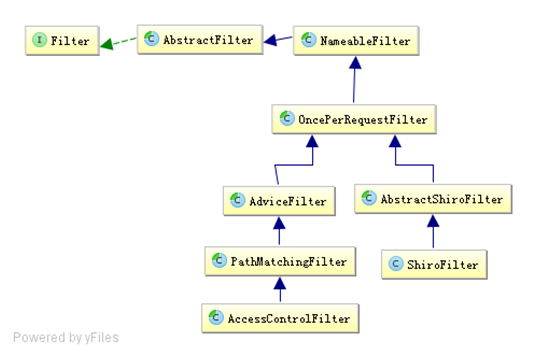
**8.1 拦截器介绍**

Shiro使用了与Servlet一样的Filter接口进行扩展；所以如果对Filter不熟悉可以参考《Servlet3.1规范》<http://www.iteye.com/blogs/subjects/Servlet-3-1>了解Filter的工作原理。首先下图是Shiro拦截器的基础类图：



**1、NameableFilter**

NameableFilter给Filter起个名字，如果没有设置默认就是FilterName；还记得之前的如authc吗？当我们组装拦截器链时会根据这个名字找到相应的拦截器实例；

**2、OncePerRequestFilter**

OncePerRequestFilter用于防止多次执行Filter的；也就是说一次请求只会走一次拦截器链；另外提供enabled属性，表示是否开启该拦截器实例，默认enabled=true表示开启，如果不想让某个拦截器工作，可以设置为false即可。

**3、ShiroFilter**

ShiroFilter是整个Shiro的入口点，用于拦截需要安全控制的请求进行处理，这个之前已经用过了。

**4、AdviceFilter**

AdviceFilter提供了AOP风格的支持，类似于SpringMVC中的Interceptor：

**Java代码  [收藏代码](javascript:void())**

1. **boolean** preHandle(ServletRequest request, ServletResponse response) **throws** Exception
2. **void** postHandle(ServletRequest request, ServletResponse response) **throws** Exception
3. **void** afterCompletion(ServletRequest request, ServletResponse response, Exception exception) **throws** Exception;

preHandler：类似于AOP中的前置增强；在拦截器链执行之前执行；如果返回true则继续拦截器链；否则中断后续的拦截器链的执行直接返回；进行预处理（如基于表单的身份验证、授权）

postHandle：类似于AOP中的后置返回增强；在拦截器链执行完成后执行；进行后处理（如记录执行时间之类的）；

afterCompletion：类似于AOP中的后置最终增强；即不管有没有异常都会执行；可以进行清理资源（如接触Subject与线程的绑定之类的）；

**5、PathMatchingFilter**

PathMatchingFilter提供了基于Ant风格的请求路径匹配功能及拦截器参数解析的功能，如“roles[admin,user]”自动根据“，”分割解析到一个路径参数配置并绑定到相应的路径：

**Java代码  [收藏代码](javascript:void())**

1. **boolean** pathsMatch(String path, ServletRequest request)
2. **boolean** onPreHandle(ServletRequest request, ServletResponse response, Object mappedValue) **throws** Exception

pathsMatch：该方法用于path与请求路径进行匹配的方法；如果匹配返回true；

onPreHandle：在preHandle中，当pathsMatch匹配一个路径后，会调用opPreHandler方法并将路径绑定参数配置传给mappedValue；然后可以在这个方法中进行一些验证（如角色授权），如果验证失败可以返回false中断流程；默认返回true；也就是说子类可以只实现onPreHandle即可，无须实现preHandle。如果没有path与请求路径匹配，默认是通过的（即preHandle返回true）。

