1.消费者调用服务者

（1）将RestTemplate注入spring

@Bean

public RestTemplate restTemplate(){

return new RestTemplate();

}

（2）使用注入的对象访问服务者

@Autowired

private RestTemplate restTemplate;

@getMapping("/hello/{id}")

public User findById(@PathVariable Long id){

return restTemplate.getForObject("http: //localhost:8080/hello/1", User.class);

}

2.Eureka 服务发现组件

（1）why Eureka?

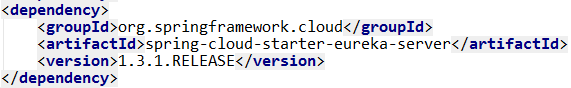
eureka来自开发环境，springcloud支持

（2）心跳机制 健康检查 客户端缓存

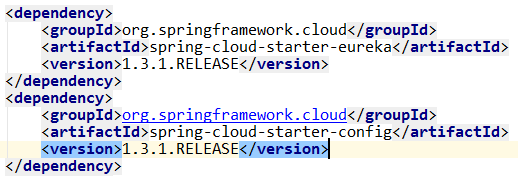
（3）demo

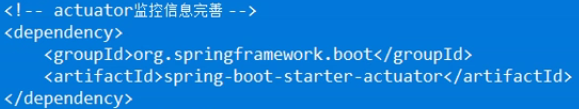
1）引入依赖jar包

服务端：



client端：





2）启动类

client端：

@EnableEurekaClient

服务端：

@EnableEurekaServer

3）配置文件

服务端：



client端：



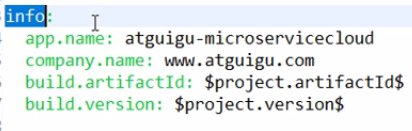
问题：在192.168.1.1:8001/info出现内容

解决：1.在client项目中添加依赖：spring-boot-start-actuator

2.在总父项目的pom中添加build



3.在client项目中配置info



3.Eureka的自我保护

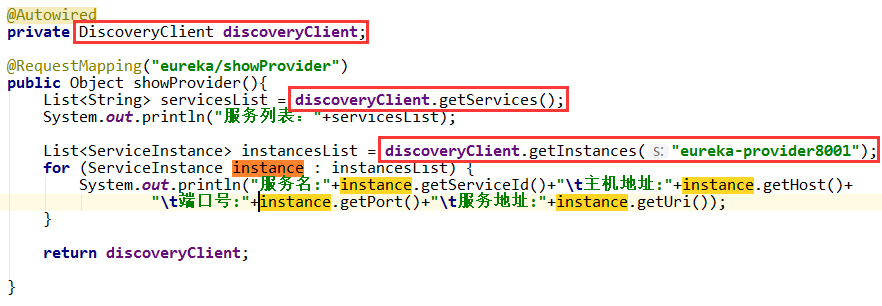
应用即使关闭（或长时间未连接），也不会立即清理，会对应用信息进行保存，直到原应用再开启（默认90秒）

