南轲梦

随笔- 29 文章- 0 评论- 149

话说Spring Security权限管理（源码）

最近项目需要用到**Spring Security**的权限控制，故花了点时间简单的去看了一下其权限控制相关的源码(版本为4.2)。

AccessDecisionManager

spring security是通过**AccessDecisionManager**进行授权管理的，先来张官方图镇楼。

AccessDecisionManager

AccessDecisionManager 接口定义了如下方法：

```
//调用AccessDecisionVoter进行投票（关键方法）
void decide(Authentication authentication, Object object,
        Collection<ConfigAttribute> configAttributes)
throws AccessDeniedException,
        InsufficientAuthenticationException;

boolean supports(ConfigAttribute attribute);
boolean supports(Class clazz);
```

接下来看看它的实现类的具体实现：

AffirmativeBased

```
public void decide(Authentication authentication, Object
object,
        Collection<ConfigAttribute> configAttributes)
```

积分与排名

积分 - 85970
排名 - 7518

随笔分类

随笔档案

最新评论

1. Re:Spring MVC错误页面配置

其实这里有问题，web.xml中配置的属于容器或者说Web应用异常。springmvc-servler.xml中配置的是

```
throws AccessDeniedException {
    int deny = 0;

    for (AccessDecisionVoter voter : getDecisionVoters())
{
        //调用AccessDecisionVoter进行vote(我们姑且称之为投票
吧)，后面再看vote的源码。
        int result = voter.vote(authentication, object,
configAttributes);

        if (logger.isDebugEnabled()) {
            logger.debug("Voter: " + voter + ", returned:
" + result);
        }

        switch (result) {
        case AccessDecisionVoter.ACCESS_GRANTED://值为1
            //只要有voter投票为ACCESS_GRANTED，则通过
            return;

        case AccessDecisionVoter.ACCESS_DENIED://值为-1
            deny++;

            break;

        default:
            break;
        }
    }

    if (deny > 0) {
        //如果有两个及以上AccessDecisionVoter(姑且称之为投票者
吧)都投ACCESS_DENIED，则直接就不通过了
        throw new
AccessDeniedException(messages.getMessage(

"AbstractAccessDecisionManager.accessDenied", "Access is
denied"));
    }

    // To get this far, every AccessDecisionVoter
abstained
    checkAllowIfAllAbstainDecisions();
}
```

源码中，有个Collection configAttributes 参数，ConfigAttribute是什么？ 这个其实是一个很灵活的东西，不同的情况代表不同的语义，比如在使用了角色控制的时候，传入的则可能是ROLE__XXX之

DispatcherServlet自己能够处理跳转到异常。

--KotVar

2. Re:shell编程其实真的很简单（一）

很好，收藏了

--atomic_weapon

3. Re:深入浅出Mybatis系列（十）---SQL执行流程分析（源码篇）

不错，入门的好文章

--陈其苗

4. Re:shell编程其实真的很简单（五）

不继续写了吗博主 写的很好啊

--孤独的静态变量

5. Re:shell编程其实真的很简单（一）

有点问题 正确应该如下 files=$(ls -al)#!/bin/bashpath=$(pwd)files=$(ls -al)echo current path: $pathecho files: ...

--代码让自己变强

阅读排行榜

1. 深入浅出Mybatis系列（八）---mapper映射文件配置之select、resultMap(97249)
2. 深入浅出Mybatis系列（七）---mapper映射文件配置之insert、update、delete(82037)
3. 深入浅出Mybatis系列（九）---强大的动态SQL(69567)
4. shell编程其实真的很简单（一）(67422)
5. 深入浅出Mybatis系列（五）---TypeHandler简介及配置（mybatis源码篇）(55270)

评论排行榜

1. 深入浅出Mybatis系列（十）---SQL执行流程分析（源码篇）(20)
2. 深入浅出Mybatis系列（一）---Mybatis入门(14)
3. 深入浅出Mybatis系列（八）---map

类的，以便ROLE_VOTER使用。具体的后面在细说。

通过以上代码可直接看到AffirmativeBased的策略：

- 只要有投通过（ACCESS_GRANTED）票，则直接判为通过。
- 如果没有投通过票且反对（ACCESS_DENIED）票在两个及其以上的，则直接判为不通过。

