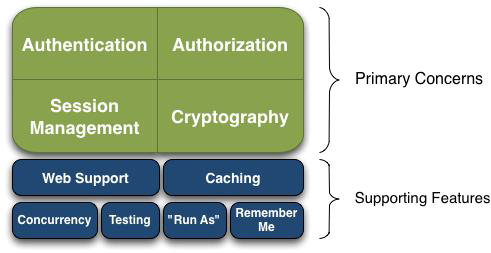
# 1.shiro简介



Authentication:身份验证/登录，验证用户有没有相应的身份。

Authorization:授权，验证已认证的用户有没有相应的权限。即判断用户有没有权限做某个事情，常见：验证某个用户有没有某个角色，或者细粒度验证某个用户对于某个资源是否具有某个权限。

Session Management:会话管理。用户登录后就是一次会话，会话结束前，相关的信息都在会话中。

Cryptography:加密，保证数据的安全性，如密码加密存储到数据库，而不是名文。

Web Support:shiro有良好的Web支持，能很容易的集成到web环境。

Caching:缓存。如用户登录后，用户信息/角色信息/权限等存入缓存，提高效率。

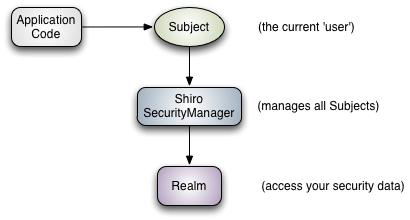
Concurrency:shiro 支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能

把权限自动传播过去。

Testing:测试支持。

“Run As”：允许一个用户假装为另一个用户（如果他们允许）的身份进行访问。

Remember Me:登录时记住我的功能。



Subject:相当于当前的“用户”，这个用户不一定是一个具体的人，与当前应用交互

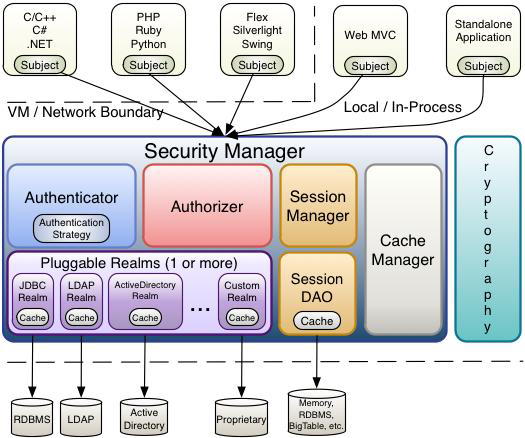
的任何东西都是Subject，如网络爬虫，机器人等；即一个抽象概念；所有Subject 都绑定

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，管理所有的subject，所有与安全相关的操作都要与SecurityManager交互。

Realm:域。shiro从realm中获取安全数据(如用户、角色、权限等)，SecurityManager验证用户时需要从realm中获取用户的数据来确定用户身份是否合法。Realm可以看作DataSource，即安全数据源。



**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。

**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。

**SessionManager：**如果写过Servlet就应该知道Session的概念，Session呢需要有人去管理

它的生命周期，这个组件就是SessionManager；而Shiro 并不仅仅可以用在Web 环境，也

可以用在如普通的JavaSE 环境、EJB 等环境；所有呢，Shiro 就抽象了一个自己的Session

来管理主体与应用之间交互的数据；这样的话，比如我们在Web 环境用，刚开始是一台

Web 服务器；接着又上了台EJB 服务器；这时想把两台服务器的会话数据放到一个地方，

这个时候就可以实现自己的分布式会话（如把数据放到Memcached服务器）；

**SessionDAO：**DAO 大家都用过，数据访问对象，用于会话的CRUD，比如我们想把Session

保存到数据库，那么可以实现自己的SessionDAO，通过如JDBC 写到数据库；比如想把

Session 放到Memcached 中，可以实现自己的Memcached SessionDAO；另外SessionDAO

中可以使用Cache进行缓存，以提高性能；

**CacheManager：**缓存控制器，来管理如用户、角色、权限等的缓存的；因为这些数据基本

上很少去改变，放到缓存中后可以提高访问的性能

**Cryptography：**密码模块，Shiro 提高了一些常见的加密组件用于如密码加密/解密的。

# 身份验证

**身份验证**，即在应用中谁能证明他就是他本人。一般提供如他们的身份ID 一些标识信息来

表明他就是他本人，如提供身份证，用户名/密码来证明。

在 shiro 中，用户需要提供principals （身份）和credentials（证明）给shiro，从而应用能

验证用户身份：

**principals**：身份，即主体的标识属性，可以是任何东西，如用户名、邮箱等，唯一即可。

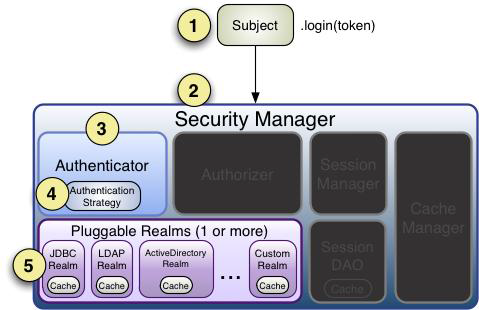
一个主体可以有多个principals，但只有一个Primary principals，一般是用户名/密码/手机号。

