尚品汇商城复习

版本：V 1.0

项目架构

一、电商行情

1、 电商行业技术特点

- 技术新

  单体应用：JSP+servlet+jdbc、SSH（Struts 和 Struts2+Spring+hibernate）、SSM（springmvc+spring+mybatis）、springboot+mybatis

  微服务：WEBservice-----远程调用框架 xml

  Dubbo+注册中心（zookeeper、redis、nacos）：rpc 远程服务框架。

  SpringCloud：nacos、openfeign、sentinel、gateway

  自主封装：阿里的 HSF、京东 JSF（杰夫）、金融的、保险.....

  持久层：mybatis、mybatis-plus、TKMapper（单表操作实现好了）

  SpringDataJPA（封装 hibernate）

  三层架构：

  控制层（controller）、业务层（service）、持久层（DAO）

  为什么分层：解耦。每层只做自己该做的事。

  控制层：处理请求和响应。

  业务层：做业务处理（条件判断、for 循环）、控制事务。

  持久层：与数据库交互，进行数据持久化。

  ORM 框架。

  当年技术面试面完了，技术面试官会问你，有什么想问他的?

  1、 了解人员配置 java 多少人 前端。

  2、 技术栈

  EJB

  0-1：什么都没有。 项目外包，给甲方客户做项目。

  迭代开发。

- 技术范围广

  Springboot+springcloud（nacos、openfeign、sentinel、gateway）+redis（4

种）+rabbitMQ（解决消息的准确性+延迟消息）+多线程异步编排（线程池）+分布式锁（演变）+docker+Lombok+ES+kibana。。。。。。

- 微服务

  为什么?

  怎么拆分的：根据业务拆分，不同业务是不同的服务，每一个服务都可以独立运行，连接不同的数据库。都可以集群部署。

  好处：解耦、解决高并发、开发独立

- 高并发

  怎么解决高并发?

  1、 架构层面：微服务

  2、 技术层面：缓存（redis）、异步（线程的异步、队列）、队排好（排队、MQ）、限流（限制访问流量）

  3、 硬件层面：服务器性能 CPU、内存、硬盘(SSD)、集群部署

  4、 网络层面：网络带宽、网络加速

- 海量数据

  用户行为数据、订单数据。（商品数据、评论数据）

  开发时有吗？没有----程序员自己填的。

  测试有吗？没有----自己填的测试数据。

  上线了有吗？不一定

  推广、引流 请明星代言、网络推广 抖音。地推

- 业务复杂

1、 订单—支付---库存————拆单

2、 秒杀

3、 其他方面的优化：商品详情的优化

4、 电商的主要模式

B2B--企业对企业

B2B （Business to Business）是指进行电子商务交易的供需双方都是商家（或企业、公司），她（他）们使用了互联网的技术或各种商务网络平台，完成商务交易的过程。电子商务是现代 B2B marketing 的一种具体主要的表现形式。

案例：阿里巴巴、慧聪网

C2C--个人对个人

C2C 即 Customer（Consumer） to Customer（Consumer），意思就是消费者个人间的电子商务行为。比如一个消费者有一台电脑，通过网络进行交易，把它出售给另外一个消费者，此种交易类型就称为 C2C 电子商务。

案例：闲鱼、瓜子二手车、转转

B2C--企业对个人

B2C 是 Business-to-Customer 的缩写，而其中文简称为“商对客”。“商对客”是电子商务的一种模式，也就是通常说的直接面向消费者销售产品和服务商业零售模式。这种形式的电子商务一般以网络零售业为主，主要借助于互联网开展在线销售活动。B2C 即企业通过互联网为消费者提供一个新型的购物环境——网上商店，消费者通过网络在网上购物、网上支付等消费行为。

案例：联想商城、小米商城、官网电商、小米有品等

O2O--线上到线下

O2O 即 Online To Offline（在线离线/线上到线下），是指将线下的商务机会与互联网结合，让互联网成为线下交易的平台，这个概念最早来源于美国。O2O 的概念非常广泛，既可涉及到线上，又可涉及到线下,可以通称为 O2O。主流商业管理课程均对 O2O 这种新型的商业模式有所介绍及关注。

案例：美团、饿了吗、每日优鲜、京东到家、永辉生活、多点商城

滴滴

B2B2C -企业-企业-个人

B2B2C 是一种电子商务类型的网络购物商业模式，B 是 BUSINESS 的简称，C 是 CUSTOMER 的简称，第一个 B 指的是商品或服务的供应商，第二个 B 指的是从事电子商务的企业，C 则是表示消费者。

案例：京东商城、天猫商城

二、项目介绍和架构

1、项目介绍：（简历、面试，仅供参考）

尚品汇商城是 B2C 模式的综合性在线销售平台。商城分为后台管理部分与用户前台使用部分。后台管理部分包括：商品管理模块（商品分类、品牌、平台属性、SPU 与 SKU 以及销售属性、商品上下架和商品评论管理等）、内容广告模块、库存管理模块、订单管理模块、促销活动管理（优惠券满减活动、秒杀等商品设置）、客户管理模块、统计报表模块和系统基础权限等模块。

