查看docker是否安装成功：docker version

修改拉取镜像的国内地址

增加或者修改文件 /etc/docker/daemon.json

加入以下内容：

{

"registry-mirrors": ["http://hub-mirror.c.163.com"]

}

然后重启docker

删除镜像：

docker image rm ubuntu:18.04

docker rmi ubuntu:18.04

启动容器：docker run -p 8080:8080 tomcat

进入容器：docker exec -it 容器id

Dockerfile定制镜像

Dockerfile

FROM tomcat

WORKDIR /usr/local/tomcat/webapps/ROOT/

RUN rm -rf \*

RUN echo “hello world” > /usr/local/tomcat/webapps/ROOT/index.html

根据Dockerfile构建镜像

docker build -t myshop:v3 .

**注意Dockerfile中的每一行命令（比如WORKDIR,RUN等命令）都会构建一个容器在内存中，所以我们整个Dokerfile写错的时候可能会产生容器了，写对了应该没有，所以要注意这种调试时候的无用容器，要记得删除，而且Dockerfile写错了调试的时候也可能生成镜像（虚悬镜像）记得也删除。**

copy和add的区别：

add多一个解压缩的功能，一般生产环境只用copy命令，语义清晰。

RUN和CMD的区别：

run是构建镜像用的，cmd是生成的镜像当启动成容器的时候第一次执行的容器内命令，比如tomcat镜像里面会有catalina.sh run 说明tomcat镜像启动成为了容器就自动运行了

交互式运行

docker run -it -p 80:8080 --name tomcatContainer tomat

后台运行容器

[root@docker home]# docker run -p 80:8080 --name tomcat1 -d tomcat

db129f12e9c0110bb55d08445f62bb9b0da53fb67ed4f262fc609d7e0ccfe09e

[root@docker home]# docker run -p 81:8080 --name tomcat2 -d tomcat

0bdee33d8039fe234679159d8d7ba818166869fedfd97e0af3e70d3e6e4061fe

**启动了两个后台tomcat分别对应宿主机80和81端口，这样也理解了docker容器是负载均衡的入口，**

**容器前面加一层nginx就可以转发到不同的容器了**

**后台运行的怎么进去呢？**

**docker exec -it 容器id bash**

**进去后怎么退出呢？Ctrl+D（不会让容器也停止）**

容器挂载（我是在宿主机/home/docker/ROOT目录下建立了一个index.html）：

docker run -p 82:8080 --name tomcat82 -d -v /home/docker/ROOT:/usr/local/tomcat/webapps/ROOT tomcat

aa18c113043db1ea67c3fefeda357371cfe1da044d8d6ae4dcdbb49d86b75d66

安装mysql

docker pull mysql

安装的mysql是哪个版本呢？去hub.docker.com发现最新版的mysql都8.X了

mysql8.0最新特性 去掉了MyISAM引擎，InnoDB不仅支持事务而且效率提高，互联网企业转型，支持原生分布式解决方案

我们还是换成5.7吧，支持NoSql语法

docker pull mysql:5.7.22

docker run -p 3306:3306 --name mysql \

-v /usr/local/docker/mysql/conf:/var/log/mysql \

-v /usr/local/docker/mysql/logs:/var/log/mysql \

-v /usr/local/docker/mysql/data:/var/lib/mysql \

-e MYSQL\_ROOT\_PASSOWORD=123456 \

-d mysql:5.7.22

docker三剑客：

compose：还在用

machine（不怎么用）

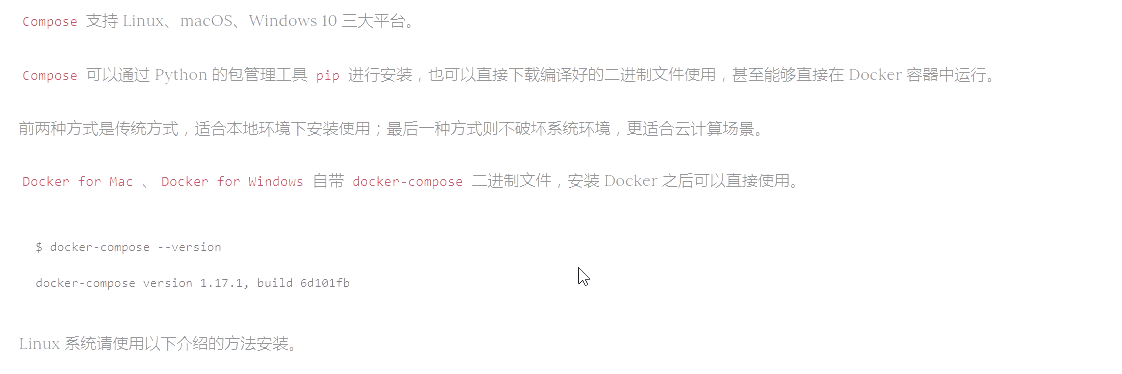
**swarm:(已经过时)**

**docker compose:**





安装



通过pip安装

yum -y install epel-release

yum -y install python3-pip

yum install gcc libffi-devel python-devel openssl-devel -y

pip3 install -U docker-compose

docker-compose文件编写(**注意用两个空格，不能用tab**)

[root@docker tomcat]# cat docker-compose.yml

version: "3.7"

services:

tomcat:(这个名字也可以随便起,有意义的就行)

restart: always

image: tomcat

container\_name: tomcat(启动的容器的名字，随便起)

ports:

- 8080:8080

docker-compose up

然后ctrl+C停止

再查看容器docker ps -a 发现退出了

docker-compose down 删除容器

docker-compose up -d 后台启动一组容器(容器也叫服务)

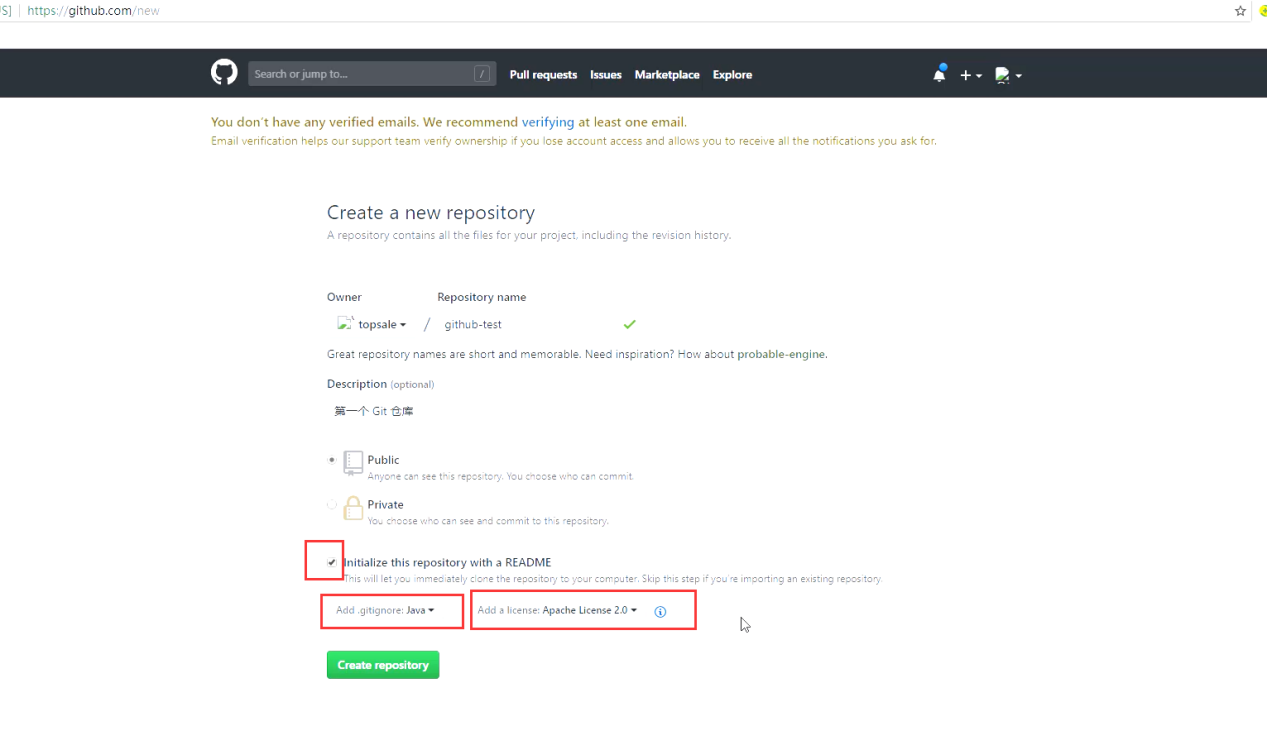
docker-compose logs tomcat 查看日志

git

先安装git，然后安装乌龟git，然后安装乌龟git中文破解包

github

注意几点最好勾选上，以前没注意过



.gitignore选成java有好处

建立好仓库后，选择clone with https的地址

然后本地找一个地方，git clone 地址就能跟仓库建立好联系了

然后.gitignore文件增加两行

.project

.idea

这两个开发工具自动生成不用提交

然后

git add .

git commit -m”增加eclipse和idea的过滤”

