Shiro框架使用介绍

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本 | 作者 | 备注 |
| 2017-10-19 | 1.0 | 刘鋆 | 初稿 |

目录

[前言 3](#_Toc496276541)

[初级部分 4](#_Toc496276542)

[一、 配置 4](#_Toc496276543)

[1.1 Web.xml 4](#_Toc496276544)

[1.2 Spring 配置 4](#_Toc496276545)

[1.3 实现realm 6](#_Toc496276546)

[1.4 login 页面 7](#_Toc496276547)

[二、 拦截过程 8](#_Toc496276548)

[2.1 Filter 介绍： 8](#_Toc496276549)

[2.2 拦截过程 9](#_Toc496276550)

[三、 Subject对象 10](#_Toc496276551)

[高级部分 12](#_Toc496276552)

[四、 Cryptography 密码 12](#_Toc496276553)

[五、 使用 redis 保存 session 13](#_Toc496276554)

[六、 使用 cache 14](#_Toc496276555)

[6.1 使用ehcache 14](#_Toc496276556)

[6.2 使用redis 15](#_Toc496276557)

[6.3 缓存信息的删除 16](#_Toc496276558)

[七、 多 realm 设置 16](#_Toc496276559)

[7.1 配置 16](#_Toc496276560)

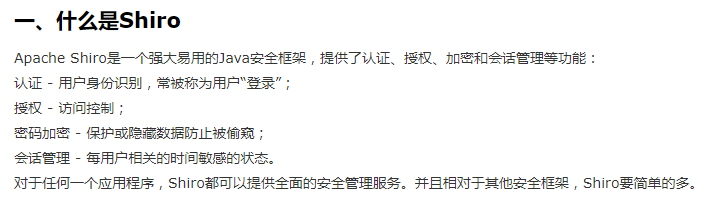
[7.2 认证策略 AuthenticationStrategy 17](#_Toc496276561)

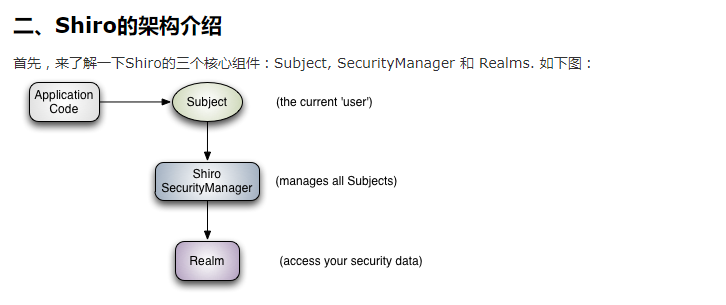
[7.2.1 介绍 17](#_Toc496276562)

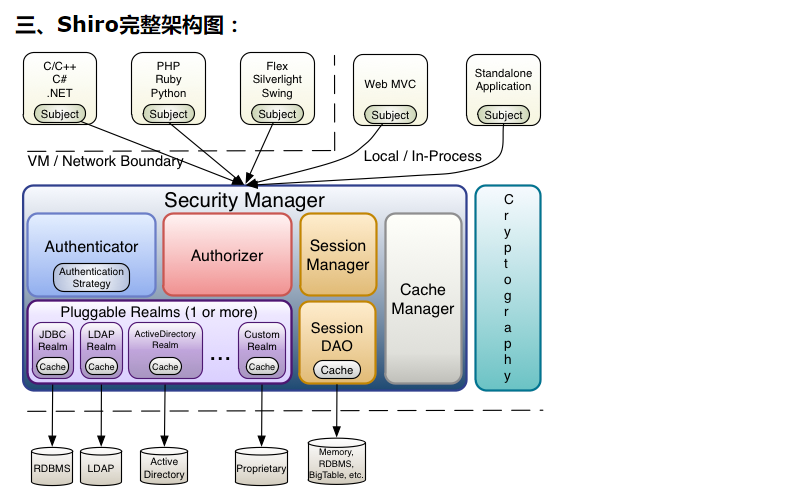
[7.2.2 配置 18](#_Toc496276563)

