阿里云开发者社区
ALIBABA CLOUD DEVELOPER COMMUNITY

Spring Cloud 微服务架构设计与实战

阿里云开发者学堂推荐配套教材

作者：侠客

揭秘阿里等一线互联网公司微服务架构体系
实战服务治理、熔断限流、链路追踪、安全监控等核心问题
涵盖 Spring Cloud Alibaba 阿里巴巴体系

藏经阁

阿里云开发者电子书系列

扫一扫

免费领取同步课程

钉钉扫一扫

进入官方答疑群

开发者学院【Alibaba Java 技术图谱】

更多好课免费学

阿里云开发者“藏经阁”

海量电子书免费下载

书籍简介

Java Spring Cloud 是全球范围内最成熟、最完善、最流行的微服务架构方案体系。被众多的互联网大公司采用，包括阿里巴巴、腾讯、支付宝、网易、IBM、谷歌、京东、百度、滴滴等。

本次课程涵盖最新版本的 Spring Cloud 微服务架构体系， 微服务架构模式、算法与典型场景、框架、优缺点，Spring Cloud 2020 的重大变化、扩展 Netflix、Spring Cloud Alibaba 阿里巴巴体系，Dubbo 等架构选型对比，淘宝微服务架构案例。

重点讲解：服务治理、注册发现、熔断限流、网关代理、链路追踪、安全监控等核心问题，循序渐进，概念为辅、实战为主，涵盖经典面试题。让您成为合格的微服务架构师。

目录

1.0 《Java Spring Cloud 微服务实战》大纲 7

1.1 什么是微服务架构 Microservice 12

1.2 微服务架构 Microservice 的优缺点 21

1.3 微服务架构 Microservice 的典型应用场景 27

1.4 微服务架构 Microservice 的淘宝改造案例 34

1.5 微服务架构 Microservice 的经典协议 40

1.6 微服务架构 Microservice 的开发框架 45

1.7 微服务架构 Microservice 的设计策略 52

1.8 微服务架构 Microservice 的经典设计模式 58

1.9 Java Spring Cloud 微服务开发环境配置 62

1.10 Spring Cloud 2020 重大变化与选型提示 69

2.1 为什么选择 Java Spring Cloud 微服务架构 74

2.2 Spring Cloud 微服务注册与发现 Eureka 81

2.3 Spring Cloud 微服务 API 实战开发并注册到 Eureka 87

2.4 Spring Cloud 客户端 Feign 调用微服务 API 93

2.5 Spring Cloud 微服务 Ribbon 负载均衡算法 101

2.6 Spring Cloud 微服务 API 的监控 Hystrix 107

2.7 Spring Cloud 微服务 API 的 Hystrix 熔断限流降级 116

2.8 Spring Cloud 微服务网关代理 Zuul 122

2.9 Spring Cloud 微服务的身份验证与安全机制 131

2.10 Spring Cloud 微服务集群 Monitor 监控中心 137

3.1 Spring CloudAlibaba 微服务体系 143

3.2 Spring Cloud AlibabaNacos 经典注册中心对比 1499

3.3 Spring Cloud 开发微服务 API 注册到 Nacos 161

3.4 Spring Cloud 客户端 Feign 集成 Nacos 中心 174

3.5 Spring Cloud 使用 Nacos 作为微服务统一配置中心 181

3.6 Spring Cloud 实战集成 Sentinel 熔断限流 193

3.7 Spring Cloud 网关 Zuul 集成 Nacos 注册中心 200

3.8 Spring Cloud Alibaba Seata 分布式事务 204

3.9 Spring Cloud Gateway 微服务新网关实战 210

3.10 Spring Cloud Gateway 实战接入 Nacos 服务 216

1.0 《Java Spring Cloud 微服务实战》大纲

内容简介：

一、微服务架构课程大纲

二、Dubbo 集成 Nacos 注册中心实战

目前微服务架构是非常的火爆，各个大型互联网公司都在使用微服务架构，目前是以 Java Spring Cloud 的微服务架构为主。本次课程是贴近目前以阿里等 BAT 为首的微服务架构的技术方案。

一、微服务架构课程大纲

1. 微服务架构理论知识

第一阶段会讲解课程最重要的一个部分—微服务架构体系的基础理论知识。介绍微服务架构的理论、分布式的协议、微服务架构技术方案的选型以及微服务架构的拆分的原则。之后会讲解目前几个典型互联公司的案例。这里是以淘宝的微服务架构作为其中一个重要的一个知识点给大家进行分享，作为供大家学习的一个参考。

2. 微服务实战开发

第二阶段以现在最成熟的 Spring Cloud 微服务架构体系作为实战的主要落地框架。希望大家不仅能够掌握扎实的微服务架构理论知识，也能够进行实战的架构设计和开发工作。希望大家成为理论和实战能力结合的技术专家或者架构师。

3. 阿里巴巴微服务

第三个阶段会给大家介绍一下阿里开源的几个经典的微服务架构方案。

在微服务架构体系概念这一板块给大家介绍比较重要的几个知识点，包括微服务架构优缺点、经典的设计原则、微服架构领域比较经典的设计模式以及淘宝微服务架构设计案例。

在 Spring Cloud 微服务架构实战阶段，会给大家系统的讲解整个微服务架构的治理知识、注册和发现、在高频化的情况下如何做到高可用、限流、熔断、网管代理相关内容以及微服务领域另外一个重要功能-安全。在这里面会涉及到一种必须使用的技术叫令牌机制。

最后作为扩展阶段的知识，以阿里巴巴开源的微服务框架为主重点介绍阿里开源的 Linux 平台以及 Sentinel -实现熔断、限流非常重要的一个框架。

二、微服务学习路线图

这个部分给大家讲一下整个微服务架构和学习路线图。参考着阿里的 P 层级作为参考，作为微服务架构设计能力对架构师的岗位要求。

微服务架构属于分布式架构，它在早期阶段也有很多重要的框架，比较经典的例如 Dubbo。Dubbo 是阿里巴巴开源非常重要的一个服务治理的框架，它也涉及到很多设计模式，而且底层也有很多优秀的设计思想。 Dubbo 也在向微服务架构靠拢。我们课程是以 Spring Cloud 微服务架构为主。Spring Cloud 是出现的最早也是最完善的一套架构体系。

目前还有另外一个比较重要的技术架构知识叫中台架构。目前有很多大型的互联网公司也在落地，但这里面涉及的技术不仅仅包括微服务架构。微服务架构可能会成为中台架构落地重要的一个技术点。

三、阿里最新 Java 课程

在国内阿里巴巴对 Java 技术的发展贡献是非常大的。阿里巴巴也是国内最大的开源框架公司，科研项目贡献最多的中国互联网公司。Java 早期缺少典型的解决方案的时候，阿里巴巴都在公司内部进行大规模的实践，包括淘宝以及支付宝等等这些典型的互联网项目。

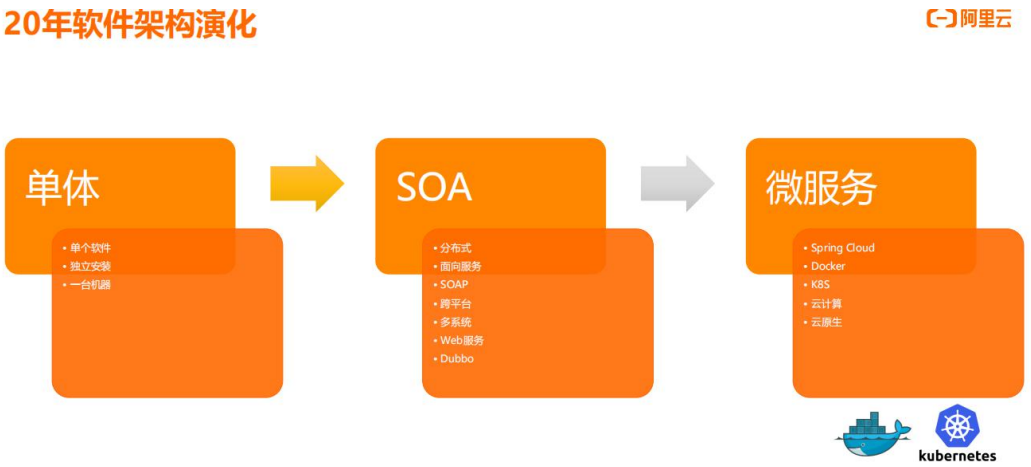
1.1 什么是微服务架构 Microservice

内容简介：

一、什么是微服务架构 Microservice?

二、微服务的发展历史

微服务架构目前非常火爆，在架构领域属于当红的明星架构，那么什么是微服务架构?



微服架构是在移动互联网时代崛起的新架构模式。现在架构模式一般称为Microservice，本身叫微服务。现在的互联网公司，尤其是国内阿里、腾讯、微博、京东、拼多多等，严格来说都是微服务架构。

回顾历史，这么多年架构的发展最具有代表性是淘宝和腾讯，但是腾讯更像 QQ 与微信的架构，后台主要以 C++为主，是典型的分布式架构软件，直播类、社交类的抖音也是一个典型的微服架构。

起步较早的淘宝经历过三大阶段，单体到 SOA，再到微服务。微服务架构是 2000 年到 2010 年之间非常火爆的架构，尤其是一些大型的银行项目。同时，它也是分布式架构非常重要的阶段，是一个代表性的架构。

当年无论是 IBM，还是各大银行的架构师，在技术峰会上基本上讨论的都是 SOA 相关的概念。微服架构作为现阶段比较火爆的架构，是在其他的架构基础上演化而来，诞生于分布式 SOA 的技术架构，淘宝是典型的案例。

淘宝早期是单体的，后面开始往分布式，转 Java 去 Oracle，并开始用 Microservice，包括引入其他的分布式解决方案，逐步构造今天的微服务架构。后续诞生的电商公司，大部分都借鉴了淘宝的架构发展历史经验，例如京东在 2010 年开始转 Java，也有类似的微服务框架和解决方案。

为什么国内大公司都是通过 Java 语言来进行编写?

本身编程语言没有优劣之分，对于项目的开发人员、工程师、架构师而言，解决问题，帮公司创造价值，在技术选型上满足公司不同阶段不同业务的需求，这是基本出发点。合格的架构师在技术选型时，需要考虑方案落地性，招人成本，组建团队成本以及后续开发过程中对应的解决方案。

目前来看，微服务架构是 Spring Cloud 出现得最早，参与公司最多，开源社区最活跃最成熟的微服务架构解决方案，并且还在不断的迭代演化。

基于 Java、Mysql、Linux 等，阿里不断摸索分布式架构的解决方案，并把积累的经典解决方案框架都开源供其他公司借鉴学习。

一、什么是微服务架构 Microservice?

1. 微服务的定义

1）微服务架构模式

2）Microservice

3）Dr. Peter Rodgers2005 Cloud Computing Expo 技术大会上提出概念

4）2007, Netflix 开始向微服务架构师进发

5）并最终开源了自己研发的 Java 微服务框架

6）开源社区命名为 Spring Cloud

7）微服务是一种新型的 软件架构风格

8）把单个巨型服务应用，分解为多个独立的、微小的服务程序

9）单独部署

10）单独伸缩

11）去中心化：数据中心、管理中心

12）敏捷性、灵活性、需求变化，更加高效的软件架构模式

微服务架构诞生在 SOS，最早的时候并不叫微服务架构，而是叫 Micro Web Service，指微小的 web service 程序，使用 Java 写了一套轻量级的微服务架构的解决方案，是移动互联网时代很重要的一个标志，服务端的接口的应用程序的开始轻量计划。

目前，微服务框架以 recipe 风格为主的一个很重要的原因，后续无论是去中心化、敏捷开发、单独部署等都是随着程序的微服务化快速开发与部署，逐步诞生了一系列的经典的工具，辅助用户提升业务应用的开发部署模式与效率。

2. 微小的服务

1）微服务架构：将单个应用拆分成多个独立的、微小的服务。

2）每个小服务程序运行在独立的进程中。

3）服务与服务之间通过轻量协议通信。

4）通信机制互相协作、互相配合,从而为终端用户提供业务价值。

5）每个小服务，可以采用不同的语言、框架、工具 独立开发、测试、部署、运维。

6）微服务：独立的小服务。

Microservice 的简称过来就是微服务，实际指微小的服务程序，之前各个服务程序都在一个项目中，现在拆开方便进行各个功能单独迭代升级。移动互联网中微服务迭代的非常快，无论是淘宝的支付宝，还是微信、微博，其他的 APP 都是微服务加工。设置手机默认浏览器也是，子功能模块它其实都在单独的进行功能迭代的，尤其是国内定制的浏览器，360 浏览器，腾讯浏览器，百度浏览器其实里面在各种功能基本上也都单独进行迭代的。杀毒软件也有各种不同的背后通信数据采集的机制。

3. Microservices

In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

-- James Lewis and Martin Fowler

James Lewis and Martin Fowler

除了麦飞公司是微服务架构的先驱公司以外，还有几个重要的技术专家，他们不断宣传微服务架构，他们工作于 thoughtworks 软件咨询架构，中国也有分公司。从他们做的架构可以看到，咨询师使用微服务架构，微服务架构已成为时代的潮流。

4. 微服务

简而言之，微服务架构风格是一种将单个应用程序开发为一套小服务程序的方法，每个小服务都在自己的进程中运行，并使用轻量级协议（通常是 HTTP 协议）进行通信。

这些服务围绕业务功能构建，可通过全自动部署机制独立部署。这些服务很少使用中心化管理模式，可以用不同的编程语言开发，也可能使用不同的数据存储技术。

-- James Lewis 与 Martin Fowler

现在看到关于微服务架构的一些书籍或公开文章里面，基本上认为现在微服务架构，通信的接口都是 Rest API，以 HTTP+Jason 格式进行交互。相比传统的 rpc、dubbo、web service 重量级的框架来说，有些业务场景需要更高性能的通信协议，后续会看到一些新版本的微服务框架在不断迭代和进化。

5. Wikipedia 定义

1）In computing, microservices is a software architecture style in which complex applications are composed of small, independent processes communi cating with each other using language-agnostic APIs.

2）在计算机领域中，微服务是一种软件架构风格，复杂的应用程序由语言无关的 API、相互通信的小型独立服务进程组成。

3) These services are small building blocks, highly decoupled and focused on doing a small task, facilitating a modular approach to system-building.

4）这些服务是小型构建模块，高度解耦，专注于完成一项小任务，是一种便捷的模块化系统构建方法。

在协议这个层次上进行了迭代改造，微服务架构并不是只一种架构，它是复杂架构的一个代表，里面涉及到很多种设计模式与框架。

二、微服务的发展历史

1）Dr. Peter Rodgers introduced the term “Micro-Web-Services”during a presentation at the Web Services Edge conference in 2005. On slide #4

2）In 2007, Netflix started on a long road towards fully operating in thecloud.

3）A workshop of software architects held near Venice in May 2011used the term “microservice” 。

4) All of these Netflix libraries and systems were open-sourced around2012

5）In May 2012, the same group decided on“microservices” as the most appropriate name。

6）James Lewis presented some of those ideas as a case study in March2012 at 33rd Degree in Kraków in Microservices - Java, the UnixWay, as did Fred George about the same time.

7）Adrian Cockcroft at Netflix, describing this approach as "fine grained SOA“

8）2014 年 4 月 25 号，Martin Fowler 发表 Microservices a definition of this new architectural term

9）In 2015, Spring Cloud Netflix reached 1.0.

10）2018 年 10 月 31 日 Spring Cloud Alibaba 宣布正式开源，提交给 Spring 方孵化器

微服务并非全新的架构，回顾计算机历史发展史，会发现基本上无论算法、框架还是理论知识，都有一个明显的时间线或者依赖关系。后续出现的框架一定比前面的框架设计的更好，因为它是借鉴或者总结前面经典的设计思想模式，然后进行改进，代表性公司如麦飞，内部实践并且把框架全部贡献给社区，做出了很大贡献。

Netflix 后续将微服务架构的解决方案全部开源，是 Spring Cloud 最早的一批微服务框架，目前社区也在用，阿里也把自己的方案打包进行了开源。

1. 微服务架构的发展历史

1）Dr. Peter Rodgers 在 2005 年的 Web Services Edge conference 大会上演讲，PPT 第 4 页引入了“Micro-Web-Services 一词

2）2007 年，Netflix 开始走向全面拆分巨型 SOA 服务的漫长道路。

3）2011 年 5 月在威尼斯附近举办的软件架构师研讨会使用了“微服务”“microservices”一词。

4）2012 Netflix 开源了所有的微服务相关工具框架的源码

5）2012 年 5 月，同一个组织宣布“microservices 是最恰当的名词。

6）James Lewis 在 2012 年 4 月 第 33 届 Degree in Kraków in Microservices-Java, the Unix Way,大会上案例研究分享时提出了类似的想法， Fred George 也大约在这个时间提出了类似观点.

7）Netflix 公司的 Adrian Cockcroft，称为：“fine grained SOA“

8）2014 年 4 月 25 号，Martin Fowler 发表 Microservices a definition of this new architectural term

9）2015，Spring Cloud Netflix 正式发布 1.0 版本. 成为微服务架构的首选

10）2018 年 10 月 31 日 Spring Cloud Alibaba 宣布正式开源

2. 微服务架构先驱

Martin Fowler（上图右 1），写了很多经典的书：企业应用架构模式重构等、敏捷宣言 17 发起人之一，目前一直活跃软件架构领域。他是在技术大会上公开进行与演讲、推广，为领域做出了很大贡献

James Lewis 詹姆斯里维斯（上图下方）微服务架构发明人，在 Thoughtworks 主要负责服务架构宣传与咨询。

Adrian Cockcroft 阿德里安（上图左一）为麦飞的技术总监，将理论和实践结合的先驱，基于 Java 框架做出了一套自研微服务解决方案，并且开源给社区。

1.2 微服务架构 Microservice 的优缺点

内容简介：

一、微服务 Microservice 优点

二、微服务 Microservice 缺点

一、微服务 Microservice 优点

1. 微服务特点一：快速响应需求变化

微服务架构诞生在 SOA 时代，在移动互联网时代蓬勃发展并崛起，从早期的互联网公司开始，快速过渡到现在的移动互联网公司，都在大量使用微服务架构，包括大家熟悉的淘宝、微博、微信、抖音等平台，都是很典型的代表。微服务架构很重要的特点就是：**快速响应需求变化**，业务迭代非常快，每月甚至每周都会有大量的改版信息。

之前在采用单体巨型非微服务架构有个问题，系统里面的业务模块非常多，大家一起发布、修改、编译很难进行协调，可能是几千人的开发团队，很难做到敏捷开发、发布、上线。

使用微信、微博、淘宝、抖音超过 5 年会感受到，迭代非常快，而且经常上线新功能，比如支付宝之前只有支付担保交易，现在可以在上面交水电费、做地铁、城市健康码功能等。淘宝的各种新功能，比如生鲜、直播等。抖音之前只做短视频，现在也开始

做电商，可以评论、加好友等等。

总的来说，之前单体巨型架构模式，已经无法适应快速变化的业务发展需求。快速响应需求变化是微服务架构的重要特点。

微服务：船小好调头

微服务本质上是小微程序，相比较来说，很重要的特点是拆分概念。微服务首先是拆分，把大的拆成小的，把整体拆成部分。每个部分单独开发迭代，是很重要的优势，在中国书画里面叫船小好调头。

中国是公有制为主体，私有制作有效补充的经济体制结构。而且私有企业民营企业，要求船小好调头，能够更灵活的根据市场需求调整经营策略。比如现在的全民电商、全民直播，是商业的微型化、敏捷化的表现形式，我们的微服务是一样的道理，各个领域有很多概念相通。

微服务优点：

微服务优点是拆完以后更灵活，各个子系统可以**独立开发**、**独立测试**、**独立部署**、**独立进程**，最后在集成。

比如账号系统比较稳定，基本上不用改，前期有三到五个成员开发，后期只需要一个人维护。后面有新业务，比如淘宝直播、菜鸟快递、余额宝项目上线，再成立新的团队，进行快速开发。现在很多项目都是前期只有一个小组，后面再组建团队上线很多新的功能，比如微信，是从腾讯内部孵化出来，前期只成立了很小的项目组，后期做成上万人的公司。支付宝也是淘宝内部孵化的项目。这种公司成立以后，又开始孵化出更多的业务部门。

独立开发拆分以后自主性更强了，独立开发、独立测试、独立部署、独立进程，是微服务快速响应业务需求变化的重要特点。

2. 特点二：敏捷开发、敏捷运维 DevOps

早期 20 年前提出敏捷开发，在微服务时代依然适用，本质上就是“快”，提升开发运维的效率，快速响应用户的需求。

传统软件公司为什么不适合大量推广微服务，比如工厂的管理软件，用 VB 开发，用 wondows xp 、wondows 7 都可以直接解决问题，20 年不变，这个项目不用敏捷开发也行，开发完成后用两个成员维护着。业务非常稳定，没有发展，或只在某个阶段平稳发展，比如银行系统，国内四大银行，早期的 Java 系统，基本上都是 oracle 或

DB2 框架开发，基本上都不会改，因为很多代码封装在存储过程中，改的话容易出问题。新项目可能开始用微服务架构，拆分出独立的数据库，独立架构。

微服务架构的优点：

微服务架构的优点，本质上就是拆完以后更好开发。通常是 HTTP 协议，能信使用语言中立协议等，优点和缺点是相对某个技术架构的，不是绝对的。

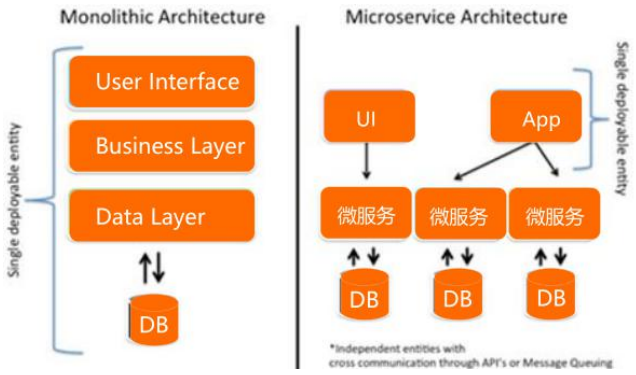
总结如下：

1）易于替换；
2）独立部署；
3）专注某个任务；
4）高度解耦；
5）基于功能进行组织：商品、支付、评论、机票、新闻、酒店、游戏等；
6）服务可以使用不同的语言、系统、平台；
7）通信使用语言中立的协议；
8）通常是 http；
9）独立技术栈；
10）易于测试。

巨型服务 VS 微服务

早期的单体巨型服务，分层架构，适用于比较稳定的系统，比如银行系统，现在还

在使用。随着业务的发展变化，可以更换新的架构。微服务架构非常灵活，合适迭代发展快的项目，比如当下流行的电商业务。



二、微服务 Microservice 缺点

微服务不是银弹：

微服务并不适合所有的场景，因为一旦拆开，通信成本就会上升，架构复杂度会上升，开发人员需要更多，集成测试、部署都会变得更复杂，所以技术选型一定要慎重。

微服务的优缺点：

合格的架构师，应该以公司业务需求作为出发点，但是很多架构师设计架构的时候，实际并不是这样，有很多其他因素在，会设计不可维护的架构。比如为了对外宣传、便于接到业务，年长的架构师担心被公司裁掉，设计不可维护的架构等。

正常情况下，需要从成本、复杂度、测试、监控等方面出发，架构服务。如果在业

务快速创新的公司，选择微服务架构，如果在业务比较稳定的公司，可以选择单体架构或者早期的架构。

微服务优点：

1）开发简单；
2）技术栈灵活；
3）协议简单；
4）服务独立无依赖；
5）独立按需扩展；
6）可用性高；
7）高伸缩性；
8）易于维护单一服务。

微服务的缺点：

1）架构复杂；
2）多服务运维难度；
3）系统部署依赖；
4）服务间通信成本；
5）数据一致性；
6）系统集成测试；
7）重复工作；
8）性能监控。

1.3 微服务架构 Microservice 的典型应用场景

内容简介：

一、微服务架构 4 大互联网公司案例

二、微服务架构 Microservice 典型场景

一、微服务架构 4 大互联网公司案例

目前打开苹果或者种安卓等机器手首页上的应用，基本上都是微服务架构，几个比较典型的代表像淘宝、支付宝、微信、微博、京东等等都是典型的微服务架构。

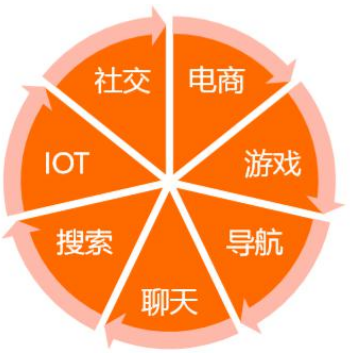
一个大的 APP 平台，里面有多子系统，但不会一起进行开发一个项目，把所有的功能全部开发完成，淘宝内部有几十个支付宝，它们都典型的微服务架构，业务模式决定了架构不可能采用一种单体形式的架构去解决所有的问题。

所以基本上互联网公司里的业务创新，BAT 这种大型公司里面都有创新孵化团队，一个新项目新业务立项以后，团队规模逐渐成百上千甚至上万，形成内部业务孵化，做起来以后，APP 平台会为大家引流，像抖后面开始做电商，电商平台做好以后直接挂进去，还有淘宝直播也是一样的，需要流量的时候直接挂进去，和大的平台不一起发布，可以单独去发，挂进去就行开个流量入口挂进去就可以。每上线一个新功能开一个入口直接挂进去就可以，已经不是单一应用了，严格来说是一个应用的生态或者应用的大的平台。

这种平台或者生态性质的这种 APP 很适合使用叫微服务加工，每个业务都会独立进行发展，通过了解几个典型的应用 APP，发现几乎业务快速发展的这种领域都适合微服务架构，但有些系统也不一定适合，因为它的业务不发展，业务比较稳定，架构也不需要迭代，一套系统用 10 年，20 年都可以达到需求，如做一套门禁系统，不需要人脸识别，能用就可以，还有银行的某个系统，不需要和移动端对接，跑个脚本就可以，总结来说和实际的业务需求有关系。

二、微服务架构 Microservice 典型场景

1. 微服务典型场景



微服务架构的案例如电商类的、微博类的、微信类、社交类的、支付类的、直播类的、游戏类、互联网类的、广告类到处都是。

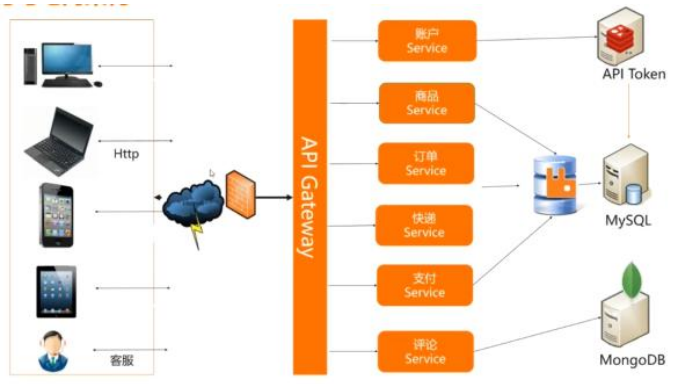
这背后反映了架构的拆分，本质上反映的是业务的一个拆分，业务的快速发展技术也一定要快速发展，技术架构快速迭代，才能去适应业务快速发展的模型，这是它的本质的特征。

2. 微服务架构经典案例

1. 《电商网站微服务架构设计案例》
2. 《新浪微博微服务架构设计案例》
3. 《微信聊天微服务架构设计案例》
4. 《支付系统微服务架构设计案例》
5. 《地图导航微服务架构设计案例》
6. 《手机游戏微服务架构设计案例》
7. 《物联IOT微服务架构设计案例》
8. 《广告数据微服务架构设计案例》

几个典型的架构做为参考，如淘宝的微服务架构，微服务架构的拆分原则，以及框架选型。

3. 微服务与电商架构



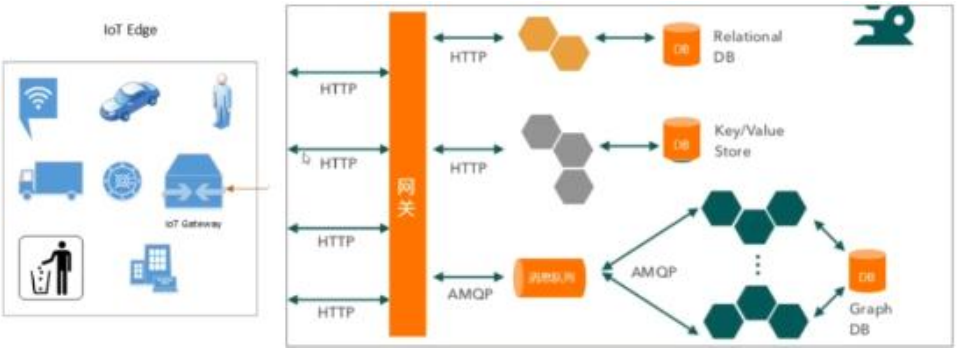
电商类以淘宝为例，是对重度使用 Java 技术架构，严格来说阿里对 Java 的整个体系的发展是做出了突出的贡献，有很多实践落地的方案，包括自己开发的一些科研框架贡献，如现在看到淘宝的账户，后面衍生出来支付的平台，剥离出支付宝，又发展成一个庞大的系统平台，里面有各种子系统，各种子业务如余额宝也是一个独立的微服务架构。

一般拆分以后维护后期，可能还要做集群，考虑高可用高并发的问题可能会做集群。这是里面体现了一个弹性伸缩的概念，如支付宝淘宝，包支付宝淘宝可以共享账号，淘宝有个概念是打通所有的平台，这些可以叫单点登录。

淘宝账号服务是独立出来的，独立进行发展，账号提供全局的统一的验证服务，可能支付宝更稍微复杂一点，有信用的接口，个人的支付的信用的大数据都在里面。

实际以淘宝京东为例这种都是典型的电商架构，前端支持的客户端，也不仅仅是传统的 PC，也包括我们说 APP、小程序都支持，客户端还会对接各种不同的系统，数据库也不是单一数据库，也不是只是 MySQL，有可能还包括大数据，包括 MangoDB、Redis 都会重度的去使用。

4. 微服务与物联网 IOT 架构



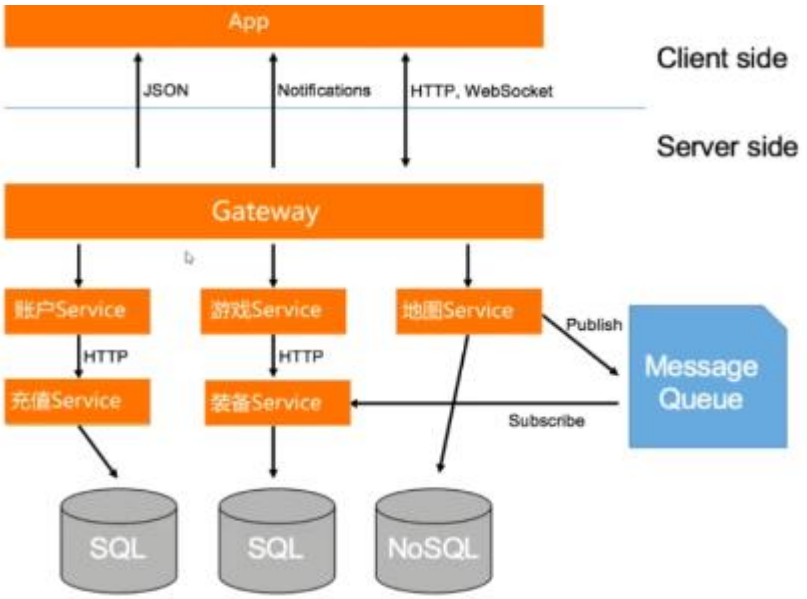
物联网现在是火爆发展，各种监控设备，包括车载的设备都是物联网的体现，移动导航的一些设备，物联网在车载设备中现在用的比较多的方向，如特斯拉、各种电动汽车大量的使用车载雷达、还有摄像头，这都是互联网，楼宇监控，尤其我交通的监控，公安的人脸识别的网络都是物联网的典型的应用，做这种解决方案很容易，尤其是汽车车载物联网系统，像特斯拉，都很适合微服务架构，还有飞机也有定位导航的设备，大楼火灾、温度、光照、湿度都有。

