

软件体系结构原理、方法与实践 (微服务技术栈)

SpringCloud

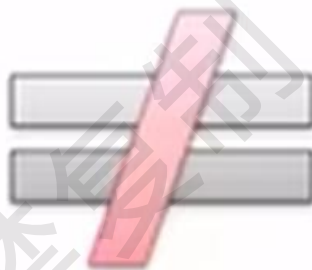


Spring Cloud是一系列框架的有序集合。它利用Spring Boot的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用Spring Boot的开发风格做到一键启动和部署。Spring Cloud并没有重复制造轮子，它只是将各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过Spring Boot风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具

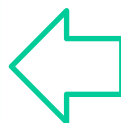
Spring Cloud的子项目，大致可分成两类，一类是对现有成熟框架“Spring Boot化”的封装和抽象，也是数量最多的项目；第二类是开发了一部分分布式系统的基础设施的实现，如Spring Cloud Stream扮演的就是kafka, ActiveMQ这样的角色。对于我们想快速实践微服务的开发者来说，第一类子项目就已经足够使用，如：

- Spring Cloud Netflix 是对Netflix开发的一套分布式服务框架的封装，包括服务的发现和注册，负载均衡、断路器、REST客户端、请求路由等。
- Spring Cloud Config 将配置信息中央化保存，配置Spring Cloud Bus可以实现动态修改配置文件
- Spring Cloud Stream 分布式消息队列，是对Kafka, MQ的封装
- Spring Cloud Security 对Spring Security的封装，并能配合Netflix使用
- Spring Cloud Zookeeper 对Zookeeper的封装，使之能配置其它Spring Cloud的子项目使用
- Spring Cloud Eureka 是 Spring Cloud Netflix 微服务套件中的一部分，它基于Netflix Eureka 做了二次封装，主要负责完成微服务架构中的服务治理功能。

