downpour
等级:

文章: 1153
积分: 2078
来自: 上海

我现在离线

发表时间：2009-01-22 最后修改：2009-01-22

论坛上看了不少Spring Security的相关文章。这些文章基本上都还是基于Acegi-1.X的配置方式，而主要的配置示例也来自于SpringSide的贡献。

众所周知，Spring Security针对Acegi的一个重大的改进就在于其配置方式大大简化了。所以如果配置还是基于Acegi-1.X这样比较繁琐的配置方式的话，那么我们还不如直接使用Acegi而不要去升级了。所以在这里，我将结合一个示例，重点讨论一下Spring Security 2是如何进行配置简化的。

相关文章:
- 帮Spring security一个忙：为Struts2 写个Plugin对Action进行权限控制
- 浅谈Acegi配置
- Spring Security优劣之我见

推荐圈子: 权限管理
更多相关推荐

搭建基础环境

首先我们为示例搭建基本的开发环境，环境的搭建方式，可以参考我的另外一篇文章：http://www.javaeye.com/wiki/struts2/1321-struts2-development-environment-to-build

整个环境的搭建包括：创建合适的目录结构、加入了合适的Library，加入了基本的Jetty启动类、加入基本的配置文件等。最终的项目结构，可以参考我的附件。

参考文档

这里主要的参考文档是Spring Security的自带的Reference。网络上有一个它的中文翻译，地址如下：http://www.family168.com/tutorial/springsecurity/html/springsecurity.html

除此之外，springside有一个比较完整的例子，不过是基于Acegi的，我也参阅了其中的一些实现。

Spring Security基本配置

Spring Security是基于Spring的的权限认证框架，对于Spring和Acegi已经比较熟悉的同学对于之前的配置方式应该已经非常了解。接下来的例子，将向大家展示Spring Security基于schema的配置方式。

最小化配置

1. 在**web.xml**文件中加入**Filter**声明

Xml代码

```xml
<!-- Spring security Filter -->
<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>
```

这个Filter会拦截所有的URL请求，并且对这些URL请求进行Spring Security的验证。

注意，springSecurityFilterChain这个名称是由命名空间默认创建的用于处理web安全的一个内部的bean的id。所以你在你的Spring配置文件中，不应该再使用这个id作为你的bean。

与Acegi的配置不同，Acegi需要自行声明一个Spring的bean来作为Filter的实现，而使用Spring Security后，无需再额外定义bean，而是使用<http>元素进行配置。

2. 使用最小的**<http>**配置

Xml代码

```
<http auto-config='true'>
    <intercept-url pattern="/**" access="ROLE_USER" />
</http>
```

这段配置表示：我们要保护应用程序中的所有URL，只有拥有ROLE_USER角色的用户才能访问。你可以使用多个<intercept-url>元素为不同URL的集合定义不同的访问需求，它们会被归入一个有序队列中，每次取出最先匹配的一个元素使用。 所以你必须把期望使用的匹配条件放到最上边。

3. 配置**UserDetailsService**来指定用户和权限

接下来，我们来配置一个UserDetailsService来指定用户和权限：

Xml代码

```
<authentication-provider>
    <user-service>
      <user name="downpour" password="downpour" authorities="ROLE_USER, ROLE_ADMIN" />
      <user name="robbin" password="robbin" authorities="ROLE_USER" />
      <user name="QuakeWang" password="QuakeWang" authorities="ROLE_ADMIN" />
    </user-service>
  </authentication-provider>
```

在这里，downpour拥有ROLE_USER和ROLE_ADMIN的权限，robbin拥有ROLE_USER权限，QuakeWang拥有ROLE_ADMIN的权限

4. 小结

有了以上的配置，你已经可以跑简单的Spring Security的应用了。只不过在这里，我们还缺乏很多基本的元素，所以我们尚不能对上面的代码进行完整性测试。

如果你具备Acegi的知识，你会发现，有很多Acegi中的元素，在Spring Security中都没有了，这些元素包括：表单和基本登录选项、密码编码器、Remember-Me认证等等。

接下来，我们就来详细剖析一下Spring Security中的这些基本元素。

剖析基本配置元素

1. 有关**auto-config**属性

在上面用到的auto-config属性，其实是下面这些配置的缩写：

Xml代码

```
<http>
    <intercept-url pattern="/**" access="ROLE_USER" />
    <form-login />
    <anonymous />
    <http-basic />
    <logout />
    <remember-me />
</http>
```

这些元素分别与登录认证，匿名认证，基本认证，注销处理和remember-me对应。 他们拥有各自的属性，可以改变他们的具体行为。

这样，我们在Acegi中所熟悉的元素又浮现在我们的面前。只是在这里，我们使用的是命名空间而已。

2. 与**Acegi**的比较

我们仔细观察一下没有auto-config的那段XML配置，是不是熟悉多了？让我们来将基于命名空间的配置与传统的Acegi的bean的配置做一个比较，我们会发现以下的区别：

1) 基于命名空间的配置更加简洁，可维护性更强

例如，基于命名空间进行登录认证的配置代码，可能像这样：

Xml代码

```
<form-login login-page="/login.jsp" authentication-failure-url="/login.jsp?error=true" default-target-url=
"/work" />
```