现在互联网架构体系的，本身也是前端对接各种不同的 IOT 设备，数据采集以后进行存储，然后进行分析，手机比如苹果手机丢了，可以设置提交你最后一次关机的手机的所在的位置，丢了手机以后会帮你找手机，通过 APP 账户对，去登录去查找最后一次

手机位置，还有很多车载，典型的哈罗单、小黄车、包括摩拜单车都是典型的物联网设备，开锁都是远程控制，它的数据定位里面有 GPS 定位的装置，来计算里程。

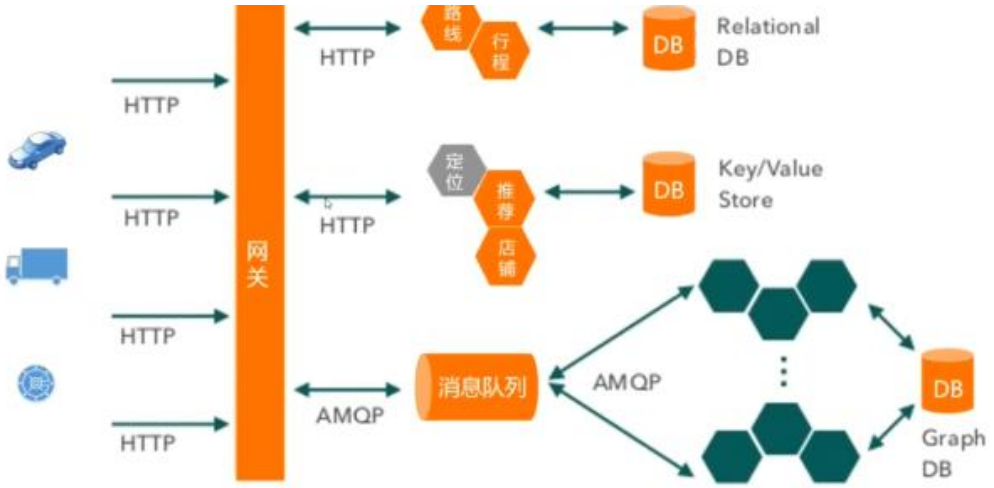
这些都这都是和典型的业务场景结合的物联网行业，物联网也是互联网，对移动互联网，没有完全划清界限，

5. 微服务与游戏 IOT 架构



游戏平台更多，腾讯做游戏在国内是最大的游戏厂商，如王者荣耀、穿越火线等等一系列游戏，微信里面可以提供入口，也是典型的微服务架构。因为账号登录其中任何一个游戏平台，都是腾讯的微信账号，这些数据独立统一以后，方便用户去访问不同的游戏平台，快速进行推广上线，用户的体验。

6. 微服务与游戏 IOT 架构



导航是不一样的，、话里面有各种业务如打车、旅游线路、推广、酒店用手机导航的时候，它会采集你的数据，这些数据会用来分析大数据，分析以后做后续的一业务的创新，都是典型的应用，都是平台性质，而且平台还有一个很重要特点，就是不断的进行业务创新，不断的进行业务创新，后面才有技术强大的一个驱动力。

1.4 微服务架构 Microservice 的淘宝改造案例

内容简介：

一、微服务架构典型案例公司

二、淘宝微服务架构改造案例

一、微服务架构典型案例公司

微服务架构典型的行业包括：电商、游戏、直播、社交、导航、互联网等，微服务典型企业有：淘宝、支付宝、微信、微博、美团、拼多多等。国内的互联网公司 Java 技术架构相对比较多一些，像阿里是 Java 技术站，企业级开发无论是分布式 SOA 服务治理，还是微服务架构，阿里在 Java 领域贡献非常大。

当然亚马逊也在去 0racle,也有自己的 Java SDK 分支。亚马逊是全球最大的电商和云计算公司，阿里紧随其后，腾讯主要是社交和游戏为主，服务端大部分是 C++，也有 Java 项目。蚂蚁金服是阿里的基因，偏金融类。另外国内的京东、拼多多、网易、滴滴打车、哈罗单车、陌陌、微博等，都是以 Java 技术站为主的公司。

Java 诞生时间比较早，企业级架构解决方案应该是无敌的，生态非常完善，有众多案例公司。现在比较重要的云计算领域，能够使用云计算平台的公司，很多使用 Java

技术架构，还有微服务也是使用 Java。

从全球来看，谷歌和亚马逊技术创新能力非常强的两大公司。还有 Spring Cloud 最早成熟的解决方案来自于 Netflix 公司，项献的是公司内部自研的微服务架构方案。还有一些游戏公司，如做愤怒的小鸟的公司，优步的公司，也都是 Java 技术站为主。

国内的华为公司现在也大量引入 Java 技术站，早期做通信主要是 C++为主，现在做华为云、华为手机的很多后台应用基本上都是 Java 技术，安卓开发基本上用的都是 Java 语言。还有中国平安、IBM 老牌 Java 技术站、头条、陌陌、携程 2016 年转 Java、京东 2010 年转 Java、饿了吗、小米等等。

百度也有 Java 的项目，各种语言为主，企业级开发方案相对比较少。目前国内实践探索比较多的是阿里巴巴，贡献了很多开源的解决方案。

二、淘宝微服务架构改造案例

1. 淘宝高并发架构 1.0—PHP+MySQL

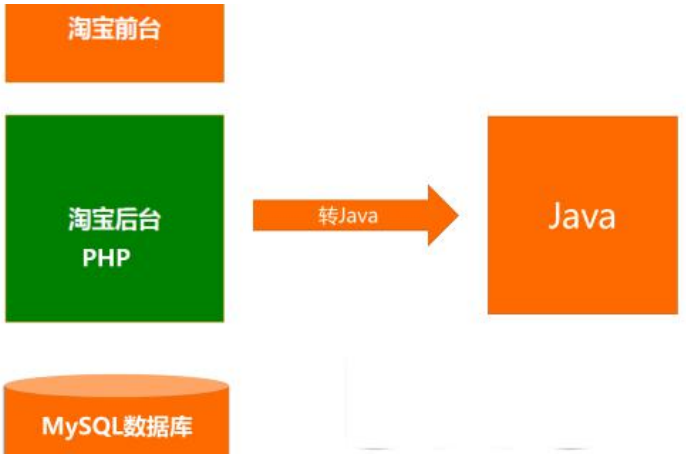
2003 年创立淘宝到 2021 年，经历过一系列技术战略转型，从**单体**到**集群**到**分布式微服务架构**，再到**云计算平台**，一系列的改造过程，几乎代表了互联网在中国发展的最典型案例。淘宝架构不断变化，也是很多人拿淘宝架构作为学习参考的原因。

当然也有公司架构一成不变的案例，说明公司业务非常成熟，架构可以重复很多年，比较典型的有银行案例，早期银行项目基本上是大型机，使用 Java、IBM DB2 数据库、Oracle 数据库，在当年的环境下，这种方案可以解决银行业务的安全性与稳定性，以及高并发的需求。

2. 淘宝高并发架构 2.0—PHP+MySQL

后来淘宝转 Java，尝试用开源的低成本路线解决企业高并发的问题，在当年面临很大的转型压力，很多阿里巴巴自研的包括 Java 分布式、MySQL 数据库中间件、Dubbo、HSF、Spring Cloud 等一系列组件，现在都项献到开源社区，共大家参考。

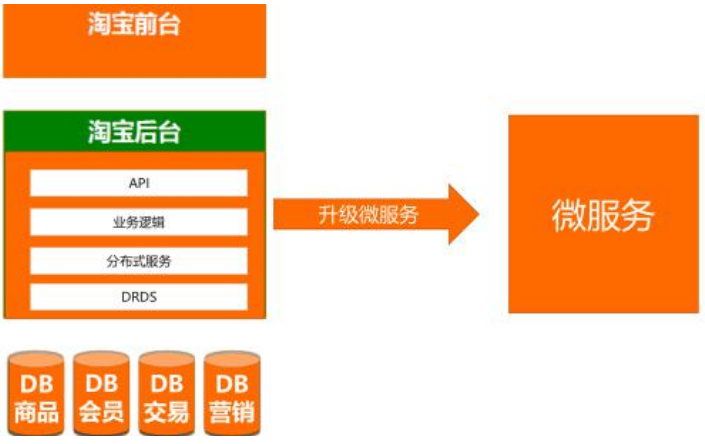
今天大家在开源社区看到这些源码，在国内是开源领域最大的互联网公司，在世界能排到前二，跟谷歌、微软开源贡献的差不多。



淘宝的技术架构，对中国技术社区的发展分享了很多宝贵的经验，值得大家学习。阿里输出的技术人才，不光在阿里集团，对于中国的互联网发展，产生了很大的影响，阿里出来的很多技术专家，到其他公司也都担任核心技术岗位，做技术架构的设计工作。

3. 淘宝高并发架构 3.0—Java 分布式架构

后面淘宝向大规模分布式集群、服务治理阶段发展，去 Oracle 数据库转 MySQL，还有很多新的技术出现。



淘宝也在逐步的微服务化，并且淘宝、天猫、咸鱼、飞猪各种平台都接到淘宝里面，淘宝的业务越来越多，包括淘宝的广告系统、营销系统、客服系统，是典型的分布式过渡到微服务架构阶段。

前端从 PC 阶段逐步过渡到移动互联网 APP 的时代，包括小程序，这是很典型的互联网平台发展的代表。

4. 淘宝高并发架构 4.0—微服务架构

目前火热的中台架构，严格来说是技术总监 CTO 应该解了的战略层次的方案方针，现在各种技术大会上也在炒作这个概念，但是无论是做微服务架构，还是中台架构，一定要能够落地，不能不懂装懂，只知皮毛。

实战阶段以目前最成熟 Spring Cloud 微服务架构为主，不管你是 Java 出身的成员，还是非 Java 出身的成员，都希望大家能动手写代码。

1.5 微服务架构 Microservice 的经典协议

内容简介：

一、微服务常用的通信模式

二、经典的 RPC 协议

三、微服务常用的通信协议

一、微服务常用的通信模式

1. 微服务的消息通信模式

消息交换的模式有很多种，使用较多的是同步消息的交互模式，典型特征是发送完消息后会等待一个结果，

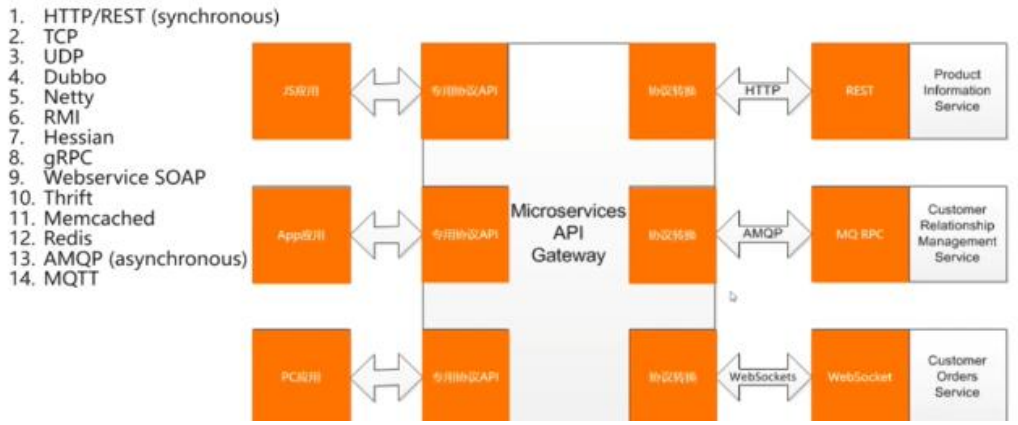
浏览器发送一个网页请求后，会等待网页返回，中间存在请求应答的过程，这就属于同步请求的模式。

异步请求模式的常见场景是消息推送，发送完某个消息后，这个消息并不会立即到达，可能会经过一定延迟才到达接收方。异步模式的优点是并发或吞吐量较高；缺点是无法保证消息的实时性。

目前在分布式架构上，同步与异步相结合的消息交换场景也很常见。

协议上绝大部分都是同步模式，个别支持异步，例如邮件协议或者消息协议。

二、经典的 RPC 协议



上图是在分布系统中常用的一些 RPC 协议。RPC 本身是远程过程调用，主要解决远程的通信问题，而不仅是封装原始的数据通信协议与网络协议。

在此基础上，需要借助某些框架语言来实现功能的交互。例如，希望客户端通过调用服务器端的某个订单或者加密的功能，实现远程的功能调用。

这是通过网络来暴露自己代码功能较早的一种方式。RPC 协议非常多，不仅是 REST API,APP 协议暴露接口，前端分裂架构基本上也都是 API 加前端、小程序、APP、PC 网页等这种模式，是在移动互联网时代用得较多的架构。

前端分离，后端演化成微服务架构。微服务架构一般和业务模式有关，业务需求是第一位，技术服务于业务。在内部通信领域，并非只有一种协议，可能多种协议并存，如 TCP、UDP、HTTP 等协议并存。

Rest API 基本上走的是 APP 协议，一般是接收数据格式。在这个领域里面，RPC 概念在 native 本身也支持分布式通信框架。但是相对来说，在大规模分布式集群治理领域，阿里的 Dubbo 设计非常优秀，不断迭代，表现优异。

数据库 JDBC 属于分布式通信解决方案之一，但通信协议是 GDB 框架定义，专有的协议格式，支持引入中间链、消息队列等，使用不同的协议进行通信。

这并非表示跨平台是最优秀的，它的性能越好，安全性越高，可能越封闭。但是它开放性标准化有助于大范围的行业推广，适用性更强。有些公司的协议不开放，这是由于从公司的业务角度来说，微服务属于分布式架构。

分布架构继承了早期的分布式框架特点，在 RVICES API 这些应用基础上进行了架构的升级改造，在大规模的服务接口集群化治理走向了更高的层次，架构师面临的挑战也更大。

三、微服务常用的通信协议

1. 微服务之间的通信协议与框架中间件

- 同步调用
  - HTTP/REST (synchronous)
  - REST（http, websocket, JAX-RS，Spring Boot, node.js, Web API)
  - RPC（Netty, Dubbo, Thrift , gRPC)
- 异步调用
  - AMQP (asynchronous)
  - 消息队列(ActiveMQ, RabbitMQ, Kafka, RocketMQ，Notify, MetaQ)

目前，微服务以 Spring Cloud 开发为代表，选取的是 Rest 通信接口格式，后续的微服框架可能更多，有些微服务支持更多的协议和数据格式。目前主流的是 HTTP，属于同步消息通信模式，H5 走 websocket。

当下的移动互联网时代大部分追求轻量级接口，目前的框架如 Rest API、Java、Go 还是 node，都非常方便，可以直接在 APP 协议站基础上进行扩展。

不同语言的 REST 框架基本由 WEB 框架改造，再加一层数据序列化反序列化即可满足大部分场景需求。因为对于绝大部分前端 APP 来说，数据格式基本是主要的那些格式，没有复杂的路由策略。早期已经具备了基础的接口开发框架，但是还不具备整体性复杂的架构层级的微服务架构风格。

消息队列在早期分布式架构中经常使用，比较经典的是 RabbitMQ 兔子消息队、Kafka、RocketMQ 火箭消息队，Kafka 的消息并发量和吞吐量能达到百万级别。

对于全新的微服加工来说，可能会存在几种协议，有同步、异步和两种形式并存，也有可能是使用 APP 协议或其他的 TCP 或者二进制等相关协议，这几种协议经典且各有所长。

在物联网行业存在特殊情况，可能进行对接时候，设备可能还有自己的数据协议，但是每个分布式框架底层可能支持一种协议或者多种协议。

1.6 微服务架构 Microservice 的开发框架

内容简介：

一、微服务架构的开发框架

二、微服务开发框架对比

三、Spring Cloud 微服务架构生态最完善、最成熟

四、Spring Cloud 微服务架构

各位同学大家好，我们继续来学习微服务架构设计课程，咱们今这一节课来讲一下微服务架构，经典的开发框架，咱们这个系列课程是以 Java Spring Cloud 的为主导的重点的开发框架，目前微服务生态经过 5 年以上的这种发展历程，其中也出现了其他语言，包括 Java 本身 Spring Cloud 也在不断迭代，也出现新的贡献微服务架构的框架。

Spring Cloud 是出现的时间比较早框架，并且它的生态也是最完善的，咱们选择了 Spring Cloud 作为目前我们课程的主要的实践案例的练习的框架。

下面一起来看下微服务架构有哪些经典的开发框架，以及他们有什么差异，做实际的项目选择的时，大家应该怎么来进行选择。如果技术选型的话，可能直接会影响后面一个架构的落地，因为有些架构并非完善，需要话很大精力去处理相同的问题。

一、微服务架构的开发框架

1）Spring Cloud：最早最成熟，Java 开源微服务框架方案

2）Dubbo ：阿里巴巴开源 Java 服务治理框架

3）Spring Cloud Alibaba 阿里开源 Java 微服务框架方案

4）SOFA：蚂蚁金服开源 Java 金融微服务框架方案

5）Go Micro：Go 语言开源微服务框架

6）Seneca Microservices ，Node.js 微服务框架

7）KumuluzEE：Java 的微服务框架

8）Enduro/X： C/C++/Go

目前 Spring Cloud 是最早的开源的版本，主要几个核心框架是谁贡献的呢？是叫麦飞美国的视频网站公司，他们把自己公司内部实践的开发的微服务的解决方案框架贡献给了社区。它主要也是想体现云计算的这样的时代的特征，这个核心框架基本上都是用 Java 来进行编写的，而且也现经过这么多年的发展，目前在全球范围内来看的，它是最成熟的一套生态。像国内的阿里巴巴、蚂蚁金服、京东、微博、拼多多、美团等新崛起的互联网公司都在使用 Spring Cloud 微服务框架体系。

像淘宝、阿里的发展实际代表、Java 或者来讲分布式架构，不断发展历程。淘宝做双十一在商业上非常成功运营模式，但是它在另外一个技术层次来说是倒逼着阿里的技术团队去不断的迎接各种技术挑战，并解决这些问题。

除了 Dubbo 以外，内部还有 hsf 框架，早期是解决大规模服务治理的问题，后面进行在内部不断优化协议、性能。Dubbo 在开源以后，国内有很多互联网公司都在用，影响也比较大。作为微服务架构设计的选型的话，Dubbo 不会作为首选，但是 Dubbo 是一个有效的补充。它的优点：经过阿里巴巴集团大规模验证、在不断的迭代、支持高频发，成为响应式框响应式框架、Java 在不断的升级演化，Dubbo 也不例外，协议支持的更多。

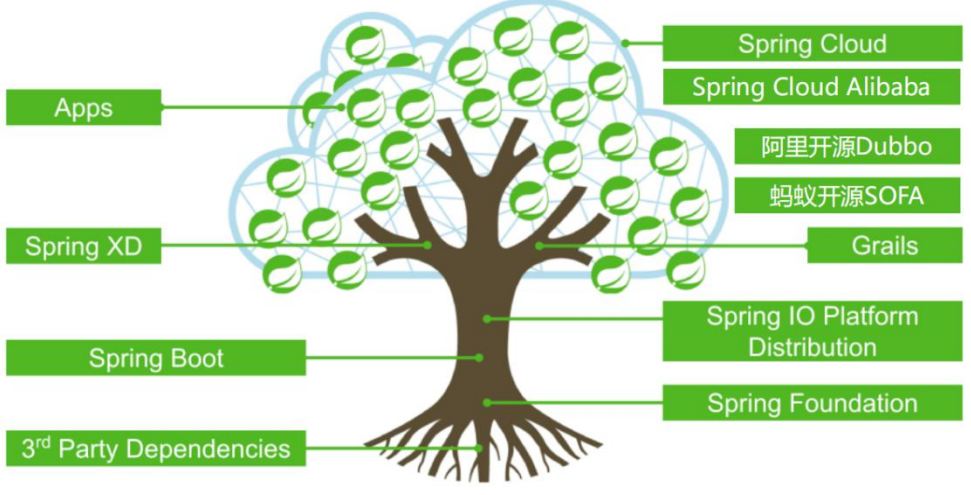
并不是所有的场景，用 HTTP 协议是最优秀的，后续 Spring Cloud 的版本或者其他的微服务框架，会在协议，通讯协议，数据格式类型尝试做一些优化，因为阿里打包开源一些列的微服务给 Spring Cloud 作为贡献的一部分。外国就是麦飞，国内就是阿里巴巴作为最大的贡献者。像蚂蚁金服的 SOFA、GO 语言的 Go Micro 都是仿制 Spring Cloud。生态都不是很完善，没办法和 Spring Cloud 一样的流行，成熟。Java 并不是强在语法，开发工具都不是最好的，强在框架和生态，升级模式。新的设计模式的书都是率先在 Java 实践出来的。

二、微服务开发框架对比

名称	协议	说明	成熟度	案例公司
Spring Cloud	REST/HTTP	Java开源微服务框架体系	最成熟，最流行	BAT、IBM等
Dubbo	Dubbo/TCP/REST	阿里巴巴开源Java服务治理框架	最近支持微服务	阿里巴巴
Spring Cloud Alibaba	REST/HTTP	阿里开源Java微服务框架	Spring Cloud补充扩展	阿里巴巴
SOFA	REST/HTTP	蚂蚁金服开源Java金融微服务框架	刚发布，互联网金融领域	蚂蚁金服、中国人民保险，南京银行
Go Micro	REST/HTTP	Go开源微服务框架	个别公司	不详
Seneca	REST/HTTP	Node.js开源微服务框架	个别公司	不详
KumuluzEE	REST/HTTP	Java EE开源微服务框架	个别公司	不详
Enduro/X	REST/HTTP	C/C++开源微服务框架	个别公司	不详

服务框架本身也有存在各种差异，Spring Cloud 是出现最早的，最成熟最完善的一套微服务架构综合解决方案。它协议上 HTTP 为主，国内外公司大部分都在使用。阿里的 Dubbo 也开源了，并向微服务进发，开始也支持其他的框架的集成。虽然 Go 语言和 C++都有微服务框架，但是出现的比较晚，生态并不是太完善。作为微服务架构师来说，Spring Cloud 和 Dubbo 体系最完善。

三、Spring Cloud 微服务架构生态最完善、最成熟

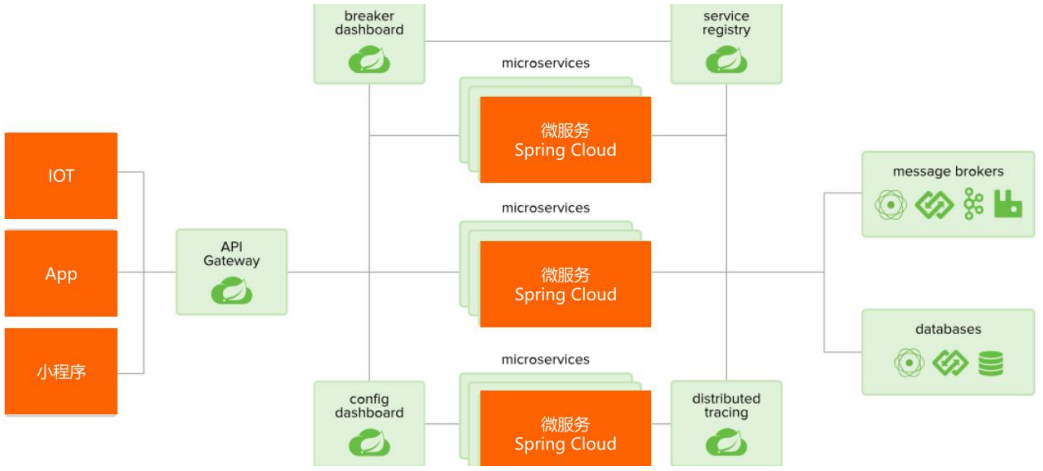


另外 Spring Cloud 本身成长于 Java Spring 整个平台体系中来，之前 Java Spring 积累的所有生态工具都可以拿过使用。可快速开发、集成、安全设计模块工具、容器的工具可直接使用。加上国内互联网公带头扩展实践落地。Spring Cloud 是架构选型风险成本最低的。如需单独开发新的框架或者新的协议，费时费力。所有的技术发展一定是和公司的业务发展紧密结合，工程师本身是帮别人去开发系统解决问题的。做架构师的话选型一定要选成熟的、完善的、风险最小的、资料多的，遇到问题的时候更容易解决问题。

Java Spring 开发平台

Spring Boot、Spring Data 都在协助 Spring Cloud 维护开发，拥有一套很大，并完善生态。

四、Spring Cloud 微服务架构



框架它是在不断迭代不断发展的，微服务开发的话不仅仅是 Spring VC 或者 Spring Boot，还有各种工具 ，也可以来开发这种微服务，实现各种需求：网关、熔断、注射、包括数据库交互、前端对应 APP、小程序。微服务拆分好处在于走小块零的路线，也可以单独进行部署。比如：有的系统订单服务支付服务用的比较多，比如淘宝单独做大规模集群，客服投诉服务使用就比较少，拆分之后占用的服务器少。拆分的原因取决业务需求，存在差异之后，可以分开处理问题。

Spring Cloud微服务架构挑战性问题与方案 阿里云

1. 服务开发：Spring Cloud, Spring Cloud Alibaba, Dubbo
2. 服务注册：Eureka/Nacos
3. 服务发现：Eureka/Nacos
4. 服务部署：Docker、K8s
5. 服务网关：Zuul
6. 服务通信：Http和Websocket、gRPC
7. 服务容错：Hystrix
8. 服务监控：Hystrix
9. 集群监控：Hystrix、Turbine
10. 负载均衡：Ribbon,

每个环节解决不同的问题，无论是注册、发现、部署、网关都有自己的对应解决方案框架。

目前的话就是选型的话，首推是 Spring Cloud，因为是一个成熟完善的生态。在技术选型的时候还要尊重业务需求、技术落地、风险问题、考虑公司研发成本等综合考虑，单单是自己语言的喜好的问题。

目前来讲，这里面给大家介绍了系统各个不同语言的微服务开发方向，但是目前 Spring Cloud 还是最成熟最完善的。下节课的话咱们也看看微服务架构里面经典的设计模式，包括我们整个的拆分的微服的一些原则。

1.7 微服务架构 Microservice 的设计策略

内容简介：

一、微服务 Microservice 的设计原则

二、微服务 Microservice 的拆分原则

三、微服务 Microservice 设计的关注点 1

四、微服务 Microservice 设计的关注点 2

五、微服务 Microservice 设计的五大考量

咱们这节课重点关注如何进行微服务架构设计。第一阶段我们主要是偏重理论体系和微服务相关的整个系统的重要概念原则，会给大家介绍打下坚实的基础，为后面的实战练习做铺垫。

学习分三大阶段

一、微服务 Microservice 的设计原则

1. 微服务架构的设计原则

首先大家来思考一个问题，如果让你做微服务架构设计的话，你怎么样去设计微服务？你的微服务应该具备哪些特点的特性？

给大家总结重要的一点，我们记住一个问题，微服务架构本身也属于分布式架构，只不过它是更复杂的分布式架构。我们在讲微服务概念的时候，咱们提到过微服务实际它是诞生于 SoA 时代所以它还具备 SoA 架构的一些特点，记住我们所有的架构设计有很重要的一个原则：

需求第一：一定要以需求为出发点。所有的架构好与坏一定是相对的，相对他处的一个需求背景。因为微服务架构能够在某些业务场景中具备优势，所以它相比传统的架构，他有一些优点但是同时也存在着缺点，它不完美。

单一职责：我们的服务尽量是体现单一职责的思想，粒度不是越细越好，也不是越粗越好。

协议统一：还有尽量去统一协议，不包括不得已的话，我们不引入其他协议像我们一般微服务的话，咱们讲现在目前的协议主要是 rest 有可能会比如说有可能你会引入消费者的协议，或者引入其他的这种通讯协议，当然在我们说都是基于实际的需求

独立开发：独立开发一般咱们这里面提到的我们说的是模块拆分以后开发人员一般是独立我们按照模块进行拆分，然后每个人负责一块，每个人熟悉一块代码和逻辑业务逻辑这样的话开发时间都会相对来说高很多

独立部署：独立部署这也是微服架构的很重要的一个原则，咱们讲了微服务架构拆分以后又会出现可能很多程序很多进程，而且每一个模块不是所有的都更新只需要迭代我那一块就行了，就是体现了我们说叫分而治之的这样一个思想，大家一起统一部署。

二、微服务 Microservice 的拆分原则

1. 按照业务模块拆分

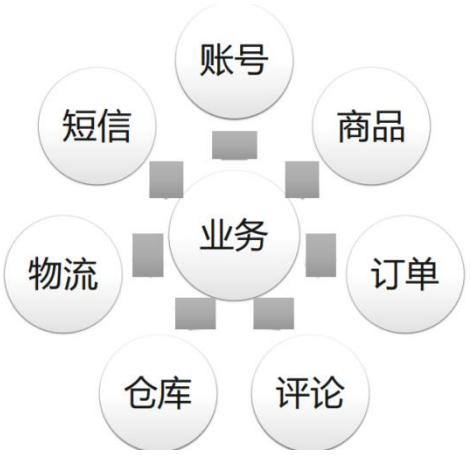
你找一个大概的服务，拆分力度的话一般的话还是基于业务。

2. DDD 思路可以借鉴，不能照搬

DDD 本身不是架构设计模式，DDD 是一种但是在向对象设计的一个思想或者原则，它是用来解决复杂业务逻辑的一个拆分问题的，它本身并不解决整个架构层次的问题，它是解决业务层的，理解这一点。

3. 单一职责 Single Responsibility

4. 电商架构设计：



1）账号模块

2）商品模块

3）订单模块

4）评论模块

5）快递模块

6）短信模块

7）支付模块

8）卡券模块

9）信用模块

10）酒店模块

场景它本身就像个生态一样，它里面接入的功能模块多，这里面天生适合和足够庞大的基础上适合分人制快速迭代。微服务架构，新的业务诞生早期可能只有两三台服务器，后面的话做起来可能上千台服务器。包括游戏也一样前端很多平台是属于导游的模式，加入一个模块进来，这个平台作为一个入口。拆分原则一般的话我们是基于业务进行拆分，或者你也可以说是 DDD.