什么是微服务

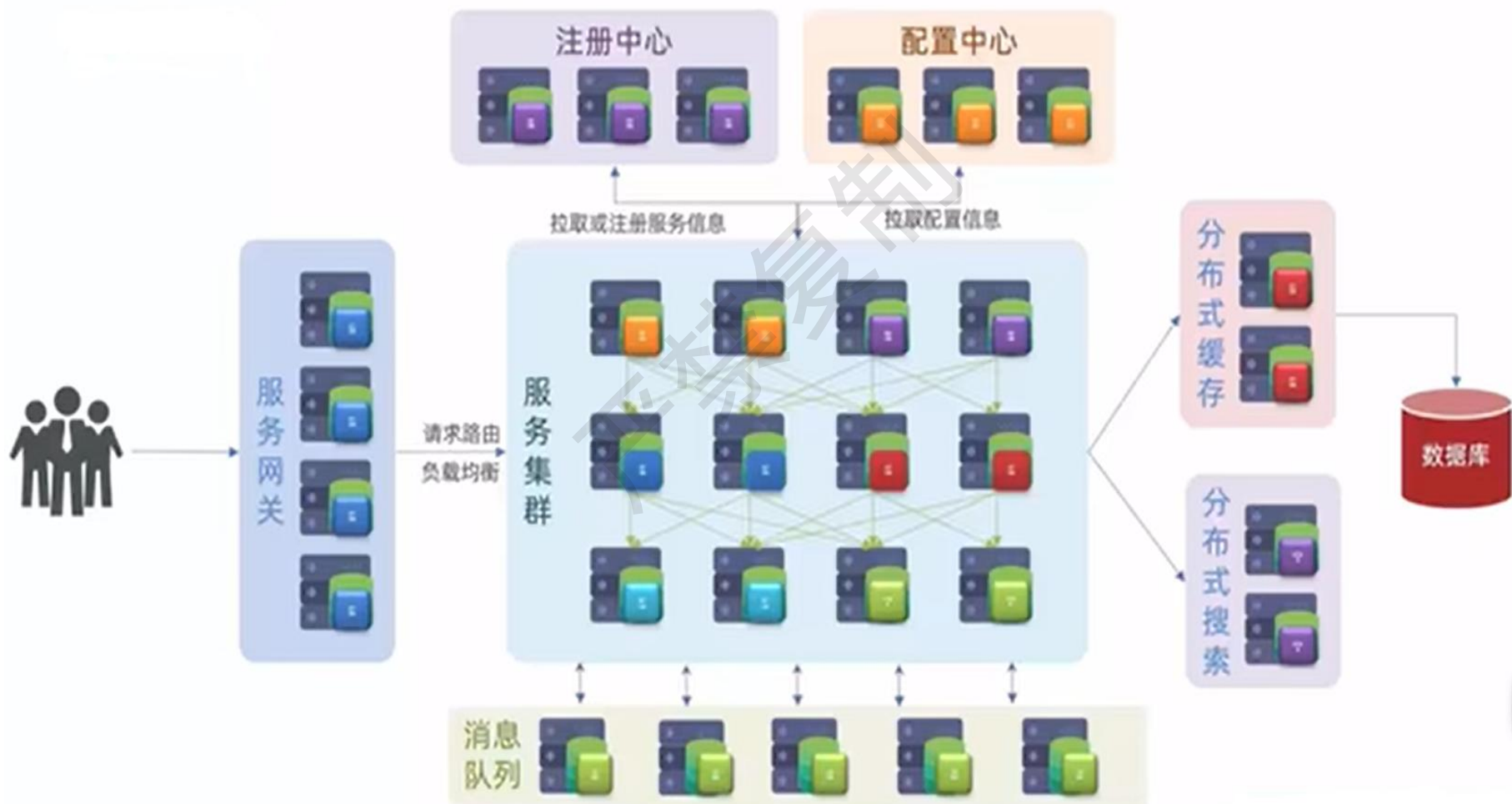


微服务技术栈

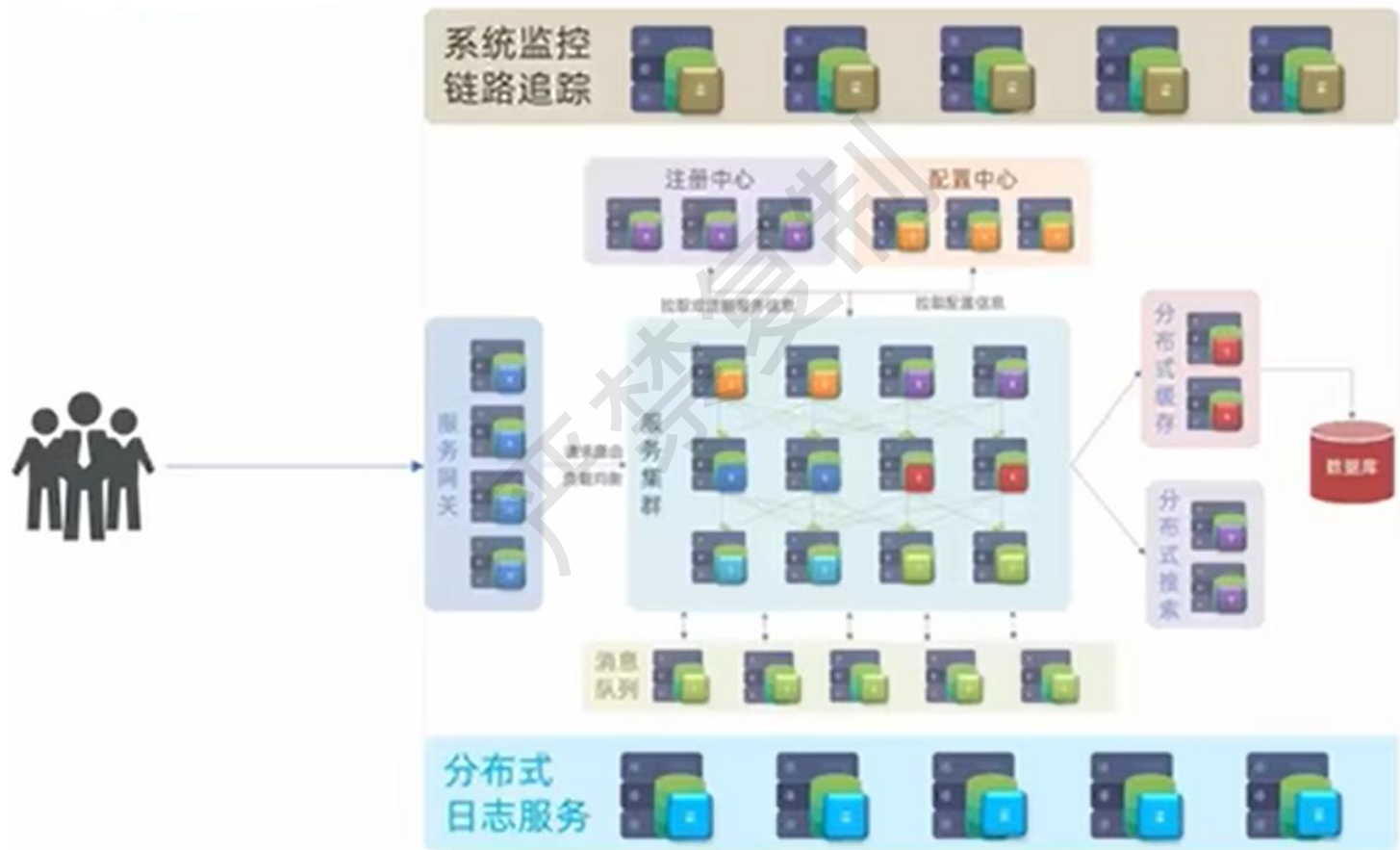


新的开始，新的起点，让我们一起为梦想而努力。

微服务技术栈



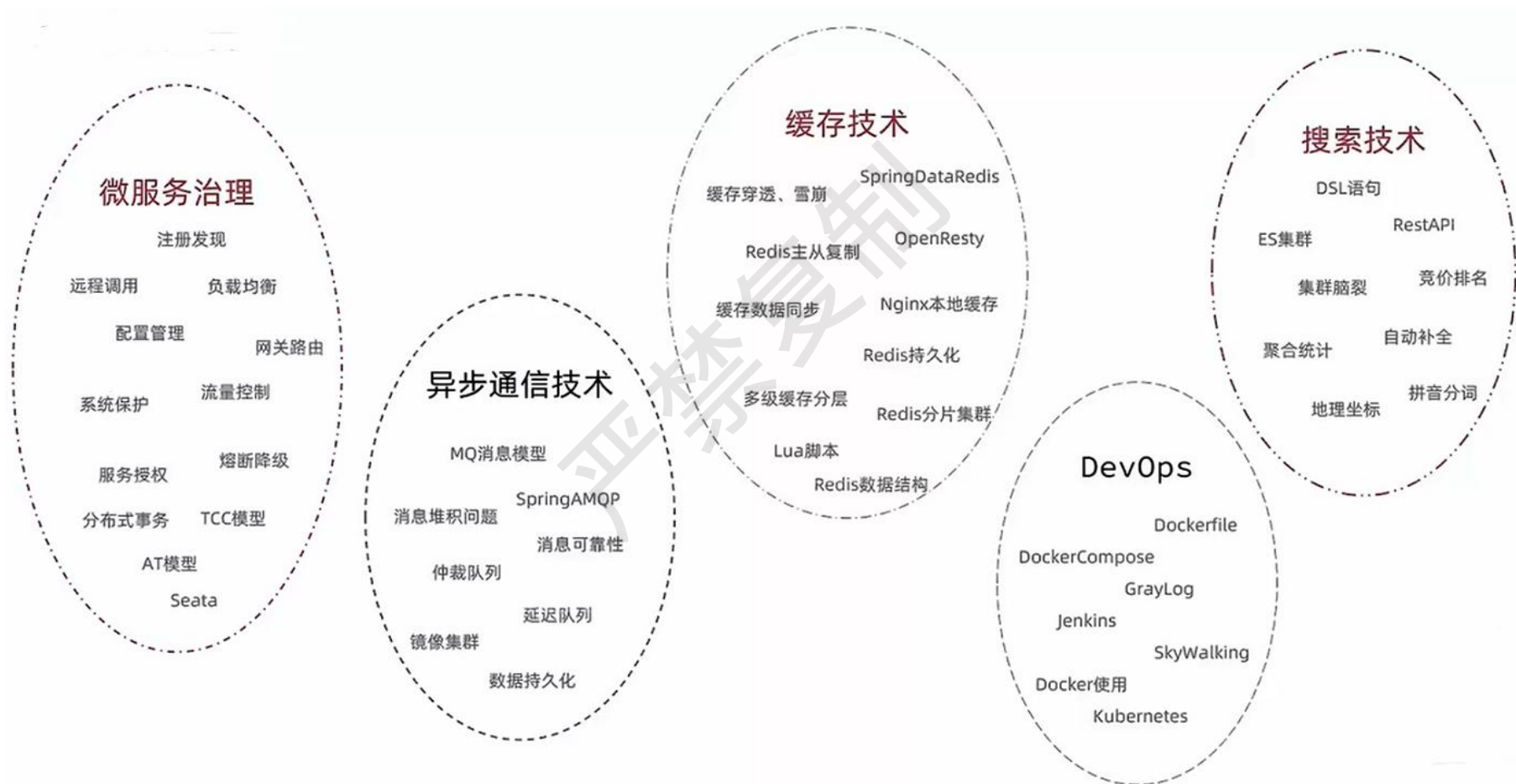
微服务技术栈



微服务技术栈



微服务技术栈



新的开始，新的起点，让我们一起为梦想而努力。

微服务技术栈

实用篇

01 微服务治理

Eureka、Nacos
OpenFeign
网关 Gateway
配置中心 Nacos

02 Docker

Docker原理
Docker使用
Dockerfile
DockerCompose

03 异步通信

同步和异步
MQ技术选型
SpringAMQP
消费者限流

04 分布式搜索

DSL语法
HighLevelClient
拼音搜索
自动补全
竞价排名
地理搜索
聚合统计
分片集群

05 微服务保护

流量控制
系统保护
熔断降级
服务授权

06 分布式事务

XA模式
TCC模式
AT模式
Saga模式

07 分布式缓存

数据持久化
Redis主从集群
哨兵机制
Redis分片集群

08 多级缓存

多级缓存分层
Nginx缓存
Redis缓存
Canal数据同步

09 可靠消息服务

消息三方确认
惰性队列
延迟队列
镜像集群
仲裁队列

高级篇

面试篇

01 Nacos源码

Nacos的服务发现原理
Nacos服务注册原理
Nacos心跳机制
Nacos、Eureka差异

02 Sentinel源码

Sentinel滑动窗口算法
令牌桶算法
漏桶算法

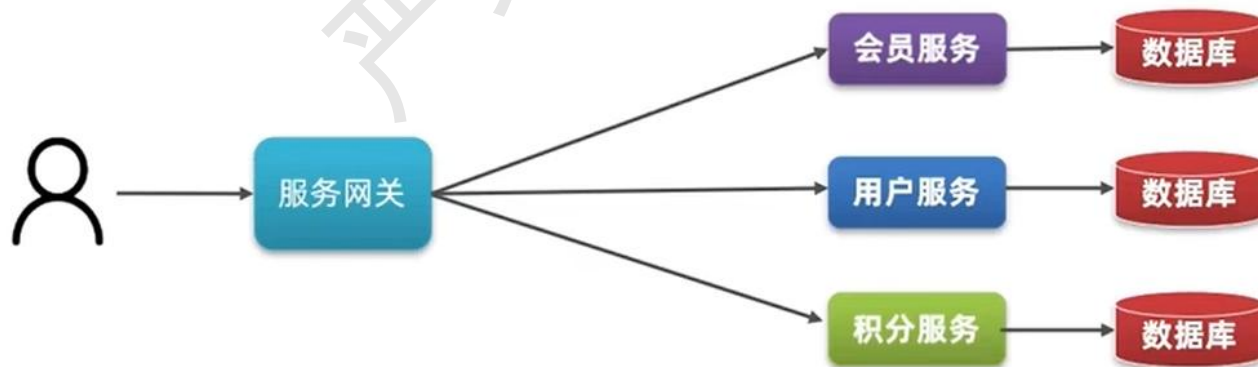
03 Redis热点问题

分布式锁问题
缓存穿透
缓存击穿
缓存雪崩

认识微服务

微服务是一种经过良好架构设计的**分布式**架构方案，微服务架构特征：

- 单一职责：微服务拆分粒度更小，每一个服务都对应唯一的业务能力，做到单一职责，避免重复业务开发
- 面向服务：微服务对外暴露业务接口
- 自治：团队独立、技术独立、数据独立、部署独立
- 隔离性强：服务调用做好隔离、容错、降级，避免出现级联问题



❁ 微服务与SOA的区别

	微服务	SOA
设计理念	微服务是以业务能力为中心，重视服务的自治性，每个服务都有自己独立的团队、数据、环境、技术栈等，强调敏捷和快速迭代。	SOA注重的是服务的重用性，倾向于通过统一的服务总线将所有服务连接起来，形成全局的服务视图，强调的是系统的整体性和一致性。
数据管理	在微服务架构中，每个服务都有自己的独立的数据存储，以实现服务之间的松耦合。	在SOA架构中，常常使用统一的数据存储，以便进行全局的数据管理和数据共享。
服务规模	微服务倾向于创建小型、轻量级的服务，可以快速启动和停止，便于进行快速迭代和持续交付	SOA的服务通常比较大型，包含多个功能，通常更加复杂和庞大。
通信方式	微服务通常使用轻量级的通信方式，如HTTP/REST，JSON等。	SOA通常使用企业服务总线（ESB）进行通信，更加复杂和重量级。
服务治理	微服务倾向于使用轻量级的服务治理方式，强调服务的自治性	SOA通常需要更加复杂的服务治理机制，包括服务的注册、发现、路由、版本控制等。
部署方式	微服务支持独立部署，可以进行持续集成和持续部署。	SOA的服务通常需要一起部署，更新和维护更加复杂。

新的开始，新的起点，让我们一起为梦想而努力。



认识微服务

	Dubbo	SpringCloud	SpringCloudAlibaba
注册中心	zookeeper、Redis	Eureka、Consul	Nacos、Eureka
服务远程调用	Dubbo协议	Feign (http协议)	Dubbo、Feign
配置中心	无	SpringCloudConfig	SpringCloudConfig、Nacos
服务网关	无	SpringCloudGateway、Zuul	SpringCloudGateway、Zuul
服务监控和保护	dubbo-admin, 功能弱	Hystrix	Sentinel

