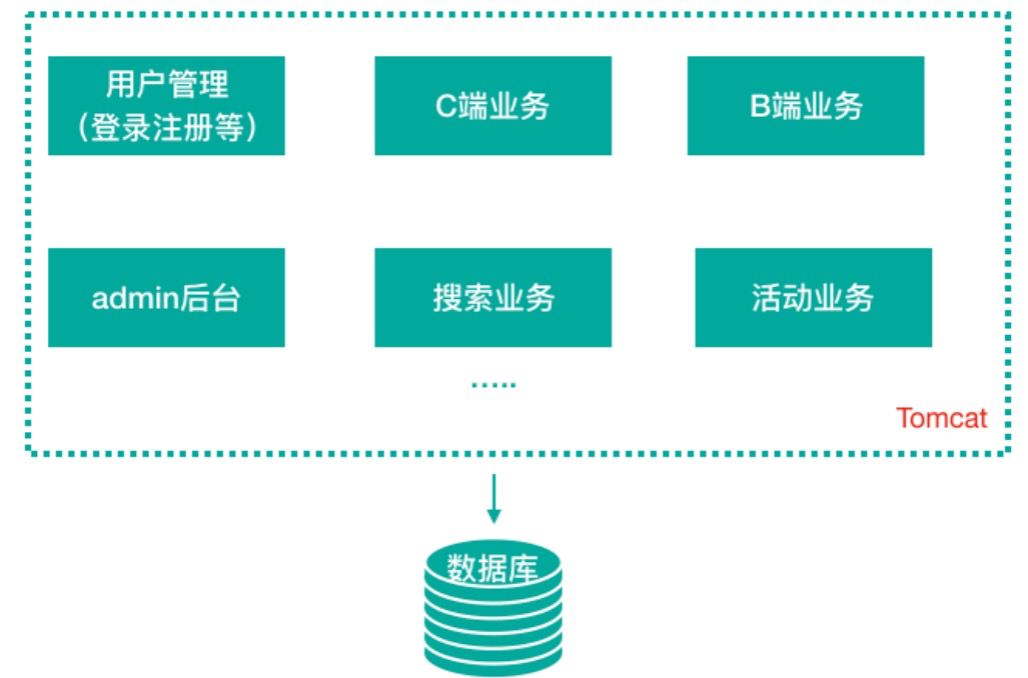
Spring Cloud 微服务

微服务架构—>Spring Cloud提供了⼀个⼀站式的微服务解决⽅案

第⼀部分：微服务架构

随着互联⽹的发展，⽤户群体逐渐扩⼤，业务系统的流量成倍增⻓，常规的单体架构已⽆法满⾜请求压⼒和业务的快速迭代，架构的变化势在必⾏。下⾯我们就以一个人才（空想/模拟）的架构演进为例，从最开始的单体架构分析，⼀步步的到现在的微服务架构。

单体



#### 优点：

1. 项⽬前期开发节奏快，团队成员少的时候能够快速迭代
2. 架构简单：MVC架构，只需要借助IDE开发、调试
3. 可易于测试：只需要通过单元测试或者浏览器完成
4. 易于部署：打包成单⼀可执⾏的jar或者打成war包放到容器内启动