三、微服务 Microservice 设计的关注点

1. 微服务架构设计的关注点

微服务架构设计的时候，大家关注并发性，可用性、安全性、密等性、重用性。

四、微服务 Microservice 设计的关注点

1. 微服务架构的新特性

服务隔离，各个服务之间互不影响，高并发了也不要影响，扩容也不要影响。

五、微服务 Microservice 设计的五大考量

1. 微服务架构设计的 5 大考量

一个架构师它采用的微服务架构一定要说服别人需要给个合适的理由。服务架构里面在做事物的话是比较难的，所以要注意一个数据一致性问题。

1.8 微服务架构 Microservice 的经典设计模式

内容简介：

一、微服务架构中的经典设计模式

二、微服务设计模式分类

一、微服务架构设计模式

1）业务分解：DDD 模式

2）DataBase PerService 每数据库单服务

3）API Gateway pattern API 网关模式

4）Client-side Discovery 和 Server-side Discovery 模式

5）Messaging 和 Remote Procedure Invocation 模式

6）SingleServiceper Host 和 Multiple Services per Host 模式 7

7）AOP: Microservice chassis pattern

8）Externalized configuration

9）Service Component Test 和 Service Integration Contract Test

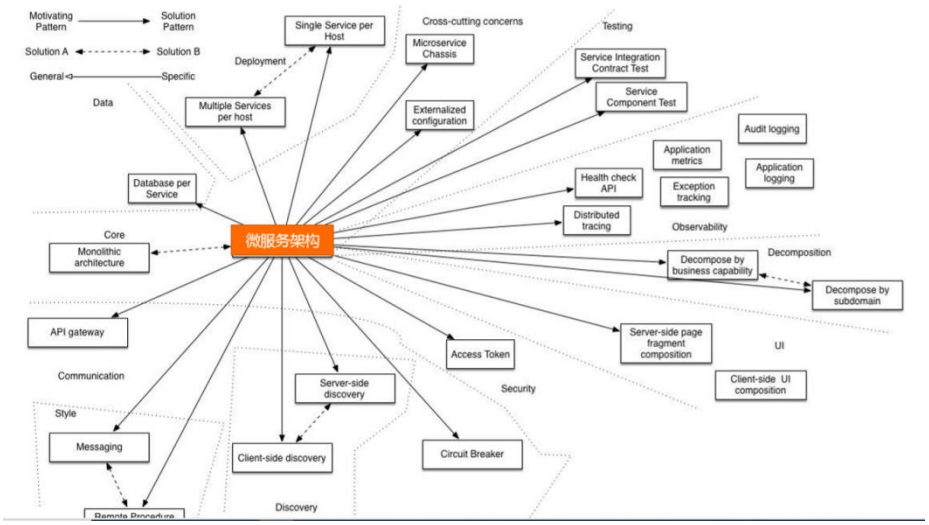
10）Circuit Breaker 断路器模式

11）Access Token 访问令牌模式

12）观察者模式:Distributed tracing、 Health check API

13）UI 模式:MVC、MVPMVVM 模式

微服务架构中的经典的设计模式，一般提到微服务会认为微服务架构指的是一种架构，实际上微服务架构本身涵盖几十种设计模式，可能后续还有更多的设计模式。



如图所示，以微服务架构为中心向外发散，有许多设计模式，正下方有两个，一个叫客户端发现，一个叫服务端发现，服务的注册和发现机制也是一个设计模式，微服架构属于更复杂的分布架构，里面也会用到消息通信，通过消息和数据库、其他微服务进行消息补偿。

网关的微服务太多，只有一个出口，需要给它同一个代理；安全问题，如图中 Access token，和令牌相关的；另外还有高并发的熔断限流，circuit breaker 叫断路器模式，熔断相关，分布式日志、分布式加策、追踪、服务拆分模式、单数据库模式、单实例、单数据库模式多服务共享数据库模式、服务编排模式、统一配置模式等等。

这里主要是分布式架构领域相关的设计模式，还有分布式事务模式，一般用的都是补偿的方式。

服务拆分的一般借鉴 DDD 模式，但不是照搬，不能完全等同。

二、微服务设计模式分类

应用架构模式：

单点登录

注册发现

熔断限流

断路器

网关模式

消息补偿模式

令牌模式

数据库:

分库 Single Service

共库多 Service

多库同步

事务性补偿

日志追踪模式:

观察者模式 patterns

Log aggregation

Application metrics

Audit logging

Distributed tracing

Exception tracking

Health check API

Log deployments and changes

分布式链路追踪模式

UI 模式:

MVC

MVP

MVVM

Server-side page fragment composition

Client-side Ul composition

1.9 Java Spring Cloud 微服务开发环境配置

内容简介：

一、Java Spring Cloud 开发工具

二、Eclipse 开发工具

这节课讲的是微服务架构 spring Club 的这套微服务体系开发环境的一个搭配，我们有一些技巧，一些方面的开发工具大家直接拿过来使用，来节约我们的整个的一个开发时间。但原始 Spring Cloud 做开发方式的话你可以自己完全的去手动搭配，但是使用有些工具对大家来说叫事半功倍，你可以去投入更少的精力，得到更好的一个学习效果。

一、Java Spring Cloud 开发工具

1. Java Spring Cloud 开发工具

1）Linux、Windows 、Mac OS

2）Java 8+

3）IDE：Eclipse 4.6+或者 IntelliJ IDEA

开发环境的话主要是根据个人习惯，用 Mac 开发会稍微贵一点，Windows 开发相对于较便宜可能一万能买苹果两万元的配置，java 环境一般都是 8 这个版本，IDE 大家用的基本上就是国内破解版，企业用的话也要慎重，企业版的功能更强一点，你这个社区版的话可能有很多功能用不了有 Eclipse 你可以装个插件，智能提示可以设置一下。

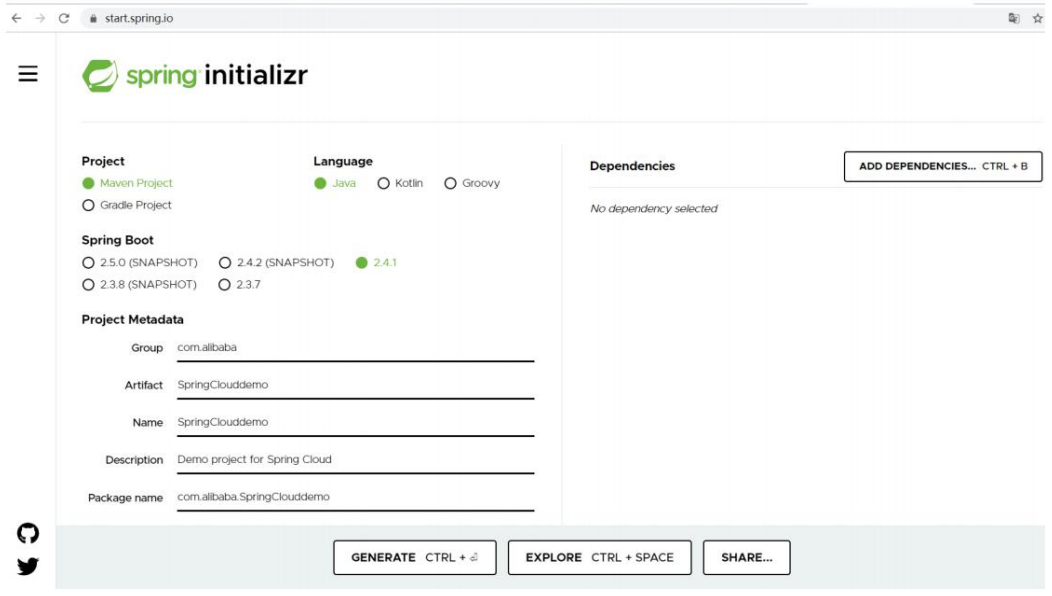
2. Java Spring Cloud 开发工具下载地址

1）Eclipse: http://www.eclipse.org/

2）IntelliJ IDEA https://www.jetbrains.com/idea/

配置 jdk（不会的上网搜索流程）苹果环境下稍微麻烦一点苹果快捷键的话你要适应。

3. Spring Cloud 在线开发工具



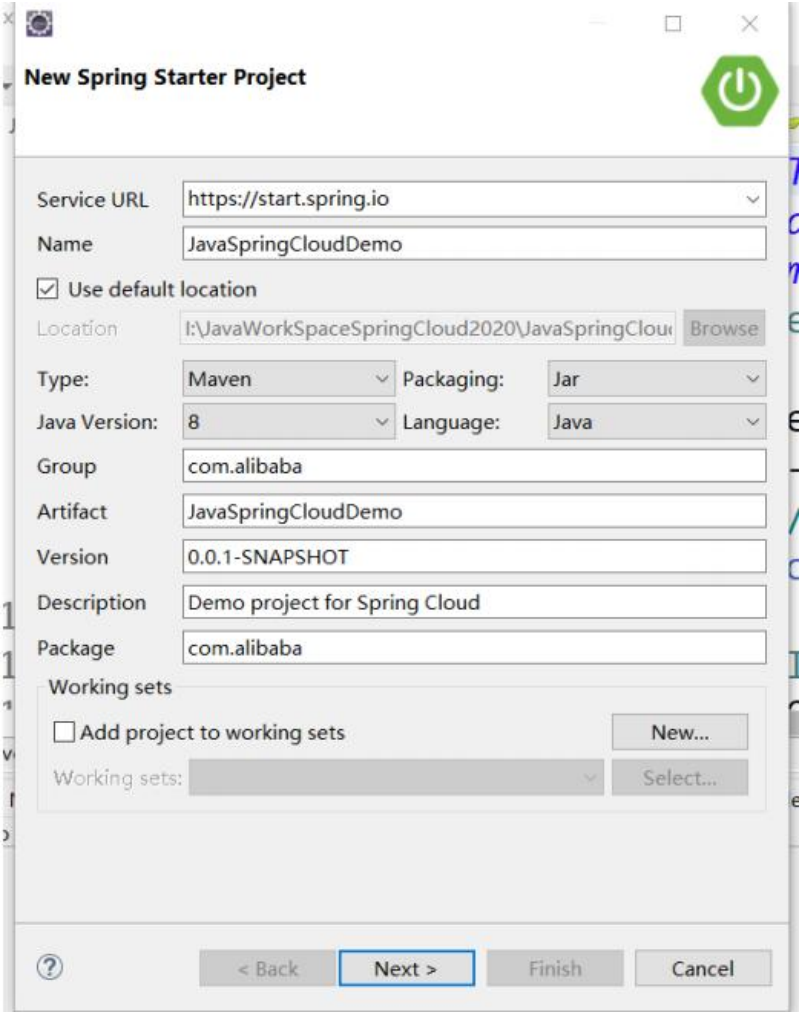
官方也提供了在线的一个项目的一个快速生成器这个地方的话你可以直接选 2.5 版本这是指的 spring boot 版本不是 Spring Cloud 的版本因为 Spring Cloud 本身寄生于 spring boot 之上所以它的版本的话其实都是英文名字。而我们 spring boot 版本是数字编码的基本上相对来说比较好记。一般的话现在大家做 Spring Cloud 或者做 spring boot ，基本上都是 2.0 以后的版本。

依赖的话你可以在右侧直接输入进去然后添加 Api 或者消息网关做容量限流在里面，组建可以收日志都可以然后生成一个压缩包，那么你在解压导入 Eclipse 就行了不行导入这个 IDEA 也可以

这个工具也可以手动来做，改配置文件完全都不用也行对于特别熟练的成员来说，他这么干是可以的，刚入行的同学的话，一般的话开发咱们还是需要智能提示，借助一些工具。

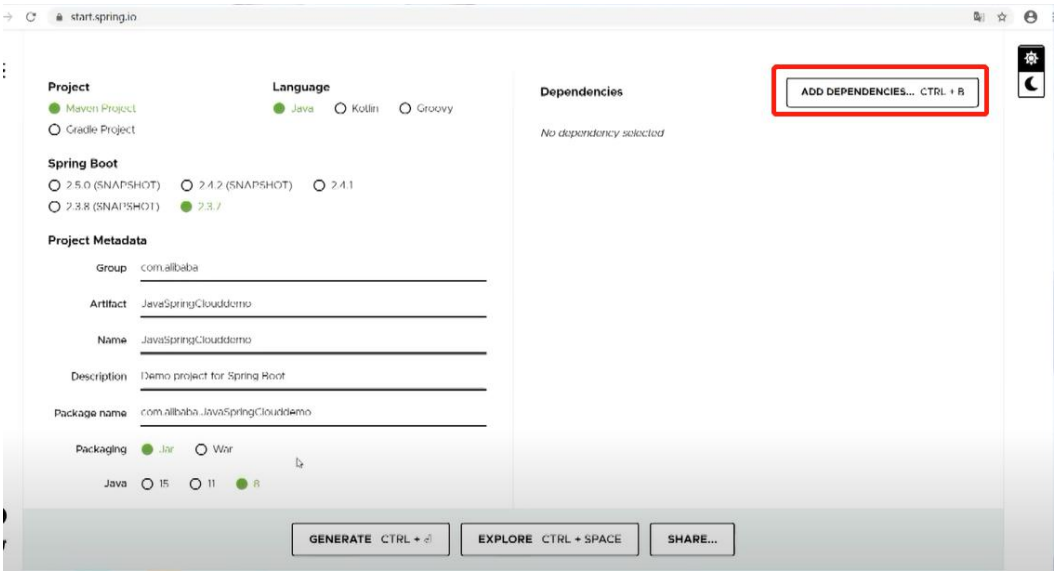
二、Eclipse 开发工具

1. Spring Cloud 微服务模板向导

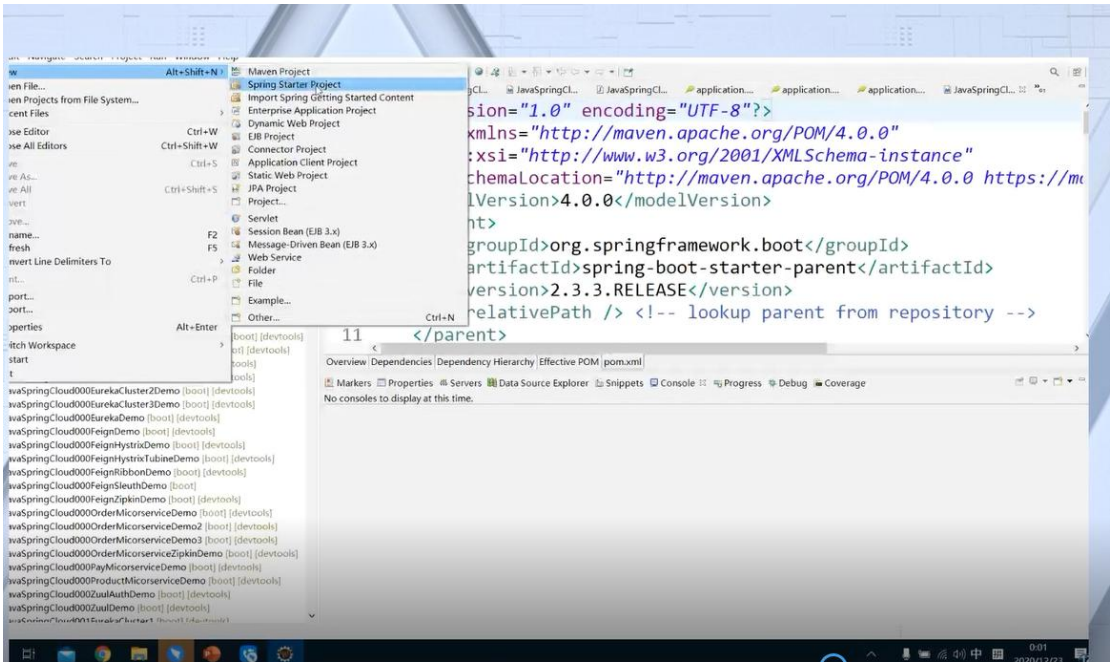


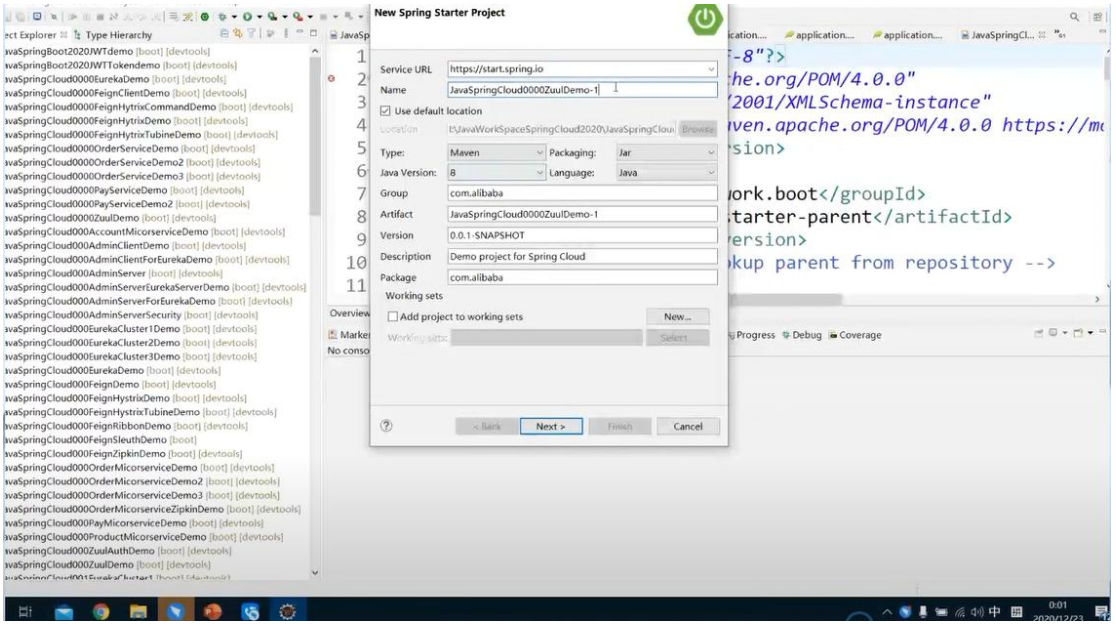
有 Eclipse 你可以装个 spring 插件，安装插件后你自己可以创建项目的时候，可以直接去快速的去创业一个 spring boot 项目跟在线的版本是一样的。IDEA 大家自己也可以装一下但是 IDEA 本身装完以后可能稍微吃内存一点因为它大量的智能提示所以比 Eclipse 更废内存

网址：start.spring.io



这里可以搜索需要的，搜索的都可以用，然后点生成就可以使用。





插件装完后，这里面也可以添加，新建项目的时候会有一个 spring starter 的向导效果然后下一步就可以了。

1.10 Spring Cloud 2020 重大变化与选型提示

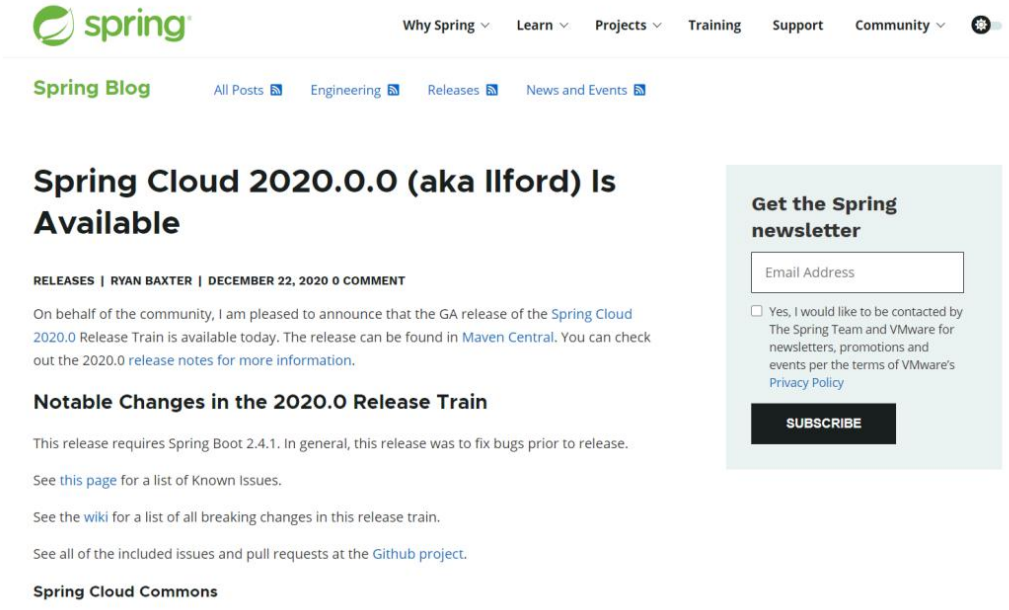
内容简介：

一 、Spring Cloud2020 发布及重大变化

二 、Spring Cloud2020 提醒

一 、Spring Cloud2020 发布及重大变化

1. Spring Cloud 2020 发布



Spring Cloud 2020 重新规划了发展路线版本号,命名规则已经改成了年度+扩展版本号，作为一个微服务框架来说，它里面有很多成熟的设计模式思想包括算法可以供大家借鉴。因为大家在开源社区学一些技术时，很重要的就是研究底层的代码和设计思想设计模式还有算法。2020 这个版本采用的命名也是伦敦的一个地点名，但我们这里面实际版本改成数字形式。

2. Spring Cloud 与 Spring Boot 版本对应关系

Spring Cloud版本	发布时间	Spring Boot版本
2020.0.x aka Ilford	2020/12	2.4.x
Hoxton	2019-07	2.2.x, 2.3.x (Starting with SR5)
Greenwich	2018-11	2.1.x
Finchley	2017-10	2.0.x
Edgware	2017-08	1.5.x
Dalston	2017-05	1.5.x

目前新版本 Spring Cloud 版本 2020 为 2020 年 12 月份发布，Spring Boot 对应版本为 2.4.X 目前课程为 2.3 版本，但是实际不影响。新项目请尽量使用 2.1.x 以后版本，实际操作中可选表格中绿色部分，Spring Cloud Dalston, Edgware, and Finchley 由于时间较久部分不在支持. 不要太旧也不要太新，新版本刚发布会有很多坑不完善。

3. Spring Cloud2020 重大变化

1）架构选型注意版本差别

2）Spring Cloud 2020 基于 Spring Boot 2.4，不支持低版本

3）删除部分 Spring Cloud Netflix 组件

4）Bootstrap 默认禁用，可以使用依赖兼容老项目

5）org.springframework.cloud:spring-cloud-starter-bootstrap

6）Spring Cloud LoadBalancer 支持服务端统计

7）Spring Cloud Kubernetes 新增响应式 Java Client、LB 统计

8）Spring Cloud Openfeign 支持 Spring Cloud CircuitBreakers

9）Spring Cloud Security 代码移到独立的项目中

10）Spring Cloud Gateway 支持 LoadBalancer 统计

11）Eureka Client 的 RestTemplate 支持 TLS 属性

架构师也是很重要的，要有技术深度也有技术广度，见多识广，你才知道好坏，你只知道一个框架，你很难分辨出它的好或者坏。

4. Spring Cloud2020 以下项目从 spring-cloud-netflix 删除

- spring-cloud-netflix-archaius
- spring-cloud-netflix-concurrency-limits
- spring-cloud-netflix-core
- spring-cloud-netflix-dependencies
- spring-cloud-netflix-hystrix
- spring-cloud-netflix-hystrix-contract
- spring-cloud-netflix-hystrix-dashboard
- spring-cloud-netflix-hystrix-stream
- spring-cloud-netflix-ribbon
- spring-cloud-netflix-sidecar

- spring-cloud-netflix-turbine
- spring-cloud-netflix-turbine-stream
- spring-cloud-netflix-zuul
- spring-cloud-starter-netflix-archaius
- spring-cloud-starter-netflix-hystrix
- spring-cloud-starter-netflix-hystrix-dashboard
- spring-cloud-starter-netflix-ribbon
- spring-cloud-starter-netflix-turbine
- spring-cloud-starter-netflix-turbine-stream
- spring-cloud-starter-netflix-zuul
- Support for ribbon, hystrix and zuul was removed across the release train projects

主要原因还是两个公司的利益纷争，一个要可能要考虑商业化，另外一个贡献了大量代码，公司掏钱养了员工做，这个项目源码贡献出来以后，没得到任何好处。整个生态来看的话，应该还是支持更多公司参与，这样的才能促进整个微服务架构社区的繁荣发展。

二 、Spring Cloud2020 提醒

1. Spring Cloud 2020 微服务架构学习提醒

学习建议

- 建议学习 Greenwich 以上（ 2.1.X 版本）版本
- Spring Cloud Netflix Greenwich 以上（ 2.1.X 版本）相对成熟
- Spring Cloud Alibaba 相对成熟，部分组件可以替换
- 企业使用多，踩坑基本完毕，容易落地架构
- 新版本 2020 可以作为扩展学习

国内很多公司选择 Netflix Greenwich 这相对来说比较好用。一般架构设计的话不会选最旧的，也不会选最新的，一般的话选隔个一年半年的时间的检验期的这样一个版本是比较稳妥。

2. Spring Cloud 2020 微服务架构设计提醒慎重选择新版本

- 企业微服务架构大量的 1.5.X 版本、2.0.X 版本
- 推荐架构设计使用 Greenwich 以上（ 2.1.X 版本） •Spring Cloud Netflix 相对成熟
- Spring Cloud Alibaba 相对成熟
- 新的 Spring Cloud2020 版本需要一段时间检验，踩坑

框架体系完善度、成熟度，文档丰富度、规范度等都是我们选型很重要的考虑，有些语言压根没有微服务完整的微服务框架。

2.1 为什么选择 Java Spring Cloud 微服务架构

内容简介：

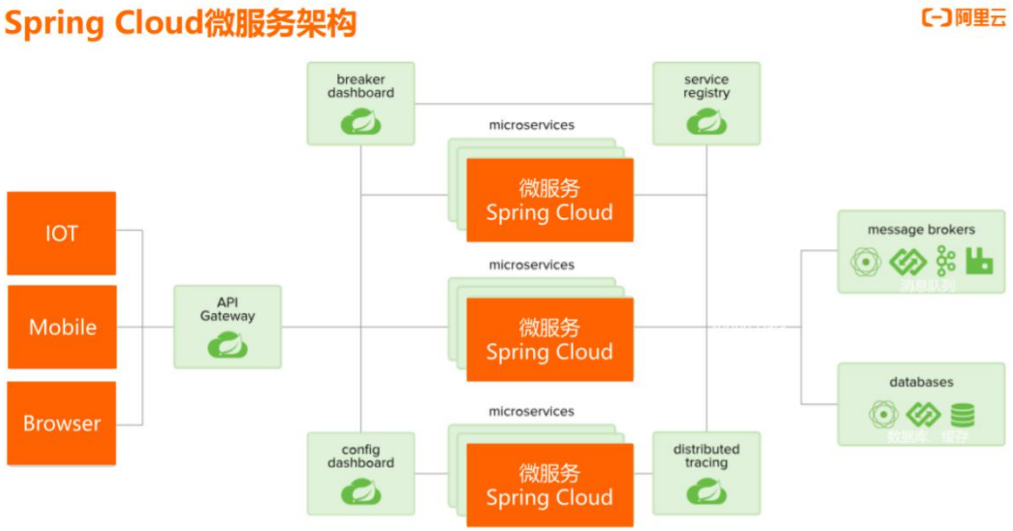
一、Spring cloud 的使用现状

二、Spring cloud 的优势

三、学会 Spring cloud 对就业的好处

一、Spring cloud 的使用现状

我们这节课继续讲解 Spring cloud 的微服务架构设计，在上节课已经要求大家去安装 Spring cloud 的开发工具和环境。这节课我们来讲一下选择 Spring cloud 的一个原因，就是告诉大家的我们为什么要选 Spring cloud。我们前面已经强调过多次，Spring cloud 它是出现最早并且生态最完善的这样的一套微服务架构的解决方案，它不仅仅是一个框架，它是一系列框架。



现在的话大家能看到的 Spring cloud 官网，我们整个的框架体系达到数 10 个应该超过 30 个以上。大家应该了解过 Spring cloud，实际是奈飞公司内部实践的一套微服务架构的逻辑框架，然后贡献给社区 2014 年开源，在之前已经经过很长时间的落地实践。它在社区里面贡献出来以后，是对整个的一个微服务架构的发展做出了巨大贡献，许多公司使用 Spring cloud 做微服务架构的一个落地和开发工作。

这里面我们选 Spring cloud 很重要的一个原因，就是因为它非常的成熟，非常的完善，并且也是非常的流行使用范围最广泛。这里面使用的公司很多，我们在现在能看到的这些，头部互联公司里面几乎都有 Spring cloud 微服务架构的这种项目。

二、Spring cloud 的优势

无论是在国内还是在国外，基本上都是首当其冲是属于排名第一的微服务架构的整套解决方案，其他的语言一般的微服务架构的方案的是仿 Spring cloud 但是还不够完善，像公语言、或者 C#都有后续的一些仿制，但是我们在生态上都是有所欠缺。其它语言的话本身语法上可能是比 java 语言更好开发，工具可能更好用一点，但是在这个企业级架构实际上 Java 是没有对手的， Java 强在生态、架构这一块，这一点是 Java 它最大的一个优势。

当然我们说实际上公语言作为后起之秀，也有自己的一些设计方面的一些优点。比如说他有携程的概念，已经把微服、现成的概念进一步进行封装。当然后面改进版本的这种 Java 的详细编程，实际在吞吐量这一块的话性能有大幅提升。

另外一个问题，技术选型很重要一点，作为技术专家或者项目负责人，架构师选一个框架能解决问题，还要考虑一个框架的成熟度、社区是不是完善文档是不是完善、好不好招人。如果是选择这个框架基本找不到人用，这个就要考虑一个研发成本，自己掉坑踩坑的一个成本，公司是不是允许，所以这里面的大家也要稍微注意一下。

三、学会 Spring cloud 对就业的好处

Spring cloud 目前在国内的这种头部互联网公司，不光是 BAT 了，其他这些大公司的招聘，基本上 Spring cloud 都是作为技术专家和架构师招聘的一个必备要求，可能很多同学的话过段时间要换工作，所以你要花时间去准备一下这些知识。

这些知识咱们在前面几节课里给大家大量讲了一些我们做微服务架构的关键的面试题、拆分原则，微服架构哪些设计模式。我们今天讲这些落地的话，告诉大家如何通过具体的框架去开发微服务，怎么样去把微服务架构搭建起来，并且进行落地实践。这里面咱们需要架构师或者技术专家一定要能力上一定能够做到了这样的一个很重要的考量的标准。

之前我们提到过，很多公司大家可能遇到过这种所谓的架构师或者技术专家，什么技术架构都懂，但是自己你让他去配一个框架，它框架都不会用，开发工具都不会配。在座的各位一定要能够做既懂理论又懂实践这样的一个架构师，叫真专家、真架构师而

不是伪专家，只会吹牛逼的这种是非常招人烦的。

但我们希望各位的能力的话能够匹配自己的岗位，目前无论在国外还是在国内，实际上十分可靠的公司是非常多。虽然有些公司以某些语言为主，但是咱们也说一个问题，很多公司它其实是并没有这种技术实力，绝大部分公司没有这种技术实力自己去搞研发的，比如你自己做一套微服务框架出来，成本非常高。因为绝大部分公司都是在中国互联网公司里面，主要都是业务驱动，大家看这些公司主要关注业务，比如说怎么样去把做电商平台，怎么样做个游戏平台，怎么做个社交平台，同质化的东西很多，主要靠运营。怎么把这个产品做大，做强用户量做起来，交易额做起来，然后上市圈一笔钱赚一把，基本上都这个套路。

国内的这些大银行或者这种金融公司，像平安做的不错。像我之前也去平安、中国银行、中国银联、花旗银行做过这种架构的分享的课程，大家也可以看一下，这里面比较有意思。比较注重技术的这些金融证券公司银行，也是在紧跟时代的潮流，当然我们说现在这些公司基本上也是以 Java 为主。互联公司比如滴滴、美团、四通一达，这些也都是典型的代表，这都是基本上也都是 Java 技术架构为主。

说明一个问题，在中国公司里面阿里可能做创新做的还好一点。他有达摩院招了一批科学家去研究各种新的技术，但是实际对于小互联网公司来说生存都是问题。所以基本上用成熟的方案是最好的选择。避免公司踩坑，前面有人带路已经把路给踩通了，比如奈飞。阿里巴巴已经把技术架构方案给验证过了，并且完全是能够满足我们当前的这种业务架构的需要，直接参考它来做这是最好的。并且像阿里开展了很多框架，而且是它的架构方案的话也会在网上进行分享，大家也能看到很多这种文章，包括一些技术大会的讲座。

当然另外一个就是我们选 Spring cloud 的原因就是 Spring cloud 本身做微服务架构生态非常完善。提供的微服务开发框架是超过 35 个以上，有一系列框架对，对接不同的数据源。包括 Spring boot 也非常好用，简化了整个开发过程。但是另外一点也稍微注意一下，作为一个 Java 开发者，有些人是用框架很熟，不懂底层。尤其是新入行的一些成员可能会被 Spring boot 给迷惑住。

后面的开发企业越来越简单封装的越来越好，很多人就不懂底层原理了，需要稍微注意一下。在座的各位花时间可以抽空研究一下底层原理。Spring cloud 它是植根于 Spring 平台之上，并且可以很好的和一些 Spring cloud 框架进行集成，我们可以快速的去落地我们微服务架构的一个方案。

大家在搭建微服务过程中，前面已经强调很多次，维护结构本身的坑比较多，或者叫挑战性问题比较多，架构师的职责是非常重的。之前的话，大家很容易就说自己是一个架构师。比如我会设计一个三层，或者设计一个四层，打个安全卡的集群，我就认为我是个架构师，在一个公司里面就可以混得很好。