**自我保护模式中，Eureka Server会保护服务注册表中的信息，不在注销任何服务实例。当它收到的心跳数量重新恢复到阈值以上时，该Eureka Server节点就会自动退出自我保护模式。**

4.Eureka服务发现

（1）启动类加上 @EnableDiscoveryClient *//eureka 服务发现*

（2）controller层 服务发现代码

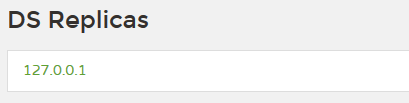


打印输出：

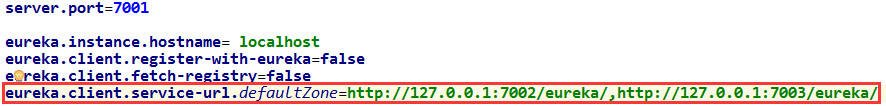


5.eureka集群

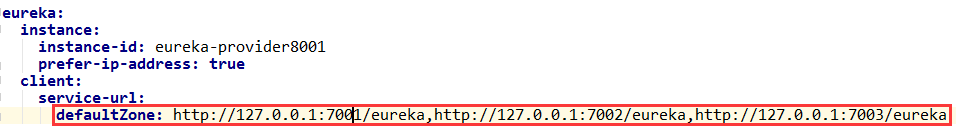
（1）集群要有不同的ip/域名，如果相同，则只能显示一个



（2）server的配置：与其他服务关联



（3）provider的配置：发布到所有的服务上



6.eureka遵循AP原则

关系型数据库 ACID

A atomicity 原子性

C consistency 一致性

I isolation 独立性

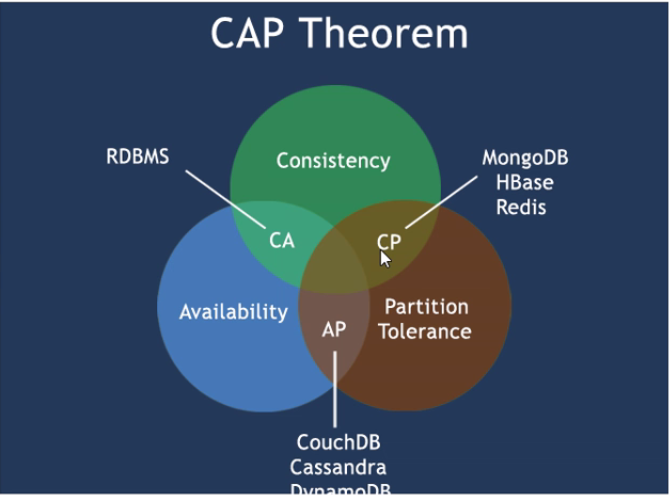
D durability 持久性

非关系型数据库 CAP

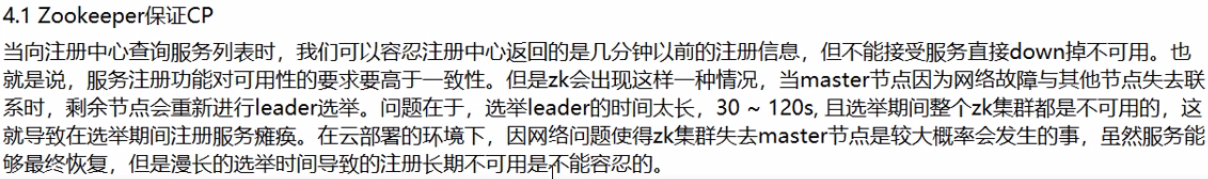
C consistency 强一致性

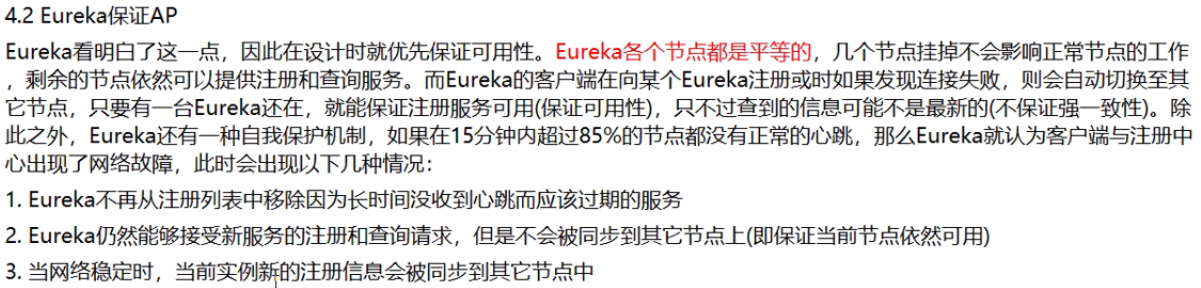
A availability 可用性

P partition tolerance 分区容错性



7.eureka和zookeeper的区别



