

Java

博客 微博 相册 收藏 留言 关于我



Dead\_knight

浏览: 227535 次

性别:

来自: 杭州

我现在离线

最近访客 [更多访客>>](#)



[doudou15223](#) [renwenlong9](#)



[clearblack](#) [dylinshi126](#)

博客专栏

[Spring Security...](#)  
浏览量: 57482

[clojure 专题](#)  
浏览量: 8494

[WebLogic11g](#)  
浏览量: 23706

文章分类

- 全部博客 (90)
- Java 基础 (8)
- J2EE (11)
- DataBase (5)
- flex (1)
- web (3)
- 性能优化 (2)
- Spring Security (21)

Spring Security3源码分析-FilterSecurityInterceptor分析

博客分类: [Spring Security](#)

FilterSecurityInterceptor过滤器对应的类路径为

org.springframework.security.web.access.intercept.FilterSecurityInterceptor

这个filter是filterchain中比较复杂,也是比较核心的过滤器,主要负责授权的工作

在看这个filter源码之前,先来看看spring是如何构造filter这个bean的

具体的构造过程的代码片段为

Java代码



```
1. //这个方法源自HttpConfigurationBuilder类
2. void createFilterSecurityInterceptor(BeanReference authManager) {
3.     //判断是否配置了use-expressions属性
4.     boolean useExpressions = FilterInvocationSecurityMetadataSourceParser.isUseExpressions(httpElt);
5.     //根据intercept-url标签列表创建授权需要的元数据信息。后面仔细分析
6.     BeanDefinition securityMds = FilterInvocationSecurityMetadataSourceParser.createSecurityMetadataSource(interceptUrls, httpElt, pc);
7.
8.     RootBeanDefinition accessDecisionMgr;
9.     //创建voter列表
10.    ManagedList<BeanDefinition> voters = new ManagedList<BeanDefinition>(2);
11.    //如果是使用了表达式,使用WebExpressionVoter
12.    //没使用表达式,就使用RoleVoter、AuthenticatedVoter
13.    if (useExpressions) {
14.        voters.add(new RootBeanDefinition(WebExpressionVoter.class));
15.    } else {
16.        voters.add(new RootBeanDefinition(RoleVoter.class));
17.        voters.add(new RootBeanDefinition(AuthenticatedVoter.class));
18.    }
19.    //定义授权的决策管理类AffirmativeBased
20.    accessDecisionMgr = new RootBeanDefinition(AffirmativeBased.class);
21.    //添加依赖的voter列表
22.    accessDecisionMgr.getPropertyValues().addPropertyValue("decisionVoters", voters);
23.    accessDecisionMgr.setSource(pc.extractSource(httpElt));
24.
25.    // Set up the access manager reference for http
26.    String accessManagerId = httpElt.getAttribute(ATT_ACCESS_MGR);
27.    //如果未定义access-decision-manager-ref属性,就使用默认的
28.    //AffirmativeBased
29.    if (!StringUtils.hasText(accessManagerId)) {
30.        accessManagerId = pc.getReaderContext().generateBeanName(accessDecisionMgr);
31.        pc.registerBeanComponent(new BeanComponentDefinition(accessDecisionMgr, accessManagerId));
32.    }
33.    //创建FilterSecurityInterceptor过滤器
34.    BeanDefinitionBuilder builder = BeanDefinitionBuilder.rootBeanDefinition(FilterSecurityInterceptor.class);
35.    //添加决策管理器
36.    builder.addPropertyReference("accessDecisionManager", accessManagerId);
```

- [weblogic \(20\)](#)
- [Spring Integration \(2\)](#)
- [clojure \(12\)](#)
- [Snaker \(2\)](#)
- [Shiro 源码分析 \(2\)](#)

社区版块

- [我的资讯](#) (6)
- [我的论坛](#) (71)
- [我的问答](#) (465)

## 存档分类

- [2014-04](#) (2)
- [2013-11](#) (3)
- [2013-09](#) (9)
- [更多存档...](#)

评论排行榜

- [开源流程引擎Snaker](#)
- [WebLogic11g-负载均衡](#)
- [技术资料汇总分享](#)
- [Shiro源码分析-初始化-Realm](#)
- [Shiro源码分析-初始化-SecurityManager](#)

## 最新评论

[a492846462](#)：我执行了installNodeMgrSvc.cmd这个命令。 ...