但是咱们讲一个问题，你现在再吹自己是架构师，微服架构师的话，在这里面就有点麻烦了。前面已经讨论过很多次了，很多公司都有这种就声称自己技术很牛逼，或者声称自己是技术非常厉害的人，但是你让他写代码或者让他自己去搭一套框架，他基本上搭不出来。有些只是会画大饼，这种其实是比较遭人反感。

微服架构师一般人不敢随便说，很多原因就是因为他知识点太多了，容易说错，你说你是微服架构，我就问其中那个框架你是怎么用的，你解释不出来，你这就丢人了，所以现在的话大家也可以看到在技术圈里面装逼的话是不轻易装大数据架构师，也不轻

易装微服务架构师，因为这两个体系都足够复杂，这也是比较逗的一个现象，当然你说你是一个普通的 Java 架构师，这就不好衡量。你是一个 3 层架构师，还是一个 4 层架构师，那就不好说了，所以如果你说你是微服架构师的话，我就问一下其中的某一个点你是怎么做的，怎么解决这个问题，那就不一样。衡量的难度一下尺度就上来了，我们说是这对架构师来说能力要求是上了几个台阶，而不是一个台阶。微服务架构里面，拆分以后有许多的问题，列入应用、服务注册发现等这种单点问题。

在昨天我们装的工具基础上，开始搭建微服务架构，我们先来以注册中心为例，大家先把注册中心给搭建起来，在讲为什么选十分架构的时候，也把理由充分给电压进行了一些讲解，主要还是为了后面的学习。另外就是大家的以后在汇报微服务架构的技术选型是不是可以和领导讲得比较全面透彻，这也是给大家做了这样一个总结性的第二阶段的那一刻。

咱们下节课来讲，数字中心的落地和实践由大家去写代码，我们希望大家是属于能够懂理论，又能够进行时间架构设计落地的这样的一个名副其实的架构师。

2.2 Spring Cloud 微服务注册与发现 Eureka

内容简介：

一、Spring Cloud 服务注册与发现

二、Eureka 注册中心开发实战

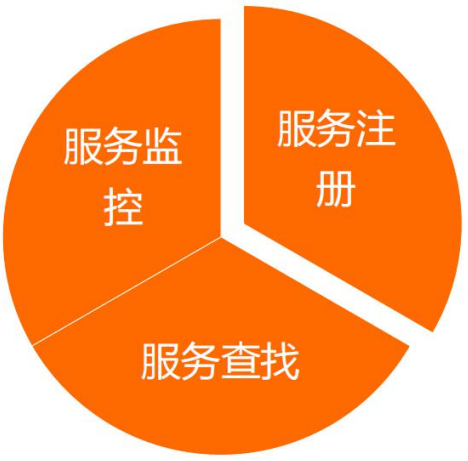
三、Eureka 底层原理与源码分析

各位同学大家好，我们继续学习 Spring Cloud 微服务架构，设计与实践开发课程。这节课的话我们讲下实战的开发工作，我们现在要做什么？我们要做一个微服务的注册中心，来解决微服务的一个注册和发现问题。

一、Spring Cloud 服务注册与发现

如何解决微服务的注册和发现，对于微服务来说，我们要实地进行拆分，进行部署的时候，实例数量是不固定的，可能是 2 个或者 200 个，2000 个，都有可能。微服务的实际数量是弹性伸缩的，这一点它和传统的架构不太一样。微服务弹性实际上有点像云计算原生靠拢，这点也是他的 优势。微服务本身拆分以后能够很好的进行治理，进行快速的部署。

服务注册与发现



1）大规模微服务集群架构

2）许多服务实例

3）客户端要找到自己调用的服务

4）新服务上线

5）某个服务宕机，下线

6）实时监控服务的状态

解决大规模服务的集群的注册和发现问题主要为了方便方便客户端的一个调用，假设我们开发了一个微服微的订单服务，开始只取一台服务，客户端直接调用微服务就可以了，但是如是取两台，写 2 个服务 IP 地址或者做轮巡都不太理想，因为遇到大型促销场景，需要增加 10 台，或者 1000 台都要更改配置列，服务的实力实力全部给拿进来。

当服务的微服务的集群的数量增加的时候，弹性增加的时候，不定增加的时候，这时候需摇有一种方案能够去解决这个问题，把客户端和服务集群能够解偶。解偶很重要就是注册中心，帮我们去管理这些服务。

Spring Cloud Eureka 服务发现与注册

1）Netflix 公司开源的项目

2）Eureka：注册中心

3）一个基于 REST 的中心服务，管理服务

4）实现云端的服务注册和服务发现

5）Eureka 组件组成：Eureka 服务器和 Eureka 客户端

6）竞争对手 ZooKeeper

7）服务发现模块（Eureka）是 Netflix 的核心

8）Spring Cloud Netflix 提供的简化开发模板

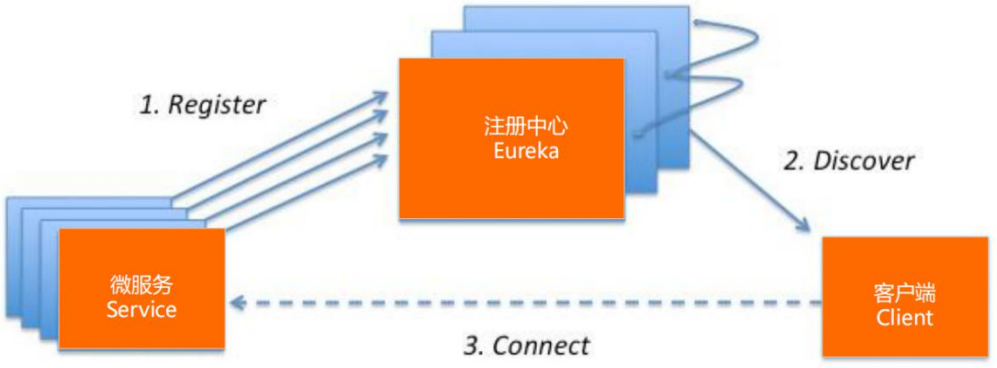
9）直接使用 spring boot，创建项目

10）添加@EnableEurekaServer 开发注册服务中心

在微服务架构体系里面，Netflix 公司贡献了其中一个很重要的项目叫 Eureka，主要是解决服务注册中心的问题。ZooKeeper 也是同类型的产品，Spring Cloud 通过扩展组件进行集成。

咱们主要是要介绍实在开发，后面叫底层原理原码给大家留做扩展作业，后面给大家介绍。

Spring Cloud 架构图



微服务的实体数量是动态的，有可能是一台，也有可能是两台，有可能是 100 台 1 000 台，我微服务的数量是弹性的，灵活弹性根据客户端的压力我来做弹性伸缩，我客户端调微服务的话，不是直接找微服务了，是先找注册中心，有哪些好的微服务，哪有哪些匹配的，比如我找订单微服务，就是找支付微服务，每次去找搜索，如有最新的列表给到，再去调用它们。客户端也可以按照各种负载均衡的策略去调，结合一些负载均衡的算法来做一个灵活的调度。

二、Eureka 注册中心开发实战

Spring Eureka 注册中心

1）创建 Eureka 服务注册中心项目

2）添加@EnableEurekaServer

3）将 spring boot 应用改造成 Eureka 服务注册中心

4）application.properties 增加配置

5）打包项目

6）运行

7）测试页面

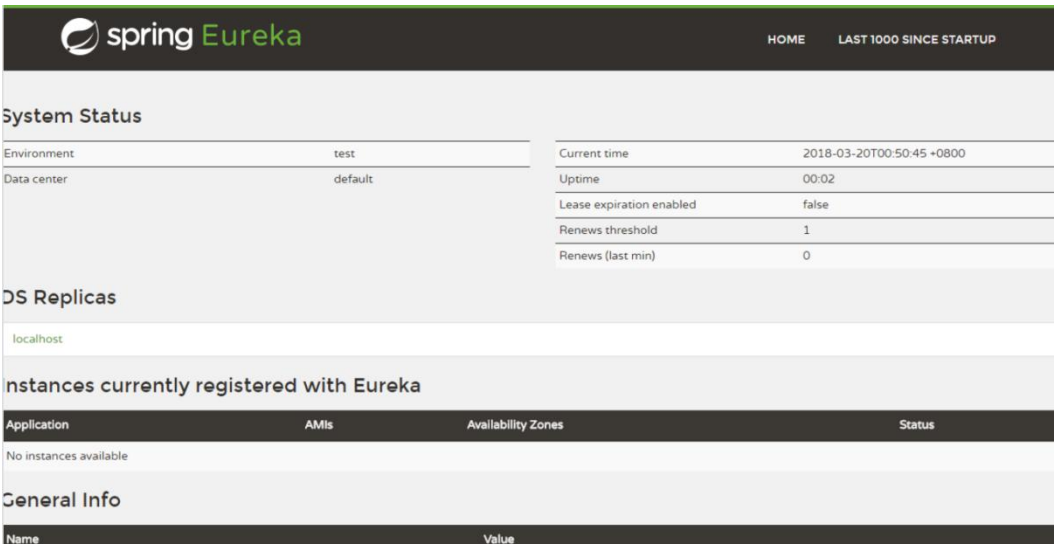
8）参考 https://spring.io/guides/gs/serviceregistration-and-discovery/

application.properties配置

- server.port=8761
- eureka.client.register-with-eureka=false
- eureka.client.fetch-registry=false
- eureka.client.serviceUrl.defaultZone=http://localhost:${server.port}/eureka/
- logging.level.com.netflix.eureka=OFF
- logging.level.com.netflix.discovery=OFF

配置文件内容

数据中心主要复杂的就是配置，如果记不住可以在文档里面拷贝过来。



配置中心正常的界面

注意事项：

```
6 @EnableEurekaServer
7 @SpringBootApplication
```

@EnableEurekaServer，主要是让服务具备注册中心的能力。

再进行配置文件，配置文件有几个核心参数，端口一定要配对。

三、Eureka 底层原理与源码分析

Netflix Eureka源码

- Eureka 官方源码
- https://github.com/Netflix/eureka
- Spring Cloud Netflix 适配 Eureka 的代码
- https://github.com/spring-cloud/spring-cloud-netflix

服务实例Instance的状态

1. Up
2. Down
3. Starting
4. Out_Of_Service
5. Unknown

2.3 Spring Cloud 微服务 API 实战开发并注册到 Eureka

内容简介：

一、开发 Spring Cloud 微服务 API

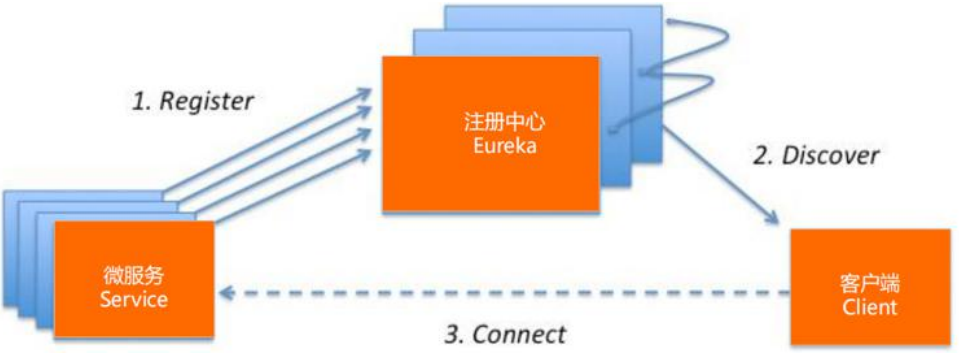
二、把 Spring Cloud 微服务 API 发布到注册中心

三、注册中心查看微服务信息

从技术角度来说，现在目前框架支持的比较友好，模板架构做的也非常完善和智能化，大家很容易去写一个 API，在 Java 中其实也不例外，通过简单的几个注解都能实现。

如果你只会写 API，这并不代表你会做微服务开发。微服务架构的话有几十种设计模式包括 Spring Cloud 的体系是目前是还在不断迭代，现在的话新版本 2000 的新版本也已经发布了基于 Spring boot 2.4 这个版本的话，默认版本号已经换掉了之前的版本，不是基于数字化的。现在改成年份加数字编码这种格式，恢复绝大部分编码形式。大部分企业后续选型的话建议大家选的还是 Spring Cloud 2.1，2.2，2.3 为主，不要再用 2.0 了因为官方可能就要只能在支持半年就不会再做支持，虽然也可以做微服务架构的开发，但是后续的话可能就会改掉，麦飞有些项目可能慢慢停止维护，Spring Cloud 可能推自己的官方，原因是 Spring Cloud 官方公司想做企业化卖解决方案，包括他要使用自己的框架，并且还希望能够把他的微服务框架针对云原生平台进行功能性扩展。

Spring Cloud 微服务基本架构图



咱们来讲如何开发一个微服务程序，并且把它微服务程序注册到我们的数据中心。你如果说只写一个简单的接口给客户端调用，可以实现前后端分离，也可以实现简单的这种 Register 调用但是它不是叫微服务架构，它只是一个简单的 Register 风格 API 应用，我们这里面的话需要借助于 Spring Cloud 提供的组件，让我们的微服务程序具备去向服务中心进行注册，并且能够被查找发现的这样的一套扩展功能。

一、开发 Spring Cloud 微服务 API

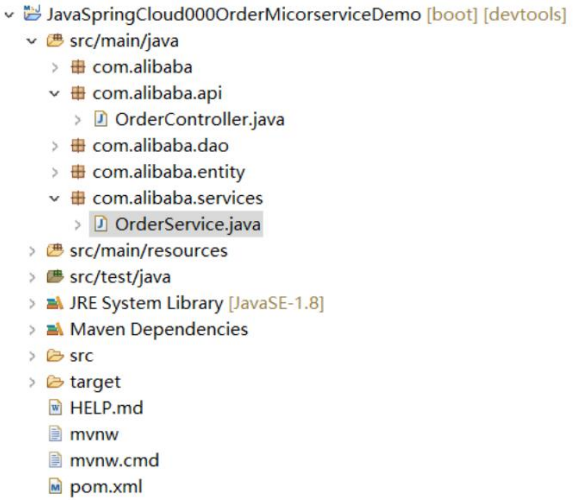
1. 开发 Spring Cloud 微服务 API

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-hystrix</arti
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
```

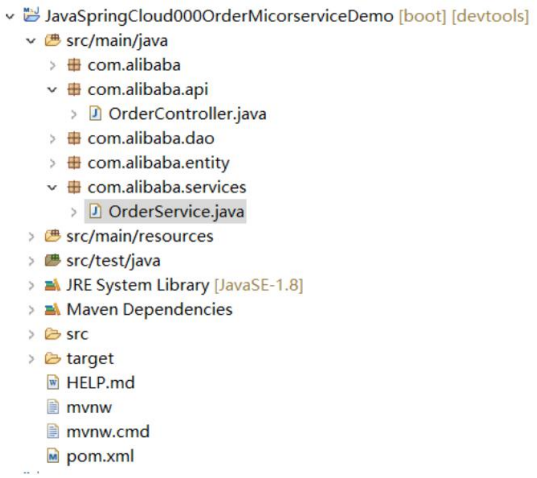
开发商服务和我们之前写的 RegisterAPI 很像但是你要多一个注册中心的一个包，如果你要做微服务监控。

2. 开发 Spring Cloud 微服务 API

- 微服务项目结构



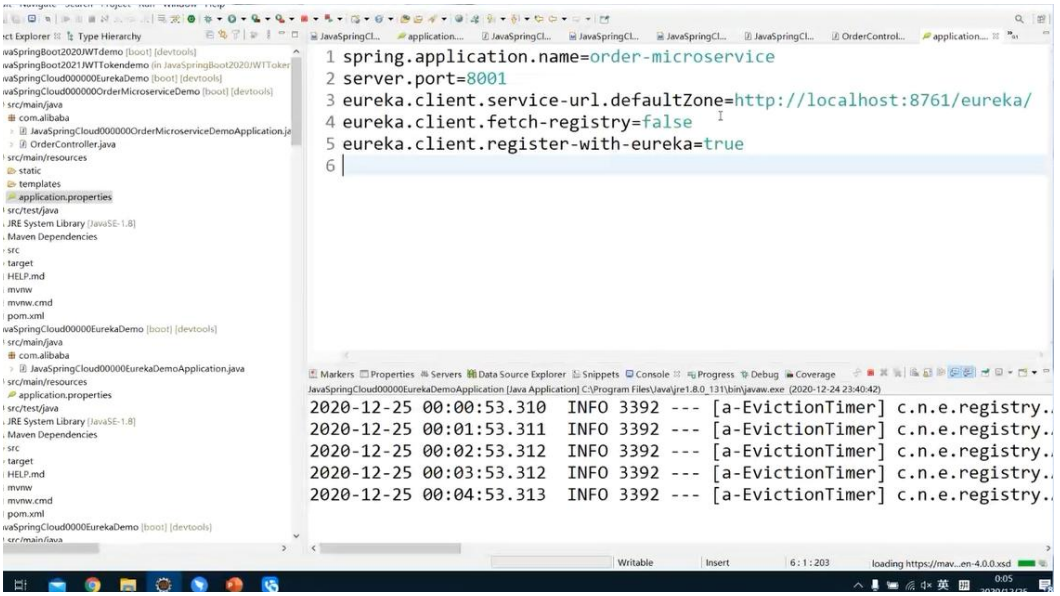
- 微服务项目结构



二、把 Spring Cloud 微服务 API 发布到注册中心

1. 配置 Eureka 客户端项目

- spring.application.name=order-microservice
- server.port=8001
- eureka.client.serviceUrl.defaultZone=http://localhost:8000/eureka/



配置文件

1）取名：订单的微服务名称

2）修改端口：改成 8001

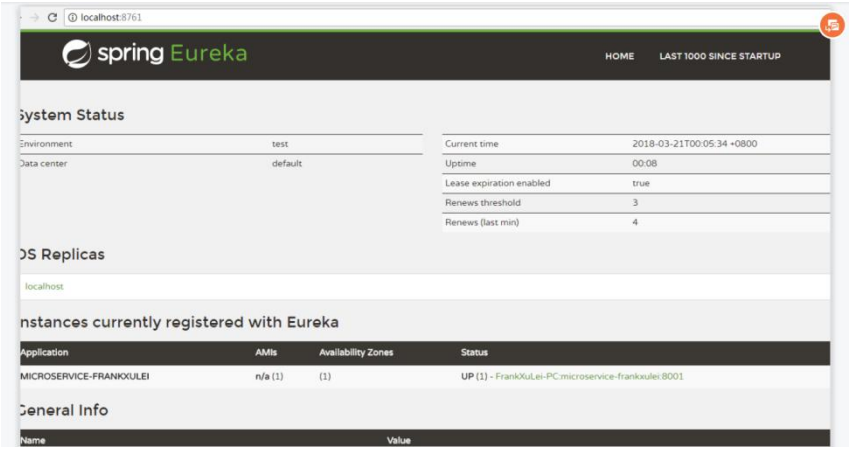
3）服务注册中心的地址，端口主要是控制注册中心的地址客户端程序要知道控制中心在哪才决定要不要给它查询。

4）表示我要不要像注册中心查询信息

5）表示我要像注册中心进行注册

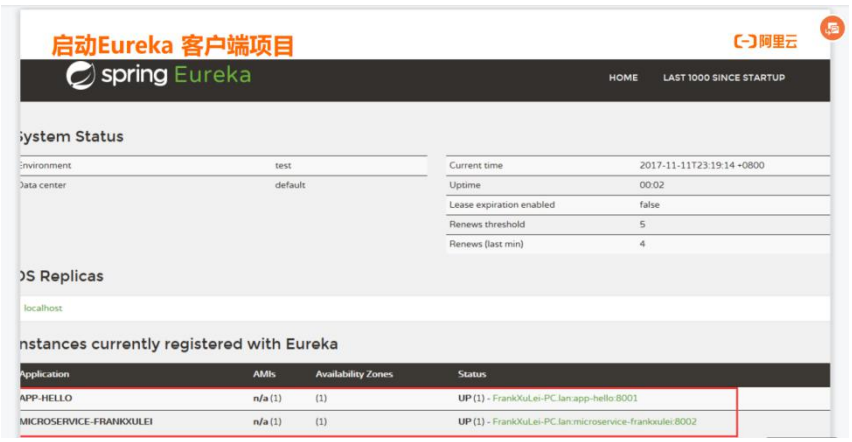
三、注册中心查看微服务信息

1. 启动 Eureka 客户端项目



MICROSERVICE 叫订单服务 UP 表示服务器活着，down 表示关闭。

2. 启动 Eureka 客户端项目



3. 查看微服务元数据

- http://localhost:8761/eureka/apps
- 可以查看所有的服务信息

2.4 Spring Cloud 客户端 Feign 调用微服务 API

内容简介：

一、Spring Cloud 声明式调用客户端 Feign

二、调用客户端 Feign 实战

一、Spring Cloud 声明式调用客户端 Feign

1. Spring Cloud 调用方 Feign

微服务架构中非常重要的技术叫声明式客户调动端 Feign，Feign 主要是做快速构建微服务 API 的调用工作，组件是非常重要的。

如果做微服务开发，开发出来一个订单的 API 或者注册的 API 给客户端用，客户端自己使用原始的 HTTP 库等等相关的库去做请求调用是可以的，但相对来说比较麻烦，还有一种情况就是构建这种复杂的微服务架构，会面临一个问题，微服务可能要调别的微服务，这时候调用，不可能所有代码全部用原生基础的原始的代码来进行构建，这时候 Spring cloud 为了方便微服 API 的调用，把整个 API 的调用工作在客户端代码大大简化，自己封装了一套工具框架。

2. Spring Cloud 调用方 Feign

1）调用方，简化微服务 API 调用

2）Feign 是一种声明式、模板化的 HTTP 客户端

3）简化 Http 客户端开发

4）只需要创建一个接口+@注解

5）Feign 注解和 JAX-RS 注解

6）Feign 支持可插拔的编码器和解码器

7）Feign 默认集成了 Ribbon，并和 Eureka 结合

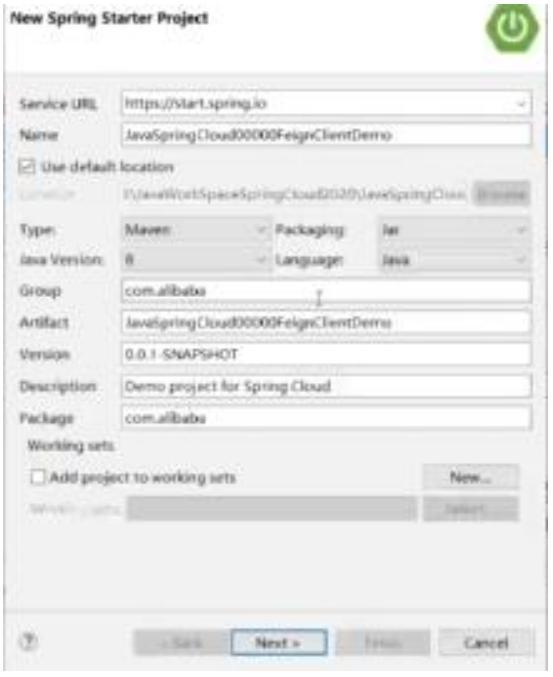
8）Eclipse 或者 IDEA 实战

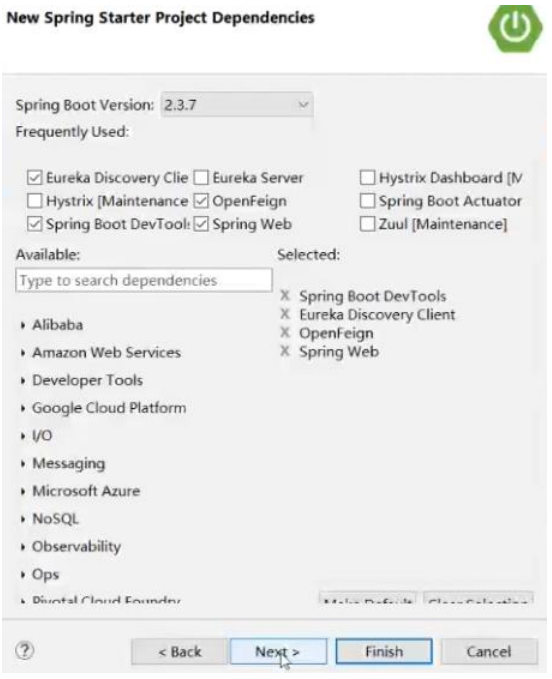
Feign 本身叫声明式调用，通过声明与客户端代理，使用简单的几个注解就能完成一个后端的微服务 API 调用工作，微服务框架本身它实际的设计思想是去简化微服务开发工程师的一个工作。

因为整个 Spring Cloud 体系，包含的数 10 种框架，都在不断的进行迭代发展，里面微服务客户端的调用工具还可以继续使用，但部分组件，有更多的选择，整个 Spring Cloud 生态是变得越来越丰富，越来越强大，利于去做一个多样化微服务架构的设计。

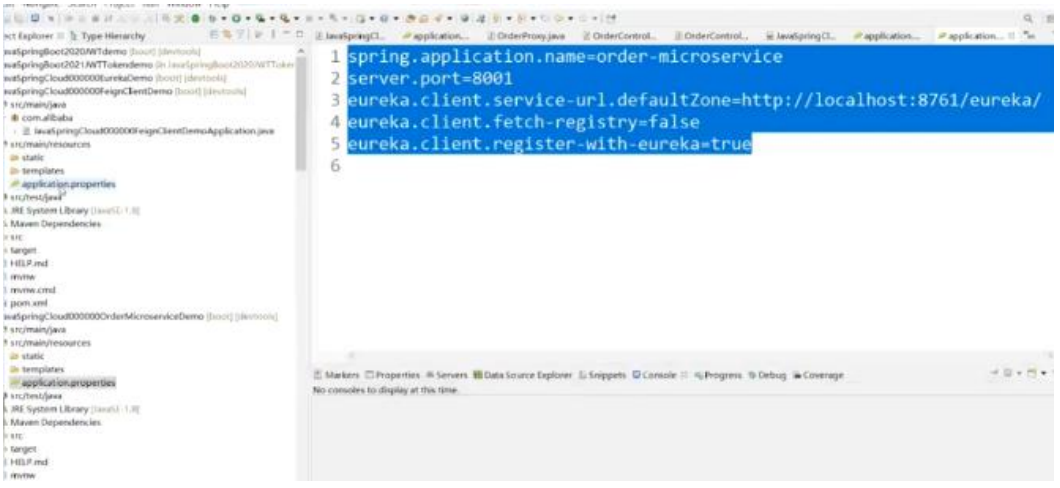
具体看 Feign 的使用，参考前几次微服务项目创建的经验，可以用 Eclipse、IDEA，也可以用在线的网站向导去生成一个 Spring Cloud 的一个项目。

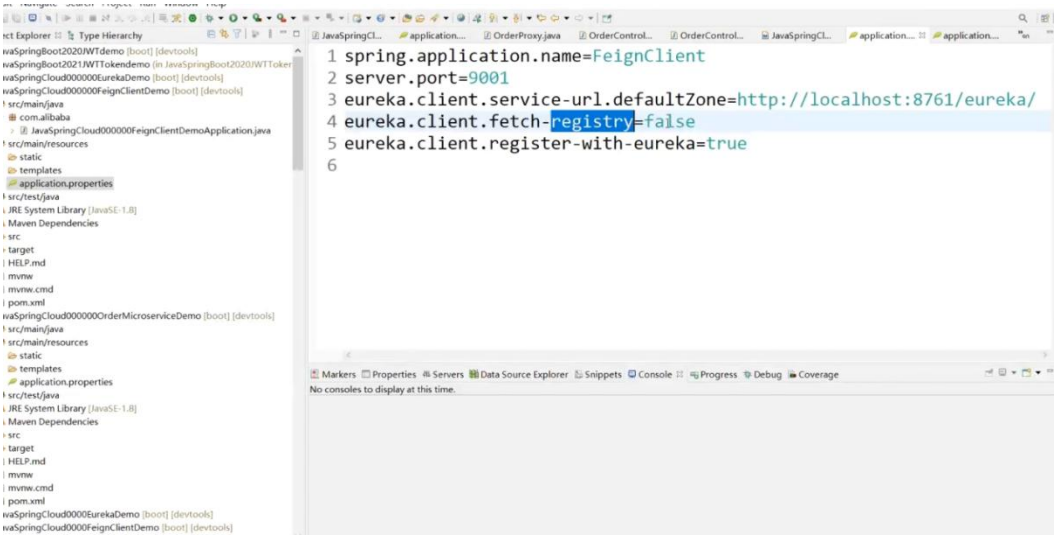
演示操作如下：


New Spring Starter Project
Service URL
https://start.spring.io
Name
Use default location
Type:
Maven
Packaging:
Java Version:
Language:
Java
Group
com.alibaba
Artifact
Version
0.0.1-SNAPSHOT
Description
Demo project for Spring Cloud
Package
com.alibaba
Working sets
Add project to working sets
New...
Next >
Cancel


New Spring Starter Project Dependencies
Spring Boot Version: 2.3.7
Frequently Used:
Eureka Discovery Clie
Eureka Server
Hystrix Dashboard [M
Hystrix [Maintenance
OpenFeign
Spring Boot Actuator
Spring Boot DevTools
Spring Web
Zuul [Maintenance]
Available:
Selected:
Type to search dependencies
Spring Boot DevTools
Eureka Discovery Client
OpenFeign
Spring Web
Alibaba
Amazon Web Services
Developer Tools
Google Cloud Platform
I/O
Messaging
Microsoft Azure
NoSQL
Observability
Ops
< Back
Next >
Finish
Cancel

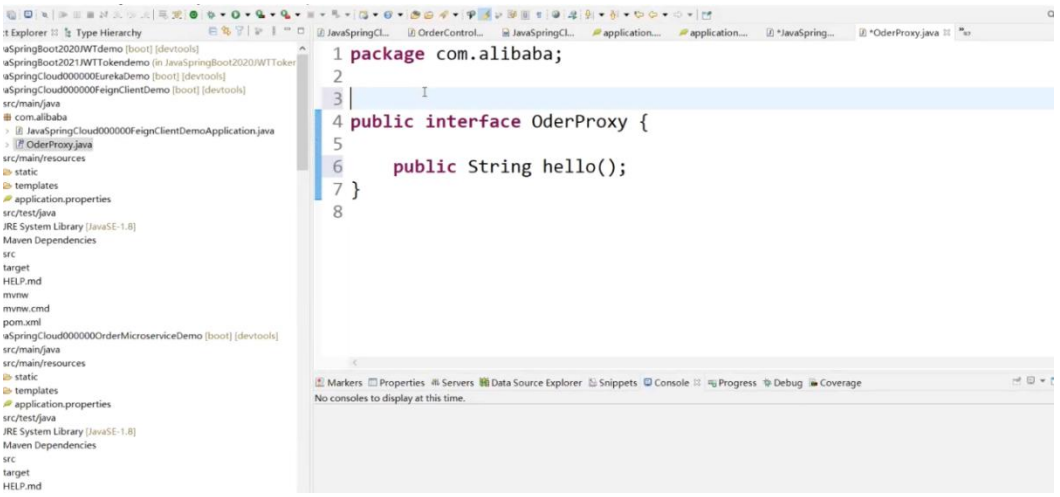
为了方便开发，要点选添加 Dev tool 式组件 web 项，主要作用是因为要调后台客户端 services，更多的时候希望包容 RecAPI 在本地，通过本地的软件 RecAPI 底层代理来调远端的后台的微服务，形成一个调动链,本地测试方便。还有 Eureka Clie，因为依赖的是 Feign 是调用端，Feign 一定要加进来,版本选 2.3.7 往后选，不用最新的，因为一般公司的项目迭代没有那么快，大部分企业可能还停留在二点零点几，特别新版本，企业是很难使用。



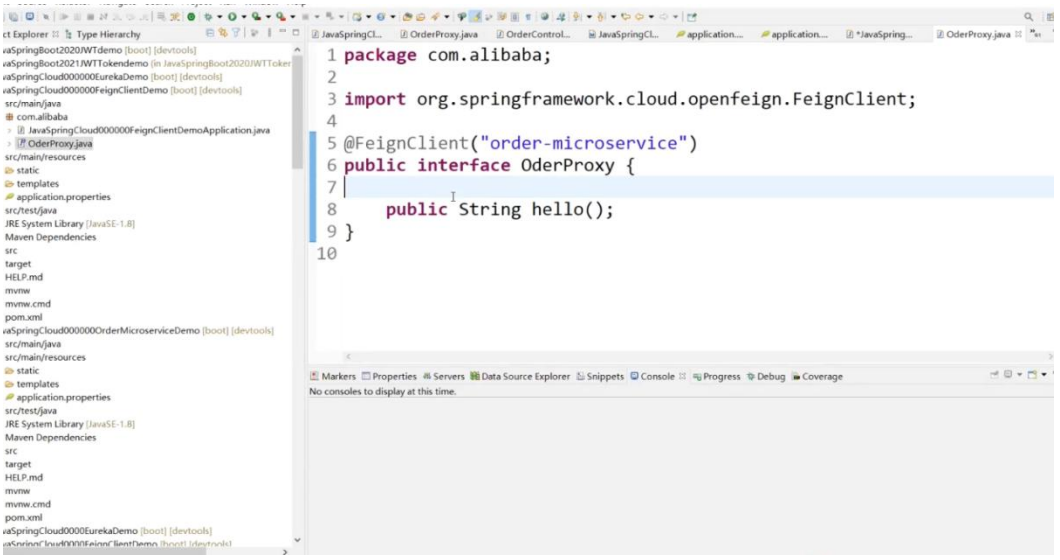


配置文件作为一个调用端，它实际和微服务端配置基本上差不多，直接复制微服务的，拷贝过来，端口因为都在同一台机器上，所以里面端口尽量不要重复，9001 表示调动端的代码，项目改成 Feign Client，端口 9001，要去注册中心查,因为要查找后台服务当前在线的微服务集群信息，地址也很重要，需要打开，还要加几个注解，作为客户端和数据中心首先打交道，接下来进行调用。

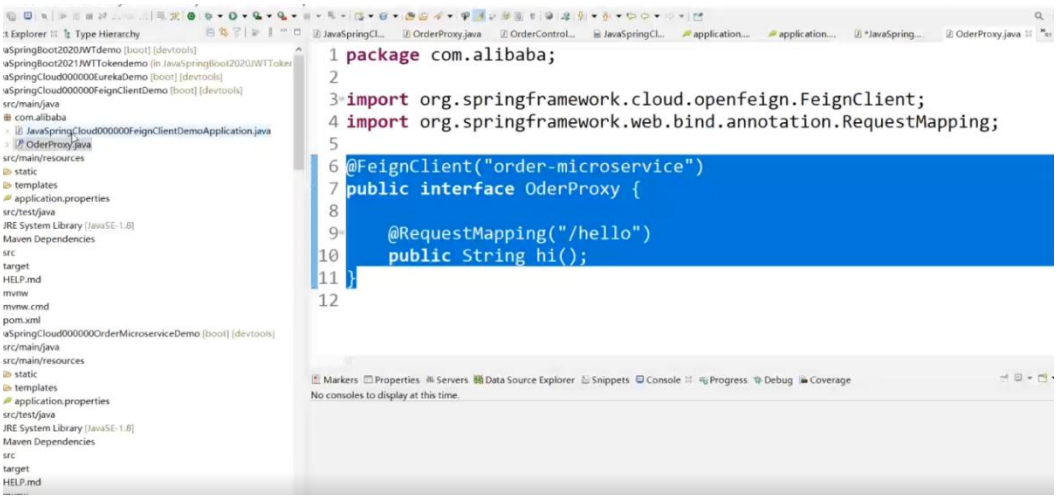
需要建一个代理接口包含一个声明，不包含方法体，具体如下：



声明一个客户端的接口，通过依赖注入，Spring 框架自动会给装配一个客户端的代理对象，然后通过代理对象来调查，还有 Feign Client 要给一个服务名，名字要统一，具体操作如下:



还有地址偏移要匹配，名字与服务端统一，不统一也可以允许一次再封装。

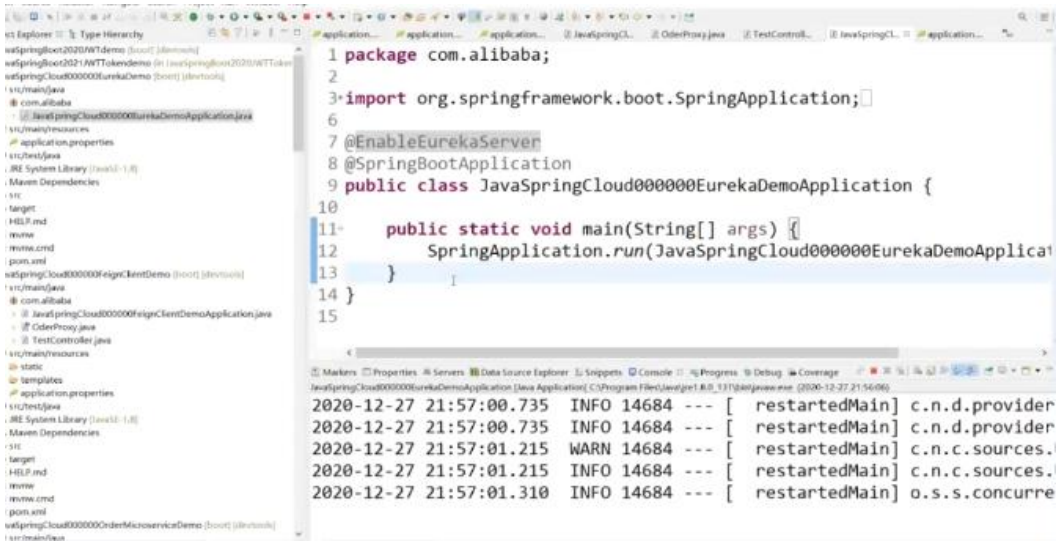


通过代理调用，浏览器测试，具体操作如下：