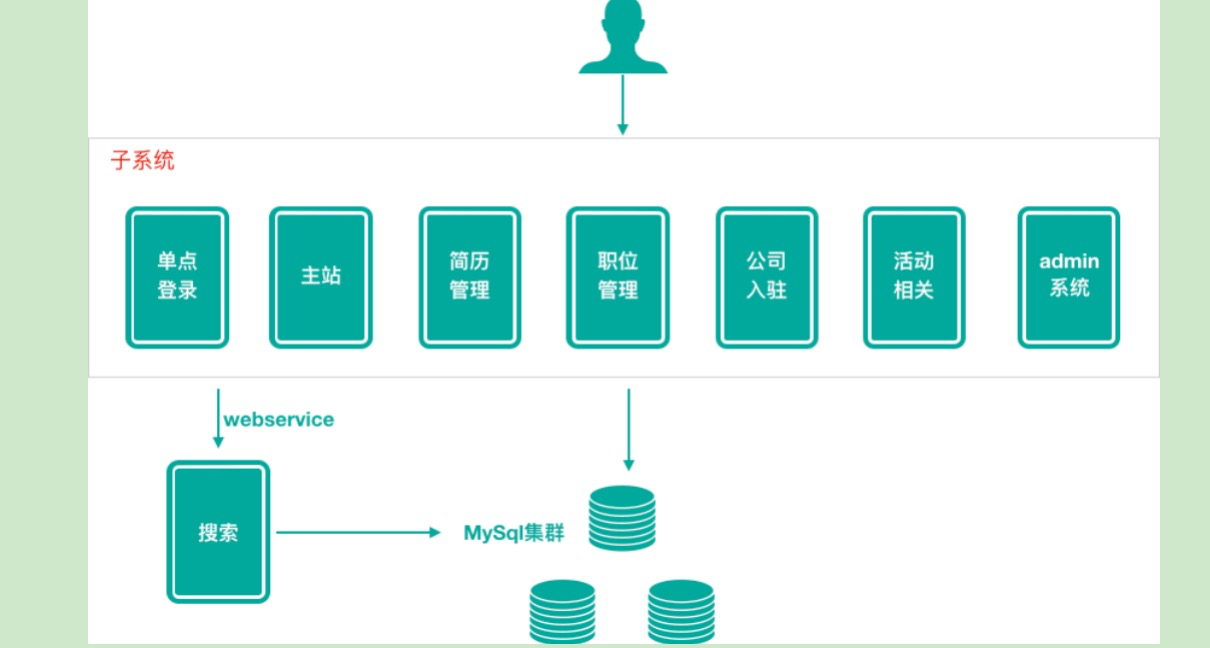
#### 缺点：

1. 随着不断的功能迭代，单个项⽬过⼤，代码杂乱，耦合严重，开发团队逐渐壮⼤以后，沟通成本变⾼， 如：代码从编译到启动耗时达到 3-5 分钟
2. 新增业务困难：在已经乱如麻的系统中增加新业务，维护旧功能，⼀脚踩进去全是不可预测 的问题。新⼈来了以后很难接⼿任务，学习成本⾼，需要⼤概很长时间才能上⼿开发
3. 核⼼业务与边缘业务混合在⼀块，出现问题互相影响，如：⼀个临时活动流量猛涨，机器负 载升⾼就会影响正常的业务服务

业务量上涨之后，单体应⽤架构进⼀步丰富变化，⽐如应⽤集群部署、使⽤Nginx进⾏负载均衡、增加缓存服务器、增加⽂件服务器、数据库集群并做读写分离等，通过以上措施增强应对⾼并发的能⼒、应 对⼀定的复杂业务场景，但依然属于单体应⽤架构。

垂直

为了避免上⾯提到的那些问题，开始做模块的垂直划分，做垂直划分的原则是基于现有的业务特性来做，核⼼⽬标第⼀个是为了业务之间互不影响，第⼆个是在研发团队的壮⼤后为了提⾼效率，减少之间的依赖。如现在的人才服务，可能不太。



#### 优点

1. 系统拆分实现了流量分担，解决了并发问题可以针对不同模块进⾏优化
2. ⽅便⽔平扩展，负载均衡，容错率提⾼
3. 系统间相互独⽴，互不影响，新的业务迭代时更加⾼效

**缺点**

1. 服务之间相互调⽤，如果某个服务的端⼝或者ip地址发⽣改变，调⽤的系统得⼿动改变

搭建集群之后，实现负载均衡⽐较复杂，如：内⽹负载，在迁移机器时会影响调⽤⽅的路 由，导致线上故障

1. 服务之间调⽤⽅式不统⼀，基于 httpclient 、 webservice ，接⼝协议不统⼀
2. 服务监控不到位：除了依靠端⼝、进程的监控，调⽤的成功率、失败率、总耗时等等这些监控指标是没有的

SOA

在做了垂直划分以后，模块随之增多，维护的成本在也变⾼，⼀些通⽤的业务和模块重复的越来越多，为了解决上⾯提到的接⼝协议不统⼀、服务⽆法监控、服务的负载均衡，引⼊了阿⾥巴巴开源的Dubbo ，⼀款⾼性能、轻量级的开源Java RPC框架，它提供了三⼤核⼼能⼒：⾯向接⼝的远程⽅法调⽤，智能容错和负载均衡，以及服务⾃动注册和发现。