**6、AccessControlFilter**

AccessControlFilter提供了访问控制的基础功能；比如是否允许访问/当访问拒绝时如何处理等：

**Java代码  [收藏代码](javascript:void())**

1. **abstract** **boolean** isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue) **throws** Exception;
2. **boolean** onAccessDenied(ServletRequest request, ServletResponse response, Object mappedValue) **throws** Exception;
3. **abstract** **boolean** onAccessDenied(ServletRequest request, ServletResponse response) **throws** Exception;

isAccessAllowed：表示是否允许访问；mappedValue就是[urls]配置中拦截器参数部分，如果允许访问返回true，否则false；

onAccessDenied：表示当访问拒绝时是否已经处理了；如果返回true表示需要继续处理；如果返回false表示该拦截器实例已经处理了，将直接返回即可。

onPreHandle会自动调用这两个方法决定是否继续处理：

**Java代码  [收藏代码](javascript:void())**

1. **boolean** onPreHandle(ServletRequest request, ServletResponse response, Object mappedValue) **throws** Exception {
2. **return** isAccessAllowed(request, response, mappedValue) || onAccessDenied(request, response, mappedValue);
3. }

另外AccessControlFilter还提供了如下方法用于处理如登录成功后/重定向到上一个请求：

**Java代码  [收藏代码](javascript:void())**

1. **void** setLoginUrl(String loginUrl) //身份验证时使用，默认/login.jsp
2. String getLoginUrl()
3. Subject getSubject(ServletRequest request, ServletResponse response) //获取Subject实例
4. **boolean** isLoginRequest(ServletRequest request, ServletResponse response)//当前请求是否是登录请求
5. **void** saveRequestAndRedirectToLogin(ServletRequest request, ServletResponse response) **throws** IOException //将当前请求保存起来并重定向到登录页面
6. **void** saveRequest(ServletRequest request) //将请求保存起来，如登录成功后再重定向回该请求
7. **void** redirectToLogin(ServletRequest request, ServletResponse response) //重定向到登录页面

比如基于表单的身份验证就需要使用这些功能。

到此基本的拦截器就完事了，如果我们想进行访问访问的控制就可以继承AccessControlFilter；如果我们要添加一些通用数据我们可以直接继承PathMatchingFilter。

**8.2 拦截器链**

Shiro对Servlet容器的FilterChain进行了代理，即ShiroFilter在继续Servlet容器的Filter链的执行之前，通过ProxiedFilterChain对Servlet容器的FilterChain进行了代理；即先走Shiro自己的Filter体系，然后才会委托给Servlet容器的FilterChain进行Servlet容器级别的Filter链执行；Shiro的ProxiedFilterChain执行流程：1、先执行Shiro自己的Filter链；2、再执行Servlet容器的Filter链（即原始的Filter）。

而ProxiedFilterChain是通过FilterChainResolver根据配置文件中[urls]部分是否与请求的URL是否匹配解析得到的。

**Java代码  [收藏代码](javascript:void())**

1. FilterChain getChain(ServletRequest request, ServletResponse response, FilterChain originalChain);

即传入原始的chain得到一个代理的chain。

Shiro内部提供了一个路径匹配的FilterChainResolver实现：PathMatchingFilterChainResolver，其根据[urls]中配置的url模式（默认Ant风格）=拦截器链和请求的url是否匹配来解析得到配置的拦截器链的；而PathMatchingFilterChainResolver内部通过FilterChainManager维护着拦截器链，比如DefaultFilterChainManager实现维护着url模式与拦截器链的关系。因此我们可以通过FilterChainManager进行动态动态增加url模式与拦截器链的关系。

DefaultFilterChainManager会默认添加org.apache.shiro.web.filter.mgt.DefaultFilter中声明的拦截器：

**Java代码  [收藏代码](javascript:void())**

1. **public** **enum** DefaultFilter {
2. anon(AnonymousFilter.**class**),
3. authc(FormAuthenticationFilter.**class**),
4. authcBasic(BasicHttpAuthenticationFilter.**class**),
5. logout(LogoutFilter.**class**),
6. noSessionCreation(NoSessionCreationFilter.**class**),
7. perms(PermissionsAuthorizationFilter.**class**),
8. port(PortFilter.**class**),
9. rest(HttpMethodPermissionFilter.**class**),
10. roles(RolesAuthorizationFilter.**class**),
11. ssl(SslFilter.**class**),
12. user(UserFilter.**class**);
13. }