如果使用老的Acegi的Bean的定义方式，可能像这样：

Xml代码

```
<bean id="authenticationProcessingFilter"
                    class="org.acegisecurity.ui.webapp.AuthenticationProcessingFilter">
        <property name="authenticationManager"
                                    ref="authenticationManager"/>
        <property name="authenticationFailureUrl"
                                    value="/login.jsp?error=1"/>
        <property name="defaultTargetUrl" value="/work"/>
        <property name="filterProcessesUrl"
                                    value="/j_acegi_security_check"/>
        <property name="rememberMeServices" ref="rememberMeServices"/>
</bean>
```

这样的例子很多，有兴趣的读者可以一一进行比较。

2) 基于命名空间的配置，我们无需再担心由于过滤器链的顺序而导致的错误

以前，Acegi在缺乏默认内置配置的情况下，你需要自己来定义所有的bean，并指定这些bean在过滤器链中的顺序。一旦顺序错了，很容易发生错误。而现在，过滤器链的顺序被默认指定，你不需要在担心由于顺序的错误而导致的错误。

3. 过滤器链在哪里

到目前为止，我们都还没有讨论过整个Spring Security的核心部分：过滤器链。在原本Acegi的配置中，我们大概是这样配置我们的过滤器链的：

Xml代码

```
<bean id="filterChainProxy"
                    class="org.acegisecurity.util.FilterChainProxy">
        <property name="filterInvocationDefinitionSource">
                <value>
                                    CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
                                    PATTERN_TYPE_APACHE_ANT
                                    /common/**=#NONE#
                                    /css/**=#NONE#
                                    /images/**=#NONE#
                                    /js/**=#NONE#
                                    /login.jsp=#NONE#
                                    /**=httpSessionContextIntegrationFilter,logoutFilter,authenticationProcess
ingFilter,securityContextHolderAwareRequestFilter,exceptionTranslationFilter,filterSecurityInterceptor
                </value>
        </property>
</bean>
```

其中，每个过滤器链都将对应于Spring配置文件中的bean的id。

现在，在Spring Security中，我们将看不到这些配置，这些配置都被内置在<http>节点中。让我们来看看这些默认的，已经被内置的过滤器：

Table 2.1. 标准过滤器假名和顺序

假名	过滤器	命名空间元素或属性
CHANNEL_FILTER	ChannelProcessingFilter	http/intercept-url
CONCURRENT_SESSION_FILTER	ConcurrentSessionFilter	http/concurrent-session-control
SESSION_CONTEXT_INTEGRATION_FILTER	HttpSessionContextIntegrationFilter	http
LOGOUT_FILTER	LogoutFilter	http/logout
X509_FILTER	X509PreAuthenticatedProcessigFilter	http/x509
PRE_AUTH_FILTER	AstractPreAuthenticatedProcessingFilter Subclasses	N/A
CAS_PROCESSING_FILTER	CasProcessingFilter	N/A
AUTHENTICATION_PROCESSING_FILTER	AuthenticationProcessingFilter	http/form-login
BASIC_PROCESSING_FILTER	BasicProcessingFilter	http/http-basic
SERVLET_API_SUPPORT_FILTER	SecurityContextHolderAwareRequestFilter	http/@servlet-api-provision
REMEMBER_ME_FILTER	RememberMeProcessingFilter	http/remember-me
ANONYMOUS_FILTER	AnonymousProcessingFilter	http/anonymous
EXCEPTION_TRANSLATION_FILTER	ExceptionTranslationFilter	http
NTLM_FILTER	NtlmProcessingFilter	N/A
FILTER_SECURITY_INTERCEPTOR	FilterSecurityInterceptor	http
SWITCH_USER_FILTER	SwitchUserProcessingFilter	N/A

这些过滤器已经被Spring容器默认内置注册，这也就是我们不再需要在配置文件中定义那么多bean的原因。

同时，过滤器顺序在使用命名空间的时候是被严格执行的。它们在初始化的时候就预先被排好序。不仅如此，Spring Security规定，你不能替换那些**<http>**元素自己使用而创建出的过滤器，比如**HttpSessionContextIntegrationFilter, ExceptionTranslationFilter** 或 **FilterSecurityInterceptor**。

当然，这样的规定是否合理，有待进一步讨论。因为实际上在很多时候，我们希望覆盖过滤器链中的某个过滤器的默认行为。而Spring Security的这种规定在一定程度上限制了我们的行为。

不过Spring Security允许你把你自己的过滤器添加到队列中，使用custom-filter元素，并且指定你的过滤器应该出现的位置：

Xml代码

```
<beans:bean id="myFilter" class="com.mycompany.MySpecialAuthenticationFilter">
    <custom-filter position="AUTHENTICATION_PROCESSING_FILTER"/>
</beans:bean>
```

不仅如此，你还可以使用after或before属性，如果你想把你的过滤器添加到队列中另一个过滤器的前面或后面。 可以分别在position属性使用"FIRST"或"LAST"来指定你想让你的过滤器出现在队列元素的前面或后面。