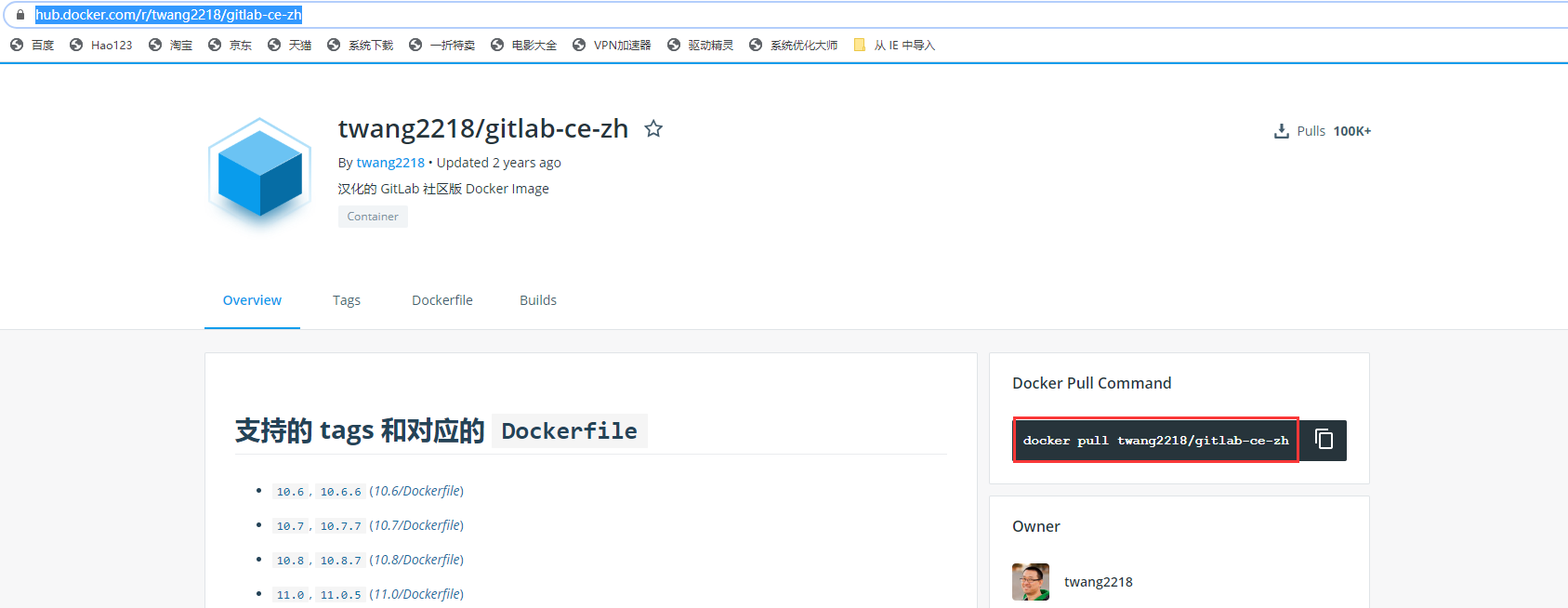
git push

公司内部安装个gitlab

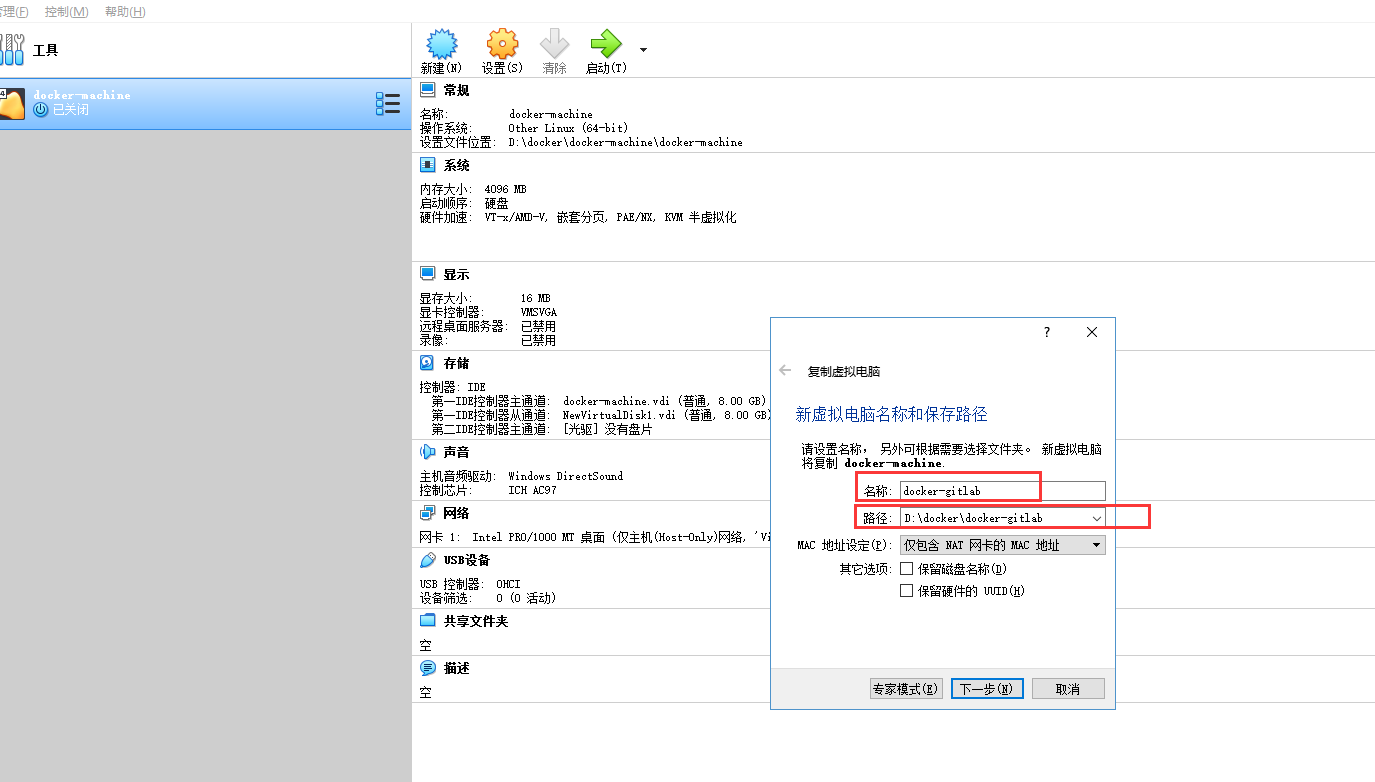
<http://hub.docker.com/>找gitlab的中文版

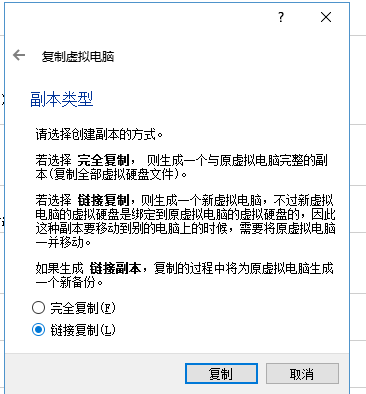
搜索这个gitlab-ce-zh关键字

<https://hub.docker.com/r/twang2218/gitlab-ce-zh>



学习gitlab之前再弄个虚拟机吧





链接复制有个特别有意思的地方，就是新复制的虚拟机跟以前的虚拟机是一样的ip，我用linux客户端工具连接，你都不知道自己连着谁，所以最好同时启动的虚拟机就一个，要不ip都一样有点乱。

题外话

docker pull过慢的解决办法,一直要执行以下三步骤

|  |
| --- |
| **在 /etc/docker/daemon.json 文件中添加以下参数（没有该文件则新建）：****{** **"registry-mirrors": ["https://9cpn8tt6.mirror.aliyuncs.com"]****}****服务重启：** systemctl daemon-reload(**以前都是没执行这个步骤，导致加速器不生效**)  systemctl restart docker |

gitlab compose

|  |
| --- |
| [root@docker gitlab]# cat docker-compose.yml  version: "3"  services:  web:  image: "tweng2218/gitlab-ce-zh"  restart: always  hostname: "192.168.137.100"  environment:  TZ: "Asia/Shanghai"  GITLAB\_OMNIBUS\_CONFIG:  external\_url "http://192.168.137.100"  gitlab\_rails['gitlab\_shell\_ssh\_port'] = 2222  unicorn['port'] = 8888  nginx['listen\_port'] = 80  ports:  - '80:80'  - '8443:443'  - '2222:22'  volumes:  - /usr/local/docker/gitlab/config:/etc/gitlab  - /usr/local/docker/gitlab/data:/var/opt/gitlab  - /usr/local/docker/gitlab/logs:/var/log/gitlab  [root@docker gitlab]# pwd  /usr/local/docker/gitlab |

执行docker-compose up总是执行不起来gitlab，估计是网络问题，先简单学学吧。

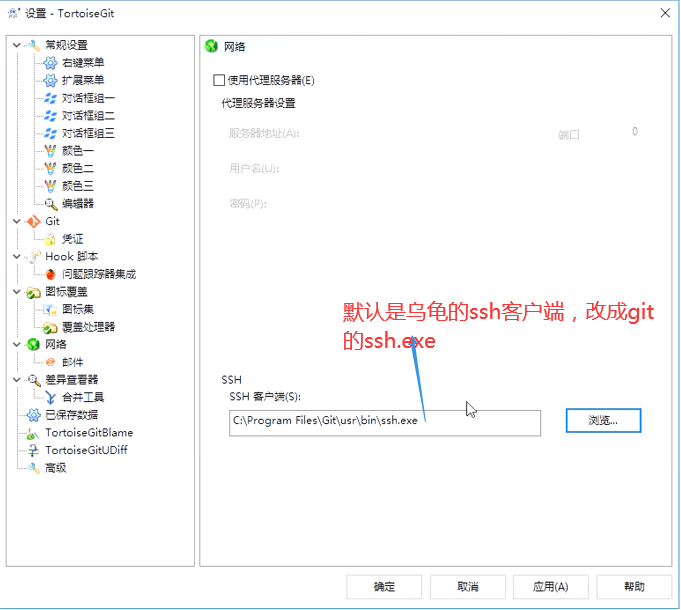
设置ssh访问gitlab

本地建立公钥和私钥，

ssh-kengen -t rsa -C “jinyuchuan@fang.com”

把公钥拷贝到gitlab上，本地留着私钥。

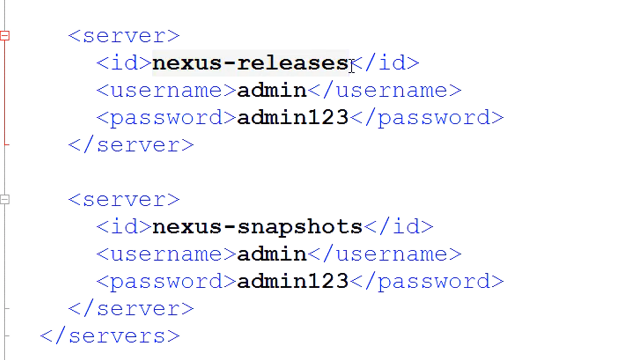
下载项目的时候记得先修改乌龟git客户端的设置



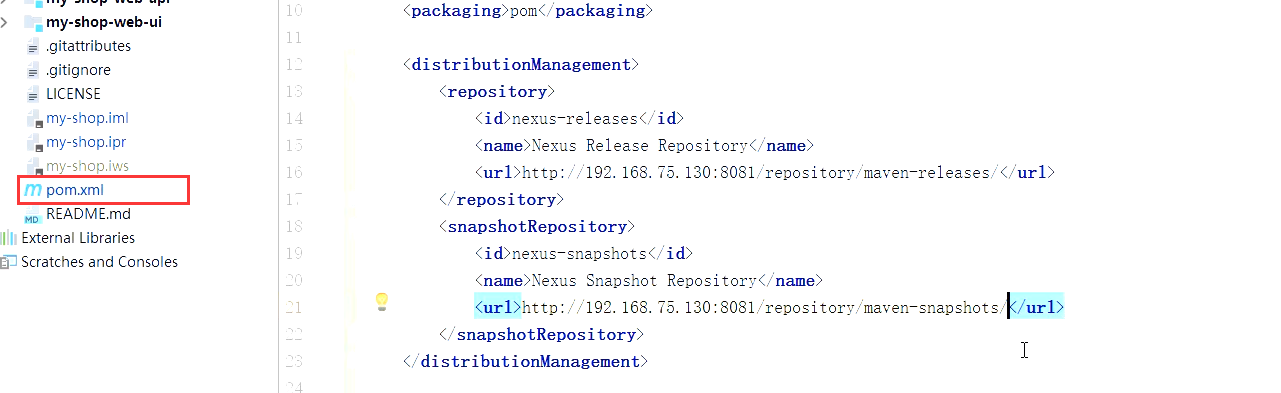
nexus(也用docker-compose安装，装坏了没事docker就是方便)