下一节会介绍这些拦截器的作用。

如果要注册自定义拦截器，IniSecurityManagerFactory/WebIniSecurityManagerFactory在启动时会自动扫描ini配置文件中的[filters]/[main]部分并注册这些拦截器到DefaultFilterChainManager；且创建相应的url模式与其拦截器关系链。如果使用Spring后续章节会介绍如果注册自定义拦截器。

如果想自定义FilterChainResolver，可以通过实现WebEnvironment接口完成：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** MyIniWebEnvironment **extends** IniWebEnvironment {
2. @Override
3. **protected** FilterChainResolver createFilterChainResolver() {
4. //在此处扩展自己的FilterChainResolver
5. **return** **super**.createFilterChainResolver();
6. }
7. }

FilterChain之间的关系。如果想动态实现url-拦截器的注册，就可以通过实现此处的FilterChainResolver来完成，比如：

**Java代码  [收藏代码](javascript:void())**

1. //1、创建FilterChainResolver
2. PathMatchingFilterChainResolver filterChainResolver =
3. **new** PathMatchingFilterChainResolver();
4. //2、创建FilterChainManager
5. DefaultFilterChainManager filterChainManager = **new** DefaultFilterChainManager();
6. //3、注册Filter
7. **for**(DefaultFilter filter : DefaultFilter.values()) {
8. filterChainManager.addFilter(
9. filter.name(), (Filter) ClassUtils.newInstance(filter.getFilterClass()));
10. }
11. //4、注册URL-Filter的映射关系
12. filterChainManager.addToChain("/login.jsp", "authc");
13. filterChainManager.addToChain("/unauthorized.jsp", "anon");
14. filterChainManager.addToChain("/\*\*", "authc");
15. filterChainManager.addToChain("/\*\*", "roles", "admin");
17. //5、设置Filter的属性
18. FormAuthenticationFilter authcFilter =
19. (FormAuthenticationFilter)filterChainManager.getFilter("authc");
20. authcFilter.setLoginUrl("/login.jsp");
21. RolesAuthorizationFilter rolesFilter =
22. (RolesAuthorizationFilter)filterChainManager.getFilter("roles");
23. rolesFilter.setUnauthorizedUrl("/unauthorized.jsp");
25. filterChainResolver.setFilterChainManager(filterChainManager);
26. **return** filterChainResolver;

此处自己去实现注册filter，及url模式与filter之间的映射关系。可以通过定制FilterChainResolver或FilterChainManager来完成诸如动态URL匹配的实现。

然后再web.xml中进行如下配置Environment：

**Java代码  [收藏代码](javascript:void())**

1. <context-param>
2. <param-name>shiroEnvironmentClass</param-name> <param-value>com.github.zhangkaitao.shiro.chapter8.web.env.MyIniWebEnvironment</param-value>
3. </context-param>

**8.3 自定义拦截器**

通过自定义自己的拦截器可以扩展一些功能，诸如动态url-角色/权限访问控制的实现、根据Subject身份信息获取用户信息绑定到Request（即设置通用数据）、验证码验证、在线用户信息的保存等等，因为其本质就是一个Filter；所以Filter能做的它就能做。

对于Filter的介绍请参考《Servlet规范》中的Filter部分：

<http://www.iteye.com/blogs/subjects/Servlet-3-1>。

**1、扩展OncePerRequestFilter**

OncePerRequestFilter保证一次请求只调用一次doFilterInternal，即如内部的forward不会再多执行一次doFilterInternal：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** MyOncePerRequestFilter **extends** OncePerRequestFilter {
2. @Override
3. **protected** **void** doFilterInternal(ServletRequest request, ServletResponse response, FilterChain chain) **throws** ServletException, IOException {
4. System.out.println("=========once per request filter");
5. chain.doFilter(request, response);
6. }
7. }