[WebLogic11g-半小时让你懂的domain集群化](#)

maksimfsf: 相当不错“楼主可否  
在重新发一下，网盘确实失效了  
技术资料汇总分享

guhanjie: 兄弟, 看你整理的资料目录很不错, 可惜百度网盘失效了, 能否分享一 ...

技术资料汇总分享

Dead\_knight: pi88dian88 写  
道LZ你好,我在Snaker网站上 h  
...  
开源流程引擎Snaker

pi88dian88: LZ你好，我

```

37. //添加认证管理类
38. builder.addPropertyValue("authenticationManager", authManager);
39.
40. if ("false".equals(httpElt.getAttribute(ATT_ONCE_PER_REQUEST))) {
41.     builder.addPropertyValue("observeOncePerRequest", Boolean.FALSE);
42. }
43. //添加授权需要的安全元数据资源
44. builder.addPropertyValue("securityMetadataSource", securityMds);
45. BeanDefinition fsiBean = builder.getBeanDefinition();
46. //向ioc容器注册bean
47. String fsiId = pc.getReaderContext().generateBeanName(fsiBean);
48. pc.registerBeanComponent(new BeanComponentDefinition(fsiBean, fsiId));
49.
50. // Create and register a DefaultWebInvocationPrivilegeEvaluator for use with taglibs etc.
51. BeanDefinition wipe = new RootBeanDefinition(DefaultWebInvocationPrivilegeEvaluator.class);
52. ;
53. wipe.getConstructorArgumentValues().addGenericArgumentValue(new RuntimeBeanReference(fsiId));
54. pc.registerBeanComponent(new BeanComponentDefinition(wipe, pc.getReaderContext().generateBeanName(wipe)));
55.
56. this.fsi = new RuntimeBeanReference(fsiId);
57. }

```

现在再仔细分析创建元数据资源的bean过程

Java代码 

```


1. static BeanDefinition createSecurityMetadataSource(List<Element> interceptUrls, Element elt, ParserContext pc) {
2.     //创建Url处理类, 有两个实现: AntUrlPathMatcher、RegexUrlPathMatcher
3.     UrlMatcher matcher = HttpSecurityBeanDefinitionParser.createUrlMatcher(elt);
4.     boolean useExpressions = isUseExpressions(elt);
5.     //解析intercept-url标签, 构造所有需要拦截url的map信息
6.     //map中的key: RequestKey的bean定义, value: SecurityConfig的bean定义
7.     ManagedMap<BeanDefinition, BeanDefinition> requestToAttributesMap = parseInterceptUrlsForFilterInvocationRequestMap(
8.         interceptUrls, useExpressions, pc);
9.     BeanDefinitionBuilder fidsBuilder;
10.
11.     if (useExpressions) {
12.         //定义表达式处理类的bean
13.         Element expressionHandlerElt = DomUtils.getChildElementByTagName(elt, Elements.EXPRESSION_HANDLER);
14.         String expressionHandlerRef = expressionHandlerElt == null ? null : expressionHandlerElt.getAttribute("ref");
15.
16.         if (StringUtils.hasText(expressionHandlerRef)) {
17.             logger.info("Using bean " + expressionHandlerRef + " as web SecurityExpressionHandler implementation");
18.         } else {
19.             BeanDefinition expressionHandler = BeanDefinitionBuilder.rootBeanDefinition(DefaultWebSecurityExpressionHandler.class).getBeanDefinition();
20.             expressionHandlerRef = pc.getReaderContext().generateBeanName(expressionHandler);
21.
22.             pc.registerBeanComponent(new BeanComponentDefinition(expressionHandler, expressionHandlerRef));
23.         }
24.         //定义表达式类型的FilterInvocationSecurityMetadataSource
25.         fidsBuilder = BeanDefinitionBuilder.rootBeanDefinition(ExpressionBasedFilterInvocationSecurityMetadataSource.class);
26.         //通过构造函数注入依赖

```

在Snaker网站上 [http://snakerf ...](http://snakerf...)  
[开源流程引擎Snaker](#)


```
26.         fidsBuilder.addConstructorArgValue(matcher);
27.         fidsBuilder.addConstructorArgValue(requestToAttributesMap);
28.         fidsBuilder.addConstructorArgReference(expressionHandlerRef);
29.     } else {
30.         //定义非表达式类型的FilterInvocationSecurityMetadataSource
31.         fidsBuilder = BeanDefinitionBuilder.rootBeanDefinition(DefaultFilterInvocationSecurity
MetadataSource.class);
32.         //通过构造函数注入依赖
33.         fidsBuilder.addConstructorArgValue(matcher);
34.         fidsBuilder.addConstructorArgValue(requestToAttributesMap);
35.     }
36.
37.     fidsBuilder.addPropertyValue("stripQueryStringFromUrls", matcher instanceof AntUrlPathMatc
her);
38.     fidsBuilder.getRawBeanDefinition().setSource(pc.extractSource(elt));
39.
40.     return fidsBuilder.getBeanDefinition();
41. }
```

