博学谷
www.boxuegu.com

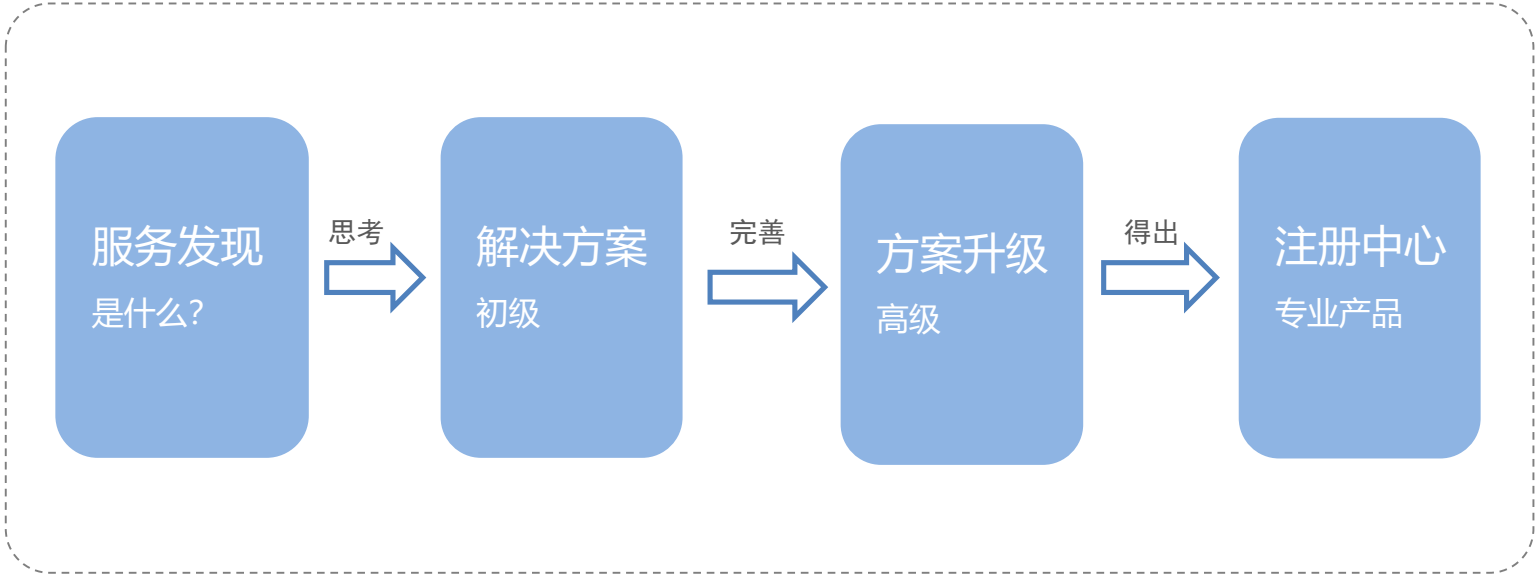
第二章 Spring Cloud Alibaba Nacos 服务发现

一样的在线教育，不一样的教学品质

目录 Contents

服务发现原理

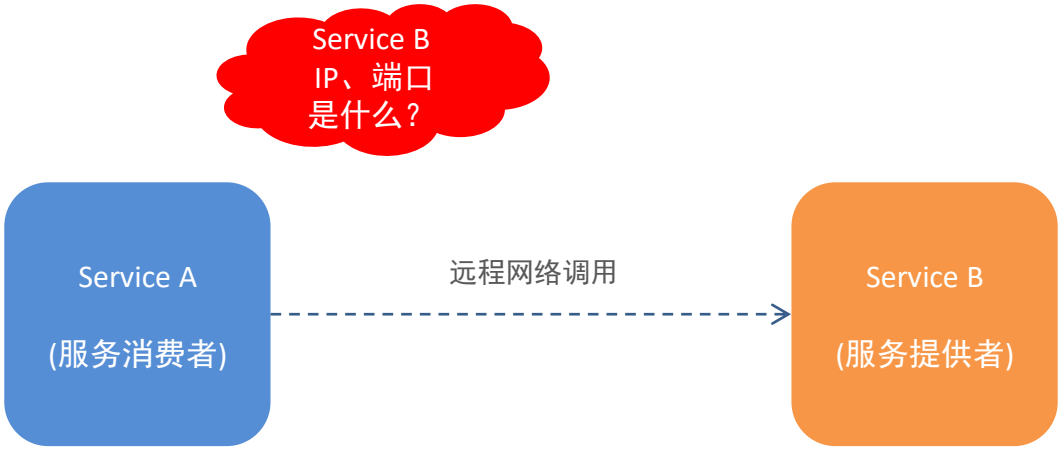
小节导学



服务发现原理

什么是服务发现?

服务消费者怎么找到服务提供者的机制就是服务发现。

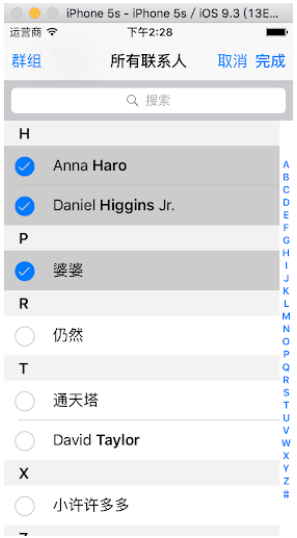


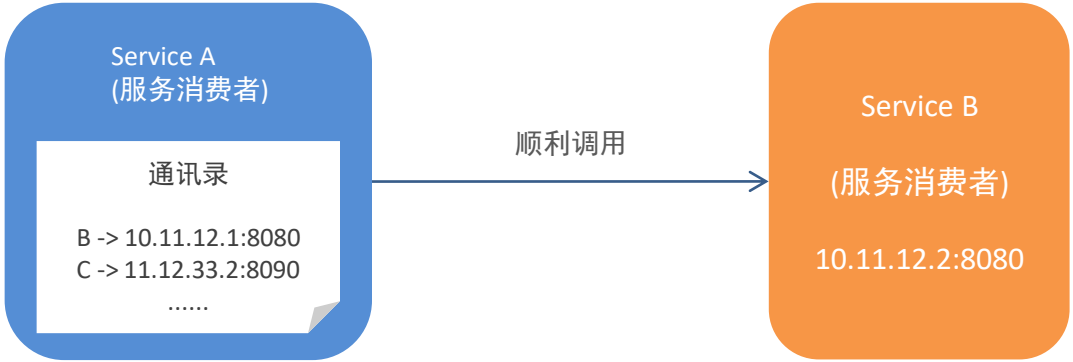
服务发现原理

如何实现服务发现?