SOA (Service-Oriented Architecture)，即⾯向服务的架构。根据实际业务，把系统拆分成合适的、独

⽴部署的模块，模块之间相互独⽴（通过Webservice/Dubbo等技术进⾏通信）。

优点：分布式、松耦合、扩展灵活、可重⽤。

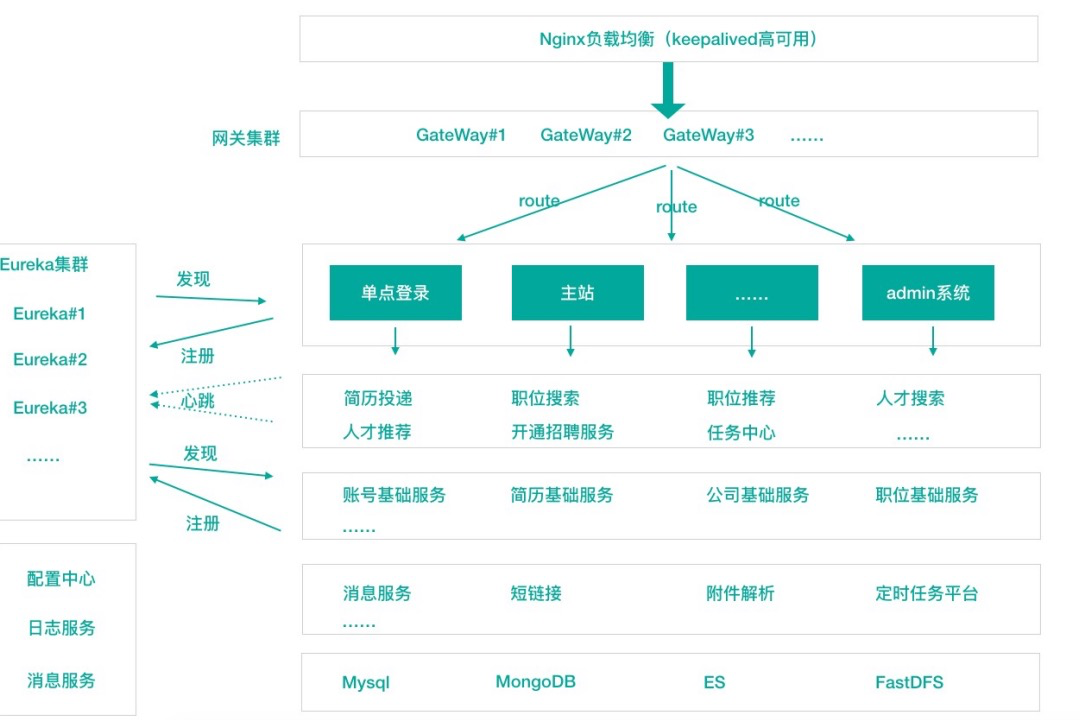
缺点：服务抽取粒度较⼤、服务调⽤⽅和提供⽅耦合度较⾼（接⼝耦合度）



微服务

微服务架构可以说是SOA架构的⼀种拓展，这种架构模式下它**拆分粒度更⼩**、服务更独⽴。把应⽤拆分成为⼀个个微⼩的服务，不同的服务可以使⽤不同的开发语⾔和存储，服务之间往往通过Restful等轻量 级通信。微服务架构关键在于**微⼩、独⽴、轻量级通信**。

微服务是在 SOA 上做的升华粒度更加细致，微服务架构强调的⼀个重点是“业务需要彻底的组件化 和服务化”



**微服务架构和SOA架构相似⼜不同**

微服务架构和SOA架构很明显的⼀个区别就是**服务拆分粒度的不同**，但是对于架构发展来说，我们所看到的SOA阶段其实服务拆分粒度相对来说已经⽐较细了，所以上述SOA到微服务，从服务拆分上来说变化并不⼤，只是引⼊了相对完整的新⼀代Spring Cloud微服务技术。⾃然，上述我们看到的都是架构演变的阶段结果，每⼀个阶段其实都经历了很多变化，服务拆分其实也是⾛过了从粗到细，并⾮绝对的⼀步到位。

举个案例来说明SOA和微服务拆分粒度不同

我们在SOA架构的初期，“简历投递模块”和“⼈才搜索模块”都有简历内容展示的需求，只不过说可能略有区别，⼀开始在两个模块中各维护了⼀套简历查询和展示的代码；后期我们将服务更细粒度拆分，拆分出简历基础服务，那么不同模块调⽤这个基础服务即可。

**微服务架构体现的思想及优缺点**

微服务架构设计的核⼼思想就是**“微”**，拆分的粒度相对⽐较⼩，这样的话单⼀职责、开发的耦合度就会降低、微⼩的功能可以独⽴部署扩展、灵活性强，升级改造影响范围⼩。

单体应⽤（1.7—>1.8）

A(升级JDK) B C D E .....

