## 场景

为解决单机结构限制业务量和低可用的情况，采用集群结构代替，即当单机处理到达瓶颈，就把单机复制多份来构成一个集群。集群中每台单机是一个节点，且都提供相同的服务这样系统的处理能力就相当于提升了好几倍。

本场景是一个普通的页面web应用服务，含有登录功能。技术上：

* 使用了thymeleaf模板做前端页面；
* 使用redis做session缓存来实现多个web应用共享session；
* 使用nginx在web应用前面做负载均衡。

## 基本概念

**一、Cookie机制和Session机制回顾**

1）定义：Session成为“会话”，具体是指一个终端用户与交互系统进行通信的时间间隔，通常指从注册进入系统到注销退出系统之间所经过的时间。Session实际上是一个特定的时间概念。

2）HTTP协议与状态保持：HTTP 协议本身是无状态的，这与HTTP协议本来的目的是相符的，客户端只需要简单的向服务器请求下载某些文件，无论是客户端还是服务器都没有必要纪录彼此过去的行为，每一次请求之间都是独立的，好比一个顾客和一个自动售货机或者一个普通的（非会员制）大卖场之间的关系一样。

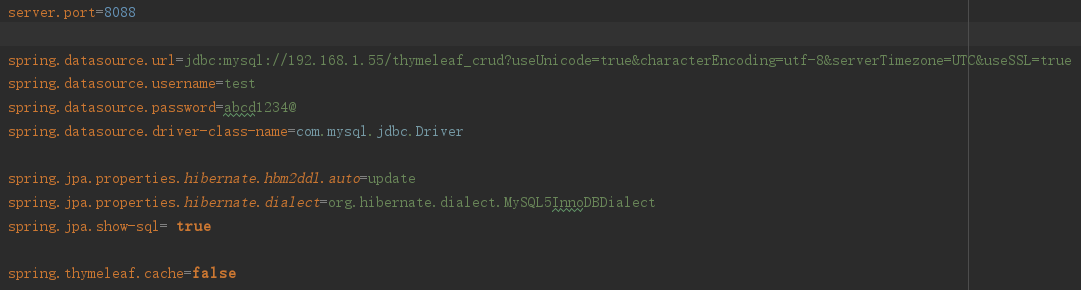
## 实现过程（案例一，使用了spring-session，非重点）

1. 编写实现对user进行CRUD的web应用

使用了thymeleaf模板引擎、spring jpa等技术

1. 使用maven打包(fat jar)