这个特性或许能够在一定程度上弥补Spring Security的死板规定，而在之后的应用中，我也会把它作为切入点，对资源进行管理。

另外，我需要补充一点的是，对于在**http/intercept-url**中没有进行定义的**URL**，将会默认使用系统内置的过滤器链进行权限认证。所以，你并不需要在http/intercept-url中额外定义一个类似/**的匹配规则。

使用数据库对用户和权限进行管理

一般来说，我们都有使用数据库对用户和权限进行管理的需求，而不会把用户写死在配置文件里。所以，我们接下来就重点讨论使用数据库对用户和权限进行管理的方法。

用户和权限的关系设计

在此之前，我们首先需要讨论一下用户（User）和权限（Role）之间的关系。Spring Security在默认情况下，把这两者当作一对多的关系进行处理。所以，在Spring Security中对这两个对象所采用的表结构关系大概像这样：

Java代码

```
CREATE TABLE users (
  username VARCHAR(50) NOT NULL PRIMARY KEY,
```

```
  password VARCHAR(50) NOT NULL,
  enabled BIT NOT NULL
);

CREATE TABLE authorities (
  username VARCHAR(50) NOT NULL,
  authority VARCHAR(50) NOT NULL
);
```

不过这种设计方式在实际生产环境中基本上不会采用。一般来说，我们会使用逻辑主键ID来标示每个User和每个Authorities（Role）。而且从典型意义上讲，他们之间是一个多对多的关系，我们会采用3张表来表示，下面是我在MySQL中建立的3张表的schema示例：

Java代码

```
CREATE TABLE `user` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(255) default NULL,
  `password` varchar(255) default NULL,
  `disabled` int(1) NOT NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `role` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(255) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `user_role` (
  `user_id` int(11) NOT NULL,
  `role_id` int(11) NOT NULL,
  PRIMARY KEY  (`user_id`,`role_id`),
  UNIQUE KEY `role_id` (`role_id`),
  KEY `FK143BF46AF6AD4381` (`user_id`),
  KEY `FK143BF46A51827FA1` (`role_id`),
  CONSTRAINT `FK143BF46A51827FA1` FOREIGN KEY (`role_id`) REFERENCES `role` (`id`),
  CONSTRAINT `FK143BF46AF6AD4381` FOREIGN KEY (`user_id`) REFERENCES `user` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

通过配置**SQL**来模拟用户和权限

有了数据库的表设计，我们就可以在Spring Security中，通过配置SQL，来模拟用户和权限，这依然通过<authentication-provider>来完成：

Xml代码

```
<authentication-provider>
    <jdbc-user-service data-source-ref="dataSource"
    users-by-username-query="SELECT U.username, U.password, U.accountEnabled AS 'enabled' FROM User U wher
e U.username=?"
    authorities-by-username-query="SELECT U.username, R.name as 'authority' FROM User U JOIN Authority A O
N u.id = A.userId JOIN Role R ON R.id = A.roleId WHERE U.username=?"/>
</authentication-provider>
```

这里给出的是一个使用SQL进行模拟用户和权限的示例。其中你需要为运行SQL准备相应的dataSource。这个dataSource应该对应于Spring中的某个bean的定义。

从这段配置模拟用户和权限的情况来看，实际上Spring Security对于用户，需要username，password，accountEnabled三个字段。对于权限，它需要的是username和authority2个字段。

也就是说，如果我们能够通过其他的方式，模拟上面的这些对象，并插入到Spring Security中去，我们同样能够实现用户和权限的认证。接下来，我们就来看看我们如何通过自己的实现，来完成这件事情。

通过扩展**Spring Security**的默认实现来进行用户和权限的管理

事实上，Spring Security提供了2个认证的接口，分别用于模拟用户和权限，以及读取用户和权限的操作方法。这两个接口分别是：UserDetails和UserDetailsService。

Java代码

```java
public interface UserDetails extends Serializable {

    GrantedAuthority[] getAuthorities();

    String getPassword();

    String getUsername();

    boolean isAccountNonExpired();

    boolean isAccountNonLocked();

    boolean isCredentialsNonExpired();

    boolean isEnabled();
}
```

Java代码

```java
public interface UserDetailsService {
    UserDetails loadUserByUsername(String username)
        throws UsernameNotFoundException, DataAccessException;
}
```

非常清楚，一个接口用于模拟用户，另外一个用于模拟读取用户的过程。所以我们可以通过实现这两个接口，来完成使用数据库对用户和权限进行管理的需求。在这里，我将给出一个使用Hibernate来定义用户和权限之间关系的示例。

1. 定义**User**类和**Role**类，使他们之间形成多对多的关系

Java代码

```java
@Entity
@Proxy(lazy = false)
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class User {

        private static final long serialVersionUID = 8026813053768023527L;

    @Id
        @GeneratedValue
        private Integer id;

        private String name;

        private String password;

        private boolean disabled;

        @ManyToMany(targetEntity = Role.class, fetch = FetchType.EAGER)
    @JoinTable(name = "user_role", joinColumns = @JoinColumn(name = "user_id"), inverseJoinColumns = @JoinColumn(name = "role_id"))
    @Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
        private Set<Role> roles;

