1. SpringCloud：分布式微服务的一站式解决方案，是多种微服务架构落地技术及集合体。
2. 声明父工程步骤：

①<packaging>pom</packaging>

②<dependencyManagement></dependencyManagement>：为了父工程进行统一依赖版本。

1. 问题：创建父工程下的子模块项目（springboot子模块）时，必须要从New Project -> Spring Initializr下创建springboot模块。

然后调整子模块的pom.xml，加入所属父工程的信息：

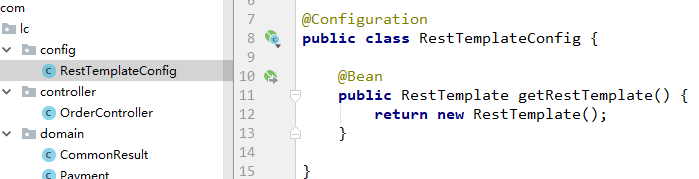
<parent>  
 <groupId>com.lc</groupId>   
 <artifactId>springcloud2021</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </parent>

最后在父工程的pom.xml下引入子模块的信息：

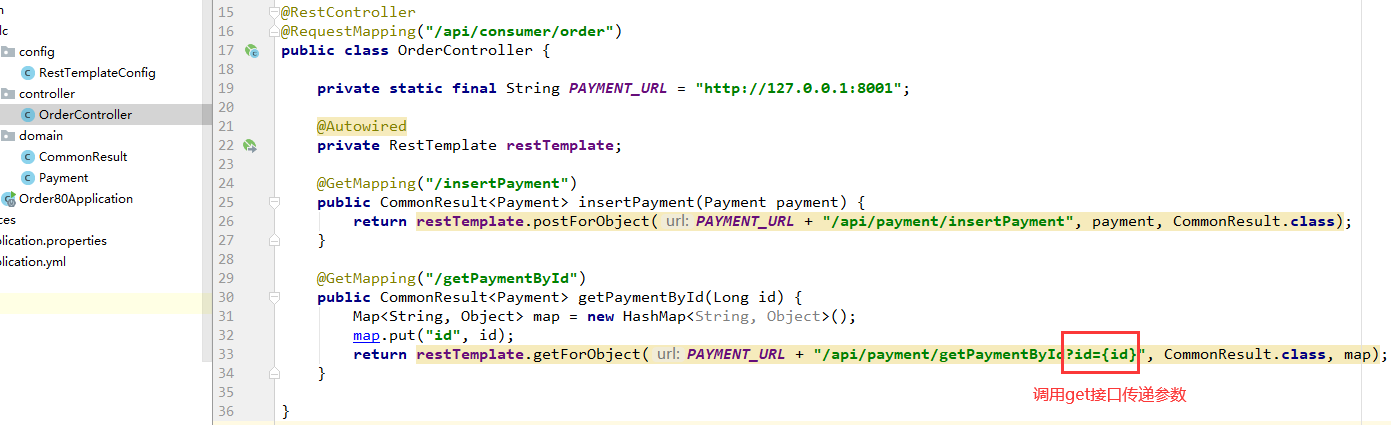
<modules>  
 <module>payment8001</module>  
 </modules>

1. domain当前构造方法调用其他构造方法用this(参数.....)即可。
2. 客户端调用服务端RestTemplate使用：当中封装了HttpClient，提供了访问远程http服务的方法，便捷访问restful服务接口，使用：

①将RestTemplate注入到spring容器中：



②使用RestTemplate调用服务端接口：



注：调用服务端post接口传递对象，参数接口必须要加@RequestBody。

调用服务端get接口传递属性，参数接口必须要加@RequestParam。

1. 扩展（待试）：多个微服务自动调用出idea的Run Dashboard

找到idea项目下 -> .idea文件夹 -> workspace.xml -> 内容在<component name="RunDashboard">下添加内容 -> 关闭idea重启

内容：<option name="configurationTypes">

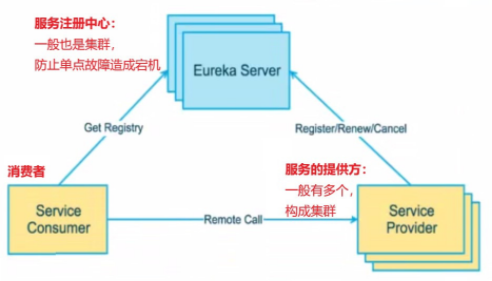
<set>

<option value="SpringBootApplicationConfigurationType" />

</set>

</option>

1. 抽离公共部分代码生成项目 -> 编译打包 -> 其他项目中删除公共代码并引入依赖即可。
2. Eureka：服务注册中心，服务的注册与发现。

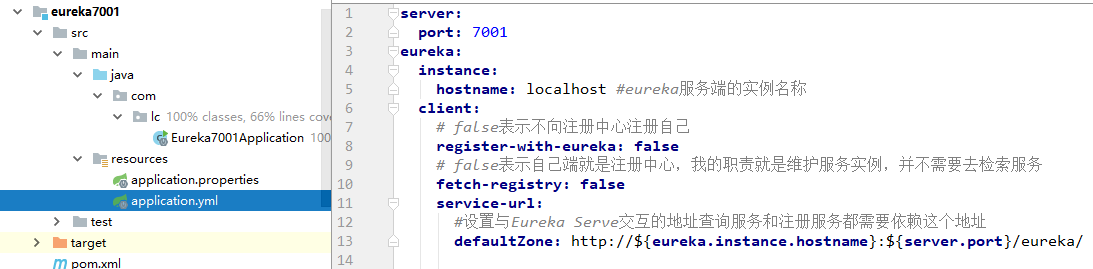


1. Eureka：开启新模块引入注册中心：

① 导入注册端jar包

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
</dependency>

② 设置单台Eureka的application.yml:



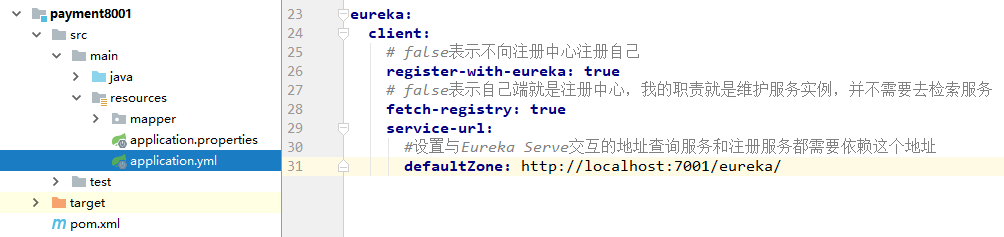
③ 主启动类上添加注解：@EnableEurekaServer

1. Eureka：提供端需注册服务配置：

① 导入提供端jar包：

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
</dependency>

② 设置单台提供者的application.yml:



③ 主启动类上添加注解：@EnableEurekaClient

④ 访问地址：<http://localhost:7001> 不带/eureka，查看服务注册内容

1. Eureka：配置注册中心集群，相互注册，相互守望

① 因多个注册中心配置的hostname都是localhost，所以需要在hosts文件中配置：

127.0.0.1 eureka7001.com

127.0.0.1 eureka7002.com

② 配置注册中心application.yml文件



③ 配置提供端application.yml集群模式



12. Eureka：配置服务端集群模式，除端口号不同外，都一致，客户端访问时，需要将地址路径改为服务端的服务名称，配置RestTemplate时需要加负载均衡注解，获取指定服务器提供服务。



13. Eureka：修改提供端在注册中心的名称及点击显示ip信息；

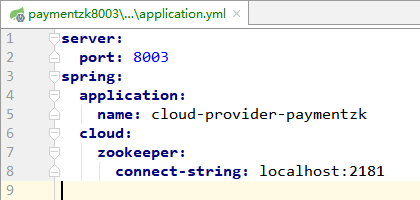
**eureka:  
 instance:  
 instance-id:** payment8001  
 **prefer-ip-address: true**

1. Zookeeper：使用zk替代Eureka作为注册中心

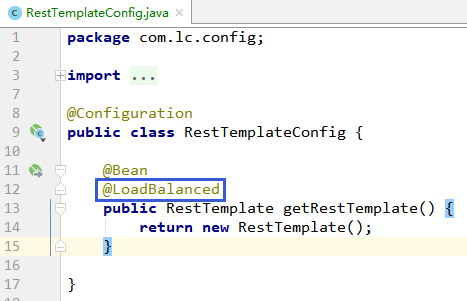
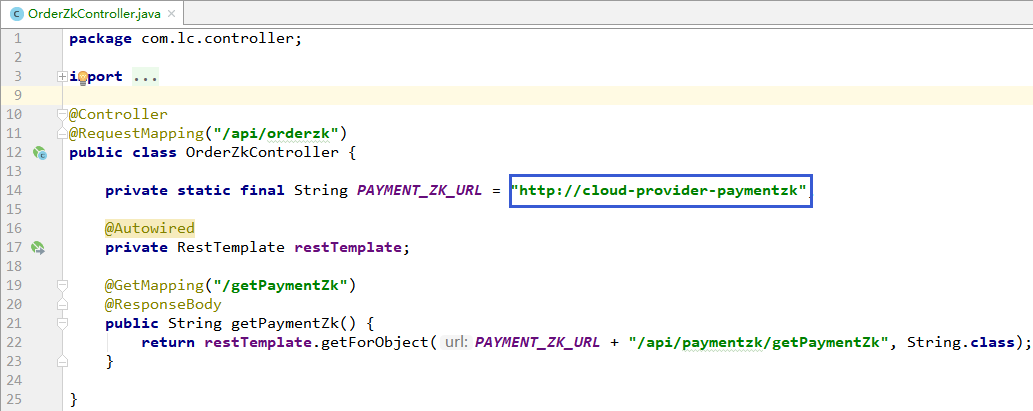
① 启动zk控制台，项目导入jar包依赖，需要注意控制版本；

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-zookeeper-discovery</**artifactId**>  
 *<!--先排除自带的zookeeper3.5.3-->* <**exclusions**>  
 <**exclusion**>  
 <**groupId**>org.apache.zookeeper</**groupId**>  
 <**artifactId**>zookeeper</**artifactId**>  
 </**exclusion**>  
 </**exclusions**>  
</**dependency**>  
*<!--添加zookeeper3.4.9版本-->*<**dependency**>  
 <**groupId**>org.apache.zookeeper</**groupId**>  
 <**artifactId**>zookeeper</**artifactId**>  
 <**version**>3.4.9</**version**>  
</**dependency**>

② 调用端和服务提供端配置application.yml，注入到zk中



③ 调用端再需要配置RestTemplate，远程进行调用服务端接口。



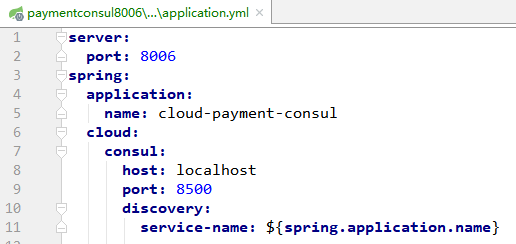
1. Consul：使用consul注册中心

①下载并执行consul.exe文件，在解压目录下，cmd->consul --version查看版本，使用开发模式启动consul：consul agent -dev，地址localhost:8500可以查看consul注册管理页面。

②调用端和服务提供端导入jar包依赖：

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-consul-discovery</**artifactId**>  
</**dependency**>

③调用端和服务提供端配置application.yml：



④调用端配置RestTemplate进行远程调用服务提供端，服务名称调用注意添加注解@LoadBalanced。

1. Eureka、Zookeeper和Consul的区别：

Eureka保证ap，保证可用性和分区容错性；

Zookeeper保证cp，保证一致性和分区容错性；

Consul保证cp，保证一致性和分区容错性；

1. Ribbon：功能服务端调用，负载均衡（默认轮询方式） + RestTemplate服务调用；
2. Ribbon：使用ribbon，详细介绍使用RestTemplate服务调用：

①如果存在Eureka并自带ribbon的jar包可以使用现有的，不存在引入jar包：

**<dependency>**

**<groupld>org.springframework.cloud</groupld> <artifactld>spring-cloud-starter-netflix-ribbon</artifactid>**

**</dependency>**

②RestTemplate方法getForObject和getForEntity的区别：getForObject返回的json串，getForEntity返回ResponseEntity对象，包含响应头，响应体等具体信息；