Fat jar可以单独使用java –jar xxx.jar运行。因为此web应用内嵌了tomcat，所以要修改其端口等。

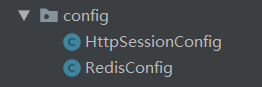


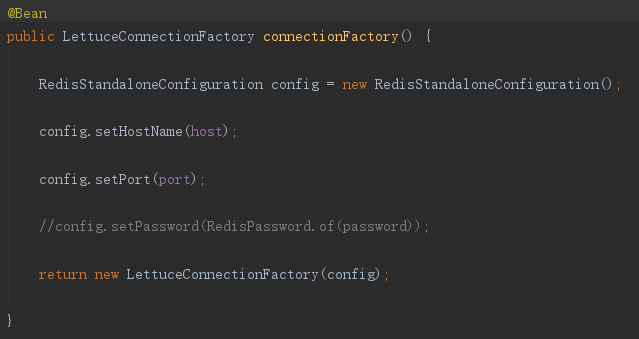
1. 使用spring session和redis做分布式会话

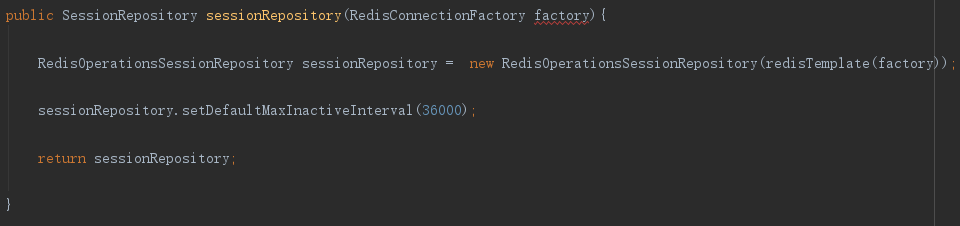
支持springboot2



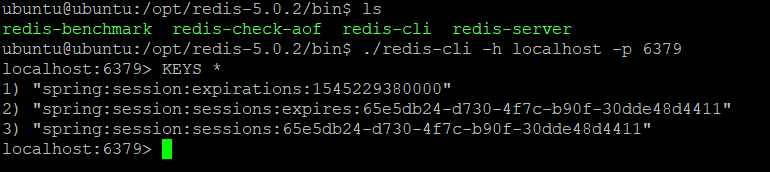
添加相应的配置





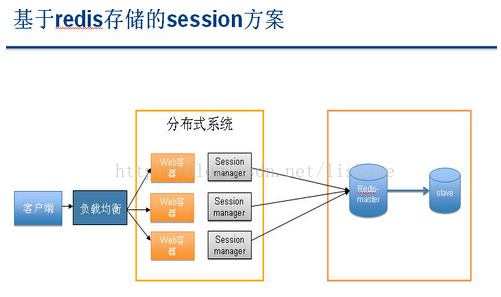


分别以8088和8099端口打包两个应用并运行，然后在两个浏览器窗口显示主页，分别得到的session id是一样的。还可以使用redis-cli连上redis服务器验证结果



## 实现过程（案例二，使用了shiro，重点）Springboot2集成shiro安全框架，并实现集群（通过redis）

* 初步架构方案



* 本应用的安全框架使用shiro
* 了解shiro框架的主要组件后，本应用场景需要使用的组件包括：

security manager（主角），

session manager，

sessionDAO（由于要共享session，所以使用redis作为存储而不是单机版的tomcat容器）

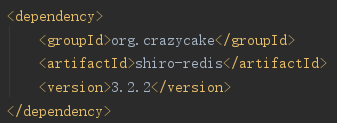
shiro realm（自定义的身份认证和鉴权模块，数据存储在mysql）

* springboot集成shiro

ShiroFilterFactoryBean，这是受spring容器管理的过滤器，任何url都经过它，由此受到shiro的安全管理。

* 本应用与redis服务器的通讯组件

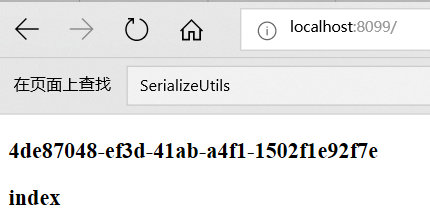
Shiro框架的sessionDAO和cacheManager组件都需要与redis服务器通讯，所以此处使用了网友开发的一个组件协助，省去了很多代码。主要组件有，

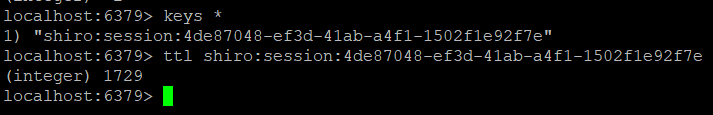


RedisSessionDAO，包含了shiro需要的session对象的CRUD

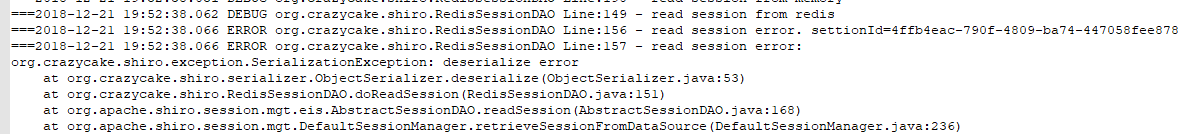
RedisManager，与redis服务器的通讯模块，内含jedis。打算以后更换成lettuce。

