRoles name="abc,123" > | 在有abc或者123角色时 |
| <shiro:hasRole name="abc"> | 拥有角色abc |
| <shiro:lacksRole name="abc"> | 没有角色abc |
| <shiro:hasPermission name="abc"> | 拥有权限资源abc |
| <shiro:lacksPermission name="abc"> | 没有abc权限资源 |
| <shiro:principal> | 显示用户身份名称 |

<shiro:principal property="username"/> 显示用户身份中的属性值

修改itemsList.jsp页面：



### 授权测试

当调用controller的一个方法，由于该 方法加了@RequiresPermissions("item:query") ，shiro调用realm获取数据库中的权限信息，看"item:query"是否在权限数据中存在，如果不存在就拒绝访问，如果存在就授权通过。

当展示一个jsp页面时，页面中如果遇到<shiro:hasPermission name="item:update">，shiro调用realm获取数据库中的权限信息，看item:update是否在权限数据中存在，如果不存在就拒绝访问，如果存在就授权通过。

问题：只要遇到注解或jsp标签的授权，都会调用realm方法查询数据库，需要使用缓存解决此问题。

## shiro缓存

针对上边授权频繁查询数据库，需要使用shiro缓存。

### 缓存流程

shiro中提供了对认证信息和授权信息的缓存。shiro默认是关闭认证信息缓存的，对于授权信息的缓存shiro默认开启的。主要研究授权信息缓存，因为授权的数据量大。

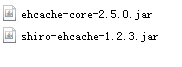
用户认证通过。

该 用户第一次授权：调用realm查询数据库

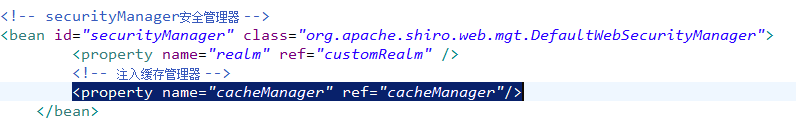
该 用户第二次授权：不调用realm查询数据库，直接从缓存中取出授权信息（权限标识符）。

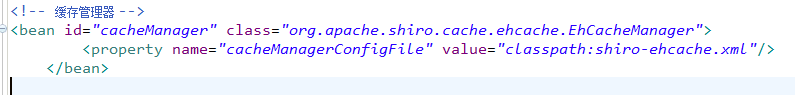
### 使用ehcache

#### 添加Ehcache的jar包

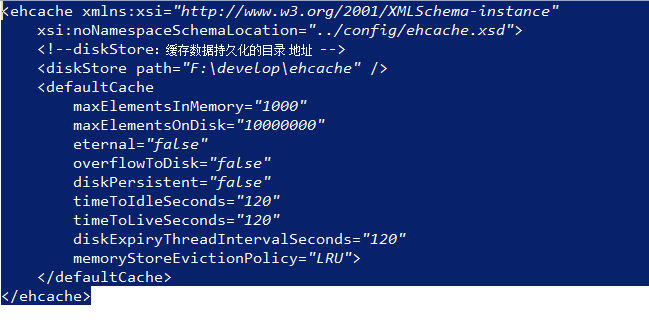


#### 配置cacheManager





#### shiro-ehcache.xml



#### 缓存清空

如果用户正常退出，缓存自动清空。

如果用户非正常退出，缓存自动清空。

如果修改了用户的权限，而用户不退出系统，修改的权限无法立即生效。

需要手动进行编程实现：

在权限修改后调用realm的clearCache方法清除缓存。

下边的代码正常开发时要放在service中调用。

在service中，权限修改后调用realm的方法。

在realm中定义clearCached方法：

//清除缓存

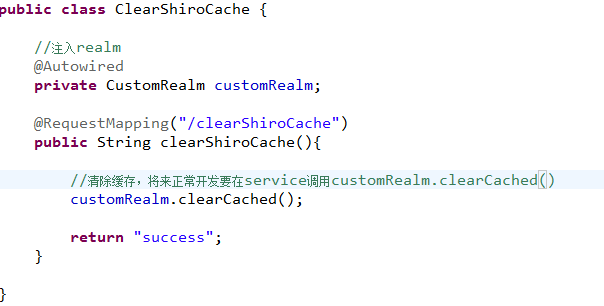
**public** **void** clearCached() {

PrincipalCollection principals = SecurityUtils.*getSubject*().getPrincipals();

