

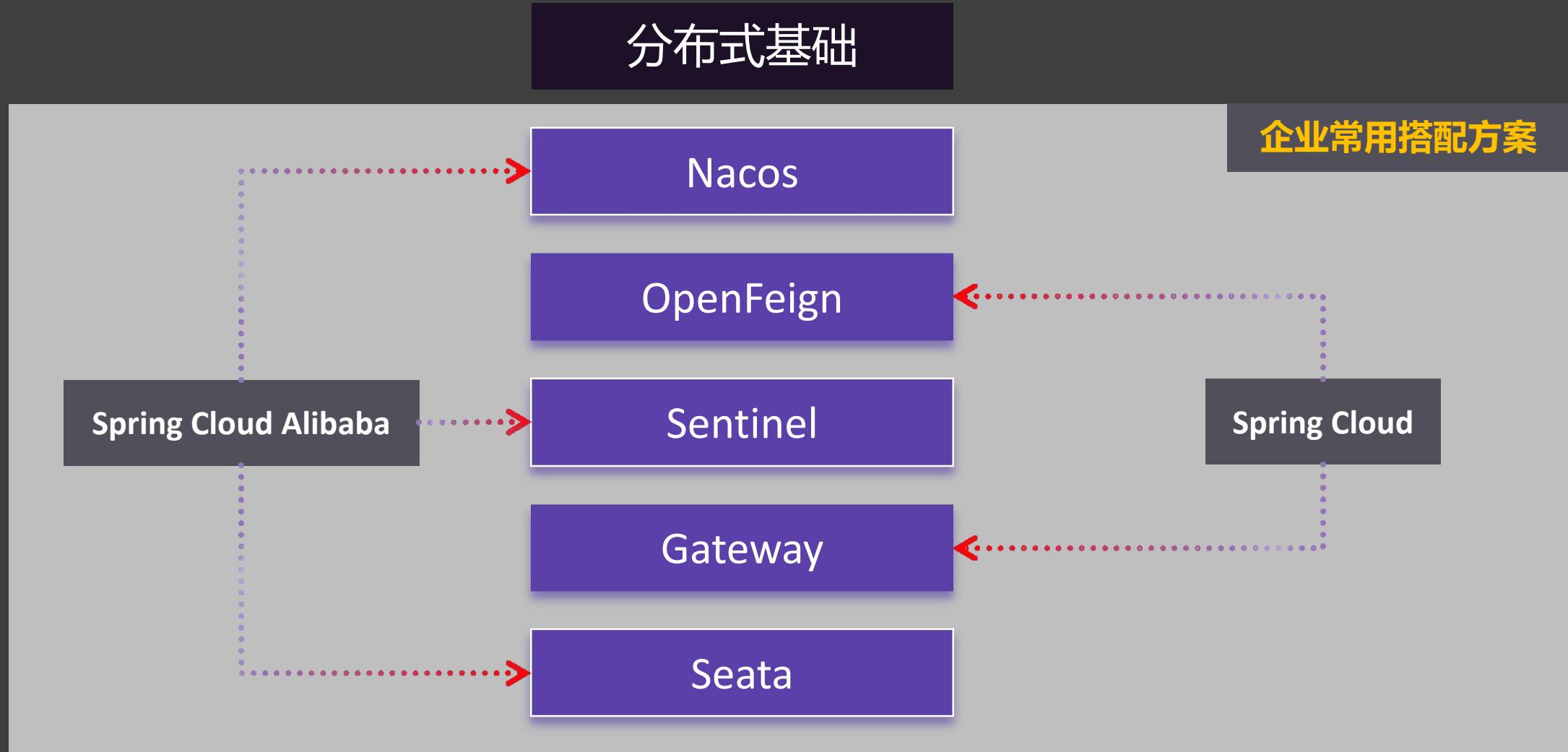
速通系列



Spring Cloud 快速通关

分布式系统一站式解决方案

课程内容



分布式配套

日志系统

指标监控

链路追踪

消息处理

...

课程特色

零基础上手

通俗易懂

全程实操

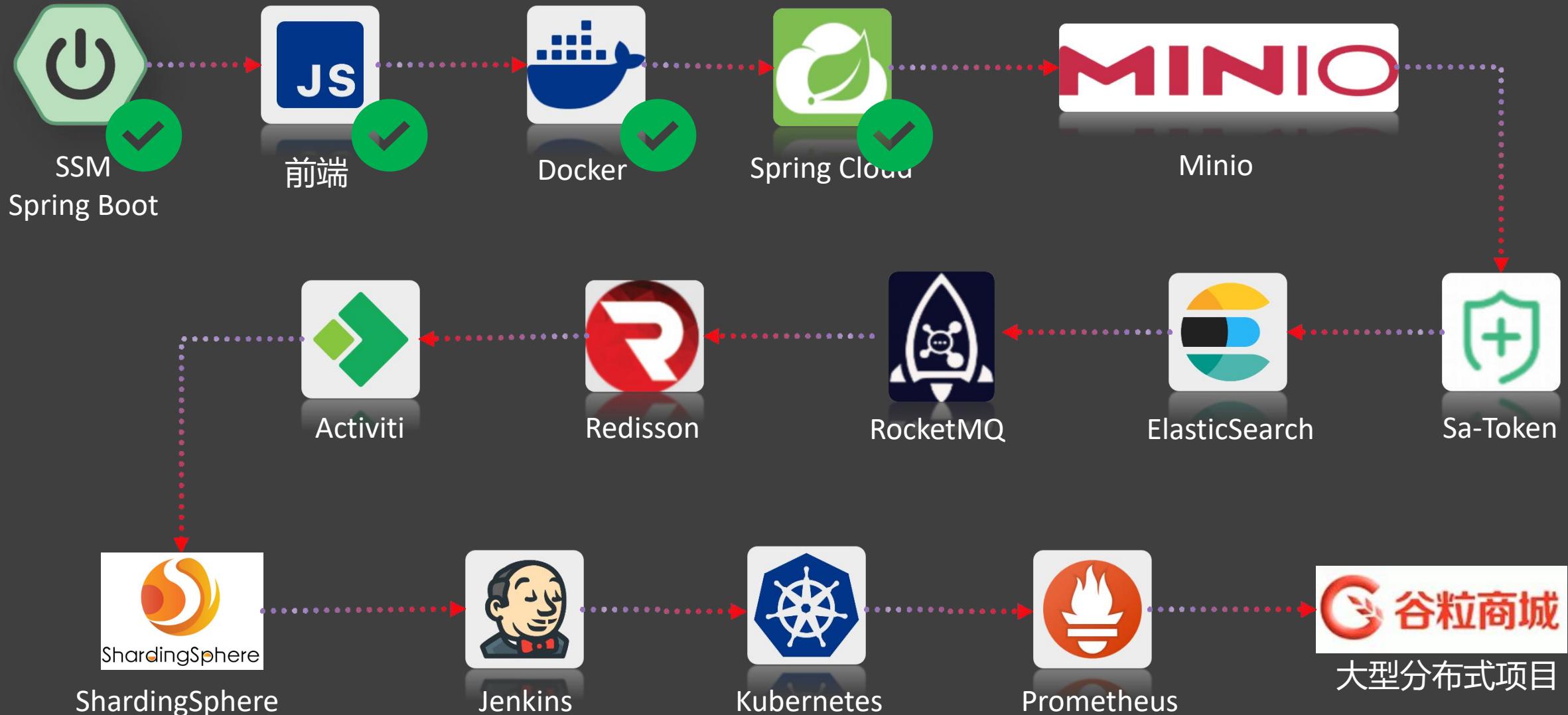
图文搭配

主次分明

重点突出

速通系列

快速点亮你的高阶技能栈



我们开始吧！

1. 分布式基础

架构演进 - 核心概念



环境准备

- 创建微服务架构项目
- 引入 SpringCloud、 Spring Cloud Alibaba 相关依赖
- 注意版本适配

SpringBoot版本	SpringCloud版本	SpringCloud Alibaba版本
3.4.x +	2024.0.x	未适配
3.2.x - 3.3.x	2023.0.x	2023.0.*
3.0.2 - 3.2.x	2022.0.x	2022.0.*
2.6.x - 2.7.x	2021.0.x	2021.0.*
2.4.x - 2.5.x	2020.0.x	2020.0.*
2.3.x -	Hoxton/Greenwich -	2.2.* -

<https://github.com/alibaba/spring-cloud-alibaba/wiki/版本说明>

版本选择

框架版本

SpringBoot
3.3.4

SpringCloud
2023.0.3

SpringCloud Alibaba 2023.0.3.2

组件版本

Nacos
2.4.3

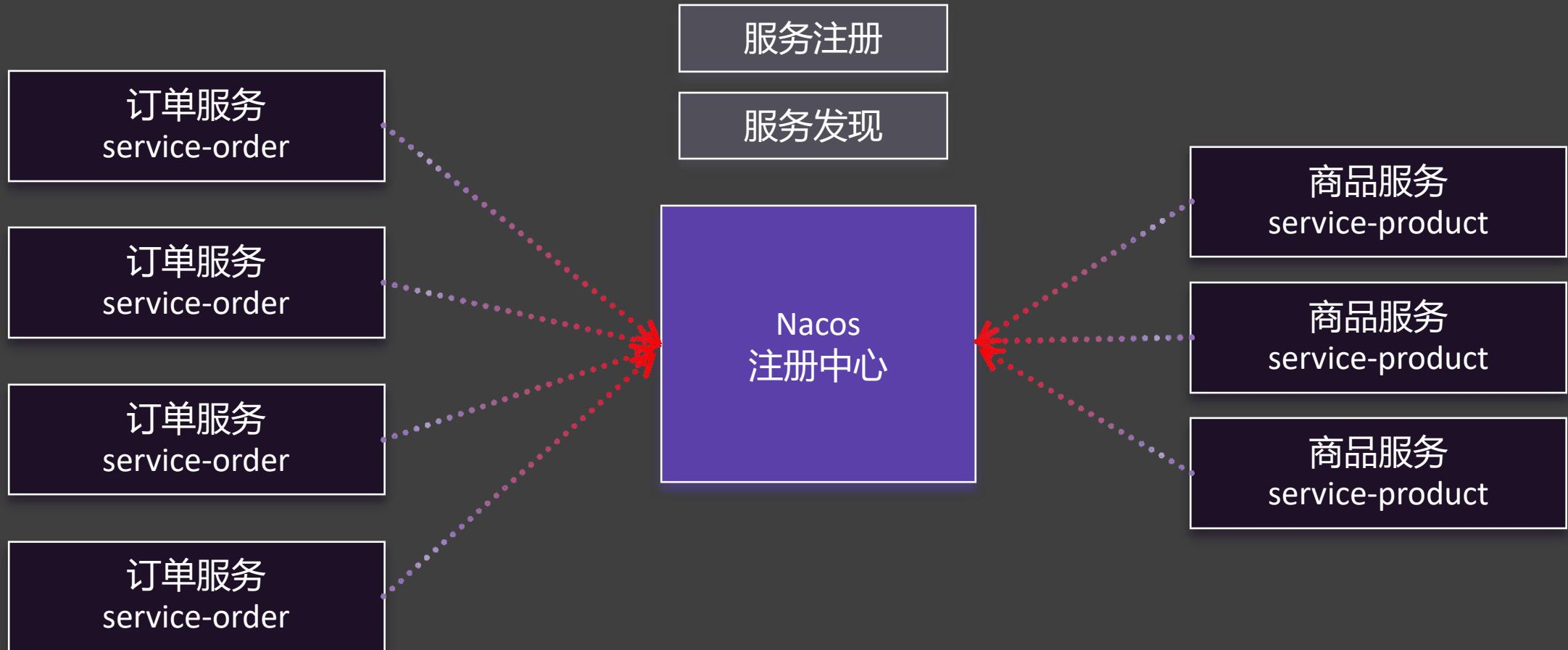
Sentinel
1.8.8

Seata
2.2.0

2. Nacos

注册中心、配置中心

注册中心



Nacos安装

- Nacos /na:kəʊs/ 是 Dynamic Naming and Configuration Service 的首字母简称，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。
- 官网：<https://nacos.io/zh-cn/docs/v2/quickstart/quick-start.html>
- 安装：
 - 下载安装包 【2.4.3】
 - 启动命令： startup.cmd -m standalone

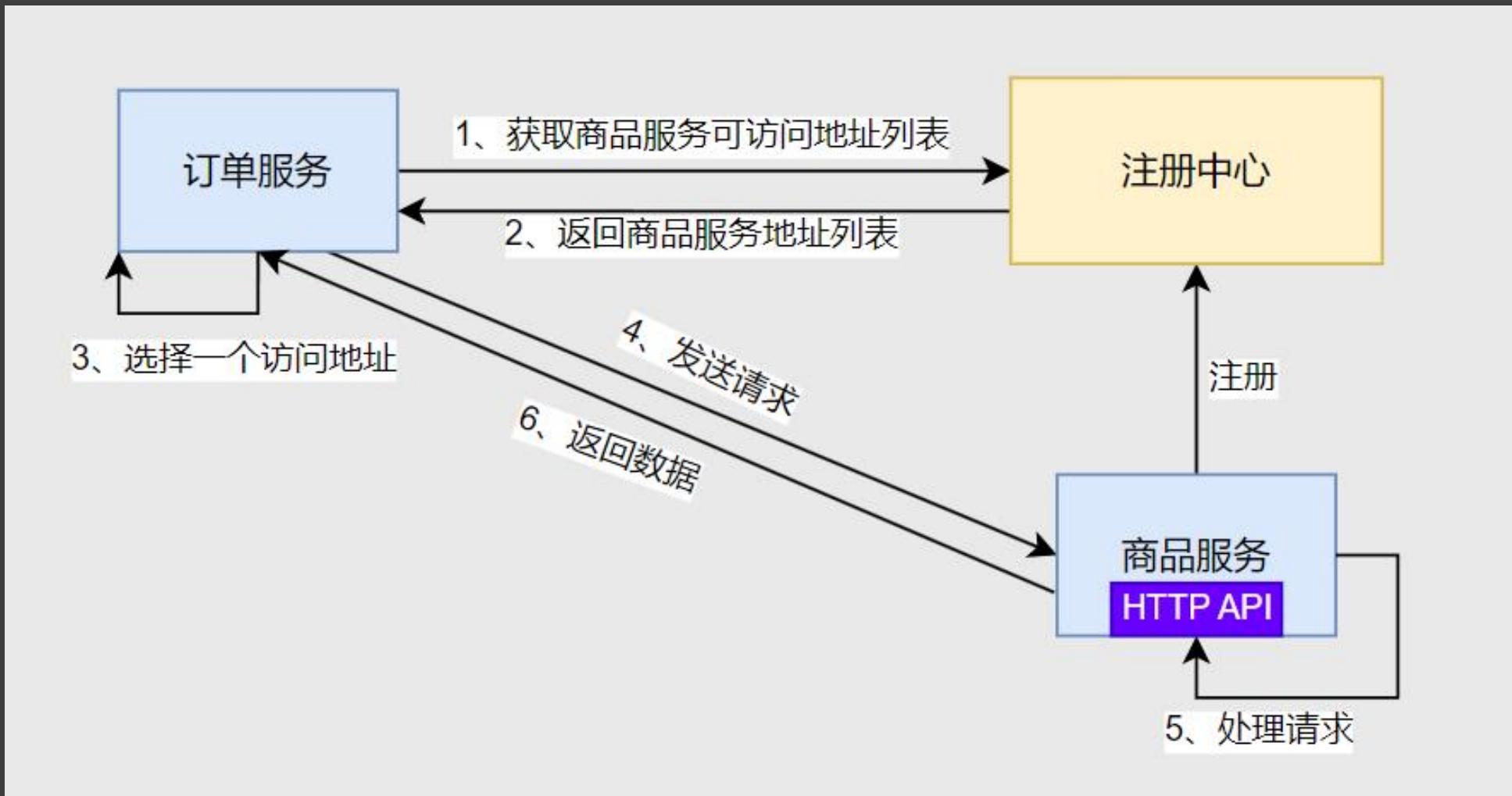
注册中心 - 服务注册

流程	内容	核心
步骤1	启动微服务	SpringBoot 微服务web项目启动
步骤2	引入服务发现依赖	spring-cloud-starter-alibaba-nacos-discovery
步骤3	配置Nacos地址	spring.cloud.nacos.server-addr=127.0.0.1:8848
步骤4	查看注册中心效果	访问 http://localhost:8848/nacos
步骤5	集群模式启动测试	单机情况下通过改变端口模拟微服务集群