**微服务架构的优点**: 微服务架构和微服务

微服务很⼩，便于特定业务功能的聚焦 A B C D

微服务很⼩，每个微服务都可以被⼀个⼩团队单独实施（开发、测试、部署上线、运维），团队合 作⼀定程度解耦，便于实施敏捷开发

微服务很⼩，便于重⽤和模块之间的组装

微服务很独⽴，那么不同的微服务可以使⽤不同的语⾔开发，松耦合

微服务架构下，我们更容易引⼊新技术

微服务架构下，我们可以更好的实现DevOps开发运维⼀体化；

#### 微服务架构的缺点

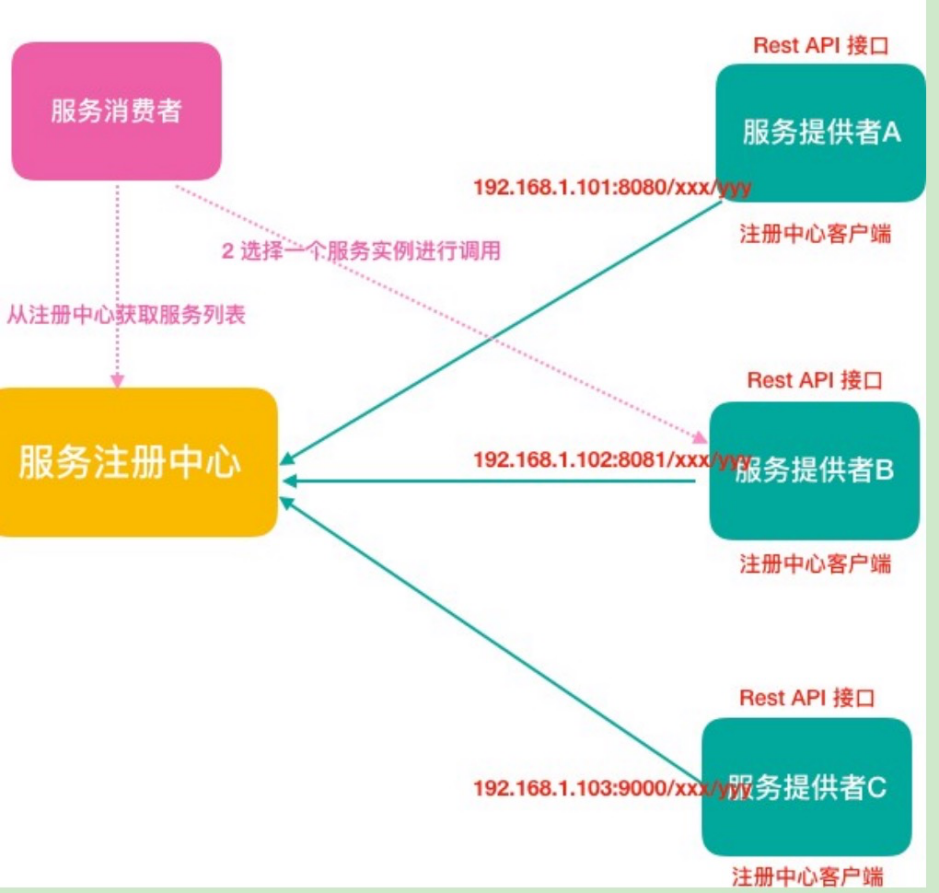
微服务架构下，分布式复杂难以管理，当服务数量增加，管理将越加复杂； 微服务架构下，分布式链路跟踪难等；

## 微服务架构中的⼀些概念

#### 服务注册与服务发现

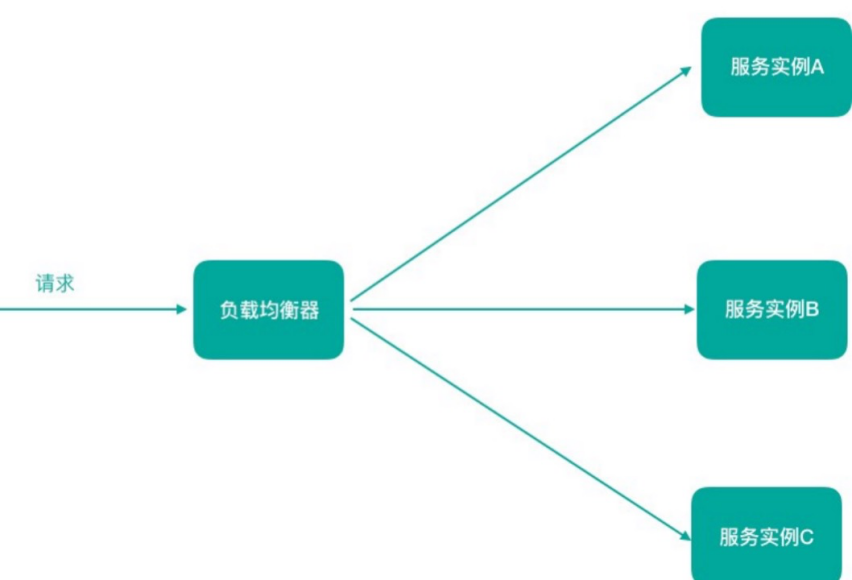
**服务注册：**服务提供者将所提供服务的信息（服务器IP和端⼝、服务访问协议等）注册/登记到注册中⼼