通过以上的bean构造过程，FilterSecurityInterceptor所依赖的决策管理器、认证管理器、安全元数据资源都具备了，该让FilterSecurityInterceptor干活了，其源码为

Java代码 

```
1. public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
2.     throws IOException, ServletException {
3.     //封装request, response, chain, 方便参数传递、增加代码阅读性
4.     FilterInvocation fi = new FilterInvocation(request, response, chain);
5.     invoke(fi);
6. }
7.
8. public void invoke(FilterInvocation fi) throws IOException, ServletException {
9.     if ((fi.getRequest() != null) && (fi.getRequest().getAttribute(FILTER_APPLIED) != null)
&& observeOncePerRequest) {
10.         if (fi.getRequest() != null) {
11.             fi.getRequest().setAttribute(FILTER_APPLIED, Boolean.TRUE);
12.         }
13.         //执行父类beforeInvocation, 类似于aop中的before
14.         InterceptorStatusToken token = super.beforeInvocation(fi);
15.
16.         try {
17.             //filter传递
18.             fi.getChain().doFilter(fi.getRequest(), fi.getResponse());
19.         } finally {
20.             //执行父类的afterInvocation, 类似于aop中的after
21.             super.afterInvocation(token, null);
22.         }
23.     }
24. }
25. }
```


继续看父类的beforeInvocation方法，其中省略了一些不重要的代码片段

Java代码 

```
1. protected InterceptorStatusToken beforeInvocation(Object object) {
2.     //根据SecurityMetadataSource获取配置的权限属性
3.     Collection<ConfigAttribute> attributes = this.obtainSecurityMetadataSource().getAttributes
(object);
4.     //省略.....
5.     //判断是否需要对认证实体重新认证，默认为否
6.     Authentication authenticated = authenticateIfRequired();
7. }
```

```
8.      // Attempt authorization
9.      try {
10.         // 决策管理器开始决定是否授权, 如果授权失败, 直接抛出AccessDeniedException
11.         this.accessDecisionManager.decide(authenticated, object, attributes);
12.     }
13.     catch (AccessDeniedException accessDeniedException) {
14.         publishEvent(new AuthorizationFailureEvent(object, attributes, authenticated,
15.             accessDeniedException));
16.
17.         throw accessDeniedException;
18.     }
19. }
```

增加说明

Java代码 

```
1.  Collection<ConfigAttribute> attributes = this.obtainSecurityMetadataSource().getAttributes(object);
```

这里获取的是权限列表信息, 比如说有这个配置

```
<security:intercept-url pattern="/index.jsp*" access="ROLE_USER,ROLE_ADMIN"/>
```


如果现在发起一个请求时index.jsp, 那么根据这个请求返回的attributes集合就是分别包含ROLE\_USER,ROLE\_ADMIN属性的两个SecurityConfig对象

至于请求url如何匹配的, 大家可以通过阅读DefaultFilterInvocationSecurityMetadataSource类的源码, 实际上, 这里用到了spring的路径匹配工具类org.springframework.util.AntPathMatcher

AntPathMatcher匹配方式的通配符有三种:

? (匹配任何单字符), \* (匹配0或者任意数量的字符), \*\* (匹配0或者更多的目录)