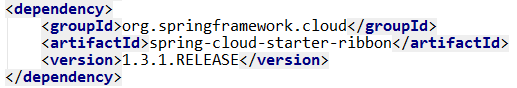


8.ribbon负载均衡

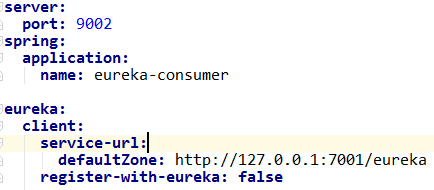
是一种客户端的负载均衡

（1）负载均衡配置

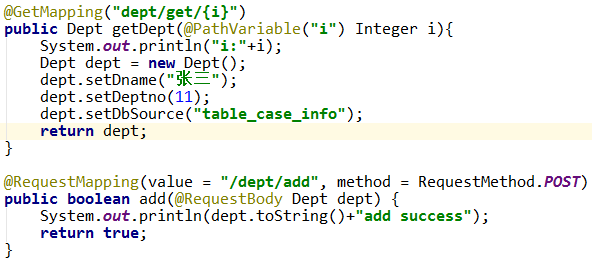
导包：



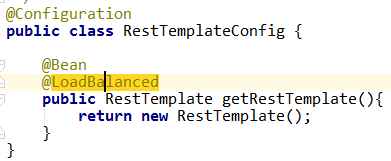
consumer配置：

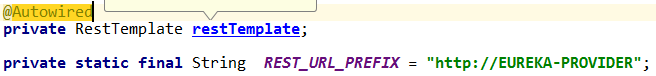


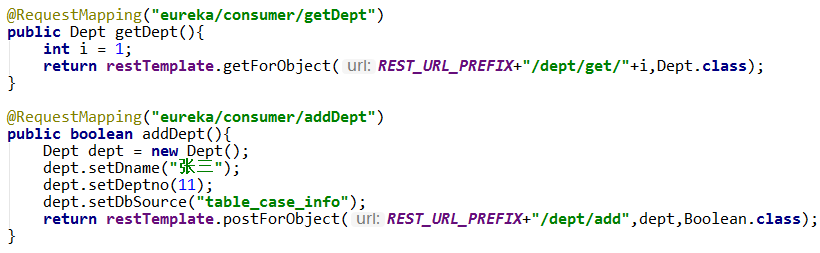
provider：



consumer引用provider





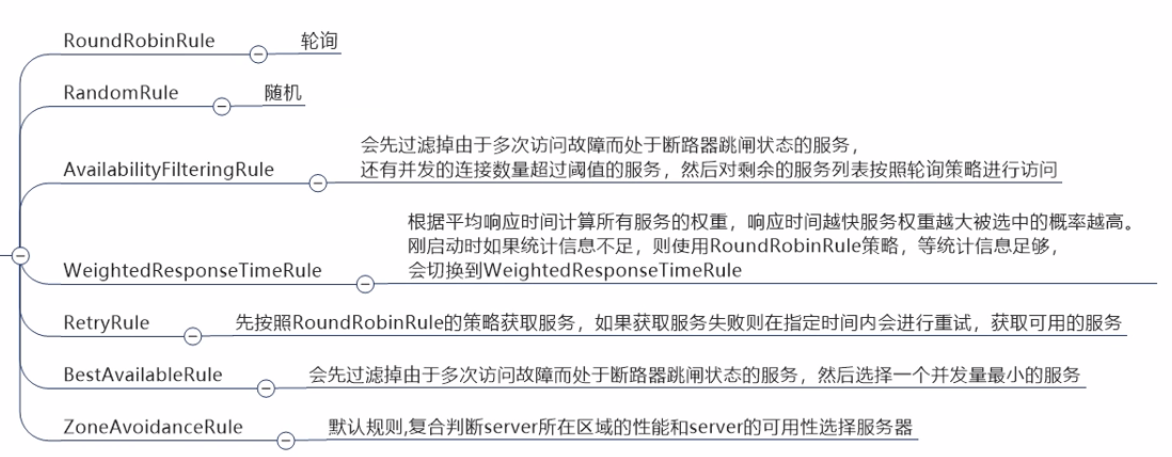


（2）ribbon负载均衡实现

要点：服务提供者的 spring.application.name 必须一致，这样可以使用 http://实例名 来访 问

总结：ribbon是一个软负载均衡的客户端（provider）组件

（3）Ribbon核心组件IRule



实现方式：



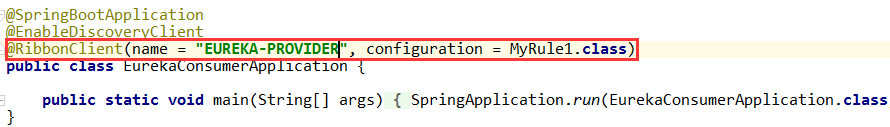
（4）自定义负载均衡算法

源码：

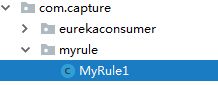
https://github.com/Netflix/ribbon/blob/master/ribbon-loadbalancer/src/main/java/com/netflix/loadbalancer