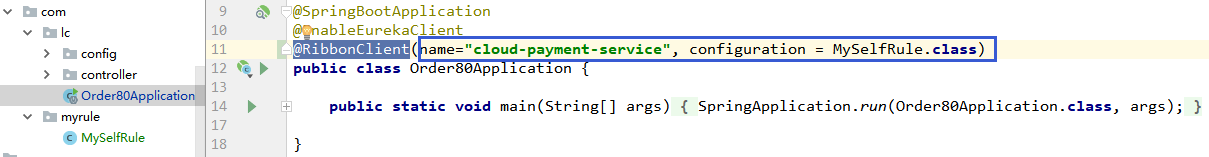
1. Ribbon：将默认的轮询算法改为随机算法服务调用：

①在服务调用端实现，切记更改的负载算法不能在@ComponentScan所扫描的当前包下以及子包下；

②改为随机算法方式：



③在主启动类上使用注解@RibbonClient并表明调用的服务提供端及使用的算法：

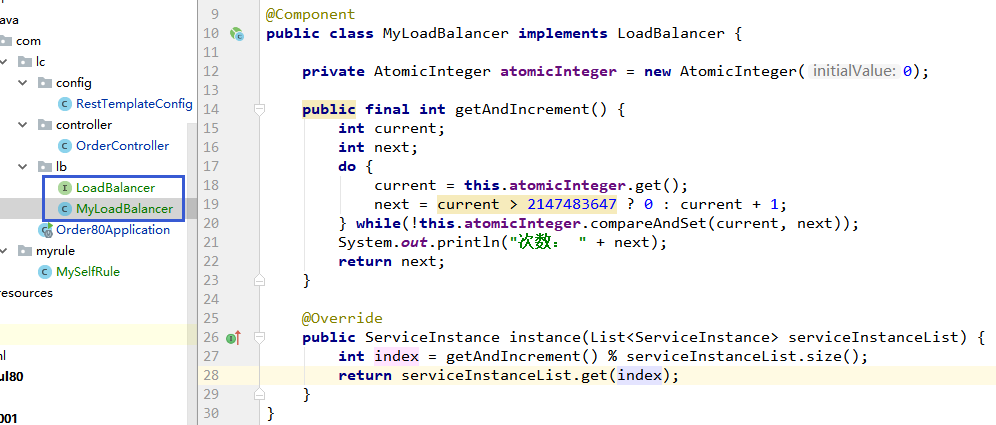


1. Ribbon：手写轮询算法

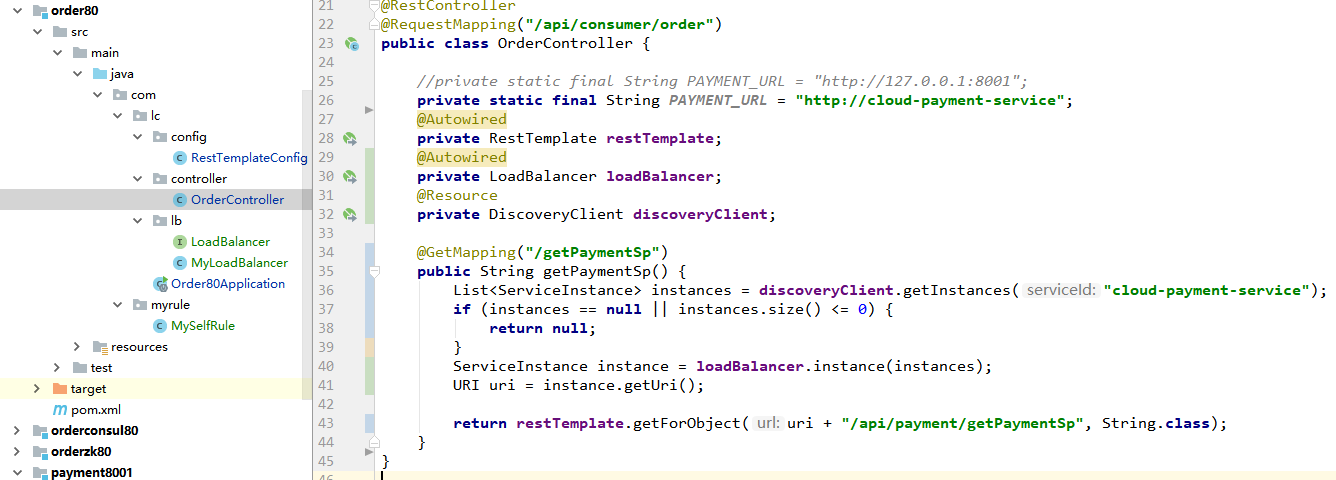
①原理：rest接口第几次请求数 % 服务器集群总数量 = 实际调用服务器位置下标；

②去除Ribbon自带的@LoadBalanced负载注解；

③定义接口和实现类，构造自定义轮询算法；



④调用服务提供端接口：



1. OpenFeign：功能服务端调用，服务接口绑定器，接口 + 注解；
2. OpenFeign：使用流程：

①导入jar包依赖，其中包含了ribbon：

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

②要使用OpenFeign服务，在主启动类上添加注解@EnableFeignClients;

③添加调用接口，接口上添加@FeignClient(value=”服务提供者名称”)，接口方法要求和服务提供端Controller层一致，注：传递的是单个属性必须要加@RequestParam(value=”id”)，调用的地址是服务提供端的全路径地址，不需要@ResponseBody注解，服务提供端接受的是对象需要加@RequestBody，参数需要加@RequestParam(value=””);



