1. Shiro拦截机制

本章主要介绍Shiro安全框架中拦截器拦截机制，关于本章，我们主要把握如下几个方面：

* Shiro安全框架中拦截器体系
* Shiro安全框架的执行流程
* 自定义拦截器

目录

[10 Shiro拦截机制 1](#_Toc495437850)

[10.1 Shiro拦截器基本类 2](#_Toc495437851)

[10.2 自定义拦截器 4](#_Toc495437852)

[10.2.1 继承OncePerRequestFilter自定义拦截器 4](#_Toc495437853)

[10.2.2 继承AdviceFilter自定义拦截器 6](#_Toc495437854)

[10.2.3 继承PathMatchingFilter自定义拦截器 7](#_Toc495437855)

[10.3 自定义拦截器实现身份验证和角色控制 8](#_Toc495437856)

[10.3.1 基于ini配置文件实现 8](#_Toc495437857)

[10.3.2 动态注册拦截器和url实现 12](#_Toc495437858)

[10.4 内置功能拦截器 14](#_Toc495437859)

[10.5 总结 14](#_Toc495437860)

通过研究Shiro同Web应用的集成，我们可以大致了解，在Shiro安全框架中，是通过自带的Filter(拦截器)实现的身份验证和授权的总入口。如果我们查看Shiro安全框架中的任何一个拦截器，比如ShiroFilter的源代码，会发现它最终实现了javax.servlet.Filter接口，这是Servlet中的过滤器接口。也就是说，Shiro里面的Filter的，本质上是Servlet技术中的Filter。只是在Shiro安全框架里面，用ProxiedFilterChain代理了Web容器中的过滤器，通过ProxiedFilterChain代理对象，控制Shiro安全框架的拦截器同Web容器中的过滤器的执行顺序：先执行完Shiro安全框架的拦截器，再执行Web容器的过滤器。

接下来我们从Shiro拦截器体系、自定义Shiro拦截器和Shiro拦截器链三个方面，研究Shiro拦截器。

* 1. Shiro拦截器基本类

在Shiro安全框架中定义了几个基本的Filter的类，这些类都是Web容器中的过滤器的子类，它们体系结构如 图10-1Shiro拦截器基本类。

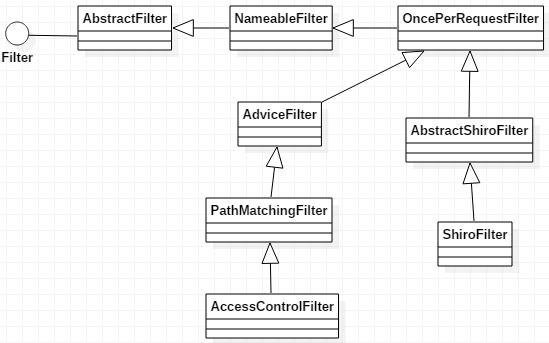


图10-1 Shiro拦截器基本类

* NameableFilter

为Filter提供一个唯一的名称，如果没有设置，就是FilterName。在Shiro安全框架的拦截器链中的每个Filter需要有一个唯一的名称，根据这些名称可以唯一找到对应的Filter对象。

* OncePerRequestFilter

保证每个Filter在一个请求过程中只会执行一次。同时提供enabled属性，通过enabled属性可以控制当前Filter是否有效（启动/关闭）。默认值为true，表示启动状态，false为关闭状态。

* ShiroFilter

ShiroFilter是整个Shiro的web入口，用来拦截所有需要安全控制的请求处理。

* AdviceFilter

AdviceFilter提供了AOP风格的支持，它提供了三个方法，分别在拦截前，拦截后和最终拦截调用。它们分别是：

boolean preHandle(ServletRequest request, ServletResponse response) throws Exception

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）

void postHandle(ServletRequest request, ServletResponse response) throws Exception

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）

void afterCompletion(ServletRequest request, ServletResponse response, Exception exception) throws Exception;