**服务发现：**服务消费者能够从注册中⼼获取到较为实时的服务列表，然后根究⼀定的策略选择⼀个服务 访问



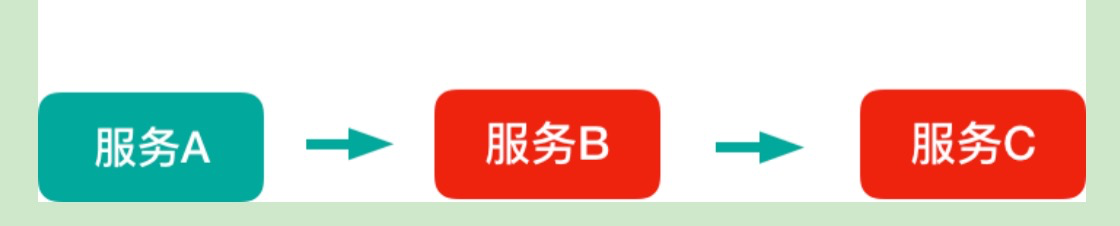
#### 负载均衡

负载均衡即将请求压⼒分配到多个服务器（应⽤服务器、数据库服务器等），以此来提⾼服务的性 能、可靠性



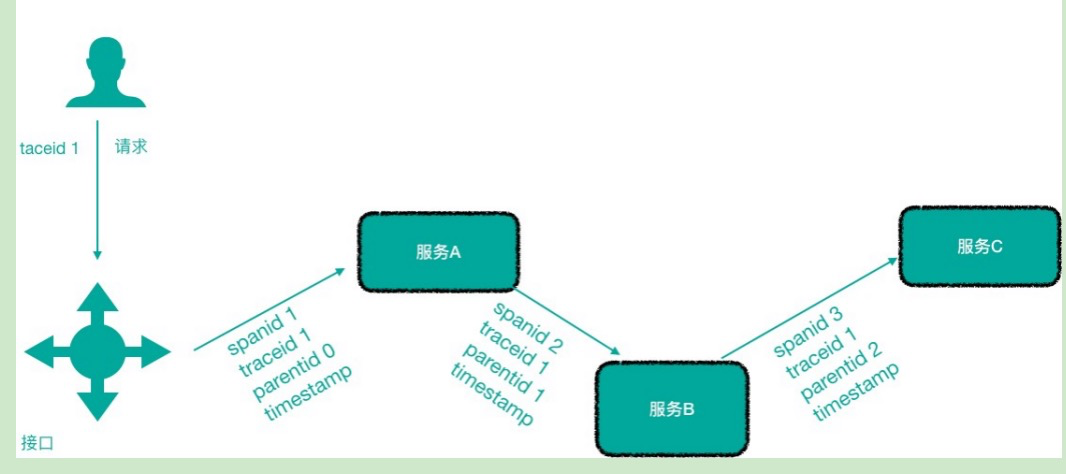
#### 熔断

熔断即断路保护。微服务架构中，如果下游服务因访问压⼒过⼤⽽响应变慢或失败，上游服务为了保护系统整体可⽤性，可以暂时切断对下游服务的调⽤。这种牺牲局部，保全整体的措施就叫做熔断。



#### 链路追踪

微服务架构越发流⾏，⼀个项⽬往往拆分成很多个服务，那么⼀次请求就需要涉及到很多个服务。不同的微服务可能是由不同的团队开发、可能使⽤不同的编程语⾔实现、整个项⽬也有可能部署在了很多服务器上（甚⾄百台、千台）横跨多个不同的数据中⼼。所谓链路追踪，就是对⼀次请求涉及的很多个服务链路进⾏⽇志记录、性能监控