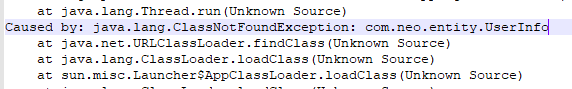
* 登录后验证session是否已经存入redis服务器（单tomcat）



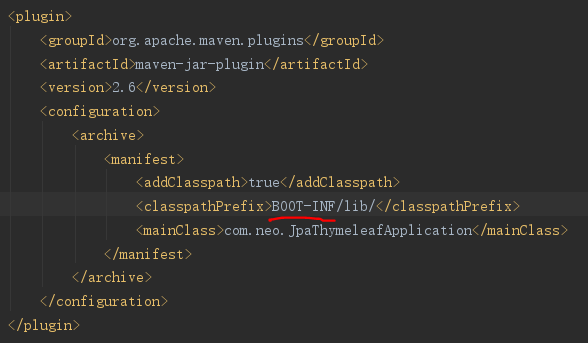


* 打包成fat jar后运行抛异常的原因分析

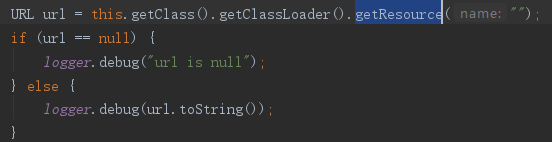




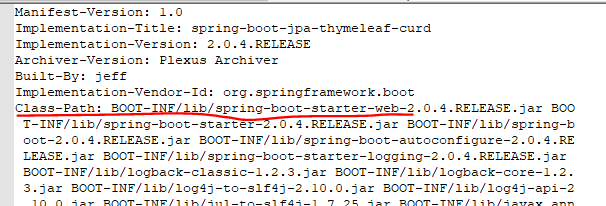
Shiro库抛异常说找不到UserInfo类，开始以为是devtools使用两个类加载器加载文件造成，后来发现原来是打可执行jar包时写lib路径不对造成的。没有把红色下滑线的目录写上。



