Reids 的使用

下载网址<https://github.com/MicrosoftArchive/redis/releases>

详细教程<http://www.runoob.com/redis/redis-install.html>

下载安装好后用cm’d即可进入目录执行redis-server.exe redis.windows.conf

然后登陆客户端 redis -cli.exe –h 127.0.0.1 –p 6379

Redis和ehcache 的区别

1 ,EhCache 是一个纯Java的进程内缓存框架，具有快速、精干等特点，是Hibernate中默认的CacheProvider

Redis和memache（是一套分布式缓存系统） 并且后者是完全的基于内存

互联网现在主要用两种方式用来储存 关系数据库和key-value

一般都是get和set用来方式来使用redis

两种存储方法 （一种是snapshot 和 aof） 都是存储在内存中

Snapshot 是将数据先存在内容中，然后累加到某个设定的阀值后，一次触发DUMP操作，将变化的数据一次性的存到数据文件（RDB文件） （性能高一点）

AOF工作原理也是将数据存在内存，但是在存储的过程会调用fsync完成本次的日志文件

（安全性高一点）

使用redis时需要开启redis客户端 然后再进行spring-cache.xml 的编写 设置密码的还需要进行密码的设置修改

<import resource="spring-cache.xml"/>

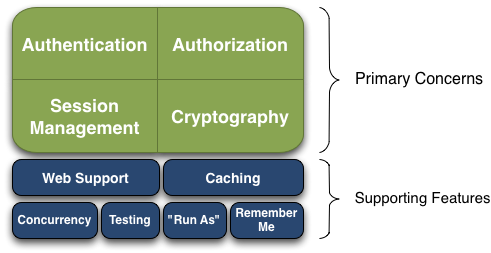
<import resource="spring-shiro.xml"/ 一般都是在是spring.xml添加其他配置文件

一般redis设置时 一个spring-redis.xml文件——服务器加载 还有一个redis.properties (相当于一个连接页面 密码什么的) 后台需要启动即可

Shiro使用

<https://www.jianshu.com/p/a956006bceee>

shiro 是一个功能强大和易于使用的Java安全框架，为开发人员提供一个直观而全面的解决方案的认证，授权，加密，会话管理



具体的四个功能

Authentication 身份认证

Authorization 授权 角色和权限是否匹配

Session management 会话管理（web环境或者javase）

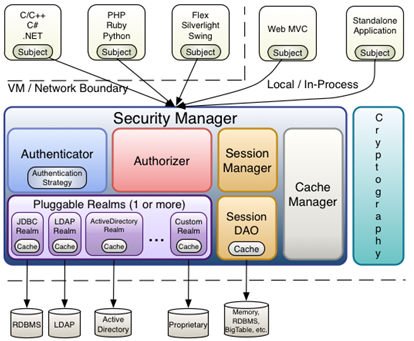
Cryptography 加密

Shiro 的其他特点（web支持， caching缓存，用户登陆之后，个人信息不用每次都查）

Concurrency 支持多线程并发，支持测试 run as 允许一个用户 假装另一个用户访问

Remember me 一次登陆之后，不用下次登陆了

**Shiro的架构**



SecurityManager 拦截并进行相应的处理，几乎所有的功能都是由他管理

其中：

Subject ：主体，相当于请求的用户

Securitymanager 是shiro的心脏；所有的具体交互都是由securitymanager进行拦截控制

，它管理所有的subject，并且负责进行认证授权以及会话，缓存

Authenticator 认证器，负责主体认证

Authrize 授权器

Realm 可以一个或者多个，登录认证时所需要的安全数据来源（账号密码）

Sessiondao 进行sessionde crud操作

Cryptography 密码模块，提高了一些加密组件

实际使用时

入门及使用手册http://blog.csdn.net/q547550831/article/details/54174158

首先进行相应的依赖注入org.apache.shiro(shiro-all)或者是jar包的引入

spring-shiro.xml 的配置整合添加 在配置文件中 选择相应的功能设置好相应的id

eg：<bean id=”securityManager” class=””>

<properity name=”realm”>

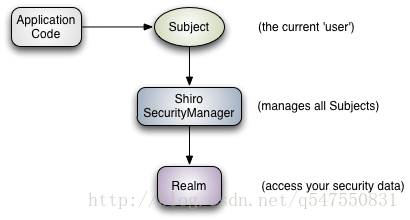
</bean> 最重要的是要进行相应的初始权限加载

（在配置文件中还可以添加相应的权限校验器Filter）及功能的实现

还可以添加自定义的realm 等其shiro可实现的功能<>

2 核心架构介绍

Shiro的三个核心组件 ： subject，secuirtyManager和realms



Subject：即“当前操作用户”。但是，在Shiro中，Subject这一概念并不仅仅指人，也可以是第三方进程、后台帐户（Daemon Account）或其他类似事物。它仅仅意味着“当前跟软件交互的东西”。但考虑到大多数目的和用途，你可以把它认为是Shiro的“用户”概念。