```java
import org.springframework.web.bind.annotation.RestController;

@RestController
public class TestController {

    @Autowired
    private OderProxy proxy;

    @RequestMapping("/test")
    public String hi() {
        return proxy.hi();
    }
}
```

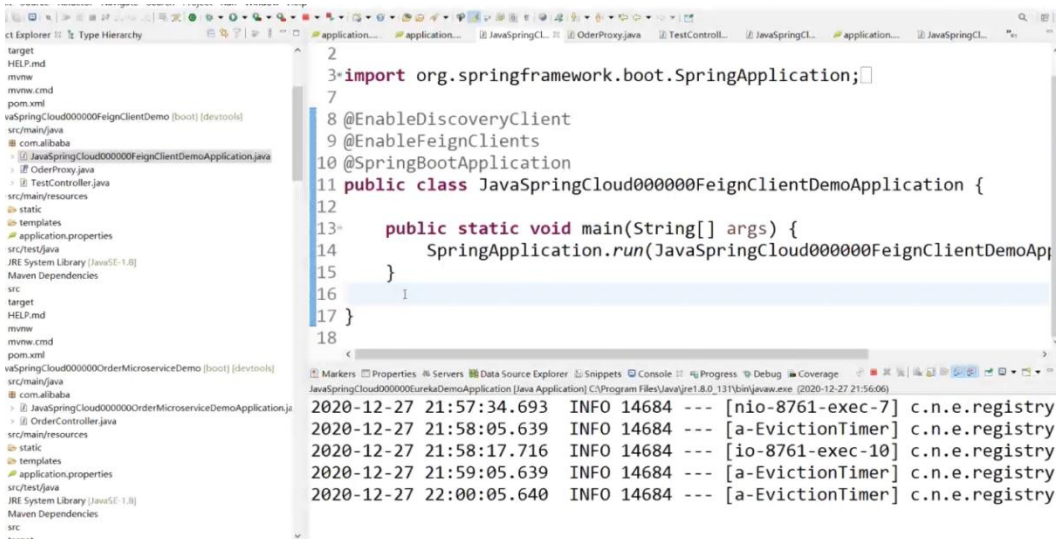
```java
package com.alibaba;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.RequestMapping;

@FeignClient("order-microservice")
public interface OderProxy {

    @RequestMapping("/hello")
    public String hi();
}
```

先启动注册中心测试保证微服务再到客户端：



```
1 package com.alibaba;
2
3 import org.springframework.boot.SpringApplication;
6
7 @EnableEurekaServer
8 @SpringBootApplication
9 public class JavaSpringCloud000000EurekaDemoApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(JavaSpringCloud000000EurekaDemoApplica
13     }
14 }
15
```

```
2020-12-27 21:57:00.735  INFO 14684 --- [  restartedMain] c.n.d.provider
2020-12-27 21:57:00.735  INFO 14684 --- [  restartedMain] c.n.d.provider
2020-12-27 21:57:01.215  WARN 14684 --- [  restartedMain] c.n.c.sources.
2020-12-27 21:57:01.215  INFO 14684 --- [  restartedMain] c.n.c.sources.
2020-12-27 21:57:01.310  INFO 14684 --- [  restartedMain] o.s.s.concurre
```



```
2
3 import org.springframework.boot.SpringApplication;
7
8 @EnableDiscoveryClient
9 @EnableFeignClients
10 @SpringBootApplication
11 public class JavaSpringCloud000000FeignClientDemoApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(JavaSpringCloud000000FeignClientDemoApp
15     }
16
17 }
18
```

```
2020-12-27 21:57:34.693  INFO 14684 --- [nio-8761-exec-7] c.n.e.registry
2020-12-27 21:58:05.639  INFO 14684 --- [a-EvictionTimer] c.n.e.registry
2020-12-27 21:58:17.716  INFO 14684 --- [io-8761-exec-10] c.n.e.registry
2020-12-27 21:59:05.639  INFO 14684 --- [a-EvictionTimer] c.n.e.registry
2020-12-27 22:00:05.640  INFO 14684 --- [a-EvictionTimer] c.n.e.registry
```

2.5 Spring Cloud 微服务 Ribbon 负载均衡算法

内容简介：

一、Ribbon 负载均衡与底层算法

二、Ribbon 负载均衡算法

三、Ribbon 负载均衡架构图

大家好，我们继续学习 Java Spring 微服务架构设计与实战系列课程。这节课给大家讲一下 Spring Cloud 负载均衡算法，这里面有个很重要的技术 Ribbon。也是麦飞公司贡献的微服架构体系里面很重要的一个技术组件。

一、Ribbon 负载均衡与底层算法

Spring Cloud 负载均衡器 Ribbon

1）Spring Cloud 客户端负载均衡器 Ribbon。

2）Ribbon 是 Netflix 发布的开源项目。

3）Ribbon 主要功能是提供客户端的软件负载均衡算法。

4）Ribbon 将 Netflix 的中间层服务连接在一起。

5）Ribbon 客户端组件提供许多配置如连接超时，重试等。

6）配置文件中列出后台所有的机器。

7）Ribbon 会自动（如简单轮询，随即连接等）去连接这些机器。

8）Spring Cloud 使用 Ribbon 实现 自定义的负载均衡算法。

基本微服务架构是经过注册中心的一个交互，找到自己的服务，然后完成一个匹配服务的调用。实际生产环境下的话，像订单服务，评论服务，支付服务等会存在多台服务器的情况，实际数量的话是不定数量的，会出现浪费服务器的情况。或者在高频的场景下，订单服务可能有 100 台、1000 台的情况，如果客户端不能把压力就能分散到后台的服务上的话，最大的问题就会出现在我们做后端服务的上，所以这里面的话会有一个集群和负载均衡的概念，负载均衡算法在大规模 web 集群中是非常常见的。这个时候就需要 Ribbon，Ribbon 是做微服务集群的一个负载均衡，来帮助我们客户端调用的时候，要体现出来这样一个叫均衡。

Ribbon 技术组件也支持几种主要的算法：随机、路由、权重、最小链接、IP 地址增加路由策略等。Ribbon 是在 Netflix 里是已经集成了，使用简单的轮巡策略，但不一定合理，在负载均衡大家稍微注意一下。官方也出了多个类似组件，可以在实习落地方案的时候多个选择。

二、Ribbon 负载均衡算法

Ribbon 负载均衡算法

1）默认规则：轮训 RoundRobin

2）简单轮询负载均衡（RoundRobin）

3）随机负载均衡 （Random）随机选择 UP 的 Server

4）加权响应时间负载均衡 WeightedResponseTime

5）区域感知轮询负载均衡（ZoneAware）

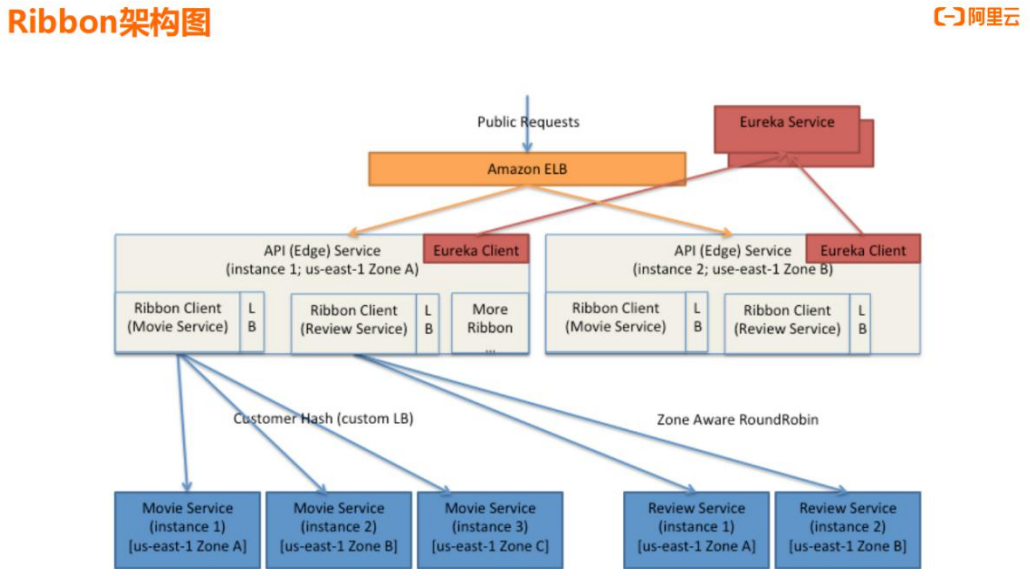
轮训是最常见的的一种算法，还有随机、权重也都是一种算法，区域感知是可用区域数据中心的可用性有关系。当跨这个数据中心或者多数据中心部署的时，就可以用上。根据客户端的 Ip 地址，就近 DNS 解析来录入到某个数据中心里面，是一种更优的跨多税中心地域的这样的一个负载均衡算法。一般的话用轮巡或者随机当然其实权重相对来说更合理一点。

Ribbon负载均衡算法 阿里云

策略名	策略描述
BestAvailableRule	选择一个最小的并发请求的server
AvailabilityFilteringRule	过滤掉那些因为一直连接失败的被标记为circuit tripped的后端server，并过滤掉那些高并发的的后端server（active connections 超过配置的阈值）
WeightedResponseTimeRule	根据相应时间分配一个weight，相应时间越长，weight越小，被选中的可能性越低。
RetryRule	对选定的负载均衡策略机上重试机制。
RoundRobinRule	roundRobin轮询选择server
RandomRule	随机选择一个server
ZoneAvoidanceRule	复合判断server所在区域的性能和server的可用性选择server

这些算法可以直接配置，或者用代码配置，配置的话一般我们用代码可以直接去替换。

三、Ribbon 负载均衡架构图



Ribbon 负载均衡关键点

1）服务发现，发现依赖服务列表 ServiceList

2）服务选择规则，多个服务中选择一个有效服务

3）服务监听，检测失效的服务，高效剔除失效服务

4）Ribbon 核心组件：

- Rule- 从服务列表中如何获取一个有效服务
- Ping- 后台运行线程用来判断服务是否可用
- ServerList- 服务列表

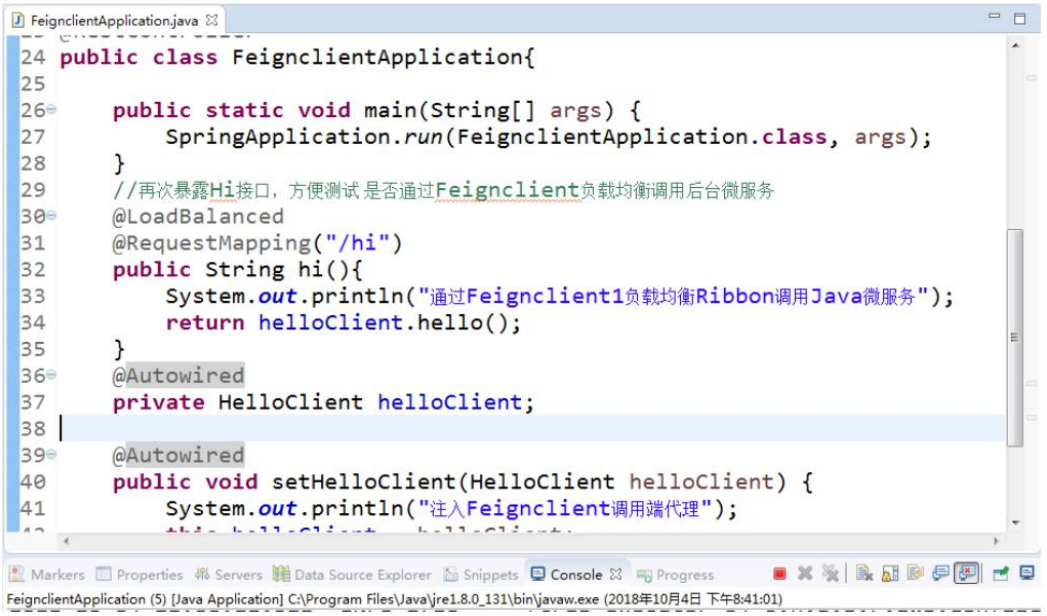
案例：

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

引用 Ribbon 包

```
@RibbonClient(name = "HelloSpringCloudMicroService", configuration = HelloServiceConfiguratio
public class HelloSpringCloudClientApplication {
    ////////////////// 客户端使用了负载均衡/////////////////////////////
    @LoadBalanced
    @Bean
    RestTemplate restTemplate() {
        return new RestTemplate();
    }

    @Autowired
    RestTemplate restTemplate;

    @RequestMapping("/hello")
    public String hi(@RequestParam(value = "name", defaultValue = "Rafael") String name) {
        String greeting = this.restTemplate.getForObject("http://HelloSpringCloudMicroService
        return String.format("%s, %s!", greeting, name);
    }
    ////////////////////////////////////////////////////////////////
    @Autowired
```

Spring Cloud Ribbon 负载均衡代码



@LoadBalanced

Feign 默认轮训

2.6 Spring Cloud 微服务 API 的监控 Hystrix

内容简介：

一、Spring Cloud 微服务监控

二、Hystrix 监控面板

一、Spring Cloud 微服务监控

1. Netflix Hystrix

1）Netflix 发布了 Hystrix 熔断器框架，保护系统

2）通过控制那些访问远程系统、服务和第三方库的节点

3）从而对延迟和故障提供更强大的容错能力

4）Fallback 灾备操作，出错以后返回的值

5）Hystrix 中，主要通过线程池来实现资源隔离

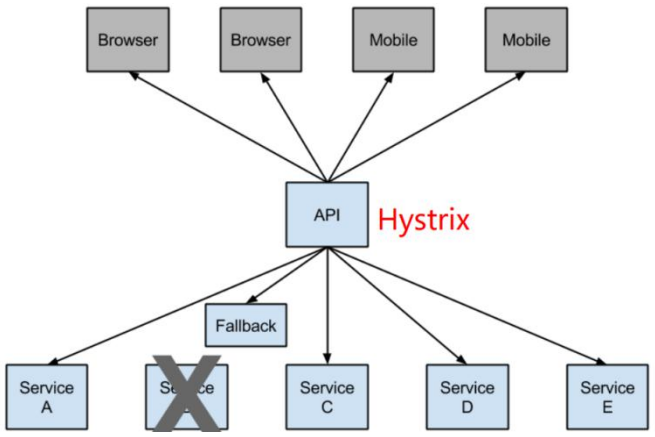
6）Hystrix 的信号模式(Semaphores)来隔离资源

7）Hystrix 支持 dashboard 控制面板 监控信息

8）Feign 可以和 Hystrix 结合使用，也可以独立使用

Hystrix 本义指的是豪猪，我们的微服务架构我们在生产环境下你有为了支持高并发高可用，你可能有 10 台甚至一 100 台 1000 台，微服的实力，但有一点也有可能比如说你有事态服务器的时候支持的并发，比如说是 1 万，但是双 11 的时候支持的并发可能瞬间达到了 10 万，能够去做一定的措施，我们去限制一部分的流量，然后服务其中的一部分流量。

2. 断路器模式



整个的熔断工具还属于是我们一般的话是放在服务调用端，因为一个服务端的话可能调用多个服务，所以在这一侧的话，我们做限流的话是比较方便。

二、Hystrix 监控面板

1. Spring Cloud Hystrix 熔断管理

1）Netflix ： 容断管理工具。

2）旨在通过控制服务和第三方库的节点。

3）从而对延迟和故障提供更强大的容错能力。

4）防止服务器过载。

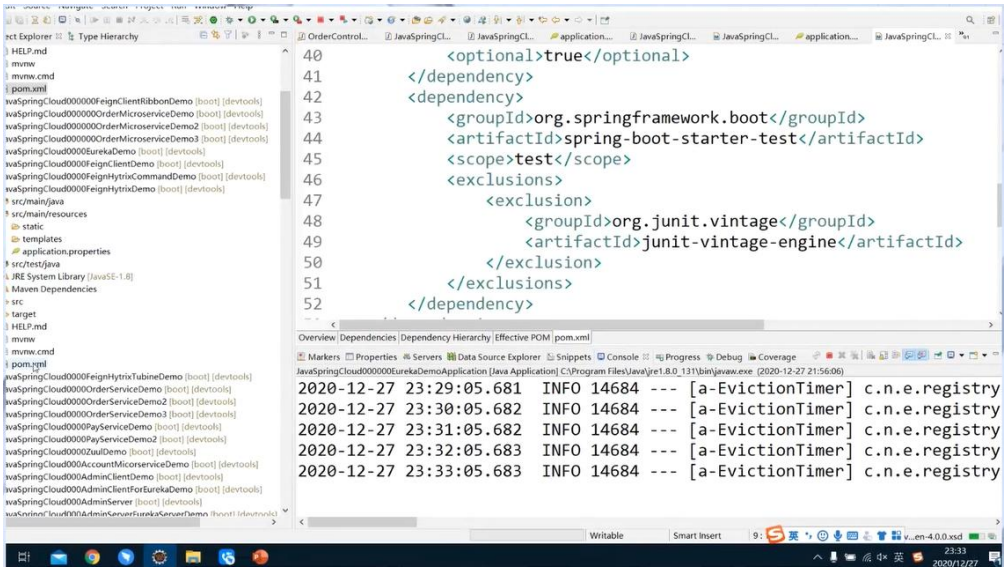
5）防止系统雪崩。

6）Spring Cloud Hoxton 版本后需要特殊配置。

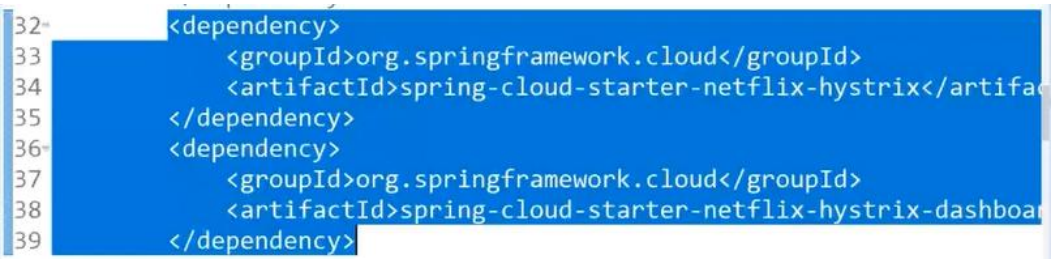
2. Pom 依赖

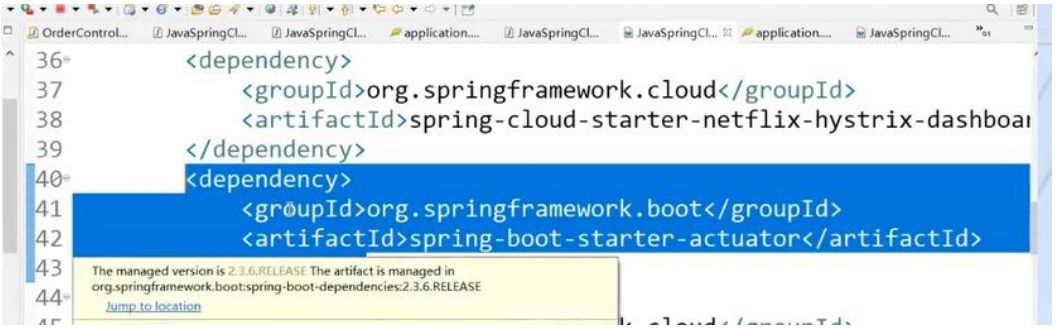
```
<!--远程调用-->
<!--熔断器-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-hystrix</artifactId>
        </dependency>
        <!--hystrix-dashboard 监控-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>
        </dependency>
```

练习：

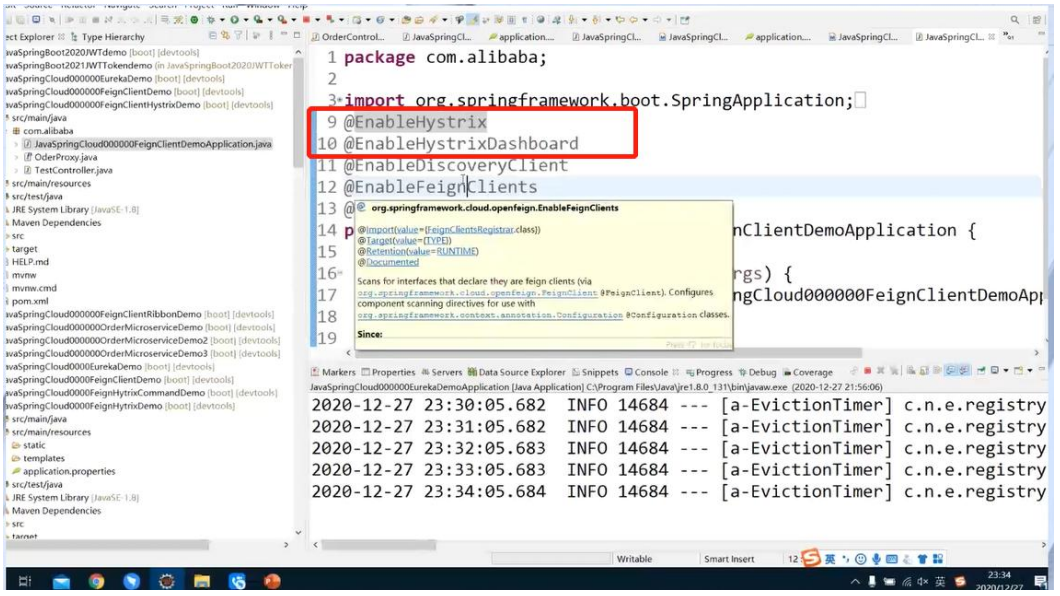


第一件事情加两个依赖，保存让他来拉一下依赖包。监控和采集用 hustrix，控制面板用 spring-cloud-starter-nutflix-dashboard</artifactid

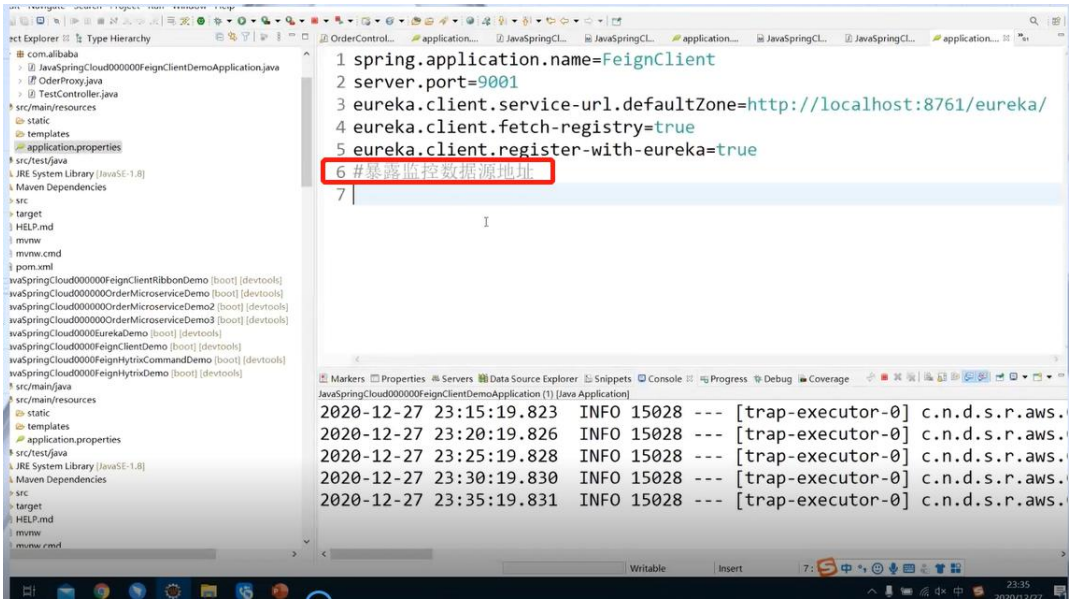




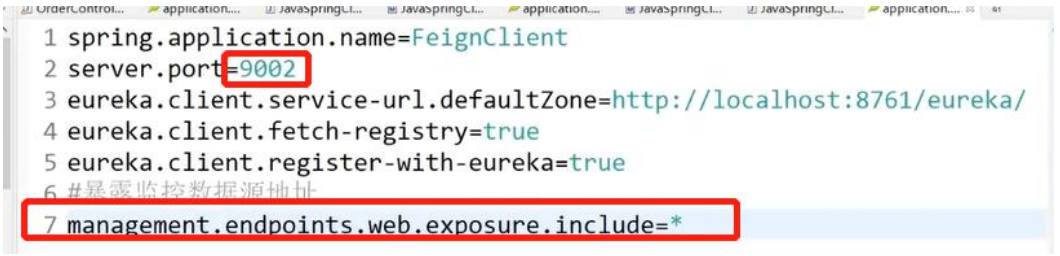
artifactId 依赖也要加进来是 2.0 提供的数据采集的组件用于暴露核心的应用程序的数据。



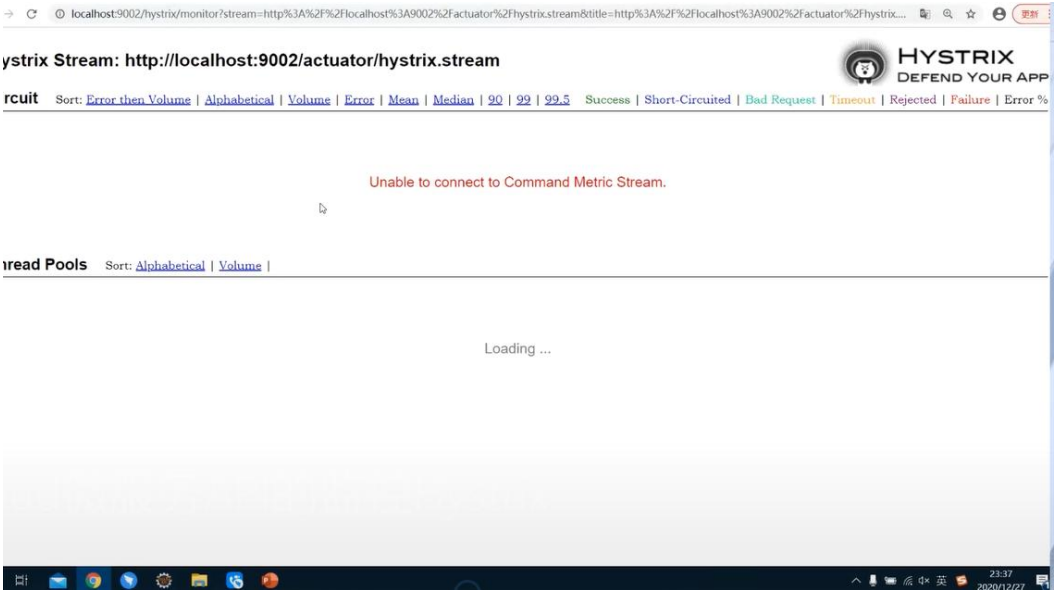
改完加两个重要注解 @EnableHystrix（启用监控面板），@EnableHystrixDashboard



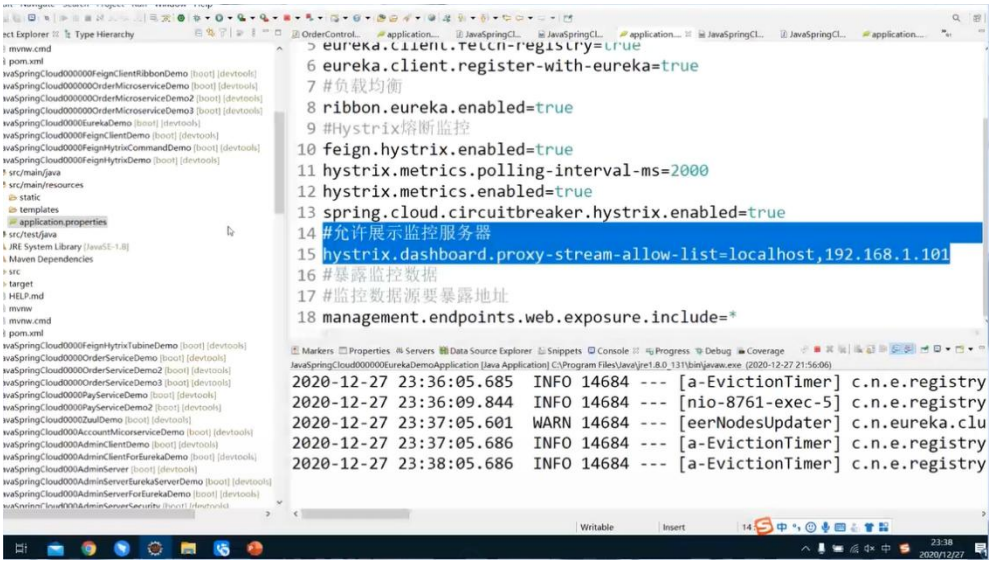
还要改配置文件用于暴露监控数据：



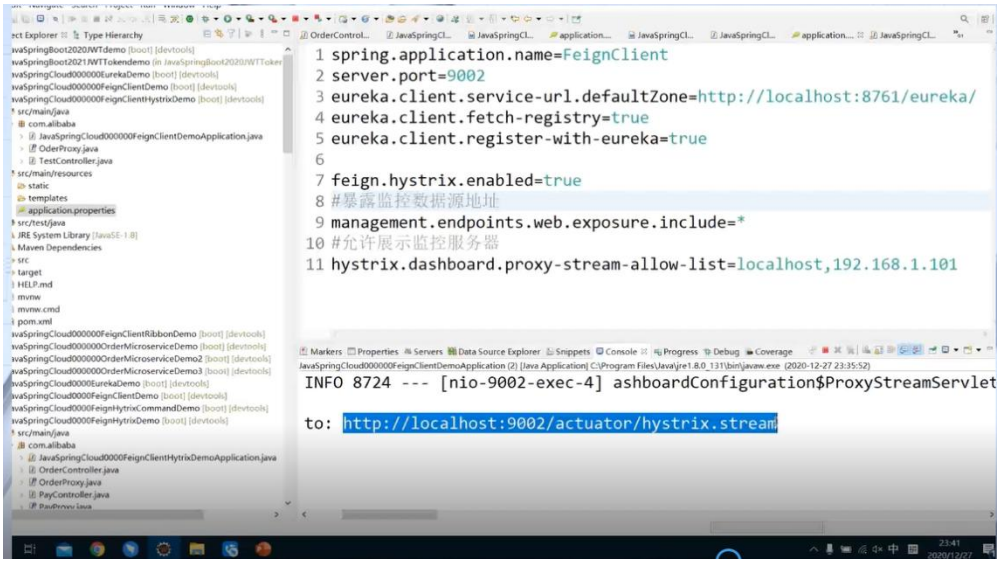
这里面的话把监控的 Excel 的地址给拿进来这里面我们要改个端口，咱们用 9602 来进行测试。



以后做监控后一定会遇到，叫无法链接到数据流。因为默认在 2.2 后版本修改默认的策略又改掉了，不允许你直接链接。



你要允许本地的服务器调用，采集面板跟代理服务器之间有个依赖关系。



代理服务器在想办法去监控它的数据流，采集它的数据流，处罚一下，就会有数据流显示。

3. 启用 Hystrix

1）• @SpringBootApplication

2）• @EnableEurekaClient

3）• @EnableFeignClients

4）• @• @• public class SpringCloudEurekaConsumerApplication { • public static void main(String[] args) {

5）•

6）SpringApplication.run(SpringCloudEurekaConsumerApplication

7）.class, args);

8）• } • }

4. Hystrix

- 监控服务调用
- 熔断代码
- 监控面板：独立部署，也可以 Zuul。

5. 开发步骤

1）依赖包 Hystrix

2）启用注解

3）修改配置参数，监控指标采集

4）打开监控面板 http://localhost:9001/hystrix

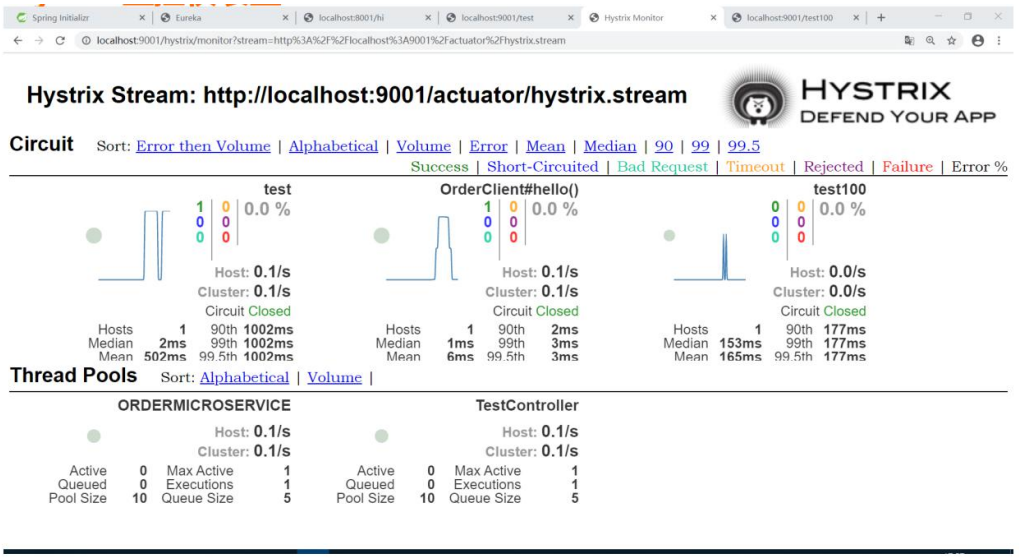
5）监控数据源

http://localhost:9001/actuator/hystrix.stream

6）调用一次微服务

7）修改熔断代码

6. Hystrix 监控仪表盘



7. Hystrix 底层原理

- Hystrix 使用了命令模式，
- 对命令对象抽象了两个抽象类：
- HystrixCommand 和 HystrixObservableCommand 。

2.7 Spring Cloud 微服务 API 的 Hystrix 熔断限流降级

内容简介：

一、Java Spring Cloud 熔断限流

二、Java Spring Cloud 熔断限流实战

咱们这节课看一下如何实现熔断降级和限流。上节课的话我们已经讲过基于 Hystrix 组建的来做启动监控面板，并且进行数据流的采集，这里面也给大家讲了一些在实战过程中的一些特殊的坑，比如我们说在 2.3.几以后，这个版本实际是在默认的安全性方面又做了加强，不能够私自去请求监控的数据源，它要求进行特殊的参数设置，你要允许监控面板去链接某个数据源才行，这个参数我们当时也演示了。

一、Java Spring Cloud 熔断限流

1. Netflix Hystrix

1）Netflix 发布了 Hystrix 熔断器框架，保护系统

2）通过控制那些访问远程系统、服务和第三方库的节点

3）从而对延迟和故障提供更强大的容错能力

4）Fallback 灾备操作，出错以后返回的值

5）Hystrix 中，主要通过线程池来实现资源隔离

6）Hystrix 的信号模式(Semaphores)来隔离资源

7）Hystrix 支持 dashboard 控制面板 监控信息

8）Feign 可以和 Hystrix 结合使用，也可以独立使用

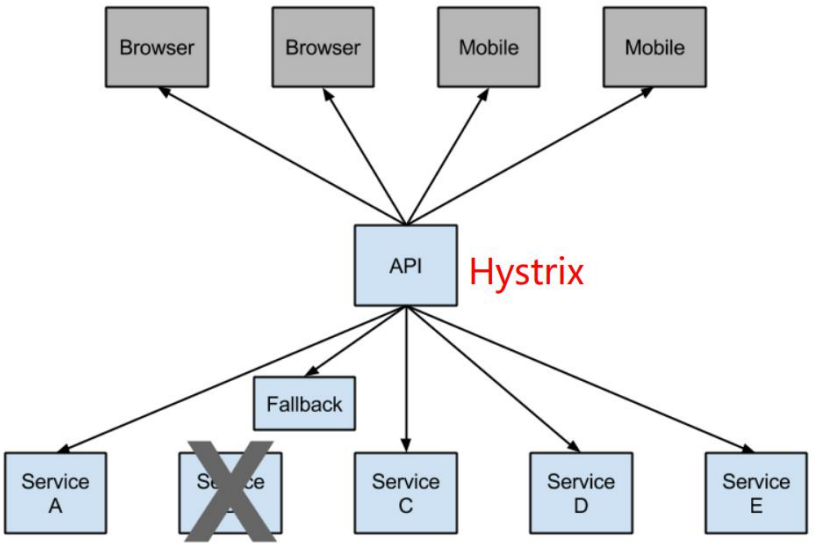
9）Hystrix 使用了命令模式，对命令对象抽象了两个抽象

10）类：HystrixCommand 和

11）HystrixObservableCommand

接下来咱们来看一下如何去做熔断，熔断是对于高并发系统来保证它的高考用性的时候，来采用了一个很重要的措施。咱们比较常见的一种场景就是淘宝的双 11，比如我们讲了大家如果做了一个高频化系统的话，如果你的服务器机群理论上比如说支持每秒 1 万的并发，你现在的话希望在双 11 的时候，如果流量超过 1 万，比如达到 2 万，就是说我不希望系统直接瘫痪。服务器直接全部崩溃，是这种场景的话，我们说这种结果不是我们期望的，咱们找个解决办法，就是我们说的要启用一个保护措施，熔断器模式，这里面的话也叫断路器模式 b

2. 断路器模式



这个其实在淘宝双 11 的时候体现的比较明显，如果各位有印象的话，参加过之前的双 11 的话，早期双 11 的时候淘宝的服务器容易卡，京东也一样也容易卡。为什么说白了早期的话没有见过电商公司的话是没有经历过这种如此高规模的这种并发流量，包括另外一个 12306 火车票网站也是一样的，现在的话加上限流，现在的话起码不会让服务瘫痪，相比谈话来说我们起码还能够去处理其中的比如说一部分的请求或者部分我可以继续处理请求。所以这种场景大家在生活中也经常看到，比如说北京的单双号限行，上海的也开始启用限行了，高峰时间限行，包括外地牌以后也不能进内环地面了，这里面其实这都是限流的一个措施。剥离一些基于某些策略，我们说剥离一些请求压力，另外来保护我们的服务器，使服务器能够以较低的一个并发来进行正常的处理请求，早期 Netflix 也实现了一个关键性的技术叫 Hystrix。

二、Java Spring Cloud 熔断限流实战

1. Hystrix 核心参数

•请求最大次数

• circuitBreaker.requestVolumeThreshold（默认值：20 个请求）

•滚动窗口

• metrics.rollingStats.timeInMilliseconds（默认值：10 秒）

•失败百分比

• circuitBreaker.errorThresholdPercentage（默认值：> 50%）

我们 Hystrix 的时候有控制面板，它还可以监控面板还可以进行数据采集，这个数据采集了很重要的一点和熔断降级限流有关系，熔断以后我们说可以降低，比如说之前并发是 1000，现在的话可以降到 500，之前是 1 万我可以降到 5000，起到一个限流的保护的作用。你可以基于最高并发、最大并发率、最大并发量达到这个时候是我开始消峰，把峰值给消掉。

2. Spring Cloud 熔断限流实战