**credentials**：证明/凭证，即只有主体知道的安全值，如密码/数字证书等。

最常见的principals和credentials组合就是用户名/密码了。接下来先进行一个基本的身份认

证。

另外两个相关的概念是之前提到的**Subject**及**Realm**，分别是主体及验证主体的数据源。



流程如下：

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。

# 授权

授权，也叫访问控制，即在应用中控制谁能访问哪些资源（如访问页面/编辑数据/页面操作

等）。在授权中需了解的几个关键对象：主体（Subject）、资源（Resource）、权限（Permission）、

角色（Role）。

**主体**

主体，即访问应用的用户，在Shiro中使用Subject代表该用户。用户只有授权后才允许访

问相应的资源。

**资源**

在应用中用户可以访问的任何东西，比如访问JSP 页面、查看/编辑某些数据、访问某个业

务方法、打印文本等等都是资源。用户只要授权后才能访问。

**权限**

安全策略中的原子授权单位，通过权限我们可以表示在应用中用户有没有操作某个资源的

权力。即权限表示在应用中用户能不能访问某个资源，如：

访问用户列表页面

查看/新增/修改/删除用户数据（即很多时候都是CRUD（增查改删）式权限控制）

打印文档等等。。。

如上可以看出，权限代表了用户有没有操作某个资源的权利，即反映在某个资源上的操作

允不允许，不反映谁去执行这个操作。所以后续还需要把权限赋予给用户，即定义哪个用

户允许在某个资源上做什么操作（权限），Shiro 不会去做这件事情，而是由实现人员提供。

Shiro 支持粗粒度权限（如用户模块的所有权限）和细粒度权限（操作某个用户的权限，即

实例级别的），后续部分介绍。

**角色**

角色代表了操作集合，可以理解为权限的集合，一般情况下我们会赋予用户角色而不是权

限，即这样用户可以拥有一组权限，赋予权限时比较方便。典型的如：项目经理、技术总

监、CTO、开发工程师等都是角色，不同的角色拥有一组不同的权限。

**隐式角色**：即直接通过角色来验证用户有没有操作权限，如在应用中CTO、技术总监、开

发工程师可以使用打印机，假设某天不允许开发工程师使用打印机，此时需要从应用中删

除相应代码；再如在应用中CTO、技术总监可以查看用户、查看权限；突然有一天不允许

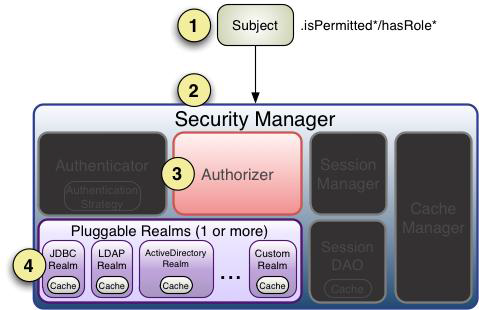
技术总监查看用户、查看权限了，需要在相关代码中把技术总监角色从判断逻辑中删除掉；

即粒度是以角色为单位进行访问控制的，粒度较粗；如果进行修改可能造成多处代码修改。

**显示角色**：在程序中通过权限控制谁能访问某个资源，角色聚合一组权限集合；这样假设

哪个角色不能访问某个资源，只需要从角色代表的权限集合中移除即可；无须修改多处代

码；即粒度是以资源/实例为单位的；粒度较细。



流程如下：

1、首先调用Subject.isPermitted\*/hasRole\*接口，其会委托给SecurityManager，而

SecurityManager接着会委托给Authorizer；

2、Authorizer是真正的授权者，如果我们调用如isPermitted(“user:view”)，其首先会通过

PermissionResolver把字符串转换成相应的Permission实例；

3、在进行授权之前，其会调用相应的Realm获取Subject相应的角色/权限用于匹配传入的

角色/权限；

4、Authorizer会判断Realm的角色/权限是否和传入的匹配，如果有多个Realm，会委托给

ModularRealmAuthorizer 进行循环判断，如果匹配如isPermitted\*/hasRole\*会返回true，否

则返回false表示授权失败。

# Realm及相关对象