23. OpenFeign：服务调用端默认只等待一秒钟，但服务端处理超过一秒钟，继续等待需要设置配置项：

*#设置feign客户端超时时间(OpenFeign默认支持ribbon)***ribbon:** *#指的是建立连接后从服务器读取到可用资源所用的时间*

**ReadTimeout:** 5000  
 *#指的是建立连接所用的时间，适用于网络状况正常的情况下,两端连接所用的时间* **ConnectTimeout:** 5000

24. OpenFeign：开启日志功能：

①定义配置类：



②在application.yml定义配置项：

**logging:  
 level:** *# feign日志以什么级别监控哪个接口（openFeign接口类的全路径名）* **com.lc.service.PaymentFeignService:** debug

25. Hystrix：处理分布式系统的延迟和容错，不会导致整理服务失败，避免级联故障，以提高分布式系统的弹性。

26. Hystrix：重要概念：

①服务降级（FallBack）：向调用方返回一个符合预期的、可处理的备选响应，而不是长时间的等待或者抛出调用方无法处理的异常。使用场景：程序运行异常、超时、服务熔断触发服务降级、线程池满也会导致服务降级。

②服务熔断（break）：直接拒绝访问，再服务降级的方法并返回友好提示。服务降级 ->进而熔断 -> 恢复调用链路。

③服务限流（flowlimit）：秒杀高并发等操作，一秒钟N个，有序进行。

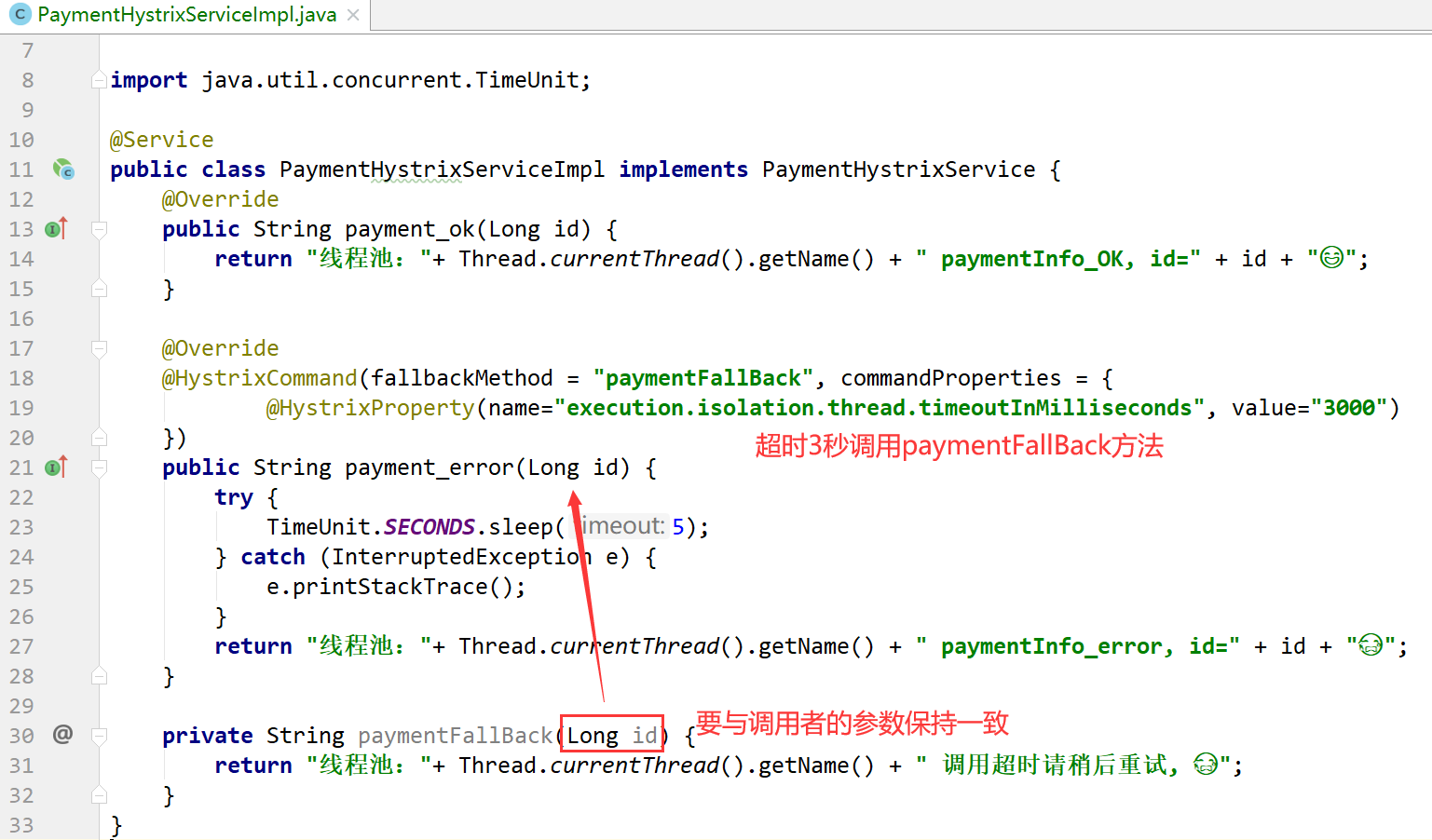
27. Hystrix：服务提供端使用Hystrix，适用于超时和异常情况：

①导入Hystrix的jar包依赖：

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-hystrix</**artifactId**>  
 </**dependency**>

②主启动类添加@EnableCircuitBreaker注解；

③使用注解@HystrixCommand定义服务降级方法，注：\*FallBack方法参数要与调用者保持一致：



28. Hystrix：服务调用端使用Hystrix，适用于超时和异常情况:

①导入Hystrix的jar包依赖，在主启动类上添加@EnableHystrix注解;

②在application.yml添加配置项：

**feign:  
 hystrix:  
 enabled: true**

③配置方法的FallBack方法：



29. Hystrix：定义全局的FallBack方法，在类上添加注解@DefaultProperties(defaultFallBack=””);



30. Hystrix：解决服务调用端代码膨胀和混乱，对服务提供端所有方法进行Fallback处理：

①服务调用端实现openFeign对应的接口类，配置fallback方法；

②在接口上@FeignClient注解添加fallback属性及对应的fallback方法类：@FeignClient(value = "cloud-provider-payment-hystrix", fallback = FallBackHystrixService.class)