        // setters and getters
}
```

Java代码

```java
@Entity
```

```java
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Role {

        @Id
        @GeneratedValue
        private Integer id;

        private String name;

        // setters and getters
}
```

请注意这里的Annotation的写法。同时，我为User和Role之间配置了缓存。并且将他们之间的关联关系设置的lazy属性设置成false，从而保证在User对象取出之后的使用不会因为脱离session的生命周期而产生lazy loading问题。

2. 使**User**类实现**UserDetails**接口

接下来，我们让User类去实现UserDetails接口：

Java代码

```java
@Entity
@Proxy(lazy = false)
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class User implements UserDetails {

        private static final long serialVersionUID = 8026813053768023527L;

    @Id
        @GeneratedValue
        private Integer id;

        private String name;

        private String password;

        private boolean disabled;

        @ManyToMany(targetEntity = Role.class, fetch = FetchType.EAGER)
    @JoinTable(name = "user_role", joinColumns = @JoinColumn(name = "user_id"), inverseJoinColumns = @JoinColumn(name = "role_id"))
    @Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
        private Set<Role> roles;

        /**
         * The default constructor
         */
        public User() {

        }

        /* (non-Javadoc)
         * @see org.springframework.security.userdetails.UserDetails#getAuthorities()
         */
        public GrantedAuthority[] getAuthorities() {
                List<GrantedAuthority> grantedAuthorities = new ArrayList<GrantedAuthority>(roles.size());

        for(Role role : roles) {
                grantedAuthorities.add(new GrantedAuthorityImpl(role.getName()));
        }
        return grantedAuthorities.toArray(new GrantedAuthority[roles.size()]);
        }

        /* (non-Javadoc)
         * @see org.springframework.security.userdetails.UserDetails#getPassword()
         */
```

```
44.         public String getPassword() {
45.                 return password;
46.         }
47. 
48.         /* (non-Javadoc)
49.          * @see org.springframework.security.userdetails.UserDetails#getUsername()
50.          */
51.         public String getUsername() {
52.                 return name;
53.         }
54. 
55.         /* (non-Javadoc)
56.          * @see org.springframework.security.userdetails.UserDetails#isAccountNonExpired()
57.          */
58.         public boolean isAccountNonExpired() {
59.                 return true;
60.         }
61. 
62.         /* (non-Javadoc)
63.          * @see org.springframework.security.userdetails.UserDetails#isAccountNonLocked()
64.          */
65.         public boolean isAccountNonLocked() {
66.                 return true;
67.         }
68. 
69.         /* (non-Javadoc)
70.          * @see org.springframework.security.userdetails.UserDetails#isCredentialsNonExpired()
71.          */
72.         public boolean isCredentialsNonExpired() {
73.                 return true;
74.         }
75. 
76.         /* (non-Javadoc)
77.          * @see org.springframework.security.userdetails.UserDetails#isEnabled()
78.          */
79.         public boolean isEnabled() {
80.                 return !this.disabled;
81.         }
82. 
83.       // setters and getters
84. }
```

实现UserDetails接口中的每个函数，其实没什么很大的难度，除了其中的一个函数我需要额外强调一下：

Java代码

```
1.  /* (non-Javadoc)
2.   * @see org.springframework.security.userdetails.UserDetails#getAuthorities()
3.   */
4.  public GrantedAuthority[] getAuthorities() {
5.          List<GrantedAuthority> grantedAuthorities = new ArrayList<GrantedAuthority>(roles.size());
6.          for(Role role : roles) {
7.              grantedAuthorities.add(new GrantedAuthorityImpl(role.getName()));
8.          }
9.          return grantedAuthorities.toArray(new GrantedAuthority[roles.size()]);
10. }
```

这个函数的实际作用是根据User返回这个User所拥有的权限列表。如果以上面曾经用过的例子来说，如果当前User是downpour，我需要得到ROLE_USER和ROLE_ADMIN；如果当前User是robbin，我需要得到ROLE_USER。

了解了含义，实现就变得简单了，由于User与Role是多对多的关系，我们可以通过User得到所有这个User所对应的Role，并把这些Role的name拼装起来返回。

由此可见，实现UserDetails接口，并没有什么神秘的地方，它只是实际上在一定程度上只是代替了使用配置文件的硬编码：

Xml代码

```xml
<user name="downpour" password="downpour" authorities="ROLE_USER, ROLE_ADMIN" />
```

3. 实现**UserDetailsService**接口

Java代码

```java
@Repository("securityManager")
public class SecurityManagerSupport extends HibernateDaoSupport implements UserDetailsService {

