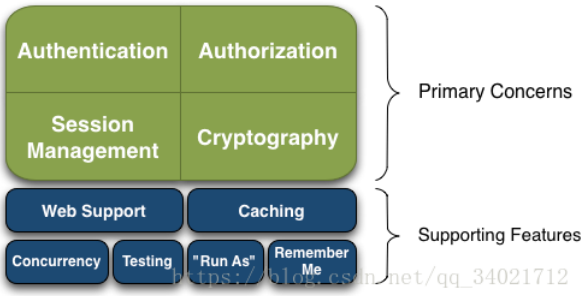
springboot整合shiro –shiro介绍(一)

shiro介绍收集于网络 ©王赛超

shiro 介绍

Shiro是Apache下的一个开源项目，我们称之为Apache Shiro。它是一个很易用与Java项目的的安全框架，提供了认证、授权、加密、会话管理，与Spring Security 一样都是做一个权限的安全框架，但是与Spring Security 相比，在于 Shiro 使用了比较简单易懂易于使用的授权方式。shiro属于轻量级框架，相对于security简单的多，也没有security那么复杂。所以我这里也是简单介绍一下shiro的使用。

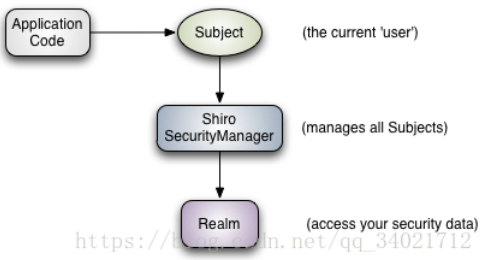
Shiro的架构



Authentication（认证）, Authorization（授权）, Session Management（会话管理）, Cryptography（加密）被 Shiro 框架的开发团队称之为应用安全的四大基石。
- **Authentication**（认证）：用户身份识别，通常被称为用户“登录”
- **Authorization**（授权）：访问控制。比如某个用户是否具有某个操作的使用权限。
- **Session Management**（会话管理）：特定于用户的会话管理,甚至在非web 或 EJB 应用程序。
- **Cryptography**（加密）：在对数据源使用加密算法加密的同时，保证易于使用。

同时Shiro还提供了其他特性来在不同的应用程序环境下使用强化以上的四大基石：
- **Web支持**：Shiro 提供的 web 支持 api ，可以很轻松的保护 web 应用程序的安全。
- **缓存**：缓存是 Apache Shiro 保证安全操作快速、高效的重要手段。
- **并发**：Apache Shiro 支持多线程应用程序的并发特性。
- **测试**：支持单元测试和集成测试，确保代码和预想的一样安全。
- **“Run As”**：这个功能允许用户假设另一个用户的身份(在许可的前提下)。
- **“Remember Me”**：跨 session 记录用户的身份，只有在强制需要时才需要登录。

Shiro 的三大核心组件



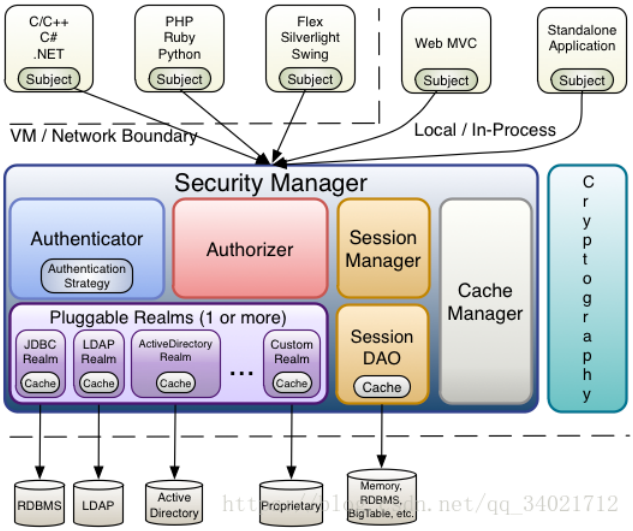
- **Subject**其实代表的就是当前正在执行操作的用户，只不过因为“User”一般指代人，但是一个“Subject”可以是人，也可以是任何的第三方系统，服务账号等任何其他正在和当前系统交互的第三方软件系统。所有的Subject实例都被绑定到一个SecurityManager，如果你和一个Subject交互，所有的交互动作都会被转换成Subject与SecurityManager的交互。