31. Hystrix：服务提供端实现服务熔断代码，规定次数指定达到设置的错误率就会服务调用失败，慢慢再进行成功调用服务恢复正常，中间正常情况也会出现服务降级：



1. Hystrix：配置图形化管理页面：

①开发新模块导入jar包依赖

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-hystrix-dashboard</**artifactId**>  
 </**dependency**>

②主启动类添加注解@EnableHystrixDashboard

③访问地址：<http://localhost:9001/hystrix>

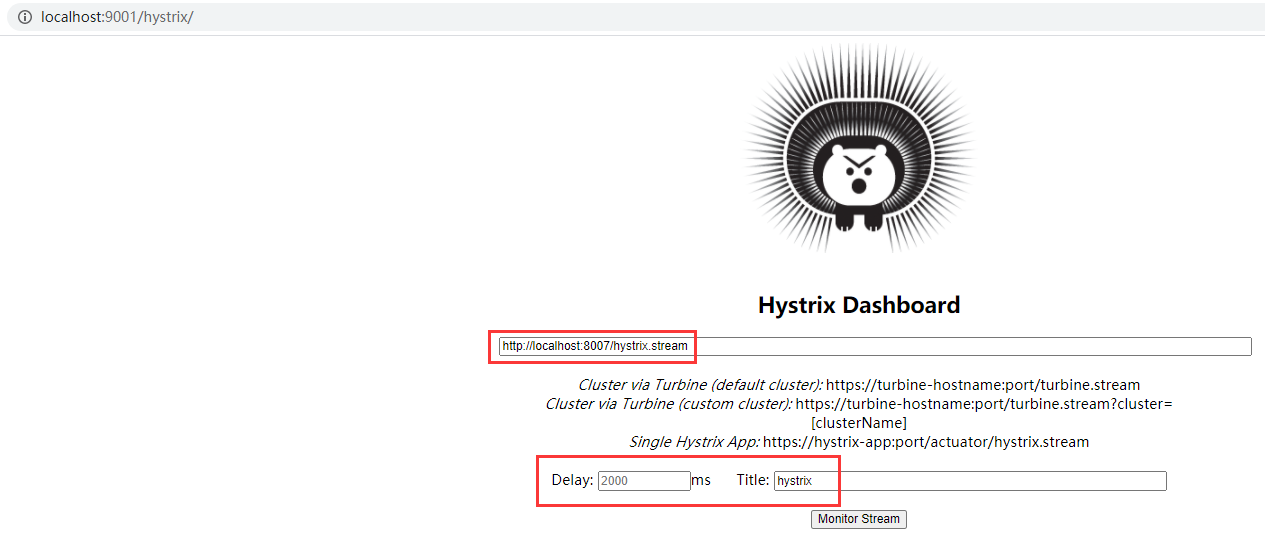
④在被监控的模块中需要导入jar包：

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-actuator</**artifactId**>  
 </**dependency**>

⑤在被监控模块主启动类中引入代码：

@Bean  
**public** ServletRegistrationBean getServlet() {  
 HystrixMetricsStreamServlet streamServlet = **new** HystrixMetricsStreamServlet();  
 ServletRegistrationBean registrationBean = **new** ServletRegistrationBean(streamServlet);  
 registrationBean.setLoadOnStartup(1);  
 registrationBean.addUrlMappings(**"/hystrix.stream"**);  
 registrationBean.setName(**"HystrixMetricsStreamServlet"**);  
 **return** registrationBean;  
}

⑥配置Hystrix监控页面被监控模块地址参数：

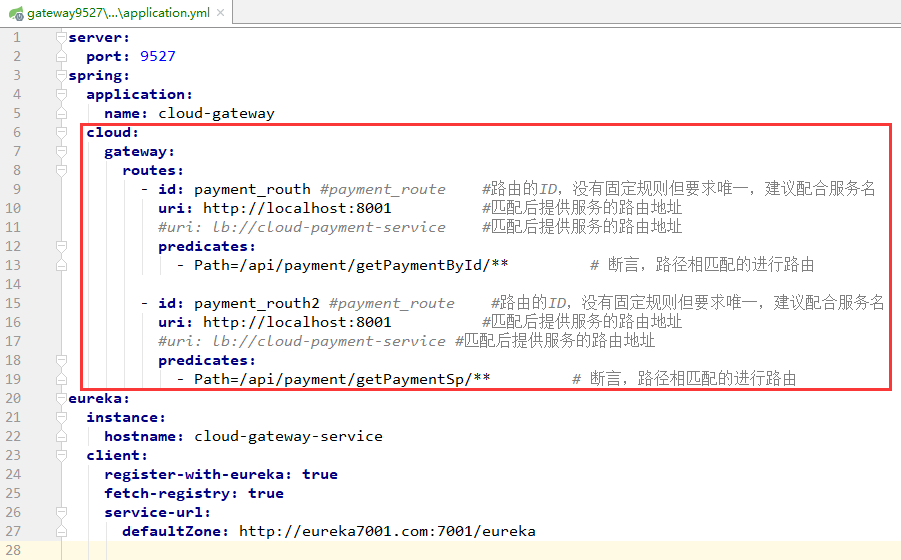


1. Gateway：对API进行路由以及提供一些强大的过滤功能，底层Netty框架（非阻塞异步模型）。
2. Gateway：利用yml开发网关路由：

①导入gateway的jar包依赖，去除web和actuator的jar包；

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-eureka-client</**artifactId**>  
 </**dependency**>

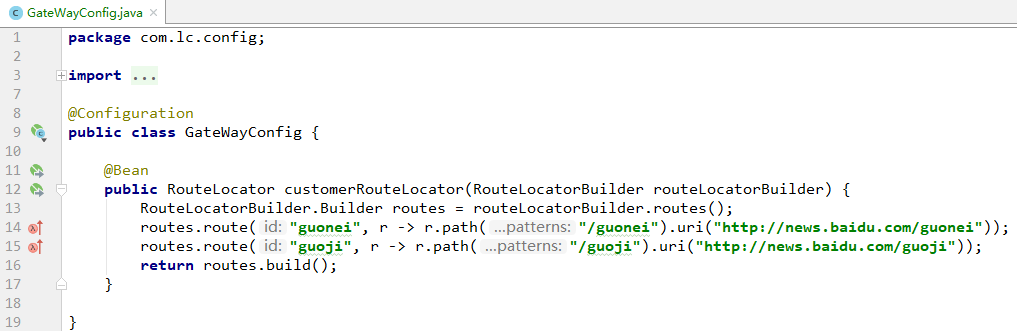
②application.yml配置网关



③地址：[http://localhost:9527/api/payment/getPaymentById，即可路由8001服务器接口；](http://localhost:9527/api/payment/getPaymentById，即可访问8001接口；)