这里内存要分配两个G，内存溢出了，用htop小工具看内存，比top好用

maven settings里面配置



项目pom里面配置（用来往私服放东西）



注意id名称必须与settings.xml中的的Servers配置的id名称保持一致

项目pom里面配置（用来从私服拉取文件）



下面装docker registry docker镜像私服，略过

私服一般地址是:ip:5000/v2

然后我们docker pull的时候怎么知道走私服的配置呢？

配置机器的/etc/docker/daemon.json文件

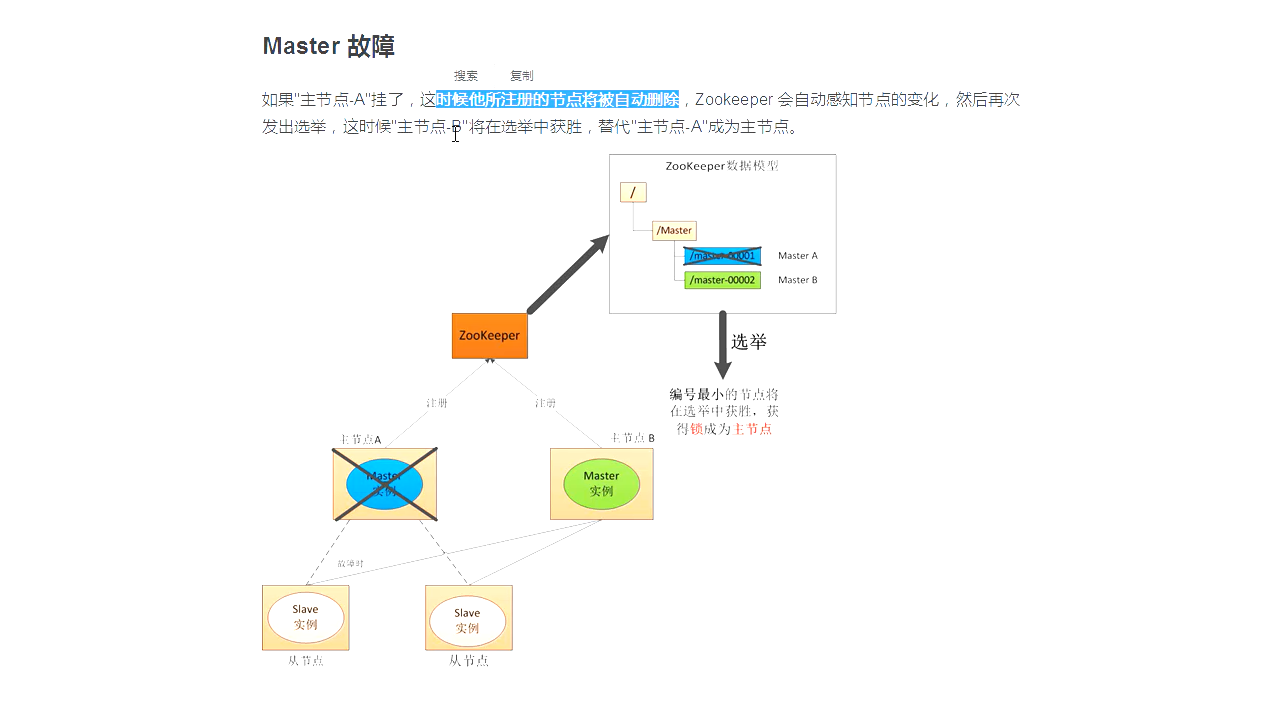


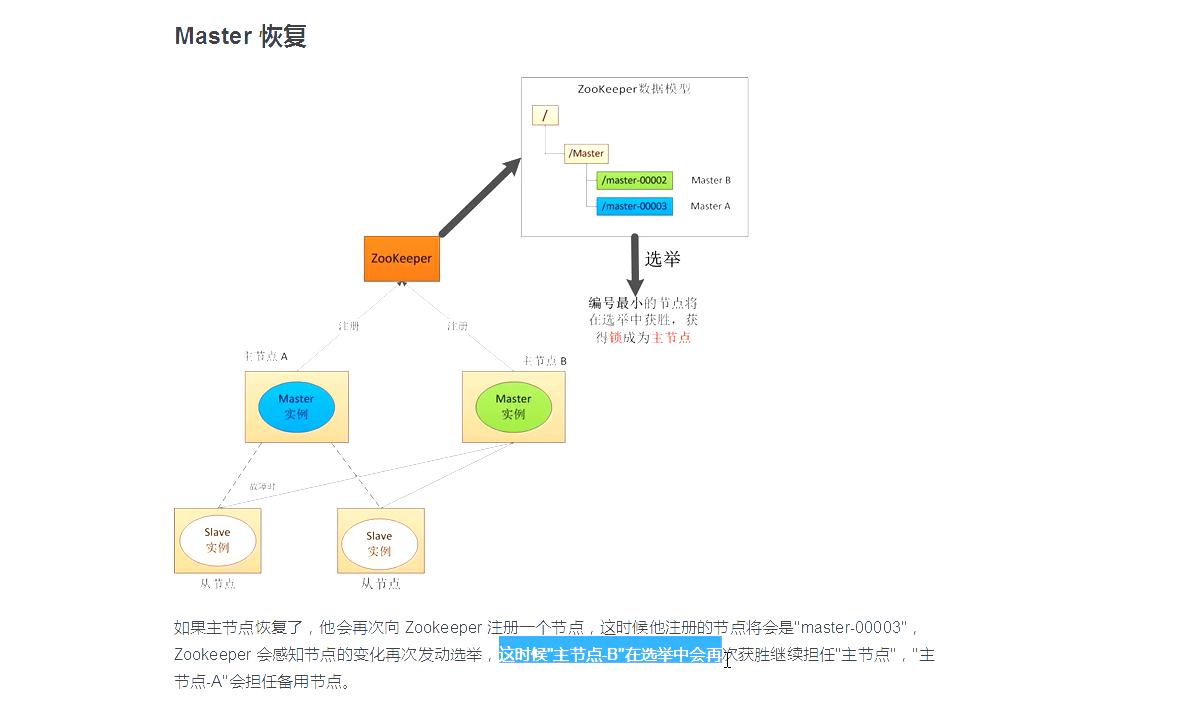
对内(公司内部)rpc（dubbo）,对外rest(springboot)

**单点故障解决办法：使用zookeeper选举主节点（有几个备用主节点，怎么知道哪个主节点真正不能提供服务，而我这个备用节点却可以当上主节点继续提供服务了呢，方法就是所有主节点向zookeeper注册节点，有zk在所有节点上选举出主节点，当主节点挂掉以后，zk会把这个主节点拿掉，从剩余的节点重新选举出来一个当主节点）**

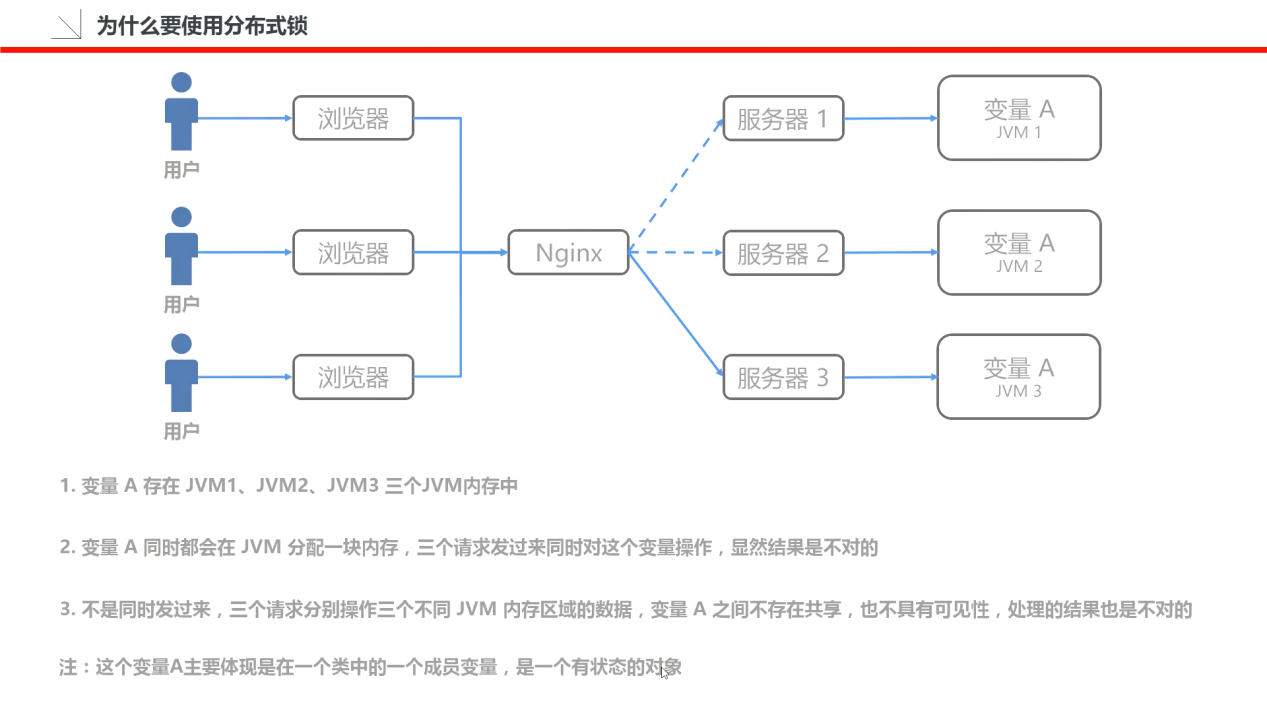
**问题来了，为什么不能备用节点ping主节点，从而知道主节点是否挂了呢？因为由于网络震荡，主节点可能没挂掉，这时候备用节点以为主节点挂了，那就麻烦了，就会启动自己，从而出现了两个主节点，这样系统就乱了**，解释图如下







分布式锁的应用场景？（**注意这时候你用多线程已经解决不了下面这个截图的问题了，因为分布式部署项目，你已经有好多个jvm了，而我们要操作的是同一个变量！update jyc**）