最简单的方法就是 A 做一个**通讯录**。

记住 B 的信息，把B的通信地址记在自己的程序中，就可以调用了。

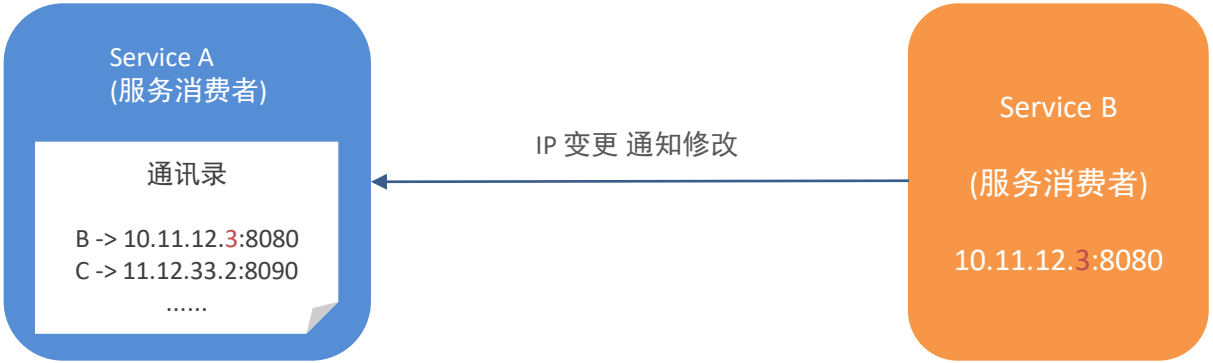




服务发现原理

缺点：

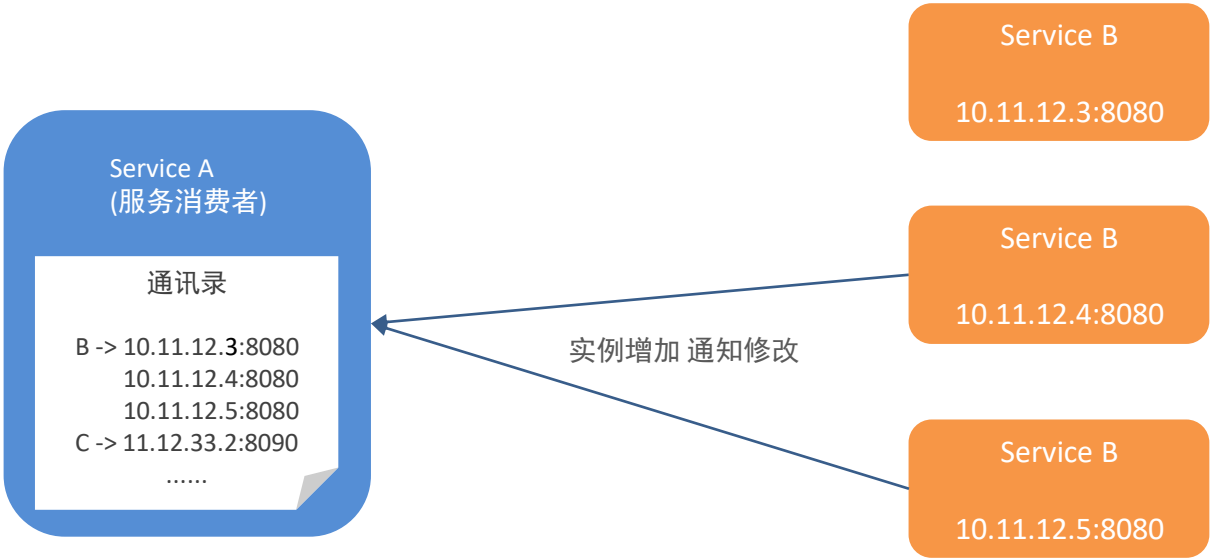
Service B 的地址一变，A 的通讯录就得改。



服务发现原理

缺点：

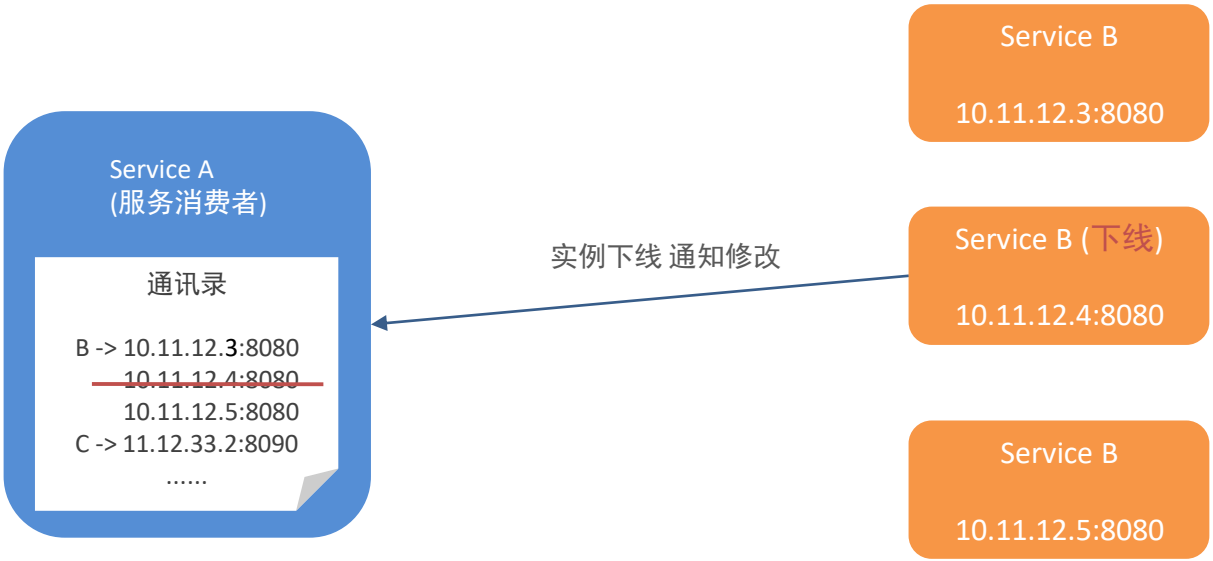
Service B 的实例增加，A 的通讯录也要添加。



服务发现原理

缺点：

Service B 的实例下线，A 的通讯录也要删除。



服务发现原理

缺点：

维护噩梦！

每个服务

维护

所有相关服务的地址

心好累 ……

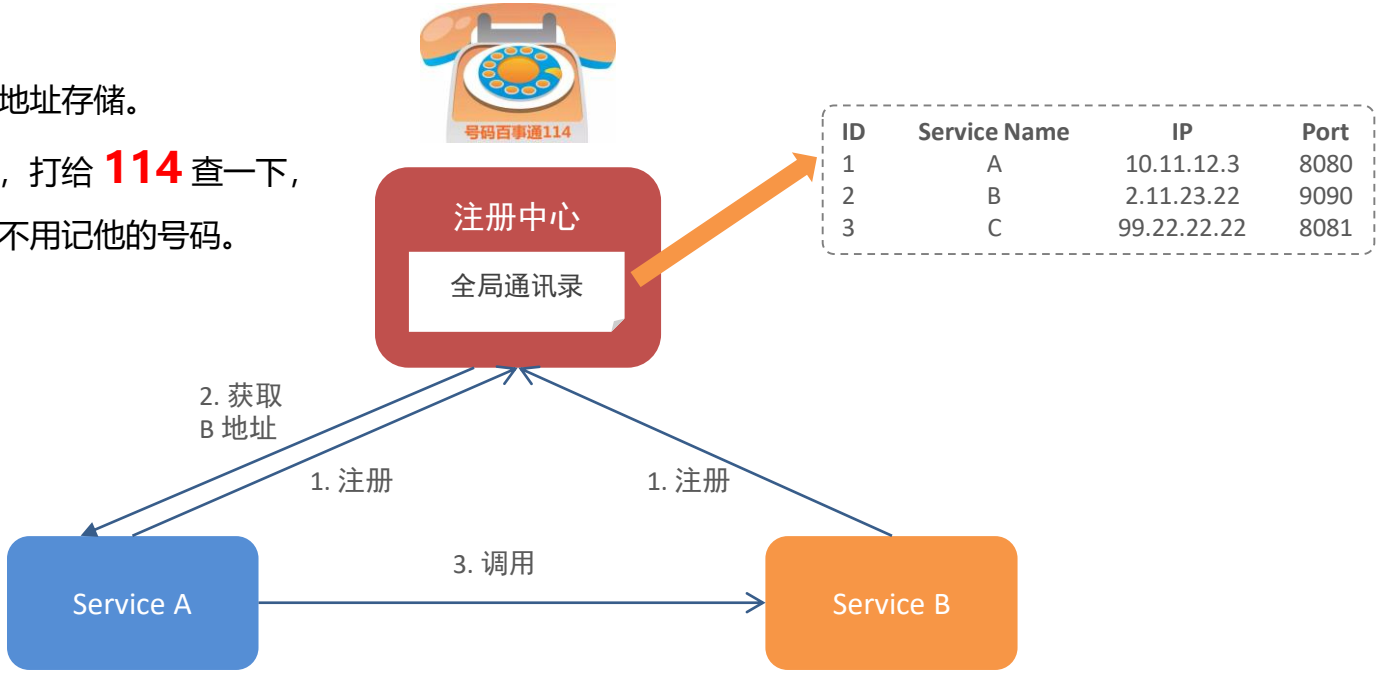
服务发现原理

解决方案：

使用第三方通信地址存储。

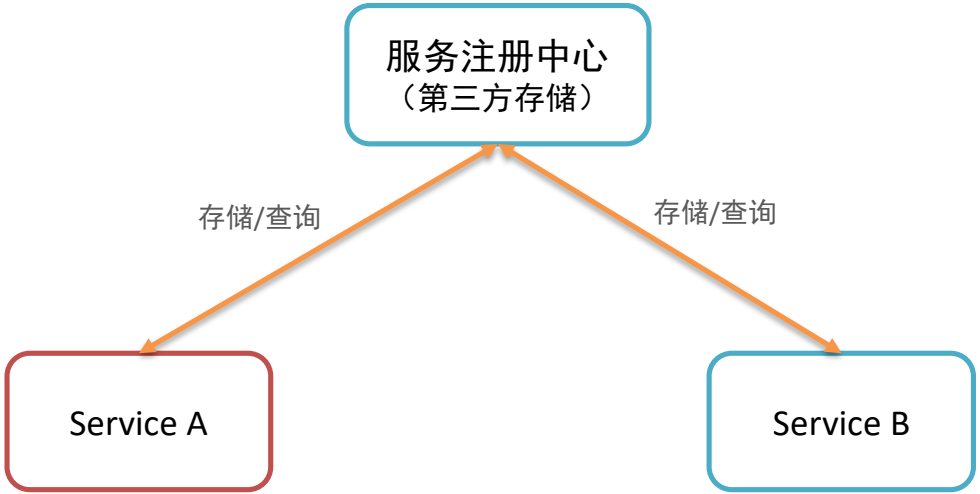
就像想要找谁时，打给 **114** 查一下，

知道名字就行，不用记他的号码。



服务发现原理

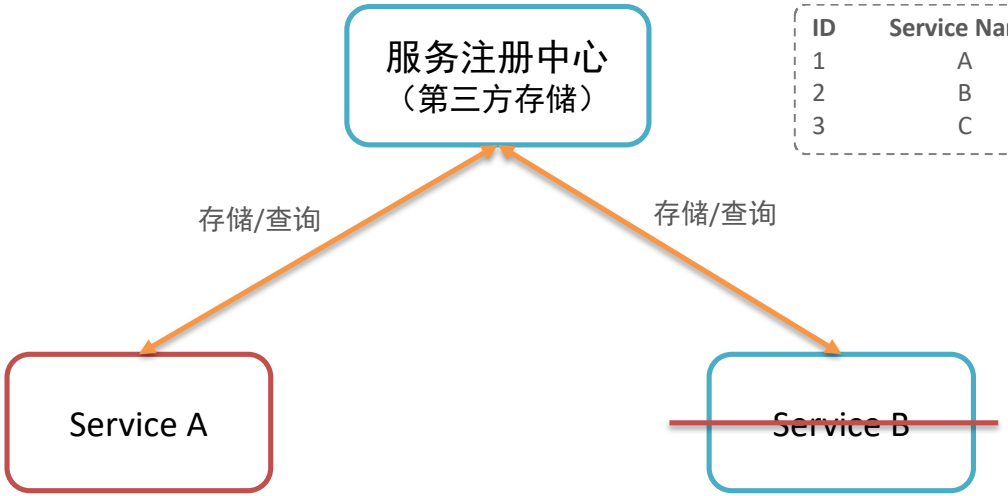
实现思路



服务发现原理

方案升级 – 健康检查

问题：服务实例故障了，无法主动注销自己的信息怎么办?

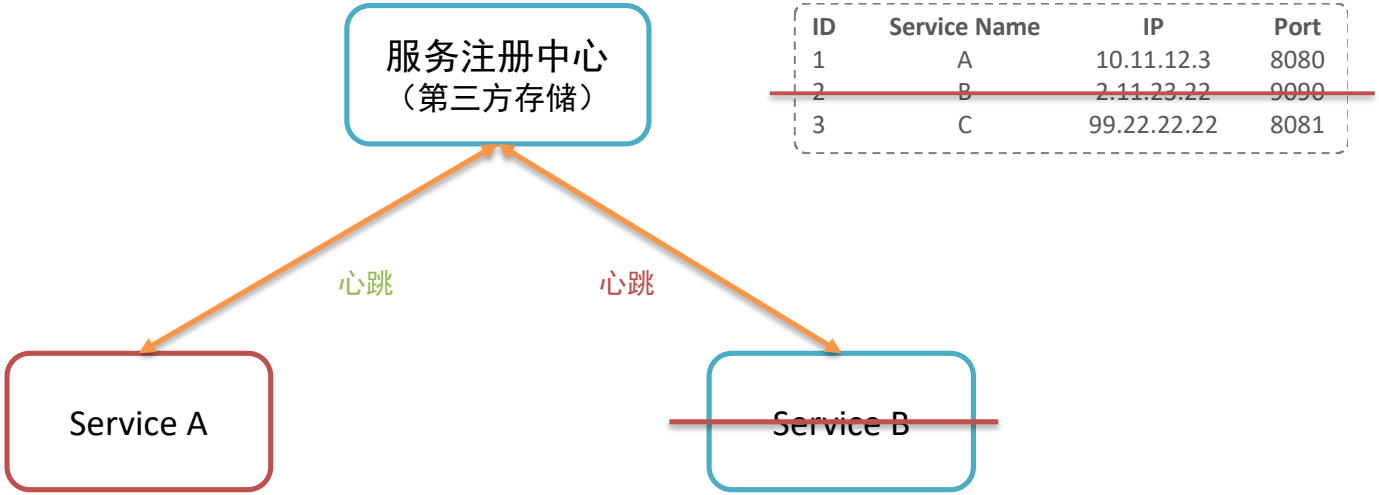


ID	Service Name	IP	Port
1	A	10.11.12.3	8080
2	B	2.11.23.22	9090
3	C	99.22.22.22	8081