    /**
     * Init sessionFactory here because the annotation of Spring 2.5 can not support override inject
     *
     * @param sessionFactory
     */
    @Autowired
    public void init(SessionFactory sessionFactory) {
        super.setSessionFactory(sessionFactory);
    }

    public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException, DataAccessException {
        List<User> users = getHibernateTemplate().find("FROM User user WHERE user.name = ? AND user.disabled = false", userName);
        if(users.isEmpty()) {
            throw new UsernameNotFoundException("User " + userName + " has no GrantedAuthority");
        }
        return users.get(0);
    }
}
```

这个实现非常简单，由于我们的User对象已经实现了UserDetails接口。所以我们只要使用Hibernate，根据userName取出相应的User对象即可。注意在这里，由于我们对于User的关联对象Roles都设置了lazy="false"，所以我们无需担心lazy loading的问题。

4. 配置文件

有了上面的代码，一切都变得很简单，重新定义authentication-provider节点即可。如果你使用Spring 2.5的Annotation配置功能，你甚至可以不需要在配置文件中定义securityManager的bean。

Xml代码

```xml
<authentication-provider user-service-ref="securityManager">
        <password-encoder hash="md5"/>
</authentication-provider>
```

使用数据库对资源进行管理

在完成了使用数据库来进行用户和权限的管理之后，我们再来看看http配置的部分。在实际应用中，我们不可能使用类似/**的方式来指定URL与权限ROLE的对应关系，而是会针对某些URL，指定某些特定的ROLE。而URL与ROLE之间的映射关系最好可以进行扩展和配置。而URL属于资源的一种，所以接下来，我们就来看看如何使用数据库来对权限和资源的匹配关系进行管理，并且将认证匹配加入到Spring Security中去。

权限和资源的设计

上面我们讲到，用户（User）和权限（Role）之间是一个多对多的关系。那么权限（Role）和资源（Resource）之间呢？其实他们之间也是一个典型的多对多的关系，我们同样用3张表来表示：

Java代码

```java
CREATE TABLE `role` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(255) default NULL,
  `description` varchar(255) default NULL,
  PRIMARY KEY  (`id`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `resource` (
  `id` int(11) NOT NULL auto_increment,
  `type` varchar(255) default NULL,
  `value` varchar(255) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `role_resource` (
  `role_id` int(11) NOT NULL,
  `resource_id` int(11) NOT NULL,
  PRIMARY KEY  (`role_id`,`resource_id`),
  KEY `FKAEE599B751827FA1` (`role_id`),
  KEY `FKAEE599B7EFD18D21` (`resource_id`),
  CONSTRAINT `FKAEE599B751827FA1` FOREIGN KEY (`role_id`) REFERENCES `role` (`id`),
  CONSTRAINT `FKAEE599B7EFD18D21` FOREIGN KEY (`resource_id`) REFERENCES `resource` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

在这里Resource可能分成多种类型，比如MENU,URL,METHOD等等。

针对资源的认证

针对资源的认证，实际上应该由Spring Security中的FilterSecurityInterceptor这个过滤器来完成。不过内置的FilterSecurityInterceptor的实现往往无法满足我们的要求，所以传统的Acegi的方式，我们往往会替换FilterSecurityInterceptor的实现，从而对URL等资源进行认证。

不过在Spring Security中，由于默认的拦截器链内置了FilterSecurityInterceptor，而且上面我们也提到过，这个实现无法被替换。这就使我们犯了难。我们如何对资源进行认证呢？

实际上，我们虽然无法替换FilterSecurityInterceptor的默认实现，不过我们可以再实现一个类似的过滤器，并将我们自己的过滤器作为一个**customer-filter**，加到默认的过滤器链的最后，从而完成整个过滤检查。

接下来我们就来看看一个完整的例子：

1. 建立权限（Role）和资源（Resource）之间的关联关系

修改上面的权限（Role）的Entity定义：

Java代码

```
@Entity
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Role {

        @Id
        @GeneratedValue
        private Integer id;

        private String name;

        @ManyToMany(targetEntity = Resource.class, fetch = FetchType.EAGER)
    @JoinTable(name = "role_resource", joinColumns = @JoinColumn(name = "role_id"), inverseJoinColumns = @JoinColumn(name = "resource_id"))
    @Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
        private Set<Resource> resources;

        // setters and getter
}
```

增加资源（Resource）的Entity定义：

Java代码

```
@Entity
```

```
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)

public class Resource {

        @Id
    @GeneratedValue
        private Integer id;

        private String type;

        private String value;

        @ManyToMany(mappedBy = "resources", targetEntity = Role.class, fetch = FetchType.EAGER)
    @Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
        private Set<Role> roles;

        /**
         * The default constructor
         */
        public Resource() {

        }
}
```

注意他们之间的多对多关系，以及他们之间关联关系的缓存和lazy属性设置。

2. 在系统启动的时候，把所有的资源load到内存作为缓存

由于资源信息对于每个项目来说，相对固定，所以我们可以将他们在系统启动的时候就load到内存作为缓存。这里做法很多，我给出的示例是将资源的存放在servletContext中。