由于之前在bean的定义过程已经知道决策管理器是AffirmativeBased, 接着看AffirmativeBased的决策过程

Java代码 

```
1.  public void decide(Authentication authentication, Object object, Collection<ConfigAttribute> configAttributes)
2.      throws AccessDeniedException {
3.      int deny = 0;
4.      // 循环voters, 实际上是RoleVoter、AuthenticatedVoter
5.      for (AccessDecisionVoter voter : getDecisionVoters()) {
6.          // 把具体的决策任务交给voter处理
7.          // voter只返回-1、0、1, 只有为1才算授权成功
8.          int result = voter.vote(authentication, object, configAttributes);
9.
10.         if (logger.isDebugEnabled()) {
11.             logger.debug("Voter: " + voter + ", returned: " + result);
12.         }
13.
14.         switch (result) {
15.             case AccessDecisionVoter.ACCESS_GRANTED:
16.                 return;
17.
18.             case AccessDecisionVoter.ACCESS_DENIED:
19.                 deny++;
20.
21.                 break;
22.
23.             default:
24.                 break;
25.         }
26.     }
```

```
27. //只要有一个voter拒绝了，则直接抛出访问拒绝异常
28. if (deny > 0) {
29.     throw new AccessDeniedException(messages.getMessage("AbstractAccessDecisionManager.accessDenied",
30.                                                         "Access is denied"));
31. }
32.
33. // To get this far, every AccessDecisionVoter abstained
34. checkAllowIfAllAbstainDecisions();
35. }
```

实际上，有三种决策管理器，分别为**AffirmativeBased**、**ConsensusBased**、**UnanimousBased**，各自决策的区别是：


**AffirmativeBased**：只要有一个voter投同意票，就授权成功

**ConsensusBased**：只要投同意票的大于投反对票的，就授权成功

**UnanimousBased**：需要一致通过才授权成功具体决策规则很简单，只是根据voter返回的结果做处理


接下来，分别看RoleVoter、AuthenticatedVoter的源码

RoleVoter：

Java代码 

```
1. public int vote(Authentication authentication, Object object, Collection<ConfigAttribute> attributes) {
2.     int result = ACCESS_ABSTAIN;
3.     //从认证实体中获取所有的权限列表
4.     Collection<GrantedAuthority> authorities = extractAuthorities(authentication);
5.     //循环intercept-url配置的access权限列表
6.     for (ConfigAttribute attribute : attributes) {
7.         if (this.supports(attribute)) {
8.             result = ACCESS_DENIED;
9.
10.            // Attempt to find a matching granted authority
11.            //循环认证实体所拥有的权限列表
12.            for (GrantedAuthority authority : authorities) {
13.                if (attribute.getAttribute().equals(authority.getAuthority())) {
14.                    //只要有相同的权限，直接返回成功1
15.                    return ACCESS_GRANTED;
16.                }
17.            }
18.        }
19.    }
20.
21.    return result;
22. }
```

AuthenticatedVoter：

Java代码 

```
1. public int vote(Authentication authentication, Object object, Collection<ConfigAttribute> attributes) {
2.     int result = ACCESS_ABSTAIN;
3.
4.     for (ConfigAttribute attribute : attributes) {
5.         if (this.supports(attribute)) {
6.             result = ACCESS_DENIED;
7.
8.             if (IS_AUTHENTICATED_FULLY.equals(attribute.getAttribute())) {
9.                 if (isFullyAuthenticated(authentication)) {
10.                    return ACCESS_GRANTED;
11.                }
12.            }
13.        }
14.    }
15.
16.    return result;
17. }
```

```
13.         if (IS_AUTHENTICATED_REMEMBERED.equals(attribute.getAttribute())) {
14.             if (authenticationTrustResolver.isRememberMe(authentication)
15.                 || isFullyAuthenticated(authentication)) {
16.                 return ACCESS_GRANTED;
17.             }
18.         }
19.     }
20.
21.     if (IS_AUTHENTICATED_ANONYMOUSLY.equals(attribute.getAttribute())) {
22.         if (authenticationTrustResolver.isAnonymous(authentication) || isFullyAuthent
23.             icated(authentication)
24.                 || authenticationTrustResolver.isRememberMe(authentication)) {
25.             return ACCESS_GRANTED;
26.         }
27.     }
28. }
29.
30. return result;
31. }
```

由于RoleVoter在list列表中的位置处于AuthenticatedVoter前面，只要RoleVoter通过，就不会再执行AuthenticatedVoter了。实际上AuthenticatedVoter只会对IS\_AUTHENTICATED\_FULLY、IS\_AUTHENTICATED\_REMEMBERED、IS\_AUTHENTICATED\_ANONYMOUS三种权限做vote处理。

分享到:  

◀ [Spring Security3源码分析-BasicAuthentica ...](#) | [Spring Security3源码分析-SecurityContext ...](#) ▶

2012-05-07 17:31 | 浏览 3251 | [评论\(0\)](#) | 分类:[企业架构](#) | [相关推荐](#) [▶ MORE](#)

评论

发表评论



[您还没有登录,请您登录后再发表评论](#)