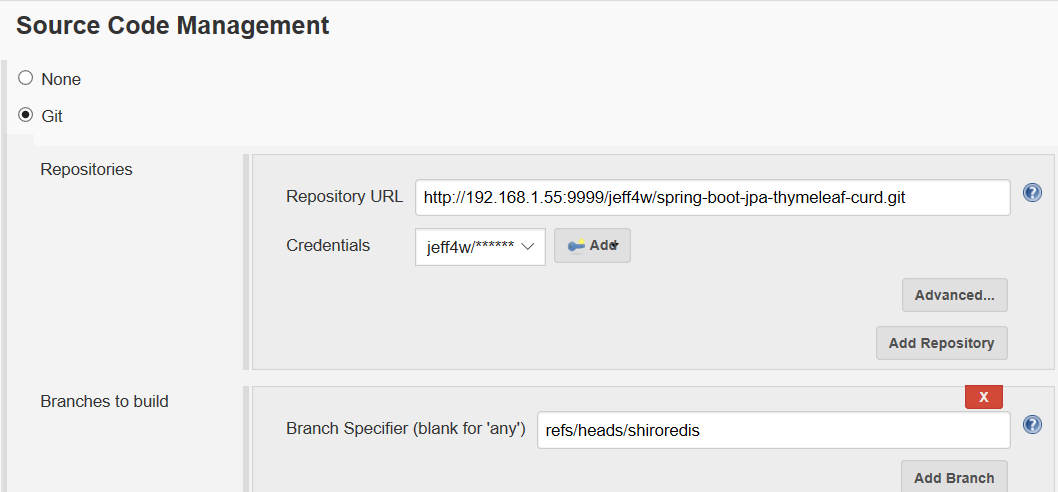
从以下代码打日志发现的，url等于null

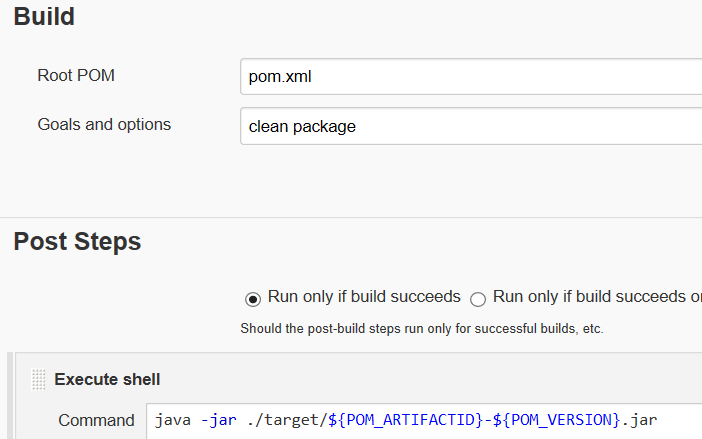


从jar包的MANIFEST.MF文件可以看出classpath路径问题



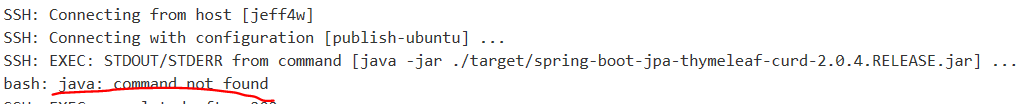
* 在jenkins中做打包和部署的动作
* 在jenkins新建一个job，然后配置其build environment。目的要在jenkins master上打包完成后直接本地run应用

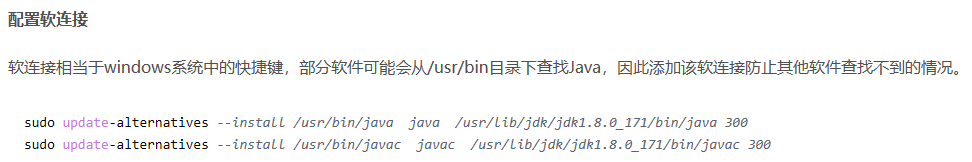




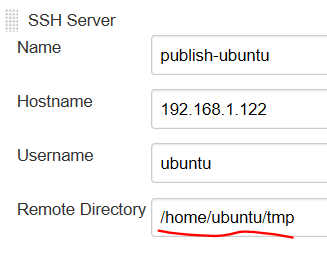
* 不在jenkins master同一台机器上部署web app，即使用SSH传送web app到指定部署机上运行