例子：给某一个实例里面的所有服务添加自定义算法

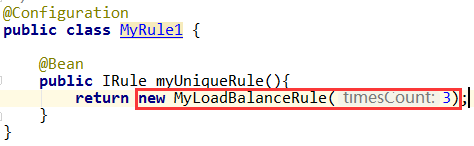
启动类：



启动中的配置类（MyRule.class）不能被启动类扫描：



在配置类中，实现自己的算法类（每3次在访问下一个）

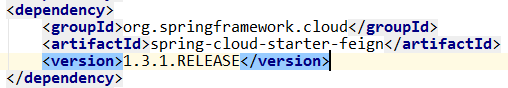


继承AbstractLoadBalancerRule

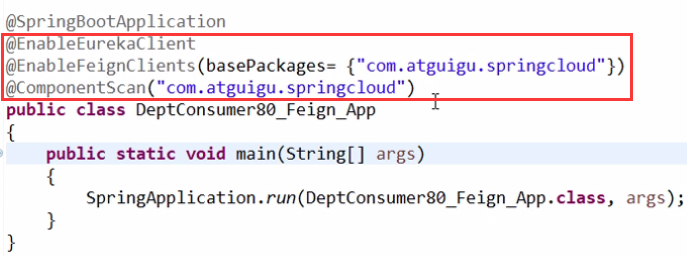


9.feign负载均衡（通过注解和微服务来实现ribbon负载均衡）

（1）导入jar包



（2）主启动类

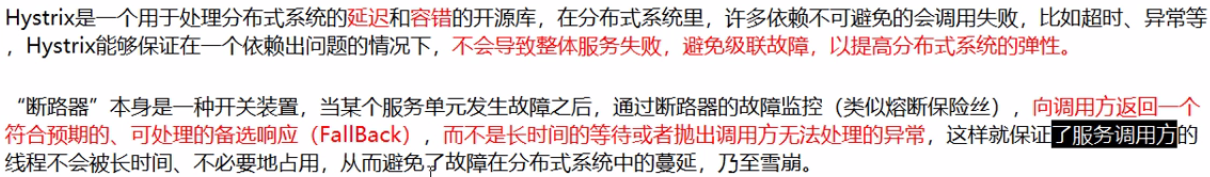


（3）定义接口



10.Hystrix断路器

（1）简介



（2）官网源码

https://github.com/Netflix/Hystrix/wiki/How-To-Use

（3）导入jar包



（4）主启动类



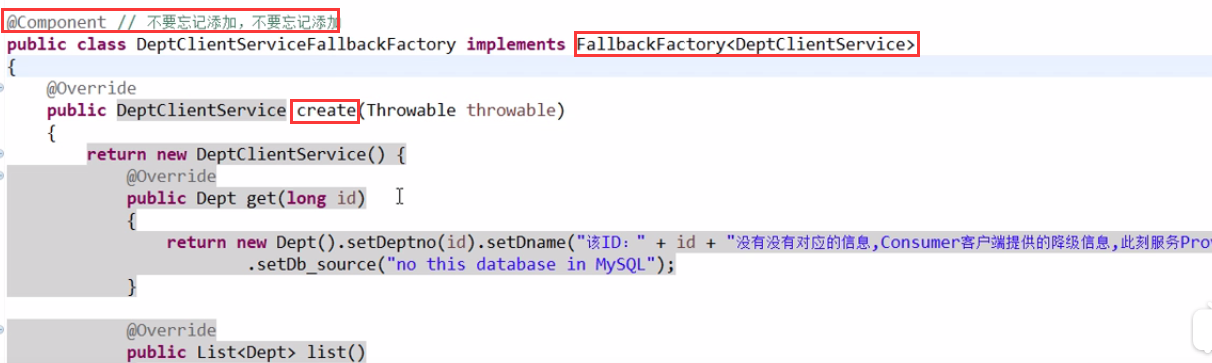
（5）在controller层实现



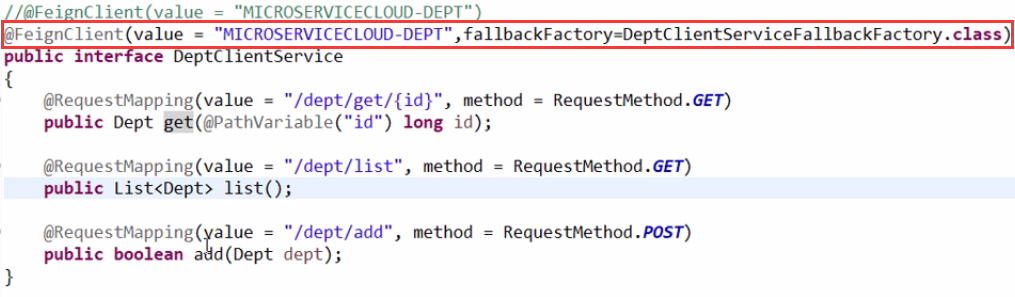
11.springcloud 服务降级（provider端）

（1）如果程序出现问题，就使用plan B（使用下一个方案）

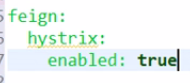
（2）实现：将service层放入FallbackFactory中，重写里面的方法（如果原service中方法异常，则使用这里的方法，即给出了程序异常处理方案）