服务发现原理

方案升级 – 健康检查

方案：通过心跳机制进行健康检查，注册中心删除无心跳的实例信息

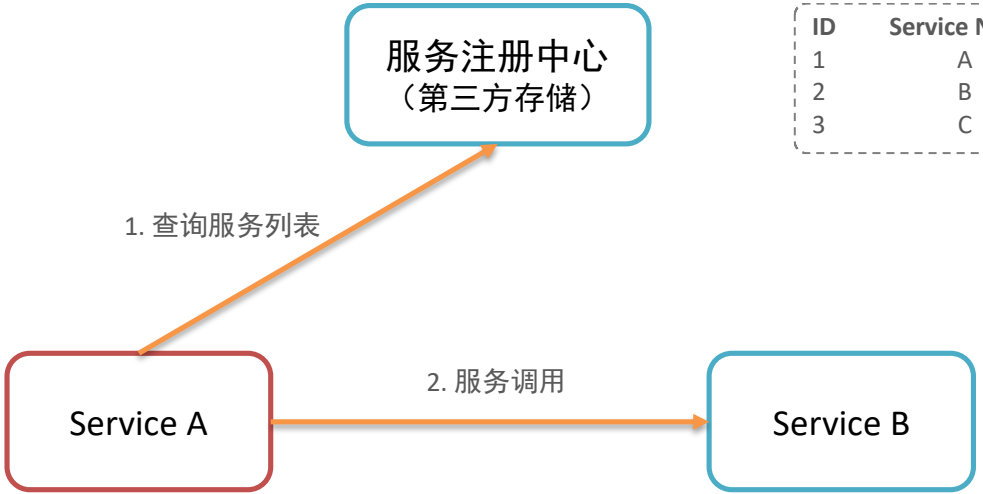


ID	Service Name	IP	Port
1	A	10.11.12.3	8080
~~2~~	~~B~~	~~2.11.23.22~~	~~9090~~
3	C	99.22.22.22	8081

服务发现原理

方案升级 – 本地缓存

问题：每次服务调用之前都查询注册中心，性能差、不可靠，怎么办？

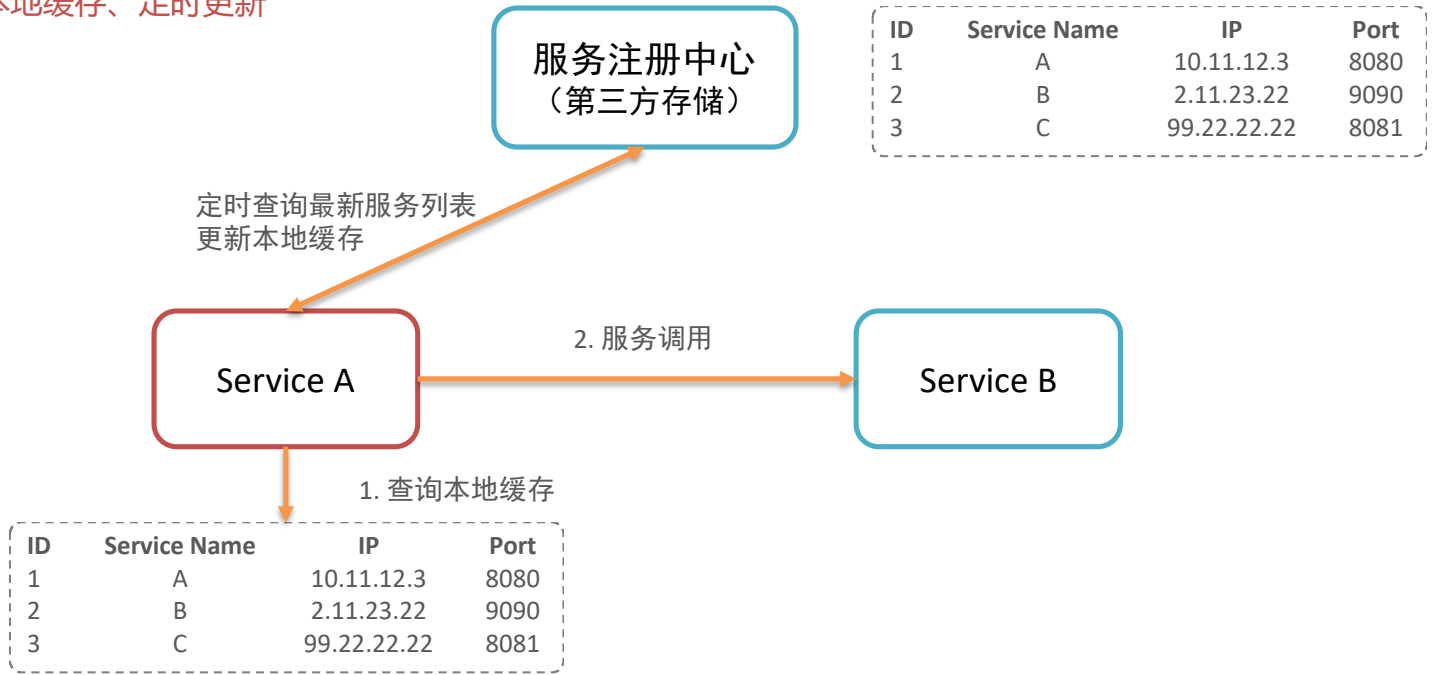


ID	Service Name	IP	Port
1	A	10.11.12.3	8080
2	B	2.11.23.22	9090
3	C	99.22.22.22	8081

