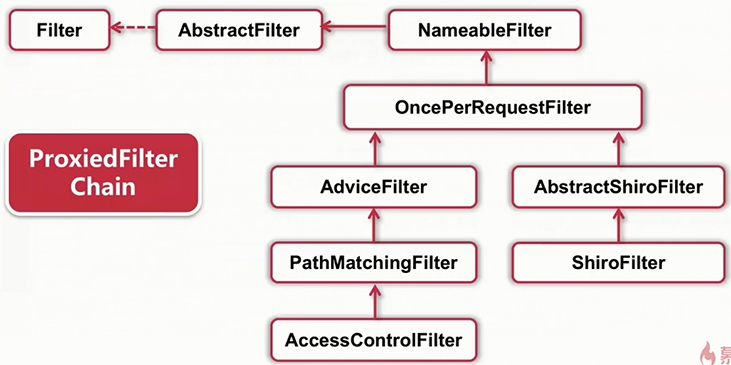
1. **拦截器介绍**



**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：

1. boolean preHandle(ServletRequest request,

ServletResponse response) throws Exception

1. void postHandle(ServletRequest request,

ServletResponse response) throws Exception

1. 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：

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：

1. abstract boolean isAccessAllowed(ServletRequest request,

ServletResponse response, Object mappedValue) throws Exception;

1. boolean onAccessDenied(ServletRequest request, ServletResponse response, Object mappedValue) throws Exception;
2. 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：

boolean onPreHandle(ServletRequest request, ServletResponse response,

Object mappedValue) throws Exception {

return isAccessAllowed(request, response, mappedValue) ||

onAccessDenied(request, response, mappedValue);

}

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：

void setLoginUrl(String loginUrl) //身份验证时使用，默认/login.jsp

String getLoginUrl()

Subject getSubject(ServletRequest request, ServletResponse response)

//获取Subject实例

boolean isLoginRequest(ServletRequest request, ServletResponse response) //当前请求是否是登录请求

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

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

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**。

1. **拦截器链**

Shiro对Servlet容器的FilterChain进行了代理，即ShiroFilter在继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。