# 前言

先从网上贴一段shiro的介绍：







Authenticator ： 认证

Authorizer ： 授权

本文主要对shiro与spring mvc的结合做大概介绍，未提及部分需要各位童鞋自行搜索学习。

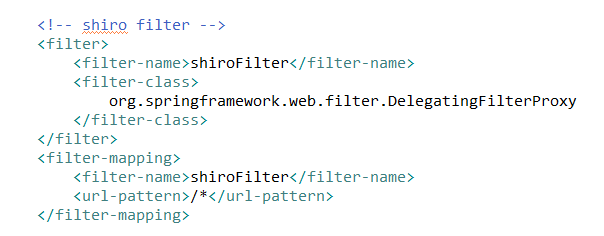
# 初级部分

# 一、 配置

一般情况下，需要在两个位置增加 shiro 的相关配置。

## 1.1 Web.xml

在 web.xml 中，增加shiro的filter：



一般来说，增加filter都是在web.xml中进行配置，而且filter是有顺序的。

这个filter-class使用的是spring框架中的DelegatingFilterProxy，而不是shiro自己的filter。

**DelegatingFilterProxy** 是对于 servlet filter 的代理，优点为：

1） 通过 spring 容器来管理 servlet filter 的生命周期。

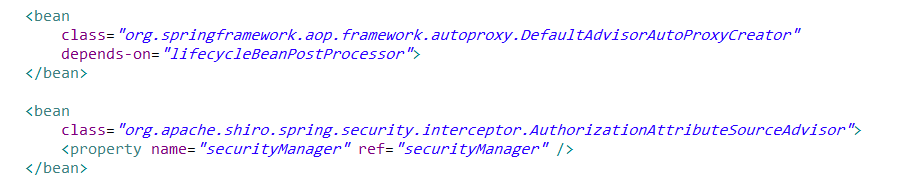
2） 如果 filter 中需要一些 Spring 容器的实例，可以通过 spring 直接注入。

## 1.2 Spring 配置

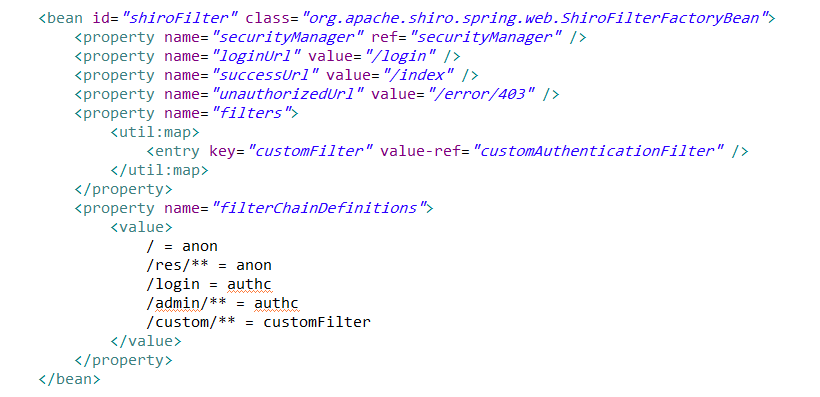
在spring 配置文件中增加 shiro 相应bean的定义（这里只列出几个常用的，更多的配置请自行搜索）：