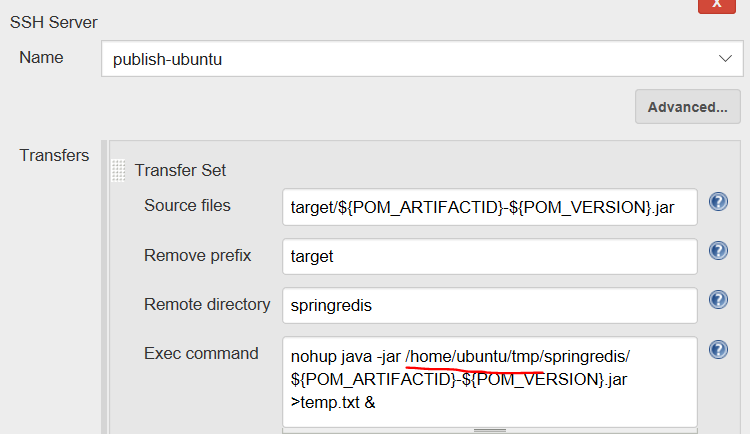
指定机器安装了jdk但说找不到java命令的解决，原因是





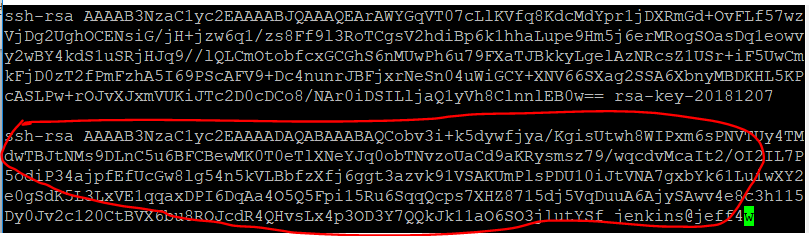
配置远程脚本命令





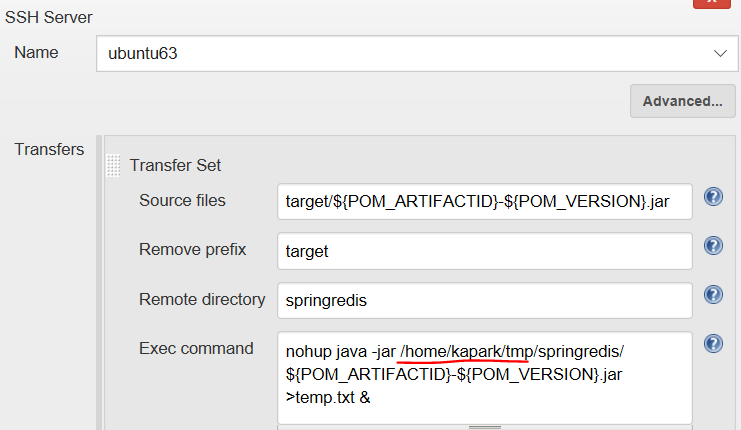
* 由于要把web app部署到两台不同的机器上，然后做负载均衡，所以要配置多一个SSH Server

如果jenkins master要SSH连上目的机，需要把jenkins的公钥告诉目的机（将id\_rsa.pub里边的内容放到目的机的.ssh/authorized\_keys文件中），





另外命令要写成nohup形式，即后台跑的程序，这样不会卡jenkins的job



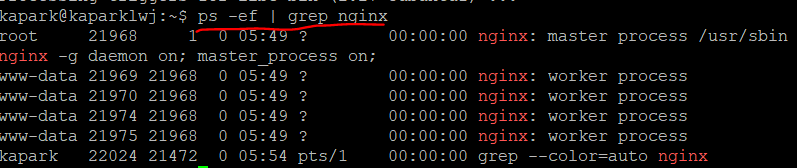
* 使用nginx负载均衡集群tomcat应用

单机web app如果脱机后用户会使用不到此应用，使用nginx做负载均衡后会扩展多几台web app，变成高可用。必须先解决session共享，即把session存储到redis服务器上，这样同一个浏览器只需拿着sessionid这条有钥匙就可以打开任何一台tomcat服务器。