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）

* 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。它提供了两个方法：

boolean pathsMatch(String path, ServletRequest request)

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onPreHandler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）。

* 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，它提供如下三个方法：

abstract boolean isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue) throws Exception;

boolean onAccessDenied(ServletRequest request, ServletResponse response, Object mappedValue) throws Exception;

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，样例代码如下：

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。

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

* 1. 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Shiro安全框架的拦截器，本质上就是Web容器中的过滤器，所以Web容器中过滤器可以实现的功能，都可以使用Shiro安全框架的拦截器实现。

* + 1. 继承OncePerRequestFilter自定义拦截器

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，同时提供一个enabled属性，控制当前拦截器启动/关闭状态。

1. 编写MyOncePerRequestFilter代码

编写MyOncePerRequestFilter.java类，继承OncePerRequestFilter,重写doFilterInternal方法，里面就一行打印语句，样例代码块如下，详细内容请查看随书代码[code/ShiroTP10/src/main/java/cn/com/shiro/book/tp10/filter/authc/My  
OncePerRequestFilter.java](../code/ShiroTP10/src/main/java/cn/com/shiro/book/tp10/filter/authc/MyOncePerRequestFilter.java)。

public class MyOncePerRequestFilter extends OncePerRequestFilter {

@Override

protected void doFilterInternal(ServletRequest request,

ServletResponse response, FilterChain chain)

throws ServletException, IOException {

**System.out.println("执行MyOncePerRequestFilter---------");**

chain.doFilter(request, response);

}

}

1. 编写shiro\_onceFilter.ini

在resources目录中，新增shiro\_onceFilter.ini配置文件，在里面添加onceFilter拦截器的配置，样例配置如下，详细内容请查看随书代码[code/ShiroTP10/src/main/resources/shiro\_onceFilter.ini](../code/ShiroTP10/src/main/resources/shiro_onceFilter.ini)。

[filters]

myOnceFilter=cn.com.shiro.book.tp10.filter.authc.MyOncePerRequestFilter

myOnceFilter.enabled=true

#配置每个url的控制

[urls]

/\*\*=myOnceFilter

1. 配置web.xml

在web.xml中设置ShiroFilter入口、环境初始化Listener和配置文件参数，样例代码块如下，详细内容请查看随书代码[code/ShiroTP10/src/main/webapp/WEB-INF  
/web.xml](../code/ShiroTP10/src/main/webapp/WEB-INF/web.xml)。

<!-- 设置配置文件 -->

<context-param>

<param-name>shiroConfigLocations</param-name>

<param-value>classpath:shiro\_onceFilter.ini</param-value>

</context-param>

<!-- 设置环境初始化监听器 -->

<listener>

<listener-class>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-class>

</listener>

<!-- 设置ShiroFilter入口 -->

<filter>

<filter-name>ShiroFilter</filter-name>

<filter-class>org.apache.shiro.web.servlet.ShiroFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>ShiroFilter</filter-name>

<url-pattern>/\*</url-pattern>

<dispatcher>REQUEST</dispatcher>

<dispatcher>FORWARD</dispatcher>

<dispatcher>INCLUDE</dispatcher>

<dispatcher>ERROR</dispatcher>

</filter-mapping>

1. 测试

将第9章的Servlet和jsp代码移入到ShiroTP10，没发给请求，都会在控制台打印MyOncePerRequestFilter.java代码中的打印语句，而且每次只会打印一次。另外把shiro\_onceFilter.ini中的myOnceFilter.enabled改成false后，会发现myOnceFilter被禁用了。

* + 1. 继承AdviceFilter自定义拦截器

1. 编写自定义拦截器

编写MyAdviceFilter类继承AdviceFilter，重写preHandler、postHandle和afterCompletion方法，样例代码块如下，详细内容请查看随书代码[code/ShiroTP10  
/src/main/java/ cn/com/shiro/book/tp10/filter/authc/MyAdviceFilter.java](../code/ShiroTP10/src/main/java/cn/com/shiro/book/tp10/filter/authc/MyAdviceFilter.java)。

package cn.com.shiro.book.tp10.filter.authc;