UnanimousBased

```
public void decide(Authentication authentication, Object object,
        Collection<ConfigAttribute> attributes) throws AccessDeniedException {

    int grant = 0;
    int abstain = 0;

    List<ConfigAttribute> singleAttributeList = new ArrayList<ConfigAttribute>(1);
    singleAttributeList.add(null);

    for (ConfigAttribute attribute : attributes) {
        singleAttributeList.set(0, attribute);

        for (AccessDecisionVoter voter : getDecisionVoters()) {
            //配置的投票者进行投票
            int result = voter.vote(authentication, object, singleAttributeList);

            if (logger.isDebugEnabled()) {
                logger.debug("Voter: " + voter + ", returned: " + result);
            }

            switch (result) {
            case AccessDecisionVoter.ACCESS_GRANTED:
                grant++;

                break;

            case AccessDecisionVoter.ACCESS_DENIED:
                //只要有投票者投反对票就立马判为无权访问
                throw new AccessDeniedException(messages.getMessage(

"AbstractAccessDecisionManager.accessDenied",
```

```
                        "Access is denied"));

            default:
                abstain++;

                break;
            }
        }
    }

    // To get this far, there were no deny votes
    if (grant > 0) {
        //如果没反对票且有通过票，那么就判为通过
        return;
    }

    // To get this far, every AccessDecisionVoter
abstained
    checkAllowIfAllAbstainDecisions();
}
```

由此可见UnanimousBased的策略：

- 无论多少投票者投了多少通过（ACCESS_GRANTED）票，只要有反对票（ACCESS_DENIED），那都判为不通过。
- 如果没有反对票且有投票者投了通过票，那么就判为通过。

ConsensusBased

```
public void decide(Authentication authentication, Object
object,
        Collection<ConfigAttribute> configAttributes)
throws AccessDeniedException {
    int grant = 0;
    int deny = 0;
    int abstain = 0;

    for (AccessDecisionVoter voter : getDecisionVoters())
{
        //配置的投票者进行投票
        int result = voter.vote(authentication, object,
configAttributes);

        if (logger.isDebugEnabled()) {
            logger.debug("Voter: " + voter + ", returned:
" + result);
        }
```

```
            switch (result) {
            case AccessDecisionVoter.ACCESS_GRANTED:
                grant++;

                break;

            case AccessDecisionVoter.ACCESS_DENIED:
                deny++;

                break;

            default:
                abstain++;

                break;
            }
        }

        if (grant > deny) {
            //通过的票数大于反对的票数则判为通过
            return;
        }

        if (deny > grant) {
            //通过的票数小于反对的票数则判为不通过
            throw new
AccessDeniedException(messages.getMessage(

"AbstractAccessDecisionManager.accessDenied", "Access is
denied"));
        }

        if ((grant == deny) && (grant != 0)) {
            //this.allowIfEqualGrantedDeniedDecisions默认为
true
            //通过的票数和反对的票数相等，则可根据配置
allowIfEqualGrantedDeniedDecisions进行判断是否通过
            if (this.allowIfEqualGrantedDeniedDecisions) {
                return;
            }
            else {
                throw new
AccessDeniedException(messages.getMessage(

"AbstractAccessDecisionManager.accessDenied", "Access is
denied"));
            }
        }
```

```
    // To get this far, every AccessDecisionVoter
abstained
    checkAllowIfAllAbstainDecisions();
}
```

由此可见，ConsensusBased的策略：

- 通过的票数大于反对的票数则判为通过。
- 通过的票数小于反对的票数则判为不通过。
- 通过的票数和反对的票数相等，则可根据配置allowIfEqualGrantedDeniedDecisions（默认为true）进行判断是否通过。

到此，应该明白AffirmativeBased、UnanimousBased、ConsensusBased三者的区别了吧，spring security**默认使用的是AffirmativeBased**， 如果有需要，可配置为其它两个，也可自己去实现。

投票者

以上AccessDecisionManager的实现类都只是对权限（投票）进行管理（策略的实现），具体投票（vote）的逻辑是通过调用AccessDecisionVoter的子类（投票者）的vote方法实现的。spring security默认注册了RoleVoter和AuthenticatedVoter两个投票者。下面来看看其源码。