企业需求

- 使用SpringCloud技术栈
- 服务接口采用Restful风格
- 服务调用采用Feign方式

SpringCloud + Feign

SpringCloudAlibaba
+ Feign

- 使用SpringCloudAlibaba技术栈
- 服务接口采用Restful风格
- 服务调用采用Feign方式

微服务

SpringCloudAlibaba
+ Dubbo

Dubbo原始模式

- 基于Dubbo老旧技术体系
- 服务接口采用Dubbo协议标准
- 服务调用采用Dubbo方式

- 使用SpringCloudAlibaba技术栈
- 服务接口采用Dubbo协议标准
- 服务调用采用Dubbo方式

新的开始，新的起点，让我们一起为梦想而努力。

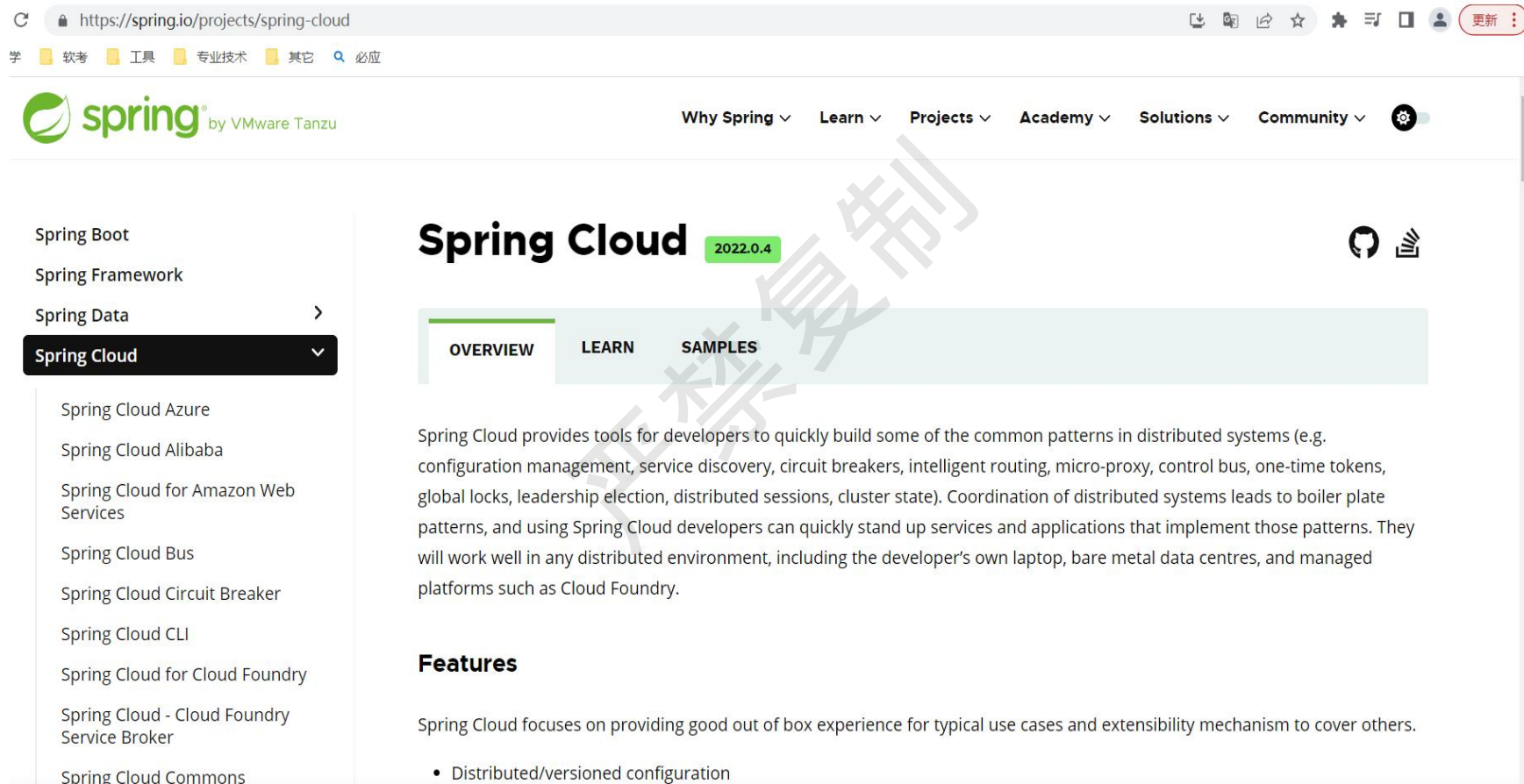
认识微服务

SpringCloud

- SpringCloud是目前国内使用最广泛的微服务框架。官网地址: <https://spring.io/projects/spring-cloud>。
- SpringCloud集成了各种微服务功能组件, 并基于SpringBoot实现了这些组件的自动装配, 从而提供了良好的开箱即用体验:



认识微服务



Spring Cloud 2022.0.4

OVERVIEW LEARN SAMPLES

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state). Coordination of distributed systems leads to boiler plate patterns, and using Spring Cloud developers can quickly stand up services and applications that implement those patterns. They will work well in any distributed environment, including the developer's own laptop, bare metal data centres, and managed platforms such as Cloud Foundry.

Features

Spring Cloud focuses on providing good out of box experience for typical use cases and extensibility mechanism to cover others.

- Distributed/versioned configuration

Spring Boot
Spring Framework
Spring Data
Spring Cloud

Spring Cloud Azure
Spring Cloud Alibaba
Spring Cloud for Amazon Web Services
Spring Cloud Bus
Spring Cloud Circuit Breaker
Spring Cloud CLI
Spring Cloud for Cloud Foundry
Spring Cloud - Cloud Foundry Service Broker
Spring Cloud Commons

认识微服务

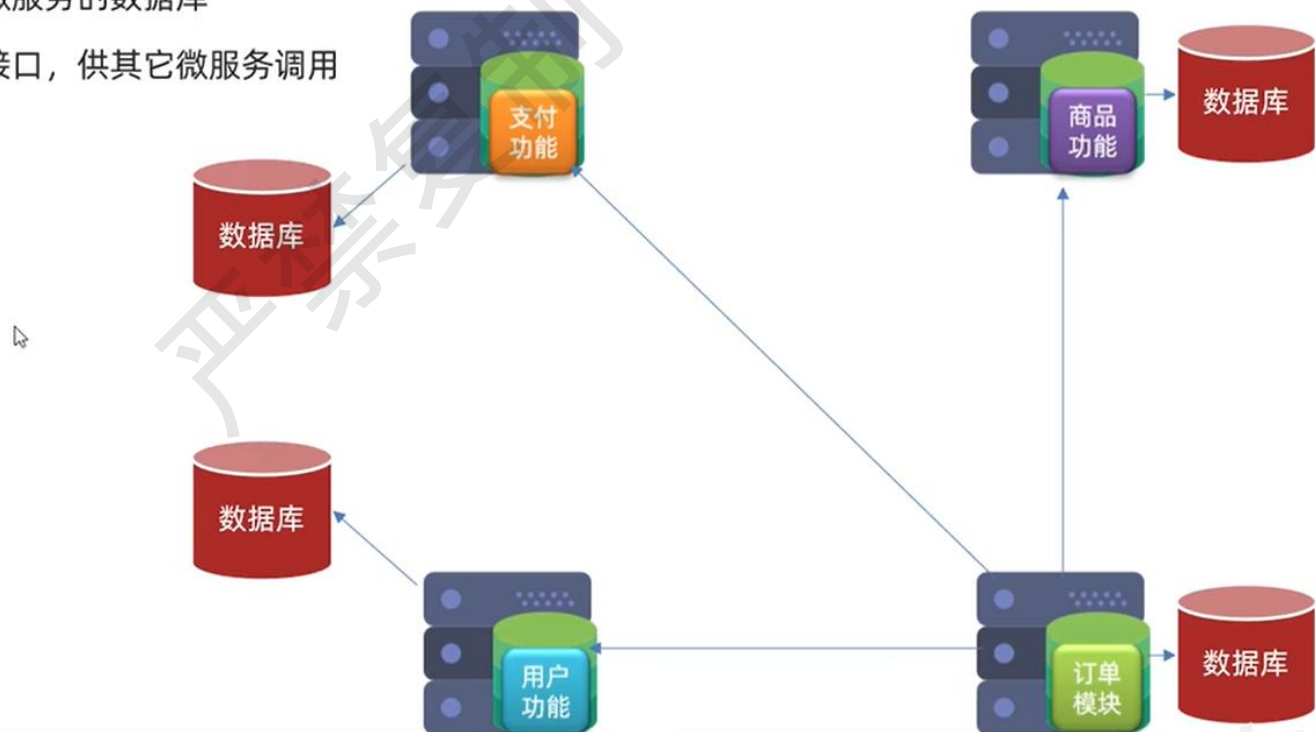
- SpringCloud与SpringBoot的版本兼容关系如下:

Release Train	Boot Version
2020.0.x aka Ilford	2.4.x
Hoxton	2.2.x, 2.3.x (Starting with SR5)
Greenwich	2.1.x
Finchley	2.0.x
Edgware	1.5.x
Dalston	1.5.x