… …

public class MyAdviceFilter extends AdviceFilter {

@Override

protected boolean preHandle(ServletRequest request, ServletResponse response)

throws Exception {

System.out.println("执行MyAdviceFilter->preHandle");

return super.preHandle(request, response);

}

@Override

protected void postHandle(ServletRequest request, ServletResponse response)

throws Exception {

super.postHandle(request, response);

System.out.println("执行MyAdviceFilter->postHandle");

}

@Override

public void afterCompletion(ServletRequest request,

ServletResponse response, Exception exception) throws Exception {

super.afterCompletion(request, response, exception);

System.out.println("执行MyAdviceFilter->afterCompletion");

}

}

preHandle在拦截前执行，结果返回boolean值，如果是true，继续拦截，false终止拦截。

postHandle在拦截后执行

afterCompletion拦截最终执行

1. 编写ini配合文件

在shiro\_onceFilter.ini中的[filters]部分，添加MyAdviceFilter拦截器的注册，同时在[urls]部分，对所有请求添加MyAdviceFilter拦截器的使用，样例代码块如下，详细内容请查看随书代码[code/ShiroTP10/src/main/resources/shiro\_onceFilter.ini](../code/ShiroTP10/src/main/resources/shiro_onceFilter.ini)。

[filters]

… …

myAdviceFilter=cn.com.shiro.book.tp10.filter.authc.MyAdviceFilter

#配置每个url的控制

[urls]

/\*\*=myAdviceFilter,myOnceFilter

1. 修改web.xml

web.xml同前面测试MyOncePerRequestFilter自定义拦截器的web.xml内容一样

1. 测试

发布新的web应用，基于浏览器发请求，查看控制台关于拦截器的日志输出。

注意单一个请求有定义了多个拦截器的时候，拦截的顺序有url中引用的顺序确定。

* + 1. 继承PathMatchingFilter自定义拦截器

1. 编写自定义拦截器

编写MyPathMatchingFilter类，继承PathMatchingFilter，因为PathMatchingFilter类中有实现了对Ant风格字符串的解析匹配功能和拦截器参数解析功能，通过阅读PathMatchingFilter类中的preHandle方法的源代码，我们可以理解在preHandle中已经调用pathsMatch方法完成了对请求路径的匹配，如果在匹配范围内（需要控制），就会调用onPreHandle方法进行具体的验证处理，否则就返回true，继续执行拦截器。所以我们的MyPathMatchingFilter只需要重写PathMatchingFilter类的onPreHandle方法就行，样例代码块如下，详细内容请查看随书代码[code/ShiroTP10/src/main/java/ cn/com/shiro/book/tp10/filter/authc/MyPathMatching  
Filter.java](../code/ShiroTP10/src/main/java/cn/com/shiro/book/tp10/filter/authc/MyPathMatchingFilter.java)。

public class MyPathMatchingFilter extends PathMatchingFilter {

@Override

protected boolean onPreHandle(ServletRequest request,

ServletResponse response, **Object mappedValue**) throws Exception {

System.out.println("url matches,config is " + Arrays.toString((String[])mappedValue));

return super.onPreHandle(request, response, mappedValue);

}

}

onPreHandle中的mappedValue是拦截器的配置参数

1. 编写ini配置文件

在shiro\_onceFilter.ini中的[filters]部分，添加MyPathMatchingFilter拦截器的注册，同时在[urls]部分，对所有请求添加MyPathMatchingFilter拦截器的使用，样例代码块如下，详细内容请查看随书代码[code/ShiroTP10/src/main/resources/  
shiro\_onceFilter.ini](../code/ShiroTP10/src/main/resources/shiro_onceFilter.ini)。