#### API ⽹关

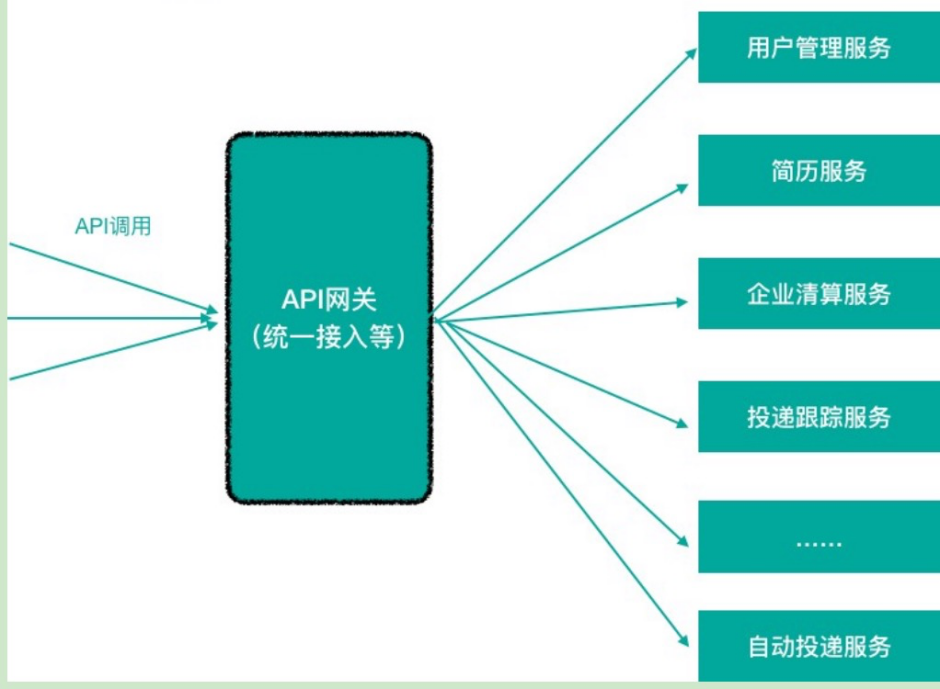
微服务架构下，不同的微服务往往会有不同的访问地址，客户端可能需要调⽤多个服务的接⼝才能完成

⼀个业务需求，如果让客户端直接与各个微服务通信可能出现：

1. 客户端需要调⽤不同的url地址，增加了维护调⽤难度
2. 在⼀定的场景下，也存在跨域请求的问题（前后端分离就会碰到跨域问题，原本我们在后端采⽤Cors就能解决，现在利⽤⽹关，那么就放在⽹关这层做好了）
3. 每个微服务都需要进⾏单独的身份认证

那么，API⽹关就可以较好的统⼀处理上述问题，API请求调⽤统⼀接⼊API⽹关层，由⽹关转发请求。 API⽹关更专注在安全、路由、流量等问题的处理上（微服务团队专注于处理业务逻辑即可），它的功 能⽐如

1. 统⼀接⼊（路由）
2. 安全防护（统⼀鉴权，负责⽹关访问身份认证验证，与“访问认证中⼼”通信，实际认证业务逻辑交移 “访问认证中⼼”处理）
3. ⿊⽩名单（实现通过IP地址控制禁⽌访问⽹关功能，控制访问）
4. 协议适配（实现通信协议校验、适配转换的功能）
5. 流量管控（限流）
6. ⻓短链接⽀持
7. 容错能⼒（负载均衡）



第⼆部分：SpringCloud概述

**是什么？**

[百度百科]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的开发⻛格做到⼀键启动和部署。Spring Cloud并没有重复制造轮⼦，它只是将⽬前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⻛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包。

Spring Cloud是⼀系列框架的有序集合（Spring Cloud是⼀个规范）

开发服务发现注册、配置中⼼、消息总线、负载均衡、断路器、数据监控等利⽤Spring Boot的开发便利性简化了微服务架构的开发（⾃动装配）

**这⾥，我们需要注意**，Spring Cloud其实是⼀套规范，是⼀套⽤于构建微服务架构的规范，⽽不是⼀个可以拿来即⽤的框架（所谓规范就是应该有哪些功能组件，然后组件之间怎么配合，共同完成什么事情）。在这个规范之下第三⽅的Netﬂix公司开发了⼀些组件、Spring官⽅开发了⼀些框架/组件，包括第三⽅的阿⾥巴巴开发了⼀套框架/组件集合Spring Cloud Alibaba，这些才是Spring Cloud规范的实现。