Java代码

```
public class ServletContextLoaderListener implements ServletContextListener {

    /* (non-Javadoc)
     * @see javax.servlet.ServletContextListener#contextInitialized(javax.servlet.ServletContextEvent)
     */
    public void contextInitialized(ServletContextEvent servletContextEvent) {
        ServletContext servletContext = servletContextEvent.getServletContext();
        SecurityManager securityManager = this.getSecurityManager(servletContext);

        Map<String, String> urlAuthorities = securityManager.loadUrlAuthorities();
        servletContext.setAttribute("urlAuthorities", urlAuthorities);
    }


    /* (non-Javadoc)
     * @see javax.servlet.ServletContextListener#contextDestroyed(javax.servlet.ServletContextEvent)
     */
    public void contextDestroyed(ServletContextEvent servletContextEvent) {
        servletContextEvent.getServletContext().removeAttribute("urlAuthorities");
    }

    /**
     * Get SecurityManager from ApplicationContext
     *
     * @param servletContext
     * @return
     */
    protected SecurityManager getSecurityManager(ServletContext servletContext) {
       return (SecurityManager) WebApplicationContextUtils.getWebApplicationContext(servletContext).getBean
("securityManager");
    }

}
```

这里，我们看到了SecurityManager，这是一个接口，用于权限相关的逻辑处理。还记得之前我们使用数据库管理User的时候所使用的一个实现类SecurityManagerSupport嘛？我们不妨依然借用这个类，让它实现SecurityManager接口，来同时完成url的读取工作。

Java代码

```java
@Service("securityManager")
public class SecurityManagerSupport extends HibernateDaoSupport implements UserDetailsService, SecurityManager {

    /**
     * Init sessionFactory here because the annotation of Spring 2.5 can not support override inject
     *
     * @param sessionFactory
     */
    @Autowired
    public void init(SessionFactory sessionFactory) {
        super.setSessionFactory(sessionFactory);
    }

    /* (non-Javadoc)
     * @see org.springframework.security.userdetails.UserDetailsService#loadUserByUsername(java.lang.String)
     */
    public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException, DataAccessException {
        List<User> users = getHibernateTemplate().find("FROM User user WHERE user.name = ? AND user.disabled = false", userName);
        if(users.isEmpty()) {
            throw new UsernameNotFoundException("User " + userName + " has no GrantedAuthority");
        }
        return users.get(0);
    }

    /* (non-Javadoc)
     * @see com.javaeye.sample.security.SecurityManager#loadUrlAuthorities()
     */
    public Map<String, String> loadUrlAuthorities() {
        Map<String, String> urlAuthorities = new HashMap<String, String>();
        List<Resource> urlResources = getHibernateTemplate().find("FROM Resource resource WHERE resource.type = ?", "URL");
        for(Resource resource : urlResources) {
            urlAuthorities.put(resource.getValue(), resource.getRoleAuthorities());
        }
        return urlAuthorities;
    }
}
```

3. 编写自己的FilterInvocationDefinitionSource实现类，对资源进行认证

Java代码

```java
public class SecureResourceFilterInvocationDefinitionSource implements FilterInvocationDefinitionSource, InitializingBean {

    private UrlMatcher urlMatcher;

    private boolean useAntPath = true;

    private boolean lowercaseComparisons = true;

    /**
     * @param useAntPath the useAntPath to set
     */
    public void setUseAntPath(boolean useAntPath) {
```

```
13.         this.useAntPath = useAntPath;
14.     }
15.
16.     /**
17.      * @param lowercaseComparisons
18.      */
19.     public void setLowercaseComparisons(boolean lowercaseComparisons) {
20.         this.lowercaseComparisons = lowercaseComparisons;
21.     }
22.
23.     /* (non-Javadoc)
24.      * @see org.springframework.beans.factory.InitializingBean#afterPropertiesSet()
25.      */
26.     public void afterPropertiesSet() throws Exception {
27.
28.         // default url matcher will be RegexUrlPathMatcher
29.         this.urlMatcher = new RegexUrlPathMatcher();
30.
31.         if (useAntPath) {  // change the implementation if required
32.             this.urlMatcher = new AntUrlPathMatcher();
33.         }
34.
35.         // Only change from the defaults if the attribute has been set
36.         if ("true".equals(lowercaseComparisons)) {
37.             if (!this.useAntPath) {
38.                 ((RegexUrlPathMatcher) this.urlMatcher).setRequiresLowerCaseUrl(true);
39.             }
40.         } else if ("false".equals(lowercaseComparisons)) {
41.             if (this.useAntPath) {
42.                 ((AntUrlPathMatcher) this.urlMatcher).setRequiresLowerCaseUrl(false);
43.             }
44.         }
45.
46.     }
47.
48.     /* (non-Javadoc)
49.      * @see org.springframework.security.intercept.ObjectDefinitionSource#getAttributes(java.lang.Object)
50.      */
51.     public ConfigAttributeDefinition getAttributes(Object filter) throws IllegalArgumentException {
52.
53.         FilterInvocation filterInvocation = (FilterInvocation) filter;
54.         String requestURI = filterInvocation.getRequestUrl();
55.         Map<String, String> urlAuthorities = this.getUrlAuthorities(filterInvocation);
56.
57.         String grantedAuthorities = null;
58.         for(Iterator<Map.Entry<String, String>> iter = urlAuthorities.entrySet().iterator(); iter.hasNext()
;) {
59.             Map.Entry<String, String> entry = iter.next();
60.             String url = entry.getKey();
61.
62.             if(urlMatcher.pathMatchesUrl(url, requestURI)) {
63.                 grantedAuthorities = entry.getValue();
64.                 break;
65.             }
66.
67.         }
68.
69.         if(grantedAuthorities != null) {
70.             ConfigAttributeEditor configAttrEditor = new ConfigAttributeEditor();
71.             configAttrEditor.setAsText(grantedAuthorities);
72.             return (ConfigAttributeDefinition) configAttrEditor.getValue();
73.         }
74.
75.         return null;
76.     }
77.
78.     /* (non-Javadoc)
```

```
     * @see org.springframework.security.intercept.ObjectDefinitionSource#getConfigAttributeDefinitions()
     */
    @SuppressWarnings("unchecked")
        public Collection getConfigAttributeDefinitions() {
        return null;
    }

    /* (non-Javadoc)
     * @see org.springframework.security.intercept.ObjectDefinitionSource#supports(java.lang.Class)
     */
    @SuppressWarnings("unchecked")
        public boolean supports(Class clazz) {
        return true;
    }

    /**
     *
     * @param filterInvocation
     * @return
     */
    @SuppressWarnings("unchecked")
        private Map<String, String> getUrlAuthorities(FilterInvocation filterInvocation) {
        ServletContext servletContext = filterInvocation.getHttpRequest().getSession().getServletContext();

        return (Map<String, String>)servletContext.getAttribute("urlAuthorities");
    }

}
```