[filters]

myOnceFilter=cn.com.shiro.book.tp10.filter.authc.MyOncePerRequestFilter

myOnceFilter.enabled=true

myAdviceFilter=cn.com.shiro.book.tp10.filter.authc.MyAdviceFilter

myPathMatchingFilter=cn.com.shiro.book.tp10.filter.authc.MyPathMatchingFilter

#配置每个url的控制

[urls]

/\*\*=myAdviceFilter,myOnceFilter,myPathMatchingFilter**[value1,value2]**

注意粗体标识部分，是MyPathMatchingFilter拦截器的配置信息，运行的时候，在onPreHandle方法中以Object mappedValue参数传入。

1. 编写web.xml

web.xml同前面测试MyOncePerRequestFilter自定义拦截器的web.xml内容一样

1. 测试

发布新的web应用，基于浏览器发请求，查看控制台关于拦截器的日志输出。

注意单一个请求有定义了多个拦截器的时候，拦截的顺序有url中引用的顺序确定。

* 1. 自定义拦截器实现身份验证和角色控制
     1. 基于ini配置文件实现

1. 编写实现身份验证的拦截器

编写拦截器类MyFormLoginFilter，继承PathMatchingFilter，重写onPreHandle方法，眼里代码块如下，详细内容请查看随书代码[code/ShiroTP10/src/main/java/ cn/com/shiro/book/tp10/filter/authc/MyFormLoginFilter.java](../code/ShiroTP10/src/main/java/cn/com/shiro/book/tp10/filter/authc/MyFormLoginFilter.java)。

package cn.com.shiro.book.tp10.filter.authc;

… …

public class MyFormLoginFilter extends PathMatchingFilter {

private String loginUrl = "/index.jsp";

private String successUrl = "/welcome.jsp";

@Override

protected boolean onPreHandle(ServletRequest request,

ServletResponse response, Object mappedValue) throws Exception {

boolean result = false;

Subject subject = SecurityUtils.getSubject();

// 判断subject是否有验证

if (subject.isAuthenticated()) {

result = true;// 有验证，直接通过

} else {

// 判断是否是登录请求

if (this.pathsMatch(loginUrl, request)) {

// 是登录请求

HttpServletRequest req = (HttpServletRequest) request;

HttpServletResponse res = (HttpServletResponse) response;

if ("post".equalsIgnoreCase(req.getMethod())) {

String userName = req.getParameter("userName");

String password = req.getParameter("password");

UsernamePasswordToken token = new UsernamePasswordToken(

userName, password);

try {

subject.login(token);

// 保存重定向到成功页面的url

WebUtils.redirectToSavedRequest(req, res, successUrl);

result = false;

} catch (Exception e) {

req.setAttribute("shiroLoginFailure",

"登录失败:" + e.getMessage());

// 保存当前请求

WebUtils.saveRequest(request);

// 保存重定向登录页面的url

WebUtils.issueRedirect(request, response, loginUrl);

}

} else {

result = true;

}

} else {

// 保存当前请求

WebUtils.saveRequest(request);

// 保存重定向登录页面的url

WebUtils.issueRedirect(request, response, loginUrl);

}

}

return result;

}

public void setLoginUrl(String loginUrl) {

this.loginUrl = loginUrl;

}

public void setSuccessUrl(String successUrl) {

this.successUrl = successUrl;

}

}

如上代码的流程是：

1. 获取当前请求的Subject
2. 判断Subject是否有登录
3. 如果有登录，直接返回true，继续拦截流程
4. 如果没有登录

获取请求的方法

如果不是post请求，保存请求，并且设置重定向url，返回false

如果是post请求，获取用户名和密码，验证登录。如果登录成功，保存请求，设置登录成功重定向url，返回false；如果登录失败，保存请求和错误信息，设置登录失败重定向url，返回false。

1. 编写实现角色控制的拦截器

编写MyRoleAccessControlFilter类继承AccessControlFilter，重写isAccessAllowed和onAccessDenied两个方法。isAccessAllowed方法返回true表示允许访问，返回false表示不允许，交给onAccessDenied等后面流程处理。onAccessDenied表示被拒绝访问后的逻辑是否处理好了，如果返回true，表示处理好，不要继续拦截流程，否则表示没有处理好，需要继续后面拦截流程。