用户前台使用部分：商城首页、商品搜索（可按条件查询展示）、商品详情信息展示、购物车、用户注册、单点登录和社交登录（微信登录）、用户会员中心、订单的创建修改、展示以及在线支付（支付宝、微信）、物流模块、商品评论以及秒杀活动等功能。

2、架构相关：

2.1、项目开发环境：

IntellijIDEA：IDE（开发工具的统称）eclipse vscode

MySQL：关系型数据库，数据之间有关联关系 一对多 多对多

Git ：版本控制工具， 1.0 1.1 1.2.。。。。。。。

Git 仓库：gitee、github 在线托管平台

自己搭建：gitlab，这种用的最多。

Maven：项目管理工具 mvn

Java –jar service-item.jar

Java -jar Jenkins.war

项目创建、编译、打包（pom、war、jar）、安装、 管理 jar 包（依赖传递）、运行

装了 maven 的 tomcat 插件。

2.2、项目的技术架构：

SpringBoot：简化新 Spring 应用的初始搭建以及开发过程，（内部集成 SpringMVC 和 Tomcat）starter-web 项目整合框架，实现项目架构整合

SpringCloud：基于 Spring Boot 实现的云原生应用开发工具，SpringCloud 使用的技术：（Spring Cloud Gateway（zuul）、Spring Cloud Alibaba Nacos（eureka、config）、Spring Cloud Alibaba Sentinel 和 Spring Cloud OpenFeign（ribbon）等）