**super**.clearCache(principals);

}

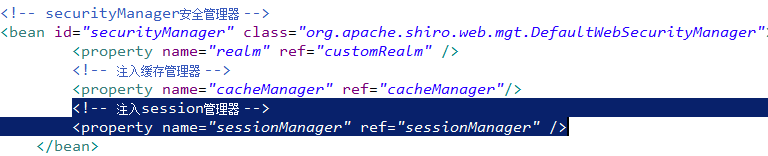
测试清除缓存controller方法：

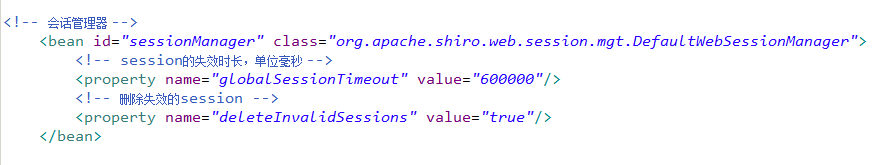


## sessionManager

和shiro整合后，使用shiro的session管理，shiro提供sessionDao操作 会话数据。

配置sessionManager





## 验证码

### 思路

shiro使用FormAuthenticationFilter进行表单认证，验证校验的功能应该加在FormAuthenticationFilter中，在认证之前进行验证码校验。

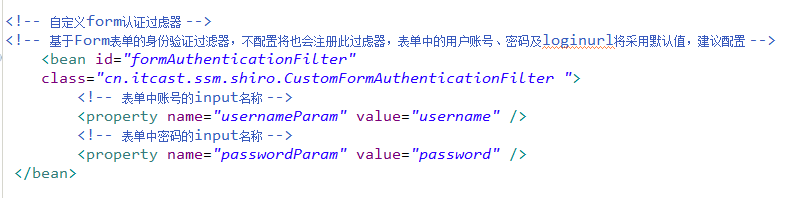
需要写FormAuthenticationFilter的子类，继承FormAuthenticationFilter，改写它的认证方法，在认证之前进行验证码校验。

### 自定义FormAuthenticationFilter



### 配置自定义FormAuthenticationFilter





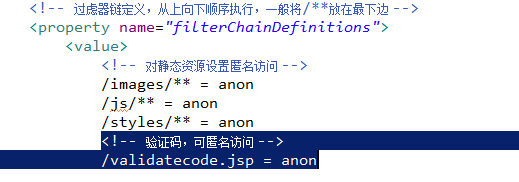
### 在login.action对验证错误 进行解析



### 在登陆页面添加验证码



### 在filter配置匿名访问验证码jsp



## 记住我

用户登陆选择“自动登陆”本次登陆成功会向cookie写身份信息，下次登陆从cookie中取出身份信息实现自动登陆。

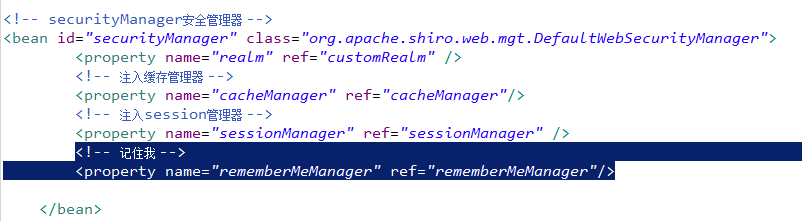
### 用户身份实现java.io.Serializable接口

向cookie记录身份信息需要用户身份信息对象实现序列化接口，如下：



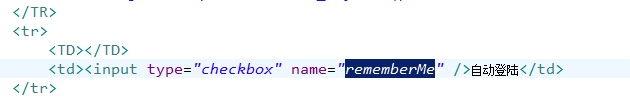


### 配置rememeberMeManager

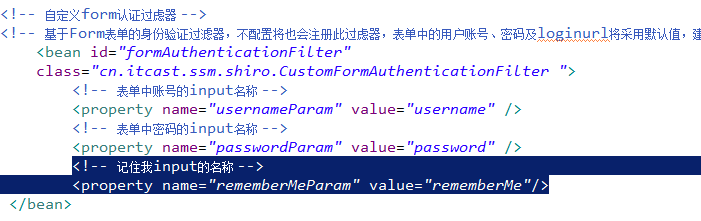




### 登陆页面



### 配置rememberMe的input名称



### 测试

自动登陆后，需要查看 cookei是否有rememberMe



### 使用UserFilter

如果设置记住我，下次访问某些url时可以不用登陆。将记住我即可访问的地址配置让UserFilter拦截。