4. 配置文件修改

接下来，我们来修改一下Spring Security的配置文件，把我们自定义的这个过滤器插入到过滤器链中去。

Xml代码

```
<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-2.5.xsd
                        http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-2.0.4.xsd">

        <beans:bean id="loggerListener" class="org.springframework.security.event.authentication.LoggerListener" />

        <http access-denied-page="/403.jsp" >
                <intercept-url pattern="/static/**" filters="none" />
                <intercept-url pattern="/template/**" filters="none" />
                <intercept-url pattern="/" filters="none" />
                <intercept-url pattern="/login.jsp" filters="none" />
            <form-login login-page="/login.jsp" authentication-failure-url="/login.jsp?error=true" default-target-url="/index" />
            <logout logout-success-url="/login.jsp"/>
            <http-basic />
        </http>

        <authentication-manager alias="authenticationManager"/>

        <authentication-provider user-service-ref="securityManager">
                <password-encoder hash="md5"/>
        </authentication-provider>

        <beans:bean id="accessDecisionManager" class="org.springframework.security.vote.AffirmativeBased">

            <beans:property name="allowIfAllAbstainDecisions" value="false"/>
```

```
            <beans:property name="decisionVoters">
                <beans:list>
                    <beans:bean class="org.springframework.security.vote.RoleVoter"/>
                    <beans:bean class="org.springframework.security.vote.AuthenticatedVoter"/>
                </beans:list>
            </beans:property>
        </beans:bean>

        <beans:bean id="resourceSecurityInterceptor" class="org.springframework.security.intercept.web.FilterSecurityInterceptor">
                <beans:property name="authenticationManager" ref="authenticationManager"/>
            <beans:property name="accessDecisionManager" ref="accessDecisionManager"/>
            <beans:property name="objectDefinitionSource" ref="secureResourceFilterInvocationDefinitionSource" />
            <beans:property name="observeOncePerRequest" value="false" />
            <custom-filter after="LAST" />
        </beans:bean>

        <beans:bean id="secureResourceFilterInvocationDefinitionSource" class="com.javaeye.sample.security.interceptor.SecureResourceFilterInvocationDefinitionSource" />

</beans:beans>
```

请注意，由于我们所实现的，是FilterSecurityInterceptor中的一个开放接口，所以我们实际上定义了一个新的bean，并通过<custom-filter after="LAST" />插入到过滤器链中去。

Spring Security对象的访问

1. 访问当前登录用户

Spring Security提供了一个线程安全的对象：SecurityContextHolder，通过这个对象，我们可以访问当前的登录用户。我写了一个类，可以通过静态方法去读取：

Java代码

```
public class SecurityUserHolder {

        /**
         * Returns the current user
         *
         * @return
         */
        public static User getCurrentUser() {
                return (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        }

}
```

2. 访问当前登录用户所拥有的权限

通过上面的分析，我们知道，用户所拥有的所有权限，其实是通过UserDetails接口中的getAuthorities()方法获得的。只要实现这个接口，就能实现需求。在我的代码中，不仅实现了这个接口，还在上面做了点小文章，这样我们可以获得一个用户所拥有权限的字符串表示：

Java代码

```
/* (non-Javadoc)
 * @see org.springframework.security.userdetails.UserDetails#getAuthorities()
 */
public GrantedAuthority[] getAuthorities() {
        List<GrantedAuthority> grantedAuthorities = new ArrayList<GrantedAuthority>(roles.size());
for(Role role : roles) {
        grantedAuthorities.add(new GrantedAuthorityImpl(role.getName()));
}
return grantedAuthorities.toArray(new GrantedAuthority[roles.size()]);
```

```
}

/**
 * Returns the authorites string
 *
 * eg.
 *    downpour --- ROLE_ADMIN,ROLE_USER
 *    robbin --- ROLE_ADMIN
 *
 * @return
 */
public String getAuthoritiesString() {
    List<String> authorities = new ArrayList<String>();
    for(GrantedAuthority authority : this.getAuthorities()) {
        authorities.add(authority.getAuthority());
    }
    return StringUtils.join(authorities, ",");
}
```