SpringTask：Spring 定时任务，实现定时发送消息等

MyBatis-Plus：持久层框架，实现数据持久化（ORM），与数据库进行交互

Redis：内存缓存，实现数据缓存-----商品详情数据缓存、分布式锁、单点登录存储用户信息、存储购物车数据等

Redisson：基于 redis 的 Java 驻内存数据网格

RabbitMQ：消息队列中间件，实现消息通知，异步解耦

ElasticSearch+Logstash+Kibana+Rest（spring 提供 javaAPI）：全文检索服务器+可视化数据监控

ThreadPoolExecutor：线程池来实现异步操作，提供效率

Swagger2：Api 接口文档工具、测试

PostMan+接口文档就得手写。

后台开发人员写接口文档。----支付宝支付的文档。

MiniO:文件存储

支付宝支付、微信支付

Lombok: 实体类的中 get，set 生成的 jar 包

Docker：容器技术

DockerFile：管理 Docker 镜像命令文本

Jenkins：持续集成工具

后端：

SpringBoot + SpringCloudAlibaba + MyBatis-Plus + Redis + RabbitMQ + Minio +ES

前端：（不是后台开发太关注的）

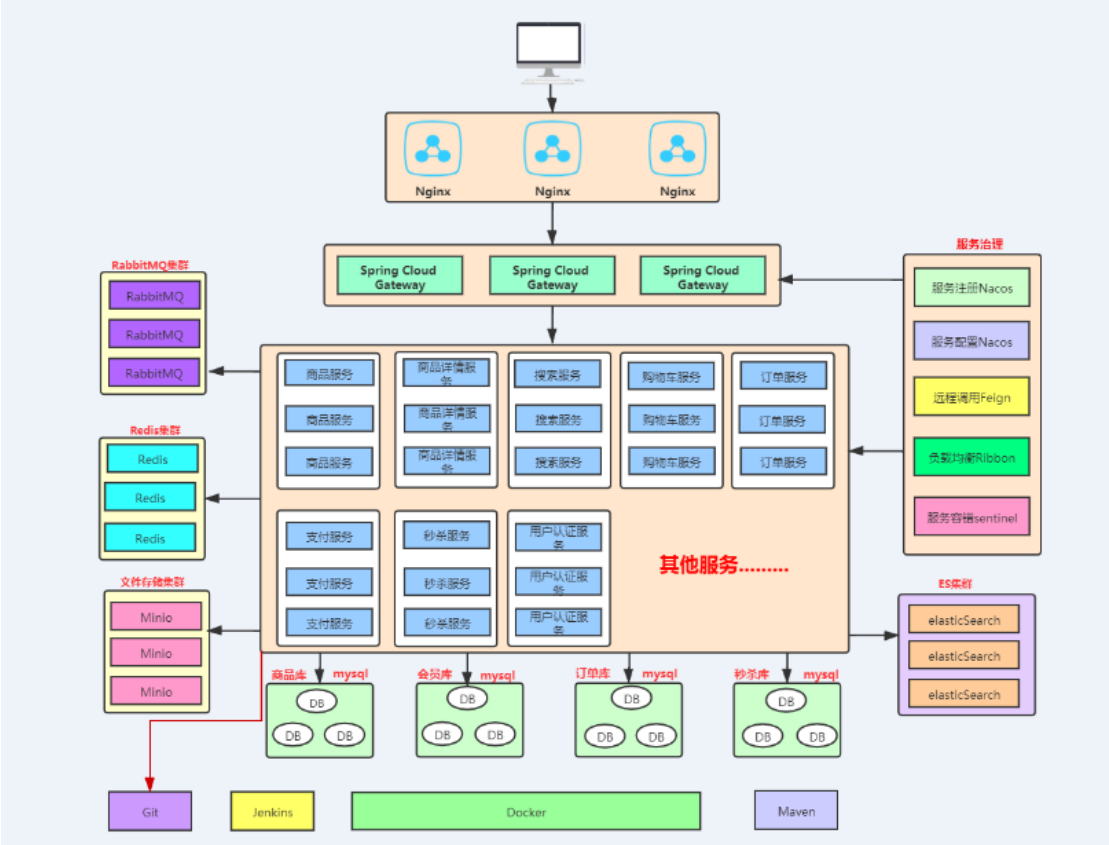
管理后台：vue + element-ui + nodejs + npm

电商网站：spring boot + thymeleaf + vue

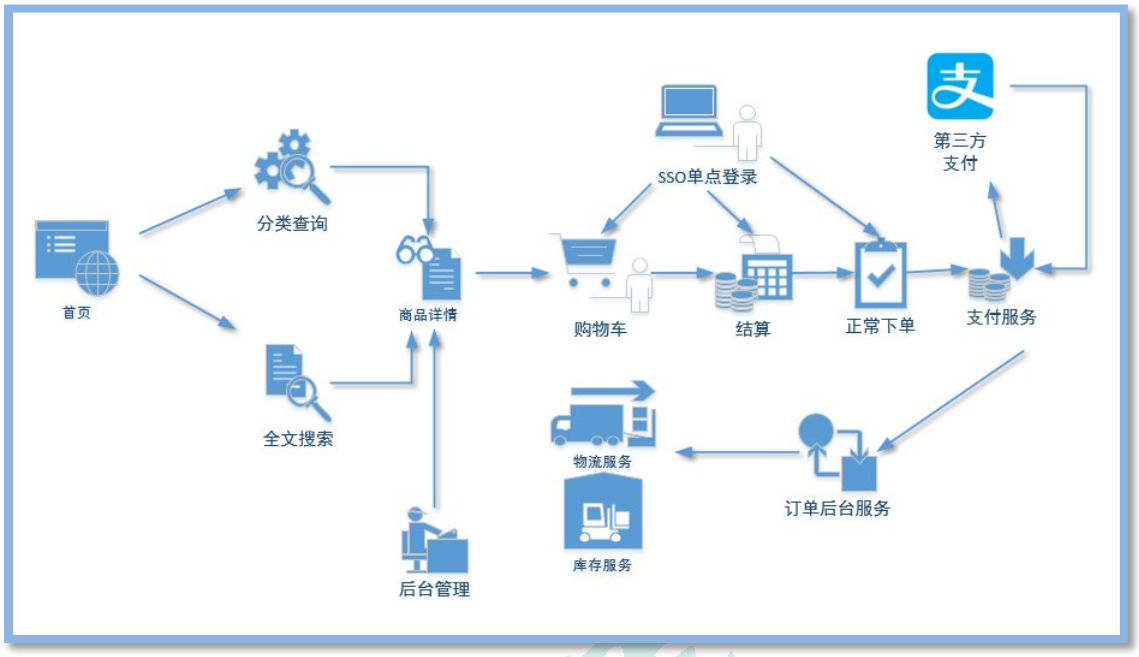
其他：Redisson 实现分布式锁、异步编排、使用支付宝支付、使用 Swagger 生成接口文档

部署方案：Jenkins+docker+maven+git

2.3、项目架构图：



3、整体业务简介：



首页	静态页面，包含了商品分类，搜索栏，商品广告位。
全文搜索	通过搜索栏填入的关键字进行搜索，并列表展示
分类查询	根据首页的商品类目进行查询
商品详情	商品的详细信息展示
购物车	将有购买意向的商品临时存放的地方
单点登录	用户统一登录的管理
结算	将购物车中勾选的商品初始化成要填写的订单
下单	填好的订单提交
支付服务	下单后，用户点击支付，负责对接第三方支付系统。
订单服务	负责确认订单是否付款成功，并对接仓储物流系统。
仓储物流	独立的管理系统，负责商品的库存。

后台管理	主要维护类目、商品、库存单元、广告位等信息。
秒杀	秒杀抢购完整方案

三、Nacos

官方地址：https://nacos.io

github 地址：https://github.com/alibaba/nacos

1、什么是 Nacos

Nacos 是阿里巴巴推出来的一个新开源项目，这是一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。

Nacos 致力于帮助您发现、配置和管理微服务。Nacos 提供了一组简单易用的特性集，帮助您快速实现动态服务发现、服务配置及管理。

常见的注册中心：

1. Eureka（原生，2.0 停止维护）

2. Zookeeper（支持，专业的独立产品。例如：dubbo、或集群管理）

3. Consul（原生，GO 语言开发）

4. Nacos

相对于 Spring Cloud Eureka 来说，Nacos 更强大。

Nacos = Spring Cloud Eureka + Spring Cloud Config

Nacos 可以与 Spring, Spring Boot, Spring Cloud 集成，并能代替 Spring Cloud Eureka, Spring Cloud Config。