微服务拆分及远程调用

微服务拆分注意事项

1. 不同微服务，不要重复开发相同业务
2. 微服务数据独立，不要访问其它微服务的数据库
3. 微服务可以将自己的业务暴露为接口，供其它微服务调用



微服务拆分及远程调用



案例

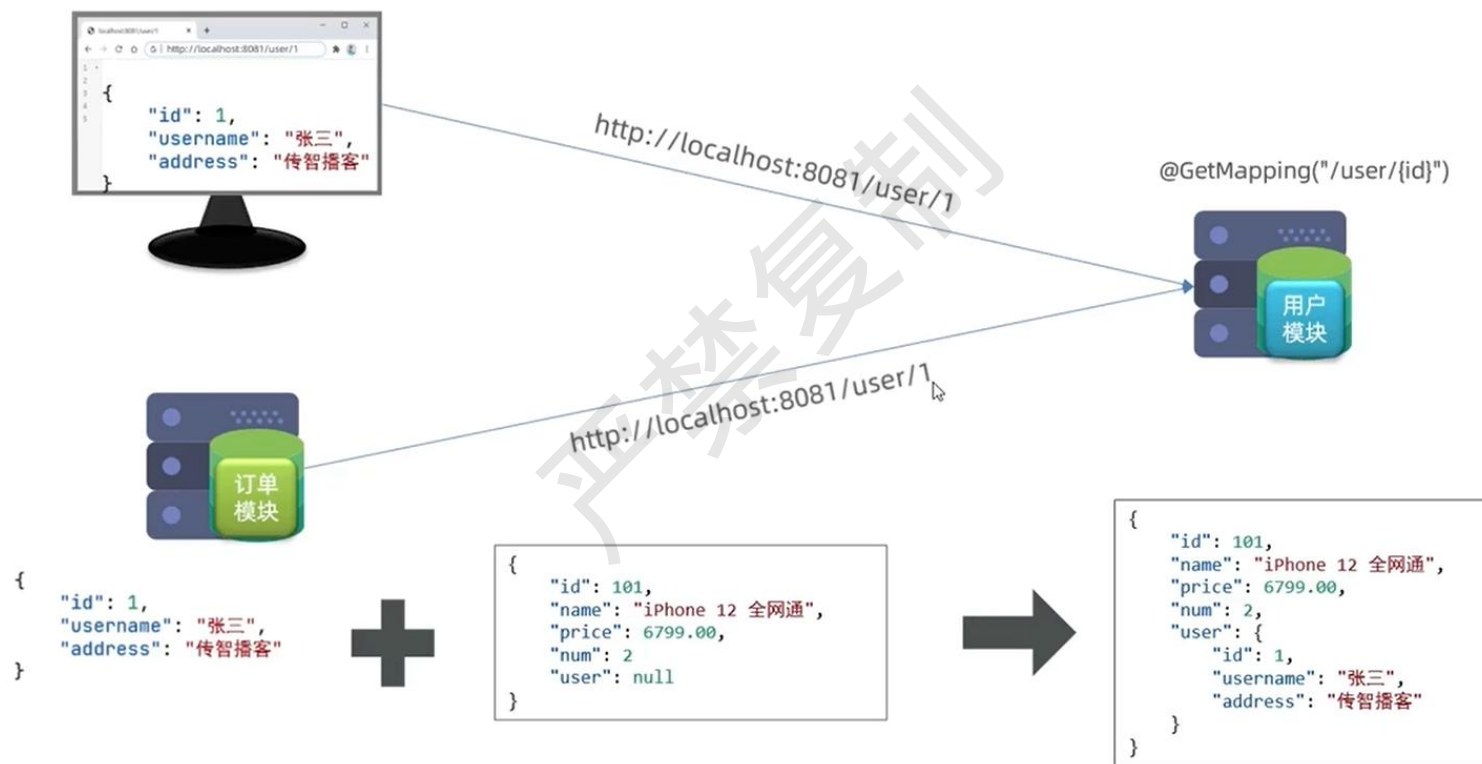
根据订单id查询订单功能

需求：根据订单id查询订单的同时，把订单所属的用户信息一起返回



新的开始，新的起点，让我们一起为梦想而努力。

微服务拆分及远程调用



🌀 Eureka服务提供者与消费者

提供者与消费者

- 服务提供者：一次业务中，被其它微服务调用的服务。（提供接口给其它微服务）
- 服务消费者：一次业务中，调用其它微服务的服务。（调用其它微服务提供的接口）



- 服务A调用服务B，服务B调用服务C，那么服务B是什么角色？

Eureka注册中心

服务调用出现的问题

- 服务消费者该如何获取服务提供者的地址信息?
- 如果有多个服务提供者, 消费者该如何选择?
- 消费者如何得知服务提供者的健康状态?

order-service
8080



user-service
8083

user-service
8081

user-service
8082


Eureka注册中心

Eureka的作用

- 消费者该如何获取服务提供者具体信息？
 - ◆ 服务提供者启动时向eureka注册自己的信息
 - ◆ eureka保存这些信息
 - ◆ 消费者根据服务名称向eureka拉取提供者信息
- 如果有多个服务提供者，消费者该如何选择？
 - ◆ 服务消费者利用负载均衡算法，从服务列表中挑选一个
- 2) use ● 消费者如何感知服务提供者健康状态？
 - ◆ 服务提供者会每隔30秒向EurekaServer发送心跳请求，报告健康状态
 - ◆ eureka会更新记录服务列表信息，心跳不正常会被剔除
 - ◆ 消费者就可以拉取到最新的信息



🌱 Eureka注册中心



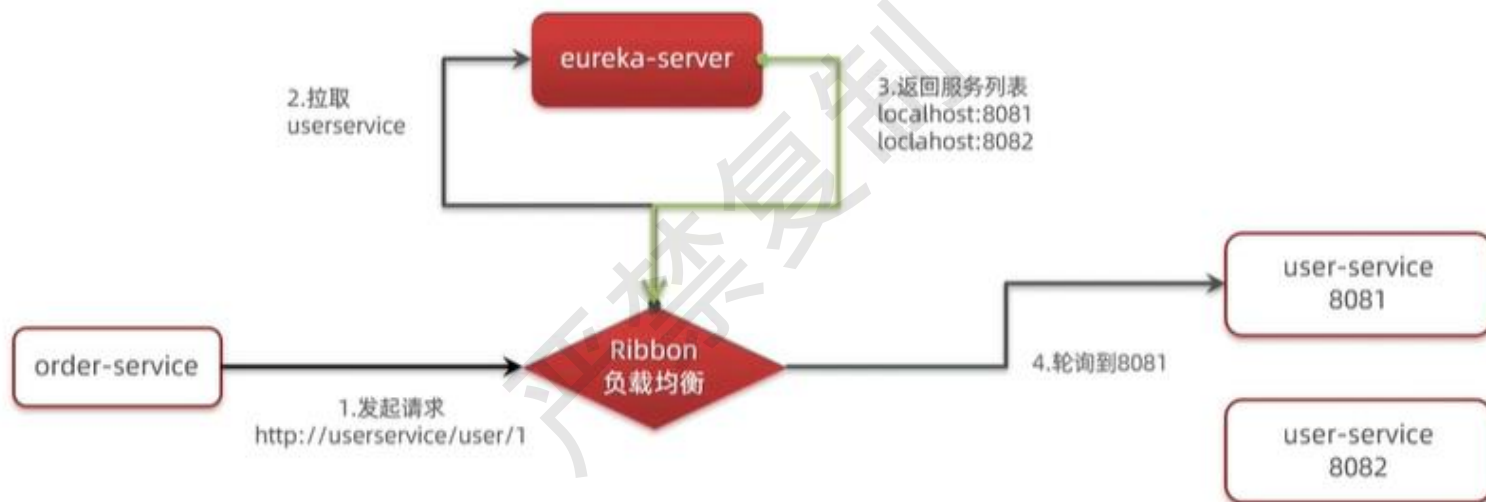
总结

在Eureka架构中，微服务角色有两类：

- EurekaServer：服务端，注册中心
 - ◆ 记录服务信息
 - ◆ 心跳监控
- EurekaClient：客户端
 - ◆ Provider：服务提供者，例如案例中的 user-service
 - ◆ 注册自己的信息到EurekaServer
 - ◆ 每隔30秒向EurekaServer发送心跳
 - ◆ consumer：服务消费者，例如案例中的 order-service
 - ◆ 根据服务名称从EurekaServer拉取服务列表
 - ◆ 基于服务列表做负载均衡，选中一个微服务后发起远程调用

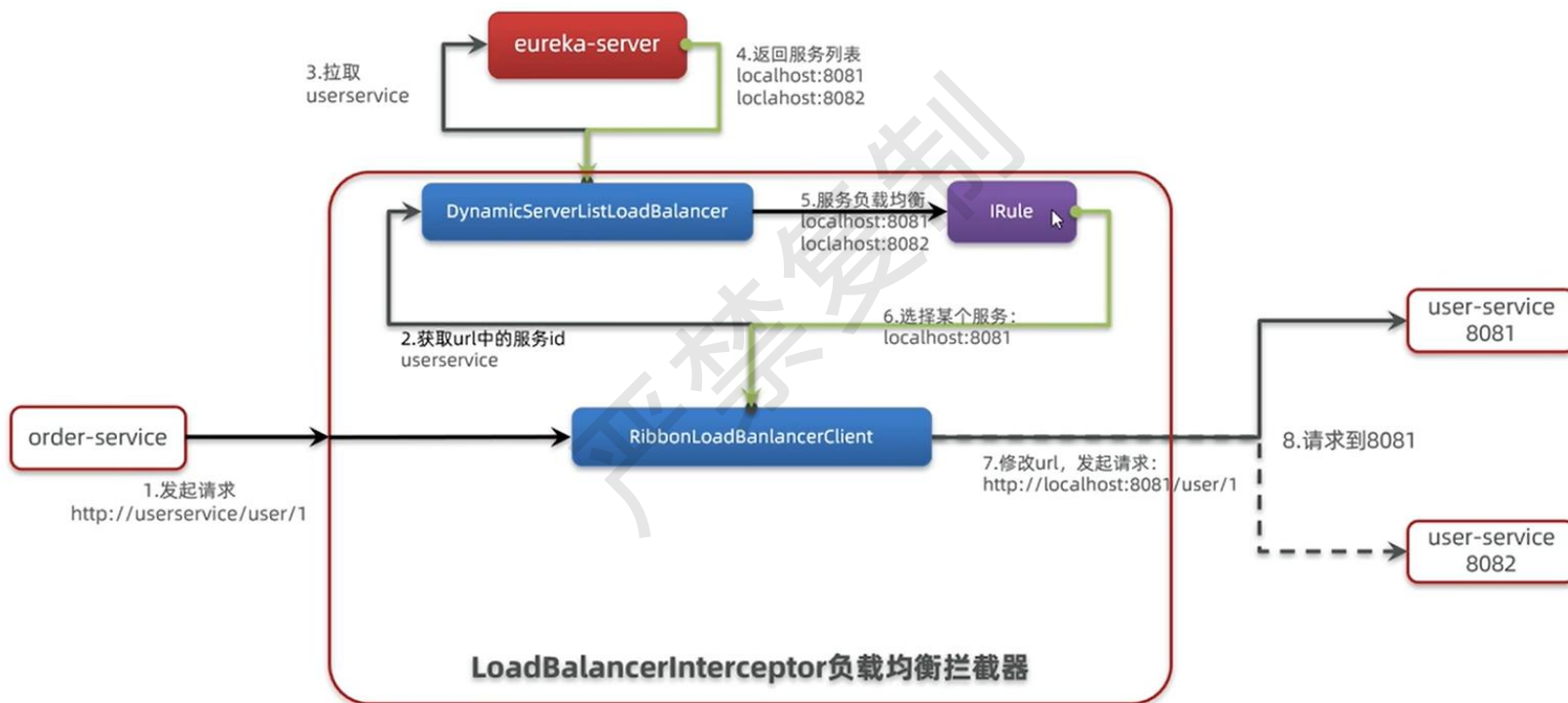
🌀 Ribbon负载均衡

负载均衡流程



Ribbon负载均衡

负载均衡流程



负载均衡策略: 轮循, 随机....