Netﬂix搞了⼀套 简称SCN

Spring Cloud 吸收了Netﬂix公司的产品基础之上⾃⼰也搞了⼏个组件

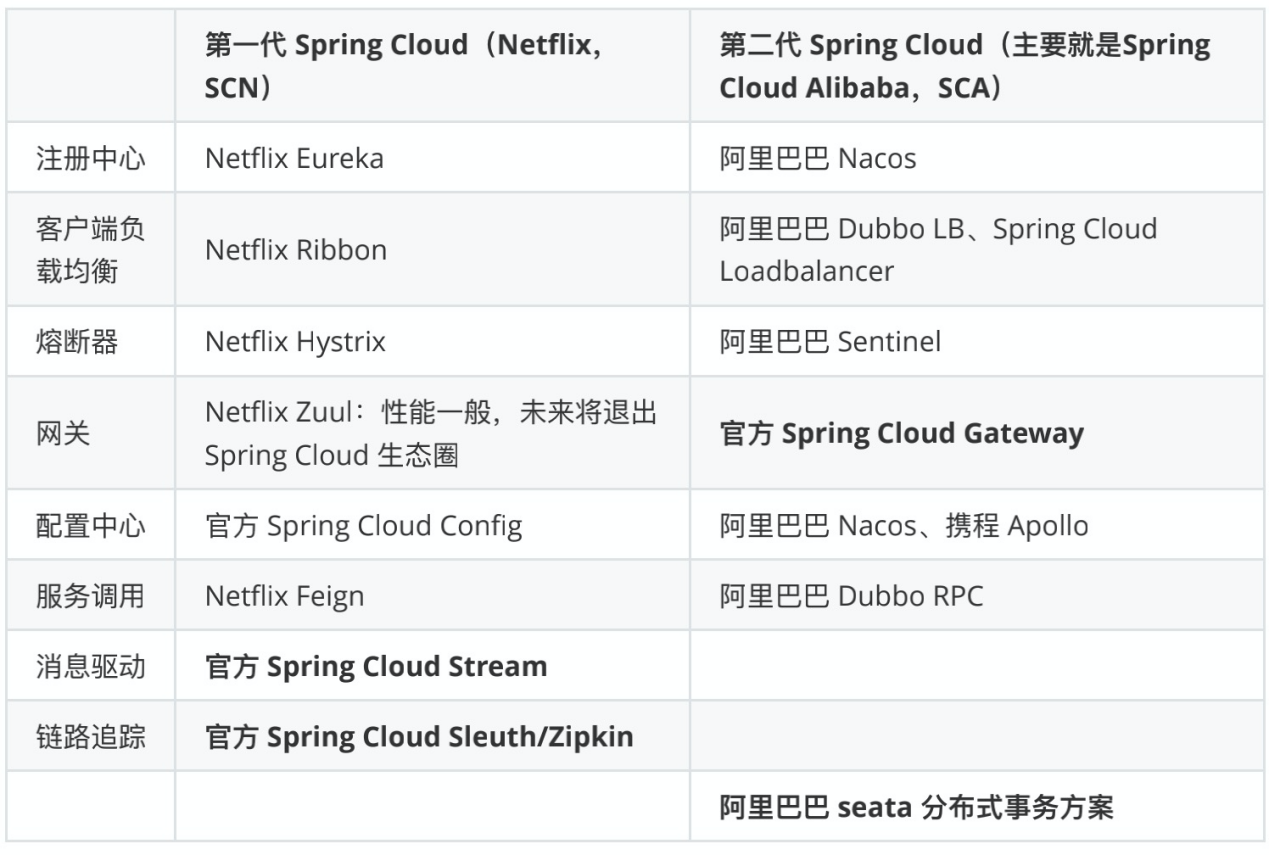
阿⾥巴巴在之前的基础上搞出了⼀堆微服务组件,Spring Cloud Alibaba（SCA）