样例代码块如下，详细内容请查看随书代码[code/ShiroTP10/src/main/java/ cn/com/shiro/book/tp10/filter/authc/MyRoleAccessControlFilter.java](file:///C:\work\课件\shiro\shiro应用实战\code\ShiroTP10\src\main\java\cn\com\shiro\book\tp10\filter\authc\MyFormLoginFilter.java)。

package cn.com.shiro.book.tp10.filter.authc;

… …

public class MyRoleAccessControlFilter extends AccessControlFilter {

private String unauthorizedUrl = "/unauthorized.jsp";

private String loginUrl = "/login";

@Override

protected boolean isAccessAllowed(ServletRequest request,

ServletResponse response, Object mappedValue) throws Exception {

boolean result = false;

String[] roles = (String[]) mappedValue;

if (roles == null) {

result = true;

} else {

Subject subject = this.getSubject(request, response);

for (String role : roles) {

if (subject.hasRole(role)) {

result = true;

break;

}

}

}

return result;

}

@Override

protected boolean onAccessDenied(ServletRequest request,

ServletResponse response) throws Exception {

boolean result = false;

Subject subject = getSubject(request, response);

if (subject.getPrincipal() == null) {// 表示没有登录，重定向到登录页面

saveRequest(request);

WebUtils.issueRedirect(request, response, loginUrl);

} else {

if (StringUtils.hasText(unauthorizedUrl)) {// 如果有未授权页面跳转过去

WebUtils.issueRedirect(request, response, unauthorizedUrl);

} else {// 否则返回401未授权状态码

WebUtils.toHttp(response).sendError(

HttpServletResponse.SC\_UNAUTHORIZED);

}

}

return result;

}

public void setUnauthorizedUrl(String unauthorizedUrl) {

this.unauthorizedUrl = unauthorizedUrl;

}

public void setLoginUrl(String loginUrl) {

this.loginUrl = loginUrl;

}

}

1. 编写ini配置文件

在resources目录下创建shiro\_authczFilter.ini，在里面配置好[users][roles][filters] [urls]等部分，样例代码块如下，详细内容请查看随书代码code/ShiroTP10/src/main  
/resources/shiro\_authczFilter.ini。

[users]

zhangsan=123456,admin

lisi=123456,role1

wangwu=123456,role2

[roles]

admin=\*

role1=user:create,update,delete

role2=user:view

#配置filters

[filters]

logout.redirectUrl=/login

myLogin=cn.com.shiro.book.tp10.filter.authc.MyFormLoginFilter

myLogin.loginUrl=/login

myLogin.successUrl=/welcome.jsp

roleAccessFilter=cn.com.shiro.book.tp10.filter.authc.MyRoleAccessControlFilter

roleAccessFilter.unauthorizedUrl=/unauthorized.jsp

roleAccessFilter.loginUrl=/login

#配置每个url的控制

[urls]

/index.jsp=anon

/login=myLogin

/logout=logout

/admin/\*\*=myLogin,roleAccessFilter[admin]

/user/\*\*=myLogin, roleAccessFilter[role1]

/\*\*=myLogin

1. 编写web.xml

把web.xml中的shiroConfigLocations的值改成classpath:shiro\_authczFilter.ini，同时注释掉shiroEnvironmentClass参数的配置(因为同10.3.2中使用的是同一个web.xml，注意避免冲突)。

1. 测试

重新发布web应用，在浏览器中发请求测试身份验证和角色控制功能。

* + 1. 动态注册拦截器和url实现
       1. 编写动态初始化Filter环境类

编写MyIniWebEvironment类继承IniWebEviroment，重写createFilterChainResolver方法，完成拦截器链中默认拦截器、自定义拦截器和url同拦截器映射的动态配置，样例代码块如下，详细内容请查看随书代码[code/ShiroTP10/src/main/java/cn  
/com/shiro/book/tp10/web/env/MyIniWebEvironment.java](../code/ShiroTP10/src/main/java/cn/com/shiro/book/tp10/web/env/MyIniWebEnviroment.java)。

package cn.com.shiro.book.tp10.web.env;