公用配置：

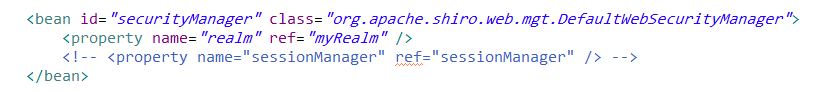




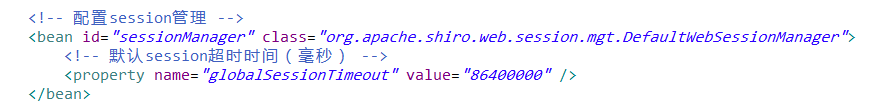
shiroFilter：



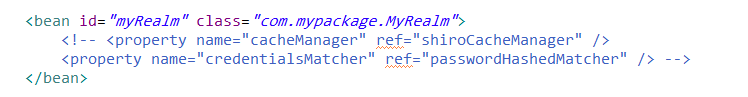
securityManager：



sessionManager ：



Realm ：

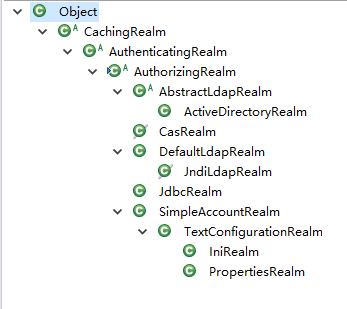


## 1.3 实现realm

主要实现 doGetAuthenticationInfo 获取认证信息接口和 doGetAuthorizationInfo 获取授权信息接口。



Shiro 已经定义了一些 Realm， 合理地利用可以达到事半功倍的效果。如下图：



比如说，如果是从 ldap 中去认证用户的话，可以使用或者继承DefaultLdapRealm，减少写代码的工作。

## 1.4 login 页面

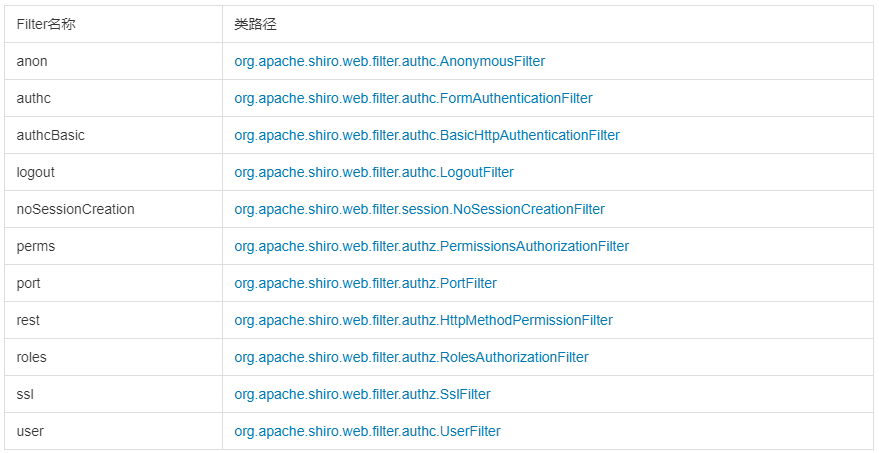
示例：



# 二、 拦截过程

## 2.1 Filter 介绍：

默认 filter 列表：



Filter 详细说明：

|  |  |
| --- | --- |
| 默认拦截器名 | 说明（括号里的表示默认值） |
| **身份验证** |  |
| anon | 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” |
| authc | 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 | 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 | 退出拦截器，示例“/logout=logout”  主要属性：  redirectUrl：退出成功后重定向的地址（/）; |
| user | 用户拦截器，用户已经身份验证/记住我登录的都可；示例“/\*\*=user” |
| **授权相关的** |  |
| roles | 角色授权拦截器，验证用户是否拥有所有角色；示例“/admin/\*\*=roles[admin]”  主要属性：  loginUrl：登录页面地址（/login.jsp）；  unauthorizedUrl：未授权后重定向的地址； |
| perms | 权限授权拦截器，验证用户是否拥有所有权限；示例“/user/\*\*=perms["user:create"]”  主要属性：  属性和roles一样； |
| port | 端口拦截器。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  主要属性：  port：可以通过的端口（80）； |
| rest | 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 | 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 | 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； |