新的开始, 新的起点, 让我们一起为梦想而努力。

🌀 Nacos注册中心

Nacos是阿里巴巴的产品，现在是SpringCloud中的一个组件。相比Eureka功能更加丰富，在国内受欢迎程度较高。

The image shows the Nacos landing page. At the top, the word "NACOS." is displayed in large white letters on a dark background. Below it, a line of text reads "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。". In the center, there are two buttons: a blue one labeled "前往 Github" and a white one with a black border labeled "手册". Below these buttons, there are two statistics: "★ 27627" and "🔖 12251". Further down, there are two version links: "V2.2.3 版本说明" and "V1.4.6". At the bottom, it says "2023年5月25日发布".

NACOS.

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。

前往 Github 手册

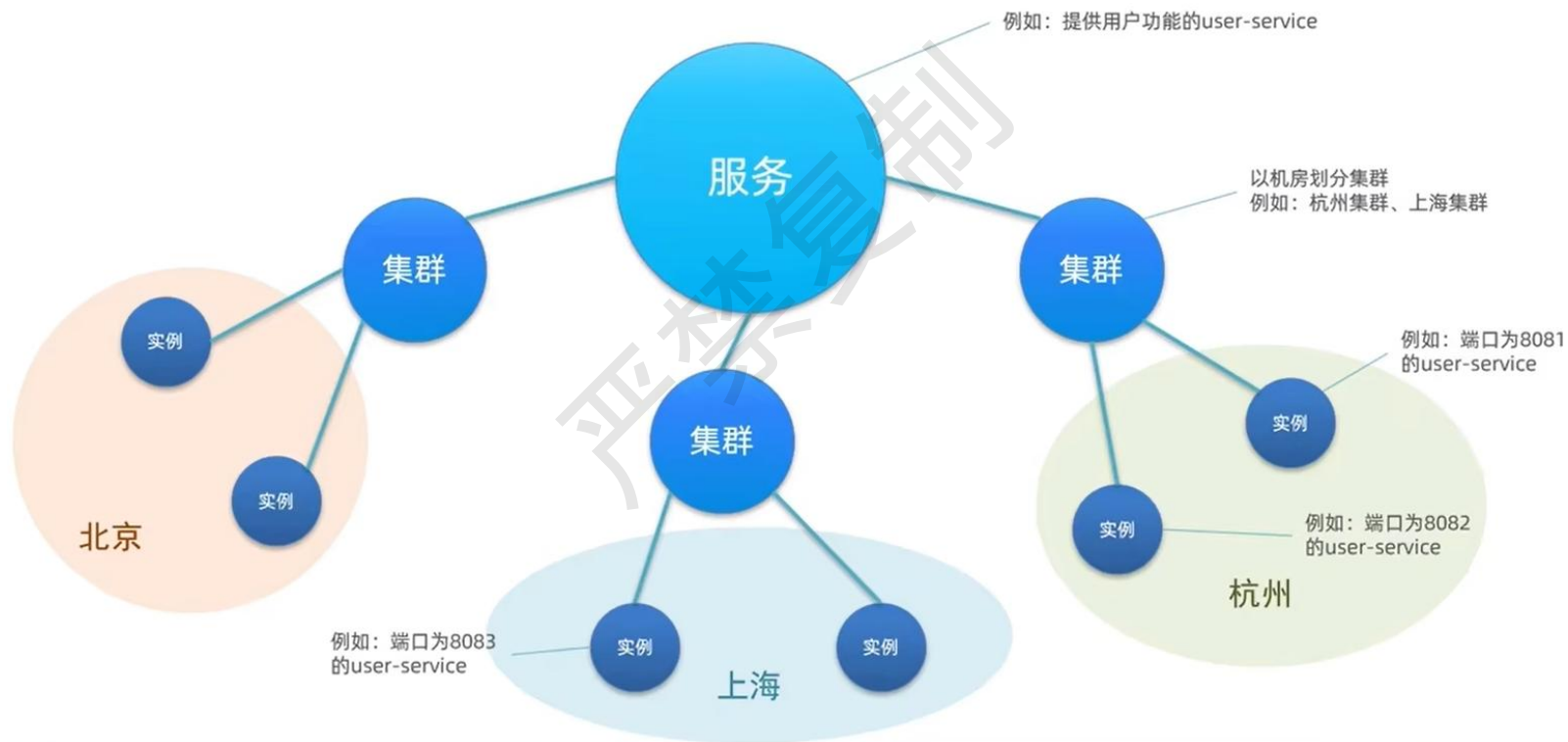
★ 27627 🔖 12251

V2.2.3 版本说明 V1.4.6

2023年5月25日发布

Nacos注册中心

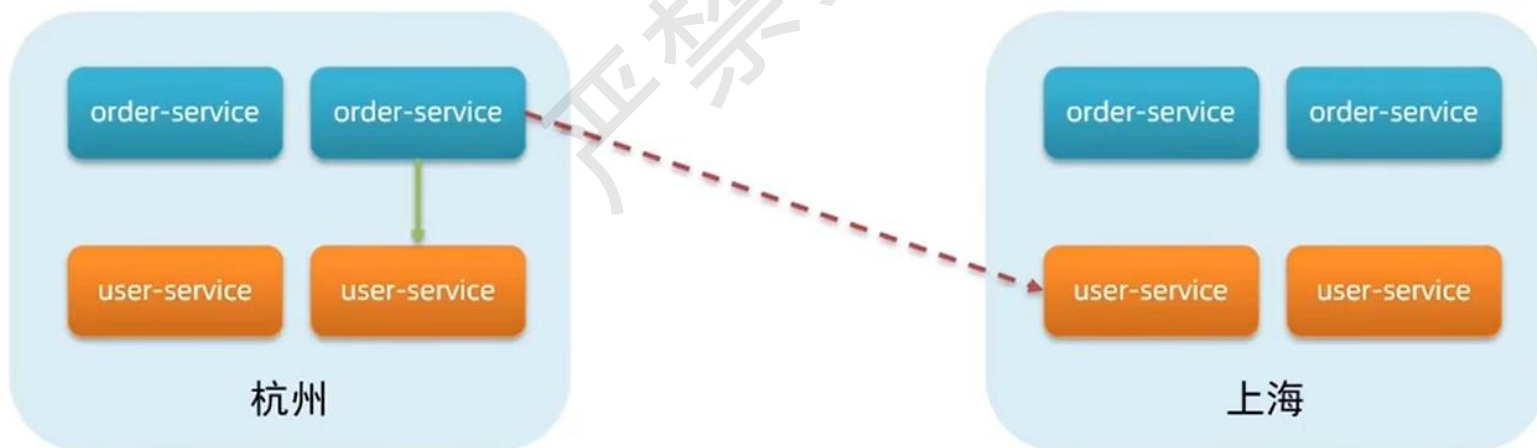
Nacos服务分级存储模型



🌐 Nacos注册中心

服务跨集群调用问题

服务调用尽可能选择本地集群的服务，跨集群调用延迟较高
本地集群不可访问时，再去访问其它集群



🌀 Nacos注册中心



总结

Nacos服务分级存储模型

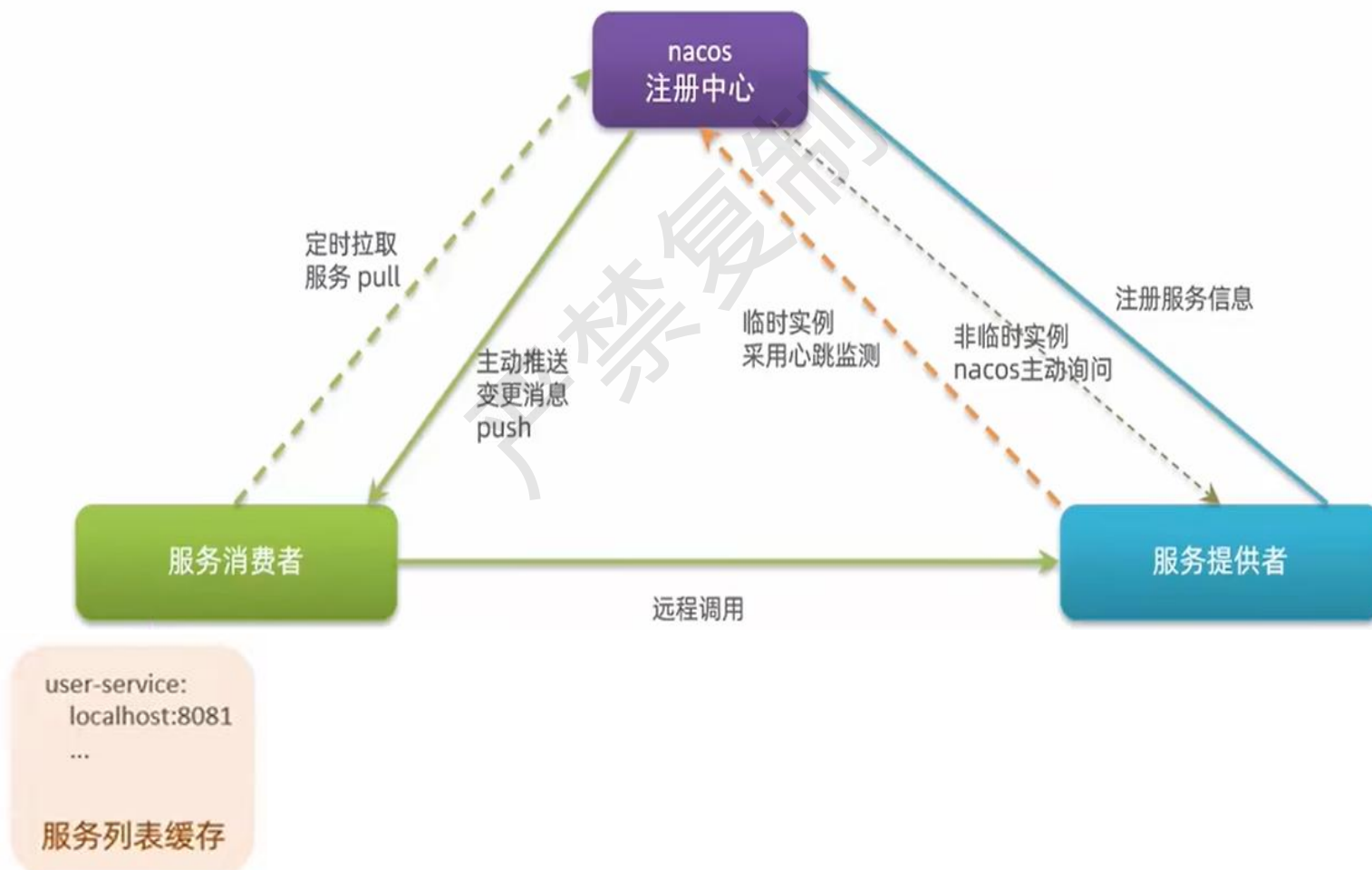
- ① 一级是服务，例如userservice
- ② 二级是集群，例如杭州或上海
- ③ 三级是实例，例如杭州机房的某台部署了userservice的服务器

NacosRule负载均衡策略

- ① 优先选择同集群服务实例列表
- ② 本地集群找不到提供者，才去其它集群寻找，并且会报警告
- ③ 确定了可用实例列表后，再采用随机负载均衡挑选实例

Nacos注册中心

nacos注册中心细节分析



⚙ HTTP客户端-Feign

RestTemplate方式调用存在的问题