服务发现原理

方案升级 – 本地缓存

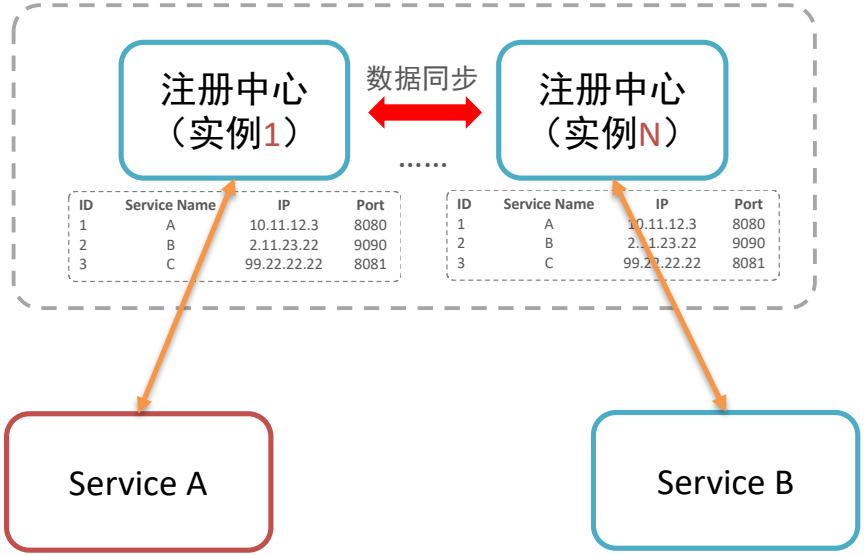
方案：本地缓存、定时更新



服务发现原理

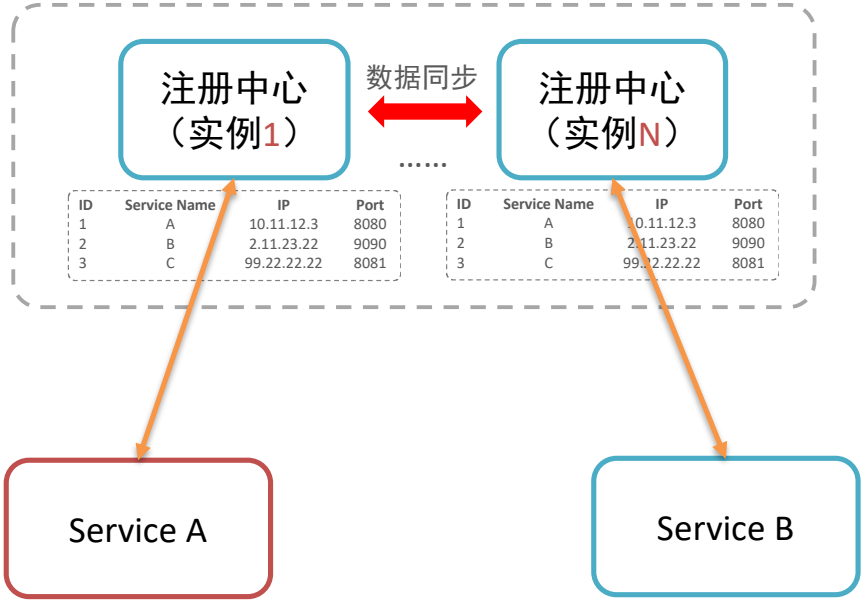
方案升级 – 数据同步

问题：注册中心集群中各个节点的数据如何同步？



服务发现原理

方案升级 – 数据同步



方案

- 强一致性，例如 ZooKeeper
- 弱一致性，例如 Eureka

服务发现原理

专业服务注册中心：

服务发现原理-总结

重难点

1. 服务发现

（一个服务如何找到另一个服务）

2. 实现服务发现机制需要解决很多的问题

（心跳健康检查、本地缓存、数据同步 ……）

服务发现原理-总结

重难点

1. 服务发现

（一个服务如何找到另一个服务）

2. 实现服务发现机制需要解决很多的问题

（心跳健康检查、本地缓存、数据同步 ……）

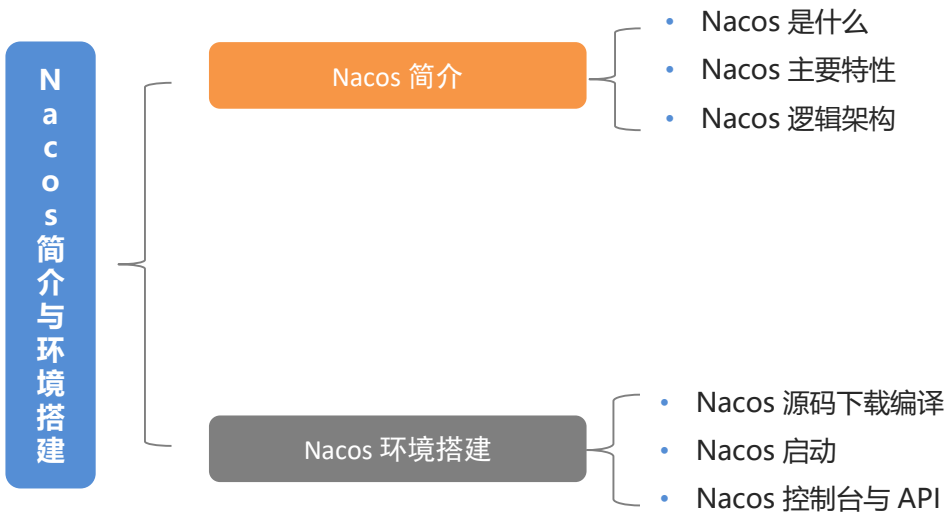
下节

把 Nacos 跑起来

目录 Contents

Nacos简介与环境搭建

小节导学



Nacos简介与环境搭建

Nacos 简介

官方网站地址 https://nacos.io/zh-cn/

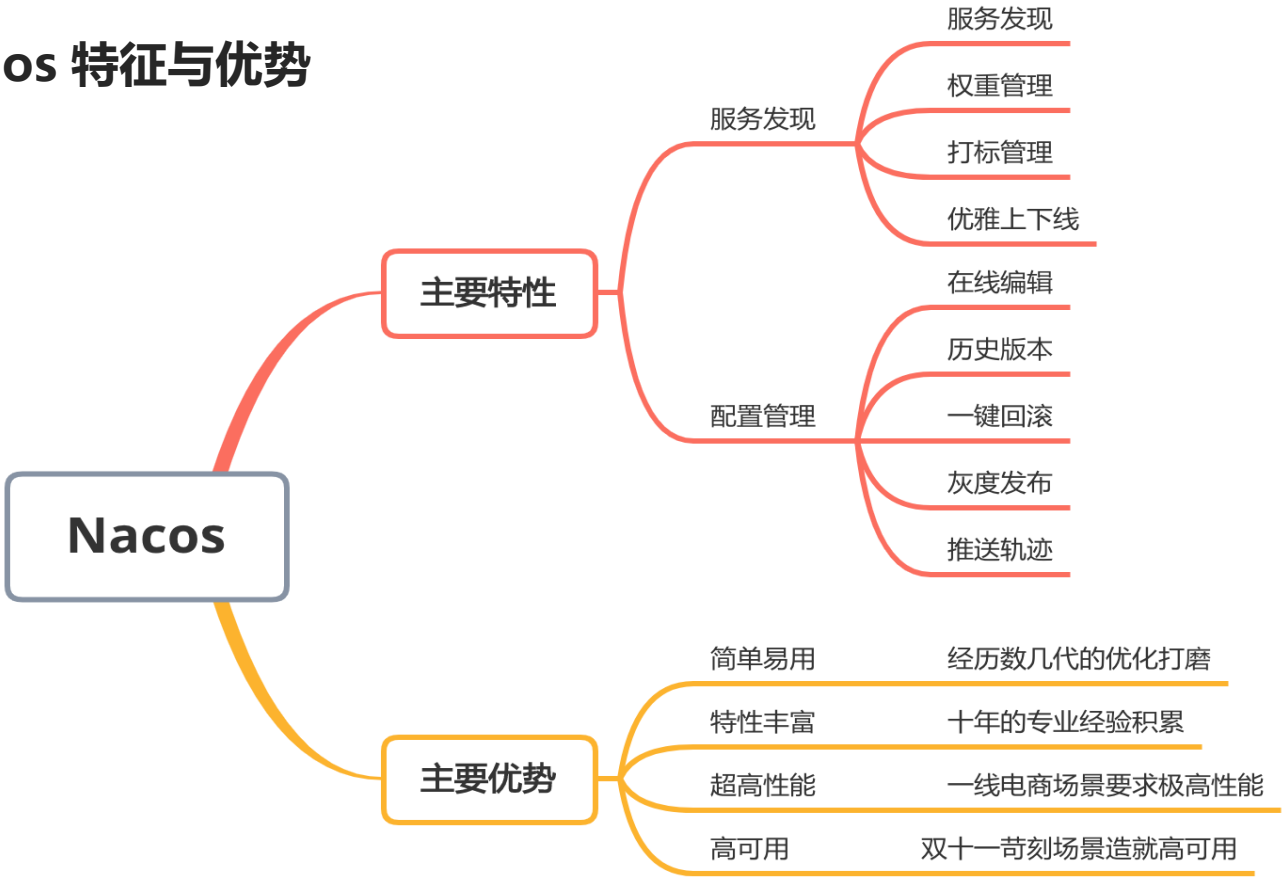
Nacos简介与环境搭建

Nacos 简介

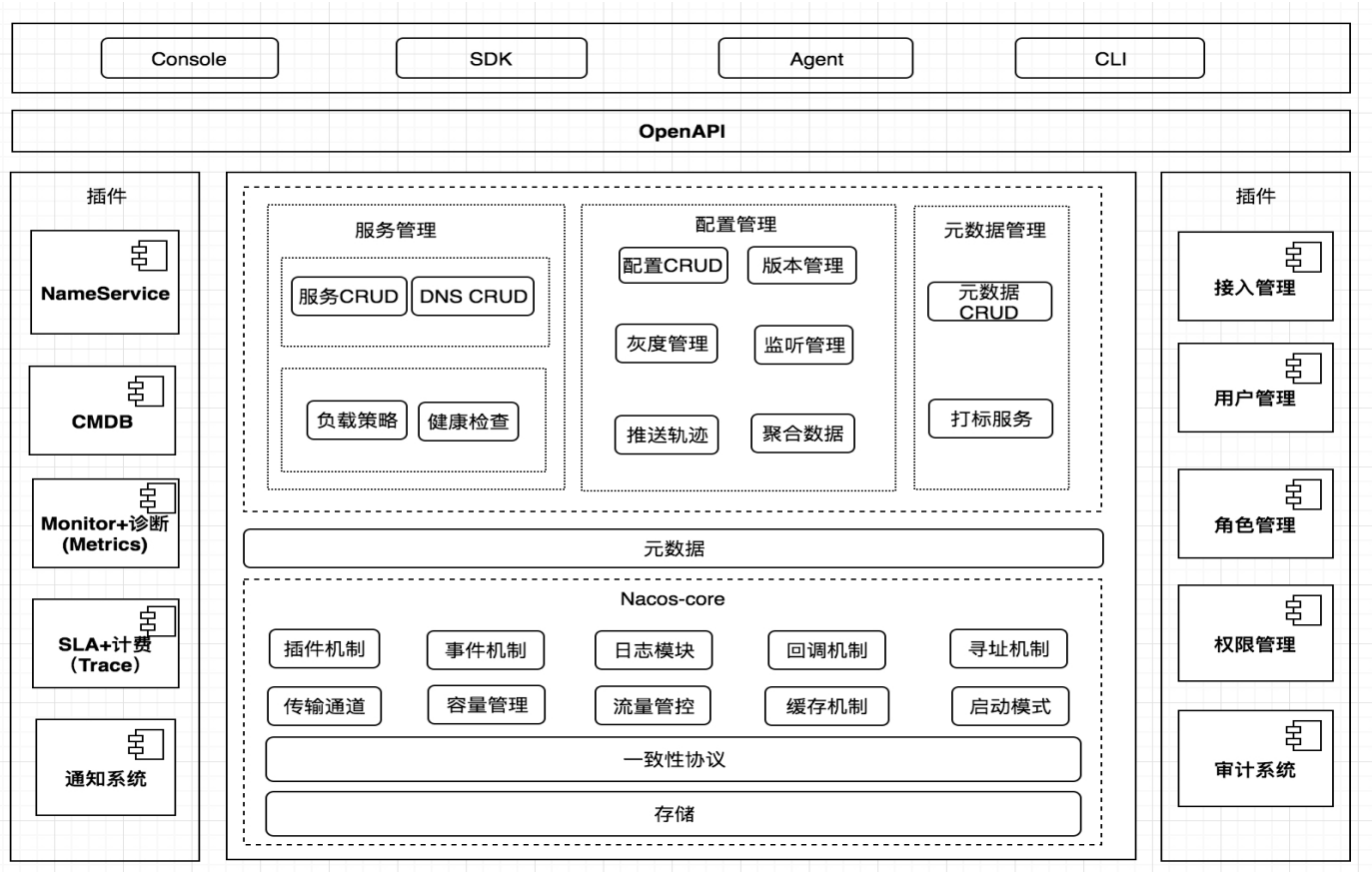
- 发音： /nɑ:kəʊs/
- 全称：Name and Config Service，nacos 是其首字母的拼写。
- Nacos 的核心功能 = 服务注册 + 动态配置
- 可以理解为 Nacos = SpringCloud Eureka + SpringCloud Config

Nacos简介与环境搭建

Nacos 特征与优势



Nacos简介与环境搭建



Nacos简介与环境搭建

步骤

1. 下载并编译源码

```
$ git clone
https://github.com/alibaba/nacos.git
$ cd nacos/
$ mvn -Prelease-nacos -Dmaven.test.skip=true
clean install -U
```

Nacos 也提供了编译好的压缩包：

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

2. 启动

Linux/Unix/Mac 下执行启动命令：

```
$ cd distribution/target/nacos-server-
$version/nacos/bin
$ sh startup.sh -m standalone
```