## 2.2 拦截过程

只列出几个常用的拦截器的拦截过程：

authc：



logout：



roles、 perms：



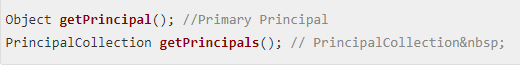
# 三、 Subject对象

Subject 是 shiro 的一个核心组件，我们可以通过一个静态方法获取到Subject对象。

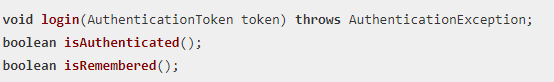


基本所有身份验证、授权都是通过 Subject 完成。

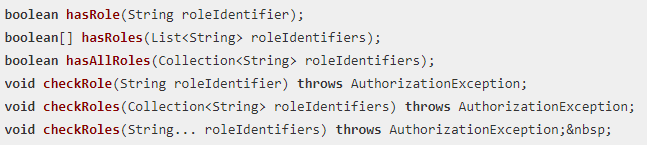
1. **身份信息获取**



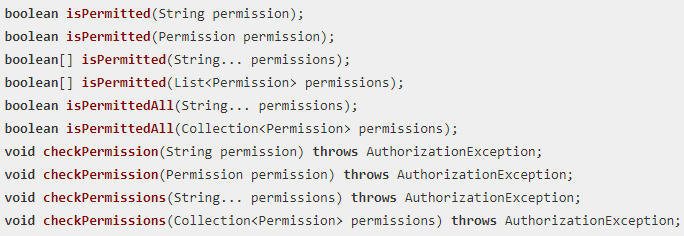
1. **身份验证**



1. **角色授权验证**



1. **权限授权验证**



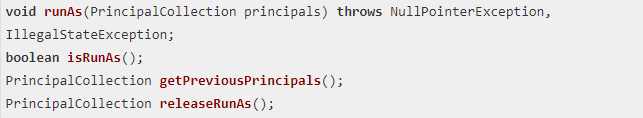
1. **会话**



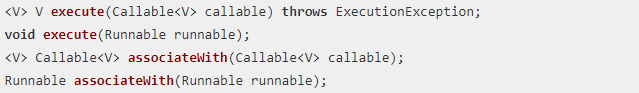
1. **退出**



1. RunAs



1. 多线程



# 高级部分

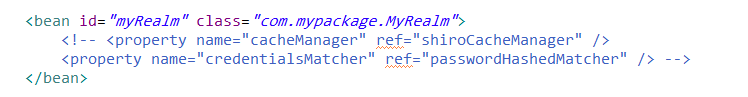
# 四、 Cryptography 密码

一般来说，数据库中的密码都是加密存储的。登录时要判断密码是否正确，就需要把提交表单的密码进行加密再跟数据库中的密码密文进行对比。

对比的过程，我们当然可以自己手动去做。其实也可以让shiro框架去做这个事情。

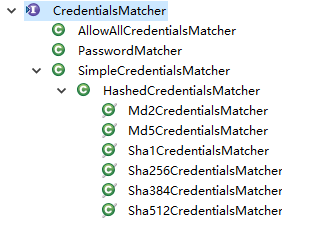
重新查看上面1.3 实现realm 一节，你会发现认证的实现代码中，仅仅是通过用户名去获取用户信息，就传递了下去，而没有进行密码验证。那么它是怎么验证的？

再贴一下上面1.2节中 realm 的定义：



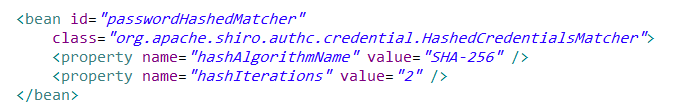
注释中有一个 credentialsMatcher， 这个就是定义密码匹配规则的接口。