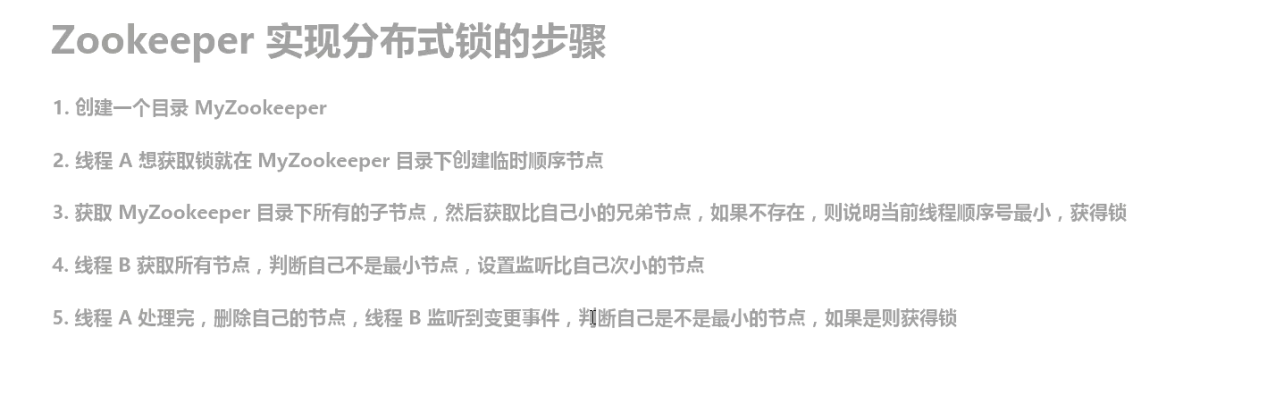




分布式锁特别难实现，要实现以下六个特点

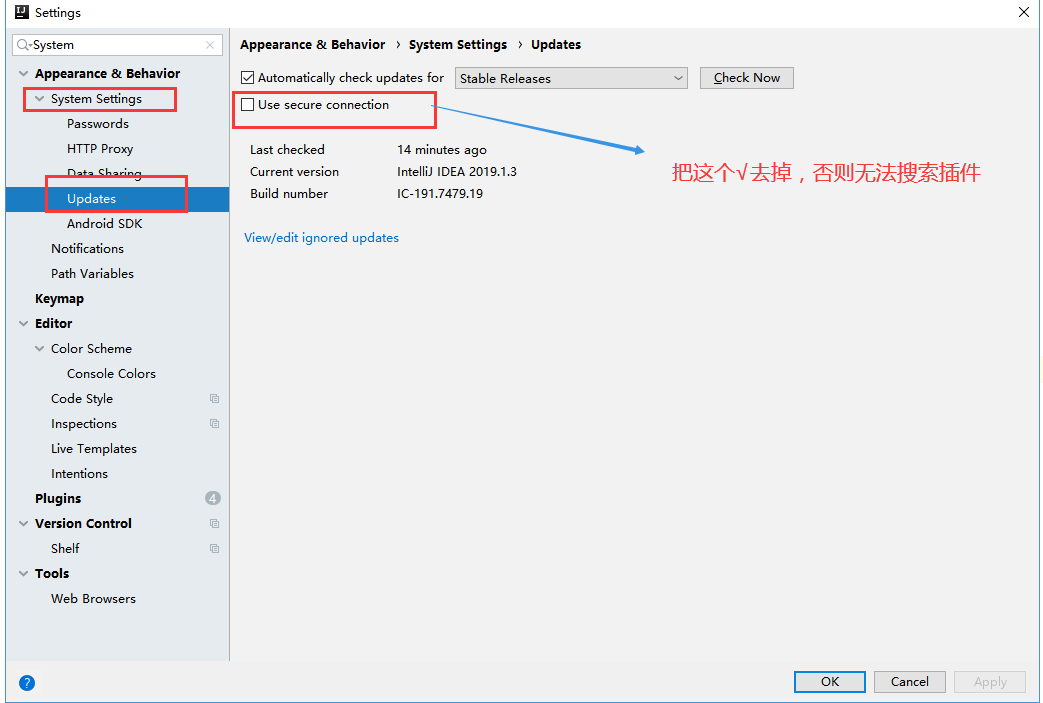


实际项目中实现分布式锁的具体实现方式，指导思想入下图，**这个指导思想很重要**

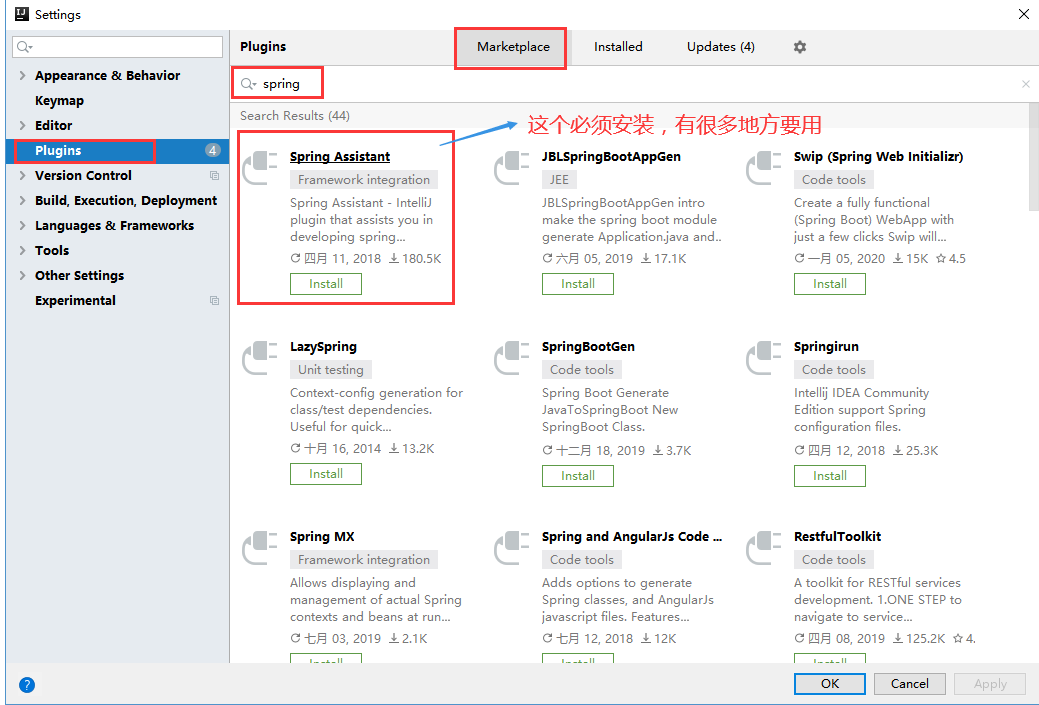


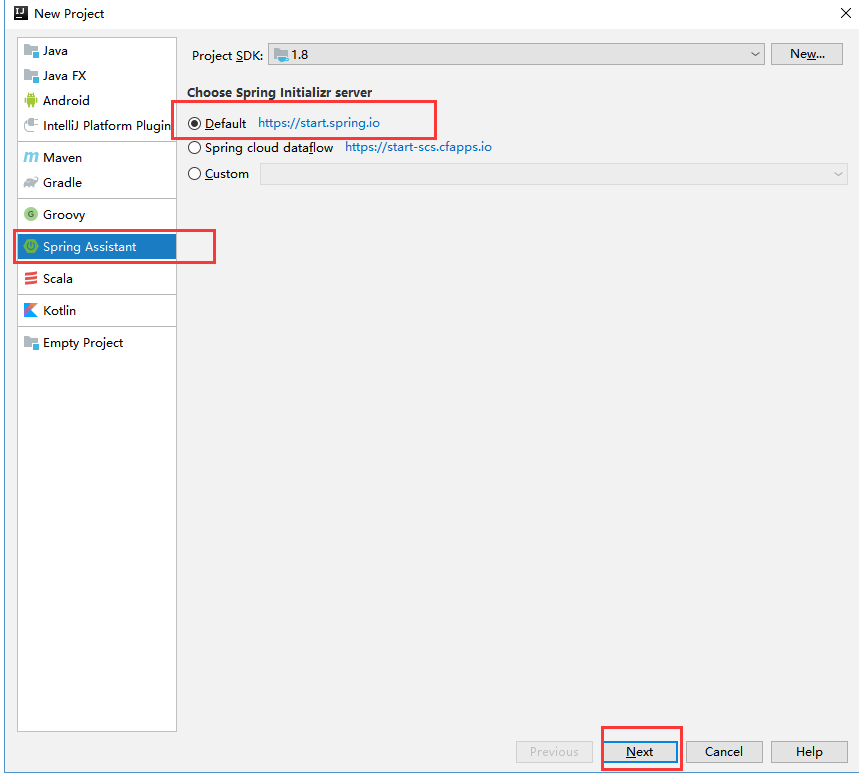
下面开始玩springboot了，先做一个非常重要的设置

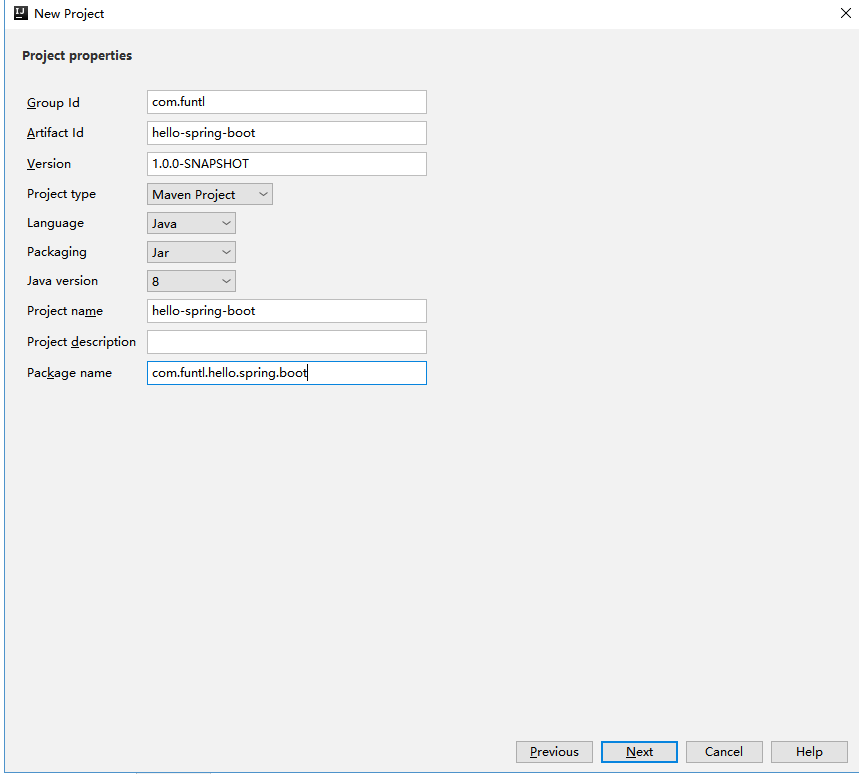
要不然在idea中搜索不到插件

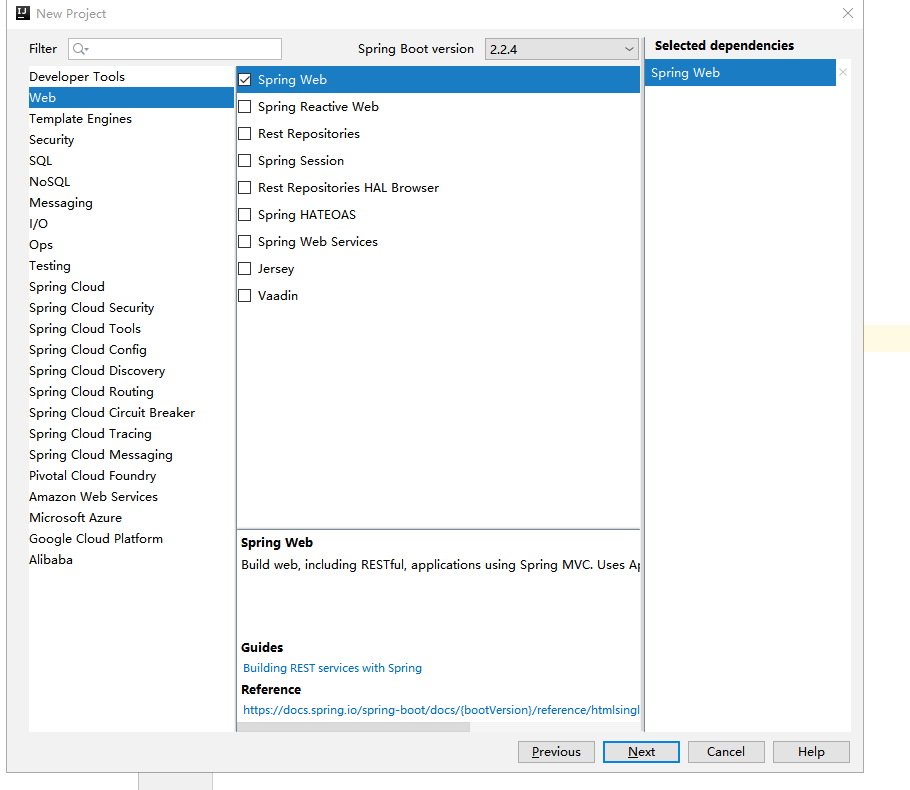


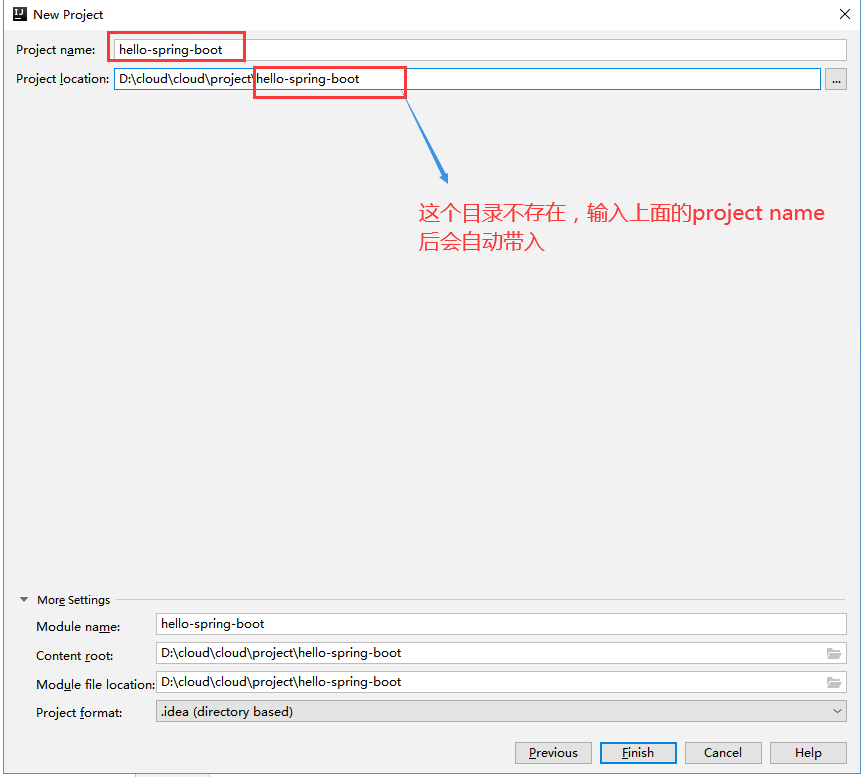
然后重启idea,并安装spring assistant











这才建立好了一个springboot项目！

以后所有的springboot和springcloud的代码放到这个目录下

<https://github.com/15801001455/cloud.git>

注意别忘了把application.properties改成application.yml

springboot单元测试

|  |
| --- |
| **package** com.funtl.hello.spring.boot; **import** org.junit.Before; **import** org.junit.Test; **import** org.junit.runner.RunWith; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.boot.test.context.SpringBootTest; **import** org.springframework.boot.test.web.client.TestRestTemplate; **import** org.springframework.boot.web.server.LocalServerPort; **import** org.springframework.http.ResponseEntity; **import** org.springframework.test.context.junit4.SpringRunner; **import** java.net.URL; **import static** org.hamcrest.CoreMatchers.*equalTo*; **import static** org.junit.Assert.*assertThat*; */\*\*  \* classes = HelloSpringBootApplication.class 这一句的意思是让单元测试类能装载到src/main/resources/下面的application.yml作为配置文件  \*/* @RunWith(SpringRunner.**class**) @SpringBootTest(classes = HelloSpringBootApplication.**class**, webEnvironment = SpringBootTest.WebEnvironment.***RANDOM\_PORT***) **public class** HelloSpringBootApplicationTests {   @LocalServerPort  **private int port**;   **private** URL **base**;   @Autowired  **private** TestRestTemplate **template**;   @Before  *//@BeforeAll* **public void** setUp() **throws** Exception {  **this**.**base** = **new** URL(**"http://localhost:"** + **port** + **"/"**);  }   @Test  **public void** contextLoads() {  ResponseEntity<String> response = **template**.getForEntity(**base**.toString(), String.**class**);  *//assertThat(response.getBody(),equalTo("Hello Spring Boot"));* **assert**(response.getBody().equals(**"Hello Spring Boot"**));  }  } |