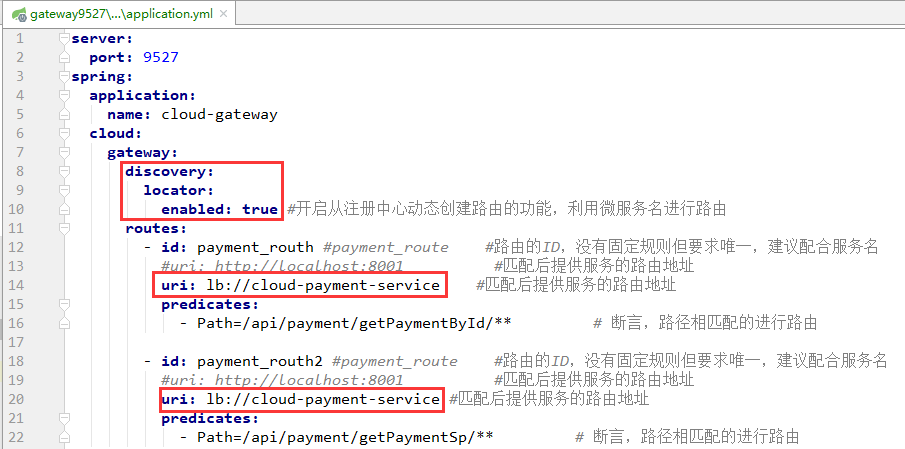
1. Gateway：代码方式实现网关路由：

①添加Gateway配置类：



②地址：<http://localhost:9527/guonei>

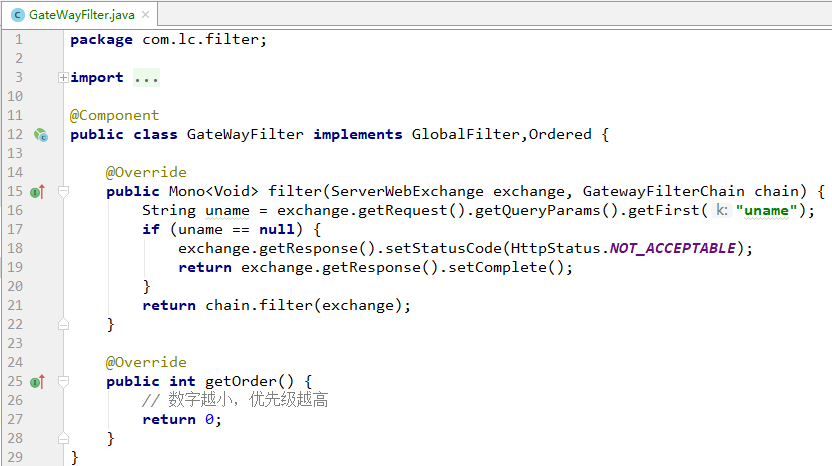
1. Gateway：实现负载均衡，通过服务名进行动态路由：



1. Gateway：predicates可配置的规则，请求对应路由：



1. Gateway：自定义全局Filter过滤器的使用：



1. Config：分布式配置中心，将多个微服务公共配置项抽离成通用的外部配置文件；
2. Config：连接github外部通用配置文件生成一个服务模块：

①导入config的jar包依赖：

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

②添加application.yml添加配置项连接github通用配置：



③主启动类上添加注解@EnableConfigServer

④地址访问三种方式：label:分支(branch)，name:环境前缀，profile:环境(dev/test/prod)

[http://localhost:3344/{label}/{name}-{profile}.yml](http://localhost:3344/{label}/{application}-{profile}.yml) /master/config-dev.yml

[http://localhost:3344/{name}-{profile}.yml](http://localhost:3344/{label}/{application}-{profile}.yml) /config-dev.yml 默认访问master分支

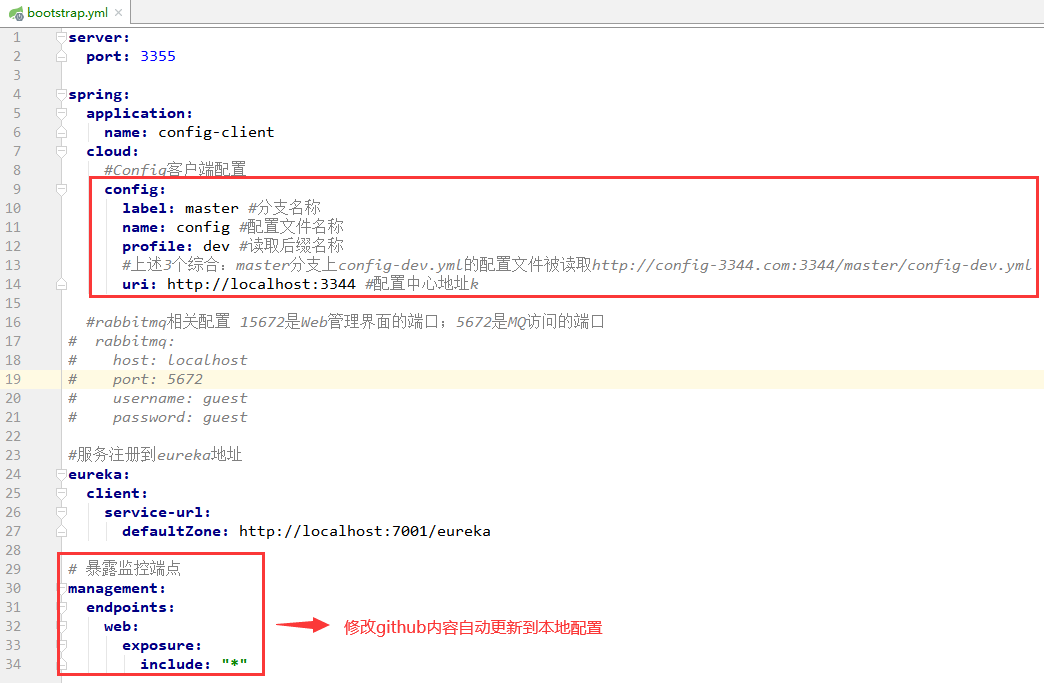
[http://localhost:3344/{application}/{profile}/{label}](http://localhost:3344/config/dev/master) /config/dev/master