… …

public class MyIniWebEnviroment extends IniWebEnvironment {

@Override

protected FilterChainResolver createFilterChainResolver() {

System.out.println("\*\*\*\*\*\*\*\*\*\*MyIniWebEnviroment\*\*\*\*\*\*");

// 创建FilterChainResolver

PathMatchingFilterChainResolver chainResolver = new PathMatchingFilterChainResolver();

// 创建FilterChainManager

DefaultFilterChainManager filterChainManager = new DefaultFilterChainManager();

// 注册默认的Filter

for (DefaultFilter filter : DefaultFilter.values()) {

filterChainManager.addFilter(filter.name(),

(Filter) ClassUtils.newInstance(filter.getFilterClass()));

}

// 注册自定义Filters

filterChainManager.addFilter("myLogin", (Filter)ClassUtils.newInstance(MyFormLoginFilter.class));

filterChainManager.addFilter("roleAccessFilter", (Filter)ClassUtils.newInstance(MyRoleAccessControlFilter.class));

// 设置Filter的属性

LogoutFilter logout = (LogoutFilter)filterChainManager.getFilter("logout");

logout.setRedirectUrl("/login");

MyFormLoginFilter myLogin = (MyFormLoginFilter)filterChainManager.getFilter("myLogin");

myLogin.setLoginUrl("/login");

myLogin.setSuccessUrl("/welcome.jsp");

MyRoleAccessControlFilter roleAccessFilter = (MyRoleAccessControlFilter)filterChainManager.getFilter("roleAccessFilter");

roleAccessFilter.setLoginUrl("/login");

roleAccessFilter.setUnauthorizedUrl("/unauthorized.jsp");

// 添加url和Filter的映射

filterChainManager.addToChain("/index.jsp", "anon");

filterChainManager.addToChain("/login", "myLogin");

filterChainManager.addToChain("/logout", "logout");

filterChainManager.addToChain("/admin/\*\*", "roleAccessFilter","admin");

filterChainManager.addToChain("/user/\*\*", "roleAccessFilter","role1,admin");

filterChainManager.addToChain("/\*\*", "myLogin");

chainResolver.setFilterChainManager(filterChainManager);

return chainResolver;

}

}

* + - 1. 编写ini配置文件

编写shiro\_myWebEvn.ini配置文件，里面内容在shiro\_authczFilter.ini的基础上，删除了[filters]和[urls]部分，这两部分的信息在代码中动态初始化，样例代码块如下，详细内容请查看随书代码[code/ShiroTP10/src/main/resources/shiro\_myWebEvn  
.ini](../code/ShiroTP10/src/main/resources/shiro_myWebEvn.ini)。

[users]

zhangsan=123456,admin

lisi=123456,role1

wangwu=123456,role2

[roles]

admin=\*

role1=user:create,update,delete

role2=user:view

* + - 1. 测试

同10.3.1一样测试，效果一样

* 1. 内置功能拦截器

请参考9.1小节

* 1. 总结

本章主要介绍了Shiro安全框架中自带的拦截器基础类，它们分别是NameableFilter、OncePerRequestFilter、AdviceFilter、PathMatchingFilter、AccessControlFilter、ShiroFilter，并且各自有自己的特色功能实现，我们可以在这些基类的基础上扩展实现自己的功能。同时还介绍了Shiro安全框架中的拦截器链机制。在Shiro中的拦截器，本质是过滤器，只是在Shiro中使用了ProxiedFilterChain代理了容器中的过滤器，控制了Shiro拦截器和容器中过滤的执行顺序。最后介绍了用代码代替ini配置文件动态初始化拦截器相关信息。