- 通过 Nacos Server 和 spring-cloud-starter-alibaba-nacos-config 实现配置的动态变更。

- 通过 Nacos Server 和 spring-cloud-starter-alibaba-nacos-discovery 实现服务的注册与发现。

Resttemplate：spring 提供的对 restful 请求模板

Restful：简易风格的 http 请求

http： localhost:8080/getUserById?id=3

restful: localhost:8080/getUserById/3/zhangsan

@getMapping（“getUserById/{id}/{name}”）

2、Nacos 可以干什么

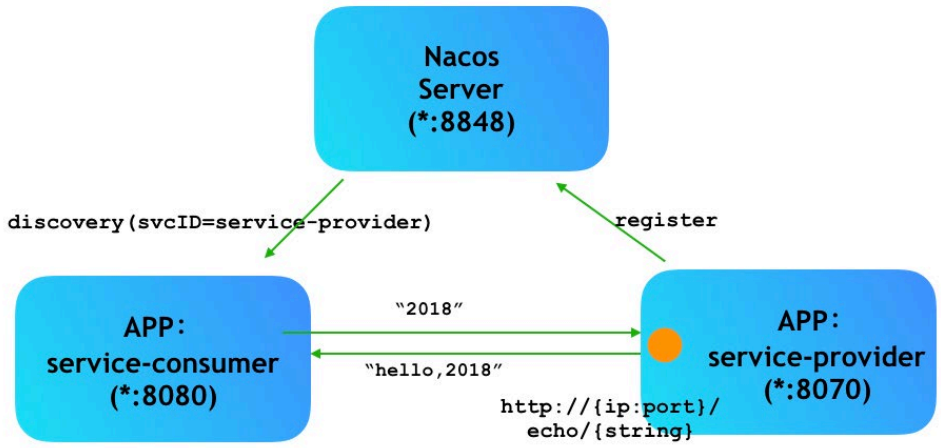
Nacos 是以服务为主要服务对象的中间件，Nacos 支持所有主流的服务发现、配置和管理。

Nacos 主要提供以下功能：

1. 服务发现和服务健康监测

2. 动态配置服务（维护的时候）

3、Nacos 注册中心的使用



4、Nacos 配置中心的使用

在系统开发过程中，开发者通常会将一些需要变更的参数、变量等从代码中分离出来独立管理，以独立的配置文件的形式存在。目的是让静态的系统工件或者交付物（如 WAR，JAR 包等）更好地和实际的物理运行环境进行适配。配置管理一般包含在系统部署的过程中，由系统管理员或者运维人员完成。配置变更是调整系统运行时的行为的有效手段。