```
    // timeoutInMilliseconds：设置熔断超时的时间
    @HystrixCommand(fallbackMethod = "fallbackMethod", commandProperties = {
            @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "2000")
    })
    @RequestMapping("/test")
    public String test() {
        String result = orderClient.hello();
        return result;
    }

    // timeoutInMilliseconds：设置熔断超时的时间
    @HystrixCommand(fallbackMethod = "fallbackMethod", commandProperties = {
            @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "2000"),
            @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "2"),
            @HystrixProperty(name = "metrics.rollingStats.timeInMilliseconds", value = "500"),
            @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value = "1"),
            @HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value = "1000") })
    @RequestMapping("/test100")
    public String test100() {
        String result = "100次调用";
        for (int i = 0; i < 100; i++) {
            orderClient.hello();
        }
```

还有就是我们说的是如果某个服务出错出的特别多，我把它给断掉，你把它给断掉，请求不要再发了再发的话出更多的错误。然后比如说是出错率达到 10%，还达到 100%，你还达到 90%的时候我开始熔断。另外的话就是我们服务接口时间过久，它的调用时间长度超过了 10 秒，达到 10 秒的时候就不要再去请求他说白了一个问题调动时间太长了，对这个客户端来说没有太大意义因为我们讲了整个的一个服务，如果调研过程的话，一定是哪个地方出了环节，比如数据库思索，或者你的缓存次数都有可能，或者说你的代码出现这个问题，比如说大面积出现，我们说这种注册的请求，可能都有这几个参数很重要。

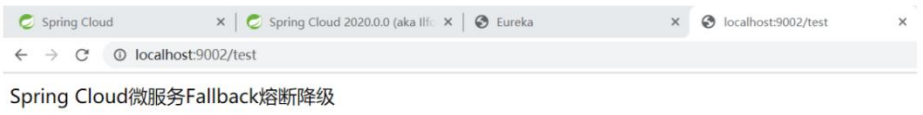
实战：添加设置熔断超时的时间

```
5 utInMilliseconds: 设置熔断超时的时间
6 Command(fallbackMethod = "fallbackMethod", commandProperties = {
7 @HystrixProperty(name = "execution.isolation.thread.timeoutInMillise
8
```

设置处罚超时时间 3S

```
21     public String getOrder() throws InterruptedException {
22         Thread.sleep(3000);
```

3. 熔断降级



每个公司的服务的并发量都不一样，服务器的配置也不一样，所以限流策略一定要根据业务根据你的架构设计的目标进行灵活调整。主要目的是要保护增加它的要可用性，增加了可用性，高并发我们说大家都期望，但是这个不可能无限高并发，每个机器的话一定有有它的一个我们说叫天花板一定会有它的一个瓶颈，所以我们在一个根据我们实际的配置，压测的结果咱们可以配一个保护阀子，保护阀子用于保护我们的系统，在达到这个瓶颈后保护阀子会进行工作从而实现熔断。

2.8 Spring Cloud 微服务网关代理 Zuul

内容简介：

一、Java Spring Cloud 网关 Zuul

二、Spring Cloud 网关 Zuul 实战

一、Java Spring Cloud 网关 Zuul

本节课讲的是微服务架构非常重要的技术“Zuul 网关”，现在已经迭代到 2.0 版本，架构底层也发生了较大的变化。微服务集群为什么需要网关，因为服务特别多，尤其是对外的时候，作为调用端，不可能知道每台服务的实例地址，中间有注册中心，实例地址会动态变化，所以希望有统一的出口，这就是网关的作用。

网关是两个网络边界的通道，相当于请求转发，就是代理服务器。

1. Spring CloudZuul 网关

Zuul 是 Netflix 开源的微服务网关工具，Netflix 公司在微服务领域贡献了早期的核心代码，当然 2020 年之后 Zuul，包括其他的一些 Spring Cloud 的组件，主要在维护阶段与 Spring Cloud 官方产生了分歧，但是仍然可以使用，而且功能也比较完善，包括实名验证、日志、路由定制等功能。2.0 版本的功能更加强大，因为后面考虑到优化，比如连接池等相关功能的扩展。

Spring Cloud 官方也出了类似 Zuul 的 Gateway 技术，Gateway 技术基于 Spring Framework 5 底层框架实现的，吞吐量方面有比较高的提升。但是使用场景要看实际情况，绝大部分公司的并发都不会超过 1000，很少有超过 10000 万的公司，因为中国互联网公司不是各个都是 BAT，所以会有差异。

Spring CloudZuul 网关总结：

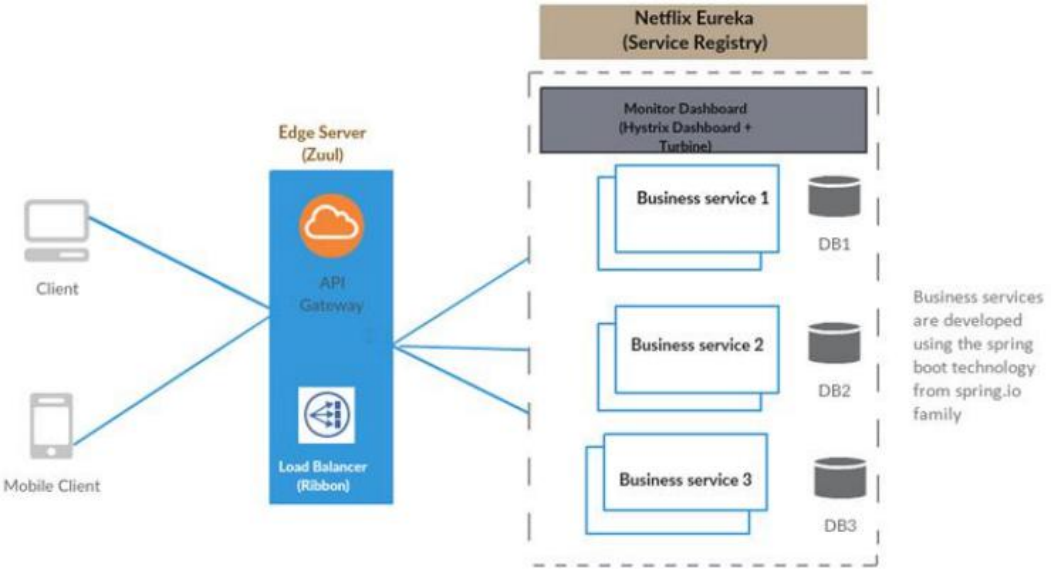
1）Zuul 是 Netflix 开源的微服务网关工具；
2）可以和 Eureka、Ribbon、Hystrix 等组件配合使用；
3）Spring Cloud 对 Zuul 进行了整合与增强，
4）Zuul 旨在实现动态路由，监视，弹性和安全性。
5）Zuul 默认使用的 HTTP 客户端是 Apache HTTPClient；
6）也可以使用 RestClient 或 okhttp3.OkHttpClient；
7）Zuul 默认会为 Eureka 注册的服务创建动态路由；
8）zuul 在 2.x 甚至 3.x 的分支中已经引入了 netty；
9）github 地址:https://github.com/Netflix/zuul；
10）官方文档:https://github.com/Netflix/zuul/wiki。

2. Spring Cloud Zuul 网关架构

网关做为代理端，后面挂接微服务集群，熔断、监控、身份验证等都可以放在代理层做，Zuul 出口对外可以做微服务代理模式，前面可以连接移动 APP 等，后面数据库设计，有一种模式叫一个微服务一个数据库，还有现在国内还比较流行的分库分表，有没有把数据库拆开，要不要把数据库拆开以后再分表。这个问题变得越来越复杂，不同的

公司不同的业务量，规模都会有差别。

当然好处是里面的组件都可以替换，不会出现多个相同的组件，比如注册中心，可以替换成阿里开源的，国内很多人喜欢阿里的技术。阿里在 Java 这块是国内贡献最大的一个公司，如果阿里不在淘宝上使用 Java，有可能 Java 在国内没有这么大市场，架构不会得到如此广泛的实践性验证。现在无论是 Spring Clord 还是 Dubbo 技术，都是很好的技术选型。



3. Zuul 网关特性

Zuul 首先是作为代理层作为网关，路由、负载均衡、日志等都是扩展功能，主要的作用是把请求转发过去。

1）Authentication and Security –验证和安全；

2）Insights and Monitoring –跟踪、统计、监控；

3）Dynamic Routing –动态路由 消息到 后台集群；

4）Stress Testing – 压力测试 逐级递增；

5）Load Shedding –过载保护；

6）Static Response handling – 静态消息处理，无需后台集群服务器处理；

7）Multiregion Resiliency – 多区域弹性伸缩，跨 AWS 区域路由请求，分散压力，请求处理更接近调用者。

二、Spring Cloud 网关 Zuul 实战

1. 创建 Zuul 代理

首先引入 Zuul 的依赖包，其次需要和注册中心进行交互，要知道那些服务可用，来代理可用的服务，是微服务集群的统一入口。

- 首先创建一个 Spring Boot 项目，
- 然后引入 Zuul 相关依赖。
- <dependency>
- <groupId>org.springframework.cloud</groupId>

<artifactId> spring-cloud-starter-zuul</artifactId>

- </dependency>
- <dependency>
- <groupId>org.springframework.cloud</groupId>

<artifactId> spring-cloud-starter-eureka</artifactId>

</dependency>

- 添加注解@EnableZuulProxy;
- Zuul 内部使用 Ribbon 实现客户端负载均衡。

2. EnableZuulServer

为了方便启动微服代理应用,Zuul 提供了 EnableZuulProxy 的快速注解，里面有组件默认的集成，直接在 POS 里实现。

- @EnableZuulServer 只启动代理服务；
- @EnableZuulProxy 简单理解为@EnableZuulServer 的增强版，
- 当 Zuul 与 Eureka、Ribbon 等组件配合使用时，
- 使用@EnableZuulProxy。

3. Spring Cloud Zuul 网关底层优化

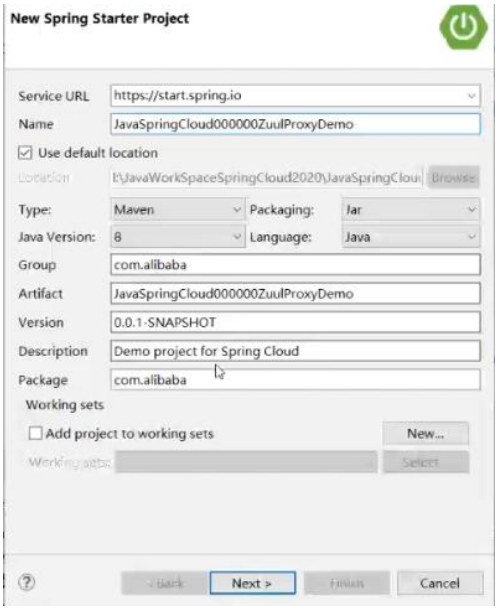
1）RestClient 或 okhttp3.OkHttpClient；

2）Zuul 默认会为 Eureka 注册的服务创建动态路由；

3）zuul 在 2.x 甚至 3.x 的分支中已经引入了 netty。

实战演示：

之前做的项目不动，里面有微服务、注册中心、调用端。打开 Eclipse 开发工具，新建项目，Name 栏直接加上：ZoolProxyDemo。



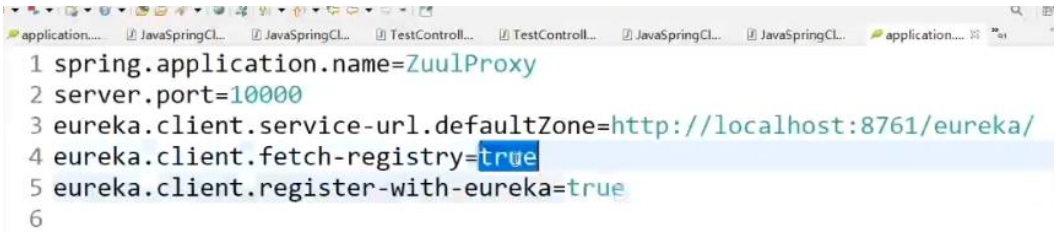
下一步，Frequently Used:勾选上 Eurka Discovery Clie、Spring Boot Devtools、Zuul[Mainternance]，版本选 2.3.7。

下一步，直接上线。

稍等，会有一个构建过程，下一步加注解，@EnableZuulProxy。注意中间不能中断，包一旦出错很难找回。

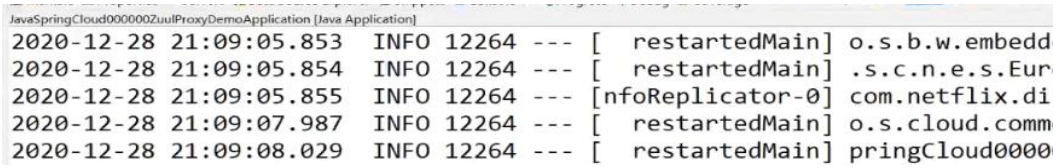
```
1 package com.alibaba;
2
3 import org.springframework.boot.SpringApplication;
6 @EnableZuulProxy
7 @SpringBootApplication
8 public class JavaSpringCloud000000ZuulProxyDemoApplication {
9
10     public static void main(String[] args) {
11         SpringApplication.run(JavaSpringCloud000000ZuulProxyDemoAppl
12     }
```

因为 Zuul 配置和调用端很像，直接复制调用端的配置改成 Zuul。Name=ZuulProxy,端口=1000，详情配置如下图所示：

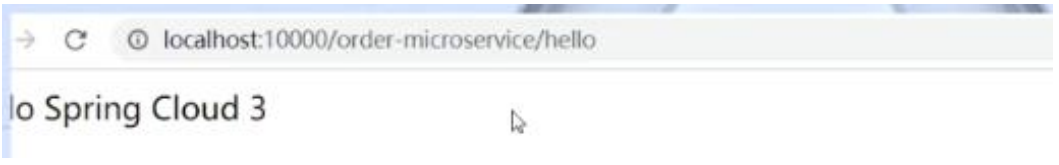


默认连接注册中心以后，会把所有的服务列表全部拉过来，默认形成一个路由策略，会基于服务名、方法名来形成一个路由规则，当然用户也可以改路由规则，可以定制，也可以改配置。暂时这里不改，后面作为扩展学习的时候再深入讲。

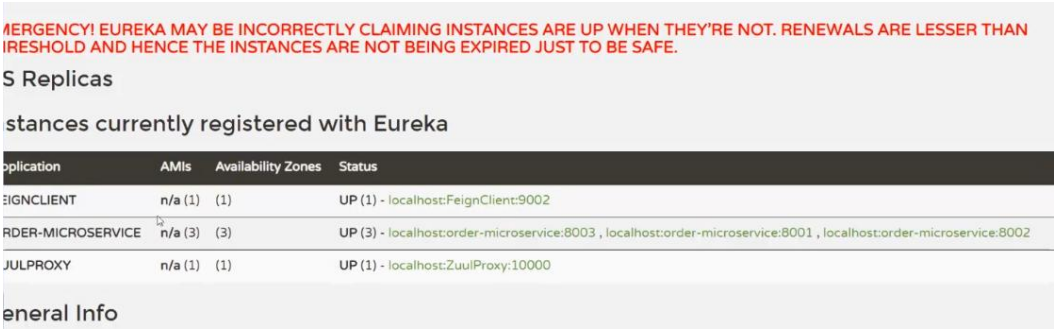
接下来启动程序，直接右键，选 Run As ,有两种启动方式，1.jave Application，2. Spring Boot APP，两种都可以启动。



启动成功之后，打开浏览器，输入 http://localhost:10000/order-microservice/hello 回车,显示已经调用成功。



看一下访问规则，端口是：10000；服务端是：hello；注册中心是：order-microservice。现在的路由规则是基于服务名和方法名调用。



通过代理服务器（网关），输入服务名字加方法名就可以了。当然也可以定制特殊的规则，这节课不演示。做演示会比较简单，主要是把整个调用链跑通，告诉大家整个微服务架构的搭建过程。

2.9 Spring Cloud 微服务的身份验证与安全机制

内容简介：

一、Java Spring Cloud 身份验证与安全机制

二、Java Spring Cloud 实战令牌身份验证 SSO

一、Java Spring Cloud 身份验证与安全机制

1. 面试题：ZuulFilter 过滤器

1）Zuul 自定义过滤器处理请求

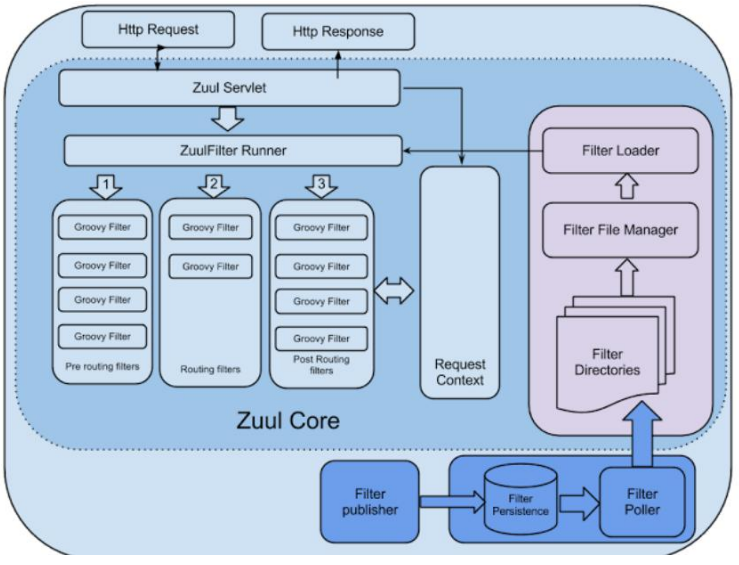
2）filterType：zuul 中定义了四种不同生命周期的过滤器类型：

- pre：可以在请求被路由之前调用
- routing：在路由请求时候被调用
- post：在 routing 和 error 过滤器之后被调用
- error：处理请求时发生错误时被调用

3. filterOrder：通过int值来定义过滤器的执行顺序，越小优先级越高
4. shouldFilter：返回一个boolean类型来判断是否要执行，
5. run：过滤器的具体逻辑。
6. ctx.setSendZuulResponse(false)令zuul不对其进行路由，
7. ctx.setResponseStatusCode(401)设置了其返回的错误码，
8. ctx.setResponseBody(body)对返回body内容进行编辑等。

网关底层比较有用的一点除了请求转发以外另外一个就是咱们有可能会做日志或者做身份验证因为它是统一的入口。就像高速公路一样，城市只有几个高速路口，入口是包括飞机场，火车站那边都是人流量入口的地方，他来做防疫检查是安全检查最可靠的一个地点，这里面也是要在这里来分析一下它底层的一个原理，然后通过实战看这个结果。这有一个面试题 ZuulFilter 本身是有过滤器的概念，过滤器的概念，分四种，主要用的是第一种，提前拦截请求来做请求的合法性校验然后如果用户是登录请求咱们得给它生成一个令牌返给他。如果不是登录请求的话就要检查它的合法身份证的有效性一般的话是通过令牌机制来做。

2. Zuul 核心模块



这里面当中最重要的是 Filter 机制，Filter 机制过滤器机制我们用它主要是做设备验证一个是登录的时候让他去生成令牌合法调研如果登陆成功来完成一个令牌生成如果登陆失败的话就不放行了。

3. 微服务身份验证

1）/authentication 访问这个地址验证

2）验证用户名密码，生成 Token，返回

3）后续客户端调用 Zuul，拦截器拦截 Token

4）比对，时间周期 2 小时，Token Server

5）过滤器 Filter，拦截请求

如果用户现在携带着令牌过来，然后我们验证它有效期或者验证这个令牌是不是正确。当用户登录的时候，校验盘用户密码的时候可以生成令牌，比如放在一个公共的人事缓存里面。当下一次效验那些请求的时候如果他访问的不是登录页面我们拦截他然后拿他的令牌做身份的校验。验证令牌服务器里面是不是有这个令牌是不是合法，是不是过期了。

二、Java Spring Cloud 实战令牌身份验证 SSO

1. Zuul 启用验证

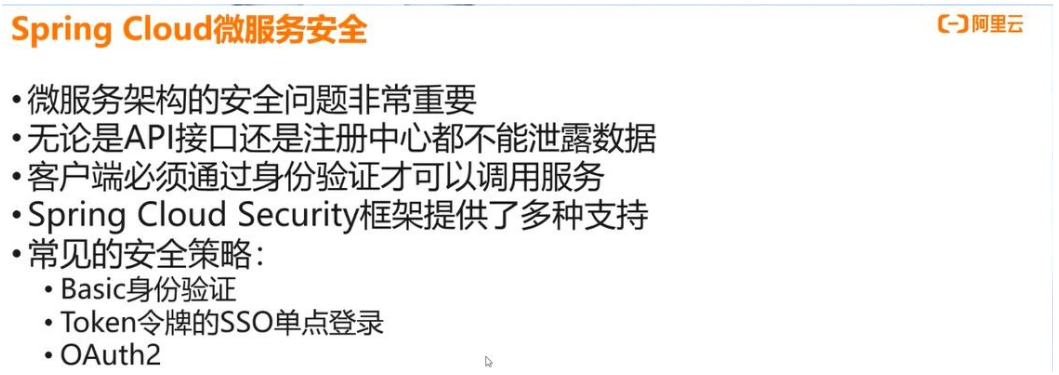
•@SpringBootApplication

•@EnableOAuth2Sso

•@EnableZuulProxy

• class Application {

•}



2. Spring Cloud 微服务集成令牌

1）客户申请令牌

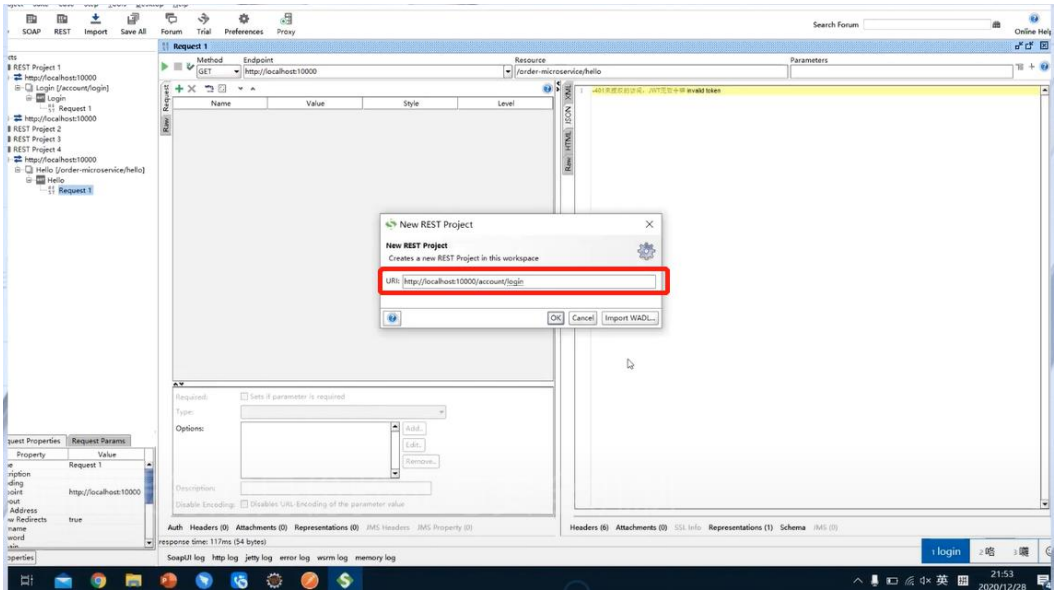
2）令牌服务器令牌签发

3）客户端调用 API，附带 Token 值

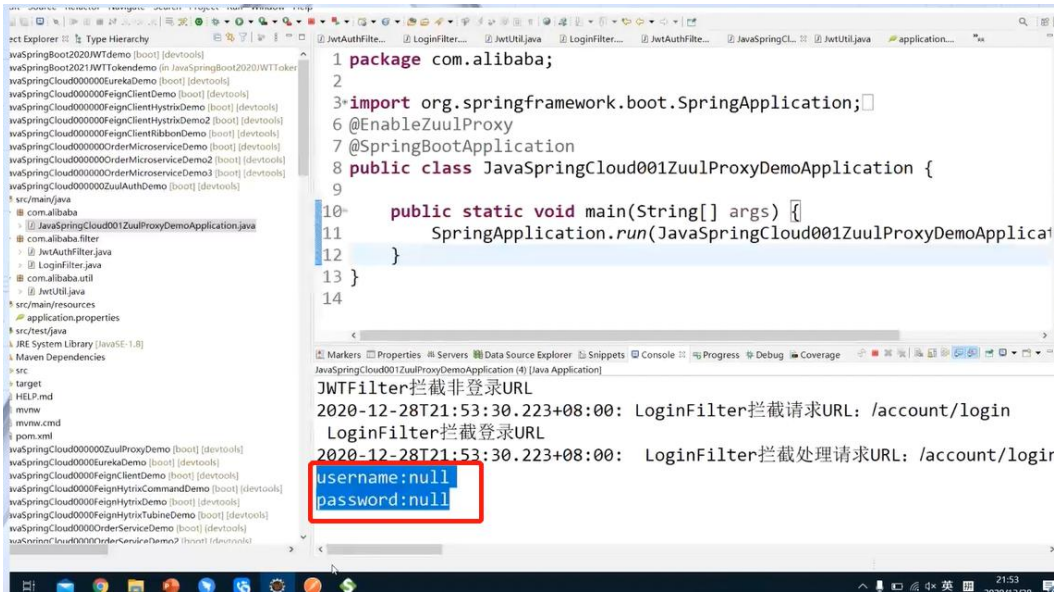
4）Zuul 代理拦截请求

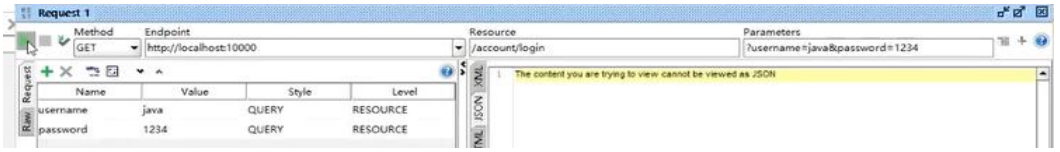
5）验证 Token 验证

注意事项：



发送请求当访问地址出现无效时，添加 accunt/login，Request 加两个参数，username，password。





返回 Request 添加 token，以及后台反馈 jwt 数值，放入到 HEADER 里加入令牌词。

通过令牌方式成功的完成了一个令牌客户端的登录令牌的校验用的是主网关里面很重要的一个过滤器，而且过滤器选择了优先级要高的在请求处理之前就开始拦截，就开始进行身份证校验，并且主要校验是基于 token。

2.10 Spring Cloud 微服务集群 Monitor 监控中心

内容简介：

一、Java Spring Cloud 微服务实例监控

二、Java Spring Cloud 微服务实例监控实战

一、Java Spring Cloud 微服务实例监控

各位同学大家好，咱们现在这节课继续学习 Java Spring Cloud 的微服务架构实战课程，这节课来讲一下微服务的监控问题，如何去监控服务的内存、运行状态。之前的看的监控相对来说偏流量，现在需要一个工具，它能够监控每个程序的运行状态，以及它的内存、它的对象、线程数量等，基础的 Java 底层的监控。

应用程序分布系统中是属于比链路追踪更重要的内容，链路追踪偏链路，而我们要的追踪（监控）偏应用的状态和内存数据结构，这也是很重要的一部分数据信息，也可以把两个的数据全部给综合起来。现在目前开发社区工具比较多，Spring Cloud 本身也提供一个比较有用的组件:Spring Boot Admin（由德国软件工程师 Johannes Edmeier 开源）

Spring Boot Admin

1）Spring Boot Admin 是用于管理和监控 Spring Boot 应用程序。

2）由德国软件工程师 Johannes Edmeier 开源。

3）已经被收纳入 Spring Initializr。

4）截至发文时间的最新正式版本为 2.1.6 ，快照为 2.2.0-SNAPSHOT。

5）C/S 架构风格 。

6) 应用程序作为 Spring Boot Admin Client 向 Spring Boot Admin Server0 注册（通过 HTTP）。

7）或使用 Spring Cloud 注册中心（如 Eureka，Consul）发现。

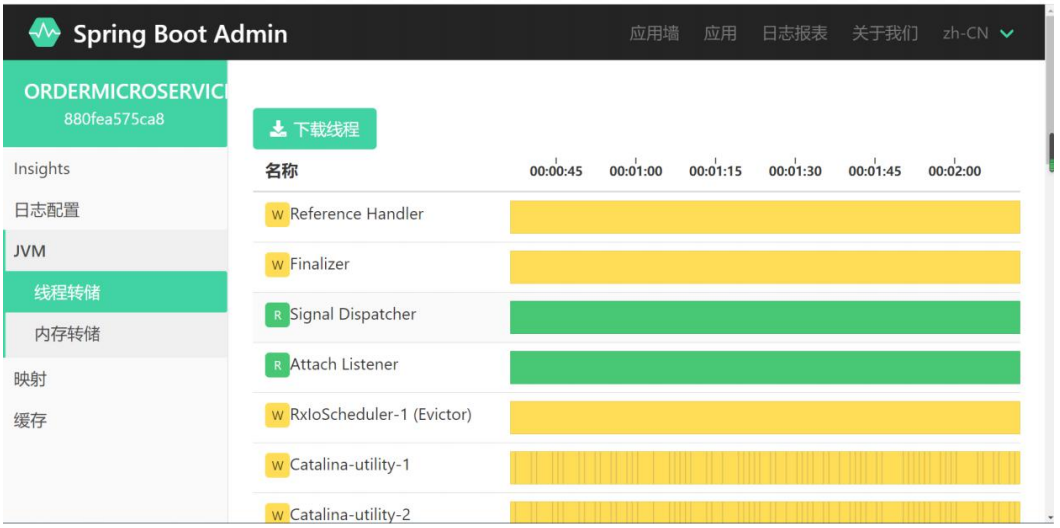
8）SERVER 程序采用了 响应式 Web 框架 Spring Webflux 。

9）展示 UI 采用了 Vue.js，

10）通过 Spring Boot Actuator 端点上的监控数据。

该组件可以单独使用，也可以和 Spring Cloud 体系结合使用，用于管理和监控 Spring Boot 应用程序，Spring Boot 也是属于微服架构的一种。可以和 Eureka，Consul 进行集成。

Spring Cloud 监控中心 Order 订单微服务



前端使用 VUE，依赖于 Actuator 端点上的监控数据，监控程序可以单独使用，也可以和 Eureka 进行集成，它自动会拉取监控的集群的数据给你生成一个监控的可视面板，上图显示是内存指标现成机构。

它监控的指标非常多，各种指标都可以去做，还可以和 G Max 开发集成，它功能非常强大。

二、Java Spring Cloud 微服务实例监控实战

1）开发监控服务端 Spring Boot Amin Server

2）注册到 Eureka 服务器

3）微服务注册到 Eureka 服务器