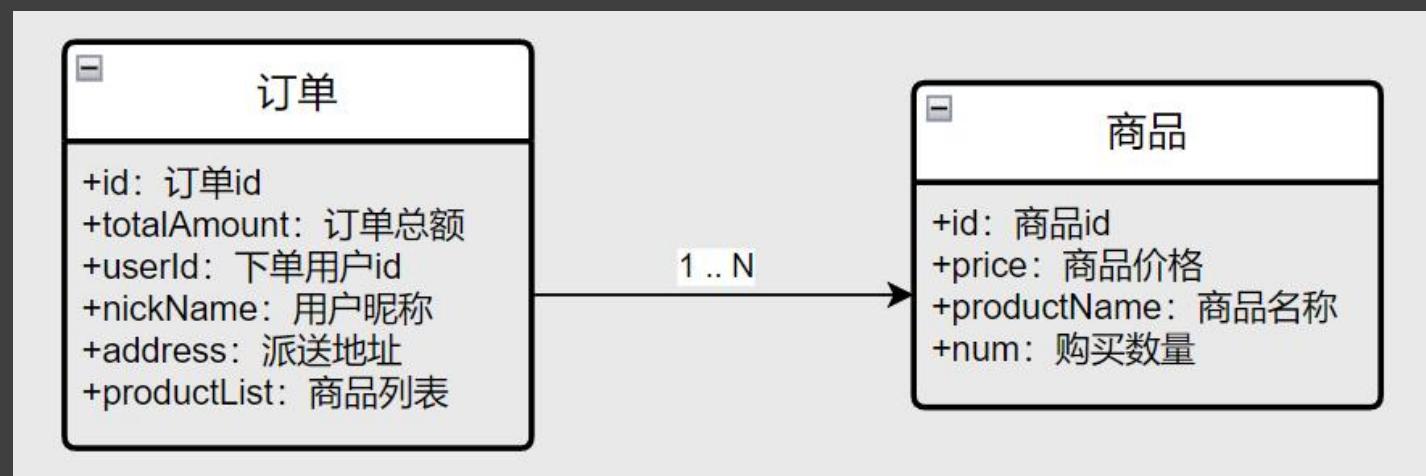
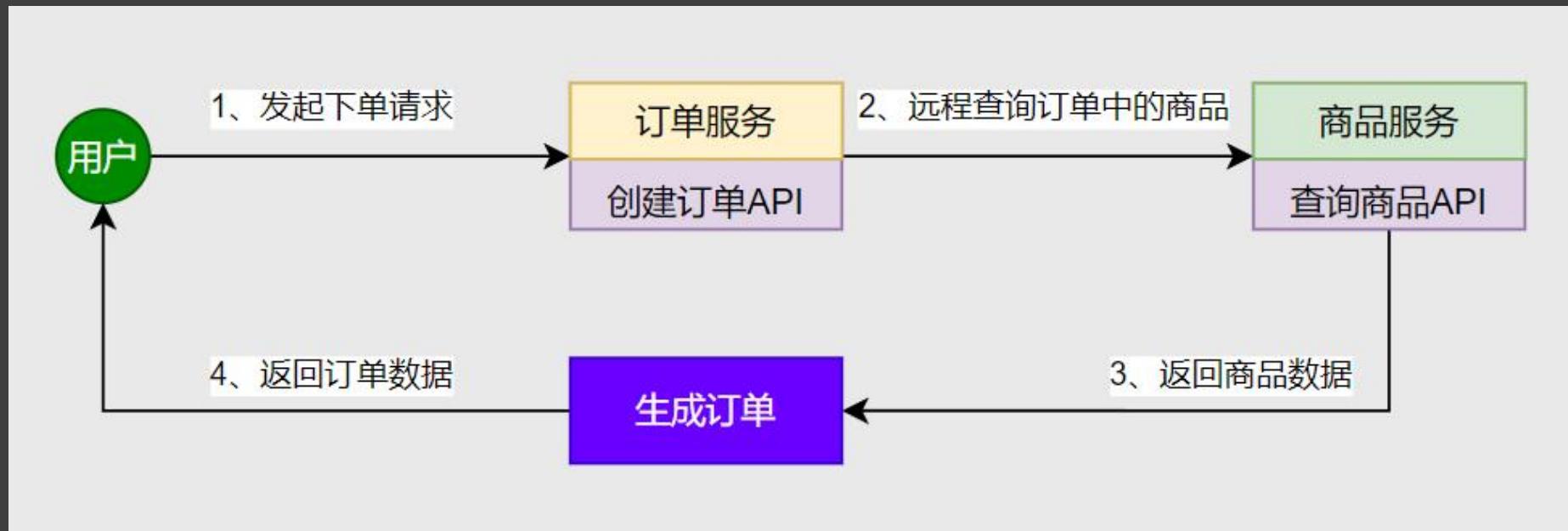
注册中心 - 服务发现

流程	内容	核心
步骤1	开启服务发现功能	@EnableDiscoveryClient
步骤2	测试服务发现API	DiscoveryClient
步骤3	测试服务发现API	NacosServiceDiscovery

远程调用 - 基本流程



远程调用 - 下单场景



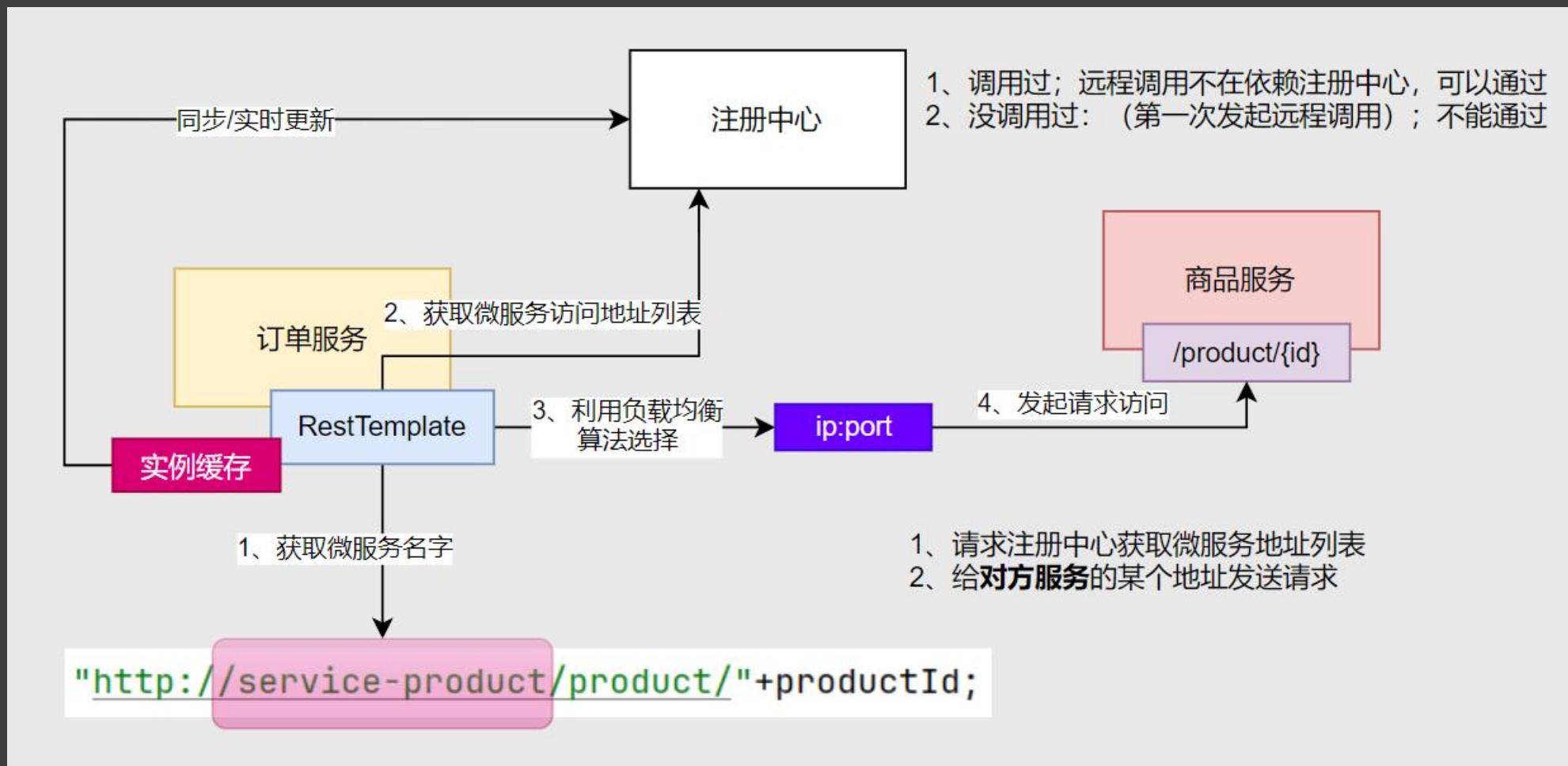
远程调用 - 实现步骤

流程	内容	核心
步骤1	引入负载均衡依赖	spring-cloud-starter-loadbalancer
步骤2	测试负载均衡API	LoadBalancerClient
步骤3	测试远程调用	RestTemplate
步骤4	测试负载均衡调用	@LoadBalanced

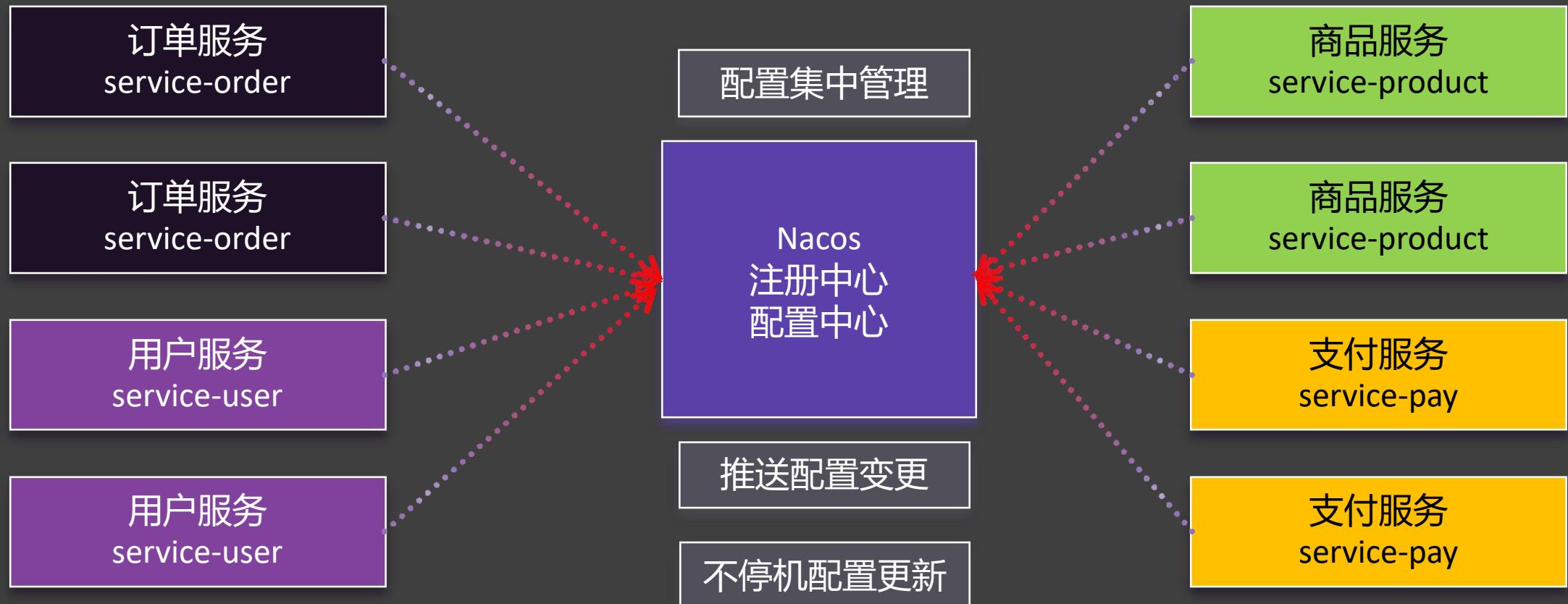
思考：注册中心宕机，远程调用还能成功吗？

远程调用 - 面试题

思考：注册中心宕机，远程调用还能成功吗？



配置中心



配置中心 - 基本使用

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

```
spring.cloud.nacos.server-addr=127.0.0.1:8848
spring.config.import=nacos:service-order.properties
```

数据集： service-order.properties

内容：

```
1  order.timeout=10min
2  order.auto-confirmed=7d
```

启动Nacos

引入依赖

application.properties 配置

创建 data-id (数据集)

配置中心 - 动态刷新

- 使用步骤

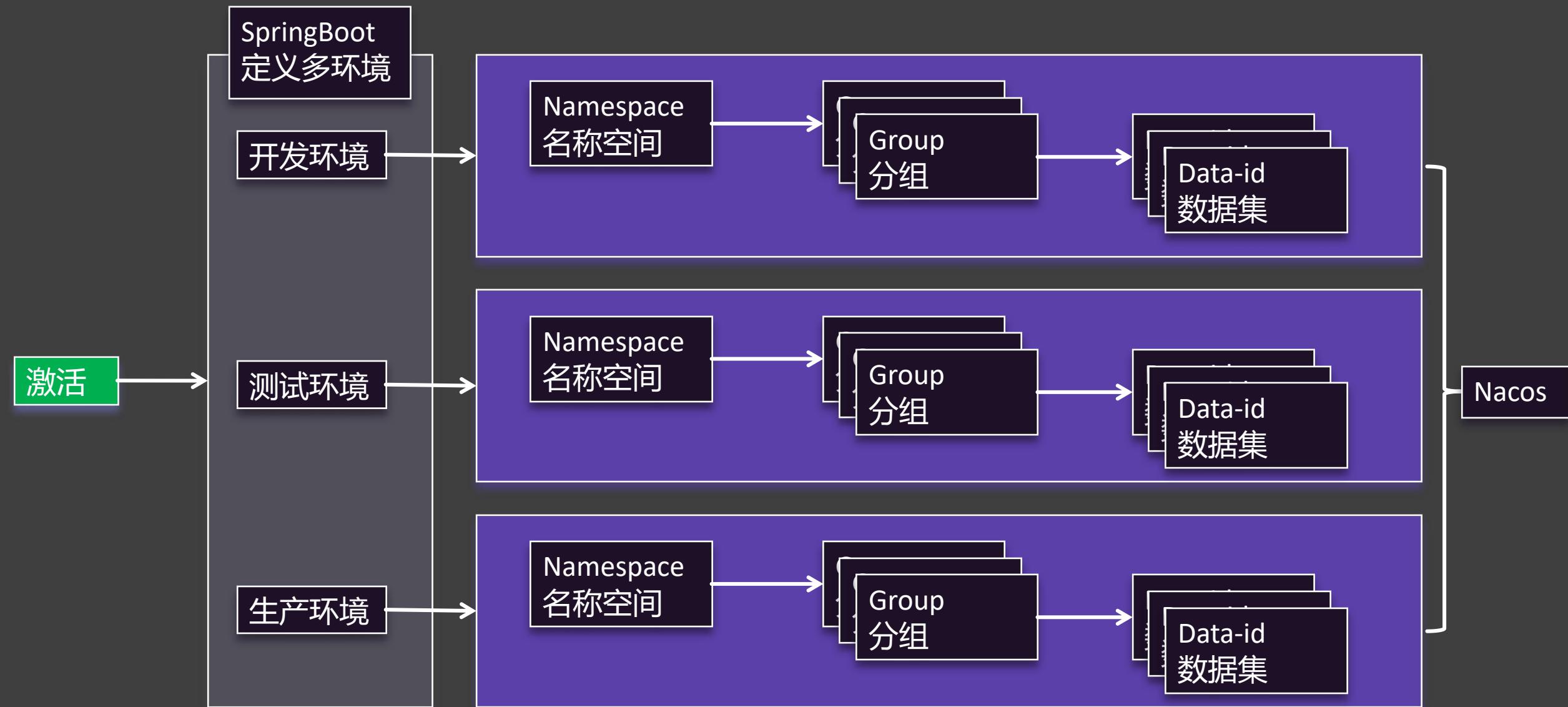
- @Value("\${xx}") 获取配置 + @RefreshScope 实现自动刷新
- @ConfigurationProperties 无感自动刷新
- NacosConfigManager 监听配置变化