项目启动时候设置佛祖版本banner

在resources目录下增加banner.txt,可以自定义启动banner

想原创进这个网址自己创新：<http://patorjk.com/software/taag/>

不要再学jsp了！！！！



thymeleaf配置

|  |
| --- |
| application.yml  spring:  thymeleaf:  cache: false #开发时关闭缓存,不然没法看到实时界面  mode: LEGACYHTML5 #用非html5严格标准  encoding: UTF-8  servlet:  content-type: text/html |
| pom.xml  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-thymeleaf</artifactId> </dependency> <dependency>  <groupId>nekohtml</groupId>  <artifactId>nekohtml</artifactId>  <version>1.9.6.2</version> </dependency> |
| index.html  <!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-spring4-4.dtd"> <html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"> <head>  <meta charset="UTF-8">  <title>Title</title> </head> <body>  <span th:text="${user.username}">张三</span> </body> </html>  MainController.java  @Controller public class MainController {  //约定优于配置,Thymeleaf默认找html为后缀的，**并且放到src/main/resources/templates**下的  @RequestMapping(value = {"","index"},method = RequestMethod.GET)  public String index(Model model){  User user = new User();  user.setUsername("Lusifer");  model.addAttribute("user",user);  return "index";  } } |

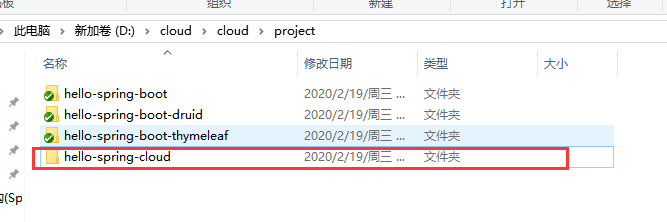
springboot集成数据库

|  |
| --- |
| 配置太多了，具体看 <https://www.funtl.com/zh/spring-boot-mybatis/%E4%BD%BF%E7%94%A8-MyBatis-%E7%9A%84-Maven-%E6%8F%92%E4%BB%B6%E7%94%9F%E6%88%90%E4%BB%A3%E7%A0%81.html#完整配置案例> 这个文章 |
| 还有这个代码：  <https://github.com/15801001455/cloud.git> |
| 这个代码自动生成mapper有点小bug |

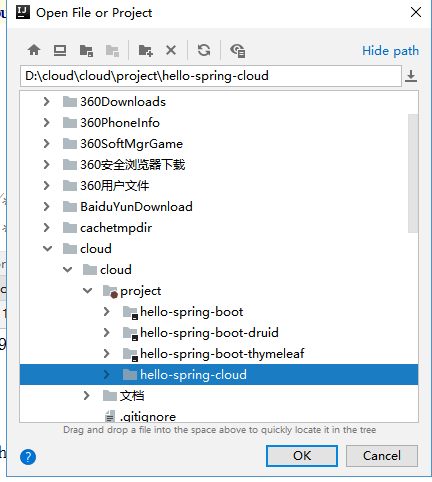
springcloud学习

目录建立的变化

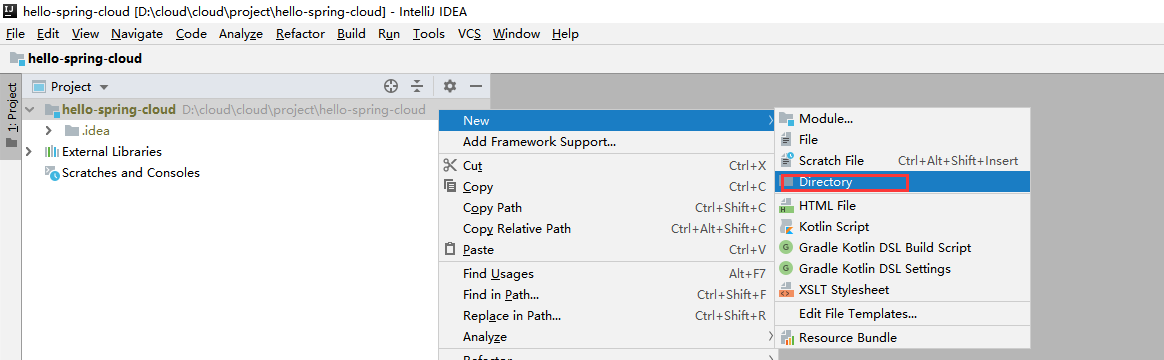
1. 先建立目录在硬盘上

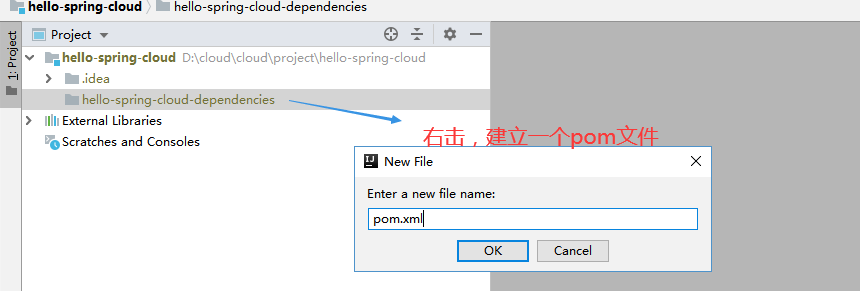


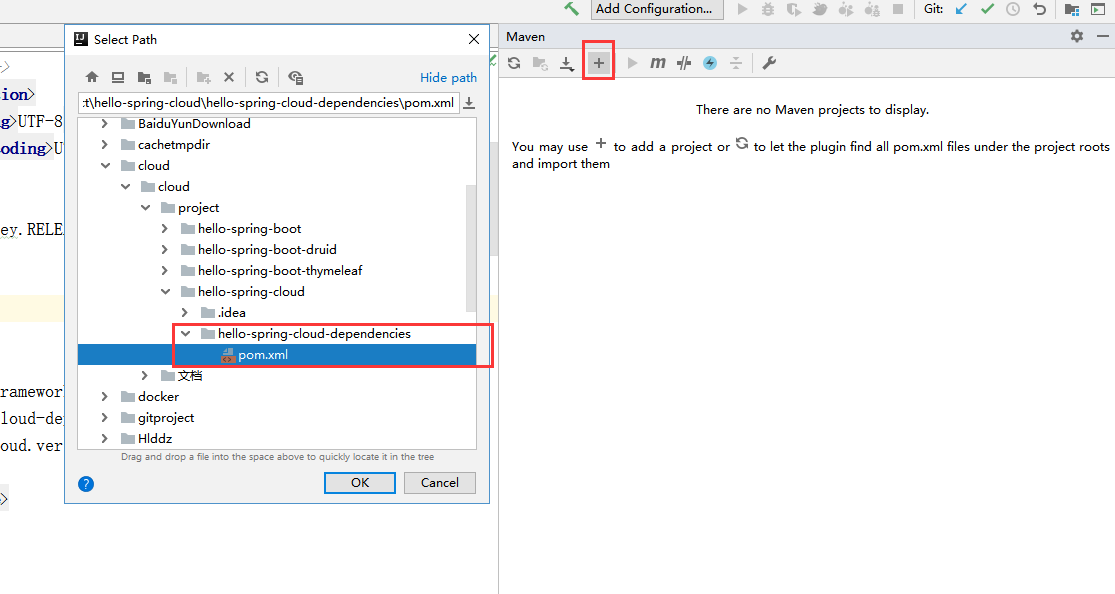
1. File->open...->找到刚建立好的目录，确定。



1. new一个目录，注意不是module或者project



1. 
2. 完成pom里面的内容(见具体代码吧)
3. 把这个项目手动托管进来idea



上面这个是依赖项目

下面开始看具体的服务注册与发现

Spring Cloud Netflix的Eureka（服务注册与发现 感觉和zk差不多功能）

下面具体看代码去了

先搭建eureka服务注册与发现的server端，访问地址 <http://localhost:8761/>

代码是hello-spring-cloud-eureka

然后搭建eureka的客户端（**什么是客户端呢？就是往eureka注册的各种服务，比如增删改查服务都是客户端，也叫服务提供者,相当于我们平时springboot做的接口站**）

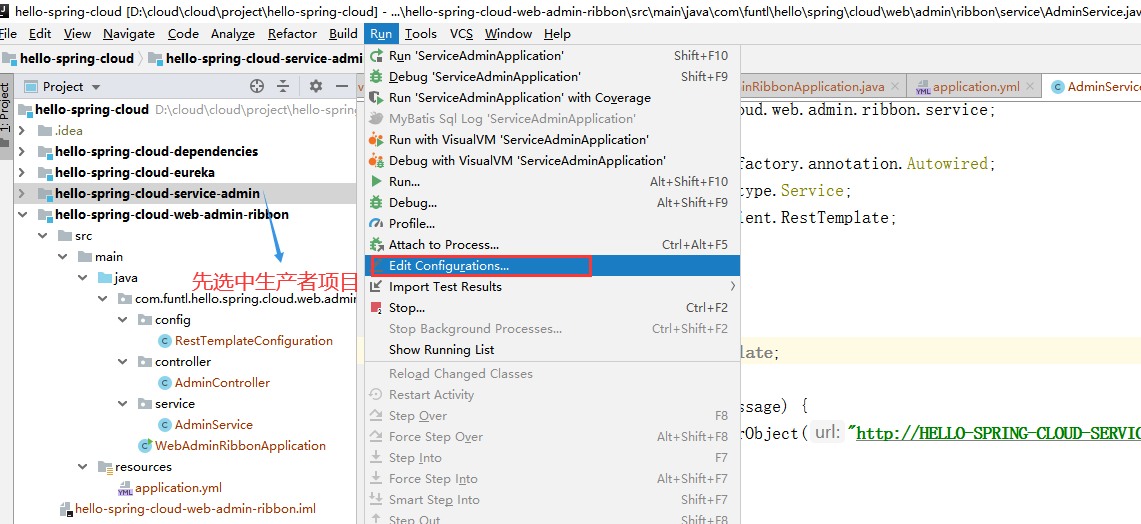
下面搭建服务消费者，有两种模式 一种ribbon，一种feign(相当于前台调用接口，一般是个web站点)

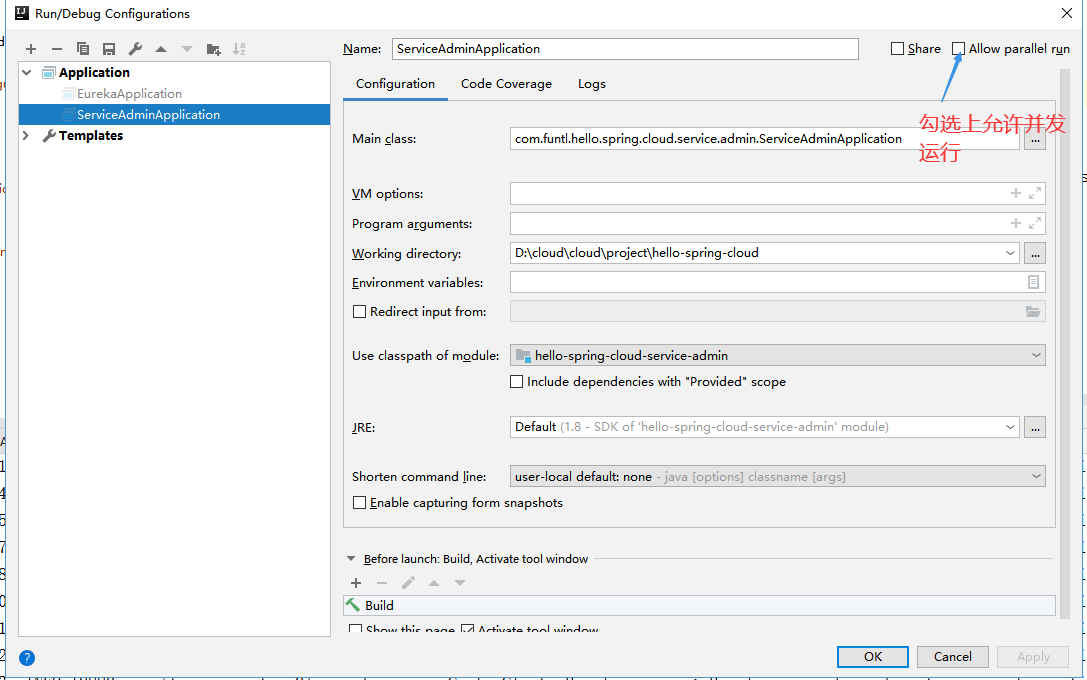
课程大概位置：<https://www.funtl.com/zh/spring-cloud-netflix/Spring-Cloud-%E5%88%9B%E5%BB%BA%E6%9C%8D%E5%8A%A1%E6%B6%88%E8%B4%B9%E8%80%85%EF%BC%88Ribbon%EF%BC%89.html#本节视频>