先来看我们以前利用RestTemplate发起远程调用的代码：

```
String url = "http://userservice/user/" + order.getUserId();  
User user = restTemplate.getForObject(url, User.class);
```

存在下面的问题：

- 代码可读性差，编程体验不统一
- 参数复杂URL难以维护

🌐 HTTP客户端-Feign

Feign是一个声明式的http客户端，官方地址：<https://github.com/OpenFeign/feign>

其作用就是帮助我们优雅的实现http请求的发送，解决上面提到的问题。

Feign makes writing java http clients easier

gitter join chat  maven central 11.1

Feign is a Java to HTTP client binder inspired by [Retrofit](#), [JAXRS-2.0](#), and [WebSocket](#). Feign's first goal was reducing the complexity of binding [Denominator](#) uniformly to HTTP APIs regardless of [ReSTfulness](#).

✿ HTTP客户端-Feign

使用Feign的步骤如下：

3. 编写Feign客户端：

```
@FeignClient("userservice")
public interface UserClient {
    @GetMapping("/user/{id}")
    User findById(@PathVariable("id") Long id);
}
```

主要是基于SpringMVC的注解来声明远程调用的信息，比如：

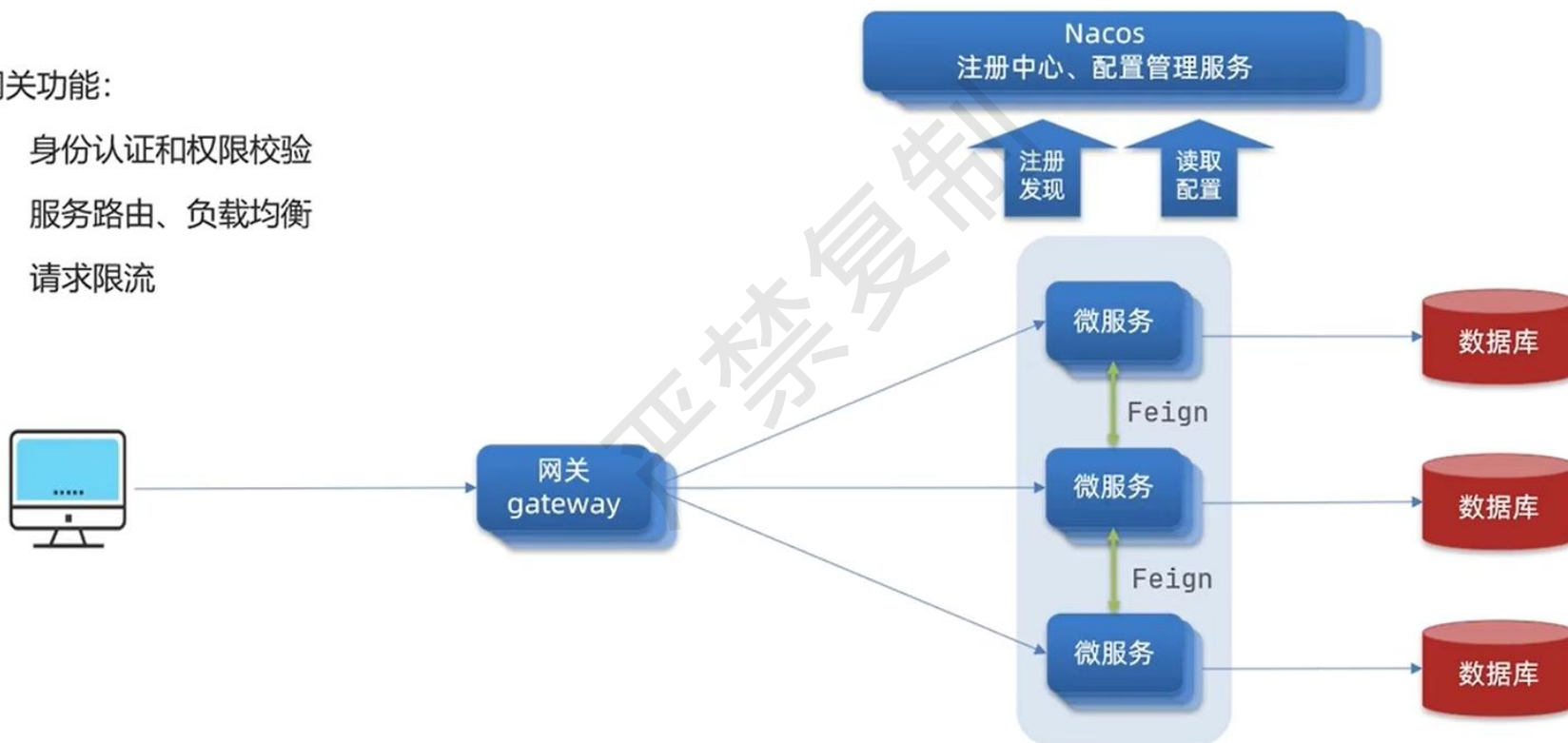
- 服务名称：userservice
- 请求方式：GET
- 请求路径：/user/{id}
- 请求参数：Long id
- 返回值类型：User

新的开始，新的起点，让我们一起为梦想而努力。

🌐 统一网关-Gateway

网关功能:

- 身份认证和权限校验
- 服务路由、负载均衡
- 请求限流





❁ 统一网关-Gateway

网关实现技术

在SpringCloud中网关的实现包括两种：

- gateway
- zuul

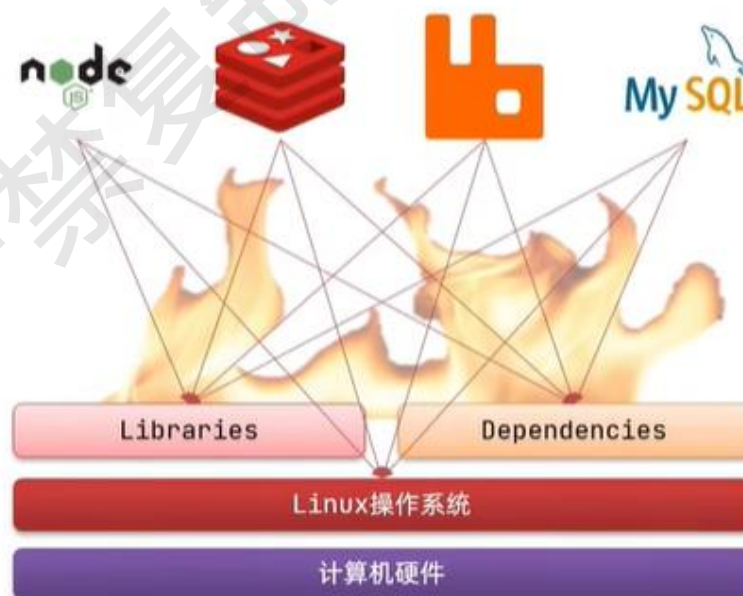
Zuul是基于Servlet的实现，属于阻塞式编程。而SpringCloudGateway则是基于Spring5中提供的WebFlux，属于响应式编程的实现，具备更好的性能。

