# 1.外部架构

## IMG_256

可以看到：应用代码直接交互的对象是Subject，也就是说Shiro的对外API核心就是Subject；其每个API的含义：

****Subject：****主体，代表了当前“用户”，这个用户不一定是一个具体的人，与当前应用交互的任何东西都是Subject，如网络爬虫，机器人等；即一个抽象概念；所有Subject都绑定到SecurityManager，与Subject的所有交互都会委托给SecurityManager；可以把Subject认为是一个门面；SecurityManager才是实际的执行者；

## ****SecurityManager：****安全管理器；即所有与安全有关的操作都会与SecurityManager交互；且它管理着所有Subject；可以看出它是Shiro的核心，它负责与后边介绍的其他组件进行交互，如果学习过SpringMVC，你可以把它看成DispatcherServlet前端控制器；

## ****Realm：****域，Shiro从从Realm获取安全数据（如用户、角色、权限），就是说SecurityManager要验证用户身份，那么它需要从Realm获取相应的用户进行比较以确定用户身份是否合法；也需要从Realm得到用户相应的角色/权限进行验证用户是否能进行操作；可以把Realm看成DataSource，即安全数据

# 内部架构



****Subject：****主体，可以看到主体可以是任何可以与应用交互的“用户”；

****SecurityManager：****相当于SpringMVC中的DispatcherServlet或者Struts2中的FilterDispatcher；是Shiro的心脏；所有具体的交互都通过SecurityManager进行控制；它管理着所有Subject、且负责进行认证和授权、及会话、缓存的管理。它管理着上图内的所有组件。

****Authenticator：****认证器，负责主体认证的，这是一个扩展点，如果用户觉得Shiro默认的不好，可以自定义实现；其需要认证策略（Authentication Strategy），即什么情况下算用户认证通过了；

****Authrizer：****授权器，或者访问控制器，用来决定主体是否有权限进行相应的操作；即控制着用户能访问应用中的哪些功能；

****Realm：****可以有1个或多个Realm，可以认为是安全实体数据源，即用于获取安全实体的；可以是JDBC实现，也可以是LDAP实现，或者内存实现等等；由用户提供；注意：Shiro不知道你的用户/权限存储在哪及以何种格式存储；所以我们一般在应用中都需要实现自己的Realm；

## 3.身份认证流程

## IMG_256

1. 首先调用Subject.login(token)进行登录，其会自动委托给Security Manager，调用之前必须通过SecurityUtils. setSecurityManager()设置；

2、SecurityManager负责真正的身份验证逻辑；它会委托给Authenticator进行身份验证；

3、Authenticator才是真正的身份验证者，Shiro API中核心的身份认证入口点，此处可以自定义插入自己的实现；

4、Authenticator可能会委托给相应的AuthenticationStrategy进行多Realm身份验证，默认ModularRealmAuthenticator会调用AuthenticationStrategy进行多Realm身份验证；

5、Authenticator会把相应的token传入Realm，从Realm获取身份验证信息，如果没有返回/抛出异常表示身份验证失败了。此处可以配置多个Realm，将按照相应的顺序及策略进行访问。

即各个组件的执行顺序

Subject→Security Manager→Authenticator→AuthenticationStrategy→Realm

## 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； |  |  |