思考： Nacos中的数据集 和 application.properties 有相同的 配置项，哪个生效？

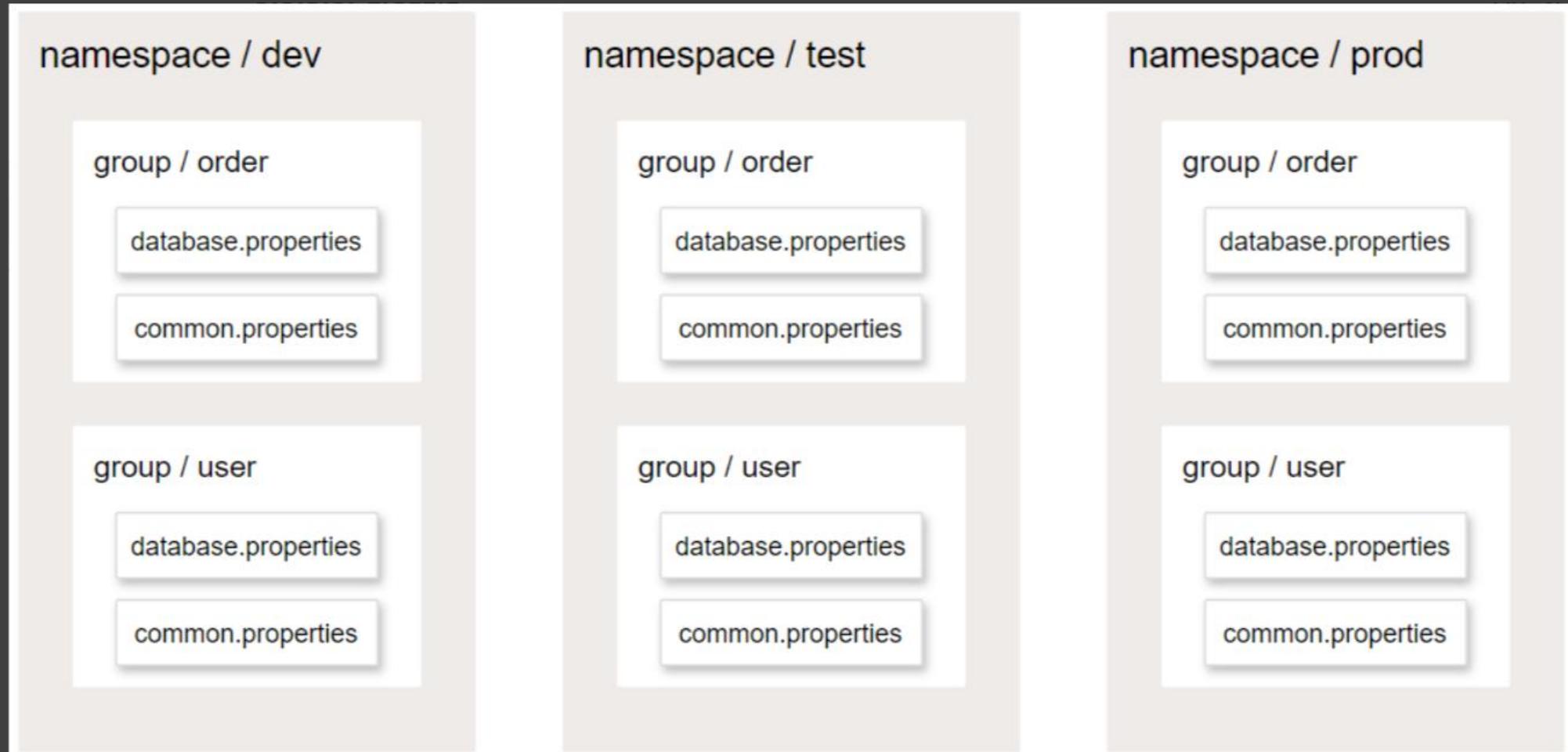
配置中心 - 数据隔离

- 需求描述
 - 项目有多套环境: dev, test, prod
 - 每个微服务, 同一种配置, 在每套环境的值都不一样。
 - 如: database.properties
 - 如: common.properties
 - 项目可以通过切换环境, 加载本环境的配置
- 难点
 - 区分多套环境
 - 区分多种微服务
 - 区分多种配置
 - 按需加载配置

配置中心 - 数据隔离



配置中心 - 数据隔离



namespace、dataId、group 配合 `spring.config.activate.on-profile` 实现配置环境隔离

总结

注册中心

1. 引入 **spring-cloud-starter-alibaba-nacos-discovery** 依赖，配置Nacos地址
2. **@EnableDiscoveryClient** 开启服务发现功能

扩展：

1. **DiscoveryClient** 获取服务实例列表
2. **LoadBalancerClient** 负载均衡选择一个实例
(需要引入 `spring-cloud-starter-loadbalancer`)
3. **RestTemplate** 可以发起远程调用

配置中心

1. 引入 **spring-cloud-starter-alibaba-nacos-config** 依赖，配置Nacos地址
2. 添加 **数据集** (data-id) , 使用 **spring.config.import** 导入数据集
3. **@Value + @RefreshScope** 取值 + 自动刷新
4. **@ConfigurationProperties**批量绑定自动刷新
5. **NacosConfigManager** 监听配置变化

扩展：

配置优先级； namespace区分环境、 group区分微服务、 data-id区分配置 实现 数据隔离+环境切换

3. OpenFeign

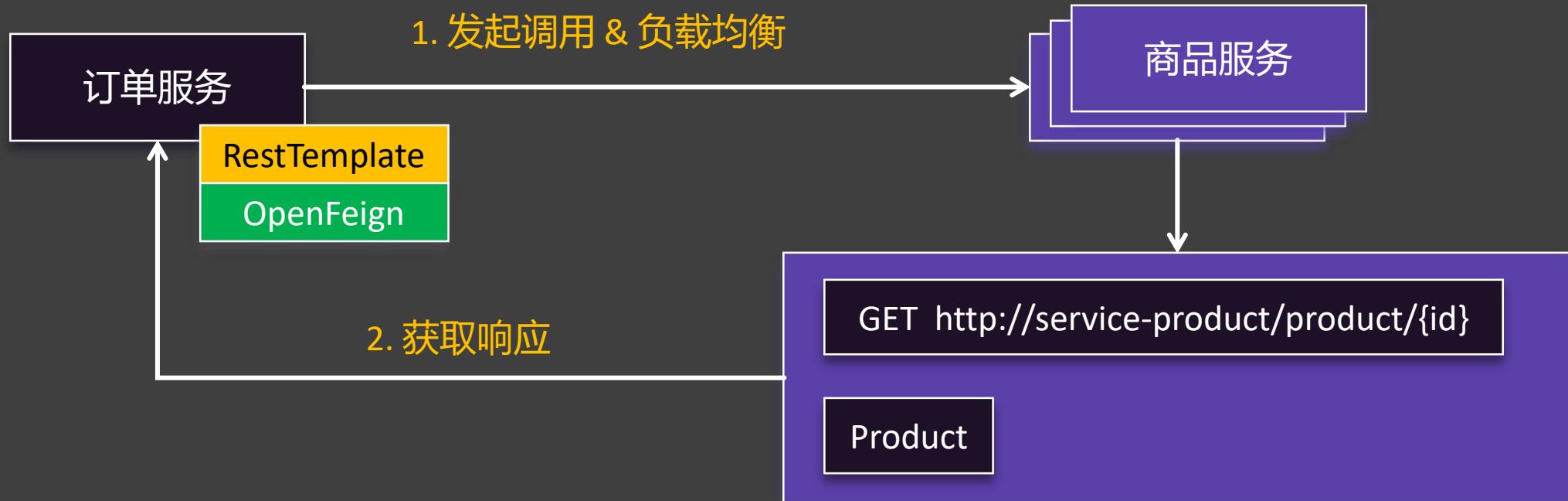
远程调用

Declarative REST Client

- 声明式 REST 客户端 vs 编程式 REST 客户端 (RestTemplate)
- 注解驱动
 - 指定远程地址: @FeignClient
 - 指定请求方式: @GetMapping、@PostMapping、@DeleteMapping ...
 - 指定携带数据: @RequestHeader、@RequestParam、@RequestBody ...
 - 指定结果返回: 响应模型

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

远程调用 - 业务API



```
@FeignClient(value = "service-product")
public interface ProductFeignClient {
```

远程调用 - 第三方API

POST http://aliv18.data.moji.com/wapi/json/alicityweather/condition

请求头	Authorization	APPCODE 93b7e19861a24c519a7548b17dc16d75
-----	---------------	---

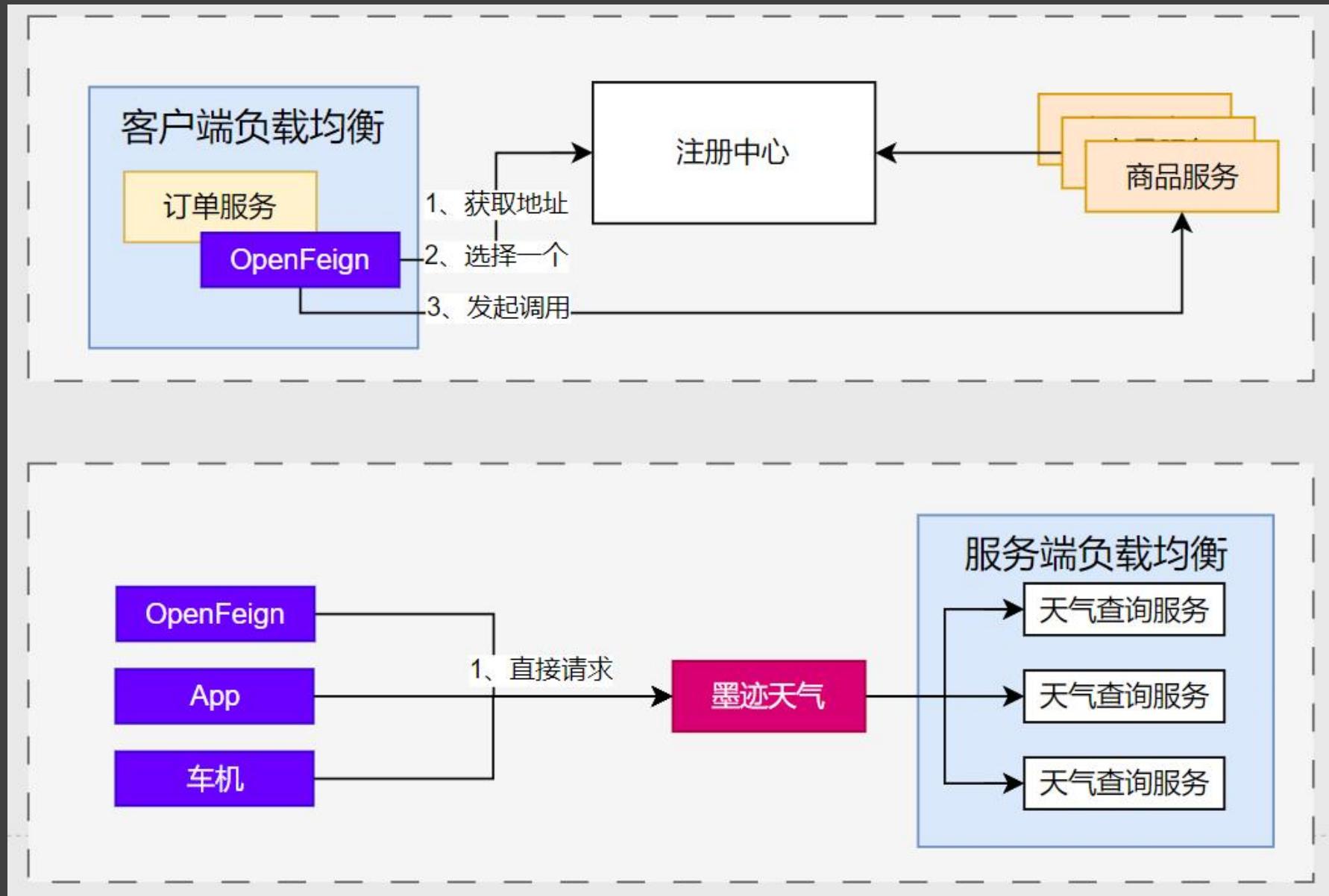


请求参数	token	50b53ff8dd7d9fa320d3d3ca32cf8ed1
	cityId	2182

```
{"code":0,"data":{"city":{"cityId":2182,"counname":"中国","ianatimezone":"Asia/Shanghai","name":"西安市","pname":"陕西省","secondaryname":"陕西省","timezone":"8"},"condition":{"condition":{"conditionId":5,"humidity":37,"icon":0,"pressure":984,"realFeel":3,"sunRise":"2024-12-25 07:49:00","sunSet":"2024-12-25 17:41:00","temp":6,"tips":"天冷了，该加衣服了！","updateTime":2024-12-25 12:15:08,"uvi":4,"vis":16100,"windDegrees":225,"windDir": "西南风","windLevel":2,"windSpeed":2.39}},"msg":"success","rc":{"c":0,"p": "success"}}
```

- 小技巧：如何编写好OpenFeign声明式的远程调用接口
 - 业务API：直接复制对方Controller签名即可
 - 第三方API：根据接口文档确定请求如何发

面试题：客户端负载均衡与服务端负载均衡区别



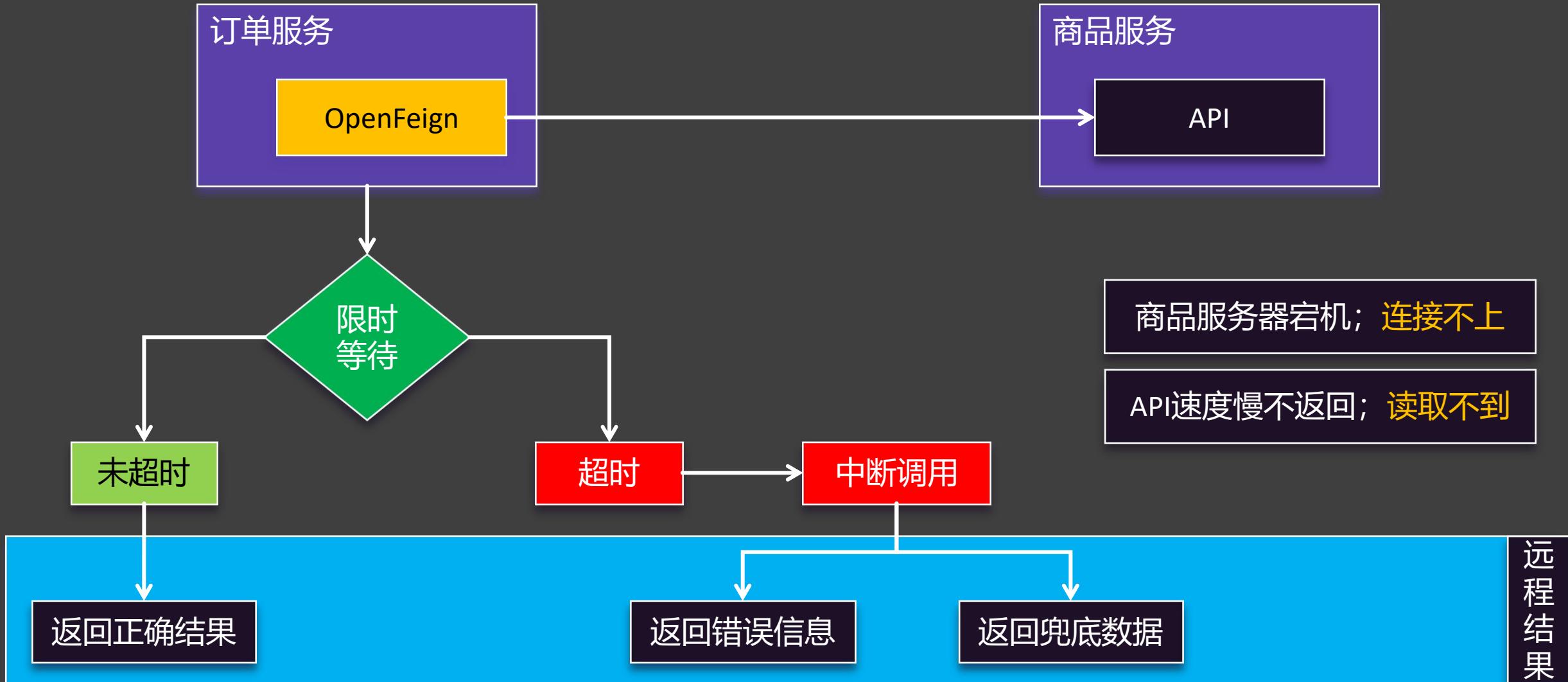
进阶用法 - 日志



```
logging:  
  level:  
    com.atguigu.order.feign: debug
```

```
@Bean  
Logger.Level feignLoggerLevel() {  
    return Logger.Level.FULL;  
}
```

进阶用法 - 超时控制



进阶用法 - 超时控制

connectTimeout

连接超时

readTimeout

读取超时

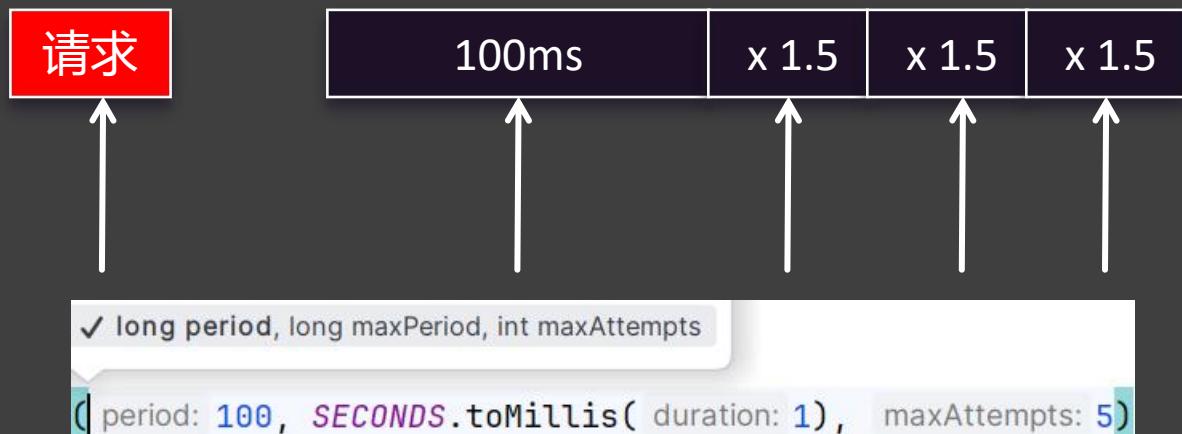
```
oo options = {Request$Options@8449}
  oo connectTimeout = 10
  > oo connectTimeoutUnit = {TimeUnit@10518} "SECONDS"
  oo readTimeout = 60
  > oo readTimeoutUnit = {TimeUnit@10518} "SECONDS"
```



