**使用 Spring Security 保护 Web 应用的安全**

在 Web 应用开发中，安全一直是非常重要的一个方面。安全虽然属于应用的非功能性需求，但是应该在应用开发的初期就考虑进来。如果在应用开发的后期才考虑安全的问题，就可能陷入一个两难的境地：一方面，应用存在严重的安全漏洞，无法满足用户的要求，并可能造成用户的隐私数据被攻击者窃取；另一方面，应用的基本架构已经确定，要修复安全漏洞，可能需要对系统的架构做出比较重大的调整，因而需要更多的开发时间，影响应用的发布进程。因此，从应用开发的第一天就应该把安全相关的因素考虑进来，并在整个应用的开发过程中。

本文详细介绍了如何使用 Spring Security 来保护 Web 应用的安全。Spring Security 本身以及 Spring 框架带来的灵活性，能够满足一般 Web 应用开发的典型需求，并允许开发人员进行定制。下面首先简单介绍 Spring Security。

## Spring Security 简介

Spring 是一个非常流行和成功的 Java 应用开发框架。Spring Security 基于 Spring 框架，提供了一套 Web 应用安全性的完整解决方案。一般来说，Web 应用的安全性包括用户认证（Authentication）和用户授权（Authorization）两个部分。用户认证指的是验证某个用户是否为系统中的合法主体，也就是说用户能否访问该系统。用户认证一般要求用户提供用户名和密码。系统通过校验用户名和密码来完成认证过程。用户授权指的是验证某个用户是否有权限执行某个操作。在一个系统中，不同用户所具有的权限是不同的。比如对一个文件来说，有的用户只能进行读取，而有的用户可以进行修改。一般来说，系统会为不同的用户分配不同的角色，而每个角色则对应一系列的权限。

对于上面提到的两种应用情景，Spring Security 框架都有很好的支持。在用户认证方面，Spring Security 框架支持主流的认证方式，包括 HTTP 基本认证、HTTP 表单验证、HTTP 摘要认证、OpenID 和 LDAP 等。在用户授权方面，Spring Security 提供了基于角色的访问控制和访问控制列表（Access Control List，ACL），可以对应用中的领域对象进行细粒度的控制。

本文将通过三个具体的示例来介绍 Spring Security 的使用。第一个示例是一个简单的企业员工管理系统。该系统中存在三类用户，分别是普通员工、经理和总裁。不同类别的用户所能访问的资源不同。对这些资源所能执行的操作也不相同。Spring Security 能帮助开发人员以简单的方式满足这些安全性相关的需求。第二个示例展示了如何与 LDAP 服务器进行集成。第三个示例展示了如何与 OAuth 进行集成。完整的示例代码见[参考资料](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#resources)。下面首先介绍基本的用户认证和授权的实现。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#ibm-pcon)

## 基本用户认证和授权

本节从最基本的用户认证和授权开始对 Spring Security 进行介绍。一般来说，Web 应用都需要保存自己系统中的用户信息。这些信息一般保存在数据库中。用户可以注册自己的账号，或是由系统管理员统一进行分配。这些用户一般都有自己的角色，如普通用户和管理员之类的。某些页面只有特定角色的用户可以访问，比如只有管理员才可以访问 /admin 这样的网址。下面介绍如何使用 Spring Security 来满足这样基本的认证和授权的需求。

首先需要把 Spring Security 引入到 Web 应用中来，这是通过在 web.xml添加一个新的过滤器来实现的，如 [代码清单 1](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#code1) 所示。

##### 清单 1. 在 web.xml 中添加 Spring Security 的过滤器

<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>

Spring Security 使用的是 Servlet 规范中标准的过滤器机制。对于特定的请求，Spring Security 的过滤器会检查该请求是否通过认证，以及当前用户是否有足够的权限来访问此资源。对于非法的请求，过滤器会跳转到指定页面让用户进行认证，或是返回出错信息。需要注意的是，[代码清单 1](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#code1) 中虽然只定义了一个过滤器，Spring Security 实际上是使用多个过滤器形成的链条来工作的。

下一步是配置 Spring Security 来声明系统中的合法用户及其对应的权限。用户相关的信息是通过org.springframework.security.core.userdetails.UserDetailsService 接口来加载的。该接口的唯一方法是loadUserByUsername(String username)，用来根据用户名加载相关的信息。这个方法的返回值是org.springframework.security.core.userdetails.UserDetails 接口，其中包含了用户的信息，包括用户名、密码、权限、是否启用、是否被锁定、是否过期等。其中最重要的是用户权限，由 org.springframework.security.core.GrantedAuthority 接口来表示。虽然 Spring Security 内部的设计和实现比较复杂，但是一般情况下，开发人员只需要使用它默认提供的实现就可以满足绝大多数情况下的需求，而且只需要简单的配置声明即可。

在第一个示例应用中，使用的是数据库的方式来存储用户的信息。Spring Security 提供了org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl 类来支持从数据库中加载用户信息。开发人员只需要使用与该类兼容的数据库表结构，就可以不需要任何改动，而直接使用该类。[代码清单 2](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#code2) 中给出了相关的配置。