**解决什么问题**

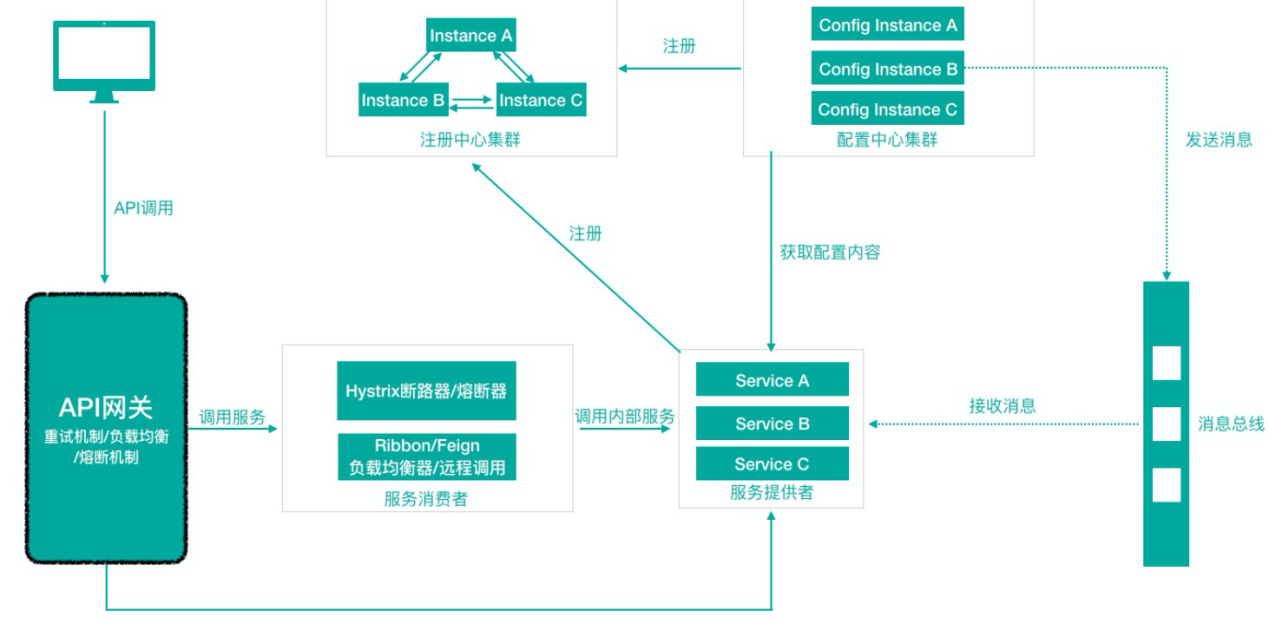
Spring Cloud 规范及实现意图要解决的问题其实就是微服务架构实施过程中存在的⼀些问题，⽐如微服务架构中的服务注册发现问题、⽹络问题（⽐如熔断场景）、统⼀认证安全授权问题、负载均衡问题、链路追踪等问题。

**架构及其核⼼组件（框架）**

如前所述，Spring Cloud是⼀个微服务相关规范，这个规范意图为搭建微服务架构提供⼀站式服务，采⽤组件（框架）化机制定义⼀系列组件，各类组件针对性的处理微服务中的特定问题，这些组件共同来构成Spring Cloud微服务技术栈



协同工作机制：



Spring Cloud中的各组件协同⼯作，才能够⽀持⼀个完整的微服务架构。⽐如

注册中⼼负责服务的注册与发现，很好将各服务连接起来

API⽹关负责转发所有外来的请求

断路器负责监控服务之间的调⽤情况，连续多次失败进⾏熔断保护。

配置中⼼提供了统⼀的配置信息管理服务,可以实时的通知各个服务获取最新的配置信息

**Spring Cloud与Dubbo的对⽐**

Dubbo是阿⾥巴巴公司开源的⼀个⾼性能优秀的服务框架，基于RPC调⽤，对于⽬前使⽤率较⾼的Spring Cloud Netflflix来说，它是基于HTTP的，所以效率上没有Dubbo⾼，但问题在于Dubbo体系的组件不全，不能够提供⼀站式解决⽅案，⽐如服务注册与发现需要借助于Zookeeper等实现，⽽Spring Cloud Netflflix则是真正的提供了⼀站式服务化解决⽅案，且有Spring⼤家族背景。

前些年，Dubbo使⽤率⾼于SpringCloud，但⽬前Spring Cloud在服务化/微服务解决⽅案中已经有了⾮常好的发展趋势。

**Spring Cloud与Spring Boot**

Spring Cloud 只是利⽤了Spring Boot 的特点，让我们能够快速的实现微服务组件开发，否则不使⽤ Spring Boot的话，我们在使⽤Spring Cloud时，每⼀个组件的相关Jar包都需要我们⾃⼰导⼊配置以及需要开发⼈员考虑兼容性等各种情况。所以Spring Boot是我们快速把Spring Cloud微服务技术应⽤起来的⼀种⽅式。

第三部分代码示例：代码分享