（3）原service层：添加注解属性，声明如果有异常就找fallbackFactory的实现类

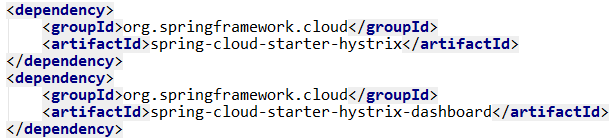


（4）配置文件中配置

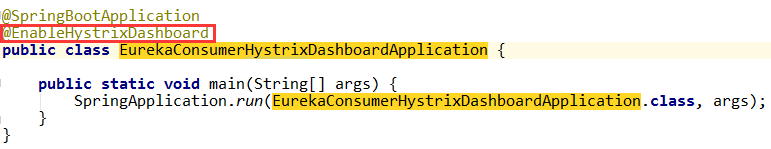


12.Hystrix Dashboard 监控微服务健康状况

（1）导入jar包



（2）主启动类



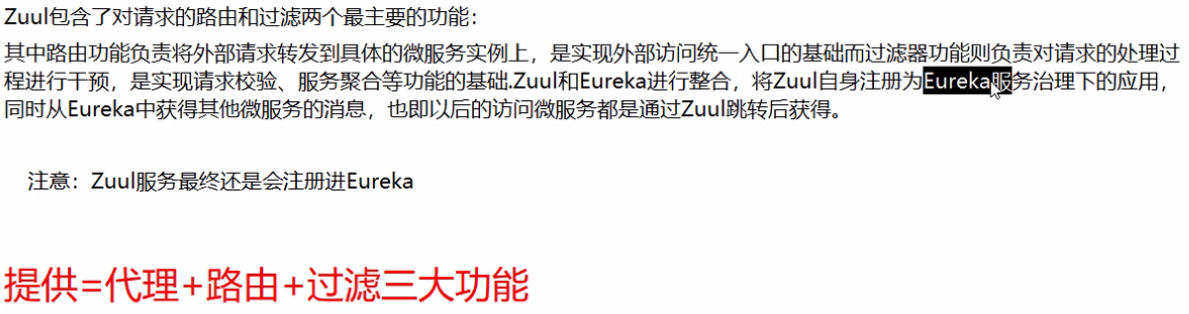
（3）localhost:9003/hystrix

（4）

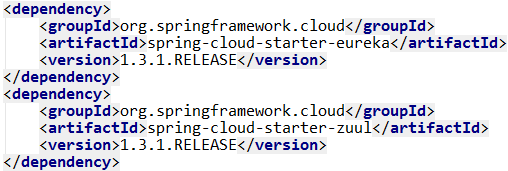
来监控服务，可视化

13.zuul路由网关

（1）是什么



（2）导入jar包



（3）配置文件配置

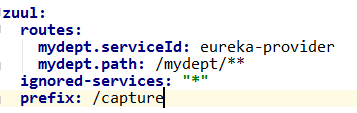


（4）访问地址

zuul地址（端口号）+服务名称+方法路径

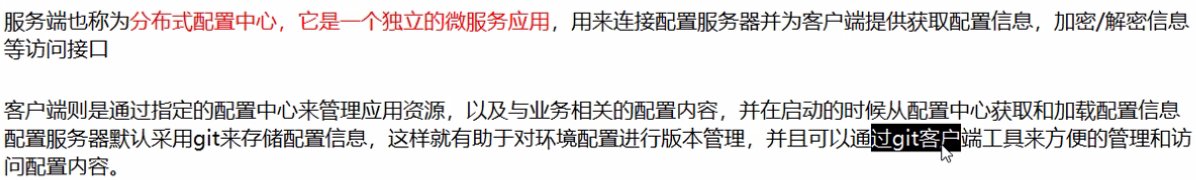