然后再shiro.ini配置文件中：

**Java代码  [收藏代码](javascript:void())**

1. [main]
2. myFilter1=com.github.zhangkaitao.shiro.chapter8.web.filter.MyOncePerRequestFilter
3. #[filters]
4. #myFilter1=com.github.zhangkaitao.shiro.chapter8.web.filter.MyOncePerRequestFilter
5. [urls]
6. /\*\*=myFilter1

Filter可以在[main]或[filters]部分注册，然后在[urls]部分配置url与filter的映射关系即可。

**2、扩展AdviceFilter**

AdviceFilter提供了AOP的功能，其实现和SpringMVC中的Interceptor思想一样：具体可参考我的SpringMVC教程中的处理器拦截器部分：

<http://www.iteye.com/blogs/subjects/kaitao-springmvc>

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** MyAdviceFilter **extends** AdviceFilter {
2. @Override
3. **protected** **boolean** preHandle(ServletRequest request, ServletResponse response) **throws** Exception {
4. System.out.println("====预处理/前置处理");
5. **return** **true**;//返回false将中断后续拦截器链的执行
6. }
7. @Override
8. **protected** **void** postHandle(ServletRequest request, ServletResponse response) **throws** Exception {
9. System.out.println("====后处理/后置返回处理");
10. }
11. @Override
12. **public** **void** afterCompletion(ServletRequest request, ServletResponse response, Exception exception) **throws** Exception {
13. System.out.println("====完成处理/后置最终处理");
14. }
15. }

preHandle：进行请求的预处理，然后根据返回值决定是否继续处理（true：继续过滤器链）；可以通过它实现权限控制；

postHandle：执行完拦截器链之后正常返回后执行；

afterCompletion：不管最后有没有异常，afterCompletion都会执行，完成如清理资源功能。

然后在shiro.ini中进行如下配置：

**Java代码  [收藏代码](javascript:void())**

1. [filters]
2. myFilter1=com.github.zhangkaitao.shiro.chapter8.web.filter.MyOncePerRequestFilter
3. myFilter2=com.github.zhangkaitao.shiro.chapter8.web.filter.MyAdviceFilter
4. [urls]
5. /\*\*=myFilter1,myFilter2

该过滤器的具体使用可参考我的SpringMVC教程中的处理器拦截器部分。

**3、PathMatchingFilter**

PathMatchingFilter继承了AdviceFilter，提供了url模式过滤的功能，如果需要对指定的请求进行处理，可以扩展PathMatchingFilter：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** MyPathMatchingFilter **extends** PathMatchingFilter {
2. @Override
3. **protected** **boolean** onPreHandle(ServletRequest request, ServletResponse response, Object mappedValue) **throws** Exception {
4. System.out.println("url matches,config is " + Arrays.toString((String[])mappedValue));
5. **return** **true**;
6. }
7. }

preHandle：会进行url模式与请求url进行匹配，如果匹配会调用onPreHandle；如果没有配置url模式/没有url模式匹配，默认直接返回true；

onPreHandle：如果url模式与请求url匹配，那么会执行onPreHandle，并把该拦截器配置的参数传入。默认什么不处理直接返回true。

然后在shiro.ini中进行如下配置：

**Java代码  [收藏代码](javascript:void())**

1. [filters]
2. myFilter3=com.github.zhangkaitao.shiro.chapter8.web.filter.MyPathMatchingFilter
3. [urls]
4. /\*\*= myFilter3[config]

/\*\*就是注册给PathMatchingFilter的url模式，config就是拦截器的配置参数，多个之间逗号分隔，onPreHandle使用mappedValue接收参数值。