如果微服务架构中没有使用统一配置中心时，所存在的问题：

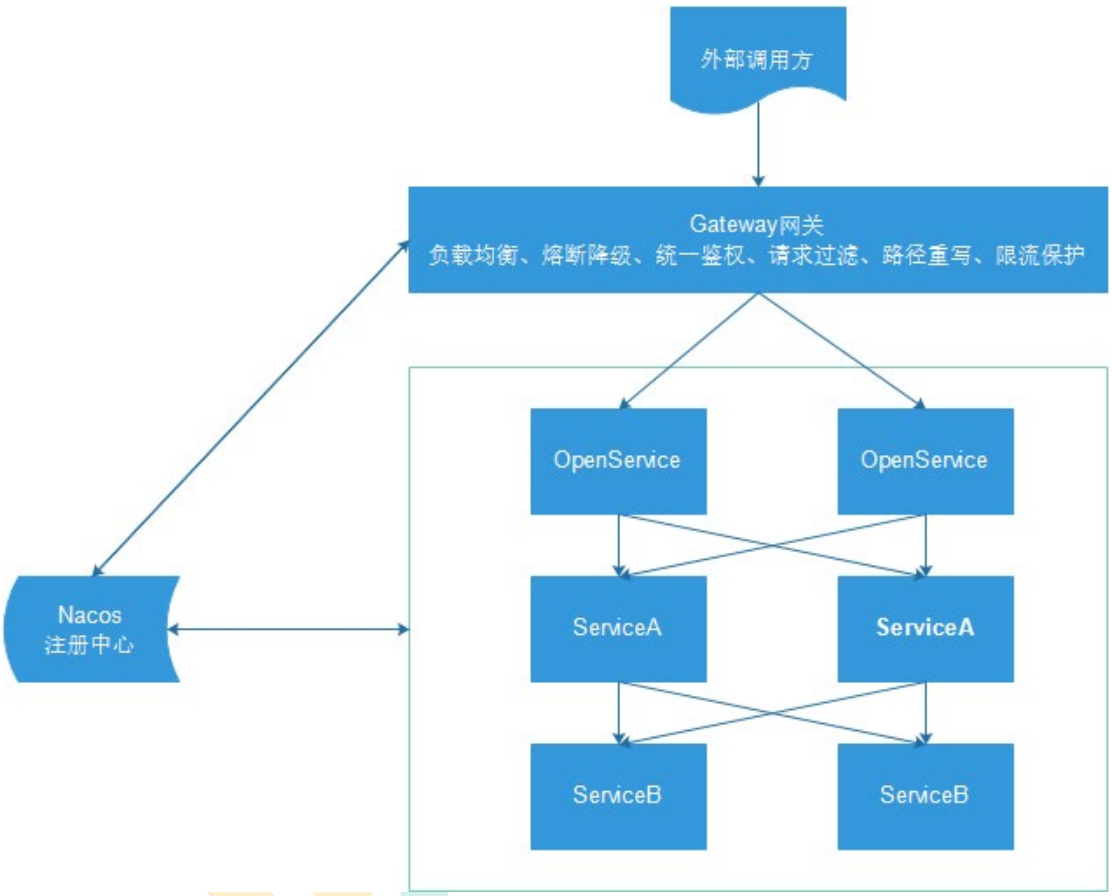
- 配置文件分散在各个项目里，不方便维护

- 配置内容安全与权限

- 更新配置后，项目需要重启

nacos 配置中心：系统配置的集中管理（编辑、存储、分发）、动态更新不重启、回滚配置（变更管理、历史版本管理、变更审计）等所有与配置相关的活动。

四、服务网关 Gateway



1、 快速开始

1.1 、引入依赖

pom.xml 中的依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

1.2、编写路由规则

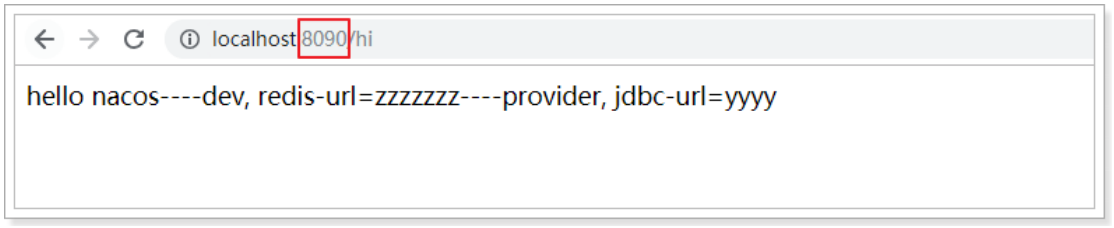
为了演示路由到不同服务，这里把消费者和生产者都配置在网关中

application.yml

```
server:
  port: 8090
spring:
  cloud:
    gateway:
      routes:
        - id: nacos-consumer
          uri: http://127.0.0.1:8080
          predicates:
            - Path=/hi
        - id: nacos-provider
          uri: http://127.0.0.1:8070
          predicates:
            - Path=/hello
```

1.3、启动测试

通过网关路径访问消费者或者生产者。



在开发中由于所有微服务的访问都要经过网关，为了区分不同的微服务，通常会在路径前加上一个标识，例如：

访问服务提供方：`http://localhost:8090/provider/hello`；

访问服务消费方：http://localhost:8090/consumer/hi

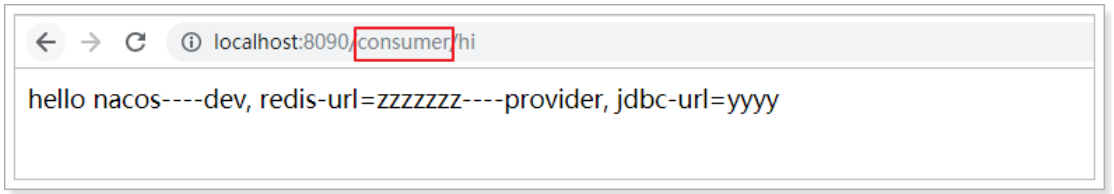
如果不重写地址，直接转发的话，转发后的路径为：`http://localhost:8070/provider/hello` 和 `http://localhost:8080/consumer/hi` 明显多了一个 provider 或者 consumer，导致转发失败。

这时，我们就用上了路径重写，配置如下：

```
server:
  port: 8090
spring:
  cloud:
    gateway:
      routes:
        - id: nacos-consumer
          uri: http://127.0.0.1:8080
          predicates:
            - Path=/consumer/**
          filters:
            - RewritePath=/consumer/(?<segment>.*),/$\{segment}
        - id: nacos-provider
          uri: http://127.0.0.1:8070
          predicates:
            - Path=/provider/**
          filters:
            - RewritePath=/provider/(?<segment>.*),/$\{segment}
```

注意： `Path=/consumer/**` 及 `Path=/provider/**` 的变化

测试：



2、面向服务的路由

```
spring:
  cloud:
    gateway:
      routes:
        - id: nacos-consumer
          uri: http://127.0.0.1:8080
          predicates:
            - Path=/consumer/**
          filters:
            - RewritePath=/consumer/(?<segment>.*), /$\{segment}
        - id: nacos-provider
          uri: http://127.0.0.1:8070
          predicates:
            - Path=/provider/**
          filters:
            - RewritePath=/provider/(?<segment>.*), /$\{segment}
```