Windows 下双击 startup.cmd 运行文件，或者执行启动命令：

```
cmd startup.cmd
```

Nacos简介与环境搭建

Nacos 代码编译

- 小技巧 -- 下载源码加速

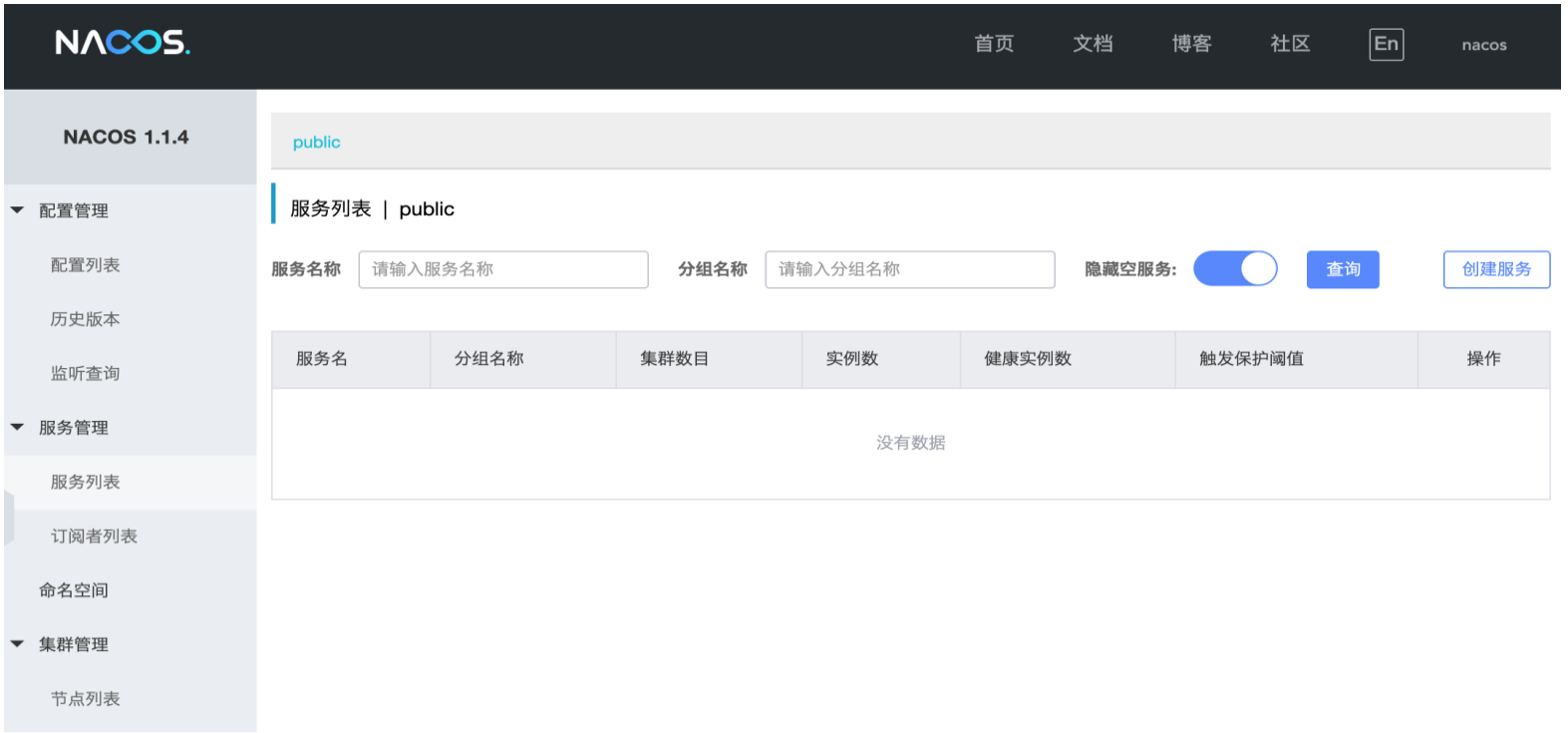
如果要下载的是主流项目，github 下载比较慢时，可以去咱们国内的代码仓库“码云”上看看，一般都有镜像。

网址 https://gitee.com/

此仓库是为了提升国内下载速度的镜像仓库，每日同步一次。 原始仓库： https://github.com/alibaba/nacos/

Nacos简介与环境搭建

Nacos 控制台



Nacos简介与环境搭建

Nacos API

发布服务：

```
$ curl -X POST \
'http://127.0.0.1:8848/nacos/v1/ns/instance?serviceName=service-user&ip=20.18.7.10&port=8080'
```

获取服务列表：

```
$ curl -X GET \
'http://127.0.0.1:8848/nacos/v1/ns/instance/list?serviceName=service-user'
```

Nacos简介与环境搭建-总结

重难点

1. Nacos = 服务管理 + 配置管理
2. Nacos 的编译启动方法（加速代码下载的小技巧）
3. Nacos API 操作方法

Nacos简介与环境搭建-总结

重难点

1. Nacos = 服务管理 + 配置管理
2. Nacos 的编译启动方法（加速代码下载的小技巧）
3. Nacos API 操作方法

下节

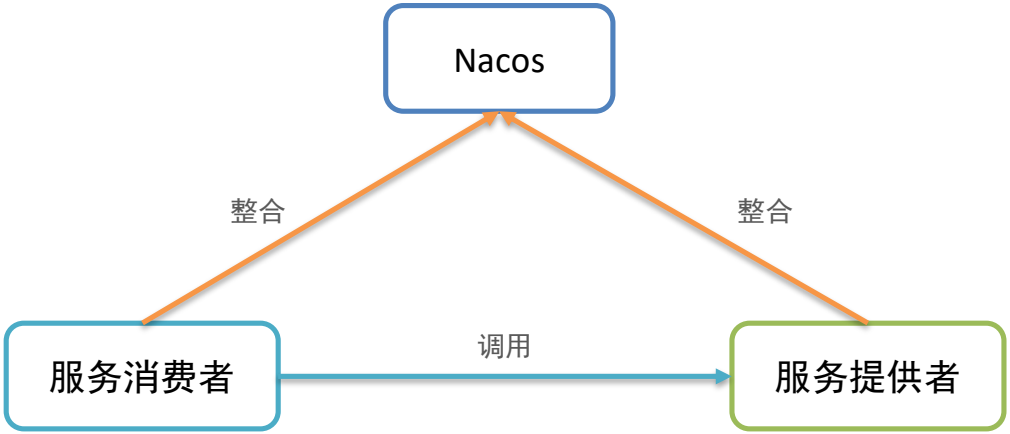
写代码，把服务**整合到 Nacos**

目录 Contents

服务提供者与消费者整合 Nacos

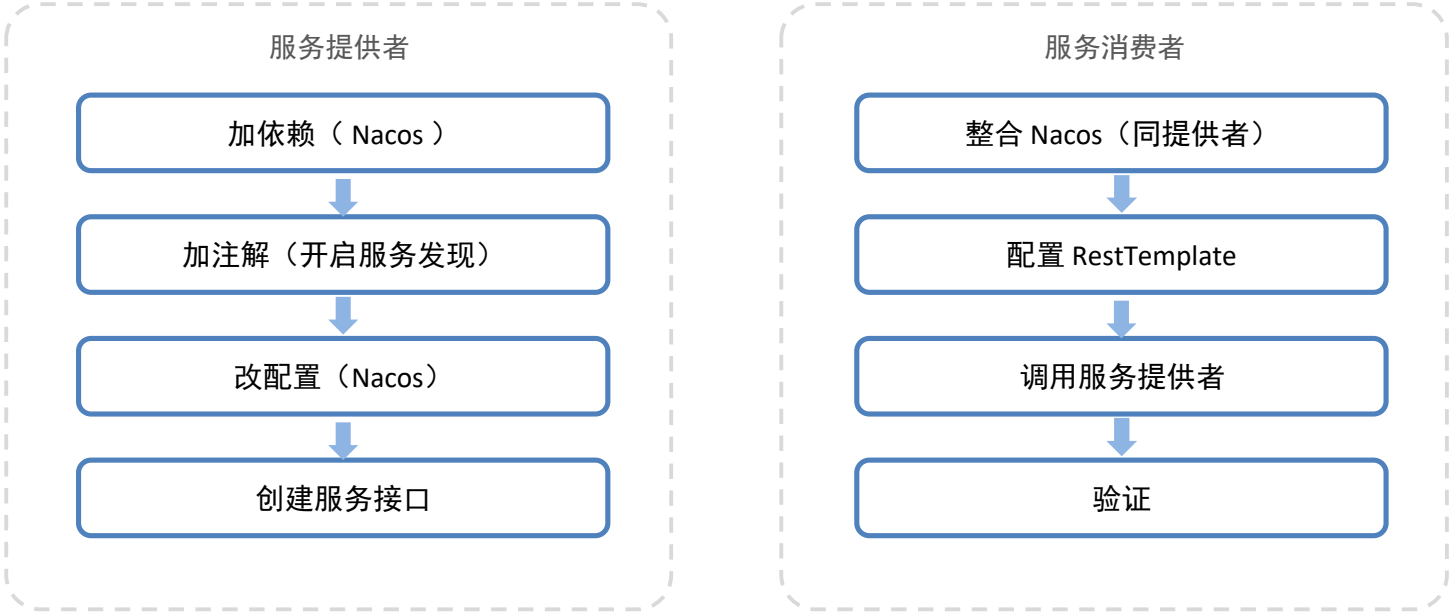
小节导学

通过代码实践来学习 Nacos 的基本用法。



服务提供者与消费者整合 Nacos

实践流程



服务提供者与消费者整合 Nacos

核心代码

添加依赖：

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

服务提供者与消费者整合 Nacos

核心代码

修改配置：

```
server:
  port: 8081
spring:
  application:
    name: service-provider
  cloud:
    nacos:
      discovery:
        server-addr: localhost:8848 # nacos 地址
```

开启服务发现：

```
@SpringBootApplication
@EnableDiscoveryClient
public class ServiceProviderApplication {
   ...
}
```

服务提供者与消费者整合 Nacos

核心代码

通过 Nacos 获取服务提供者实例，调用：

```
@Autowired
private LoadBalancerClient loadBalancerClient;
@Autowired
RestTemplate restTemplate;

@GetMapping("/test")
public String test(String name){
    // 获取服务实例，使用负载均衡客户端根据服务名选择实例
    ServiceInstance serviceInstance = loadBalancerClient.choose("service-provider");
    URI instanceUri = serviceInstance.getUri();
    // 发起调用
    return restTemplate.getForObject(instanceUri + "/demo?name=" + name, String.class);
}
```