1. Config：服务使用端连接服务提供端获取服务，并实现修改github并自动刷新使用端内容而替代重启服务：

①导入config服务使用端jar包依赖：

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

②创建bootstrap.yml(是系统级的比application.yml用户级的优先级更高，主要用来连接外部资源)，配置：



③本地能接收github更改内容还需要controller类上添加注解@RefreshScope，并cmd发送post命令curl -X POST "http://localhost:3355/actuator/refresh"

1. Bus：消息总线，用来将分布式系统的节点与轻量级消息系统(RabbitMQ和Kafka)链接起来的框架，该主题中产生的消息会被所有实例监听和消费：

①安装并导入Bus-rabbitMQ的jar包依赖：

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-bus-amqp</**artifactId**>  
 </**dependency**>

②服务提供端和服务使用端配置rabbitMQ配置项:

*#rabbitmq相关配置 15672是Web管理界面的端口；5672是MQ访问的端口***rabbitmq:  
 host:** localhost  
 **port:** 5672  
 **username:** guest  
 **password:** guest

③所有服务就都会监听MQ中同一个topic(默认是springCloudBus)。当一个服务刷新数据的时候，会把这个信息放到Topic中，这样其他监听同一个Topic的服务就能得到通知，然后去更新自身的配置。

1. CloudStream：消息驱动，屏蔽多个消息中间件的差异，通过binder对象负责与消息中间件交互，目前仅支持RabbitMQ、Kafka：

Binder: 很方便的连接中间件，屏蔽差异，input对应消费者，output对应生产者。

Channel：通道，是队列Queue的一种抽象，在消息通讯系统中就是实现存储和转发的媒介，通过Channel对队列进行配置。

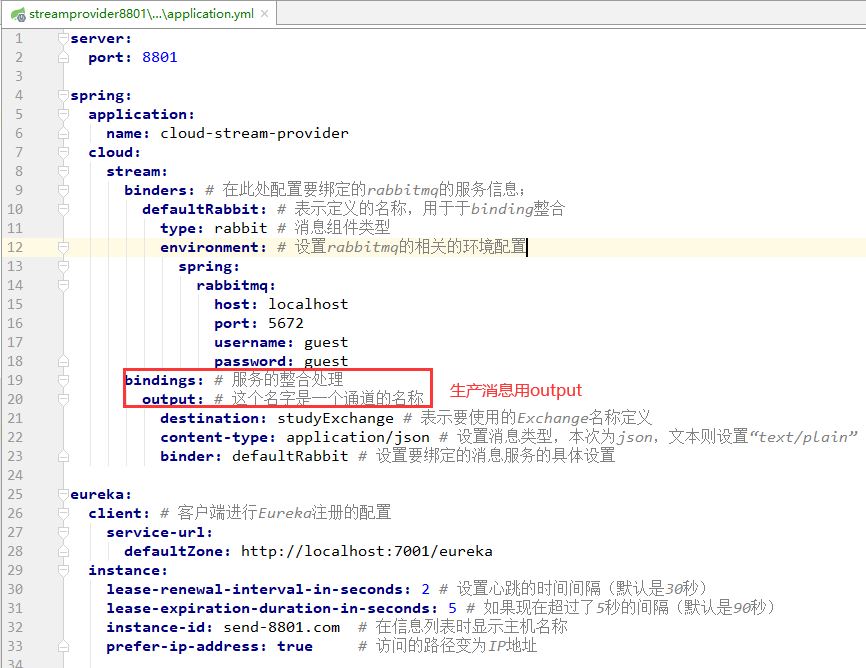
Source和Sink：简单的可理解为参照对象是Spring Cloud Stream自身，从Stream发布消息就是输出，接受消息就是输入。

1. CloudStream：以rabbitMQ为例，构建服务消息提供端生产并发送消息：

①导入rabbitMQ的jar包依赖

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-stream-rabbit</**artifactId**>  
 </**dependency**>