**4、扩展AccessControlFilter**

AccessControlFilter继承了PathMatchingFilter，并扩展了了两个方法：

**Java代码  [收藏代码](javascript:void())**

1. **public** **boolean** onPreHandle(ServletRequest request, ServletResponse response, Object mappedValue) **throws** Exception {
2. **return** isAccessAllowed(request, response, mappedValue)
3. || onAccessDenied(request, response, mappedValue);
4. }

isAccessAllowed：即是否允许访问，返回true表示允许；

onAccessDenied：表示访问拒绝时是否自己处理，如果返回true表示自己不处理且继续拦截器链执行，返回false表示自己已经处理了（比如重定向到另一个页面）。

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** MyAccessControlFilter **extends** AccessControlFilter {
2. **protected** **boolean** isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue) **throws** Exception {
3. System.out.println("access allowed");
4. **return** **true**;
5. }
6. **protected** **boolean** onAccessDenied(ServletRequest request, ServletResponse response) **throws** Exception {
7. System.out.println("访问拒绝也不自己处理，继续拦截器链的执行");
8. **return** **true**;
9. }
10. }

然后在shiro.ini中进行如下配置：

**Java代码  [收藏代码](javascript:void())**

1. [filters]
2. myFilter4=com.github.zhangkaitao.shiro.chapter8.web.filter.MyAccessControlFilter
3. [urls]
4. /\*\*=myFilter4

**5、基于表单登录拦截器**

之前我们已经使用过Shiro内置的基于表单登录的拦截器了，此处自己做一个类似的基于表单登录的拦截器。

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** FormLoginFilter **extends** PathMatchingFilter {
2. **private** String loginUrl = "/login.jsp";
3. **private** String successUrl = "/";
4. @Override
5. **protected** **boolean** onPreHandle(ServletRequest request, ServletResponse response, Object mappedValue) **throws** Exception {
6. **if**(SecurityUtils.getSubject().isAuthenticated()) {
7. **return** **true**;//已经登录过
8. }
9. HttpServletRequest req = (HttpServletRequest) request;
10. HttpServletResponse resp = (HttpServletResponse) response;
11. **if**(isLoginRequest(req)) {
12. **if**("post".equalsIgnoreCase(req.getMethod())) {//form表单提交
13. **boolean** loginSuccess = login(req); //登录
14. **if**(loginSuccess) {
15. **return** redirectToSuccessUrl(req, resp);
16. }
17. }
18. **return** **true**;//继续过滤器链
19. } **else** {//保存当前地址并重定向到登录界面
20. saveRequestAndRedirectToLogin(req, resp);
21. **return** **false**;
22. }
23. }
24. **private** **boolean** redirectToSuccessUrl(HttpServletRequest req, HttpServletResponse resp) **throws** IOException {
25. WebUtils.redirectToSavedRequest(req, resp, successUrl);
26. **return** **false**;
27. }
28. **private** **void** saveRequestAndRedirectToLogin(HttpServletRequest req, HttpServletResponse resp) **throws** IOException {
29. WebUtils.saveRequest(req);
30. WebUtils.issueRedirect(req, resp, loginUrl);
31. }
33. **private** **boolean** login(HttpServletRequest req) {
34. String username = req.getParameter("username");
35. String password = req.getParameter("password");
36. **try** {
37. SecurityUtils.getSubject().login(**new** UsernamePasswordToken(username, password));
38. } **catch** (Exception e) {
39. req.setAttribute("shiroLoginFailure", e.getClass());
40. **return** **false**;
41. }
42. **return** **true**;
43. }
44. **private** **boolean** isLoginRequest(HttpServletRequest req) {
45. **return** pathsMatch(loginUrl, WebUtils.getPathWithinApplication(req));
46. }
47. }

onPreHandle主要流程：

1、首先判断是否已经登录过了，如果已经登录过了继续拦截器链即可；

2、如果没有登录，看看是否是登录请求，如果是get方法的登录页面请求，则继续拦截器链（到请求页面），否则如果是get方法的其他页面请求则保存当前请求并重定向到登录页面；