先学习ribbon消费者(ribbon+RestTemplate,比较老旧，负载均衡还有LoadBalance注解)，再学习feign消费者（本身集成了ribbon，并自动实现了负载均衡）

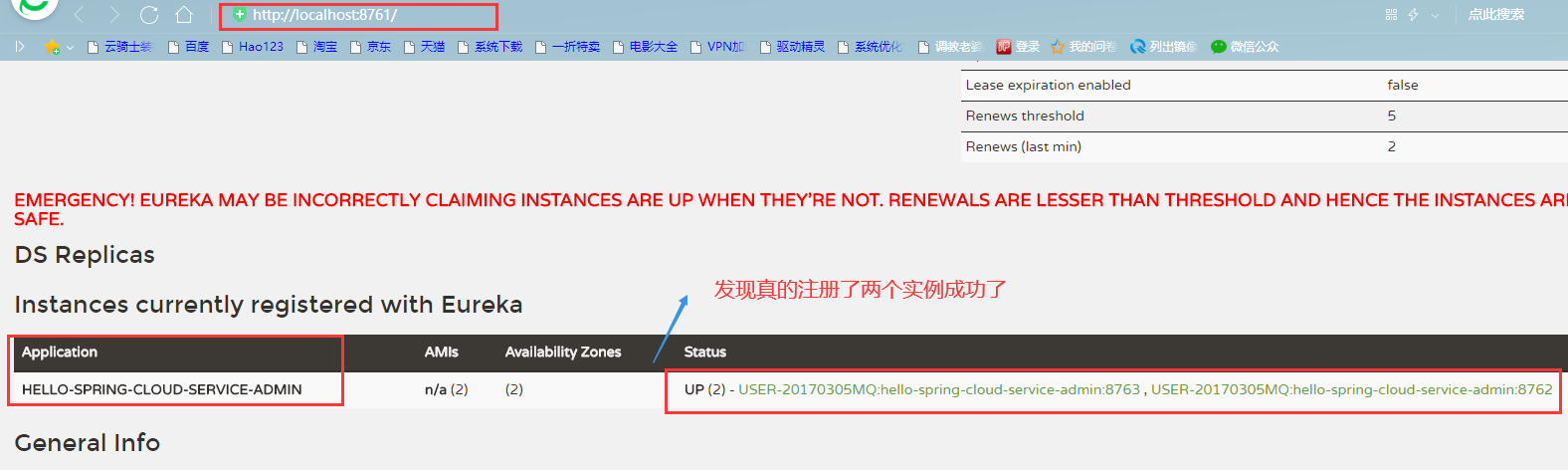
代码是hello-spring-cloud-web-admin-ribbon

注意这里我们要启动两个生产者注册到eureka，也就是刚才学习的项目hello-spring-cloud\_service-admin生产者项目，为什么启动两个生产者呢？为了模拟ribbon或者feign的客户端负载均衡,能请求到不同的消费者身上,怎么启动两个生产者是个小技巧，见下面

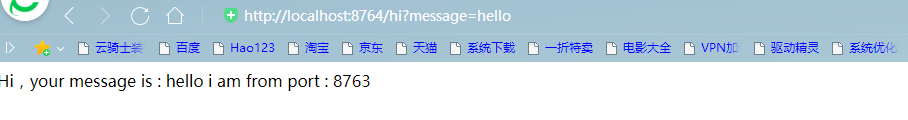


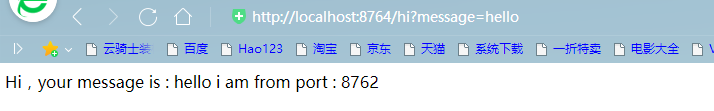


然后修改生产者的端口号为8763，然后再启动，这样一个生产者项目就能启动两个实例了，会注册到eureka上两个生产者实例。



ribbon启动后运行效果





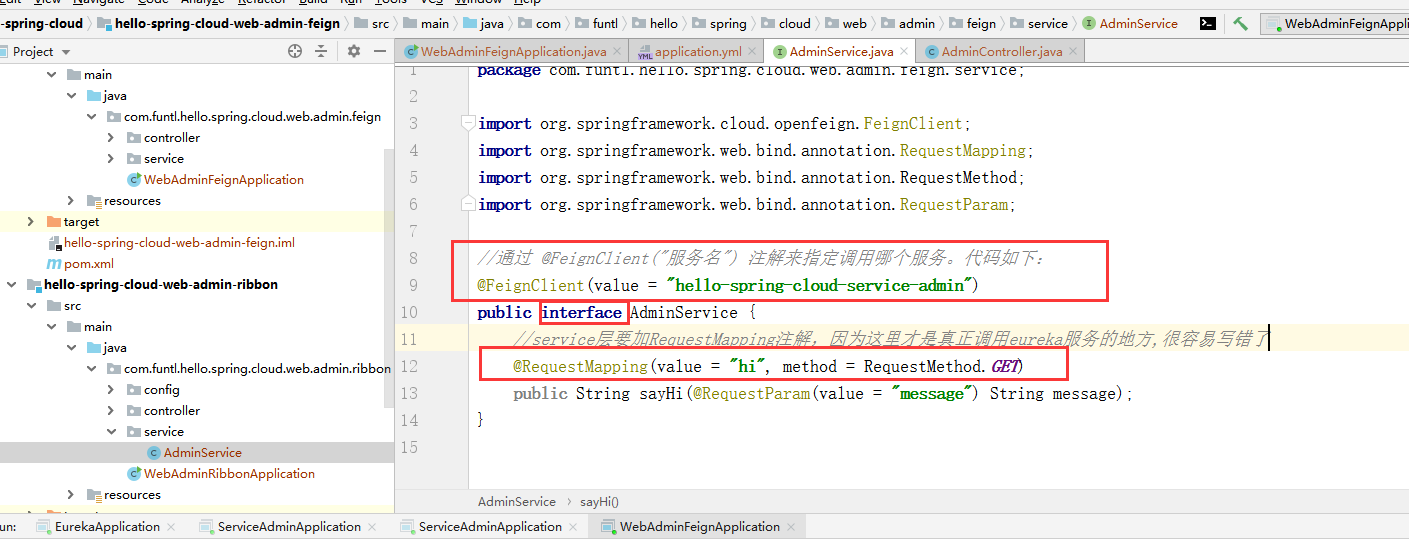
发现实现了请求不同的服务，实现了客户端负载均衡,

此时观察一下eureka服务器，发现ribbon消费者也注册上去了，说明eureka上面消费者和生产者是共存的



代码是hello-spring-cloud-web-admin-feign

feign关键类



消费端增加熔断器，防止生产者挂了，响应超时

分别在Ribbon和Feign增加,发现还是Feign更优雅

具体参考代码

具体参考文档：<https://www.funtl.com/zh/spring-cloud-netflix/Spring-Cloud-%E4%BD%BF%E7%94%A8%E7%86%94%E6%96%AD%E5%99%A8%E9%98%B2%E6%AD%A2%E6%9C%8D%E5%8A%A1%E9%9B%AA%E5%B4%A9.html#ribbon-中使用熔断器>

注意如果我们只有有一个生产者没有挂，那么熔断器方法就不会执行，会自动找到那台没有挂掉的生产者，真的很棒。如果所有生产者都挂掉了，就会进入熔断方法。

**注意两种方式的熔断方法感觉都有点问题，以后再说吧，不是很重要，实际用到再说。**

下面讲解的熔断器的仪表盘监视器，用到再看。

下面这个概念很重要：路由网关统一访问接口（感觉像是搜房的中转）

讲解的是zuul网关，实际看spring有一个gateway套件可能更好，并发性好，但是老师没讲。

客户端（app,pc用户）-> zuul -> feign(相当于web api，消费者消费)-> eureka <- 生产者生产

zuul可能好处是统一权限控制，让app，pc有统一的访问地址，与后端业务隔绝，可以在zuul这一层做一些过滤等，可能还有别的好处没想到。而且感觉zuul被抛弃的感觉，参考https://github.com/spring-cloud/spring-cloud-netflix/issues/1498

**注意如果feign挂了，或者生产者挂了都会走zuul的容错方法。**

Spring Cloud Config

类似于携程的applo

就是把所有项目的配置基于Spring Cloud Config进行修改，SCC分为客户端和服务端，服务端代码为hello-spring-cloud-config，客户端基于hello-spring-cloud-web-admin-feign这个项目先改造

首先这个项目hello-spring-cloud-config 服务端配置

|  |
| --- |
| **spring**:  **application**:  **name**: hello-spring-cloud-config  **cloud**:  **config**:  **label**: master *#github分支* **server**:  **git**:  **uri**: git@github.com:15801001455/cloud.git  **search-paths**: project/hello-spring-cloud/git-manage-config *#git项目下的搜索目录* **force-pull**: true  **server**:  **port**: 8888 *#默认就是这个端口,不能在application.yml这个文件中进行修改* **eureka**:  **client**:  **serviceUrl**:  **defaultZone**: http://localhost:8761/eureka/ *#也是要连接eureka* |

注意事项，uri填ssh地址，要保证把本地的rsa.pub文件内容复制到github里面，才能保证连接成功，不能填https地址，要不github用户名密码就暴露了。

在search-paths下手动建立文件web-admin-feign.yml,并要上传到github，这样项目就能读到这个目录下的所有配置了。

访问地址：<http://localhost:8888/web-admin-feign/master>

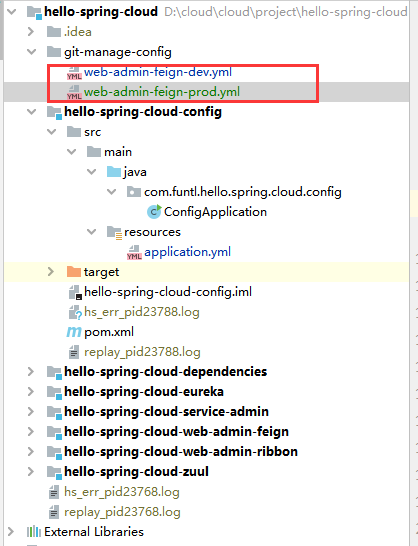
访问地址中的master代表分支，具体规则还要看下面的讲解。

web-admin-feign代表文件web-admin-feign.yml

改造：

先把配置文件名字改了 web-admin-feign-dev.yml，然后上传，

然后拷贝一份改名字为web-admin-feign-prod.yml,然后上传。



这样的话访问地址变为：

[http://localhost:8888/web-admin-feign/dev/master](http://localhost:8888/web-admin-feign/prod/master)

<http://localhost:8888/web-admin-feign/prod/master>

**注意/web-admin-feign/dev就能找到web-admin-feign-dev.yml**

**下面改造客户端读spring cloud config的配置**

**客户端基于feign改造**

**增加pom内容**

|  |
| --- |
| <**dependency**>  <**groupId**>org.springframework.cloud</**groupId**>  <**artifactId**>spring-cloud-starter-config</**artifactId**> </**dependency**> |