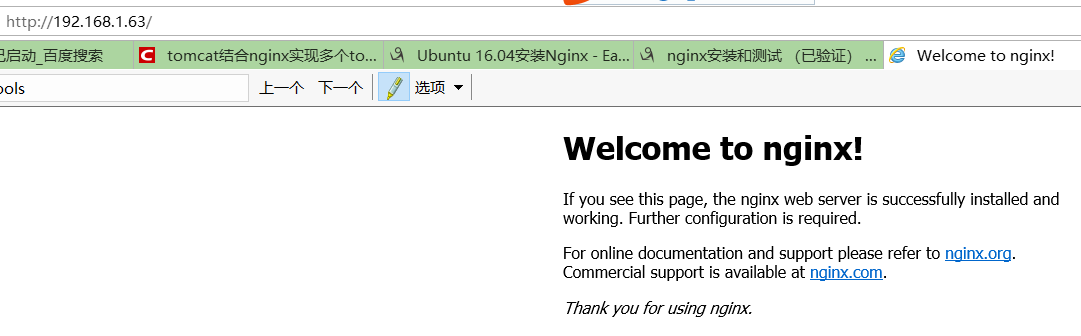
* 安装nginx

sudo apt-get install nginx

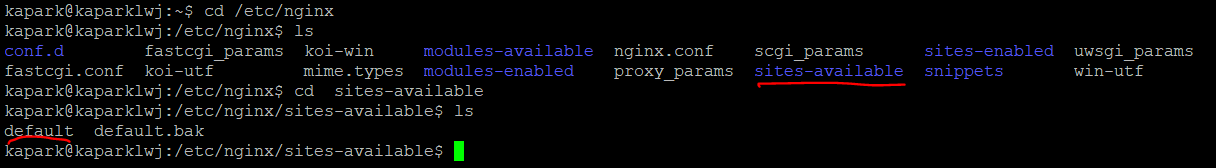
* 验证服务已启动



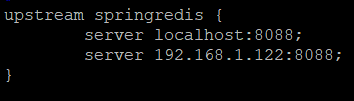
* 浏览器验证服务已启动，默认80端口

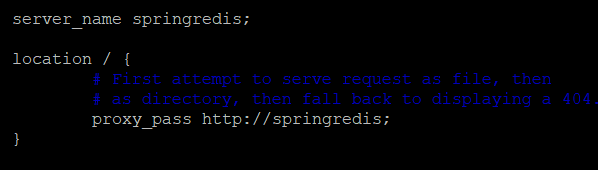


配置nginx的负载均衡



以下是运行web app的两台机器





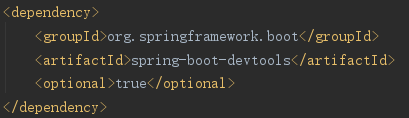
重启nginx



## 知识点

1. 热部署方便开发测试

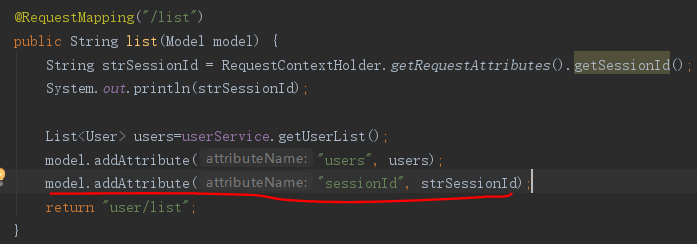
为了不需要重启就可测试改动的代码，做以下配置





对于html模板页每次更改完最好按ctrl+f9。

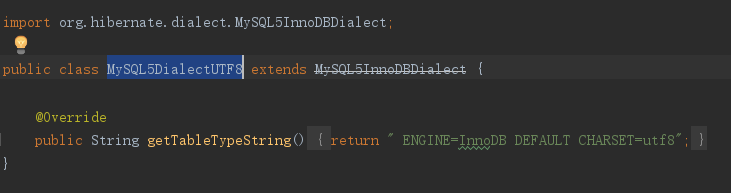
1. 获取session id并显示在前端页面





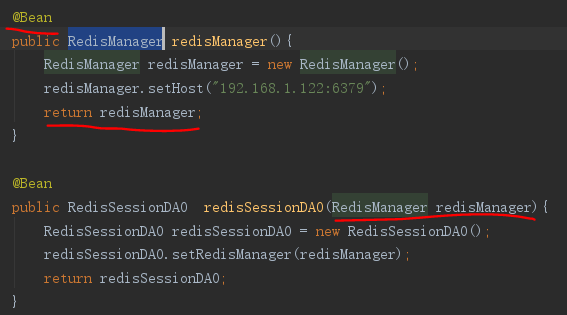
1. spring data jpa自动创建的mysql数据表字符集是拉丁，所以插入中文数据时会报错

自定义方言改进





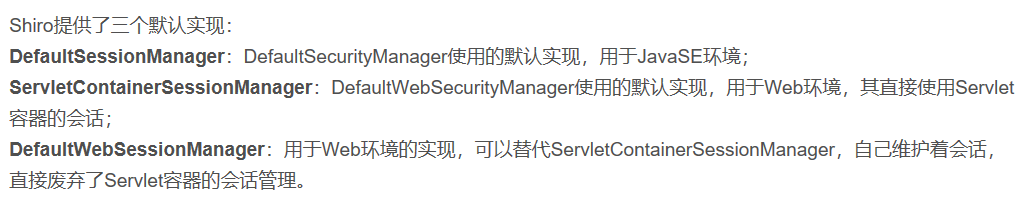
1. 往spring容器中注入第三方组件，以便集成使用



1. session管理的疑惑

tomcat有自己的session管理，shiro也可以定义自己的session管理，甚至spring-session也是spirng提供的一个session管理方案。到底选择哪种方案由具体场景决定。