3、如果是post方法的登录页面表单提交请求，则收集用户名/密码登录即可，如果失败了保存错误消息到“shiroLoginFailure”并返回到登录页面；

4、如果登录成功了，且之前有保存的请求，则重定向到之前的这个请求，否则到默认的成功页面。

shiro.ini配置

**Java代码  [收藏代码](javascript:void())**

1. [filters]
2. formLogin=com.github.zhangkaitao.shiro.chapter8.web.filter.FormLoginFilter
3. [urls]
4. /test.jsp=formLogin
5. /login.jsp=formLogin

启动服务器输入http://localhost:8080/chapter8/test.jsp测试时，会自动跳转到登录页面，登录成功后又会跳回到test.jsp页面。

此处可以通过继承AuthenticatingFilter实现，其提供了很多登录相关的基础代码。另外可以参考Shiro内嵌的FormAuthenticationFilter的源码，思路是一样的。

**6、任意角色授权拦截器**

Shiro提供roles拦截器，其验证用户拥有所有角色，没有提供验证用户拥有任意角色的拦截器。

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** AnyRolesFilter **extends** AccessControlFilter {
2. **private** String unauthorizedUrl = "/unauthorized.jsp";
3. **private** String loginUrl = "/login.jsp";
4. **protected** **boolean** isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue) **throws** Exception {
5. String[] roles = (String[])mappedValue;
6. **if**(roles == **null**) {
7. **return** **true**;//如果没有设置角色参数，默认成功
8. }
9. **for**(String role : roles) {
10. **if**(getSubject(request, response).hasRole(role)) {
11. **return** **true**;
12. }
13. }
14. **return** **false**;//跳到onAccessDenied处理
15. }
17. @Override
18. **protected** **boolean** onAccessDenied(ServletRequest request, ServletResponse response) **throws** Exception {
19. Subject subject = getSubject(request, response);
20. **if** (subject.getPrincipal() == **null**) {//表示没有登录，重定向到登录页面
21. saveRequest(request);
22. WebUtils.issueRedirect(request, response, loginUrl);
23. } **else** {
24. **if** (StringUtils.hasText(unauthorizedUrl)) {//如果有未授权页面跳转过去
25. WebUtils.issueRedirect(request, response, unauthorizedUrl);
26. } **else** {//否则返回401未授权状态码
27. WebUtils.toHttp(response).sendError(HttpServletResponse.SC\_UNAUTHORIZED);
28. }
29. }
30. **return** **false**;
31. }
32. }

流程：

1、首先判断用户有没有任意角色，如果没有返回false，将到onAccessDenied进行处理；

2、如果用户没有角色，接着判断用户有没有登录，如果没有登录先重定向到登录；

3、如果用户没有角色且设置了未授权页面（unauthorizedUrl），那么重定向到未授权页面；否则直接返回401未授权错误码。

shiro.ini配置

**Java代码  [收藏代码](javascript:void())**

1. [filters]
2. anyRoles=com.github.zhangkaitao.shiro.chapter8.web.filter.AnyRolesFilter
3. [urls]
4. /test.jsp=formLogin,anyRoles[admin,user]
5. /login.jsp=formLogin

此处可以继承AuthorizationFilter实现，其提供了授权相关的基础代码。另外可以参考Shiro内嵌的RolesAuthorizationFilter的源码，只是实现hasAllRoles逻辑。