服务提供者与消费者整合 Nacos-总结

重难点

1. 服务整合 Nacos 的流程

下节

Nacos 设计模型

目录 Contents

Nacos 设计模型

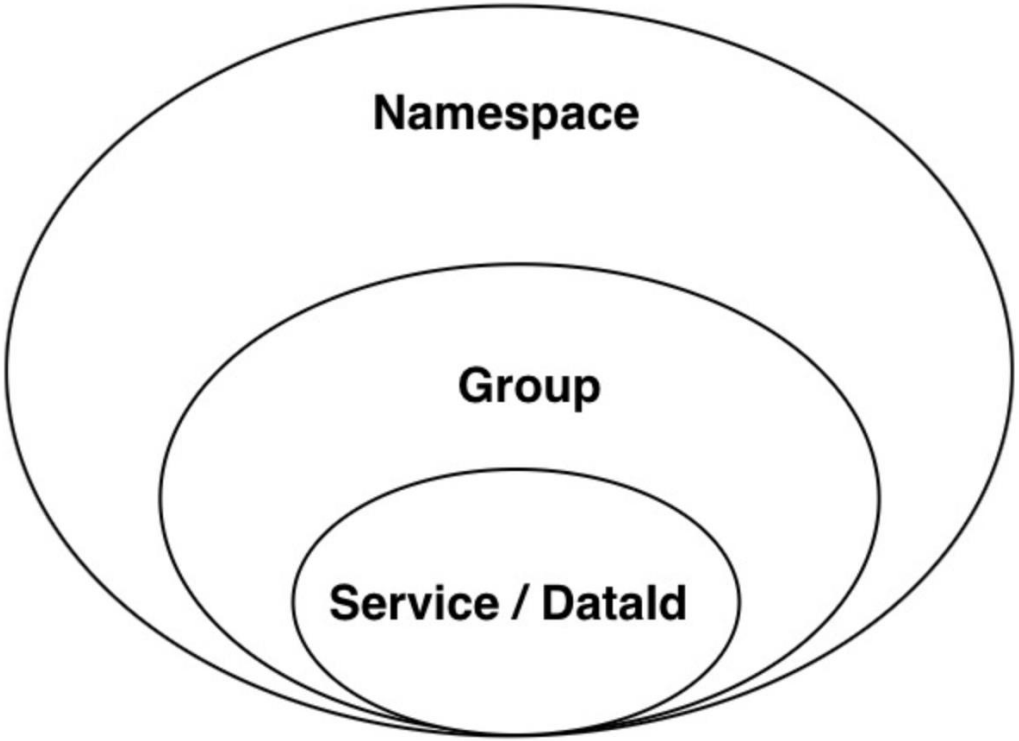
小节导学

阿里在多年的实践中总结出了数据模型、领域模型，对我们的实际应用提供了宝贵的经验和便利。本节我们就学习一下相关的重要概念，并通过实践操作来加深理解。

- Nacos 数据模型
- Nacos 领域模型
- 代码实践领域模型的应用

Nacos 设计模型

Nacos 3层数据模型



Nacos 设计模型

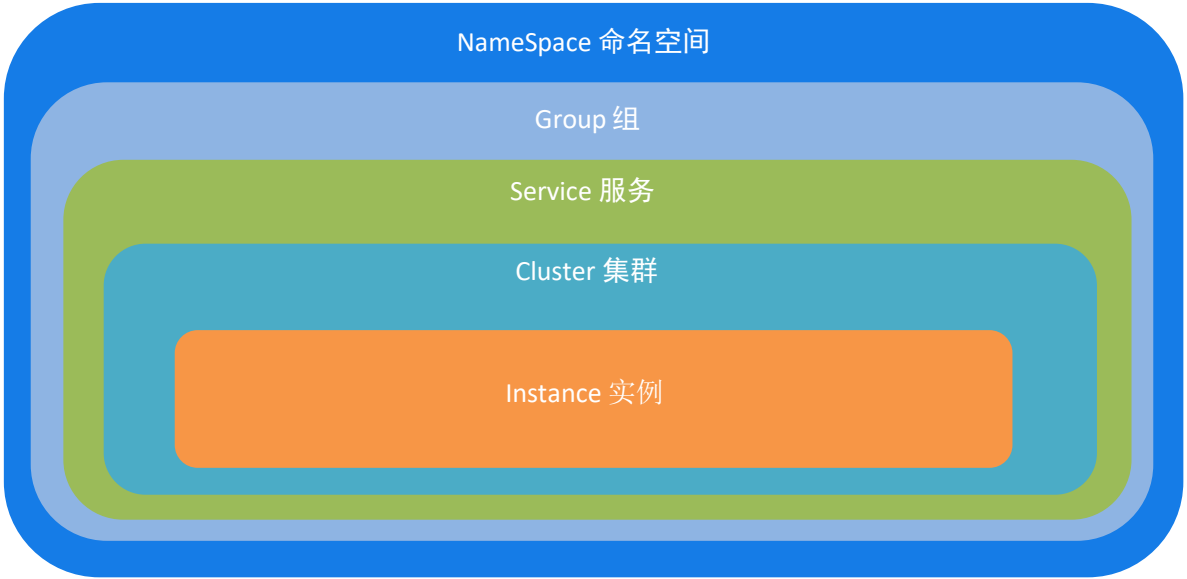
Nacos 3层数据模型

如果直接就是服务，会很混乱

服务名	说明
Service 01	产品1
Service 02	产品2
Service 03	子系统1
Service 04	子系统2
Service 05	开发环境
Service 06	测试环境

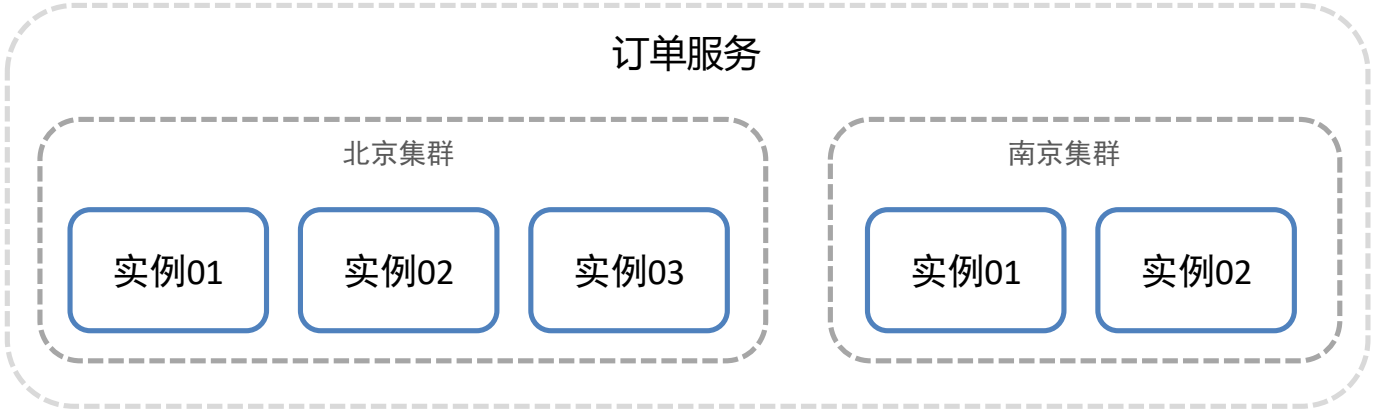
Nacos 设计模型

Nacos 服务领域模型



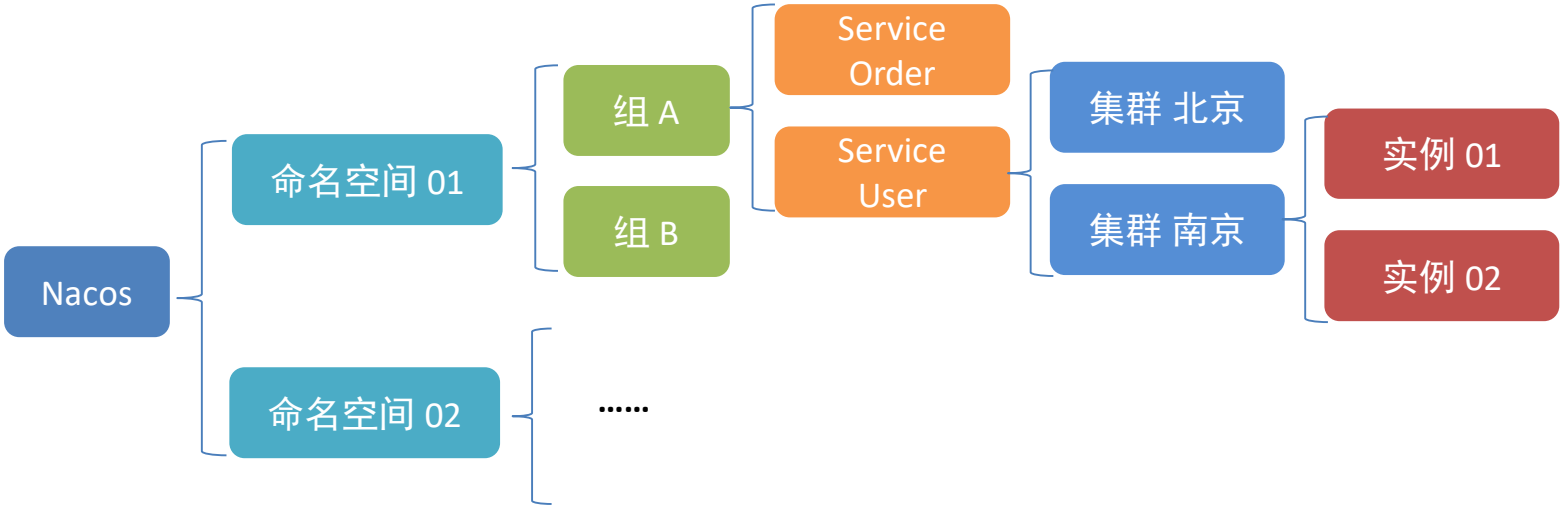
Nacos 设计模型

Nacos 服务领域模型



Nacos 设计模型

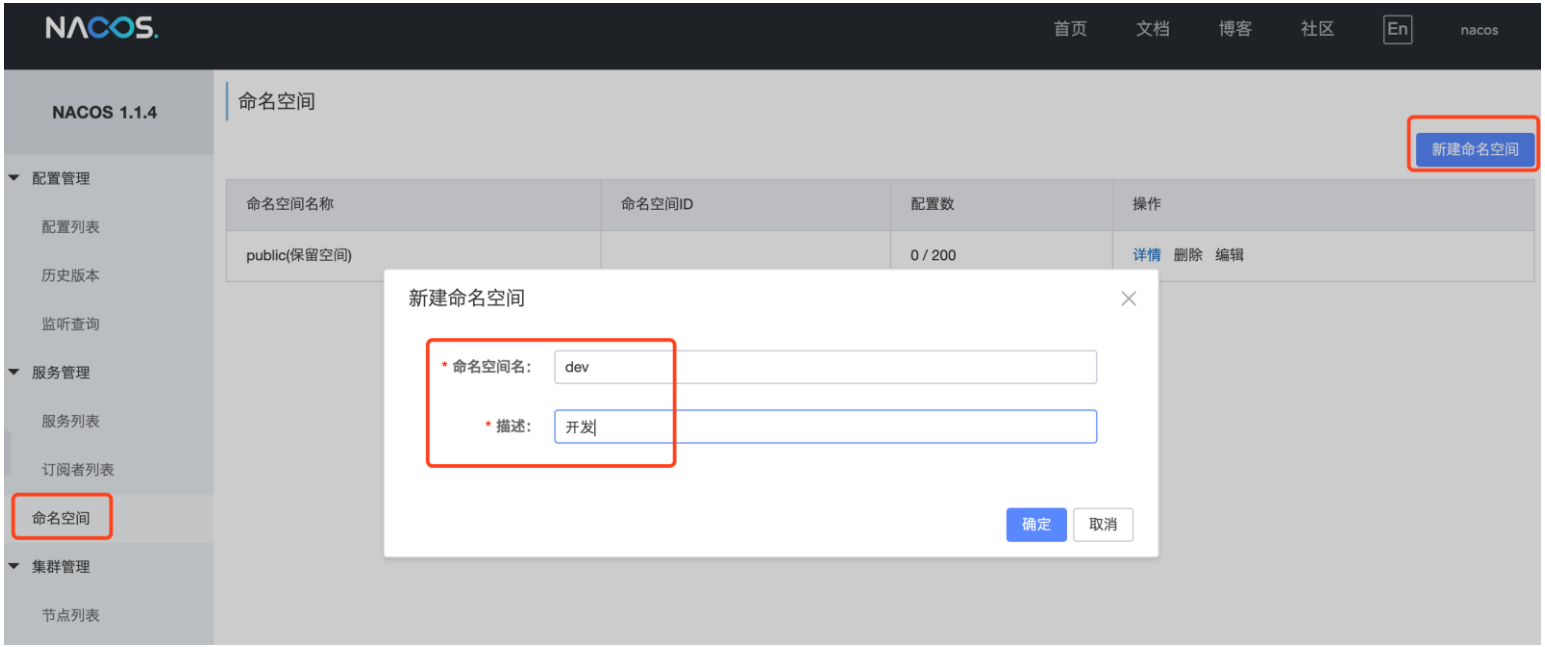
Nacos 服务领域模型



Nacos 设计模型

Nacos 服务领域模型实践

Nacos 控制台中新建一个命名空间“dev”



