### 一、Spring Security相关概念

**1.1.、Spring Security介绍：**

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（简单说是对访问权限进行控制嘛）。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。

**1.2、Spring Security实现原理：**

Spring Security对Web安全性的支持大量地依赖于Servlet过滤器。通过这些过滤器拦截进入请求，判断是否已经登录认证且具访问对应请求的权限。

要完成访问控制，Spring Security至少需要下面四个拦截器（调度器、认证管理器、权限资源关联器、访问决策器）进行配合完成：

<!-- mySecurityInterceptor这里我们把它命名为调度器吧 -->

<!-- 必须包含 authenticationManager,securityMetadataSource,accessDecisionManager 三个属性 -->

<!-- 我们的所有控制将在这三个类中实现 -->

<!-- 它继承AbstractSecurityInterceptor类并实现了Filter接口 -->

<bean id="mySecurityInterceptor" class="com.luo.Filter.MySecurityInterceptor">

<b:property name="authenticationManager" ref="authenticationManager" />

<b:property name="securityMetadataSource" ref="securityMetadataSource" />

<b:property name="accessDecisionManager" ref="accessDecisionManager" />

</bean>

<!-- 认证管理器，实现用户认证的入口 -->

<authentication-manager alias="authenticationManager">

<authentication-provider user-service-ref="myUserDetailService" />

</authentication-manager>

<!-- 在这个类中，你就可以从数据库中读入用户的密码，角色信息等 -->

<!-- 主要实现UserDetailsService接口即可，然后返回用户数据 -->

<bean id="myUserDetailService" class="com.luo.Filter.MyUserDetailService" />

<!-- 权限资源关联器，将所有的资源和权限对应关系建立起来，即定义某一资源可以被哪些角色访问 -->

<!-- 它实现了FilterInvocationSecurityMetadataSource接口 -->

<bean id="securityMetadataSource" class="com.luo.Filter.MyFilterInvocationSecurityMetadataSource" />

<!--访问决策器，决定某个用户具有的角色，是否有足够的权限去访问某个资源 -->

<!-- 它实现了AccessDecisionManager接口 -->

<bean id="accessDecisionManager" class="com.luo.Filter.MyAccessDecisionManager">

看完上面的配置，可能未必能够完全明白，下面我们再进一步说明。

（1）首先我们自定义一个过滤器（调度器，这里我们命名为mySecurityInterceptor），这个过滤器继承AbstractSecurityInterceptor类（这里先说明，本文但凡不是自定义的类或接口都是Spring Security提供的，无须深究）。 它至少包含 authenticationManager,accessDecisionManager,securityMetadataSource三个属性，我们的所有控制将在这三个类中实现。

（2）登录验证：自定义类MyUserDetailService实现UserDetailsService接口和其loadUserByUsername方法，这个方法根据用户输入的用户名，从数据库里面获取该用户的所有权限细信息（统称用户信息）。Spring Security的AuthenticationProcessingFilter拦截器调用authenticationManager，类MyUserDetailService拿到用户信息后，authenticationManager对比用户的密码（即验证用户），如果通过了，那么相当于通过了AuthenticationProcessingFilter拦截器，也就是登录验证通过。

（3）资源访问控制：MySecurityInterceptor继承AbstractSecurityInterceptor、实现Filter是必须的。登陆后，每次访问资源都会被MySecurityInterceptor这个拦截器拦截，它首先会调用MyFilterInvocationSecurityMetadataSource类的getAttributes方法获取被拦截url所需的权限，在调用MyAccessDecisionManager类decide方法判断用户是否够权限。

可能文字描述还是比较抽象，通过实例应该能让大家更加清楚其原理。

补充说明一下：

UserDetailsService在身份认证中的作用：

Spring Security中进行身份验证的是AuthenticationManager接口，ProviderManager是它的一个默认实现，但它并不用来处理身份认证，而是委托给配置好的AuthenticationProvider，每个AuthenticationProvider会轮流检查身份认证。检查后或者返回Authentication对象或者抛出异常。

验证身份就是加载响应的UserDetails，看看是否和用户输入的账号、密码、权限等信息匹配。此步骤由实现AuthenticationProvider的DaoAuthenticationProvider（它利用UserDetailsService验证用户名、密码和授权）处理。

因此，登录认证其实可以不实现UserDetailsService，而是实现AuthenticationProvider，然后在AuthenticationProvider里面获取用户输入的用户名和密码进行校验也是可以的。或者两者一起使用。

下面推荐两者一起使用的方式<http://blog.sina.com.cn/s/blog_4adc4b090102uy2f.html>

另外，只实现AuthenticationProvider而不实现UserDetailsService的方式，这类是重写AuthenticationProvider的authenticate方法的代码：

@Override

public Authentication authenticate(Authentication authentication) throws AuthenticationException {

String inputLoginId = authentication.getName(); //获取用户输入的用户名

String inputPasswd = authentication.getCredentials().toString(); /获取用户输入的密码

LOGGER.info("用户{}登录", inputLoginId);

try{

// 查询此用户信息

myUser myUser = null; //根据用户名到数据库里面查询用户数据

if (myUser == null ) {

throw new Exception("您输入的账号不存在");

}

if (myUser.getUserStatus() == UserStatus.locked) {

throw new Exception("您的账号已被锁定");

}

String encodedPassword = myUser.getLoginPasswd();

// 校验密码是否正确

boolean authenticated = verifyPassword(inputPasswd, encodedPassword);

if (authenticated) {

// 认证成功处理

updateLoginInfo(myUser.getLoginId(), 0, null);

} else {

// 认证失败处理

authenticateErrorProcess(portalUser);

}

List<GrantedAuthority> grantedAuths = new ArrayList<GrantedAuthority>();

for (MyRole myRole : myUser.allRoleList()) {

grantedAuths.add(new SimpleGrantedAuthority(myRole.getRoleCode()));

}

MyAuthUser authUser = new PortalAuthUser(inputLoginId, inputPasswd, true, true, true, true, grantedAuths);

authUser.setPortalUser(portalUser);

return new UsernamePasswordAuthenticationToken(authUser, null, authUser.getAuthorities());

}catch(Exception e){

LOGGER.warn("用户登录失败", e);

throw new Exception(" 请确认用户名或者密码是否正确);

}

}

### 二、Spring Security实例详细说明

本实例环境：eclipse + maven   
本实例采用的主要技术：spring + springmvc + spring security

时间有限这里只对其访问控制原理进行了阐述，例子后面再补上，不过关于实例推荐参考博文：<http://blog.csdn.net/u012367513/article/details/38866465>，这篇文章写得非常详细！！！