**修改feign项目配置文件**

|  |
| --- |
|  |

**平时本地测试默认的配置application.yml是走的dev。**

**正常feign项目上生产执行**

**java -jar hello-spring-cloud-web-admin-feign-1.0.0-SNAPSHOT.jar --spring.profiles.active=prod，就走application-prod配置了。**

**ZipKin 服务追踪项目，用到再说。**

**springbootadmin项目，服务管理，用到再说。**

**注意所有项目的application.yml类似的都变成bootstrap.yml，要不运行到生产环境会有问题。**

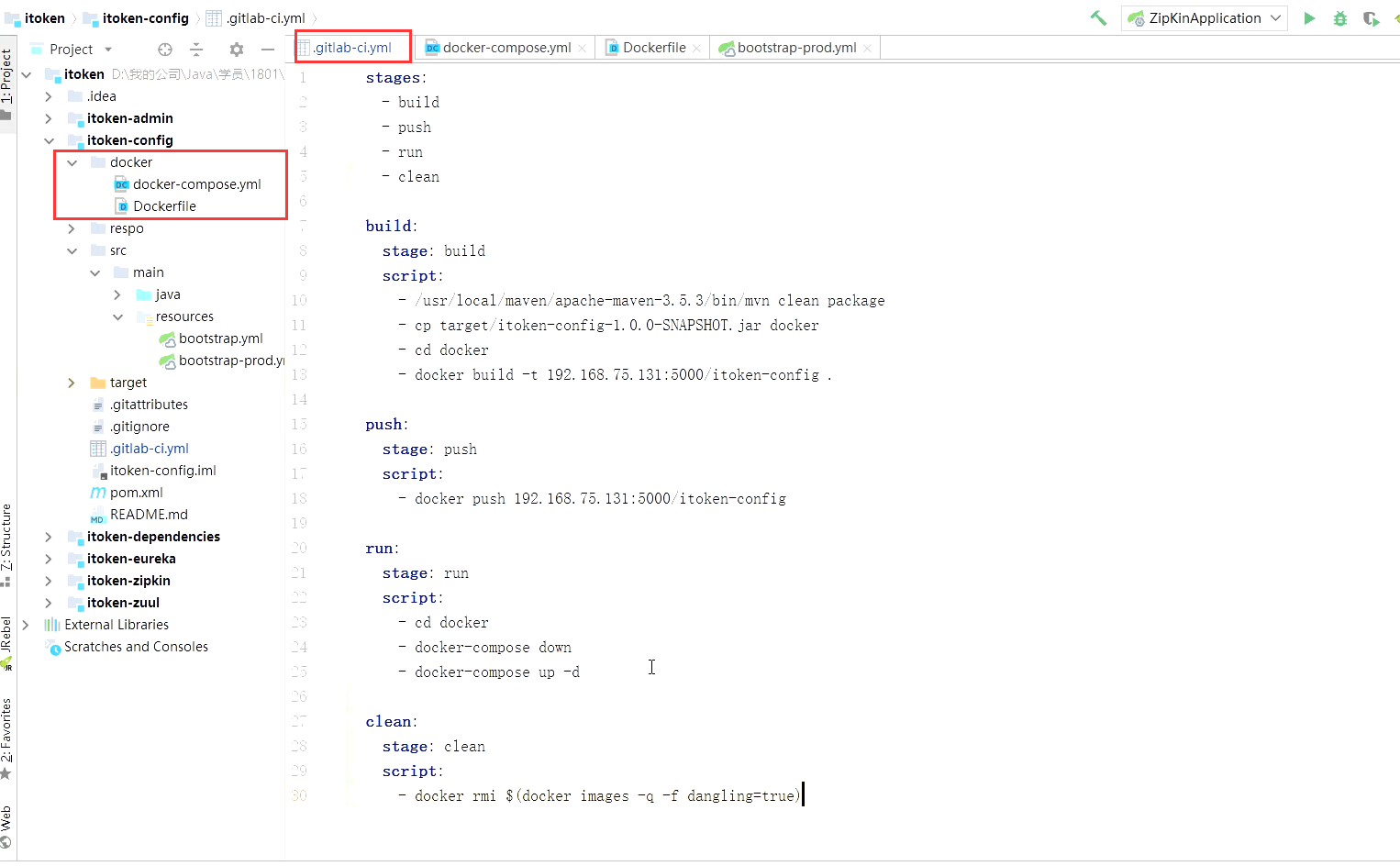
**持续集成**

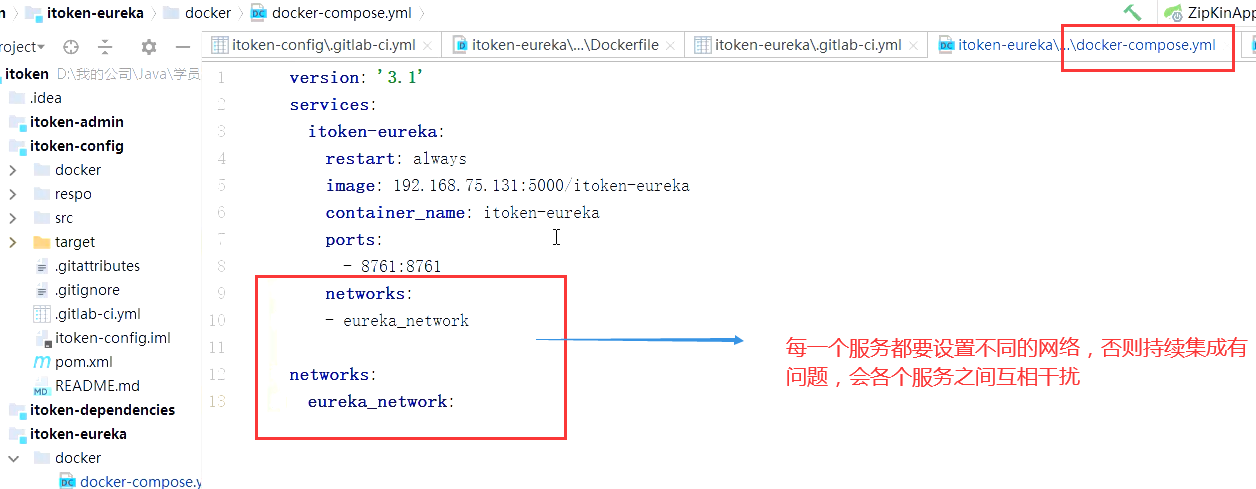
**这里有个不同的思路，一般用jenkins，老师讲的gitlab runner不知道生产环境是否会有问题。**

**下载gitlab-runner的docker镜像**

**gitlab-runner不是单独给java做持续集成的，所以我们要以gitlab-runner作为基础镜像编写自己的Dockerfile，把java，maven等也装上才行。**

**给一下代码的具体事例吧**







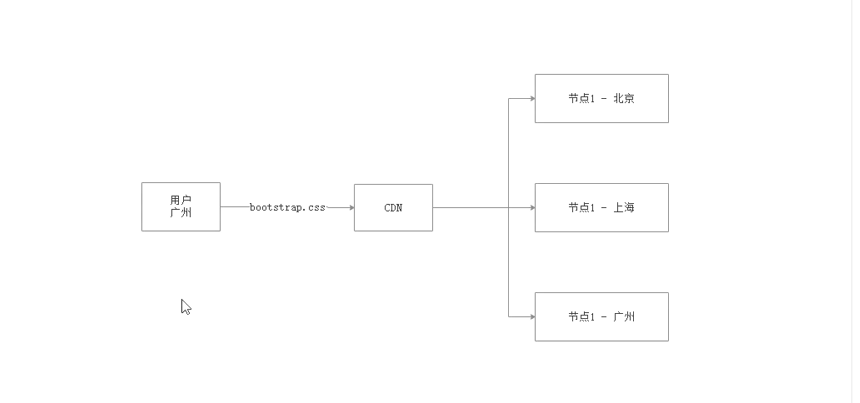
这种方式只要提交代码就会自动构建，不知道代码写错了会怎么样。

下面是实战部分

首先项目基本还是前面的建立方式，

要在feign项目里增加thyeleaf（实际项目中会使用vue等前台框架的，就是跟服务消费者feign项目彻底隔离了，不知道老师为什么讲解这种消费者增加thyeleaf的前台界面功能，而且后期我们有多个feign项目以后，这些界面功能就分散了，还不知道怎么能重新捏合到一起去）。