Nacos 设计模型

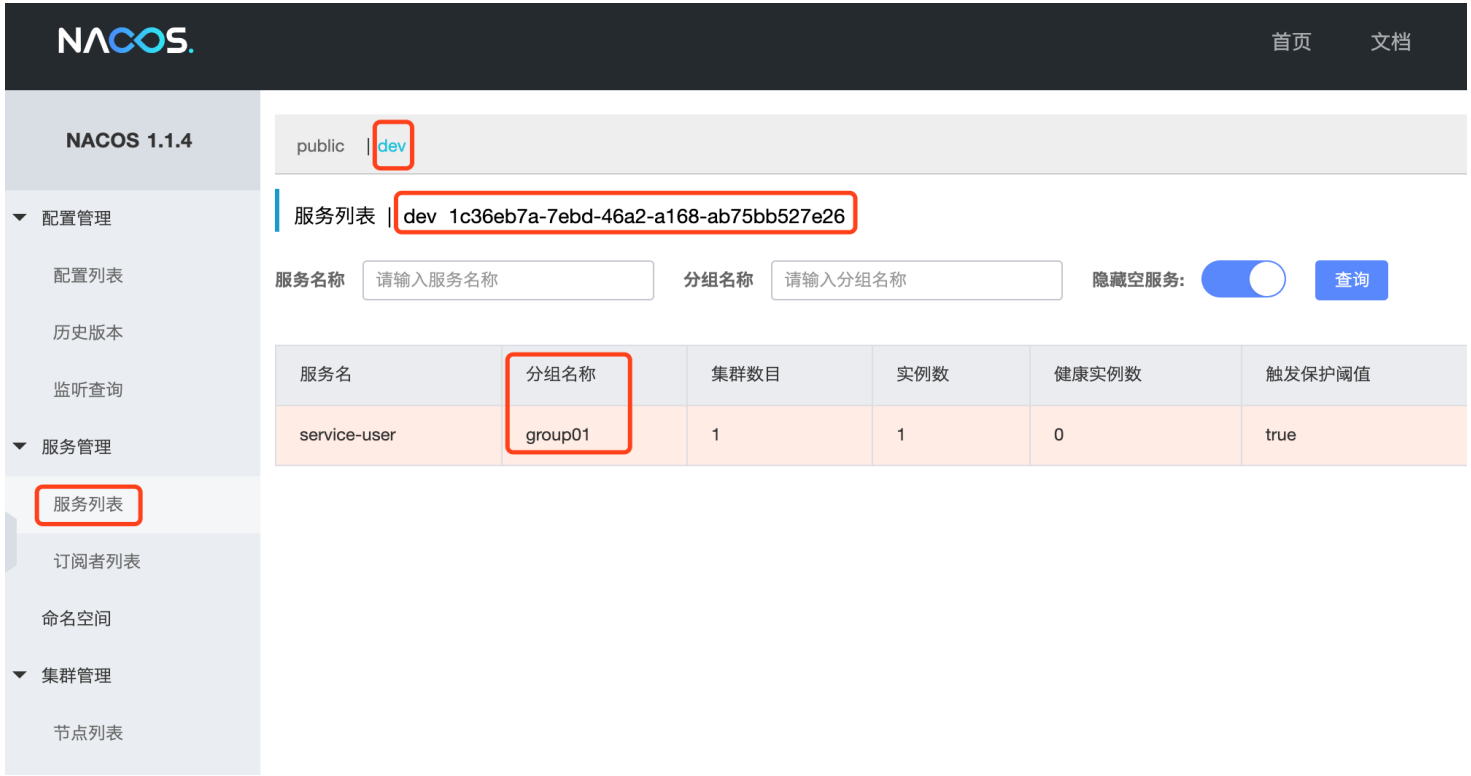
Nacos 服务领域模型实践

创建一个新的服务，设置其命名空间、组、集群。

```
spring:
  application:
    name: service-provider
  cloud:
    nacos:
      discovery:
        server-addr: localhost:8848
        namespace: a72b2b1d-4850-40d3-b831-6f21b3f3626f
        group: group01
        cluster-name: BJ
```

Nacos 设计模型

Nacos 服务领域模型实践



Nacos 设计模型-总结

重难点

1. Nacos 3层数据模型：NameSpace, Group, Service
2. Nacos 服务领域模型：NameSpace, Group, Service, Cluster, Instance
3. Nacos 模型在实际场景的应用与设置方法。

博学谷
www.boxuegu.com

目录 Contents

- 服务发现原理
- Nacos 简介与环境搭建
- 服务提供者与消费者整合 Nacos
- Nacos 设计模型
- 负载均衡与权重
- 集群部署

负载均衡与权重

小节导学

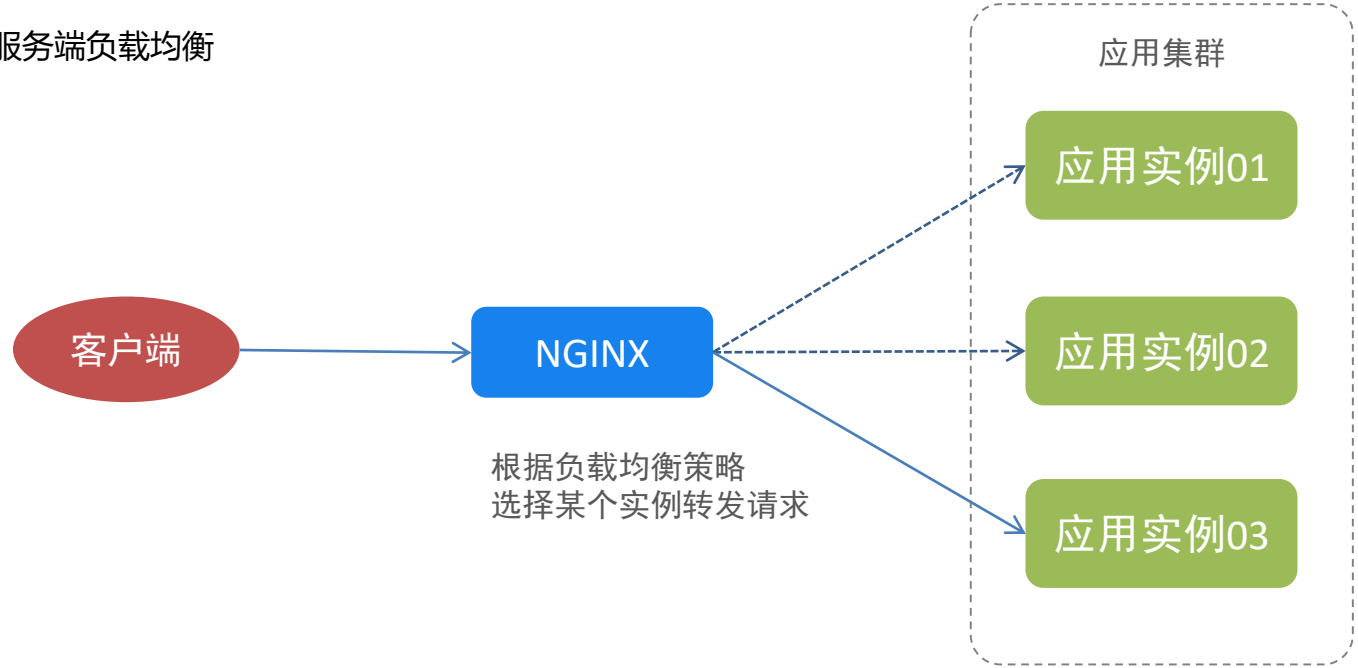
在微服务架构中，为了达到服务的高可用，每个服务都会部署多个实例。服务消费者在调用一个服务时，便面临着应该选择此服务的哪个实例的问题，这就需要做好负载均衡。权重是负载均衡时的一个重要决策依据，本节我们就学习一下如何实现负载均衡和权重。

- 负载均衡的类型
- 实现负载均衡
- Nacos 中权重的设置
- 自定义开发基于权重的负载均衡策略

负载均衡与权重

负载均衡的类型

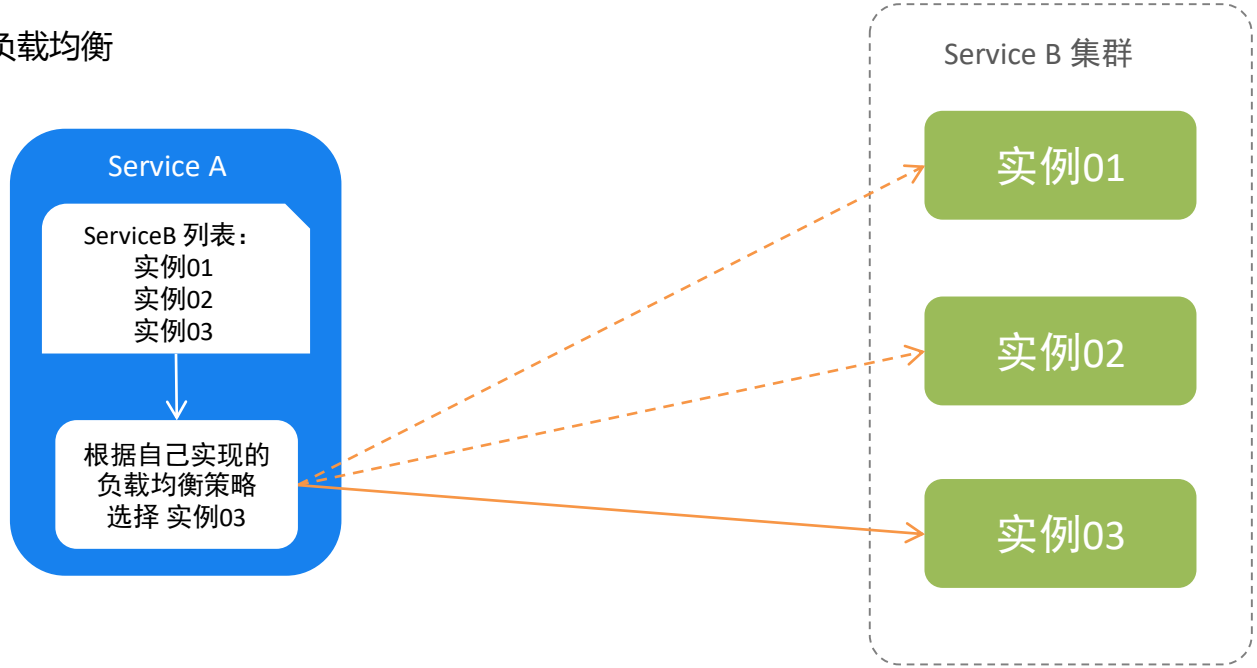
（1）服务端负载均衡



负载均衡与权重

负载均衡的类型

（1）客户端负载均衡



负载均衡与权重

实现负载均衡

实践流程

负载均衡与权重

实现负载均衡

服务提供者的API，输出当前实例的端口号，用于描述当前处理请求的实例。

```
@RestController
public class DemoController {
    @Value("${server.port}")
    private Integer port;
    @GetMapping("/demolb")
    public String demolb(String name){
        return "hello " + name + " " + port;
    }
}
```