3. 访问当前登录用户能够访问的资源

这就涉及到用户（User），权限（Role）和资源（Resource）三者之间的对应关系。我同样在User对象中实现了一个方法：

Java代码

```
/**
 * @return the roleResources
 */
public Map<String, List<Resource>> getRoleResources() {
        // init roleResources for the first time
        if(this.roleResources == null) {
                this.roleResources = new HashMap<String, List<Resource>>();

                for(Role role : this.roles) {
                        String roleName = role.getName();
                        Set<Resource> resources = role.getResources();
                        for(Resource resource : resources) {
                                String key = roleName + "_" + resource.getType();
                                        if(!this.roleResources.containsKey(key)) {
                                                this.roleResources.put(key, new ArrayList<Resource>());
                                        }
                                        this.roleResources.get(key).add(resource);

                        }
                }

        }
        return this.roleResources;
}
```

这里，会在User对象中设置一个缓存机制，在第一次取的时候，通过遍历User所有的Role，获取相应的Resource信息。

代码示例

在附件中，我给出了一个简单的例子，把我上面所讲到的所有内容整合在一起，是一个eclipse的工程，大家可以下载进行参考。

SpringSecurity.zip (48.7 KB)
下载次数: 7221

查看图片附件

推荐链接

返回顶楼

lcllcl987

等级:

文章: 120

积分: 111

来自: 武汉

我现在离线

发表时间：2009-06-05

这个堪称最佳实践啊， 比AppFuse的好

返回顶楼 回帖地址 1 0 请登录后投票

maike

等级: 初级会员

文章: 21

积分: 0

来自: 深圳

我现在离线

发表时间：2009-06-05

想问下:

```
<beans:bean id="resourceSecurityInterceptor" class="org.springframework.security.intercept.web.FilterSecurityInterceptor">
        <beans:property name="authenticationManager" ref="authenticationManager"/>
        <beans:property name="accessDecisionManager" ref="accessDecisionManager"/>
        <beans:property name="objectDefinitionSource" ref="secureResourceFilterInvocationDefinitionSource" />
        <beans:property name="observeOncePerRequest" value="false" />
        <custom-filter after="LAST" />
    </beans:bean>
```

这个resourceSecurityInterceptor在那个地方被调用？谢谢

返回顶楼 回帖地址 1 0 请登录后投票

kjj

等级:

文章: 584

积分: 266

来自: 陕西

我现在离线

发表时间：2009-06-05

je的数据库神经了吧，我记得这个帖子是老帖子了，回复很多了，怎么突然变得这么少了！！！！

返回顶楼 回帖地址 3 0 请登录后投票

SINCE1978
等级: 初级会员

文章: 41
积分: 20
来自: 济南

我现在离线

发表时间: 2009-06-08

SecurityManagerSupport类的这个方法:

```
public Map<String, String> loadUrlAuthorities() {
        ...
        for(Resource resource : urlResources) {
            urlAuthorities.put(resource.getValue(), resource.getRoleAuthorities());
        }
        ...
    }
```

里面的for循环是否应写为:

```
for(OperateRoleVO vo : urlResources) {
        if(!urlAuthorities.containsKey(vo.getOperateUrl()))
        urlAuthorities.put(vo.getOperateUrl(), vo.getRoleName());
        else
        urlAuthorities.put(vo.getOperateUrl(), urlAuthorities.get(vo.getOperateUrl())+","+vo.getRoleName());
        }
```

返回顶楼 回帖地址 1 3 请登录后投票

daquan198163
等级:

文章: 1874
积分: 1491
来自: 吉林->北京->上海

我现在离线

发表时间: 2009-06-09

对呀，n页的回复怎么都不见了，都被投隐藏了？

返回顶楼 回帖地址 2 1 请登录后投票

SINCE1978
等级: 初级会员

文章: 41
积分: 20
来自: 济南

我现在离线

发表时间: 2009-06-15

说实话ss2的配置不必acegi强多少

返回顶楼 回帖地址 0 10 请登录后投票

Koctr
等级: 初级会员

发表时间: 2009-06-16

断断续续折腾了我2个星期，总算用非注解的方式实现了，并且整合了strtus2

那个md5加密以后，没说密码是多少，汗.....

其实建议能在一开始就把表结构写全了，而不是在后面又改

文章: 1
积分: 60
来自: 呼和浩特 -> 北京

我现在离线

返回顶楼 回帖地址 0 0 请登录后投票

sogo1986
等级: 初级会员

发表时间：2009-06-17

神贴 膜拜。。。。。。。。

文章: 38
积分: 30
来自: 卡利姆多

我现在离线

返回顶楼 回帖地址 0 0 请登录后投票

harman001
等级: 初级会员

发表时间：2009-06-17

受教了！。

文章: 2
积分: 30
来自: 上海

我现在离线

返回顶楼 回帖地址 0 0 请登录后投票