##### 清单 2. 声明使用数据库来保存用户信息

<bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="org.apache.derby.jdbc.ClientDriver" />

<property name="url" value="jdbc:derby://localhost:1527/mycompany" />

<property name="username" value="app" />

<property name="password" value="admin" />

</bean>

<bean id="userDetailsService"

class="org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl">

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

</bean>

<sec:authentication-manager>

<sec:authentication-provider user-service-ref="userDetailsService" />

</sec:authentication-manager>

如 [代码清单 2](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#code2) 所示，首先定义了一个使用 Apache Derby 数据库的数据源，Spring Security 的org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl 类使用该数据源来加载用户信息。最后需要配置认证管理器使用该 UserDetailsService。

接着就可以配置用户对不同资源的访问权限了。这里的资源指的是 URL 地址。配置的内容如 [代码清单 3](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#code3) 所示。sec 是 Spring Security 的配置元素所在的名称空间的前缀。

##### 清单 3. 配置对不同 URL 模式的访问权限

<sec:http>

<sec:intercept-url pattern="/president\_portal.do\*\*" access="ROLE\_PRESIDENT" />

<sec:intercept-url pattern="/manager\_portal.do\*\*" access="ROLE\_MANAGER" />

<sec:intercept-url pattern="/\*\*" access="ROLE\_USER" />

<sec:form-login />

<sec:logout />

</sec:http>

第一个示例应用中一共定义了三种角色：普通用户、经理和总裁，分别用 ROLE\_USER、ROLE\_MANAGER 和 ROLE\_PRESIDENT 来表示。[代码清单 3](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#code3) 中定义了访问不同的 URL 模式的用户所需要的角色。这是通过 <sec:intercept-url> 元素来实现的，其属性 pattern 声明了请求 URL 的模式，而属性 access 则声明了访问此 URL 时所需要的权限。需要按照 URL 模式从精确到模糊的顺序来进行声明。因为 Spring Security 是按照声明的顺序逐个进行比对的，只要用户当前访问的 URL 符合某个 URL 模式声明的权限要求，该请求就会被允许。如果把 [代码清单 3](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/" \l "code3) 中本来在最后的 URL 模式 /\*\* 声明放在最前面，那么当普通用户访问 /manager\_portal.do 的时候，该请求也会被允许。这显然是不对的。通过 <sec:form-login> 元素声明了使用 HTTP 表单验证。也就是说，当未认证的用户试图访问某个受限 URL 的时候，浏览器会跳转到一个登录页面，要求用户输入用户名和密码。<sec:logout> 元素声明了提供用户注销登录的功能。默认的注销登录的 URL 是/j\_spring\_security\_logout，可以通过属性 logout-url 来修改。

当完成这些配置并运行应用之后，会发现 Spring Security 已经默认提供了一个登录页面的实现，可以直接使用。开发人员也可以对登录页面进行定制。通过 <form-login> 的属性 login-page、login-processing-url 和 authentication-failure-url就可以定制登录页面的 URL、登录请求的处理 URL 和登录出现错误时的 URL 等。从这里可以看出，一方面 Spring Security 对开发中经常会用到的功能提供了很好的默认实现，另外一方面也提供了非常灵活的定制能力，允许开发人员提供自己的实现。

在介绍如何用 Spring Security 实现基本的用户认证和授权之后，下面介绍其中的核心对象。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#ibm-pcon)

## SecurityContext 和 Authentication 对象

下面开始讨论几个 Spring Security 里面的核心对象。org.springframework.security.core.context.SecurityContext接口表示的是当前应用的安全上下文。通过此接口可以获取和设置当前的认证对象。org.springframework.security.core.Authentication接口用来表示此认证对象。通过认证对象的方法可以判断当前用户是否已经通过认证，以及获取当前认证用户的相关信息，包括用户名、密码和权限等。要使用此认证对象，首先需要获取到 SecurityContext 对象。通过org.springframework.security.core.context.SecurityContextHolder 类提供的静态方法 getContext() 就可以获取。再通过 SecurityContext对象的 getAuthentication()就可以得到认证对象。通过认证对象的 getPrincipal() 方法就可以获得当前的认证主体，通常是 UserDetails 接口的实现。联系到上一节介绍的 UserDetailsService，典型的认证过程就是当用户输入了用户名和密码之后，UserDetailsService通过用户名找到对应的 UserDetails 对象，接着比较密码是否匹配。如果不匹配，则返回出错信息；如果匹配的话，说明用户认证成功，就创建一个实现了 Authentication接口的对象，如 org.springframework.security. authentication.UsernamePasswordAuthenticationToken 类的对象。再通过 SecurityContext的 setAuthentication() 方法来设置此认证对象。

[代码清单 4](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#code4) 给出了使用 SecurityContext 和 Authentication的一个示例，用来获取当前认证用户的用户名。

##### 清单 4. 获取当前认证用户的用户名

public static String getAuthenticatedUsername() {

String username = null;

Object principal = SecurityContextHolder.getContext()

.getAuthentication().getPrincipal();

if (principal instanceof UserDetails) {

username = ((UserDetails) principal).getUsername();

} else {

username = principal.toString();