❁ 初识Docker

项目部署的问题

大型项目组件较多，运行环境也较为复杂，部署时会碰到一些问题：

- 依赖关系复杂，容易出现兼容性问题
- 开发、测试、生产环境有差异



新的开始，新的起点，让我们一起为梦想而努力。

❁ 初识Docker

Docker

Docker如何解决依赖的兼容问题的？

- 将应用的Libs（函数库）、Deps（依赖）、配置与应用一起打包
- 将每个应用放到一个隔离容器去运行，避免互相干扰



新的开始，新的起点，让我们一起为梦想而努力。

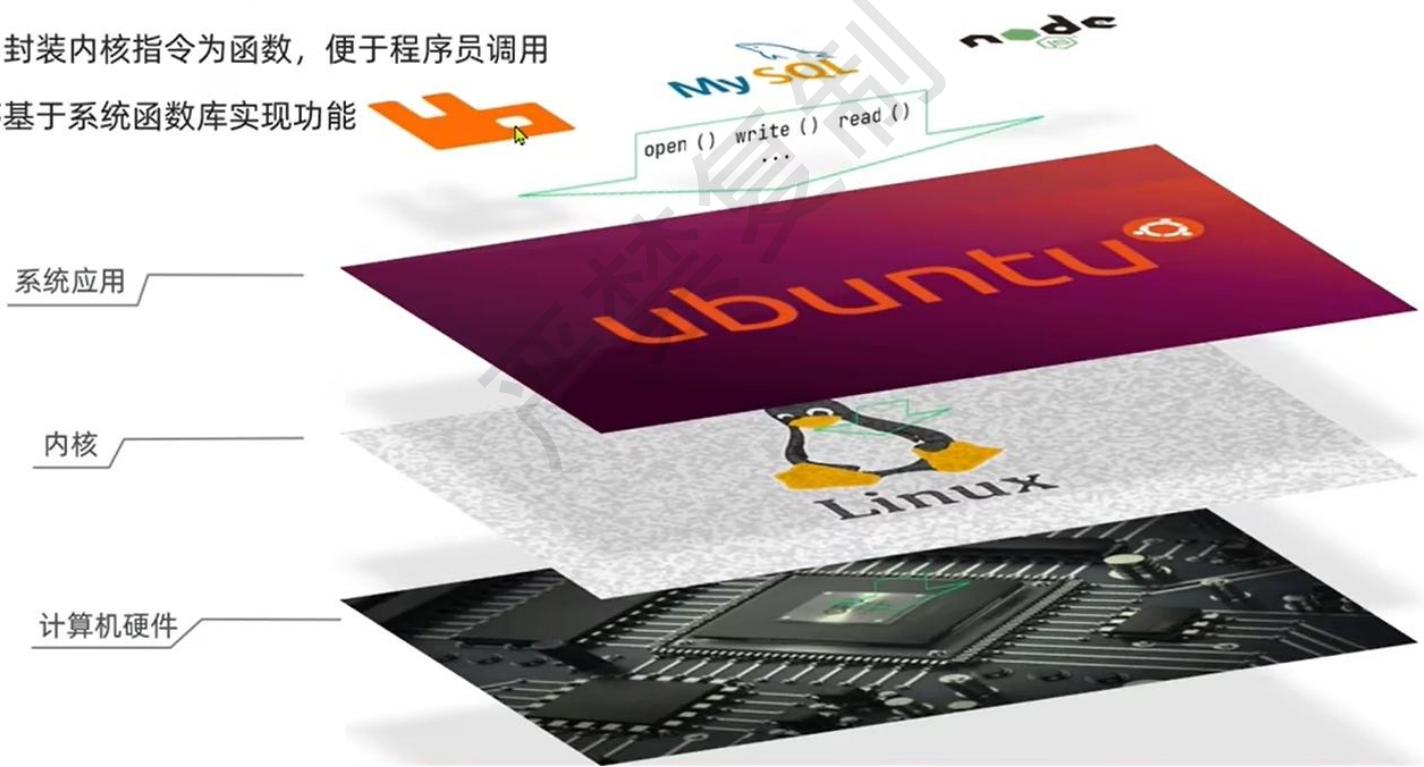
❁ 初识Docker

Docker

内核与硬件交互，提供操作硬件的指令

系统应用封装内核指令为函数，便于程序员调用

用户程序基于系统函数库实现功能



❁ 初识Docker

Docker

Ubuntu和CentOS都是基于Linux内核，只是系统应用不同，提供的函数库有差异

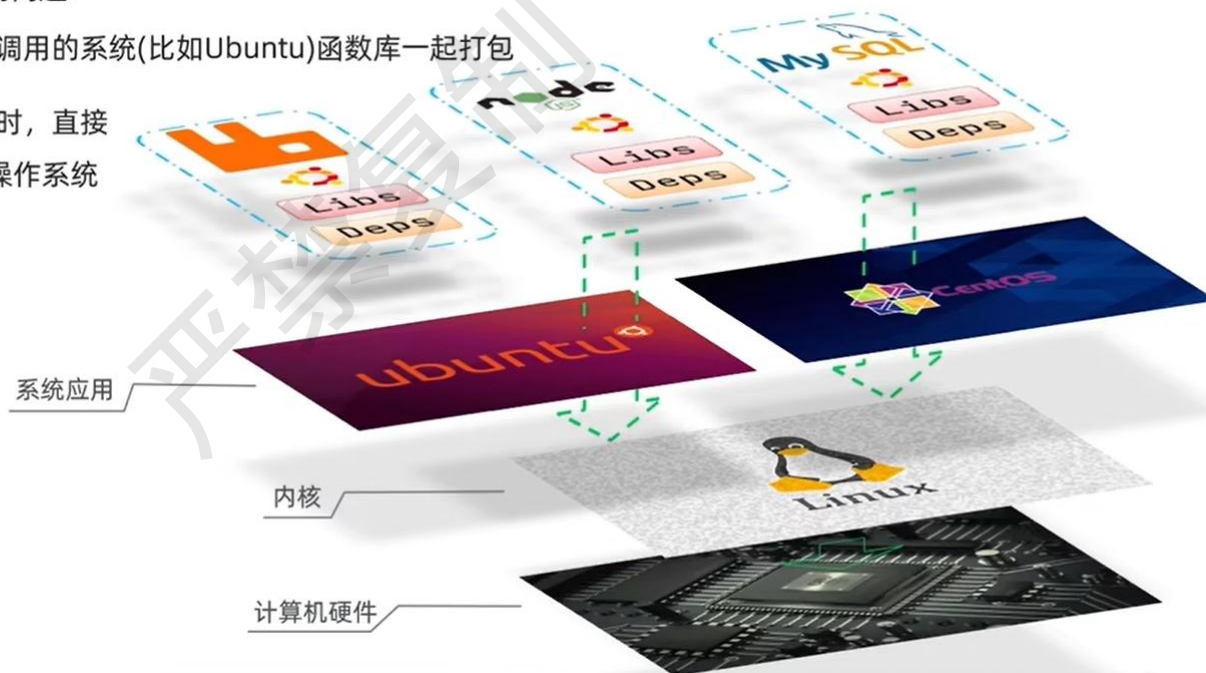


❁ 初识Docker

Docker

Docker如何解决不同系统环境的问题？

- Docker将用户程序与所需要调用的系统(比如Ubuntu)函数库一起打包
- Docker运行到不同操作系统时，直接基于打包的库函数，借助于操作系统的Linux内核来运行



❁ 初识Docker

Docker

Docker如何解决大型项目依赖关系复杂，不同组件依赖的兼容性问题？

- Docker允许开发中将应用、依赖、函数库、配置一起**打包**，形成可移植镜像
- Docker应用运行在容器中，使用沙箱机制，相互**隔离**

Docker如何解决开发、测试、生产环境有差异的问题

- Docker镜像中包含完整运行环境，包括系统函数库，仅依赖系统的Linux内核，因此可以在任意Linux操作系统上运行



新的开始，新的起点，让我们一起为梦想而努力。

❁ 初识Docker

虚拟机(virtual machine)是在操作系统中模拟硬件设备，然后运行另一个操作系统，比如在windows系统中运行Ubuntu系统，这样就可以运行任意Ubuntu应用了。