（5）隐蔽服务名称

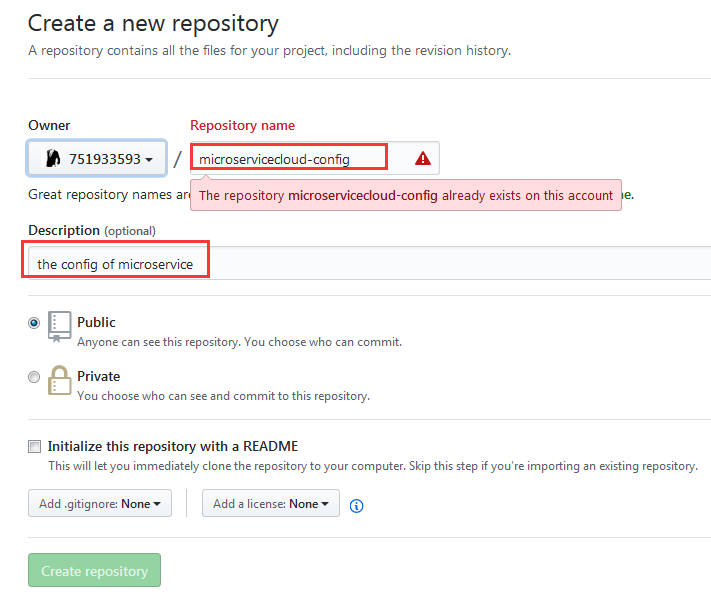


隐蔽后的唯一访问地址：localhost:7002/capture/mydept/dept/get/1

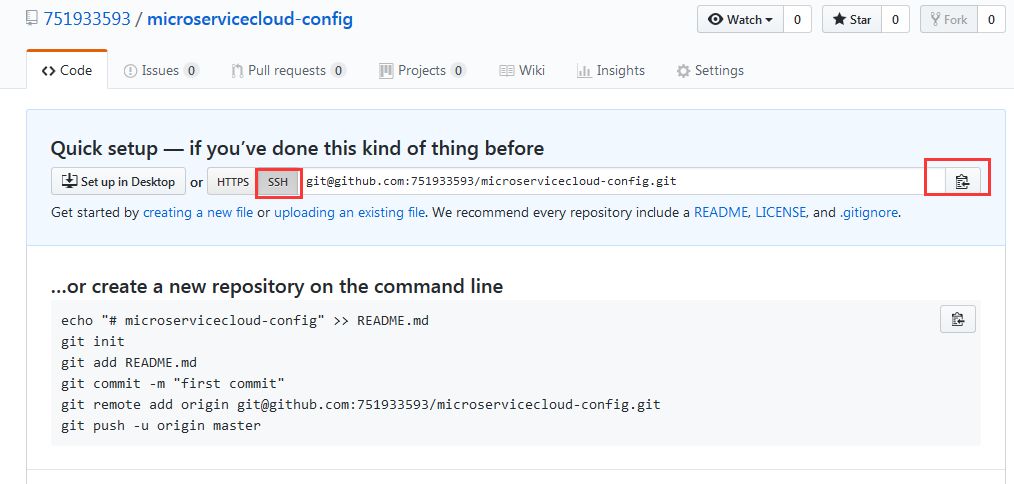
14分布式配置中心



（1）在github上 new repository



（2）获取基于ssh的repository地址

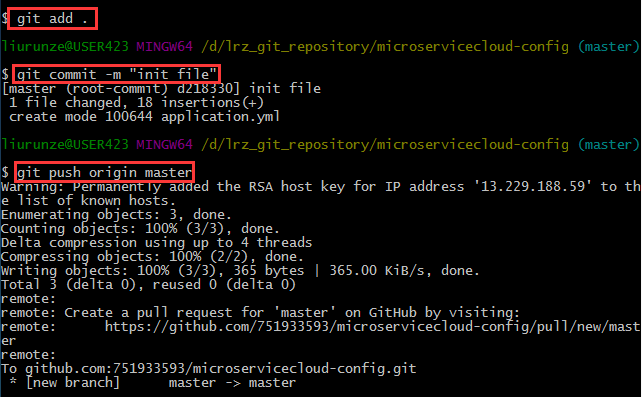


（3）本地新建git仓库并clone

git clone git@github.com:751933593/microservicecloud-config.git

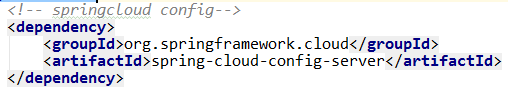
（4）在对应目录新建application.yml文件

（5）将application.yml上传至GitHub

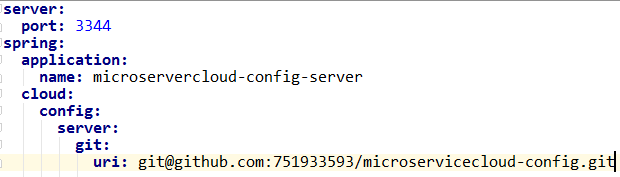