Subject代表了当前用户的安全操作，SecurityManager则管理所有用户的安全操作。

SecurityManager：它是Shiro框架的核心，典型的Facade模式，Shiro通过SecurityManager来管理内部组件实例，并通过它来提供安全管理的各种服务。

Realm： Realm充当了Shiro与应用安全数据间的“桥梁”或者“连接器”。也就是说，当对用户执行认证（登录）和授权（访问控制）验证时，Shiro会从应用配置的Realm中查找用户及其权限信息。

从这个意义上讲，Realm实质上是一个安全相关的DAO：它封装了数据源的连接细节，并在需要时将相关数据提供给Shiro。当配置Shiro时，你必须至少指定一个Realm，用于认证和（或）授权。配置多个Realm是可以的，但是至少需要一个。

Shiro内置了可以连接大量安全数据源（又名目录）的Realm，如LDAP、关系数据库（JDBC）、类似INI的文本配置资源以及属性文件等。如果缺省的Realm不能满足需求，你还可以插入代表自定义数据源的自己的Realm实现

Realm 验证是否登陆 并返回验证信息 验证是否拥有访问指定自愿的权限 并返回所拥有权限的方，判断是否支持token

。

[**http://blog.csdn.net/q547550831/article/details/54174158**](http://blog.csdn.net/q547550831/article/details/54174158)

**超好的**

**引入Maven依赖**

!-- 引入shiro框架的依赖 -->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-all</artifactId>

<version>1.2.2</version>

</dependency

**在web.xml中配置spring框架提供的过滤器，用于整合shiro框架**

<!-- 配置Spring整合shiro框架的过滤器，过滤器必须配置在Struts2核心过滤器上面 -->

<filter>

<filter-name>shiroFilter</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>shiroFilter</filter-name>

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

</filter-mapping>

**在Spring配置文件中配置bean**

配置安全管理器对象（shiro框架最核心的对象）

<!-- 配置安全管理器对象 -->

<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

</bean>

配置Shiro提供的一个工厂对象，用于产生过滤器对象

<!-- 配置Shiro提供的一个工厂对象，用于产生过滤器对象 -->

<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

<!-- 配置相关URL -->

<!-- 配置登录URL -->

<property name="loginUrl" value="/login.jsp"/>

<!-- 配置登录成功URL -->

<property name="successUrl" value="/home.jsp"/>

<!-- 配置未授权URL -->

<property name="unauthorizedUrl" value="/unauthorized.jsp"/>

<!-- 配置URL拦截规则 -->

<property name="filterChainDefinitions">

<value>

/css/\*\* = anon

/js/\*\* = anon

/images/\*\* = anon

/login.jsp\* = anon

/userAction\_login\* = anon

/manager\* = perms["manager"]

/\* = authc

</value>

</property>

<property name="securityManager" ref="securityManager"/>

</bean>

PS：Bean的ID要和过滤器中名称一致，因为过滤器在初始化时，就会加载与其名称一致的Bean

自定义一个Realm类

/\*\*

\* 用户Realm

\*

\* @author Switch

\* @data 2016年12月26日

\* @version V1.0

\*/

public class UserRealm extends AuthorizingRealm {

// 注入用户Dao对象

@Resource(name = "userDao")

private UserDao userDao;

// 注入权限Dao对象

@Resource(name = "functionDao")

private FunctionDao functionDao;

// 授予权限

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

// 创建简单授权对象

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

// 为用户授权

// 权限列表

List<Function> functions = null;

// 获取当前用户

User user = (User) principals.getPrimaryPrincipal();

if ("admin".equals(user.getUsername())) {

// 超级管理员，获取所有权限

functions = functionDao.findAll();

} else {

// 普通用户，通过用户id获取其对应的权限

functions = functionDao.findByUserId(user.getId());

}

if (functions != null && functions.size() > 0) {

for (Function function : functions) {

// 添加许可

info.addStringPermission(function.getCode());

}

}

return info;

}

// 权限认证

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {

// 强转为子类，用户名密码token，获取Subject传递过来的数据

UsernamePasswordToken userToken = (UsernamePasswordToken) token;

// 获取用户名

String username = userToken.getUsername();

// 在数据库中查询客户

DetachedCriteria detachedCriteria = DetachedCriteria.forClass(User.class);

detachedCriteria.add(Restrictions.eq("username", username));

List<User> userList = userDao.findByCriteria(detachedCriteria);

if (userList != null && userList.size() > 0) {

User user = userList.get(0);

// 获取密码

String password = user.getPassword();

// 简单认证信息对象,由SecurityManager负责对比密码数据

// 参数1：Object principal————一般是认证对象，在这里就是用户对象

// 参数2：Object credentials————证书，也就是负责校验认证的对象

// 参数3：String realmName————realm的名字，随意，但必须唯一

AuthenticationInfo info = new SimpleAuthenticationInfo(user, password, this.getName());

return info;

}

return null;

}

}