Shiro 已实现的接口：

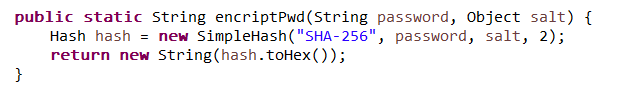


通过合理地运用这些接口，密码验证就可以做得比较简单了。

这里给出招聘系统的配置示例：



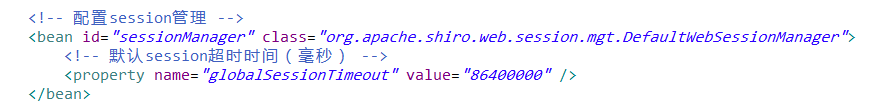
加密时，使用如下方法去加密：



其中第一个参数与最后一个参数需要与Matcher中的配置保持一致。

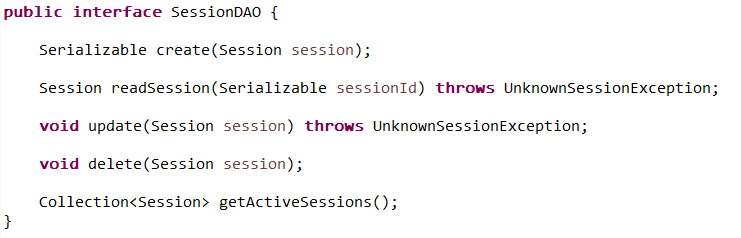
# 五、 使用 redis 保存 session

回顾shiro的配置，其中有一个sessionManager，如下：

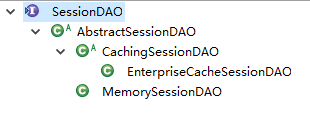


查看源码可以发现，shiro定义了一个名为SessionDao的接口来存储和获取session。

SessionDAO 接口定义如下：

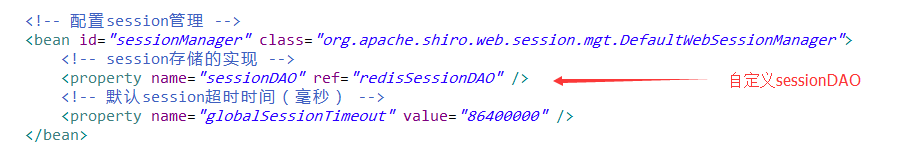


Shiro 中已经实现的类：



这里shiro默认使用的是MemorySessionDAO。

因此我们可以自己写一个实现类，继承AbstractSessionDAO，把session存到其他地方（比如redis）中。然后在sessionManager中增加如下配置：



**注意：**

RedisSessionDAO 代码暂时不在这个文档中贴出，有需要的童鞋可以从网上、ERP系统源码及招聘系统源码中获取。

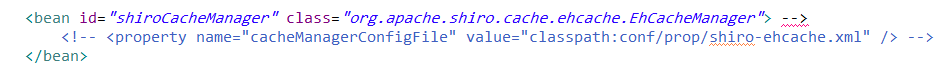
# 六、 使用 cache

## 6.1 使用ehcache

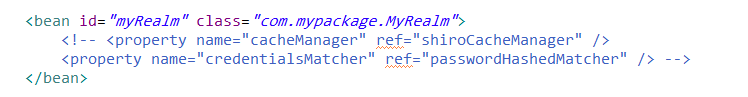
shiro官方已经实现了ehcache的集成代码，提供了一个额外的jar包：shiro-ehcache。

我们要做的，仅仅是把它引入进来。

同时修改spring配置，增加：



在 realm 中增加cacheManager的设置，如下图的注释部分：



## 6.2 使用redis

Shiro 有三个重要的缓存接口：

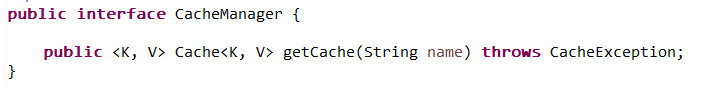
**CacheManager** - 负责所有缓存的主要管理组件，它返回 Cache 实例。

**Cache** - 维护key/value 对。

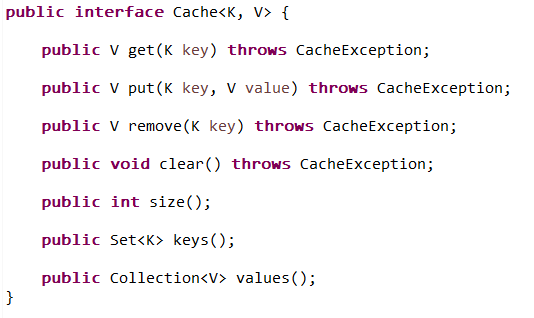
**CacheManagerAware** - 通过想要接收和使用 CacheManager 实例的组件来实现。