由于路由地址写死了，无法做到负载均衡

如果要做到负载均衡，则必须把网关工程注册到 nacos 注册中心，然后通过服务名访问。

2.1 、把网关服务注册到 nacos

```
@SpringBootApplication
@EnableDiscoveryClient
public class GatewayDemoApplication {


    public static void main(String[] args) {
        SpringApplication.run(GatewayDemoApplication.class, args);
    }
}
```

2.2 、修改配置，通过服务名路由

```
server:
```

```
  port: 8090
spring:
  cloud:
    nacos:
      discovery:
        server-addr: 127.0.0.1:8848
    gateway:
      routes:
        - id: nacos-consumer
          uri: lb://nacos-consumer
          predicates:
            - Path=/consumer/**
          filters:
            - RewritePath=/consumer/(?<segment>.*),/$\{segment}
        - id: nacos-provider
          uri: lb://nacos-provider
          predicates:
            - Path=/provider/**
          filters:
            - RewritePath=/provider/(?<segment>.*),/$\{segment}
```

语法：lb://服务名

lb：LoadBalance，代表负载均衡的方式

服务名取决于 nacos 的服务列表中的服务名

五、Sentinel

1、Sentinel 是什么

随着微服务的流行，服务和服务之间的稳定性变得越来越重要。Sentinel 是面向分布式服务架构的轻量级流量控制组件，主要以流量为切入点，从限流、流量整形、熔断降级、系统负载保护等多个维度来帮助您保障微服务的稳定性。

2、如何使用 Sentinel

如果要在您的项目中引入 Sentinel 依赖

在服务提供者，服务消费者项目中都添加

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
    <version>0.2.2.RELEASE</version>
</dependency>
```

配置控制台信息

application.properties

```
spring.cloud.sentiel.transport.dashboard=127.0.0.1:8088
```

3、Sentinel 控制台

您可以从 release 页面 下载最新版本的控制台 jar 包。

https://github.com/alibaba/Sentinel/releases
在下的 jar 包中运行{默认 8080} java -Dserver.port=8088 -jar sentinel-dashboard.jar
访问任意一个请求，在控制台中都能看见服务名称对应的请求

4、Feign 支持

Sentinel 适配了 Feign 组件。如果想使用，除了引入 spring-cloud-starter-alibaba-sentinel 的依赖外还需要 2 个步骤：

在服务消费者配置文件打开 Sentinel 对 Feign 的支持：feign.sentinel.enabled=true

加入 spring-cloud-starter-openfeign 依赖使 Sentinel starter 中的自动化配置类生效

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

服务消费者的 Feign 接口上添加实现类

```
@FeignClient(name = "nacos-provider",fallback = ProviderFeignImpl.class)
public interface ProviderFeign {
    @RequestMapping("hello")
    public String hello();
```

```
}
```

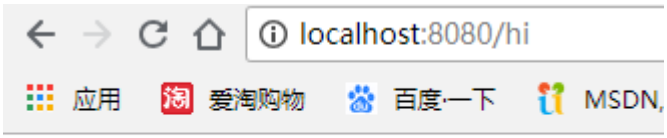
```
@Component
public class ProviderFeignImpl implements ProviderFeign {
    @Override
    public String hello() {
        return "出现异常了！";
    }
}
```

服务提供者：