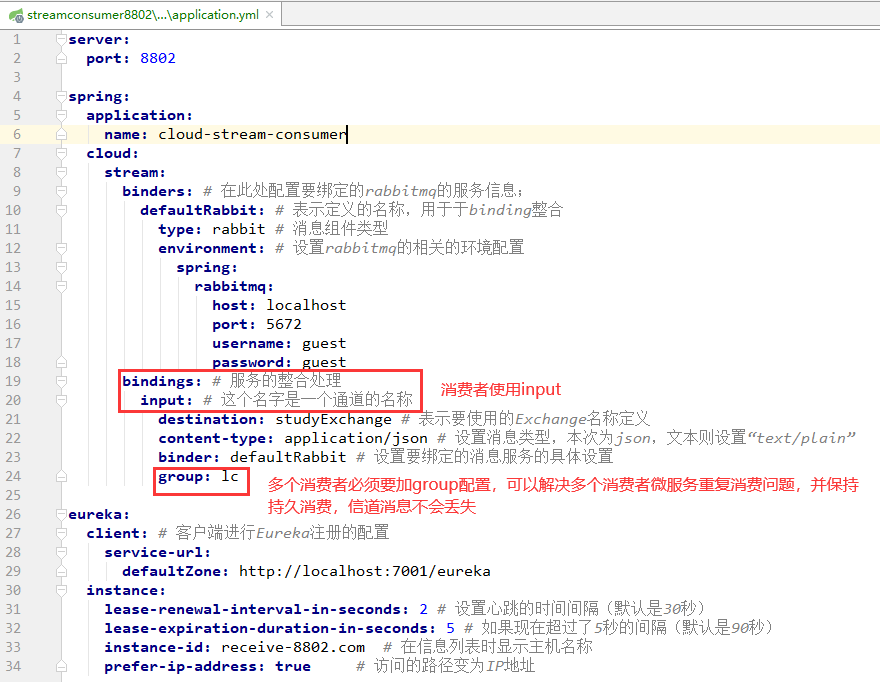
②修改服务提供者application.yml文件



③生产并向Channel中发送待消费消息：



④创建多个服务消息消费端来接收并消费消息，注：一定要加group配置项，保证消息不会重复消费，并保持消息持久，信道消息不会丢失：



⑤服务消息消费端配置消息监听并消费消息：

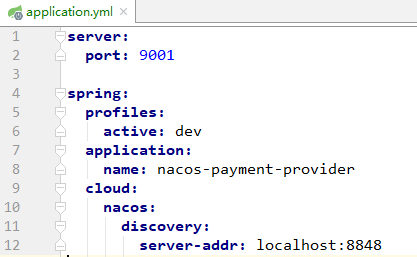


1. Nacos：注册中心 + 配置中心的组合，Nacos支持AP和CP模式的切换。
2. Nacos：服务注册中心：

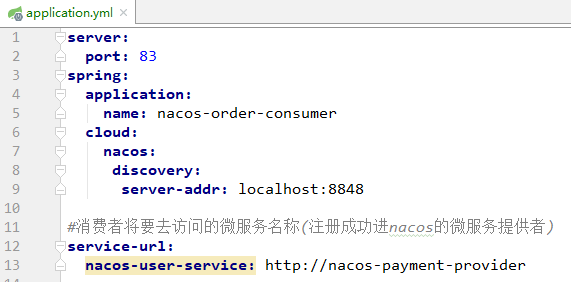
①服务提供端与消费端导入nacos的jar包依赖：

<**dependency**>  
 <**groupId**>com.alibaba.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-alibaba-nacos-discovery</**artifactId**>  
 </**dependency**>

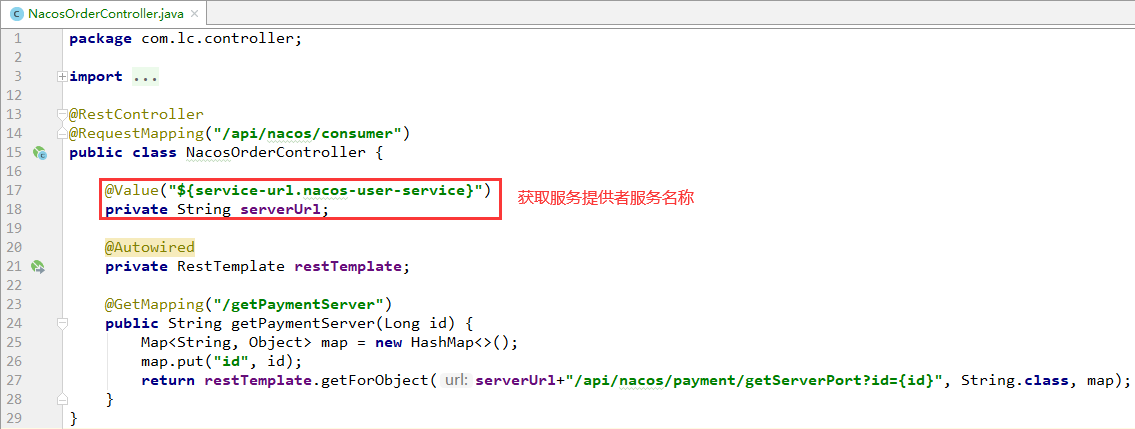
②服务提供端配置application.yml，将服务注册到nacos上，名字为服务名：



③服务消费端配置application.yml，调用nocas服务：



④服务消费端调用服务提供端，需要配置RestTemplate：

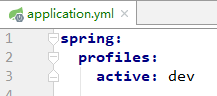
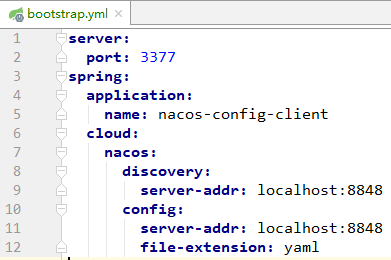


1. Nacos：服务配置中心：

①导入Nacos的config配置jar包

<**dependency**>  
 <**groupId**>com.alibaba.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-alibaba-nacos-config</**artifactId**>  
 </**dependency**>

②bootstrap.yml和application.yml配置项，就会去Nacos管理平台中找Data Id名称为nacos-config-client-dev.yaml的配置文件，公式为${spring.application.name}- ${spring.profile.active}.${spring.cloud.nacos.config.file-extension}，注意：Nacos后缀要与文件后缀一致。



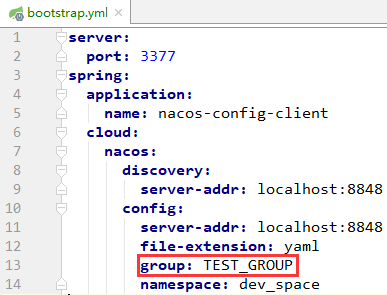
③获取Nacos管理平台配置文件内容，并实现自动接收改动本地：



1. Nacos：Namespace + Group + Data lD关系：

①Data lD：指定spring.profile.active和配置文件的DatalD来使不同环境下读取不同的配置；

②Group：通过Group实现环境区分，创建配置是可确定分组，注意字母大小写；



③Namespace：实现命名空间区分，比如不同地区的机房来调用不同空间的配置文件，添加上图14行，获取文件的路径就是dev\_space空间下的TEST\_GROUP组，spring.profile.active的配置文件。

1. Nacos：实现持久化，默认自带的是嵌入式数据库derby，多个Nacos实现数据同步，切换mysql步骤：

①nacos/conf录下找到nacos-mysql.sql文件，执行脚本；

②nacos/conf目录下找到application.properties，添加以下配置，重启Nacos:

spring.datasource.platform=mysql

db.num=1

db.url.0=jdbc:mysql://localhost:3306/nacos\_config?characterEncoding=utf8&connectTimeout=1000&socketTimeout=3000&autoReconnect=true

db.user=root

db.password=root

1. Nacos：Linux版实现Nacos集群：