负载均衡与权重

实现负载均衡

服务消费者中为 RestTemplate 配置负载均衡注解。

```
@Configuration
public class ConsumerConfig {
    @Bean
    @LoadBalanced
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }
}
```

负载均衡与权重

实现负载均衡

服务消费者中使用服务提供者的名称进行调用：

```
@RestController
public class DemoController {
    @Autowired
    RestTemplate restTemplate;

    @GetMapping("/testlb")
    public String testlb(String name){
        return restTemplate.getForObject("http://service-provider/demolb?name=" + name,
String.class);
    }
}
```

负载均衡与权重

负载均衡策略

策略	含义
RandomRule	随机策略，随机选择一个实例
RoundRobinRule	轮询策略，按照顺序循环选择
RetryRule	重试策略，如果选择的实例访问失败，则重试其他实例，直到找到可以成功访问的
BestAvailableRule	最高可用策略，选择当前并发最小的实例
AvailabilityFilteringRule	可用过滤策略，过滤掉打开熔断的实例、高并发的实例
WeightedResponseTimeRule	响应时间加权策略，响应时间越长权重越小，被选中的可能性越低，反之，被选中的概率高
ZoneAvoidanceRule	区域策略，根据实例及其所在区域的可用性综合情况来选择

负载均衡与权重

实现负载均衡

修改负载均衡策略

```
@Configuration
public class RuleConfig {
    @Bean
    public IRule ribbonRule(){
        return new RandomRule();
    }
}
```

负载均衡与权重

Nacos 控制台设置权重

负载均衡与权重

自定义基于权重的负载均衡策略

服务消费者中实现基于权重的负载均衡策略：

```
@Override
public Server choose(Object o) {
    BaseLoadBalancer loadBalancer = (BaseLoadBalancer) this.getLoadBalancer();
    NamingService namingService = discoveryProperties.namingServiceInstance();
    String name = loadBalancer.getName();
    try {
        Instance instance = namingService.selectOneHealthyInstance(name);
        return new NacosServer(instance);
    } catch (NacosException e) {
        e.printStackTrace();
    }
    return null;
}
```

负载均衡与权重

自定义基于权重的负载均衡策略

服务消费者中配置使用此策略：

```
import com.netflix.loadbalancer.IRule;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RibbonConfig {
    @Bean
    public IRule ribbonRule(){
        return new NacosWeightRule();
    }
}
```

负载均衡与权重-总结

重难点

1. 服务端负载均衡、客户端负载均衡的概念

（谁来做选择实例的决策）

2. 实现负载均衡的方法（@LoadBalanced）

3. 负载均衡的默认策略

4. 自定义基于权重的负载均衡策略的方法

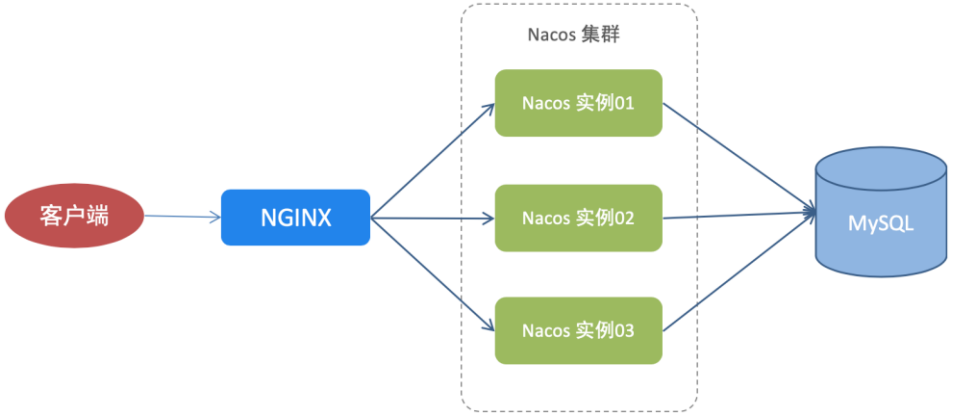
（Nacos 控制台设置权重、创建负载均衡策略）

负载均衡与权重-总结

总结

下节

集群部署（持久化+多节点）



目录 Contents

集群部署

小节导学

之前我们使用的是内存数据库 Derby，实际生产环境下需要使用数据库。

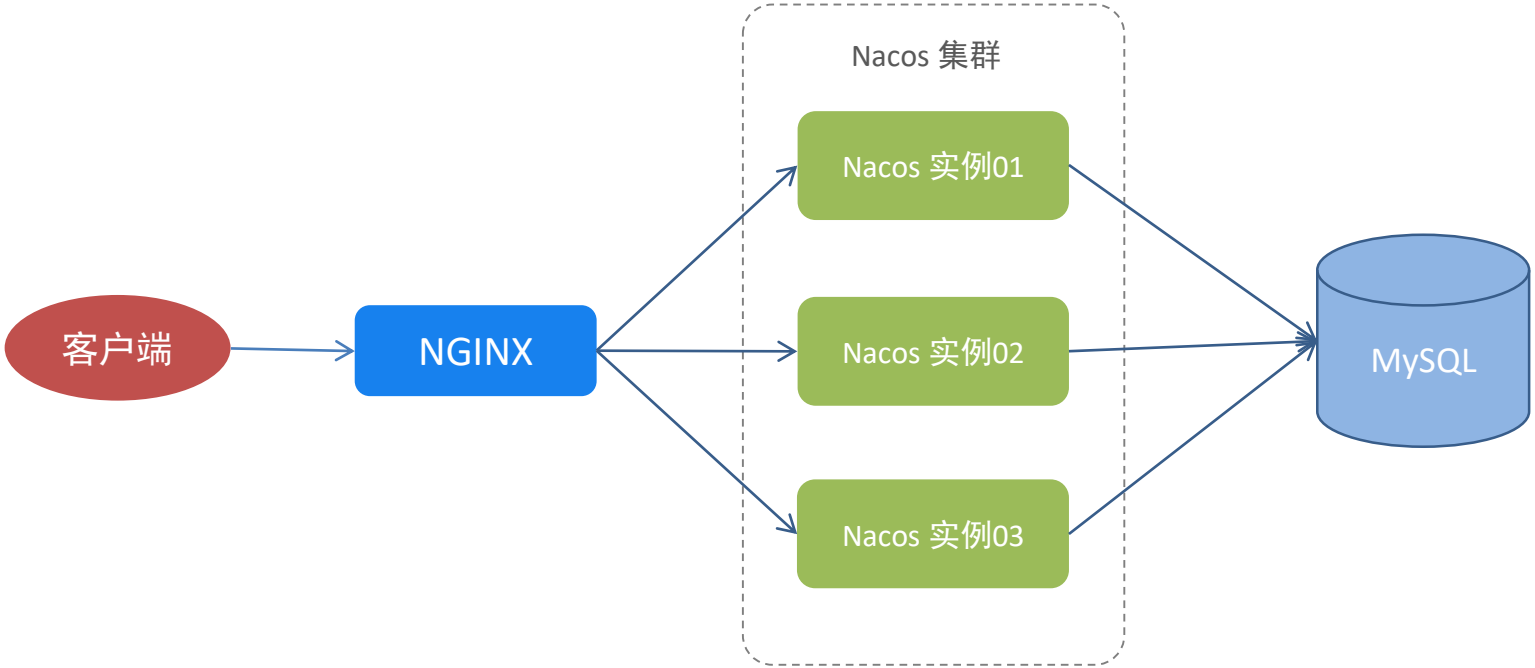
Nacos 节点也不能使用单点结构，需要多节点集群部署。

本节把数据库改为 MySQL，并部署3个Nacos节点，使用 NGINX 做负载均衡。

- 部署结构
- 使用 MySQL 数据库
- Nacos 集群搭建
- NGINX 配置

集群部署

部署结构



集群部署

实践流程

MySQL

- 创建数据库和表

Nacos

- 部署3个节点
- 配置 MySQL 连接
- 配置集群节点IP端口

Nginx

- 配置 Nacos 负载均衡

集群部署

连接 MySQL 数据库

Nacos 已经提供了初始化SQL脚本，conf/nacos-mysql.sql，导入MySQL。

修改 Nacos 配置文件 conf/application.properties，添加数据库连接信息：

```
# 表明用MySQL作为后端存储
spring.datasource.platform=mysql

# 有几个数据库实例
db.num=1

# 第1个实例的地址
db.url.0=jdbc:mysql://[mysqlIP]/nacos?characterEncoding=utf8&connectTimeout=1000&socketTimeout
=3000&autoReconnect=true
db.user=[数据库用户名]
db.password=[数据库密码]
```

集群部署

Nacos 集群搭建

修改每个 Nacos 的端口，conf/application.properties

```
# nacos01
server.port=8841

# nacos02
server.port=8842

# nacos03
server.port=8843
```

集群部署

Nacos 集群搭建

集群配置文件：

```
# 从模板复制一副
cp cluster.conf.example cluster.conf
# 内容改为
127.0.0.1:8841
127.0.0.1:8842
127.0.0.1:8843
```

启动：

```
sh startup.sh
```

集群部署

Nginx 配置

```
upstream nacos {
  server 127.0.0.1:8841;
  server 127.0.0.1:8842;
  server 127.0.0.1:8853;
}
server {
  listen 9090;
  server_name  localhost;
  location /nacos/ {
    proxy_pass http://nacos;
  }
}
```

集群部署-总结

重难点

1. Nacos 使用 MySQL 的配置方式
2. Nacos 集群结构的部署方式
3. NGINX 为 Nacos 集群做负载均衡的方式

一样的在线教育，不一样的教学品质