```
</properties>
<dependencies>
    <dependency>
        <groupId>de.codecentric</groupId>
        <artifactId>spring-boot-admin-starter-server</artifactId>
        <version>2.3.1</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
```

实战的话需要加 Admin 的依赖包，加载 Admin 服务器，其实在各个监控的程序上加入 Admin 的客户端，在配置文件上加入。

```
spring.application.name=eureka-server
server.port=8761
# 注册中心
eureka.client.service-
url.defaultZone=http://localhost:8761/eureka
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
# 健康数据
management.endpoints.web.exposure.include=*
management.endpoint.health.show-details=ALWAYS
```

监控配置

```
1  package com.alibaba.demo;
2
3* import org.springframework.boot.SpringApplication;
6
7  @EnableEurekaServer
8  @SpringBootApplication
9  public class JavaSpringCloud000AdminServerEurekaServerDemo {
10
11*     public static void main(String[] args) {
12          SpringApplication.run(JavaSpringCloud000AdminServerEurekaServerDemo.class, args);
13      }
14 }
15
```

修改配置文件

Spring Cloud 监控中心：

正常启动效果图

有多少服务都可以集成进来，如果出错了，也是正常显示，也可以去查看某一个服务实例，是集群或者是事态都可以显示出来，如果出错，会显示变黄，提供一个很好的提示。

注意点：

如果要和 Spring Cloud 进行集成，需要对 Server 进行修改配置，加入监控文件，采集数据，加入注解做好区分。

@EnableAdminServer（要启动）、@EnableDiscoveryClient 和注册中心集成，启到抓取和监控数据作用。

秉持谁监控谁暴露的原则，后续数据会自动拉取。

微服务或电脑端都可以加入暴露数据的代码，给 win Server 进行采集。

3.1 Spring CloudAlibaba 微服务体系

内容简介：

一、开源

二、服务框架

三、应用场景

一、开源

Spring Cloud Alibaba 是阿里巴巴提供的微服务开发一站式解决方案，是阿里巴巴开源中间件与 Spring Cloud 体系的融合。

提起微服务，不得不提 Spring Cloud 全家桶系列。SpringCloud 是若干个框架的集合，包括 Spring-Cloud-Config、Spring-Cloud-Bus 等近 20 个子项目，提供了服务治理、服务网关、智能路由、负载均衡、断路器、监控跟踪、分布式消息队列、配置管理等领域的解决方案。

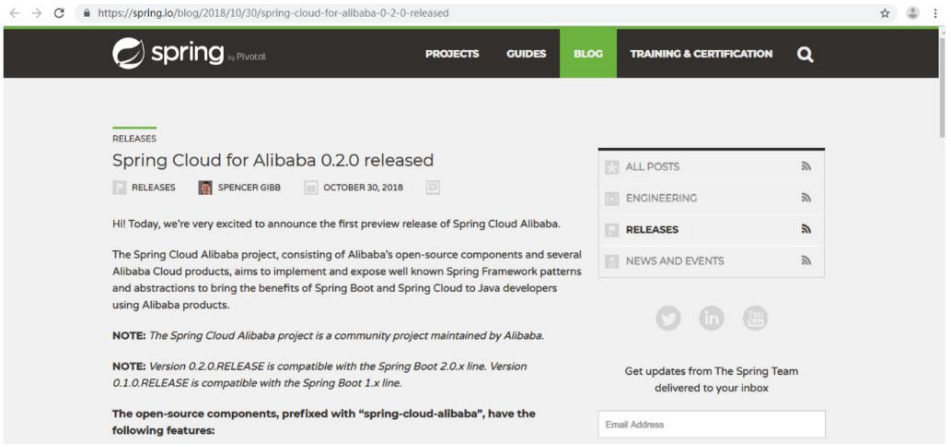
Spring Cloud 通过 Spring Boot 风格的封装，屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、容易部署的分布式系统开发工具包。

Spring Cloud Alibaba 于 2018 年 10 月 31 日凌晨开源，从 0.2.1 版本正式发布，随后捐赠给 Spring 社区，Spring Cloud 官方发布消息表示欢迎。

Spring Cloud 日益丰富的生态环境离不开亚马逊、微软等国际互联网巨头的框架贡献，伴随着阿里巴巴的加入，相信双方都会取得长足的进步。

二、服务框架

1. 阿里巴巴开源微服务框架



用户选择阿里巴巴开源微服务框架有三种方式。

一是去 Spring 官方网站通过在线向导生成，二是通过开发平台如 Eclipse 生成，三是用户自己手动进行开发工作，这种方式可以更改默认配置，推荐熟练用户选择此方式。

2. Spring Cloud Alibaba 优势

Spring Cloud Alibaba 主要有四大优势：开源免费、兼容 Spring Cloud、支持阿里云、替换老旧组件。

同时，Spring Cloud Alibaba 有阿里巴巴背书，有大规模分布式场景“淘宝双十一”作为案例。除此之外，阿里巴巴贡献了 Dubbo 等一系列框架，具有很大的影响力，阿里巴巴拥有众多分支的技术团队，技术深度在国内处于头部水平。

Spring Cloud 阿里巴巴为阿里巴巴中间件的分布式解决方案提供应用开发的一站式解决方案，此项目包含开发分布式应用微服务的必需组件，方便开发者通过 Spring Cloud 编程模型轻松使用这些组件来开发分布式应用服务，只需要添加一些注解和少量配置，就可以将 Spring Cloud 应用接入阿里微服务解决方案，通过阿里中间件来迅速搭建分布式应用系统。

3. Spring Cloud Alibaba 框架

1）微服务开源组件（spring-cloud-alibaba 开头）

2）服务发现（Nacos Service Discovery）

3）配置管理（Configuration Management）

4）高可用防护（Safeguarding for High Availability）

5）消息队列（RocketMQ）

6）任务调度（SchedulerX）

7）日志服务（SLS）

8）阿里云商业服务（spring-cloud-alicloud）

9）服务发现（ANS——Application Naming Service）

10）配置管理（ACM——Application Configuration Management）

11）对象存储服务（OSS——Object Storage Service）

4. Spring Cloud Alibaba 微服务框架

1）开源组件

- Nacos Config
- Nacos Discovery
- Sentinel
- RocketMQ
- Dubbo
- Fescar

2）商业化组件

- ANS
- ACM
- OSS
- SchedulerX

3）Example

- Sentinel
- Nacos Config
- Nacos Discovery
- RocketMQ
- OSS

5. Spring Cloud Alibaba 新特性

1）**服务限流降级：**默认支持 Servlet、Feign、RestTemplate、Dubbo 和 RocketMQ 限流降级功能的接入，可以在运行时通过控制台实时修改限 流降级规则，还支持查看限流降级 Metrics 监控。

2）**服务注册与发现：**适配 Spring Cloud 服务注册与发现标准，默认集成 Ribbon 的支持。

3）**分布式配置管理：**支持分布式系统外部化配置，配置更改时自动刷新。

4）**消息驱动能力：**基于 Spring Cloud Stream 支持微服务消息驱动能力。

5）**阿里云对象存储：**阿里云提供的海量、安全、低成本、高可靠的云存储 服务。任何应用、任何时间、任何地点存储和访问任意类型数据。

6）**分布式任务调度：**提供秒级、精准、高可靠、高可用的定时（基于 Cron 表达式）任务调度服务。同时提供分布式任务执行模型，如网格任务。网格任务支持海量子任务均匀分配到所有 Worker（Schedulerx-Client）上执行。

三、应用场景

Spring Cloud Alibaba 作为微服务有效框架的补充，可以和其他的 Spring Cloud 组件集成，用户可以根据自身实际的场景需求，选择最合适的微服务。

3.2 Spring Cloud AlibabaNacos 经典注册中心对比

内容简介：

一、Spring Cloud 服务注册与发现

二、Eureka 注册中心开发实战

三、Eureka 底层原理与源码分析

一、Spring Cloud 服务注册与发现

对于微服务业务需要先进行拆分，微服务实例部署数量不固定，可以弹性伸缩，与传统架构不一样，比较靠拢云计算、云原生。

1. 服务注册与发现

解决大规模服务集群的注册和发现问题，主要为了方便客户端调用。比如开发一个微服务是订单服务，开始只启用一台服务器，客户端和客户端直接调用微服务就可以了。但如果启用两台以上，客户端直接写多台服务的 IP 地址做轮巡是不行的，因为如果遇到类似双 11 促销场景，需要增加很多台服务，且是弹性不定数量增加，这时候最好有一种方案能够解决这个问题，把客户端和服务集群结耦。

结耦里面很重要的就是注册中心，注册中心可以帮助管理服务，当只有一个、两个服务的时候，客户端可以直接和服务建立连接，当服务数量不固定，且伸缩范围很大时，需要一个专门的机构帮助管理这些服务。

服务注册与发现总结：

1）大规模微服务集群架构；

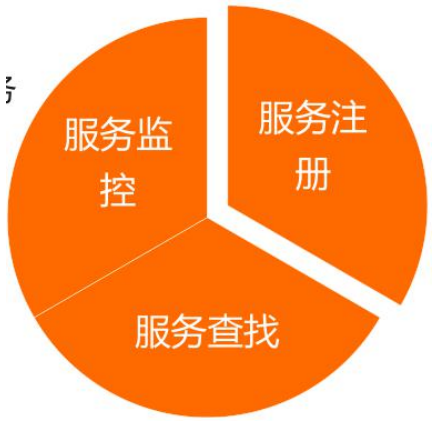
2）许多服务实例；

3）客户端要找到自己调用的服务；

4）新服务上线；

5）某个服务宕机，下线；

6）实时监控服务的状态。



2. Spring Cloud Eureka 服务发现与注册

在微服务架构体系里， Netflix 公司贡献了其中一个很重要的项目叫 Eureka，主要

解决服务注册中心的问题。在大数据里面，有同类型的产品 ZooKeeper，Spring Cloud 通过扩展组件也可以进行集成。

本节课主要讲实战开发，建立注册中心以后，怎么样把微服务注册进去，怎么让注册中心管理服务。总结：

1）Netflix 公司开源的项目。

2）Eureka：注册中心。

3）一个基于 REST 的中心服务，管理服务。

4）实现云端的服务注册和服务发现。

5）Eureka 组件组成：Eureka 服务器和 Eureka 客户端。

6）竞争对手 ZooKeeper。

7）服务发现模块（Eureka）是 Netflix 的核心。

8）Spring Cloud Netflix 提供的简化开发模板。

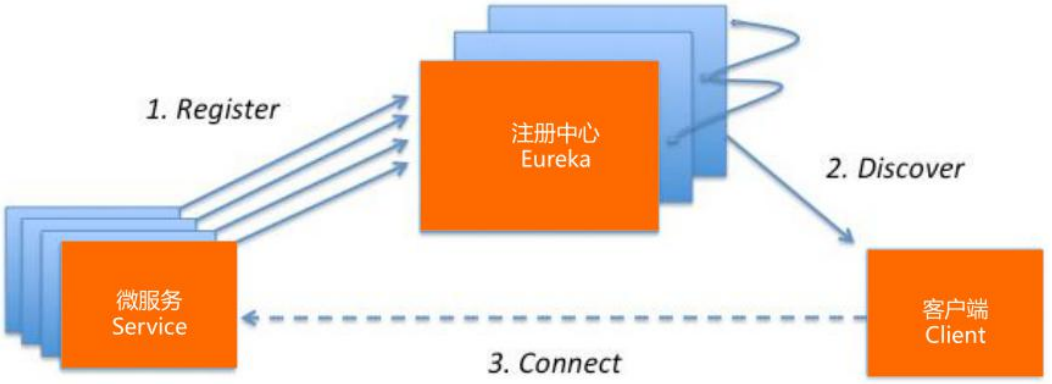
9）直接使用 spring boot，创建项目。

10）添加 @ EnableEurekaServer 开发注册服务中心。

3. Spring Cloud 架构图

下图所示，是微服务架构图，有注册中心，客户端微服务上线注册，微服务实例数量是动态的，有可能是一台，也有可能是很多台，灵活弹性根据客户端的压力做弹性伸缩。

客户端调用时不是直接找微服务，先找注册中心，有哪些好的微服务，有哪些匹配的，每次搜索最新的报表，然后调用。客户端也可以按照各种负载均衡的策略去调用，或者结合一些负责运行的算法灵活调度。



二、Eureka 注册中心开发实战

1. Spring Eureka 注册中心

1）创建 Eureka 服务注册中心项目;

2）添加@ EnableEurekaServer;

3）将 spring boot 应用改造成 Eureka 服务注册中心;

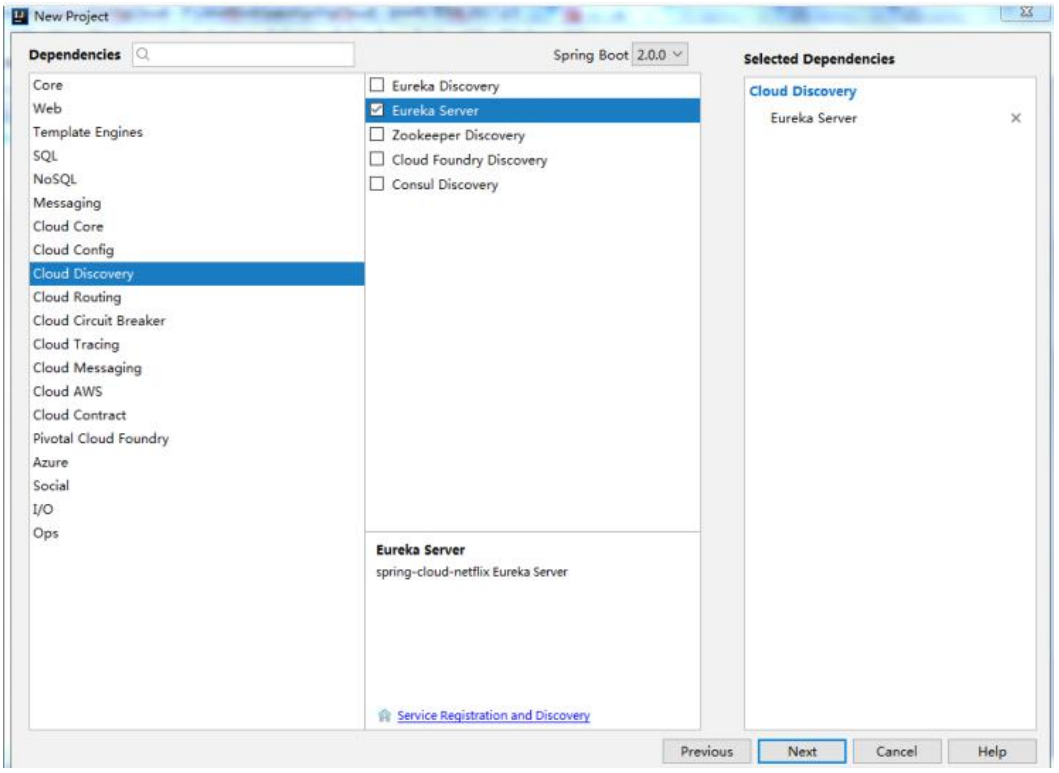
4）application.properties 增加配置;

5）打包项目;

6）运行;

7）测试页面;

8）参考 https://spring.io/guides/gs/service-registration-and-discovery/。



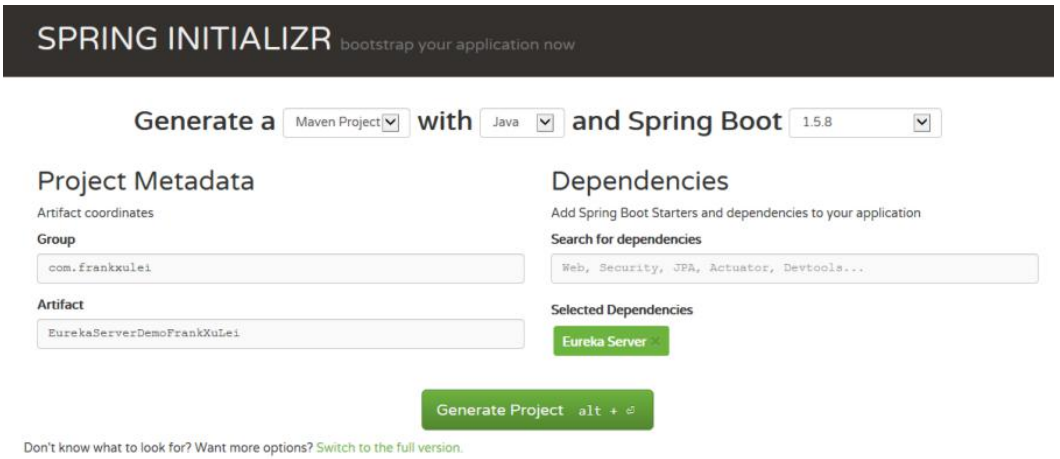
2. application.properties 配置

- server.port=8761;
- eureka.client.register-with-eureka=false;
- eureka.client.fetch-registry=false;
- eureka.client.serviceUrl.defaultZone=http://localhost:${ser ver.port}/eureka/;
- logging.level.com.netflix.eureka=OFF;
- logging.level.com.netflix.discovery=OFF;

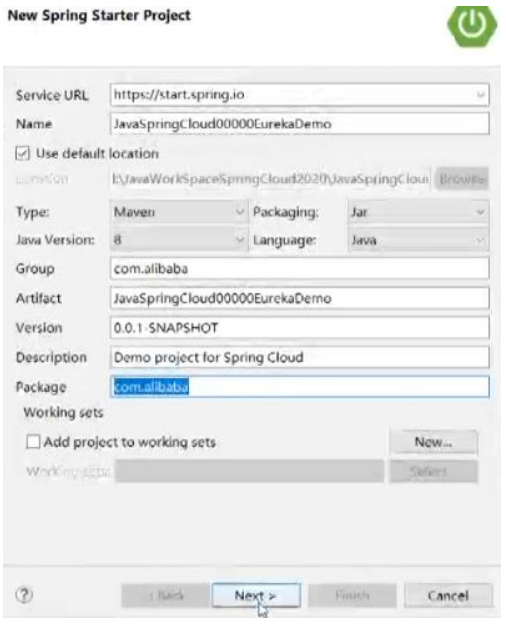
3. 创建 Spring Eureka 服务项目

演示部分：

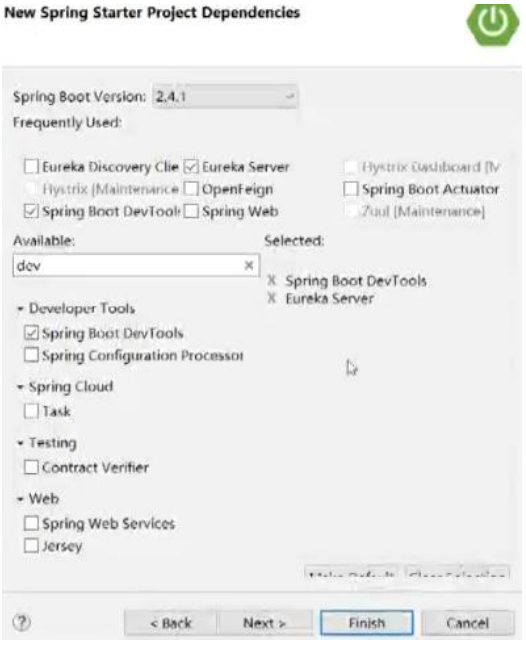
下图所示，是正常运行界面。



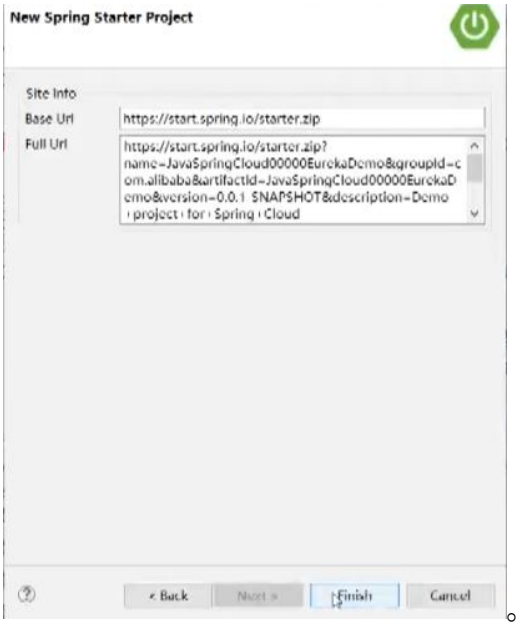
这里还没有服务实例，下面演示开发微服务，再把微服务注册进来。打开 Eclipse 开发工具，新建项目，插件里面会有 New Spring Sttarter Project 模板，Name 栏加入 Eureka，下面直接配公司的域名就行，Group 栏输入 com.alibaba，点击下一步，如下图所示：



Available 栏输入 Eureka，加入注册中心、服务端，为了开发调试再加入 dev，用于加载动态调试工具，做动态调试服务。演示版本是 2.4.1，再下一步：



下一步：



这里面会生成项目，Maven 会自动来去拉包，第一次拉包时间比较久，网络好应该会快一点，把 Maven 仓库改成阿里或其他公司，国内应该有很多 Maven 仓库镜像。

先来改配置文件，这里关键是加注解，启用 EurekaServer，EurekaServer 自动跟进配置，制作在某个端口上，包括注册中心的界面，会提供注册地址，在之前的例子上是提供 Rest API，供客户端注册。

第一步是，启动@ EnableEurekaServer，使服务器具备注册中心的能力，这一步非常重要。