小知识点（CDN）

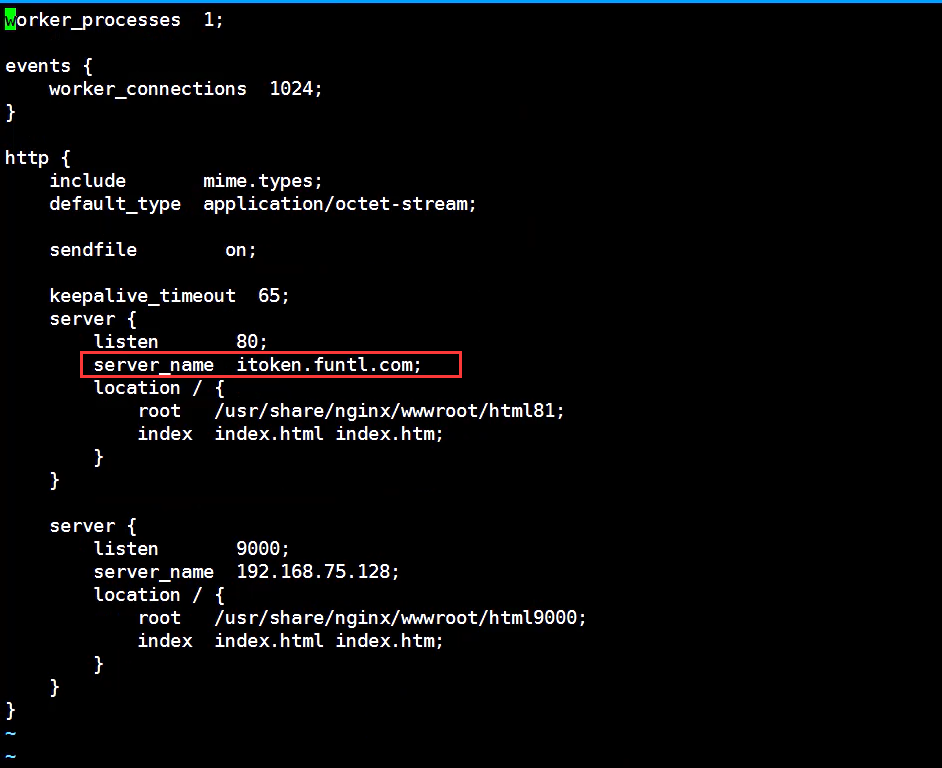


CDN一般不会自己搭建，成本太高，要在全国各地部署节点，所以一般会有CDN服务厂商，这里老师用nginx模拟CDN厂商来做，实际项目不会这么干。

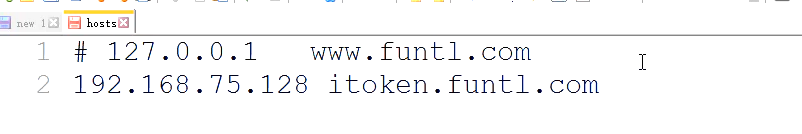


怎么用nginx虚拟主机模拟域名呢？分为两个步骤

1. 在nginx配置文件中server节点里配置域名



1. 在host中增加模拟



反向代理

这里先学一个翻墙软件 <https://www.updatestar.com/directdownload/shadowsocks/2431489>

自己学习吧，现在没工夫

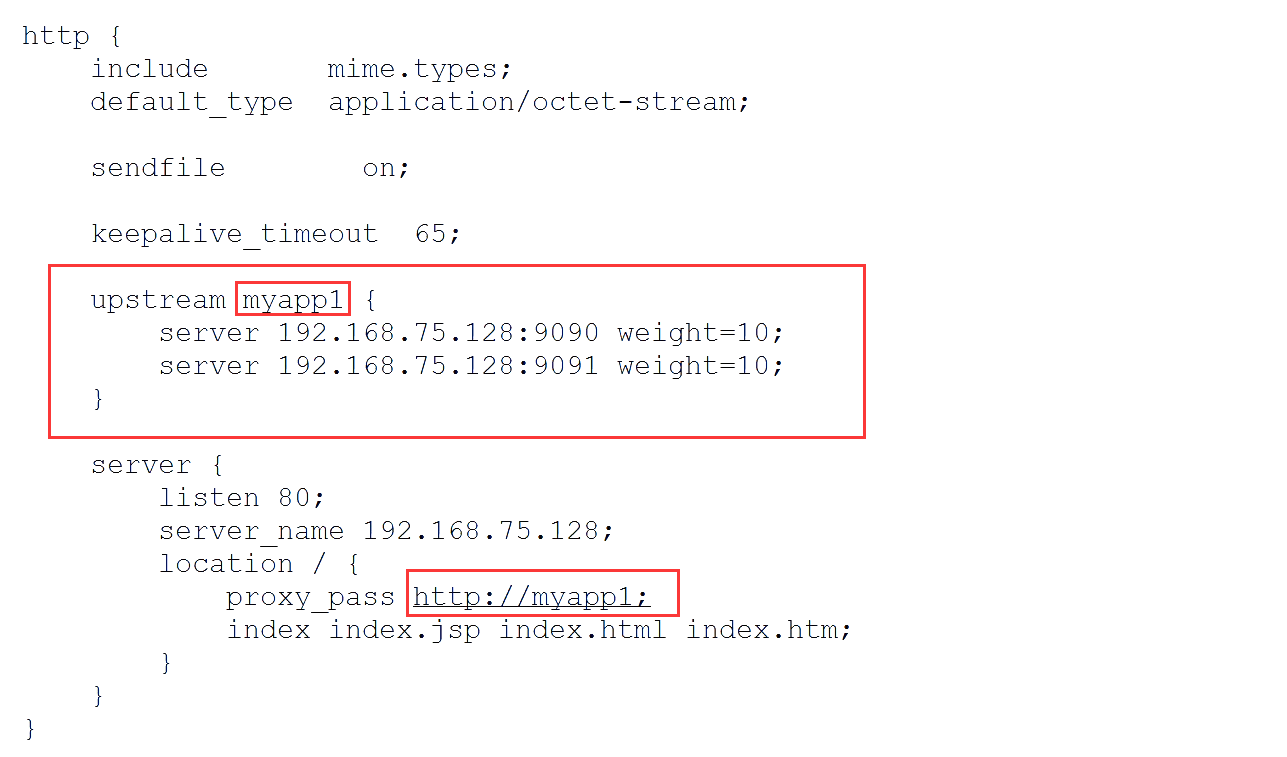
**反向代理和正向代理的区别？**

正向代理是客户端安装代理软件，由代理软件比如shadowsocks去访问服务器（比如google）

反向代理是服务器端代理要去真正访问的网址，客户端只要请求代理服务器即可(nginx)



负载均衡



redis哨兵模式 <https://www.funtl.com/zh/spring-cloud-itoken-codeing/Redis-Sentinel-%E9%9B%86%E7%BE%A4%E9%83%A8%E7%BD%B2.html#搭建-sentinel-集群>

## **搭建 Redis 集群**

搭建一主两从环境，docker-compose.yml 配置如下：

|  |
| --- |
| version: '3.1'  services:  master:  image: redis  container\_name: redis-master  ports:  - 6379:6379  slave1:  image: redis  container\_name: redis-slave-1  ports:  - 6380:6379  command: redis-server --slaveof redis-master 6379  slave2:  image: redis  container\_name: redis-slave-2  ports:  - 6381:6379  command: redis-server --slaveof redis-master 6379 |

## **搭建 Sentinel 集群**

我们至少需要创建三个 Sentinel 服务，docker-compose.yml 配置如下：

|  |
| --- |
| version: '3.1'  services:  sentinel1:  image: redis  container\_name: redis-sentinel-1  ports:  - 26379:26379  command: redis-sentinel /usr/local/etc/redis/sentinel.conf  volumes:  - ./sentinel1.conf:/usr/local/etc/redis/sentinel.conf  sentinel2:  image: redis  container\_name: redis-sentinel-2  ports:  - 26380:26379  command: redis-sentinel /usr/local/etc/redis/sentinel.conf  volumes:  - ./sentinel2.conf:/usr/local/etc/redis/sentinel.conf  sentinel3:  image: redis  container\_name: redis-sentinel-3  ports:  - 26381:26379  command: redis-sentinel /usr/local/etc/redis/sentinel.conf  volumes:  - ./sentinel3.conf:/usr/local/etc/redis/sentinel.conf |

### **修改 Sentinel 配置文件**

需要三份 sentinel.conf 配置文件，分别为 sentinel1.conf，sentinel2.conf，sentinel3.conf，配置文件内容相同

|  |
| --- |
| port 26379  dir /tmp  # 自定义集群名，其中 127.0.0.1 为 redis-master 的 ip，6379 为 redis-master 的端口，2 为最小投票数（因为有 3 台 Sentinel 所以可以设置成 2）  sentinel monitor mymaster 127.0.0.1 6379 2  sentinel down-after-milliseconds mymaster 30000  sentinel parallel-syncs mymaster 1  sentinel failover-timeout mymaster 180000  sentinel deny-scripts-reconfig yes |

### **查看集群是否生效**

进入 Sentinel 容器，使用 Sentinel API 查看监控情况：

docker exec -it redis-sentinel-1 /bin/bash

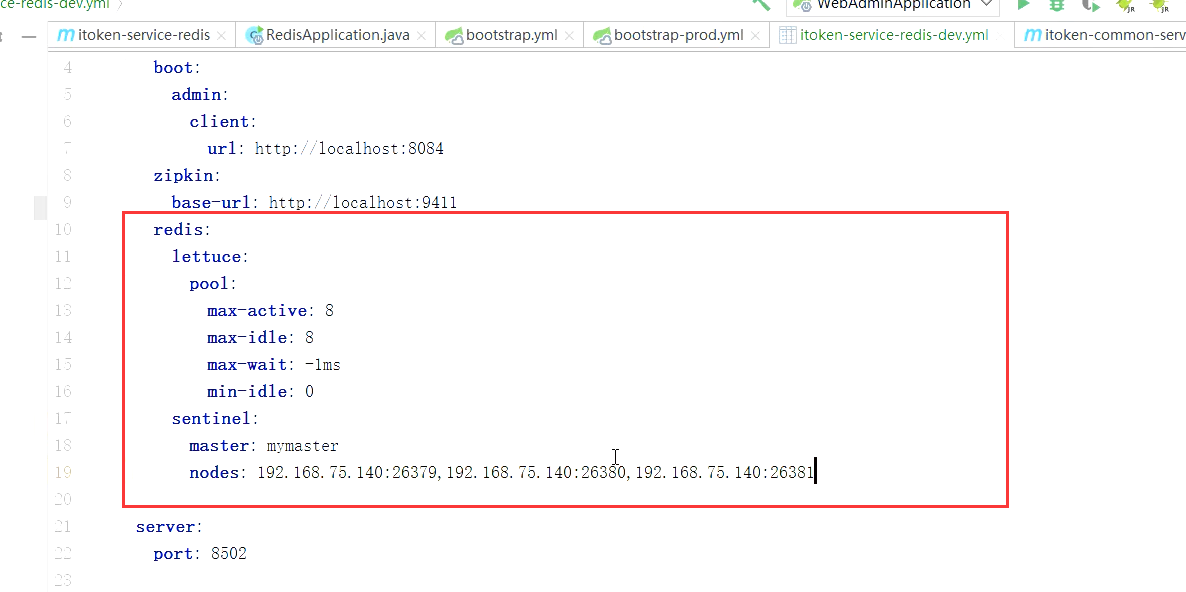
redis-cli -p 26379

sentinel master mymaster

sentinel slaves mymaster

代码

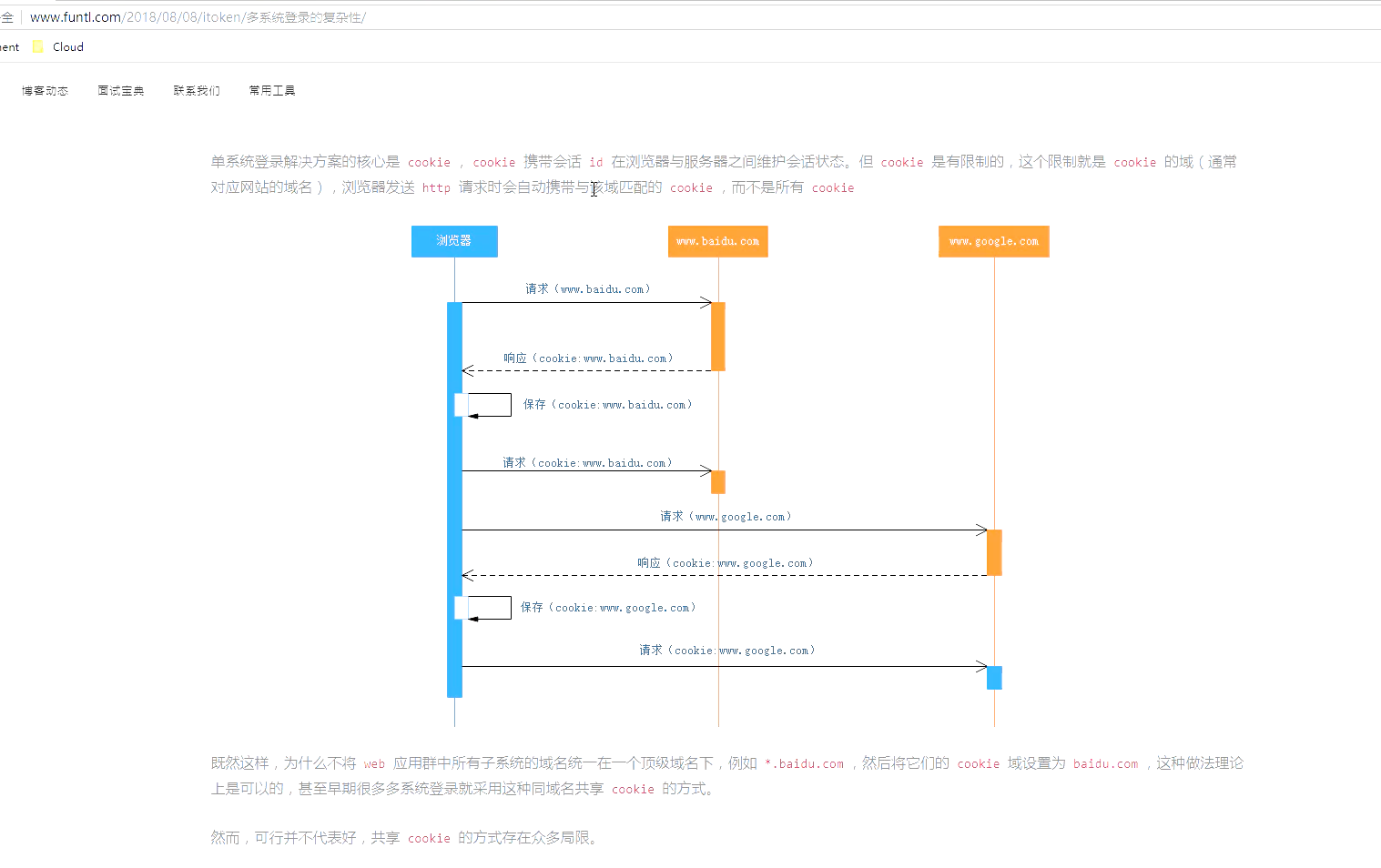






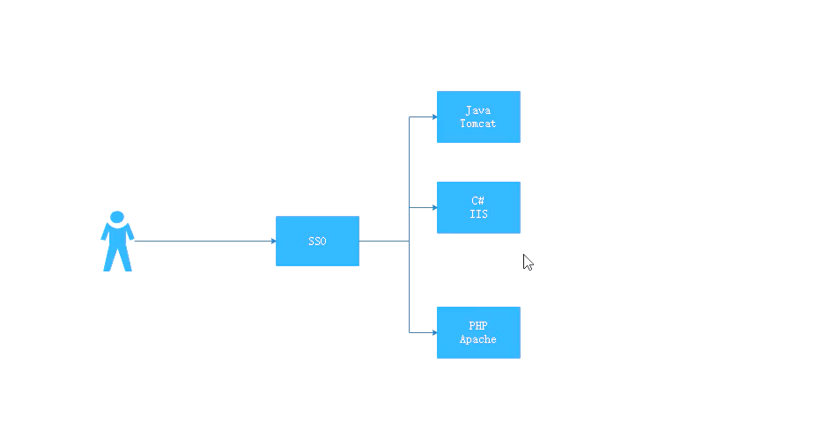
单点登录（redis实现）

仔细看看文章关于cookie和JSESSIONID的讲解





单点登录保证跨语言



单点登录需要认证中心