}

return username;

}

默认情况下，SecurityContextHolder使用 ThreadLocal来保存 SecurityContext对象。因此，SecurityContext对象对于当前线程上所有方法都是可见的。这种实现对于 Web 应用来说是合适的。不过在有些情况下，如桌面应用，这种实现方式就不适用了。Spring Security 允许开发人员对此进行定制。开发人员只需要实现接口org.springframework.security.core.context.SecurityContextHolderStrategy并通过 SecurityContextHolder的setStrategyName(String)方法让 Spring Security 使用此实现即可。另外一种设置方式是使用系统属性。除此之外，Spring Security 默认提供了另外两种实现方式：MODE\_GLOBAL表示当前应用共享唯一的 SecurityContextHolder；MODE\_INHERITABLETHREADLOCAL表示子线程继承父线程的 SecurityContextHolder。[代码清单 5](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/" \l "code5)给出了使用全局唯一的 SecurityContextHolder的示例。

##### 清单 5. 使用全局唯一的 SecurityContextHolder

public void useGlobalSecurityContextHolder() {

SecurityContextHolder.setStrategyName(SecurityContextHolder.MODE\_GLOBAL);

}

在介绍完 Spring Security 中的 SecurityContext和 Authentication之后，下面介绍如何保护服务层的方法。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#ibm-pcon)

## 服务层方法保护

之前章节中介绍的是在 URL 这个粒度上的安全保护。这种粒度的保护在很多情况下是不够的。比如相同的 URL 对应的页面上，不同角色的用户所能看到的内容和执行的操作是有可能不同的。在第一个示例应用中，系统中记录了每个员工的工资收入。所有员工都可以查看自己的工资，但是只有员工的直接经理才可以修改员工的工资。这就涉及到对应用中服务层的方法进行相应的权限控制，从而避免安全漏洞。

保护服务层方法涉及到对应用中的方法调用进行拦截。通过 Spring 框架提供的良好面向方面编程（AOP）的支持，可以很容易的对方法调用进行拦截。Spring Security 利用了 AOP 的能力，允许以声明的方式来定义调用方式时所需的权限。[代码清单 6](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#code6)中给出了对方法调用进行保护的配置文件示例。

##### 清单 6. 对方法调用进行保护

<bean id="userSalarySecurity"

class="org.springframework.security.access.intercept.aspectj.

AspectJMethodSecurityInterceptor">

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

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

<property name="securityMetadataSource">

<value>

mycompany.service.UserService.raiseSalary=ROLE\_MANAGER

</value>

</property>

</bean>

如 [代码清单 6](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#code6)所示，通过 mycompany.service.UserService.raiseSalary=ROLE\_MANAGER声明了mycompany.service.UserService类的 raiseSalary方法只有具有角色 ROLE\_MANAGER的用户才能执行。这就使得只具有角色ROLE\_USER的用户无法调用此方法。

不过仅对方法名称进行权限控制并不能解决另外的一些问题。比如在第一个示例应用中的增加工资的实现是通过发送 HTTP POST 请求到salary.do这个 URL 来完成的。salary.do对应的控制器 mycompany.controller.SalaryController会调用mycompany.service.UserService类的 raiseSalary方法来完成增加工资的操作。存在的一种安全漏洞是具有 ROLE\_MANAGER角色的用户可以通过其它工具（如 cURL 或 Firefox 扩展 Poster 等）来创建 HTTP POST 请求来更改其它员工的工资。为了解决这个问题，需要对raiseSalary的调用进行更加细粒度的控制。通过 Spring Security 提供的 AspectJ 支持就可以编写相关的控制逻辑，如 [代码清单 7](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#code7)所示。

##### 清单 7. 使用 AspectJ 进行细粒度的控制

public aspect SalaryManagementAspect {

private AspectJMethodSecurityInterceptor securityInterceptor;

private UserDao userDao;

pointcut salaryChange(): target(UserService)

&& execution(public void raiseSalary(..)) &&!within(SalaryManagementAspect);

Object around(): salaryChange() {

if (this.securityInterceptor == null) {

return proceed();

}

AspectJCallback callback = new AspectJCallback() {

public Object proceedWithObject() {

return proceed();

}

};

Object[] args = thisJoinPoint.getArgs();

String employee = (String) args[0]; // 要修改的员工的用户名

User user = userDao.getByUsername(employee);

String currentUser = UsernameHolder.getAuthenticatedUsername(); // 当前登录用户

if (!currentUser.equals(user.getManagerId())) {

throw new AccessDeniedException

("Only the direct manager can change the salary.");

}

return this.securityInterceptor.invoke(thisJoinPoint, callback);

}

}

如 [代码清单 7](http://www.ibm.com/developerworks/cn/java/j-lo-springsecurity/#code7)所示，定义了一个切入点（pointcut）salaryChange和对应的环绕增强。当方法 raiseSalary被调用的时候，会比较要修改的员工的经理的用户名和当前登录用户的用户名是否一致。当不一致的时候就会抛出 AccessDeniedException异常。

在介绍了如何保护方法调用之后，下面介绍如何通过访问控制列表来保护领域对象。