- **SecurityManager** 是Shiro的核心，用于管理所有的Subject ，它主要用于协调Shiro内部各种安全组件，不过我们一般不用太关心SecurityManager，对于应用程序开发者来说，主要还是使用Subject的API来处理各种安全验证逻辑。

- **Realm** 这是用于连接Shiro和客户系统的用户数据的桥梁。一旦Shiro真正需要访问各种安全相关的数据（比如使用用户账户来做用户身份验证以及权限验证）时，他总是通过调用系统配置的各种Realm来读取数据。

正因为如此，Realm往往被看做是安全领域的DAO，他封装了数据源连接相关的细节，将数据以Shiro需要的格式提供给Shiro。当我们配置Shiro的时候，我们至少需要配置一个Realm来提供用户验证和权限控制方面的数据。我们可能会给SecurityManager配置多个Realm，但是不管怎样，我们至少需要配置一个。
Shiro提供了几种开箱即用的Realm来访问安全数据源，比如LDAP、关系数据库、基于ini的安全配置文件等等，如果默认提供的这几种Realm无法满足你的需求，那么你也可以编写自己的定制化的Realm插件。
和其他内部组件一样，SecurityManager决定了在Shiro中如何使用Realm来读取身份和权限方面的数据，然后组装成Subject实例。

Shiro完整架构



● **Subject** (org.apache.shiro.subject.Subject)，如上所述.
● **SecurityManager** (org.apache.shiro.mgt.SecurityManager)，如上所述.
● **Authenticator**（用户认证管理器），(org.apache.shiro.authc.Authenticator) 这个组件主要用于处理用户登录逻辑，他通过调用Realm的接口来判断当前登录的用户的身份。
用户认证策略，(org.apache.shiro.authc.pam.AuthenticationStrategy) 如果系统配置了多个Realm，则需要使用AuthenticationStrategy 来协调这些Realm以便决定一个用户登录的认证是成功还是失败。(比如，如果一个Realm验证成功了，但是其他的都失败了，怎么算？还是说都成功才算成功).
● **Authorizer**（权限管理器）(org.apache.shiro.authz.Authorizer)这个组件主要是用来做用户的访问控制。通俗来说就是决定用户能做什么、不能做什么。和Authenticator类似，Authorizer也知道怎么协调多个Realm数据源的数据，他有自己的一套策略。
● **SessionManager**（会话管理器）(org.apache.shiro.session.mgt.SessionManager) SessionManager知道如何创建会话、管理用户回话的声明周期以便在所有运行环境下都可以给用户提供一个健壮的回话管理体验。Shiro在任何运行环境下都可以在本地管理用户会话（即便没有Web或者EJB容器也可以）——这在安全管理的框架中算是独门绝技了。当然，如果当前环境中有会话管理机制（比如Servlet容器），则Shiro默认会使用该环境的会话管理机制。而如果像控制台程序这种独立的应用程序，本身没有会话管理机制，此时Shiro就会使用内部的会话管理器来给应用的开发提供一直的编程体验。SessionDAO允许用户使用任何类型的数据源来存储Session数据。
● **SessionDAO** (org.apache.shiro.session.mgt.eis.SessionDAO) 用于代替SessionManager执行Session相关的增删改查。这个接口允许我们将任意种类的数据存储方式引入到Session管理的基础框架中。
● **CacheManager** (org.apache.shiro.cache.CacheManager) CacheManager用于创建和维护一些在其他的Shiro组件中用到的Cache实例，维护这些Cache实例的生命周期。缓存用于存储那些从后端获取到的用户验证与权限控制方面的数据以提高性能，缓存是一等公民，在获取数据时，总是先从缓存中查找，如果没有再调用后端接口从其他数据源获取。Shiro允许用户使用其他更加现代的、企业级的数据源来替代内部的默认实现，以提供更高的性能和更好的用户体验。
● **Cryptography** 加密技术，(org.apache.shiro.crypto.*) 对于一个企业级的安全框架来说，加密算是其固有的一种特性。Shiro的crypto包中包含了一系列的易于理解和使用的加密、哈希（aka摘要）辅助类。这个包内的所有类都是经过精心设计，相比于java本身提供的那一套反人类的加密组件，Shiro提供的这套加密组件简直不要好用太多。
● **Realm** (org.apache.shiro.realm.Realm) 就如上文所提到的，Realm是连接Shiro和你的安全数据的桥梁。任何时候当Shiro需要执行登录或者访问控制的时候，都需要调用已经配置的Realm的接口去获取数据。一个应用程序可以配置一个或者多个Realm（最少配置一个）。