特性	Docker	虚拟机
性能	接近原生	性能较差
硬盘占用	一般为 MB	一般为GB
启动	秒级	分钟级



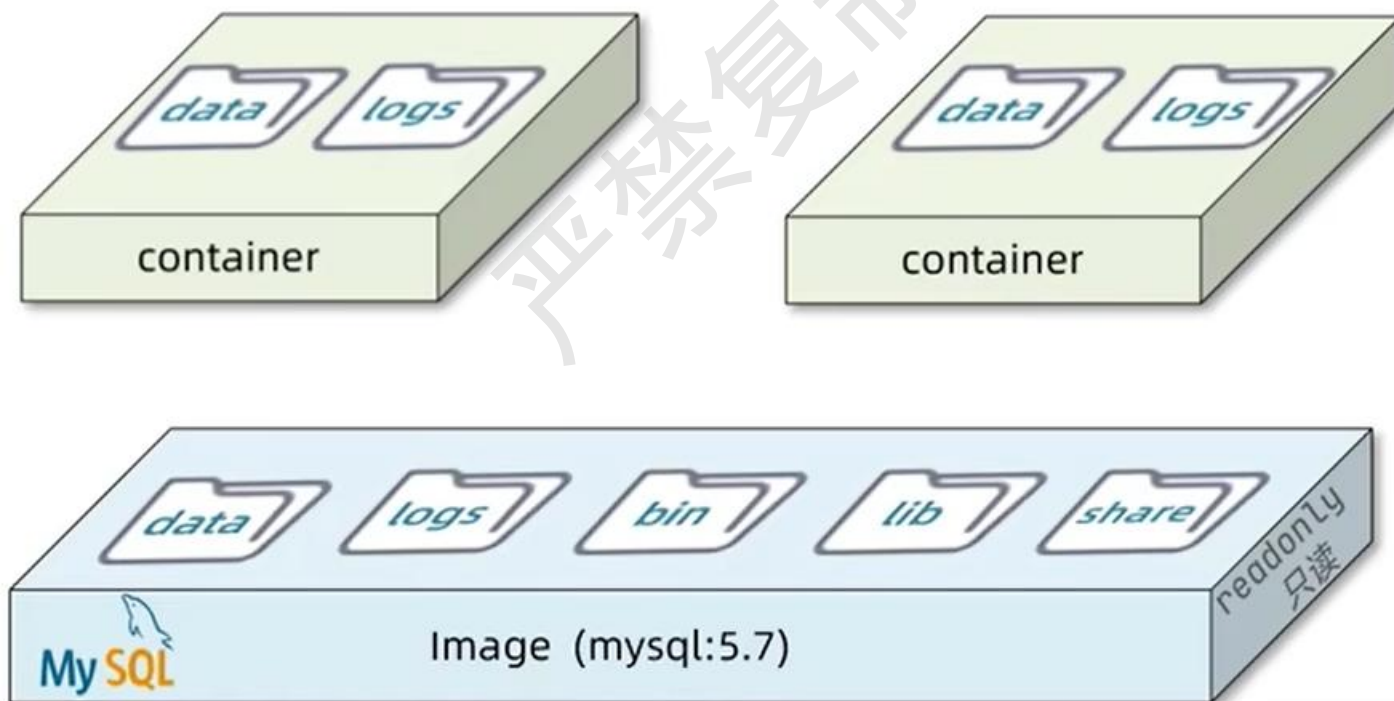
新的开始，新的起点，让我们一起为梦想而努力。

❁ 初识Docker

镜像和容器

镜像 (Image) : Docker将应用程序及其所需的依赖、函数库、环境、配置等文件打包在一起, 称为镜像。

容器 (Container) : 镜像中的应用程序运行后形成的进程就是**容器**, 只是Docker会给容器做隔离, 对外不可见。

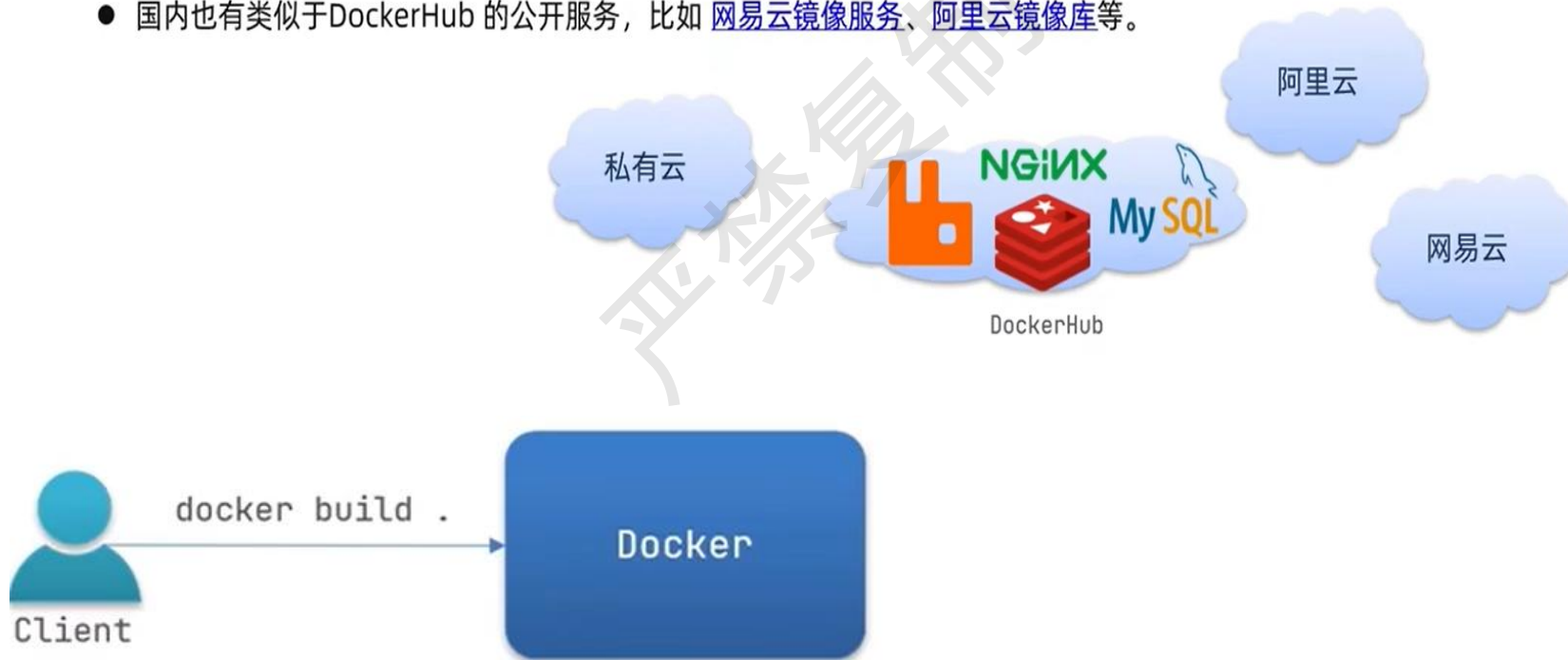


新的开始, 新的起点, 让我们一起为梦想而努力。

❁ 初识Docker

Docker和DockerHub

- DockerHub: DockerHub是一个Docker镜像的托管平台。这样的平台称为Docker Registry。
- 国内也有类似于DockerHub 的公开服务, 比如 [网易云镜像服务](#)、[阿里云镜像库](#)等。

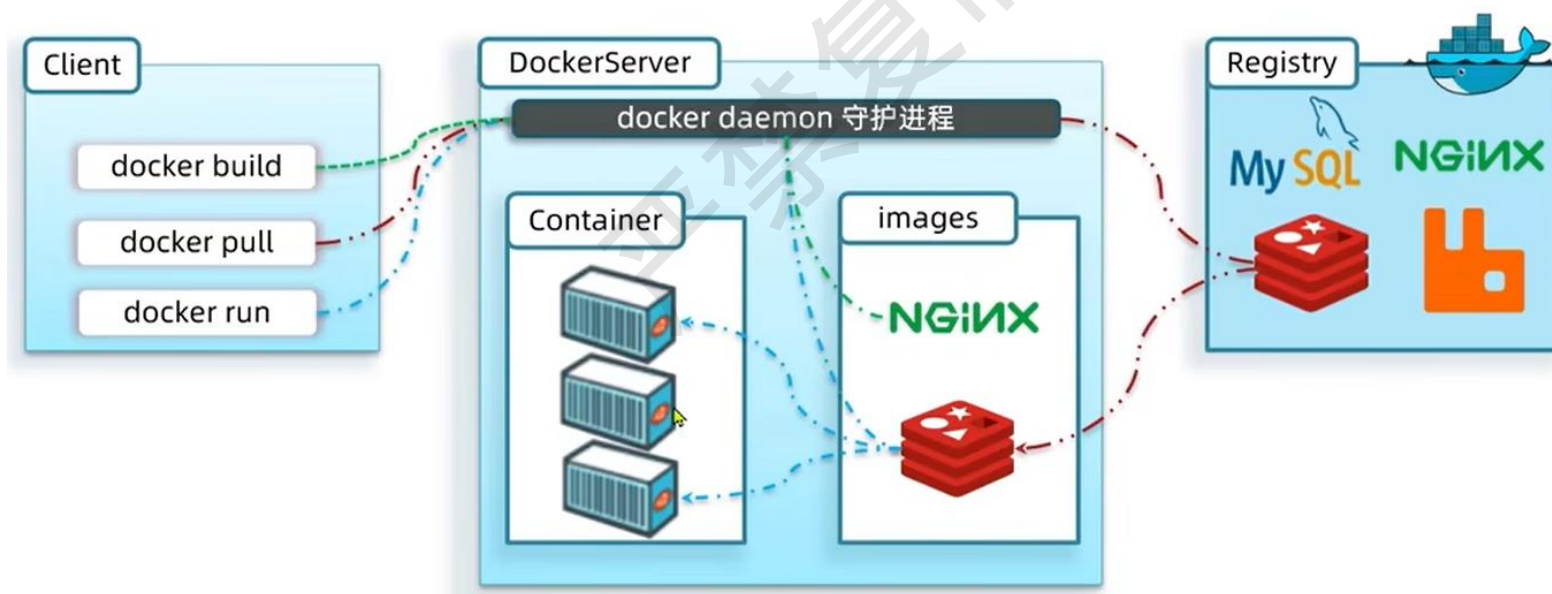


❁ 初识Docker


Docker架构

Docker是一个CS架构的程序，由两部分组成：

- ◆ 服务端(server)：Docker守护进程，负责处理Docker指令，管理镜像、容器等
- ◆ 客户端(client)：通过命令或RestAPI向Docker服务端发送指令。可以在本地或远程向服务端发送指令。



❁ 初识Docker



总结

Docker是一个快速交付应用、运行应用的技术：

1. 可以将程序及其依赖、运行环境一起打包为一个镜像，
可以迁移到任意Linux操作系统
2. 运行时利用沙箱机制形成隔离容器，各个应用互不干扰
3. 启动、移除都可以通过一行命令完成，方便快捷

镜像：

- 将应用程序及其依赖、环境、配置打包在一起

容器：

- 镜像运行起来就是容器，一个镜像可以运行多个容器

Docker结构：

- 服务端：接收命令或远程请求，操作镜像或容器
- 客户端：发送命令或者请求到Docker服务端

DockerHub：

- 一个镜像托管的服务器，类似的还有阿里云镜像服务，统称为DockerRegistry

新的开始，新的起点，让我们一起为梦想而努力。