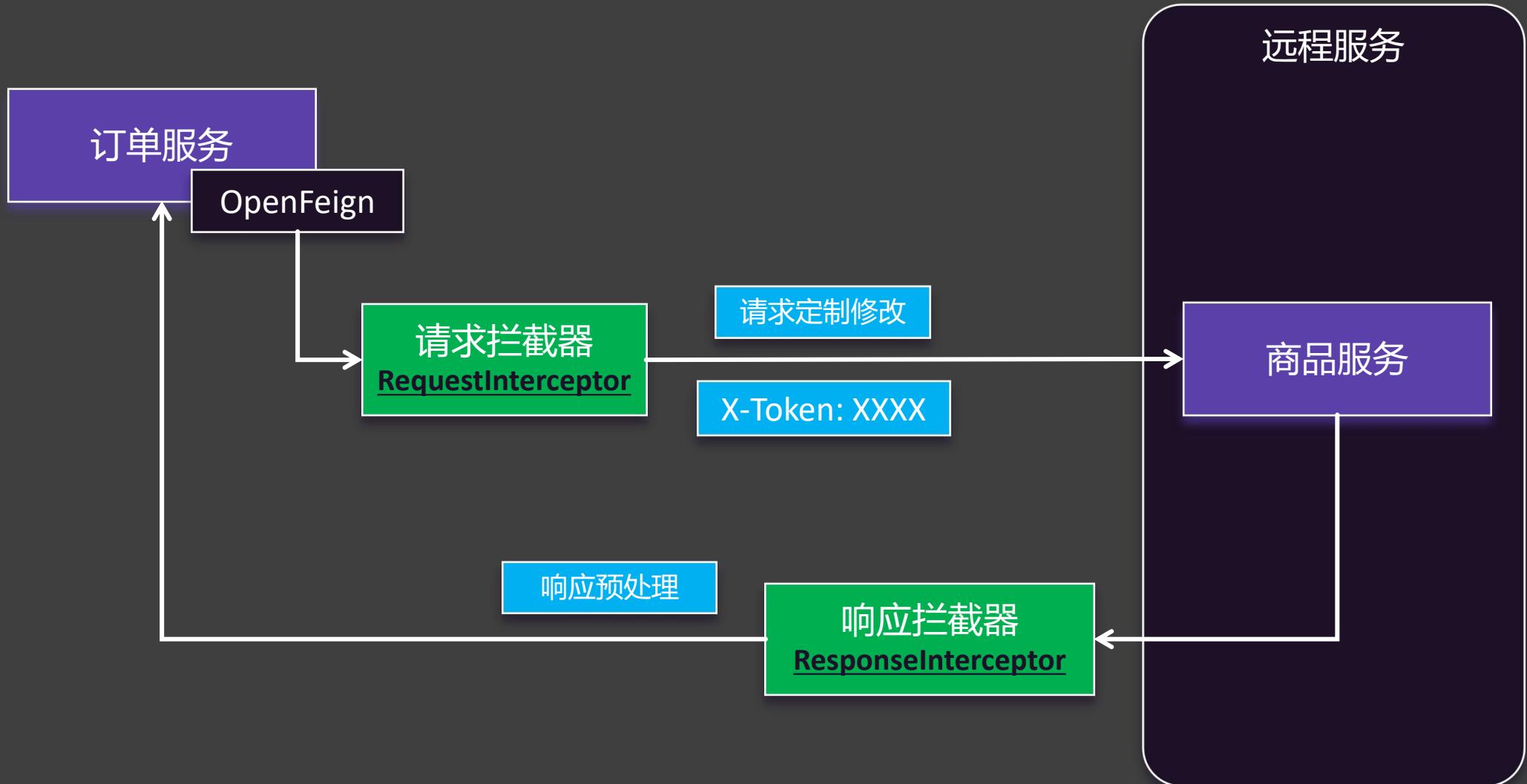
进阶用法 - 重试机制

- 远程调用超时失败后，还可以进行多次尝试，如果某次成功返回ok，如果多次依然失败则结束调用，返回错误

A bean of `Retryer.NEVER_RETRY` with the type `Retryer` is created by default which will disable retrying. Notice this retrying behavior is different from the Feign default one, where it will automatically retry `IOExceptions`, treating them as transient network related exceptions, and any `RetryableException` thrown from an `ErrorDecoder`.

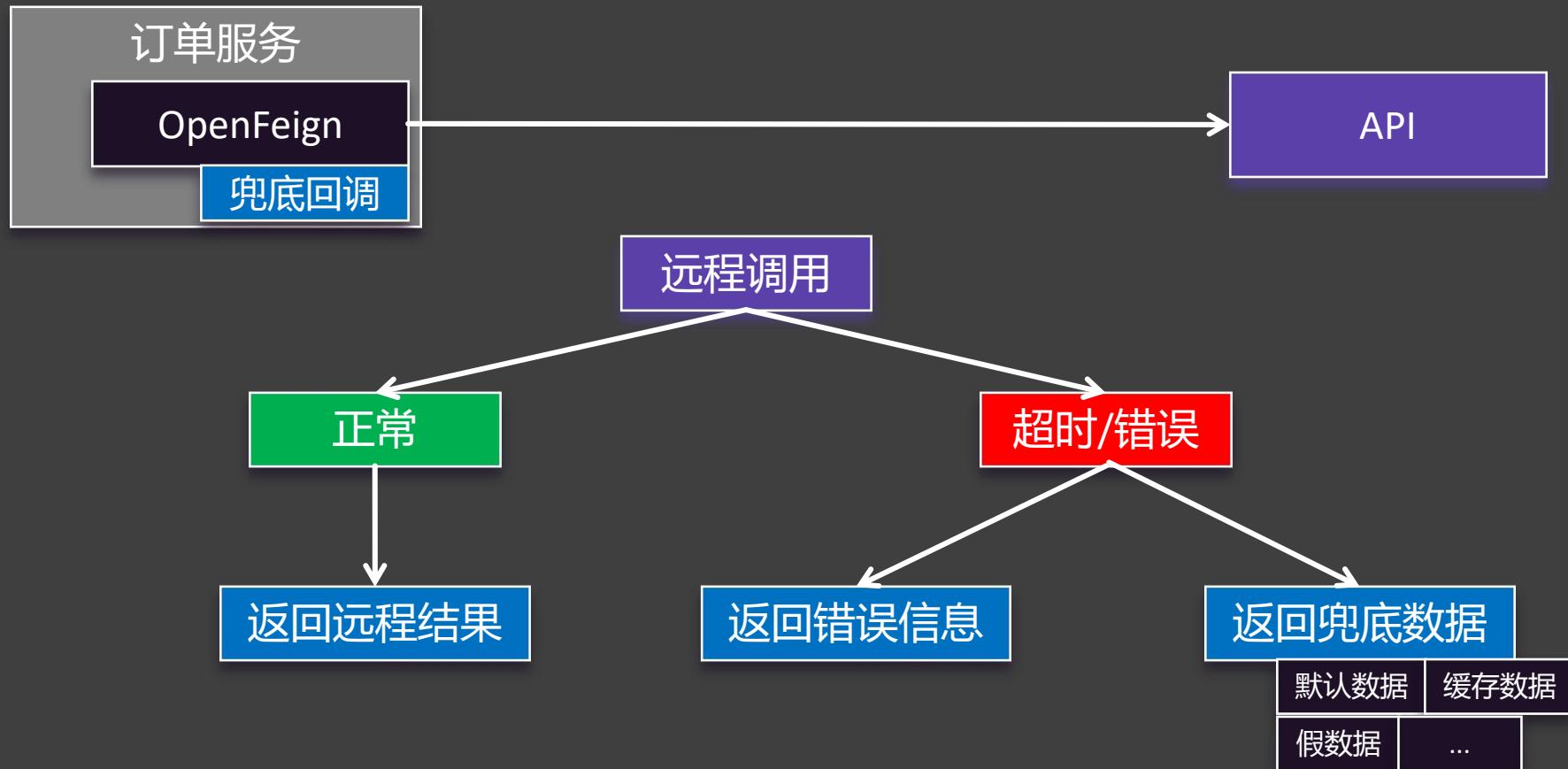


进阶用法 - 拦截器



进阶用法 - Fallback

- Fallback：兜底返回
 - 注意：此功能需要整合 Sentinel 才能实现



OpenFeign - 总结

- 1. 熟练编写 OpenFeign 远程调用客户端
- 2. 熟练配置 OpenFeign 客户端属性
 - 连接超时
 - 读取超时
 -
- 3. 掌握 拦截器 用法
- 4. 掌握 Fallback 兜底返回机制 及 用法

4. Sentinel

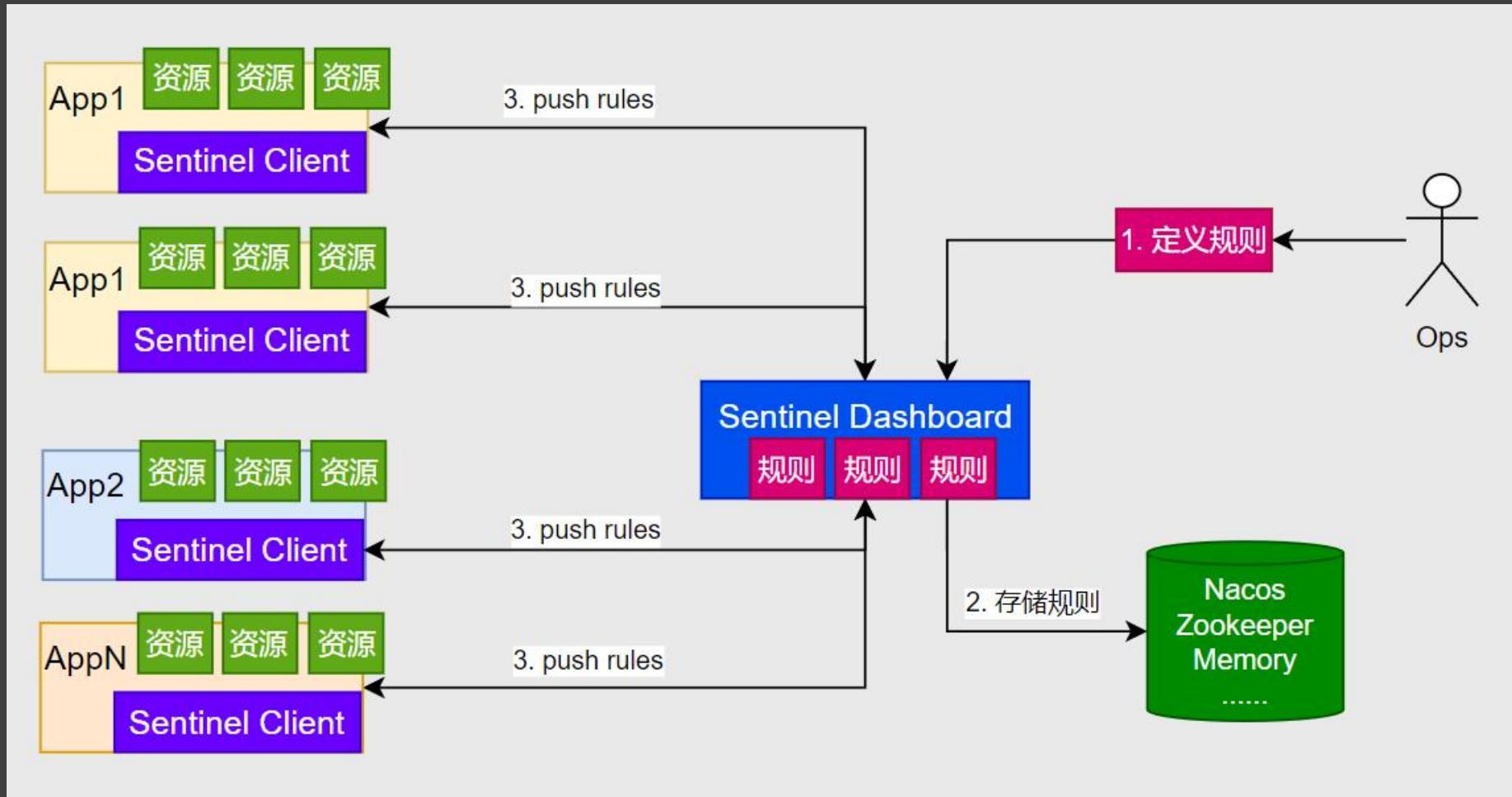
服务保护（限流、熔断降级）

功能介绍

- 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。Spring Cloud Alibaba Sentinel 以流量为切入点，从流量控制、流量路由、熔断降级、系统自适应过载保护、热点流量防护等多个维度保护服务的稳定性。



架构原理



资源&规则

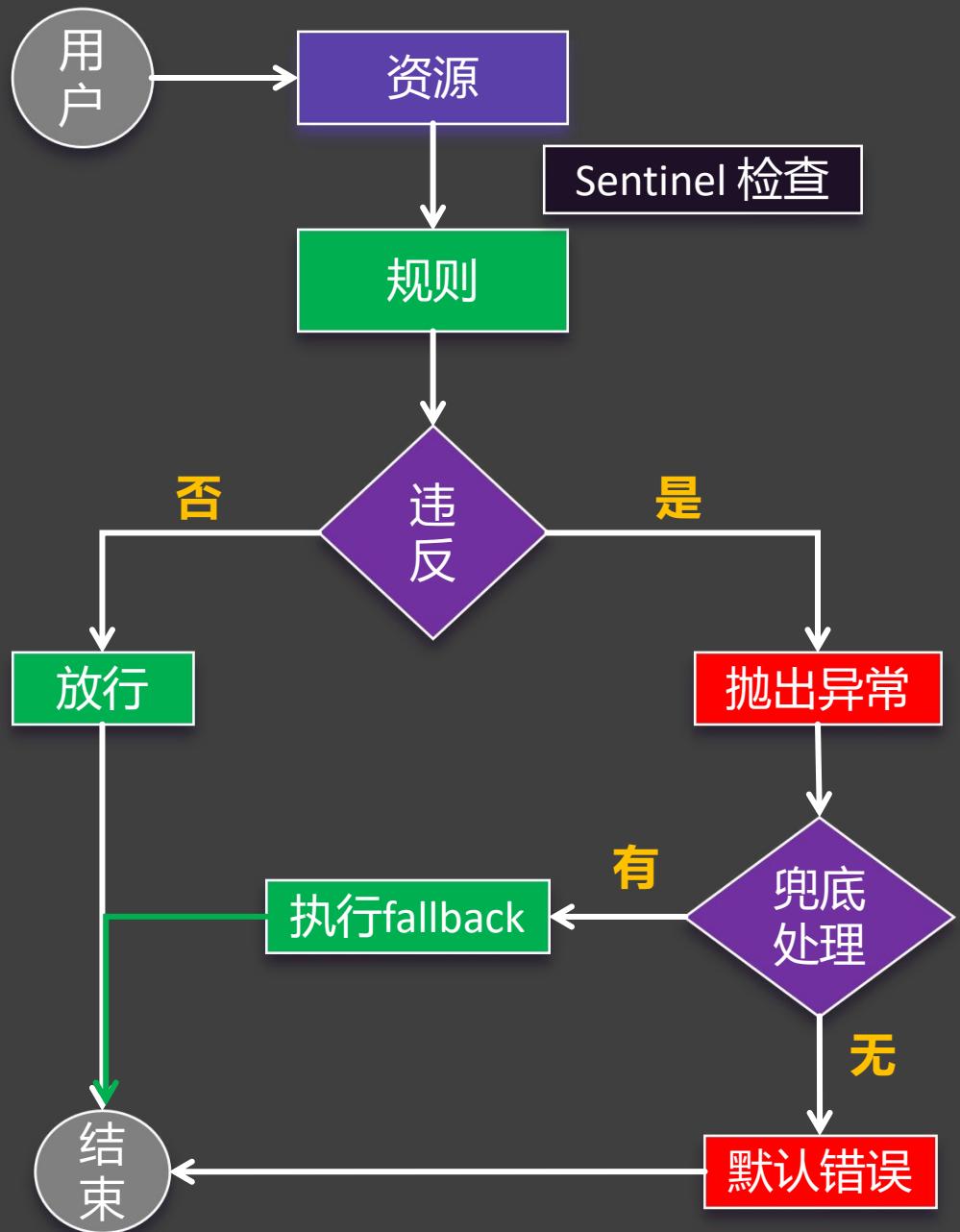
- 定义资源：

- 主流框架自动适配（Web Servlet、Dubbo、Spring Cloud、gRPC、Spring WebFlux、Reactor）；
所有Web接口均为资源
- 编程式：SphU API
- 声明式：@SentinelResource