* Shiro的方案，

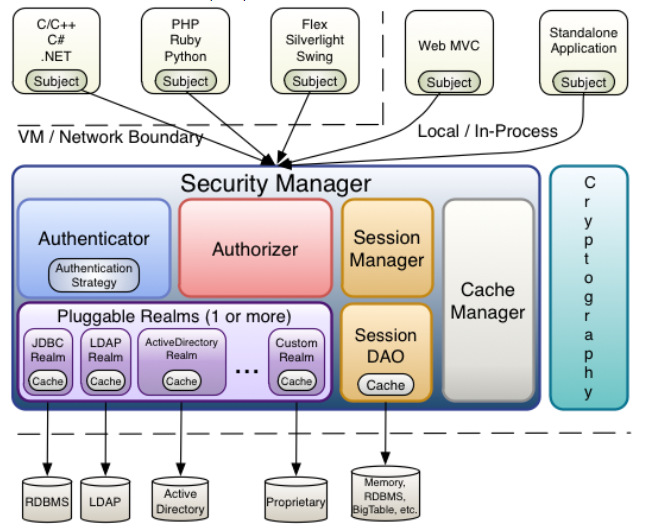


* Spring-session方案，

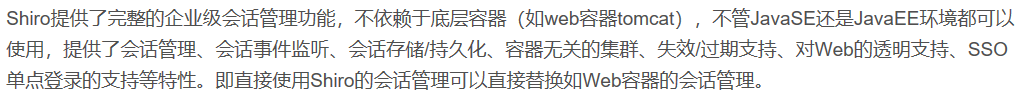
原理是相当于劫持了原有的servlet request和response，并包装为自己的request和response。目的是为了重写其getSession方法从而替换以前的session管理方案（例如使用redis来存储session等）。所以spring controller取到的session是spring-session的，并非tomcat的。



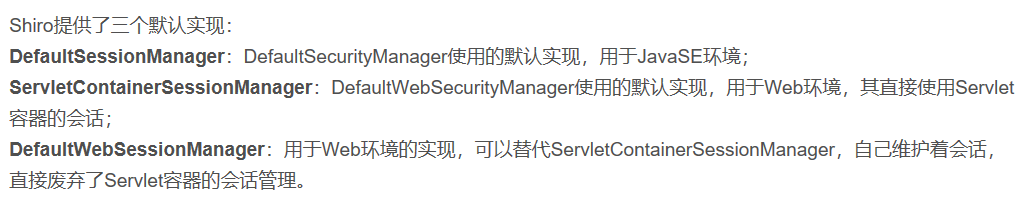
1. Shiro安全框架组件



* Session管理组件session manager
* Shiro提供了两个本框架下统一的接口session和session manager，用以让用户可以选择使用不同的管理方案但使用统一的接口处理。



* Shiro提供了三种session管理方案



* 会话监听器

会话监听器用于监听会话创建、过期及停止事件



* 会话存储/持久化

Shiro提供SessionDAO用于会话的CRUD，即DAO（Data Access Object）模式实现。可以使用redis作为会话存储介质并实现集群

* 会话验证

Shiro提供了会话验证调度器，用于定期的验证会话是否已过期，如果过期将停止会话；出于性能考虑，一般情况下都是获取会话时来验证会话是否过期并停止会话的；但是如在web环境中，如果用户不主动退出是不知道会话是否过期的，因此需要定期的检测会话是否过期，Shiro提供了会话验证调度器SessionValidationScheduler来做这件事情。

* sessionFactory

使用工厂模式创建session。根据相应的Subject上下文信息来创建会话；默认提供了SimpleSessionFactory用来创建SimpleSession会话。

## 场景考虑

登录后，在同一个浏览器再次链接网站会自动返回已登录的用户页面；在其他浏览器登录会没有反应，也可以做成其他浏览器登录把已登录浏览器踢出去。