**8.4 默认拦截器**

Shiro内置了很多默认的拦截器，比如身份验证、授权等相关的。默认拦截器可以参考org.apache.shiro.web.filter.mgt.DefaultFilter中的枚举拦截器：

|  |  |  |
| --- | --- | --- |
| 默认拦截器名 | 拦截器类 | 说明（括号里的表示默认值） |
| **身份验证相关的** |  |  |
| authc | org.apache.shiro.web.filter.authc  .FormAuthenticationFilter | 基于表单的拦截器；如“/\*\*=authc”，如果没有登录会跳到相应的登录页面登录；主要属性：usernameParam：表单提交的用户名参数名（ username）；  passwordParam：表单提交的密码参数名（password）； rememberMeParam：表单提交的密码参数名（rememberMe）；  loginUrl：登录页面地址（/login.jsp）；successUrl：登录成功后的默认重定向地址； failureKeyAttribute：登录失败后错误信息存储key（shiroLoginFailure）； |
| authcBasic | org.apache.shiro.web.filter.authc  .BasicHttpAuthenticationFilter | Basic HTTP身份验证拦截器，主要属性： applicationName：弹出登录框显示的信息（application）； |
| logout | org.apache.shiro.web.filter.authc  .LogoutFilter | 退出拦截器，主要属性：redirectUrl：退出成功后重定向的地址（/）;示例“/logout=logout” |
| user | org.apache.shiro.web.filter.authc  .UserFilter | 用户拦截器，用户已经身份验证/记住我登录的都可；示例“/\*\*=user” |
| anon | org.apache.shiro.web.filter.authc  .AnonymousFilter | 匿名拦截器，即不需要登录即可访问；一般用于静态资源过滤；示例“/static/\*\*=anon” |
| **授权相关的** |  |  |
| roles | org.apache.shiro.web.filter.authz  .RolesAuthorizationFilter | 角色授权拦截器，验证用户是否拥有所有角色；主要属性： loginUrl：登录页面地址（/login.jsp）；unauthorizedUrl：未授权后重定向的地址；示例“/admin/\*\*=roles[admin]” |
| perms | org.apache.shiro.web.filter.authz  .PermissionsAuthorizationFilter | 权限授权拦截器，验证用户是否拥有所有权限；属性和roles一样；示例“/user/\*\*=perms["user:create"]” |
| port | org.apache.shiro.web.filter.authz  .PortFilter | 端口拦截器，主要属性：port（80）：可以通过的端口；示例“/test= port[80]”，如果用户访问该页面是非80，将自动将请求端口改为80并重定向到该80端口，其他路径/参数等都一样 |
| rest | org.apache.shiro.web.filter.authz  .HttpMethodPermissionFilter | rest风格拦截器，自动根据请求方法构建权限字符串（GET=read, POST=create,PUT=update,DELETE=delete,HEAD=read,TRACE=read,OPTIONS=read, MKCOL=create）构建权限字符串；示例“/users=rest[user]”，会自动拼出“user:read,user:create,user:update,user:delete”权限字符串进行权限匹配（所有都得匹配，isPermittedAll）； |
| ssl | org.apache.shiro.web.filter.authz  .SslFilter | SSL拦截器，只有请求协议是https才能通过；否则自动跳转会https端口（443）；其他和port拦截器一样； |
| **其他** |  |  |
| noSessionCreation | org.apache.shiro.web.filter.session  .NoSessionCreationFilter | 不创建会话拦截器，调用 subject.getSession(false)不会有什么问题，但是如果 subject.getSession(true)将抛出 DisabledSessionException异常； |

另外还提供了一个org.apache.shiro.web.filter.authz.HostFilter，即主机拦截器，比如其提供了属性：authorizedIps：已授权的ip地址，deniedIps：表示拒绝的ip地址；不过目前还没有完全实现，不可用。

这些默认的拦截器会自动注册，可以直接在ini配置文件中通过“拦截器名.属性”设置其属性：

**Java代码  [收藏代码](javascript:void())**

1. perms.unauthorizedUrl=/unauthorized

另外如果某个拦截器不想使用了可以直接通过如下配置直接禁用：

**Java代码  [收藏代码](javascript:void())**

1. perms.enabled=**false**