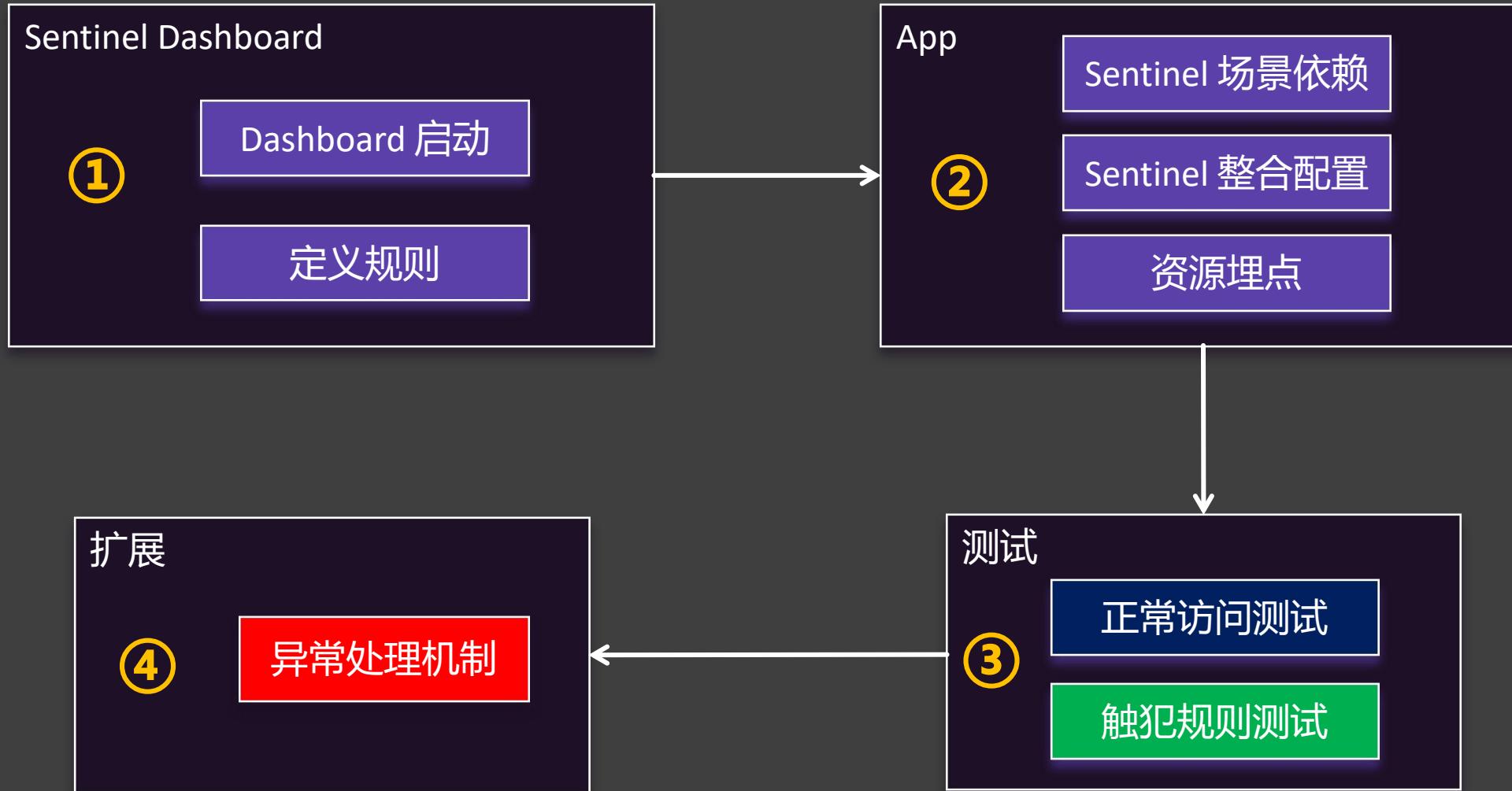
- 定义规则：

- 流量控制（FlowRule）
- 熔断降级（DegradeRule）
- 系统保护（SystemRule）
- 来源访问控制（AuthorityRule）
- 热点参数（ParamFlowRule）

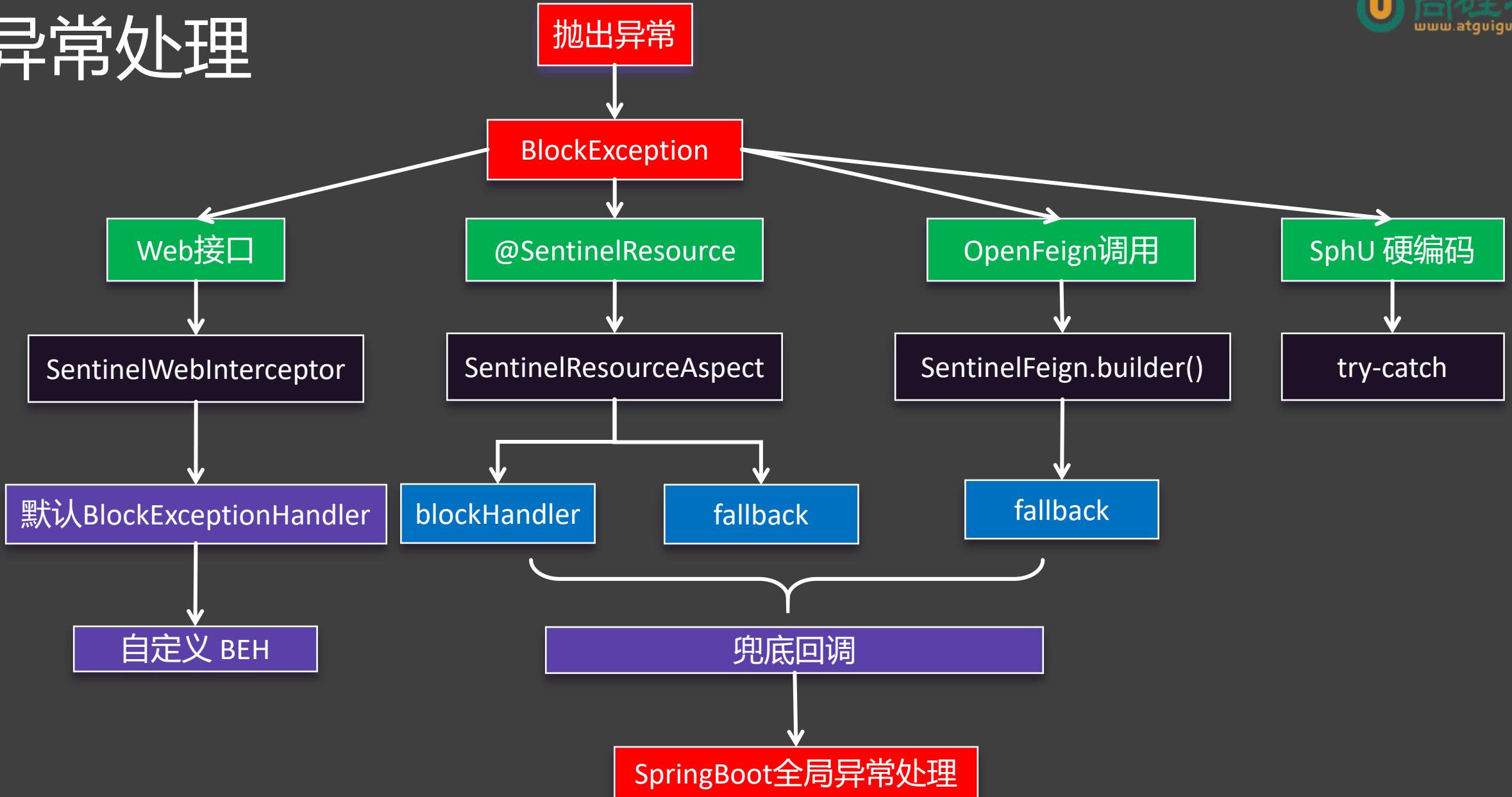
工作原理



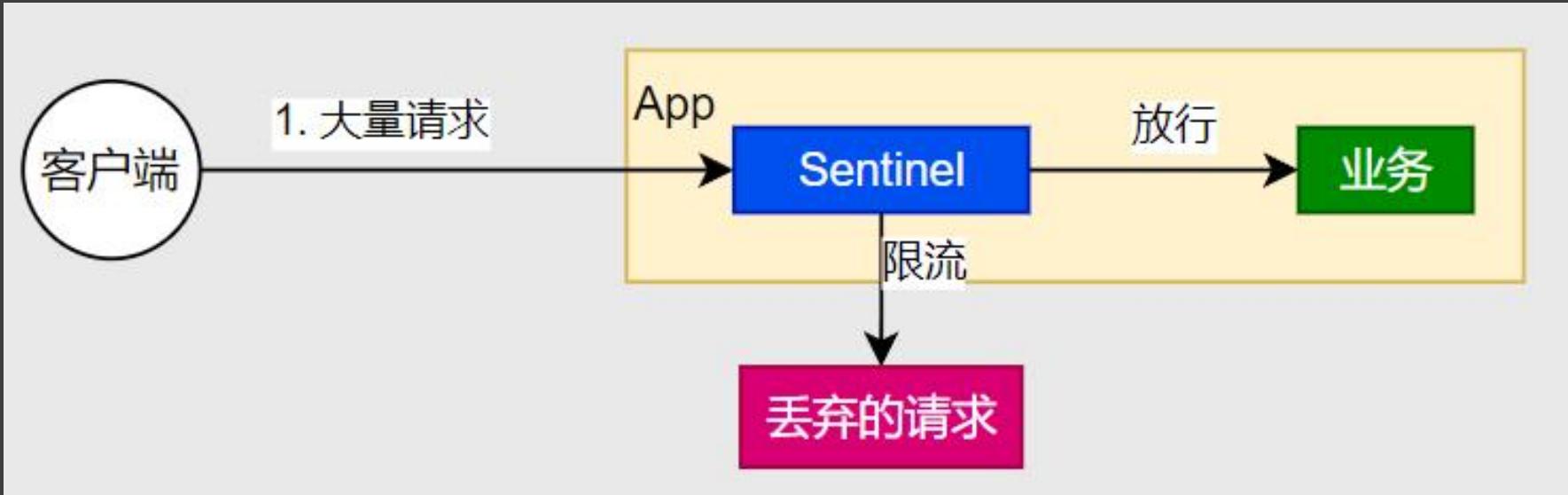
整合使用



异常处理



规则 - 流量控制 (FlowRule)



限制多余请求，从而保护系统资源不被耗尽

规则 - 流量控制 (FlowRule)

新增流控规则

资源名	/create		
针对来源	default		
阈值类型	<input checked="" type="radio"/> QPS <input type="radio"/> 并发线程数	单机阈值	1
是否集群	<input type="checkbox"/>		
流控模式	<input checked="" type="radio"/> 直接 <input type="radio"/> 关联 <input type="radio"/> 链路		
流控效果	<input checked="" type="radio"/> 快速失败 <input type="radio"/> Warm Up <input type="radio"/> 排队等待		
关闭高级选项			

规则 - 流量控制 (FlowRule)

属性	说明	默认值
resource	资源名，资源名是限流规则的作用对象	
count	限流阈值	
grade	限流阈值类型, QPS 模式 (1) 或 并发线程数模式 (0)	QPS 模式
limitApp	流控针对的调用来源	default, 代表不区分调用来源
strategy	调用关系限流策略: 直接、链路、关联	直接 (根据资源本身)
controlBehavior	流控效果 (直接拒绝/WarmUp/匀速+排队等待) , 不支持按调用关系限流	直接拒绝
clusterMode	是否集群限流	否

规则 - 流量控制 (FlowRule) - 阈值类型

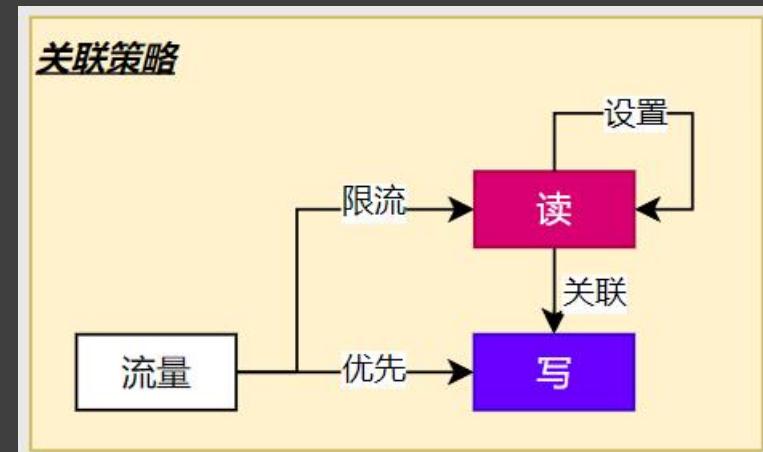
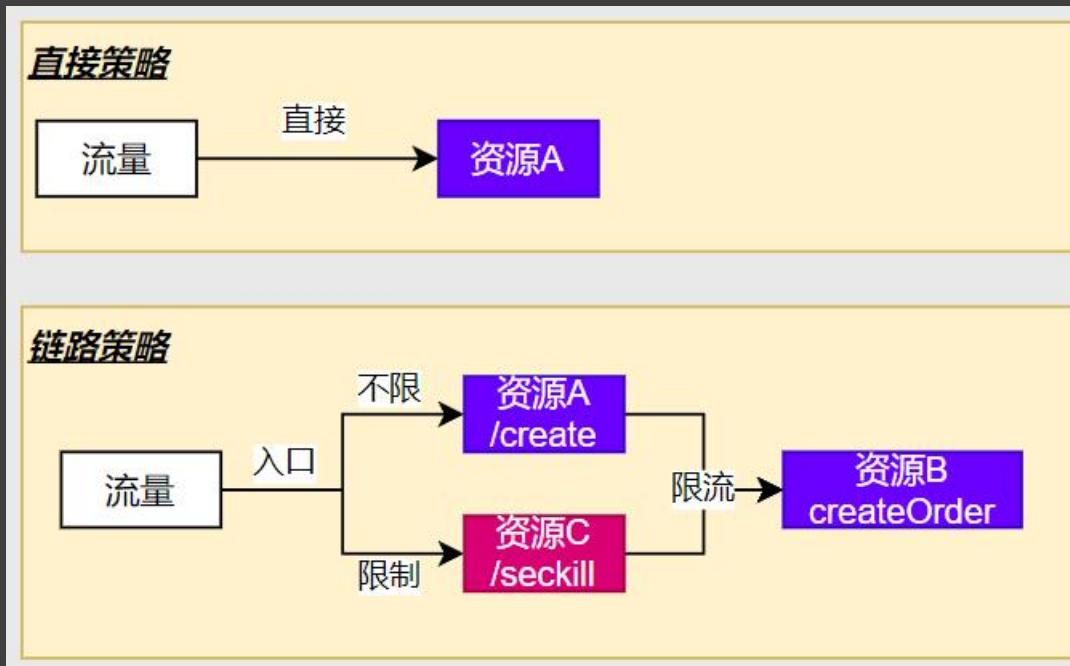


- QPS:
 - 统计每秒请求数
- 并发线程数:
 - 统计并发线程数

规则 - 流量控制 (FlowRule) - 流控模式



调用关系包括调用方、被调用方；一个方法又可能会调用其它方法，形成一个调用链路的层次关系；有了调用链路的统计信息，我们可以衍生出多种流量控制手段。



规则 - 流量控制 (FlowRule) - 流控效果

流控效果

- 快速失败
- Warm Up
- 排队等待

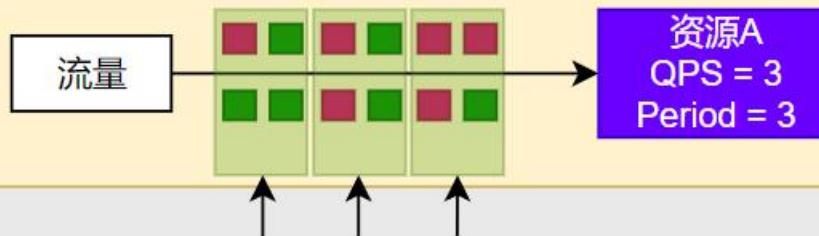
注意：只有快速失败支持流控模式（直接、关联、链路）的设置

漏桶算法

快速失败 (直接拒绝)

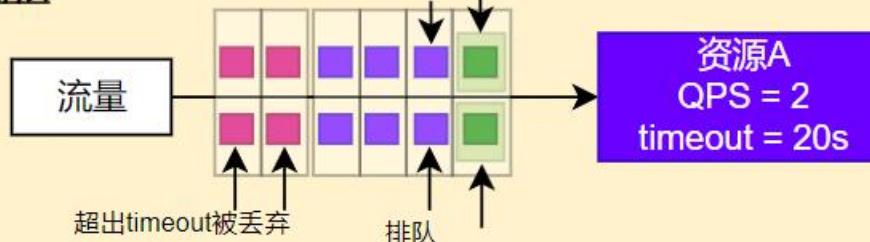


Warm Up (预热/冷启动)