**服务端配置**

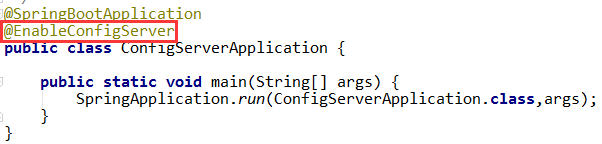
（7）pom.xml



（8）application.yml



（9）主启动类



（10）项目运行后访问：

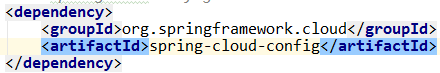
http://localhost:3344/ application-test.yml

http://localhost:3344/application/test/master

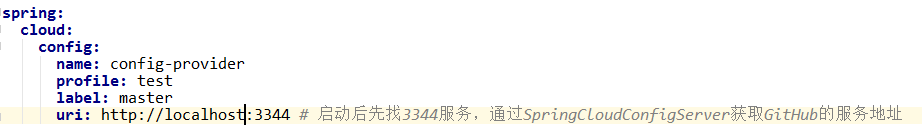
http://localhost:3344/master/ application-test.yml

**客户端配置**

（11）pom.xml



（12）application.yml



（13）项目运行后访问：

配置文件中配置dev：http://localhost:8201/getConfig

配置文件中配置dev：http://localhost: 8202/getConfig