```
@GetMapping("hello")
public String hello(HttpServletRequest request){
    int i = 1/0;
    return "hello " + myName+redisUrl+jdbcUrl;
}
```

测试：

http://localhost:8080/hi

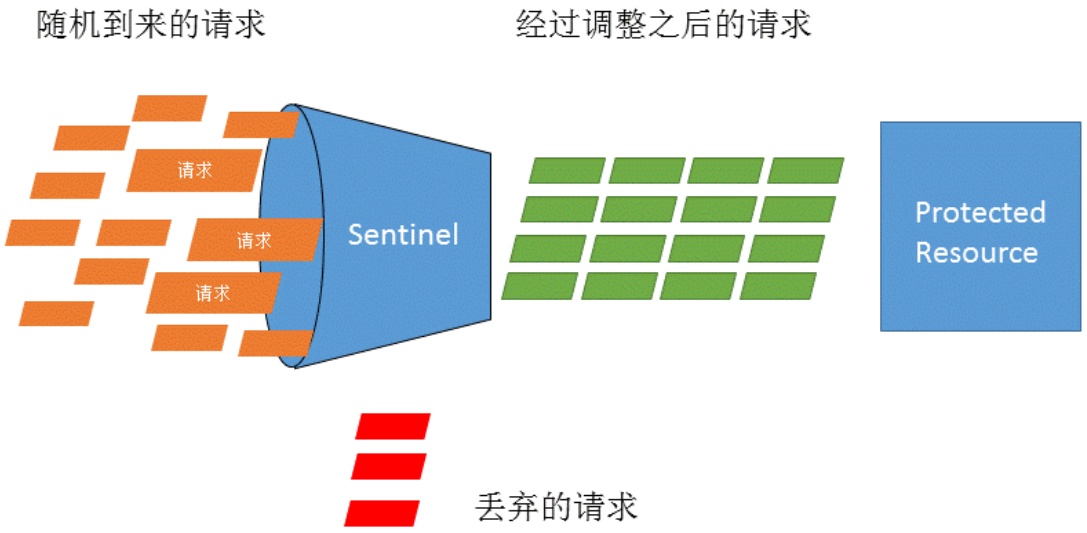


Hi出现异常了！

5、限流配置

5.1 什么是流量控制

流量控制在网络传输中是一个常用的概念，它用于调整网络包的发送数据。然而，从系统稳定性角度考虑，在处理请求的速度上，也有非常多的讲究。任意时间到来的请求往往是随机不可控的，而系统的处理能力是有限的。我们需要根据系统的处理能力对流量进行控制。Sentinel 作为一个调配器，可以根据需要把随机的请求调整成合适的形状，如下图所示：



流量控制有以下几个角度:

- 资源的调用关系，例如资源的调用链路，资源和资源之间的关系；

- 运行指标，例如 QPS、线程数等；

- 控制的效果，例如直接限流（快速失败）、冷启动（Warm Up）、匀速排队（排队等待）等。

Sentinel 的设计理念是让您自由选择控制的角度，并进行灵活组合，从而达到想要的效果。

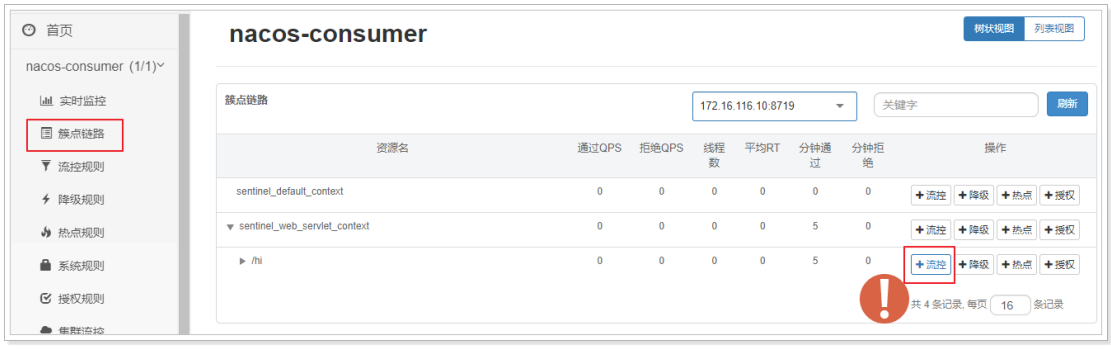
配置如下:

5.2 QPS 流量控制

当 QPS 超过某个阈值的时候，则采取措施进行流量控制。流量控制的效果包括以下几种：**直接拒绝**、**Warm Up**、**匀速排队**。

5.2.1 直接拒绝

直接拒绝（`RuleConstant.CONTROL_BEHAVIOR_DEFAULT`）方式是默认的流量控制方式，当 QPS 超过任意规则的阈值后，新的请求就会被立即拒绝，拒绝方式为抛出 `FlowException`。这种方式适用于对系统处理能力确切已知的情况下，比如通过压测确定了系统的准确水位时。



这里做一个最简单的配置：

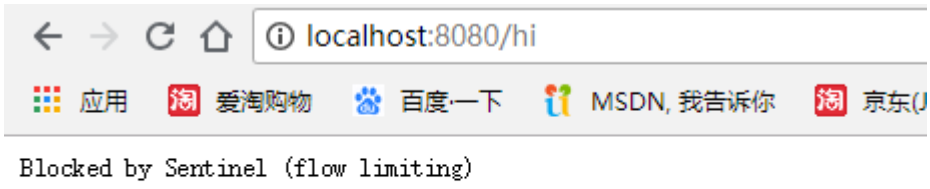
阈值类型选择：QPS

单机阈值：2

综合起来的配置效果就是，该接口的限流策略是每秒最多允许 2 个请求进入。

点击新增按钮之后，可以看到如下界面：

在浏览器访问：http://localhost:8080/hi，并疯狂刷新，出现如下信息：



Blocked by Sentinel (flow limiting)

5.2.2 Warm Up（预热）

Warm Up（`RuleConstant.CONTROL_BEHAVIOR_WARM_UP`）方式，即预热/冷启动方式。当系统长期处于低水位的情况下，当流量突然增加时，直接把系统拉升到高水位可能瞬间把系统压垮。通过"冷启动"，让通过的流量缓慢增加，在一定时间内逐渐增加到阈值上限，给冷系统一个预热的时间，避免冷系统被压垮。

疯狂访问：http://localhost:8080/hi

可以发现前几秒会发生熔断，几秒钟之后就完全没有问题了