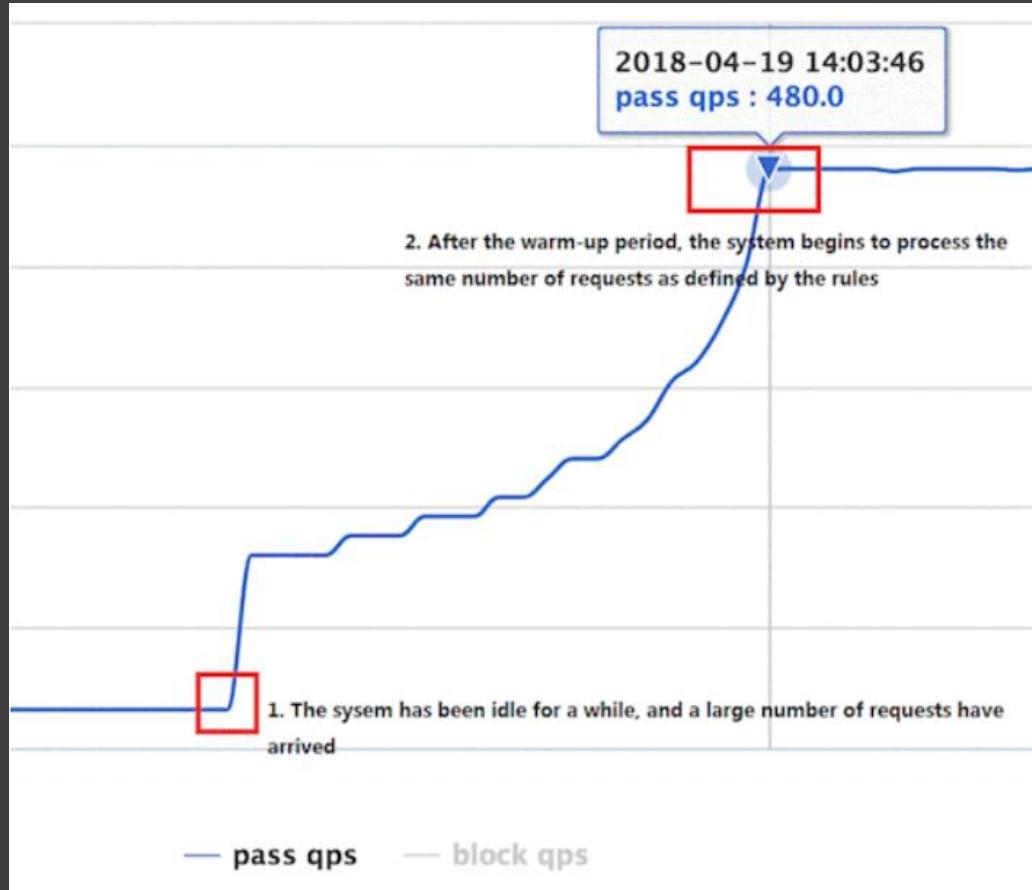


注：QPS=2，每秒2个，则每500ms一个，不支持QPS>1000

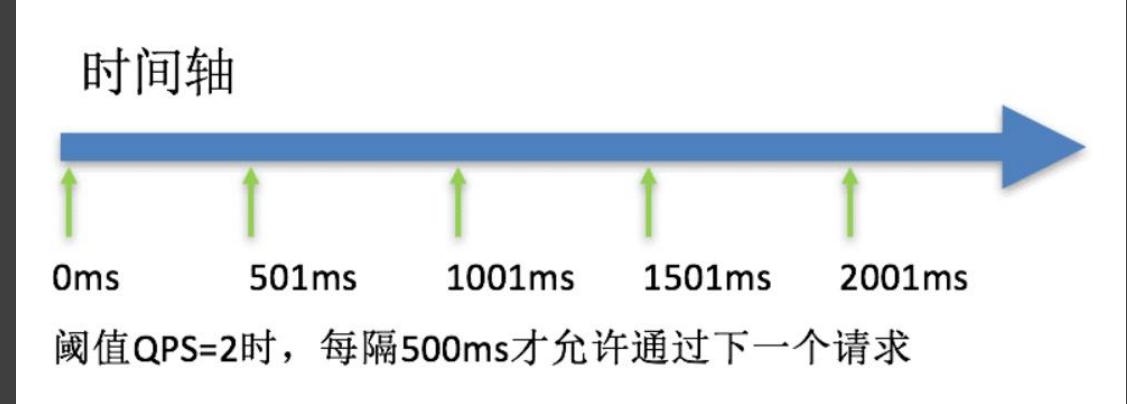
匀速排队



规则 - 流量控制 (FlowRule) - 流控效果

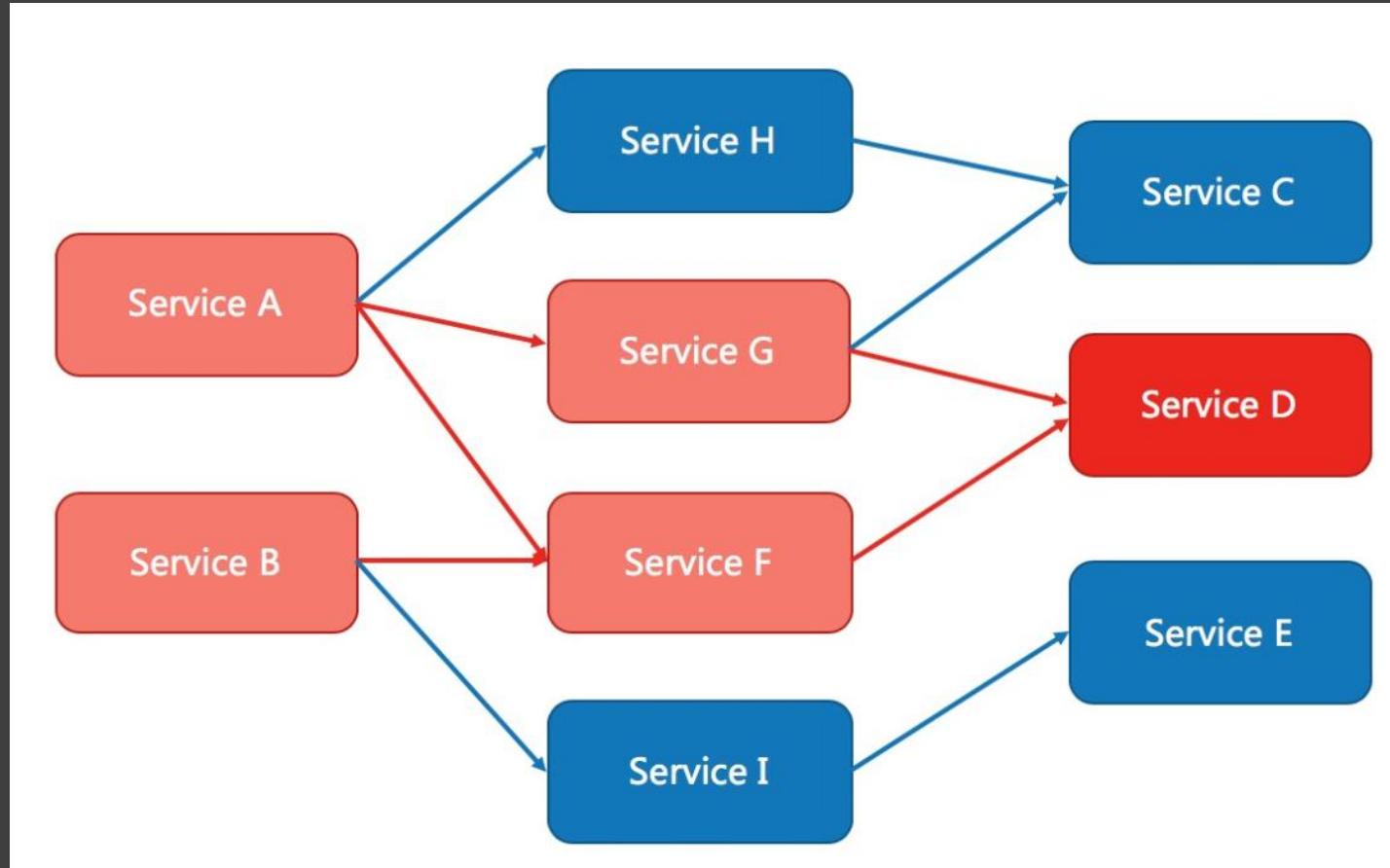


Warm Up



匀速排队

规则 - 熔断降级 (DegradeRule)



切断不稳定调用

快速返回不积压

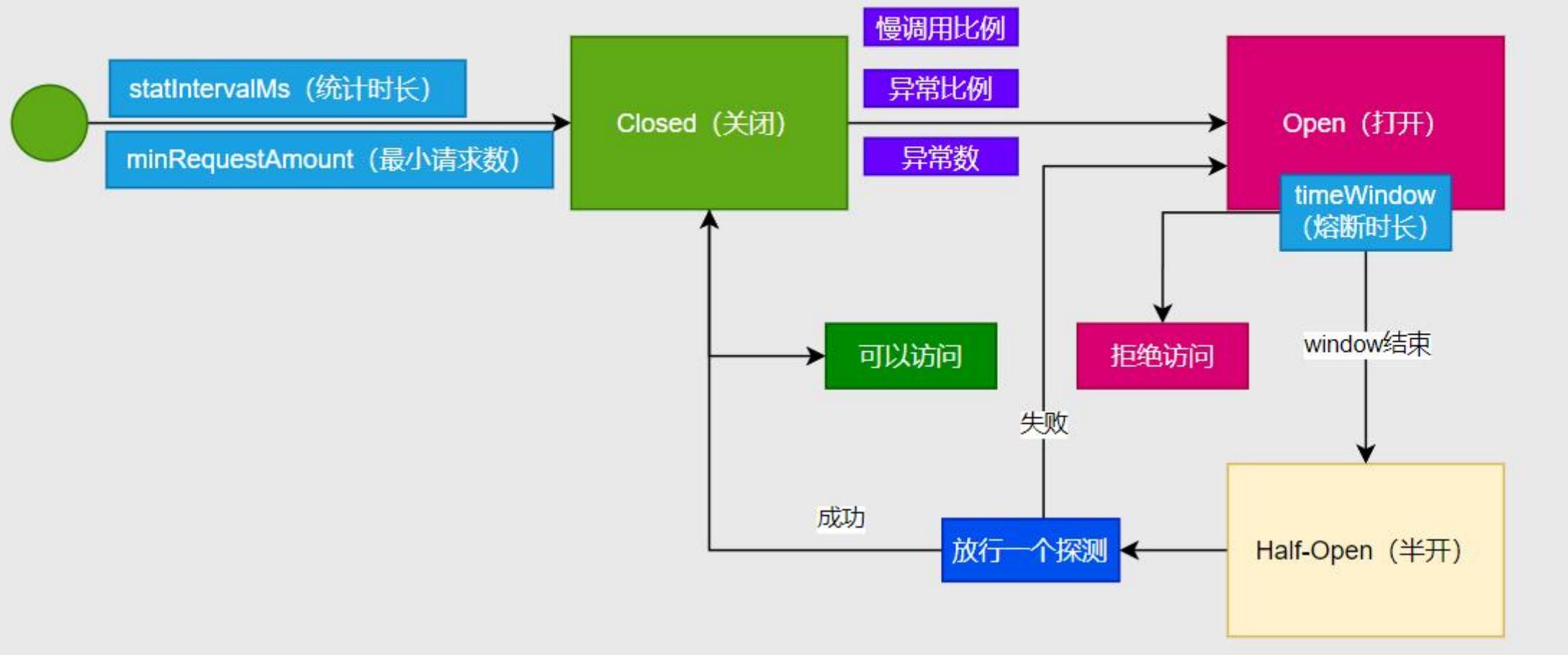
避免雪崩效应



最佳实践：熔断降级作为保护自身的手段，通常在客户端（调用端）进行配置。

规则 - 熔断降级

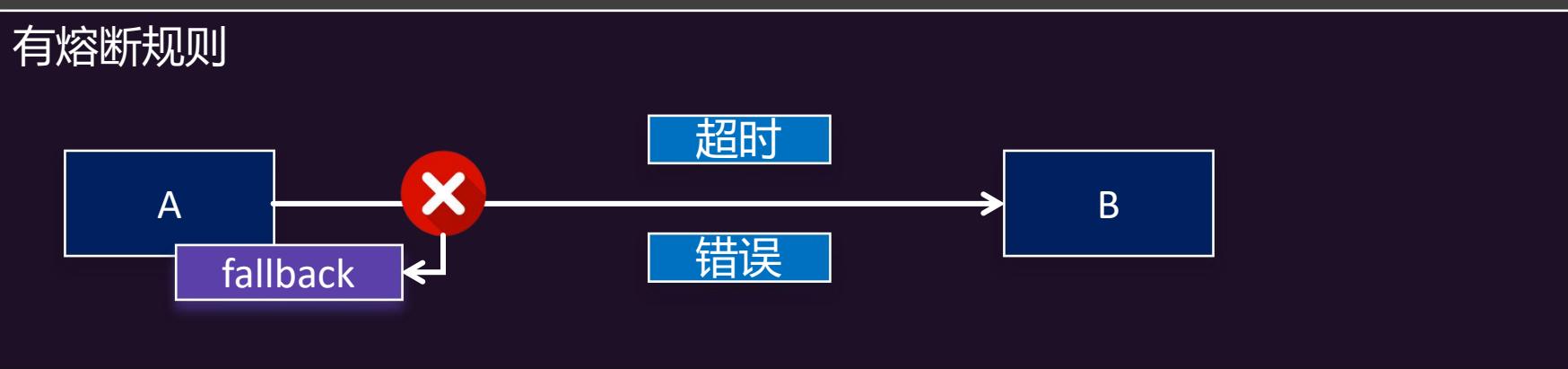
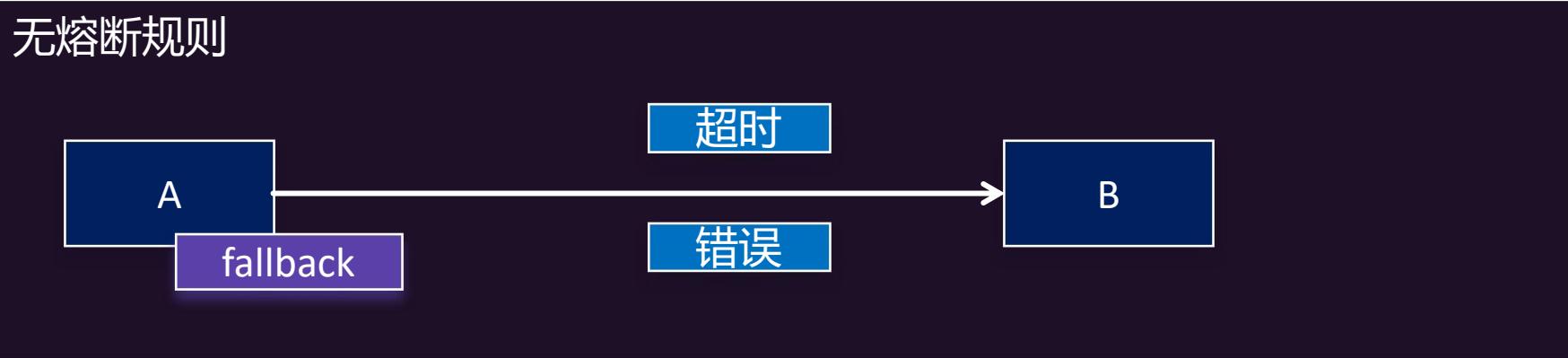
断路器工作原理



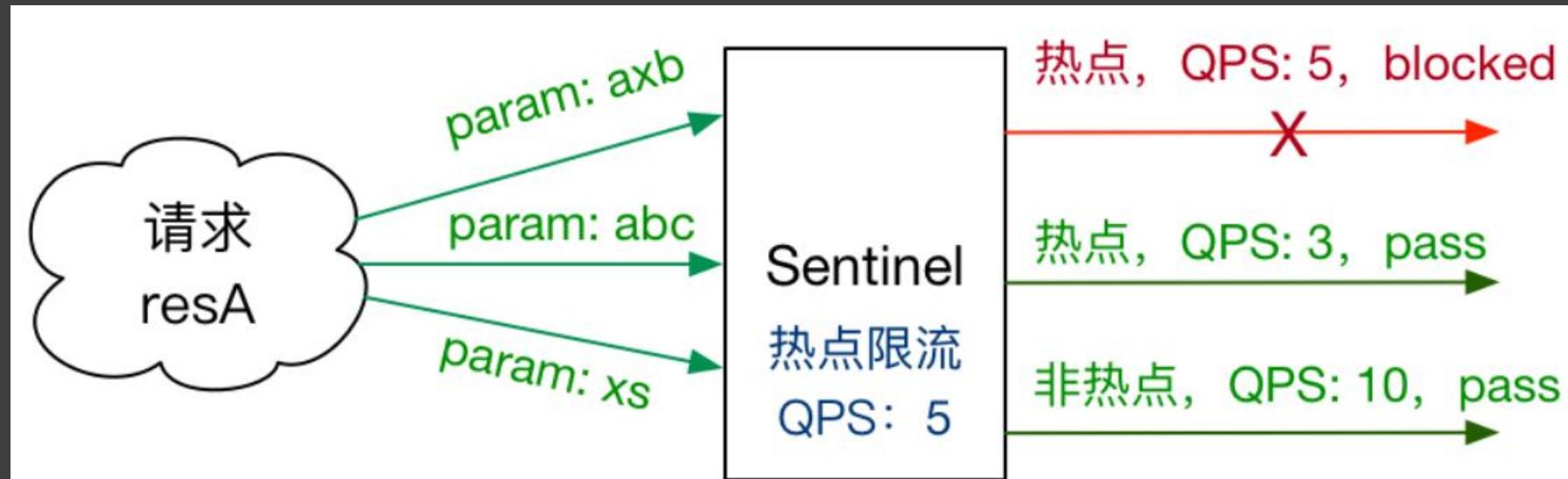
规则 - 熔断降级

属性	说明	默认值
resource	资源名，必填	
grade	熔断策略，支持 慢调用比例/异常比例/异常数 策略	慢调用比例
count	慢调用比例模式下为慢调用临界 RT（超出该值计为慢调用）； 异常比例/异常数模式下为对应的阈值	
timeWindow	熔断时长，单位为 s	
minRequestAmount	熔断触发的最小请求数，请求数小于该值时即使异常比率超出阈值也不会熔断（1.7.0 引入）	5
statIntervalMs	统计时长（单位为 ms），如 60*1000 代表分钟级（1.8.0 引入）	1000 ms
slowRatioThreshold	慢调用比例阈值，仅慢调用比例模式有效（1.8.0 引入）	

规则 - 熔断降级



规则 - 热点参数



需求1：每个用户秒杀 QPS 不得超过 1 (秒杀下单 userId 级别)

效果：携带此参数的参与流控，不携带不流控

需求2：6号用户是vvip，不限制QPS (例外情况)