通过实现前两个接口，用户就可以把任何第三方的缓存产品集成到shiro中。

CacheManager 接口：



Cache 接口：



这里也不贴出相应的实现代码。需要的可以从网上或者招聘系统中获取。

## 6.3 缓存信息的删除

Realm中提供了一个 clearCache 方法用于清除指定登录者的缓存信息。

不足的是，该方法只能清除特定的人。

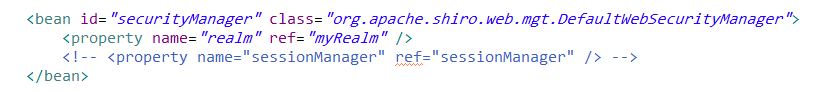
因此，需要自行实现一个清除所有的缓存信息的方法。示例如下：



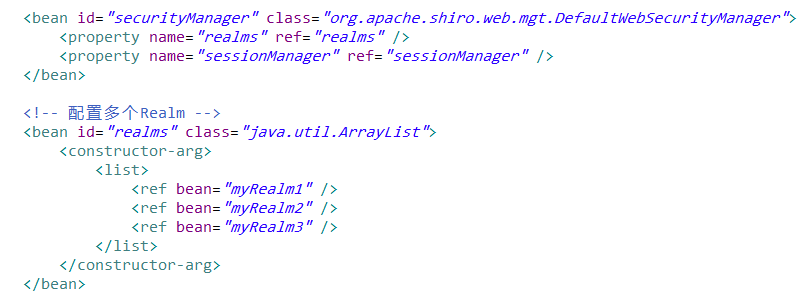
# 七、 多 realm 设置

## 7.1 配置

单一realm的配置如下图，直接把自己实现的realm赋值到securityManager的realm属性中：



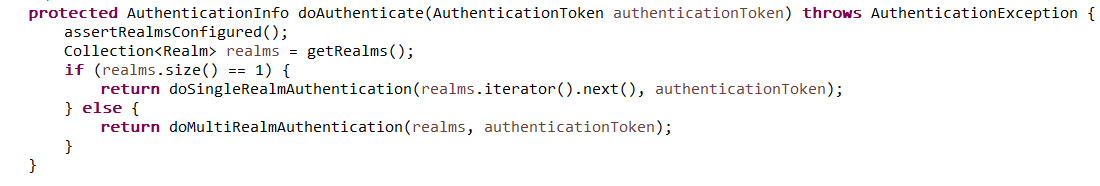
多个realm的配置示例如下，此时用到的是另一个realms 属性，需要传入一个realm列表：



## 认证策略 AuthenticationStrategy

### 7.2.1 介绍

在认证的时候，单个realm和多个realm会走不同的逻辑：



单个的情况比较简单，认证成功了就是成功了，失败就是失败，认证中的异常直接抛出。

而多个realm时，就涉及到全通过、部分通过、全不通过等不同情形。认证策略就是用于解决这个问题的。

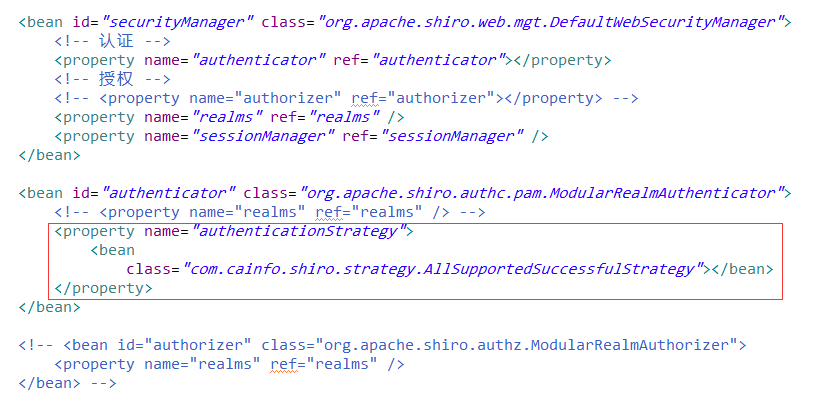
Shiro默认提供了三个认证策略：

|  |  |
| --- | --- |
| AuthenticationStrategy 类 | 说明 |
| AtLeastOneSuccessfulStrategy | 默认的认证策略。  一个realm或多个realm认证通过，则整体身份认证就会视为成功。返回的认证信息是多个通过的集合。 |
| FirstSuccessfulStrategy | 最终认证信息只取第一个认证通过的，其他的会被忽略。 |
| AllSuccessfulStrategy | 只有所有的Realm认证成功，才会被视为认证通过 |

需要注意的是，在多个realm的情形下，单个realm抛出的异常会被内部消化掉，从而无法在外层获取。如果有这类需求，需要编写自定义的认证策略。

### 7.2.2 配置

还是securityManager，看下图：



实际上 securityManager 里面有一个认证器(authenticator)和授权器(authorizer)，realms 的设置既可以在securityManager中，也可以分散设置。

注意： securityManager 的关联设置比较多，如果自定义的地方比较多，则需要正确地设置各属性的顺序。