Warm Up（`RuleConstant.CONTROL_BEHAVIOR_WARM_UP`）方式，即预热/冷启动方式。当系统长期处于低水位的情况下，当流量突然增加时，直接把系统拉升到高水位可能瞬间把系统压垮。通过"冷启动"，让通过的流量缓慢增加，在一定时间内逐渐增加到阈值上限，给冷系统一个预热的时间，避免冷系统被压垮。

5.2.3 匀速排队

匀速排队（`RuleConstant.CONTROL_BEHAVIOR_RATE_LIMITER`）方式会严格控制请求通过的间隔时间，也即是让请求以均匀的速度通过，对应的是漏桶算法。

测试配置如下：1s 处理一个请求，排队等待，等待时间 20s。

6 熔断降级

Sentinel 除了流量控制以外，对调用链路中不稳定的资源进行熔断降级也是保障高可用的重要措施之一。

Sentinel 熔断降级会在调用链路中某个资源出现不稳定状态时（例如调用超时或异常比例升高），对这个资源的调用进行限制，让请求快速失败，避免影响到其它的资源而导致级联错误。当资源被降级后，在接下来的降级时间窗口之内，对该资源的调用都自动熔断（默认行为是抛出 DegradeException）。

Sentinel 和 Hystrix 的原则是一致的: 当调用链路中某个资源出现不稳定，例如，表现为 timeout，异常比例升高的时候，则对这个资源的调用进行限制，并让请求快速失败，避免影响到其它的资源，最终产生雪崩的效果。

限流降级指标有三个，如下图：

1. 平均响应时间（RT）

2. 异常比例

3. 异常数

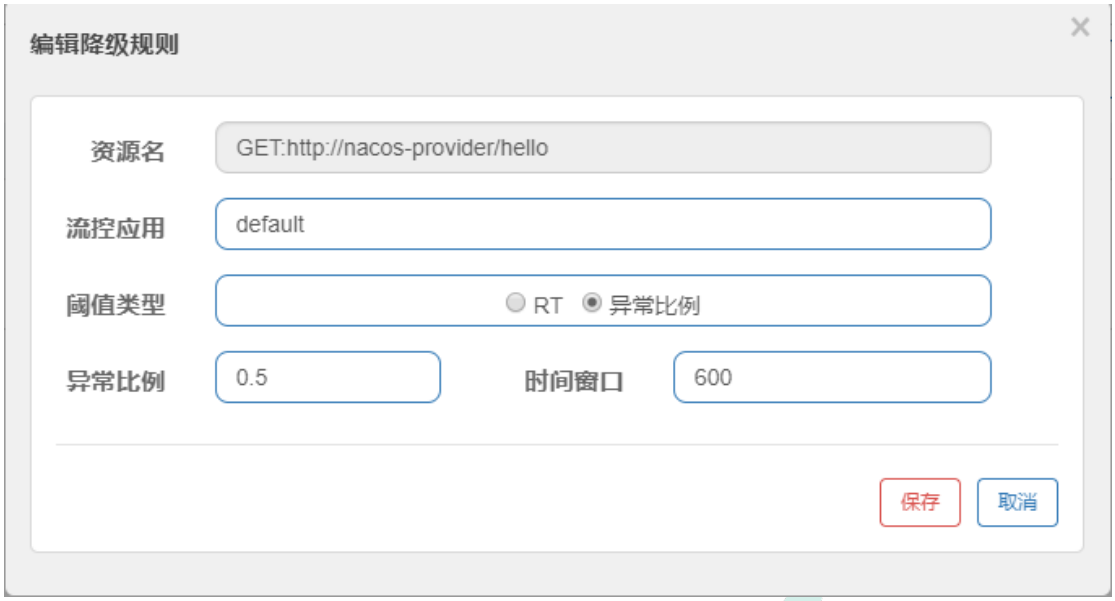
6.1 平均响应时间（RT）

平均响应时间 (`DEGRADE_GRADE_RT`)：**当资源的平均响应时间超过阈值**之后，资源进入准降级状态。如果 **1s 之内持续进入 5 个请求**，它们的 RT 都持续超过这个阈值，那么在**接下来的时间窗口**（`DegradeRule` 中的 `timeWindow`，以 s 为单位）之内，对这个方法的调用都会自动地返回（抛出 `DegradeException`）。在下一个时间窗口到来时，会接着再放入 5 个请求，再重复上面的判断。

配置如下：超时时间 100ms，熔断时间 10s

6.2 异常比例

异常比例：当资源的每秒请求量 >= 5，且每秒异常总数占通过量的比值超过阈值之后，资源进入降级状态，即在接下的时间窗口之内，对这个方法的调用都会自动地返回。异常比率的阈值范围是 `[0.0, 1.0]`，代表 0% - 100%。



6.3 异常数

异常数 (`DEGRADE_GRADE_EXCEPTION_COUNT`)：当资源近 1 分钟的异常数目超过阈值之后会进行熔断。

链路追踪 zipkin

------weball：itemcontroller------远程调用 service-item：itemApiController------itemService-------远程调用 service-product：多个接口。

能展示完整调用链的耗时。还能展示各个部分调用的耗时。

1、 后续工作

2、 做测试的时候 发现一个接口 响应特别慢，你怎么排查?