需求3：666号是下架商品，不允许访问

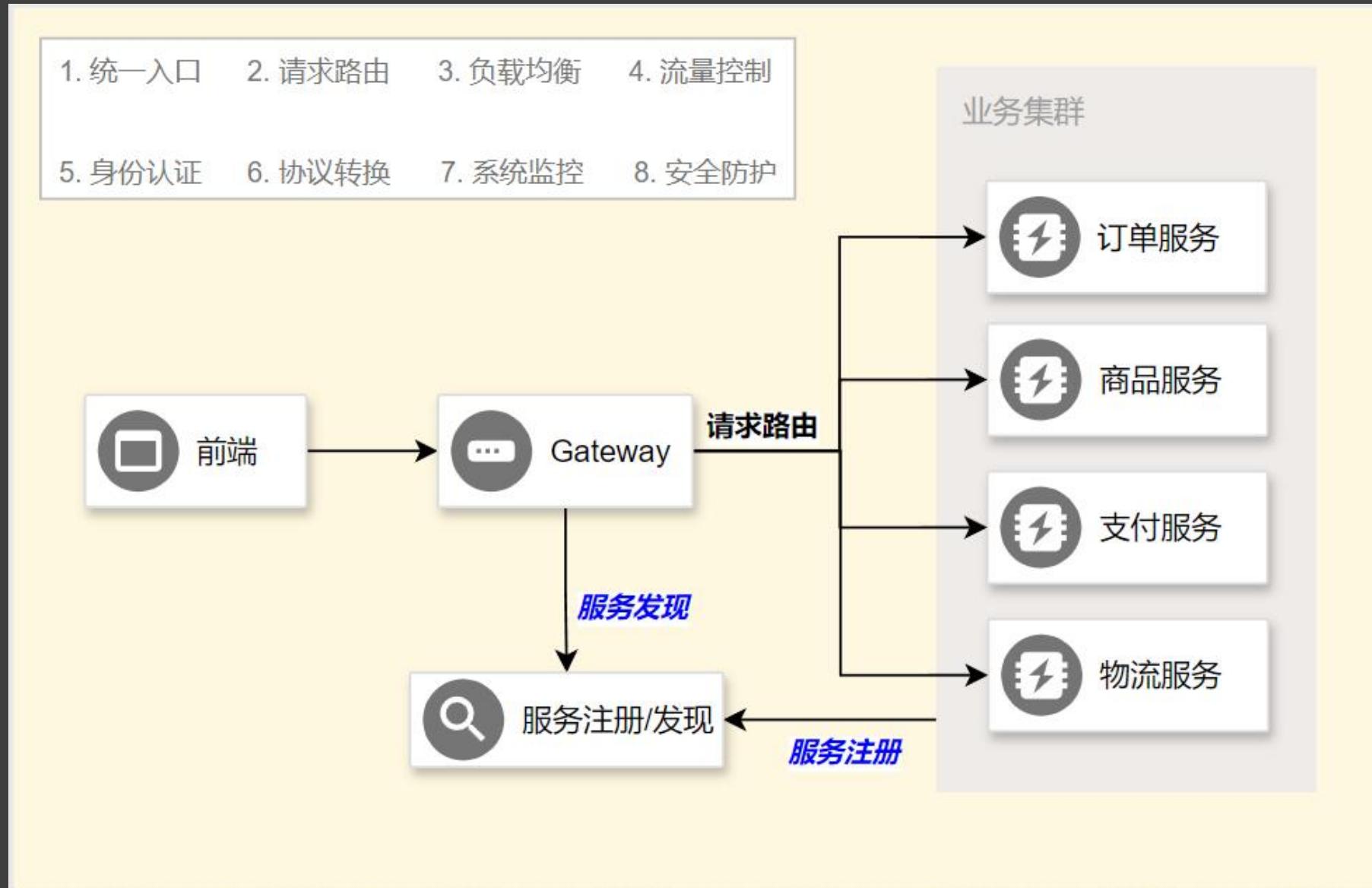
规则 - 热点参数

属性	说明	默认值
resource	资源名，必填	
count	限流阈值，必填	
grade	限流模式	QPS 模式
durationInSec	统计窗口时间长度（单位为秒），1.6.0 版本开始支持	1s
controlBehavior	流控效果（支持快速失败和匀速排队模式），1.6.0 版本开始支持	快速失败
maxQueueingTimeMs	最大排队等待时长（仅在匀速排队模式生效），1.6.0 版本开始支持	0ms
paramIdx	热点参数的索引，必填，对应 SphU.entry(xxx, args) 中的参数索引位置	
paramFlowItemList	参数例外项，可以针对指定的参数值单独设置限流阈值，不受前面 count 阈值的限制。仅支持基本类型和字符串类型	
clusterMode	是否是集群参数流控规则	false
clusterConfig	集群流控相关配置	

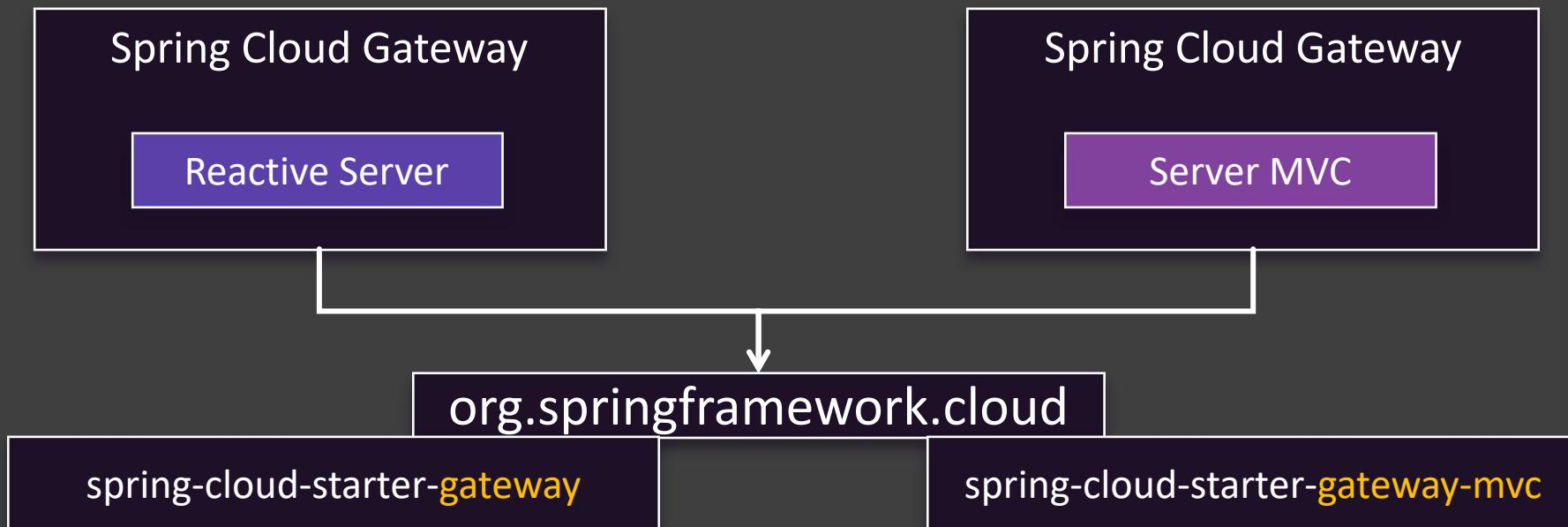
5. Gateway

网关

功能

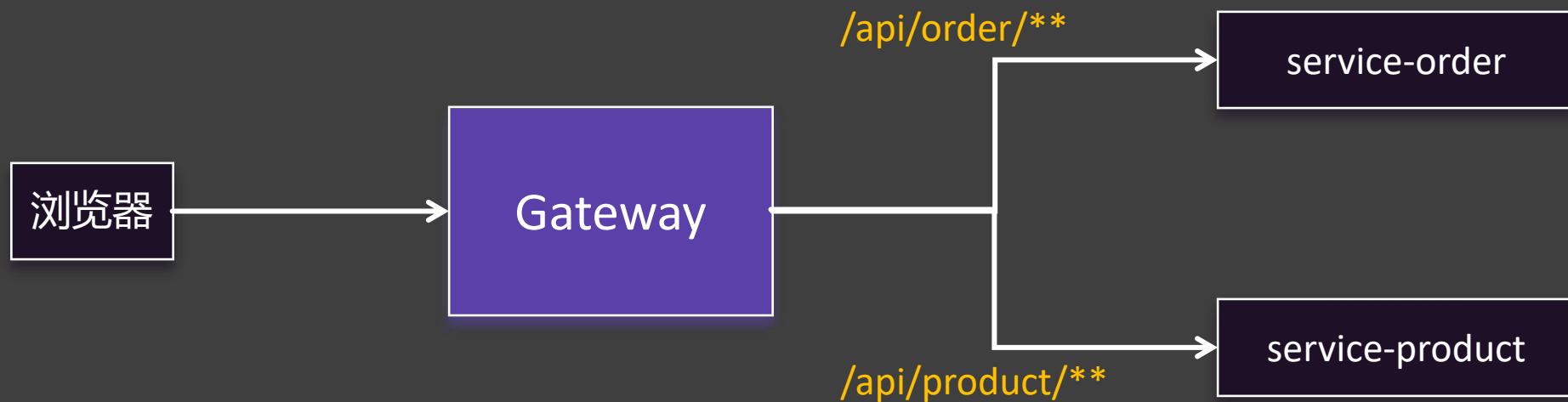


Spring Cloud Gateway

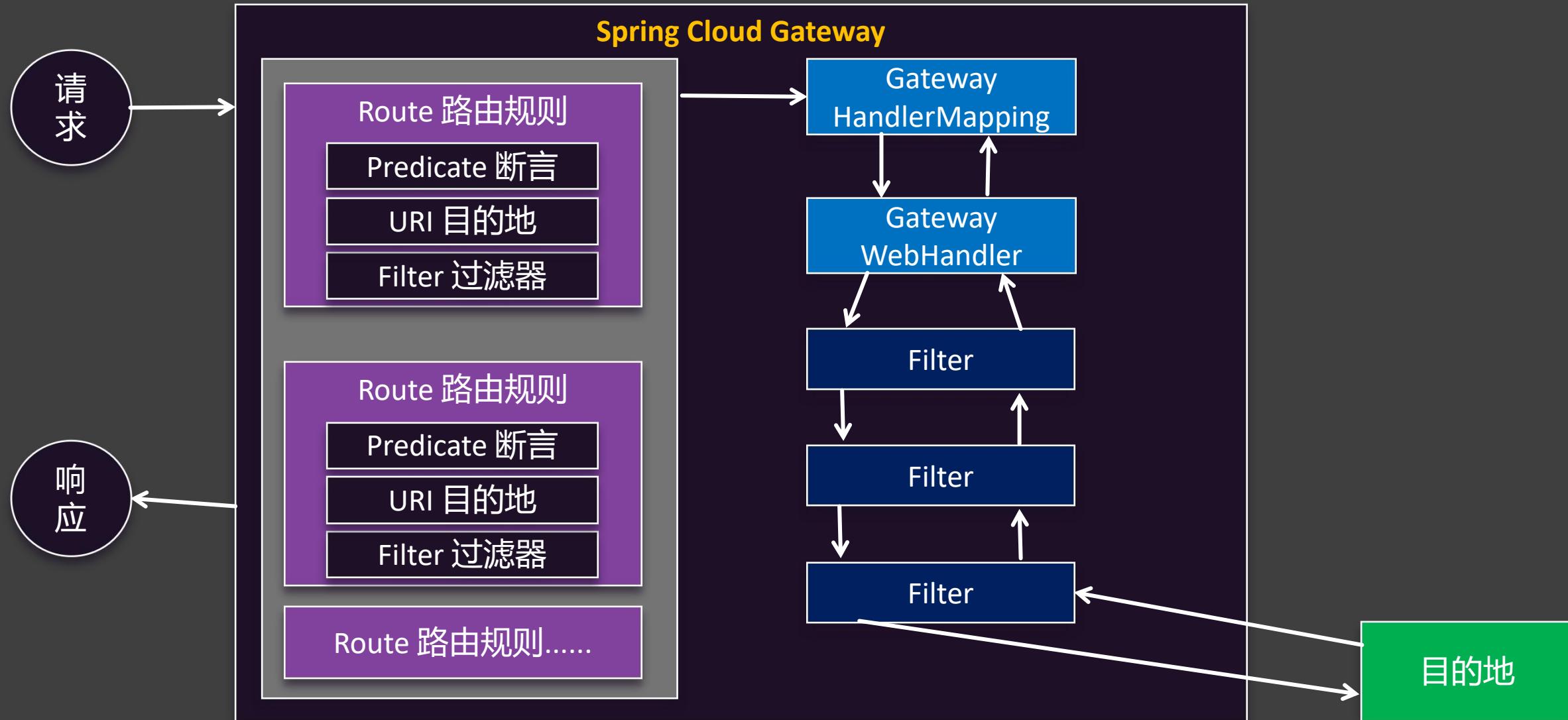


需求

- 1. 客户端发送 `/api/order/**` 转到 `service-order`
- 2. 客户端发送 `/api/product/**` 转到 `service-product`
- 3. 以上转发有负载均衡效果



基础原理



Predicate - 断言

```

spring:
  cloud:
    gateway:
      routes:
        - id: after_route
          uri: https://example.org
          predicates:
            - Cookie=mycookie,mycookievalue
  
```

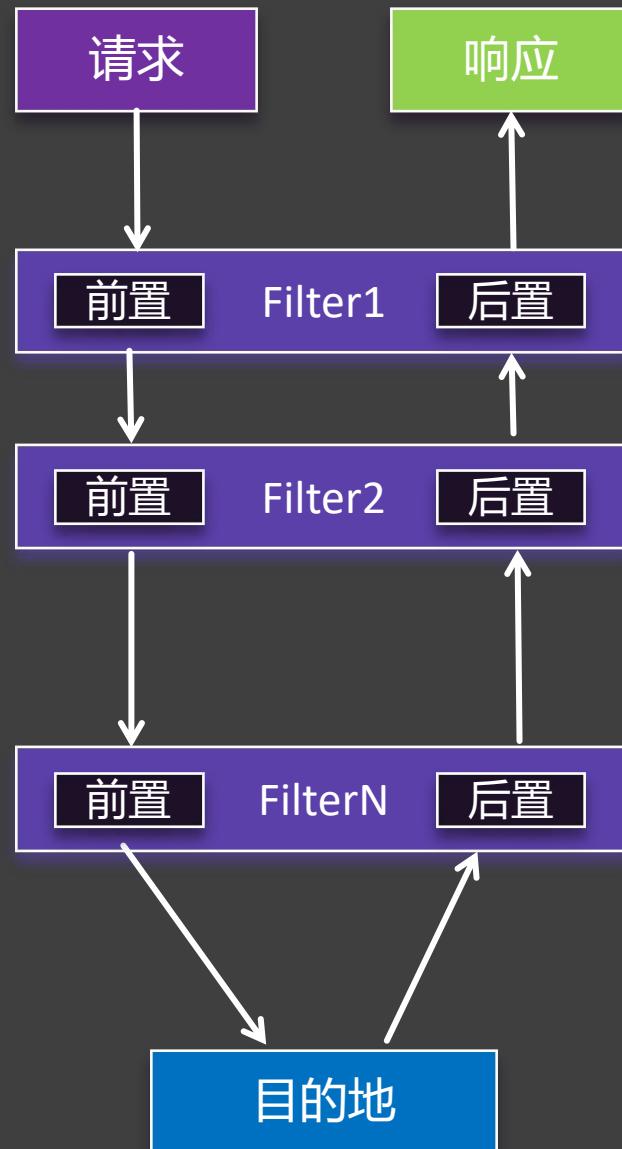
Route Predicate Factory Name
路由断言工厂

Params
参数

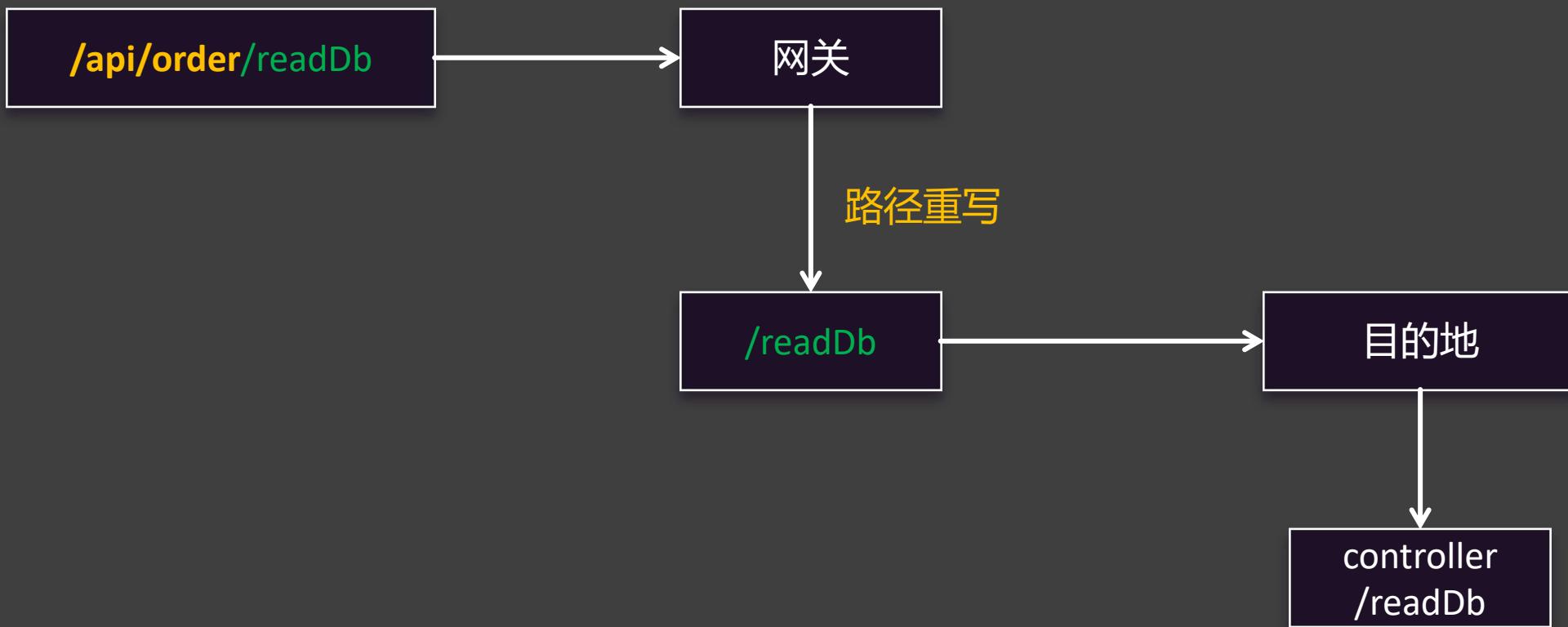
名	参数 (个数/类型)	作用
After	1/datetime	在指定时间之后
Before	1/datetime	在指定时间之前
Between	2/datetime	在指定时间区间内
Cookie	2/string,regexp	包含cookie名且必须匹配指定值
Header	2/string,regexp	包含请求头且必须匹配指定值
Host	N/string	请求host必须是指定枚举值
Method	N/string	请求方式必须是指定枚举值
Path	2/List<String>,bool	请求路径满足规则, 是否匹配最后的/
Query	2/string,regexp	包含指定请求参数
RemoteAddr	1>List<String>	请求来源于指定网络域(CIDR写法)
Weight	2/string,int	按指定权重负载均衡
XForwarded RemoteAddr	1>List<string>	从X-Forwarded-For请求头中解析请求来源, 并判断是否来源于指定网络域