AccessDecisionManager

```
boolean supports(ConfigAttribute attribute);
boolean supports(Class<?> clazz);
//核心方法，此方法由上面介绍的的AccessDecisionManager调用，子类实现此方法进行投票。
int vote(Authentication authentication, S object,
        Collection<ConfigAttribute> attributes);
```

RoleVoter

```
private String rolePrefix = "ROLE_";

//只处理ROLE_开头的（可通过配置rolePrefix的值进行改变）
public boolean supports(ConfigAttribute attribute) {
    if ((attribute.getAttribute() != null)
            &&
```

```
attribute.getAttribute().startsWith(getRolePrefix())) {
            return true;
        }
        else {
            return false;
        }
    }

    public int vote(Authentication authentication, Object
object,
            Collection<ConfigAttribute> attributes) {

        if(authentication == null) {
            //用户没通过认证，则投反对票
            return ACCESS_DENIED;
        }
        int result = ACCESS_ABSTAIN;
        //获取用户实际的权限
        Collection<? extends GrantedAuthority> authorities =
extractAuthorities(authentication);

        for (ConfigAttribute attribute : attributes) {
            if (this.supports(attribute)) {
                result = ACCESS_DENIED;

                // Attempt to find a matching granted
authority
                for (GrantedAuthority authority :
authorities) {
                    if
(attribute.getAttribute().equals(authority.getAuthority()
)) {
                        //权限匹配则投通过票
                        return ACCESS_GRANTED;
                    }
                }
            }
        }
        //如果处理过，但没投通过票，则为反对票，如果没处理过，那么视为
弃权（ACCESS_ABSTAIN）。
        return result;
    }
```

很简单吧，同时，我们还可以通过实现AccessDecisionManager来扩展自己的voter。但是，要实现这个，我们还必须得弄清楚attributes这个参数是从哪儿来的，这个是个很关键的参数啊。通过一张官方图能很清晰的看出这个问题来：

接下来，就看看AccessDecisionManager的调用者AbstractSecurityInterceptor。

AbstractSecurityInterceptor

```
...
//上面说过默认是AffirmativeBased，可配置
private AccessDecisionManager accessDecisionManager;
...
protected InterceptorStatusToken beforeInvocation(Object
object) {
    ...
    //抽象方法，子类实现，但由此也可看出ConfigAttribute是由
SecurityMetadataSource（实际上，默认是
DefaultFilterInvocationSecurityMetadataSource）获取。
    Collection<ConfigAttribute> attributes =
this.obtainSecurityMetadataSource()
            .getAttributes(object);
    ...
    //获取当前认证过的用户信息
    Authentication authenticated =
authenticateIfRequired();

    try {
        //调用AccessDecisionManager
        this.accessDecisionManager.decide(authenticated,
object, attributes);
    }
    catch (AccessDeniedException accessDeniedException) {
        publishEvent(new
AuthorizationFailureEvent(object, attributes,
authenticated,
                accessDeniedException));

        throw accessDeniedException;
    }
    ...
}

public abstract SecurityMetadataSource
obtainSecurityMetadataSource();
```

以上方法都是由AbstractSecurityInterceptor的子类（默认是FilterSecurityInterceptor）调用，那就再看看吧：

FilterSecurityInterceptor

```
...
//SecurityMetadataSource的实现类，由此可见，可通过外部配置。这也说明我们可以通过自定义SecurityMetadataSource的实现类来扩展出自己实际需要的ConfigAttribute
private FilterInvocationSecurityMetadataSource securityMetadataSource;
...
//入口
public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
    FilterInvocation fi = new FilterInvocation(request, response, chain);
    //关键方法
    invoke(fi);
}

public void invoke(FilterInvocation fi) throws IOException, ServletException {
    if ((fi.getRequest() != null)
            &&
(fi.getRequest().getAttribute(FILTER_APPLIED) != null)
            && observeOncePerRequest) {
        // filter already applied to this request and user wants us to observe
        // once-per-request handling, so don't re-do security checking
        fi.getChain().doFilter(fi.getRequest(), fi.getResponse());
    }
    else {
        // first time this request being called, so perform security checking
        if (fi.getRequest() != null) {
            fi.getRequest().setAttribute(FILTER_APPLIED, Boolean.TRUE);
        }
        //在这儿调用了父类（AbstractSecurityInterceptor）的方法， 也就调用了accessDecisionManager
        InterceptorStatusToken token = super.beforeInvocation(fi);

        try {
            fi.getChain().doFilter(fi.getRequest(), fi.getResponse());
        }
```

```
            finally {
                super.finallyInvocation(token);
            }
            //完了再执行（父类的方法），一前一后，AOP无处不在啊
            super.afterInvocation(token, null);
        }
    }
```