Filter - 过滤器

```
spring:  
  cloud:  
    gateway:  
      routes:  
        - id: add_request_header_route  
          uri: https://example.org  
          filters:  
            - AddRequestHeader=X-Request-red, blue
```

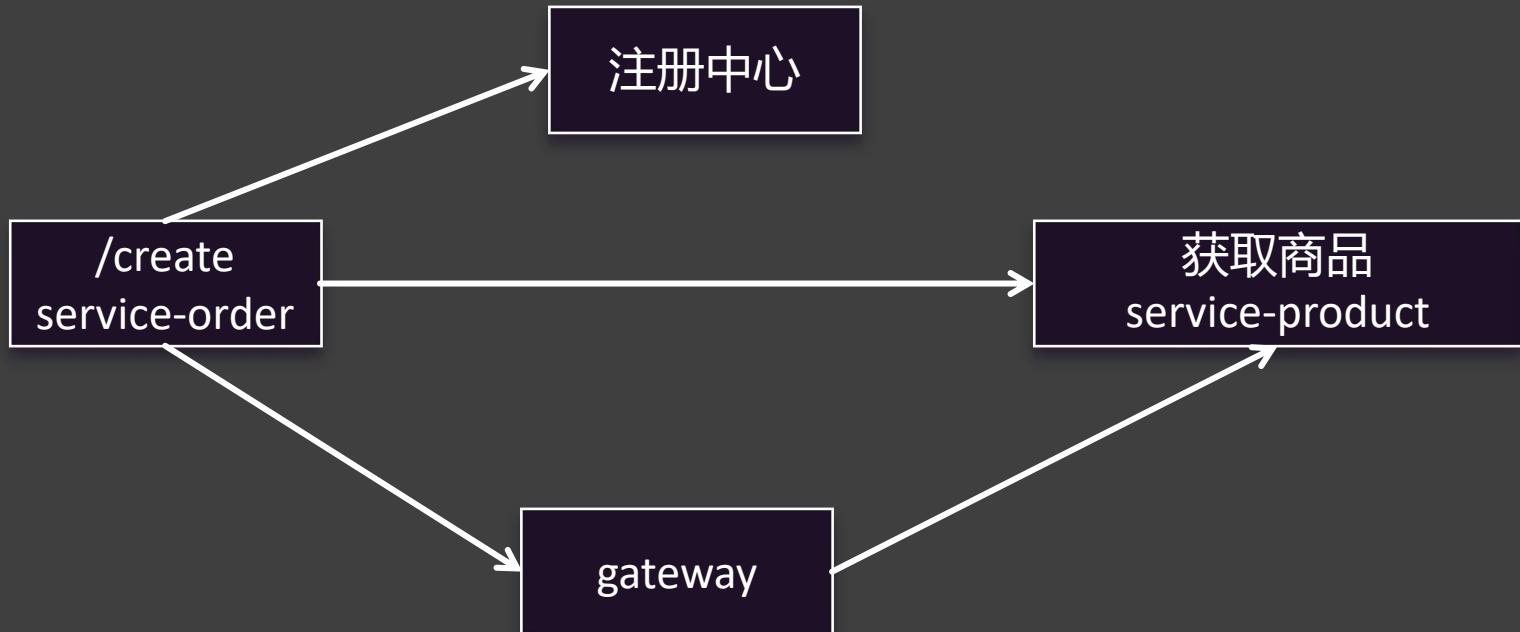


路径重写 - rewritePath



面试题

- 微服务之间的调用经过网关吗？

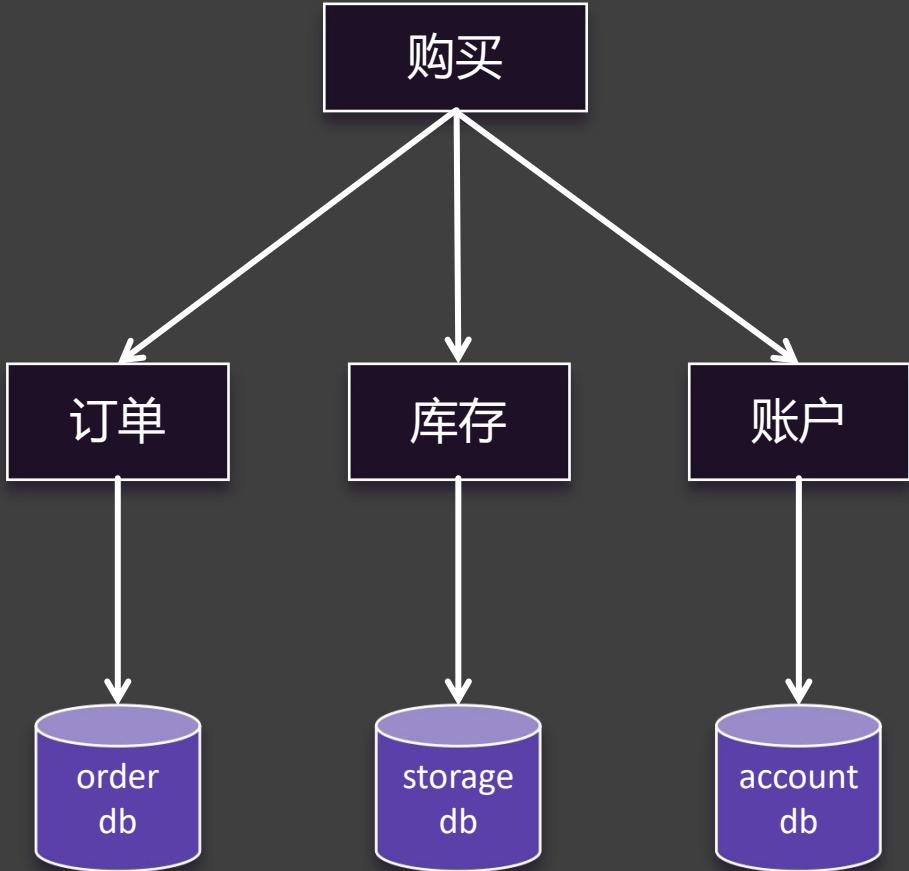


6. Seata

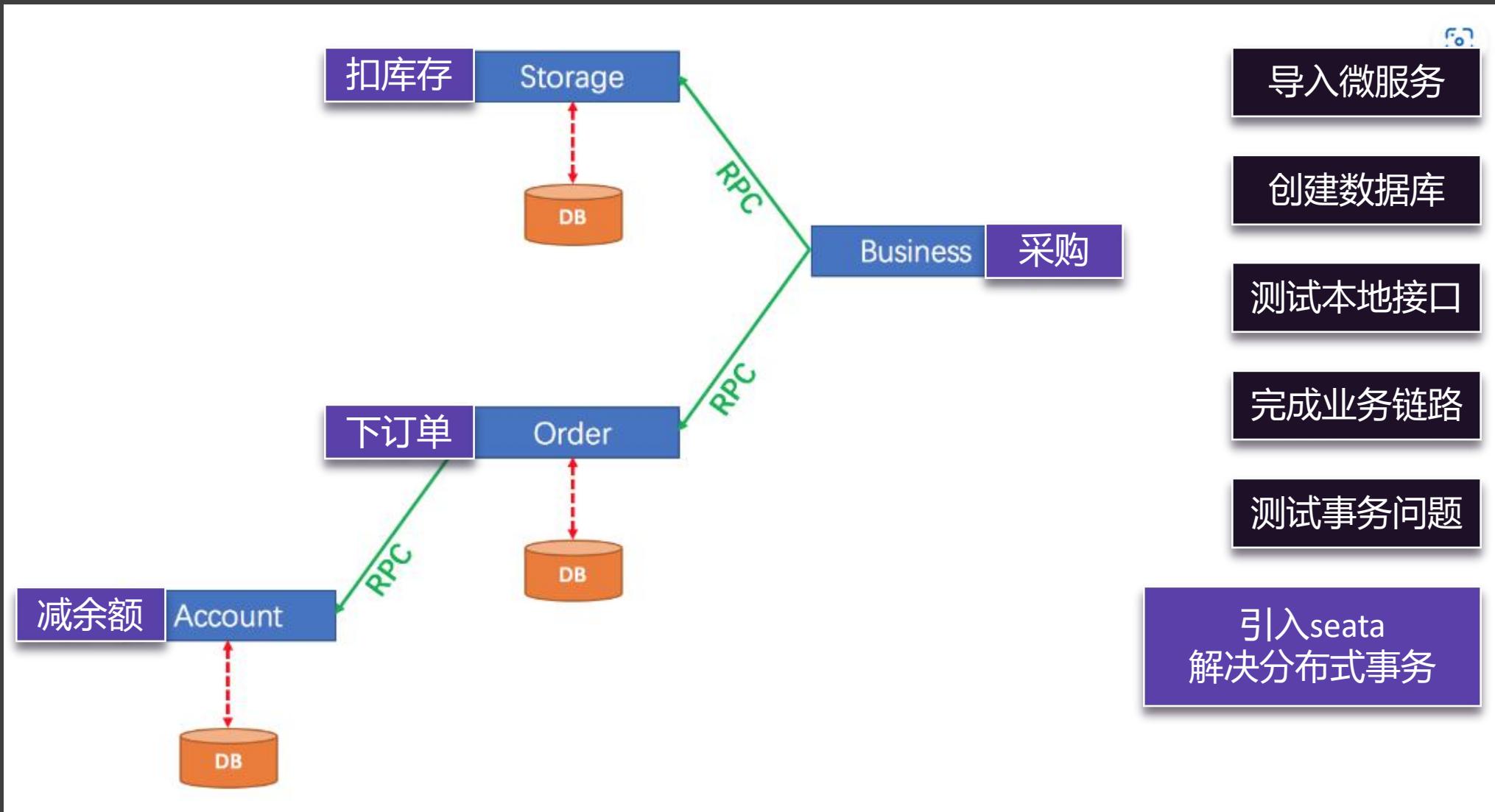
分布式事务

产生原因

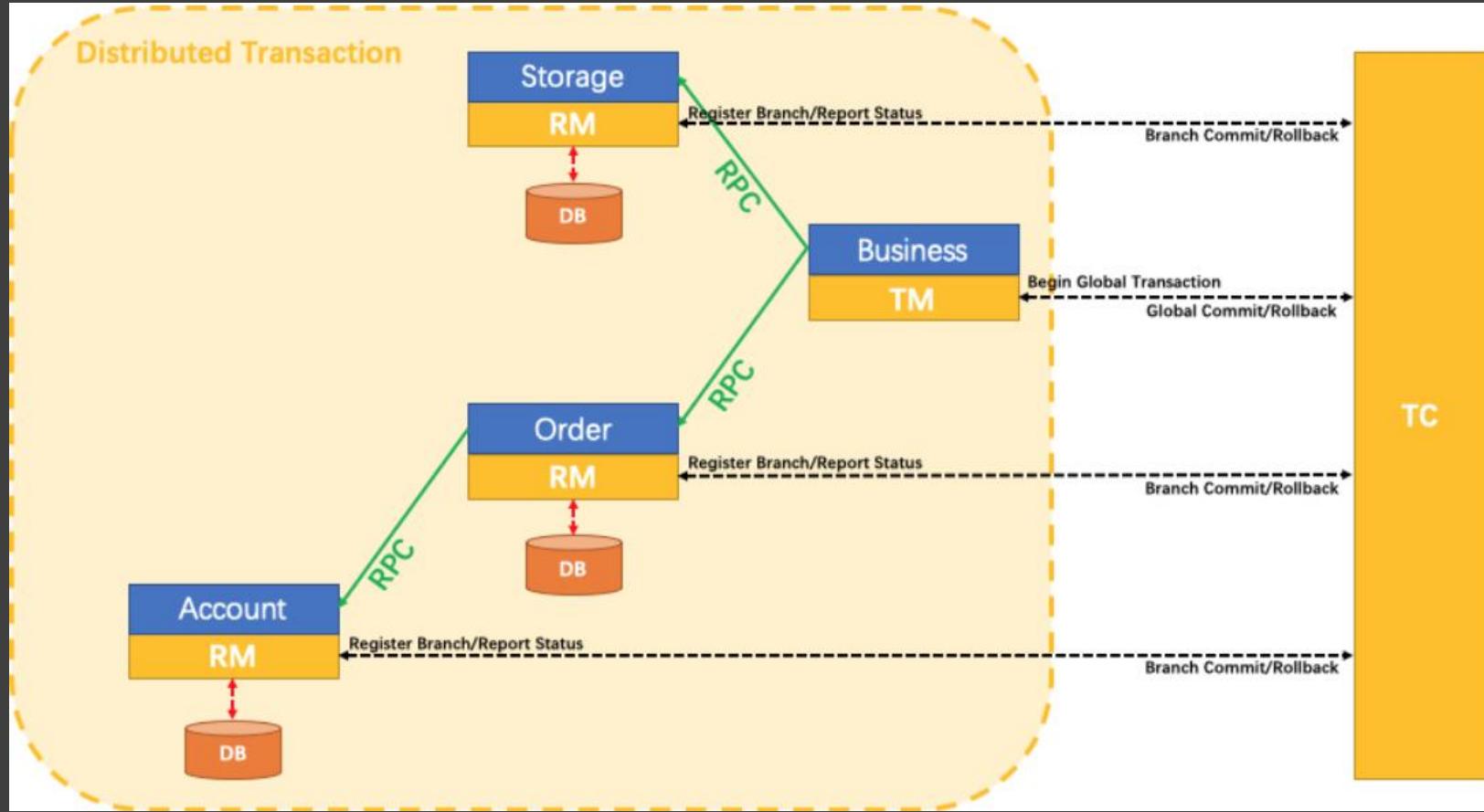
```
Connection connection = dataSource.getConnection();
try {
    connection.setAutoCommit(false);
    //TODO 业务逻辑
    connection.commit();
} catch (Exception e) {
    e.printStackTrace();
    try {
        connection.rollback();
    } catch (Exception e1) {
        e1.printStackTrace();
    }
}
```



环境准备



原理



TC (Transaction Coordinator) -
事务协调者
维护全局和分支事务的状态，驱动全局事务提交或回滚。

TM (Transaction Manager) -
事务管理器
定义全局事务的范围：开始全局事务、提交或回滚全局事务。

RM (Resource Manager) -
资源管理器
管理分支事务处理的资源，与TC交谈以注册分支事务和报告分支事务的状态，并驱动分支事务提交或回滚。

二阶提交协议

二阶提交

undo_log

一阶：本地事务提交
(业务数据 + undo_log)

全局锁 (数据级别)

二阶：成功 + 失败
成功：所有人删除undo_log
失败：所有人拿到自己的前镜像，恢复，删除undo_log

undo_log

```
"afterImage": {
    "rows": [
        "fields": [
            {
                "name": "id",
                "type": 4,
                "value": 1
            },
            {
                "name": "name",
                "type": 12,
                "value": "GTS"
            },
            {
                "name": "since",
                "type": 12,
                "value": "2014"
            }
        ]
    ],
    "tableName": "product"
},
```

```
"beforeImage": {
    "rows": [
        "fields": [
            {
                "name": "id",
                "type": 4,
                "value": 1
            },
            {
                "name": "name",
                "type": 12,
                "value": "TXC"
            },
            {
                "name": "since",
                "type": 12,
                "value": "2014"
            }
        ]
    ],
    "tableName": "product"
},
"sqlType": "UPDATE"
```

尚硅谷让天下没有难学的技术