好啦，到此应该对于Spring Security的权限管理比较清楚了。看完这个，不知你是否能扩展出一套适合自己需求的权限需求来呢，如果还不太清楚，那也没关系，下篇就实战一下，根据它来开发一套自己的权限体系。

欢迎大家访问我的独立博客：

www.javafan.cn

分类: spring security

标签: Spring Security, 权限

南轲梦
关注 - 0
粉丝 - 507
+加关注

« 上一篇： 深入springMVC源码------文件上传源码解析(下篇)
» 下一篇： 自定义Spring Security权限控制管理（实战篇）

posted on 2016-11-28 11:48 南轲梦 阅读(11045) 评论(10) 编辑 收藏

发表评论

#1楼 2016-11-28 13:32 | gowk

真帅。。不过我们用shiro

支持(0) 反对(0)

#2楼 2016-11-29 10:47 | 楚霄云

研究过一段时间，发现太复杂了，只好用shiro

支持(0) 反对(0)

#3楼 [楼主] 2016-11-29 12:54 | 南轲梦

@ gowk
嗯嗯，shiro也挺好，上手快

支持(0) 反对(0)

#4楼 [楼主] 2016-11-29 12:55 | 南轲梦

@ 楚霄云
还好，扩展性也挺好的。shiro上手是快多了，不过现在我已经习惯了spring security了

支持(0) 反对(0)

#5楼 2016-12-26 17:19 | co2y

请问博主，何时需要用到多个voter?

支持(0) 反对(0)

#6楼 [楼主] 2016-12-26 17:27 | 南轲梦

@ co2y
看具体需求哈。
spring 默认注册了RoleVoter和AuthenticatedVoter两个。
如果你还有自己的规则，那么可以根据具体情况再实现一个或者多个，然后注册到AccessDecisionManager (默认是AffirmativeBased，可根据自己具体情况进行替换AffirmativeBased的策略)

支持(0) 反对(0)

#7楼 2017-02-07 23:54 | 假程序猿

mark

支持(0) 反对(0)

#8楼 2018-08-22 16:41 | 赵孤鸿

AffirmativeBased 应该是至少一个投票者投通过票，才判为通过，其它都为不通过

支持(0) 反对(0)

#9楼 2018-08-23 14:20 | GrandKai

通过以上代码可直接看到AffirmativeBased的策略：

只要有投通过（ACCESS_GRANTED）票，则直接判为通过。
如果没有投通过票且反对（ACCESS_DENIED）票在两个及其以上的，则直接判为不通过。

这里应该是只要没有通过票，且反对票大于1票就不通过

支持(0) 反对(0)

#10楼 2018-10-29 16:38 | 更新实践

有完整的 spring 集成spring security项目吗 有偿
我急需要用 qq:1102602338

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册， 访问 网站首页。

相关博文:
· security权限管理
· Spring Security 之自定义决策管理器
· springboot+security整合（3）自定义鉴权
· 自定义Spring Security权限控制管理（实战篇）
· Spring Security（15）——权限鉴定结构
» 更多推荐...

最新 IT 新闻:
· 三星斥资34亿美元采购20台EUV光刻机 内存也要上新工艺
· 业绩下滑，赛灵思将裁员7%?
· 法庭判决苹果和博通侵犯大学专利赔偿11亿美元
· 腾讯云免费开放超算能力，加速科研团队进行病毒药物研发
· 腾讯接受 30%"苹果税"，僵局结束?
» 更多新闻...