```
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.server.EnableEurekaSe
6 @EnableEurekaServer
7 @SpringBootApplication
8 public class JavaSpringCloud00000EurekaDemoApplication {
9
10    public static void main(String[] args) {
11        SpringApplication.run(JavaSpringCloud00000EurekaDemoApplicat:
```

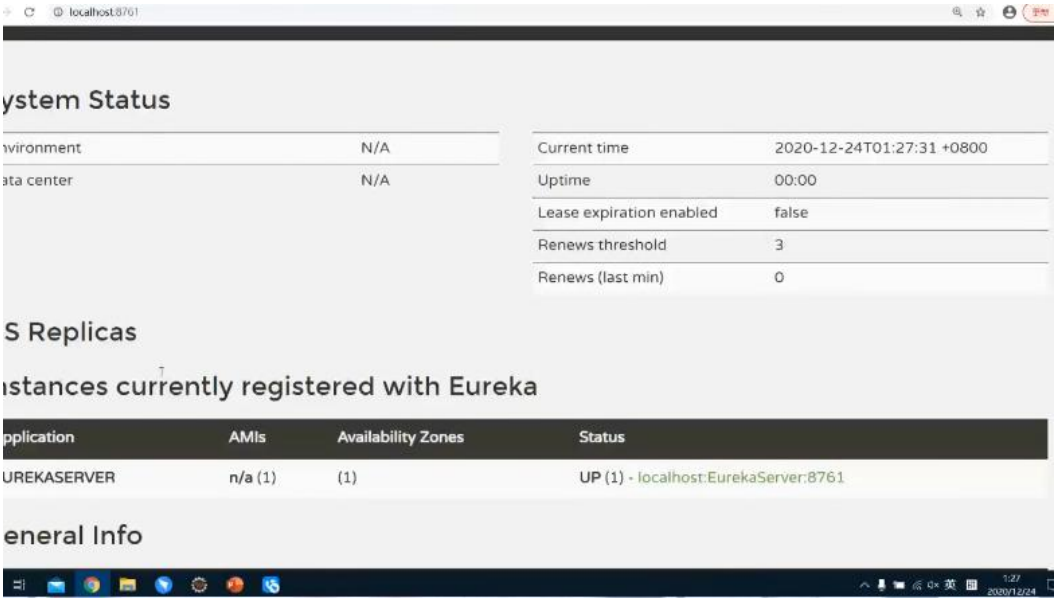
接下配一下配置文件，配置文件有几个核心参数，程序名和端口，程序名输入 EurekaServer。端口不配的话就是默认的 8080，因为官方例子是 8761，我们也配置 8761。还需要配置 Eureka 的注册客户端，如下图所示：“http://localhost:8761/eureka”是注册中心的地址，给客户端注册和查找用的。

```
1 spring.application.name=EurekaServer
2 server.port=8761
3 eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
```

接下来启动程序，直接右键，选 Run As ,有两种启动方式，1.jave Application，2. Spring Boot APP，两种都可以启动。

```
Markers Properties Servers Data Source Explorer Snippets Console Progress Debug Coverage
JavaSpringCloud00000EurekaDemoApplication [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (2020-12-24 1:26:57)
2020-12-24 01:27:15.852  INFO 13680 --- [  restartedMain] .s.c.n.e.s.Eur
2020-12-24 01:27:15.944  INFO 13680 --- [nio-8761-exec-1] o.a.c.c.C.[Tom
2020-12-24 01:27:15.945  INFO 13680 --- [nio-8761-exec-1] o.s.web.servle
2020-12-24 01:27:15.948  INFO 13680 --- [nio-8761-exec-1] o.s.web.servle
2020-12-24 01:27:16.041  INFO 13680 --- [nio-8761-exec-1] c.n.e.registry
2020-12-24 01:27:16.061  INFO 13680 --- [nfoReplicator-0] com.netflix.di
2020-12-24 01:27:17.758  INFO 13680 --- [  restartedMain] o.s.cloud.comm
2020-12-24 01:27:17.775  INFO 13680 --- [  restartedMain] avaSpringCloud
```

启动成功之后，打开浏览器，输入 http://localhost:8761 回车，出现 Eureka 注册中心的界面。里面一有个服务实例，这个实例默认没有关掉，自己可以往自己中心的注册，正常并不需要，自己往自己中心的注册实际是多余的。在比较底的版本会显示出错。

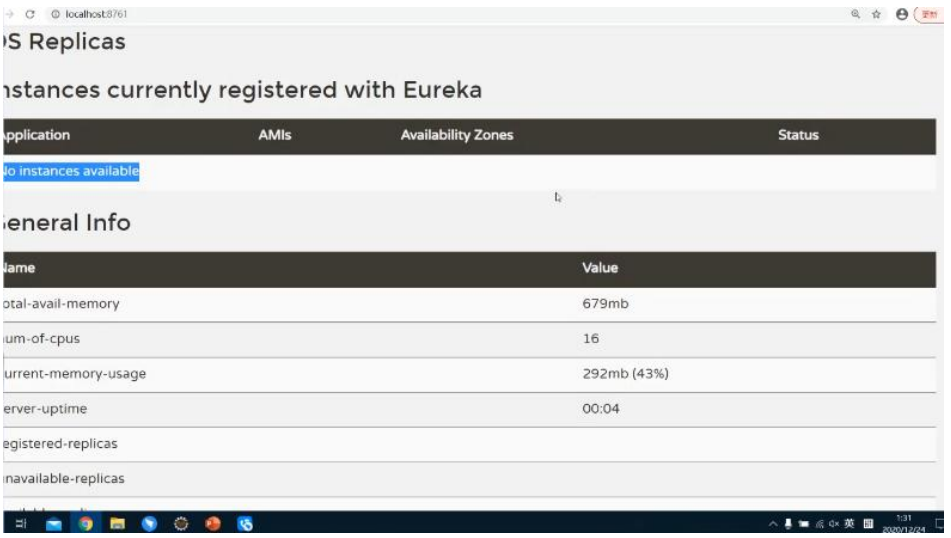


可以配置关闭上面的服务实例，输入“eureka.client.fetch-registry=false”，“eureka.client.register-with-eurka=falsr”。

```
1 spring.application.name=EurekaServer
2 server.port=8761
3 eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
4
5 eureka.client.fetch-registry=false
6 eureka.client.register-with-eureka=false
7
8
```

```
2020-12-24 01:31:30.591  INFO 13680 --- [      Thread-34] c.n.e.r.PeerAw
2020-12-24 01:31:30.592  INFO 13680 --- [      Thread-34] c.n.e.r.PeerAw
2020-12-24 01:31:30.595  INFO 13680 --- [  restartedMain] o.s.b.w.embedd
2020-12-24 01:31:30.595  INFO 13680 --- [  restartedMain] .s.c.n.e.s.Eur
2020-12-24 01:31:30.596  INFO 13680 --- [      Thread-34] e.s.EurekaServ
2020-12-24 01:31:32.371  INFO 13680 --- [  restartedMain] o.s.cloud.comm
2020-12-24 01:31:32.375  INFO 13680 --- [  restartedMain] avaSpringCloud
2020-12-24 01:31:32.379  INFO 13680 --- [  restartedMain] .ConditionEval
```

重新加载后，打开 eureka 界面，可以看到，自己往自己中心注册的实例就没有了。



三、Eureka 底层原理与源码分析

1. Netflix Eureka 源码

- Eureka 官方源码；
- https://github.com/Netflix/eureka；
- Spring Cloud Netflix 适配 Eureka 的代码；
- https://github.com/spring-cloud/spring-cloud-netflix。

2. Eureka 源码分析

Eureka 通信基于 Http(s)协议的框架

1）纯正的 servlet 应用，需构建成 war 包部署；

2）使用了 框架实现自身的 HTTP 接口；

3）peer 之间的同步与服务的注册全部通过 HTTP 协议实现；

4）定时任务(发送 、定时清理过期服务、节点同步等)通过 JDK 自带的 Timer 实现；

5）内存缓存使用 Google 的 包实现；

3. 服务实例 Instance 的状态

1）Up；

2）Down；

3）Starting；

4）Out_Of_Service；

5）Unknown。

3.3 Spring Cloud 开发微服务 API 注册到 Nacos

内容简介：

一、下载安装、启动 Nacos 服务

二、修改 Spring Cloud 微服务项目注册到 Nacos 中心

三、Nacos 微服务管理中心

一、下载安装、启动 Nacos 服务

1. 微服务集成 Nacos

Nacos 是阿里巴巴开源的注册和配置中心，Nacos 目前在 Spring Cloud 微服务开发方面用的比较多，不仅是之前的 Dubbo，和其他语言结合的也比较好。

在做实战的时候，首先下载安装整个 Nacos 服务。Nacos 服务对比 Eureka，Eureka 需要自己构建 Spring Cloud 项目，加入 Eureka 的依赖，改配置、加注解，然后启动；Nacos 是已经构建完成打包好的，直接官方下载最新的解压包，启动就可以了，当然也可以改默认的配置，不改也行。

接下来升级改造微服务架构，把微服务注册到 Nacos 服务上，再调用微服务。

操作总结：

1）启动 Nacos；

2）微服务注册到 Nacos；

3）客户端连接 Nacos，调用微服务。

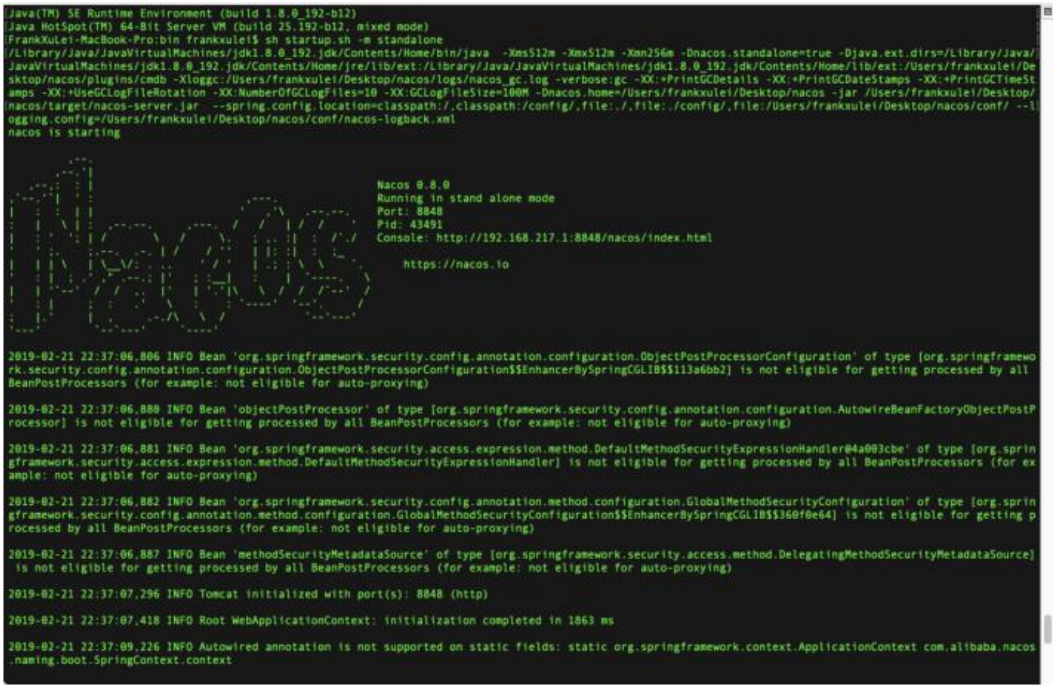
2. 启动 Nacos 服务器

打开官网 https://github.com/alibaba/nacos。下载 cmd startup.cmd 或者双击 startup.cmd 文件。

startup.sh 直接启动，“-m”是指定参数，用 class 表示是集训模式，不用 class 用 standalone 表示单点模式。开发设置阶段可以用单点模式，生产环境下可以用集群模式，保证可用性、高并发。Linux/Unix/Mac、Windows 系统都可以使用。

- https://github.com/alibaba/nacos
- Linux/Unix/Mac
- Standalone means it is non-cluster Mode. * sh startup.sh -m standalone
- Windows
- cmd startup.cmd 或者双击 startup.cmd 文件

3. Mac OS 苹果系统启动 Nacos



二、修改 Spring Cloud 微服务项目注册到 Nacos 中心

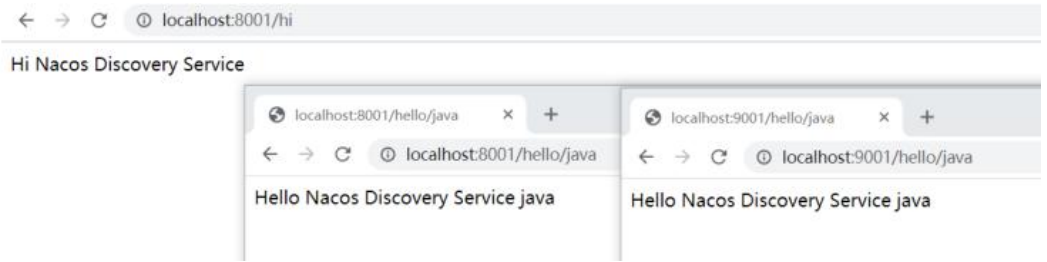
1. Java Spring Cloud 微服务

改造微服务最重要的是加入 Nacos 依赖，假设 Nacos 已经开启，接下来将改造微服务项目注册进来，改配置文件，配置文件地址把之前 Eureka 地址换掉就可以了。

- POM
- <dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
- </dependency>

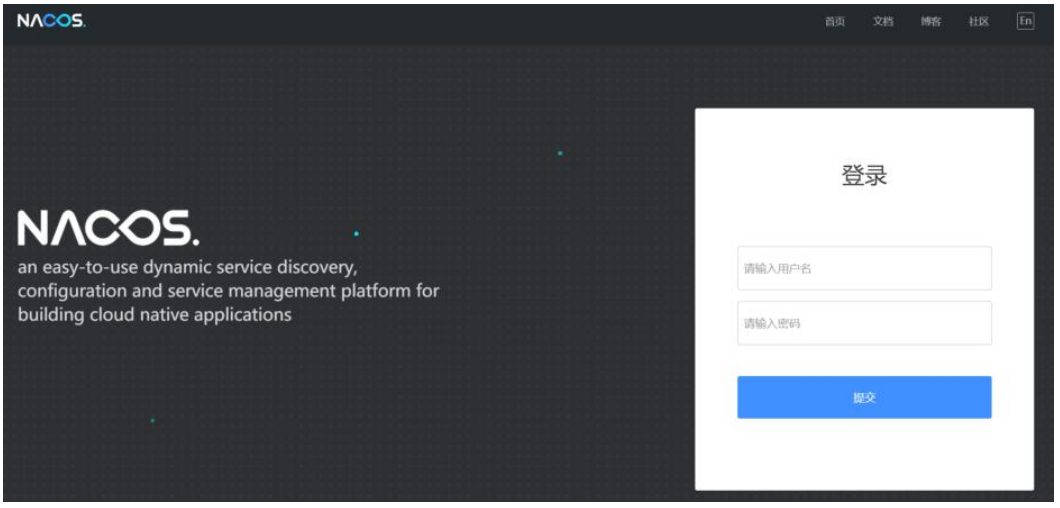
- **配置**
- server.port=18080
- spring.application.name=microservice
- spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
- **代码 REST API**

2. 验证微服务 API



三、Nacos 微服务管理中心

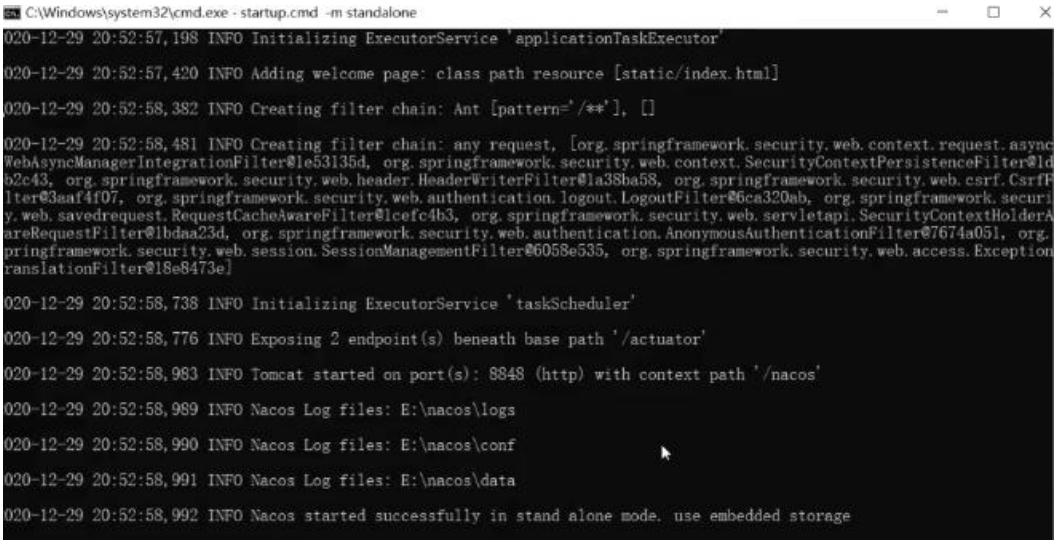
注册进来以后，Nacos 可以提供监控、管理微服务，配置一系列更新操作等，非常方便进入 Nacos 的监控界面，登录可以看到服务的监控信息、有哪些服务、服务列表状态。Nacos 有中文版和英文版，可以直接使用，还有权限控制、命名空间、集群管理、统一配置服务、配置推送等功能。



实践演示：

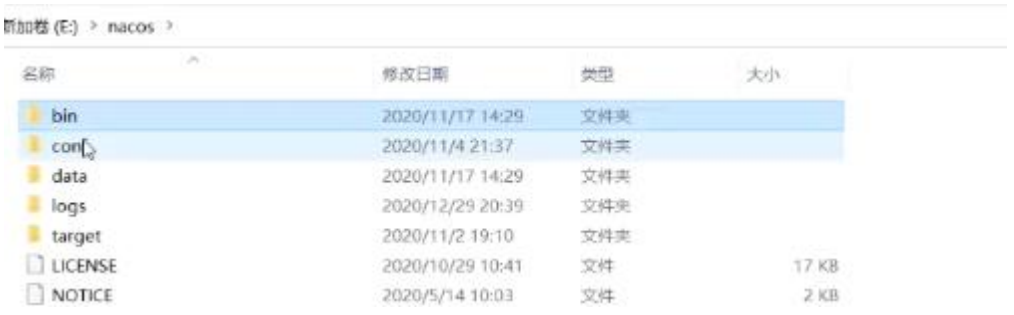
直接在 Nacos 官网下载，可以切换成中文模式，Nacos 不仅支持 Spring Cloud，还支持 Spring Boot、Dubbo、Docker、k8s、Syns 等。演示用 1.4.0 版本，点击下载。

下载后本地有“startup.sh”解压包，演示为 WIN 10 系统，运行输入“cmd”。运行框输入“cd nacos/bin”,再输入“nacos bin>startup.cmd-m standalong”启动。



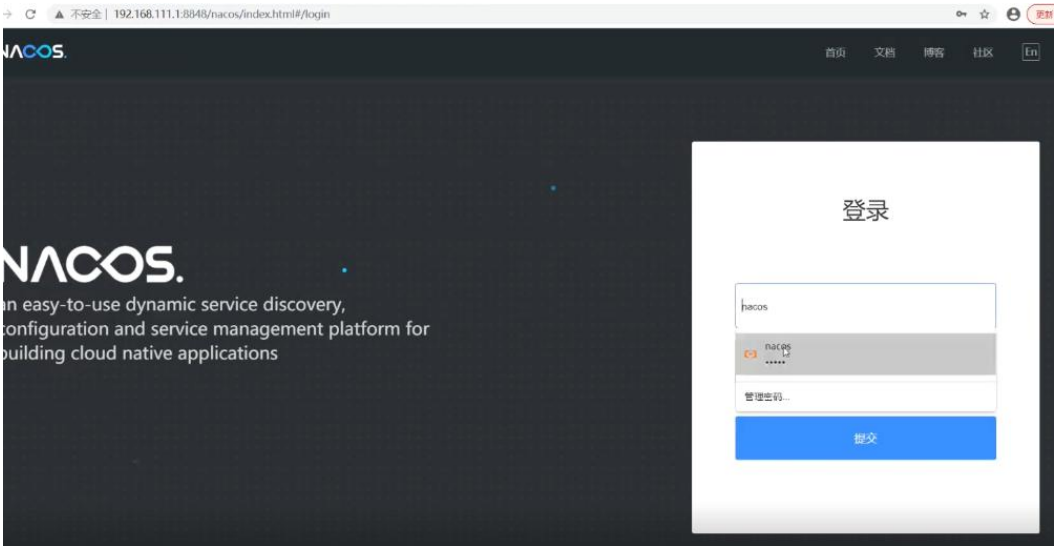
这有个管理界面，正常启动下面会有一个提示，下面有日志“nacos\logs”、配置文件“nacos\conf”用于改集群、改默认端口，数据文件的位置“nacos\data”。

大家可以看一下 nacos 的解压包，包括 data、logs、conf 等。



Conf 文件下面还有一个功能是，如果希望 nacos 与本地的 MySQL 或者远程的 MySQL 数据库进行对接，监控信息或服务令牌或配置信息不想丢掉，可以改一下服务，在配置文件里加入 MySQL 的用户密码，直接可以对接进去，这是创建表结构。

复制 http://192.168.111.1:8848/nacos/index.html 地址，到浏览器，打开 nacos 登录界面，输入用户:nacos,密码:nacos。



进入 nacos 管理界面，点开服务列表页面，目前列表里面是空的，上面有生产环境、开发环境、测试环境。

接下来改造微服务，首页要加入 nacos 依赖包，注意版本兼容性问题，这里使用的是 2.2.3 版本，

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.alibaba.cloud/sp
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery
        <version>2.2.3.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
```

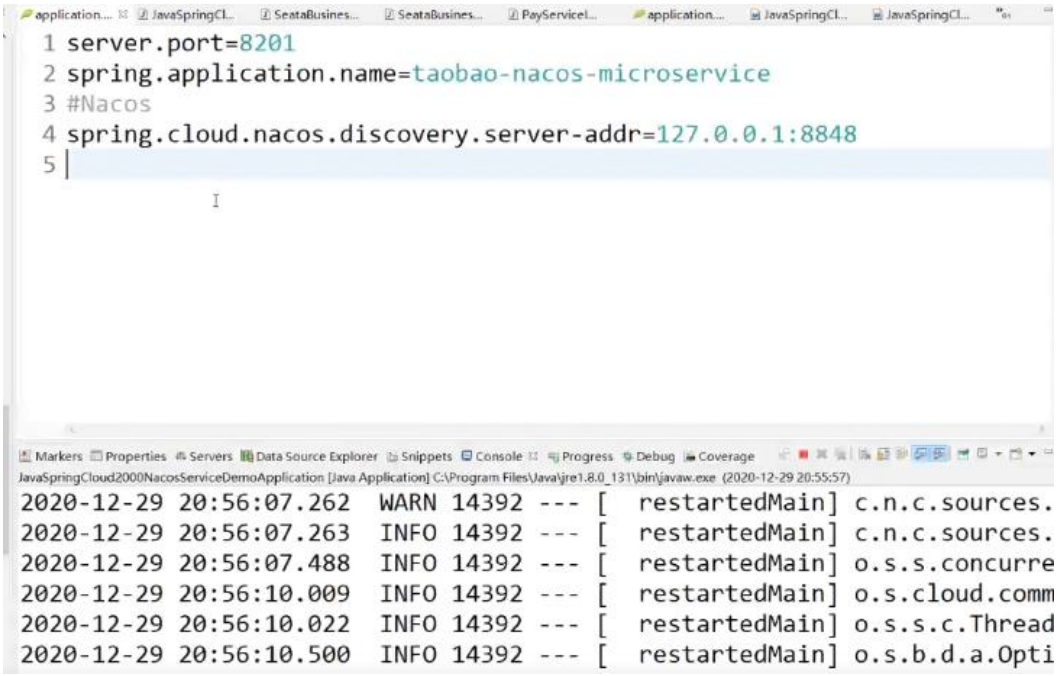
底层用的是“@EnableDoscoveryClient”,如下图所示：

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClie
@SpringBootApplication
@EnableDiscoveryClient
public class JavaSpringCloud2000NacosServiceDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(JavaSpringCloud2000NacosServiceDemoApp
    }
}
```

配置文件改成“127.0.0.1：8848”，表示注册中心的位置。

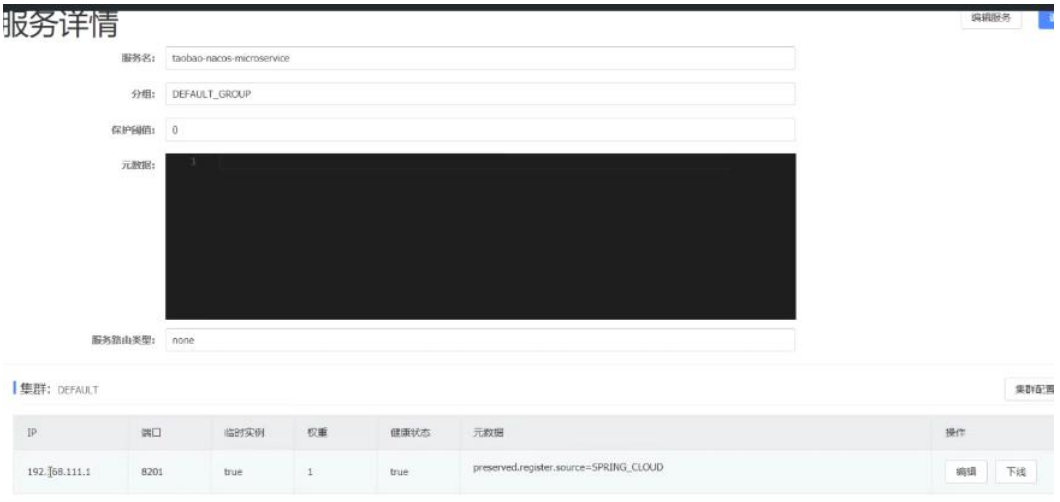
```
server.port=8201
spring.application.name=taobao-nacos-microservice
#Nacos
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

然后右击启动微服务，注意现在的端口是 8201。

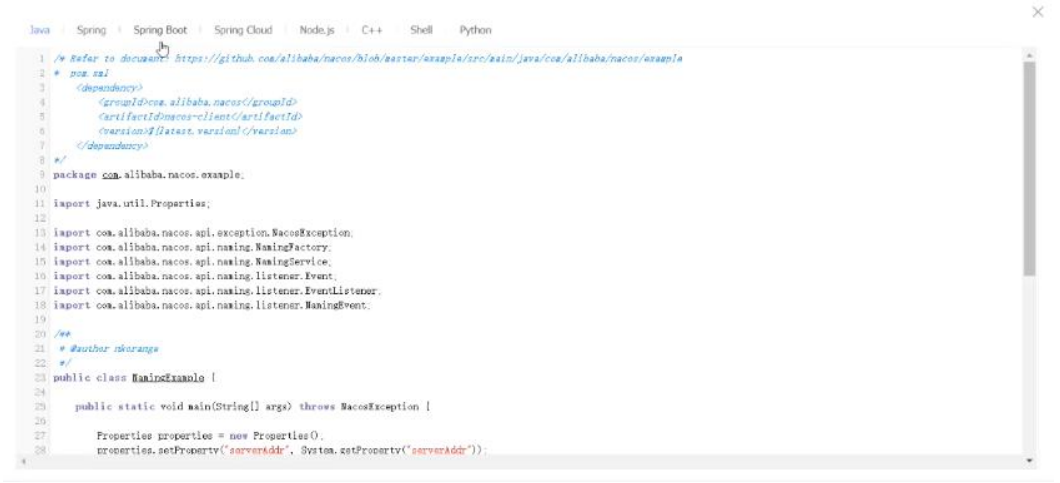


正常情况下，微服务应该上线了，进入 nacos 服务列表界面，刷新可以看到有一条“taobao-nacos-microservice”淘宝的微服务。

点击“详情”，可以看到详细的监控信息：



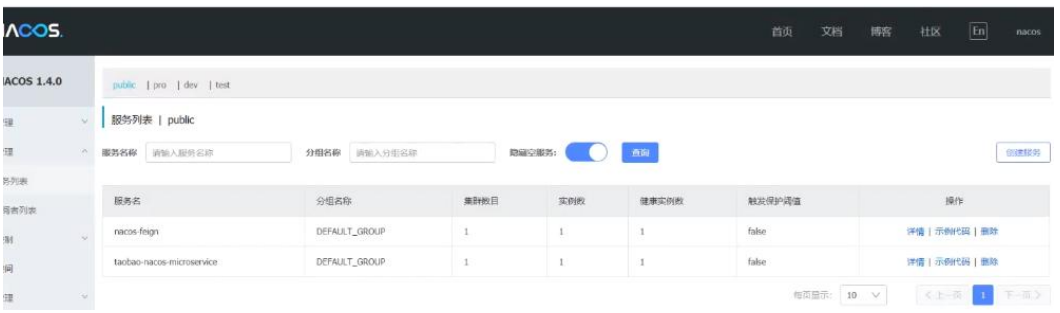
点击“示例代码”，可以看到已经生成了客户端的调用代码，而且还包括 Java、Spring、Spring Boot、C++等多种语言的调用代码。



同理，还可以改造调用端“feign”，先将依赖加进来，改配置文件，再把注册中心替换一下。想办法把程序集成进来，启动调用端。

```java
package cn.alibaba;

import org.springframework.boot.SpringApplication;

@EnableDiscoveryClient
@EnableFeignClients
@SpringBootApplication
public class JavaSpringCloud202009FeignDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(JavaSpringCloud202009FeignDemoApplicat
    }
```

再到 Nacos 服务列表界面查看，可以看到一条“nacos-feign”服务。



同理也可以多启动几个微服务，复制上面的“taobao-nacos-microservice”淘宝的微服务，将端口修改成“server.prot=8202”,然后启动。

```
server.port=8202
spring.application.name=taobao-nacos-microservice
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

再到 Nacos 服务列表界面查看，可以看到实例数变成了“2”个，属于默认的集群里已经有两个实例，健康实例数也是 2，说明操作成功。


public | pro | dev | test
服务列表 | public
服务名称
分组名称
隐藏空服务:
查询
创建服务
服务名
分组名称
集群数目
实例数
健康实例数
触发保护阈值
操作
nacos-feign
DEFAULT_GROUP
1
1
1
false
详情 | 示例代码 | 删除
taobao-nacos-microservice
DEFAULT_GROUP
1
2
2
false
详情 | 示例代码 | 删除
每页显示: 10
1

3.4 Spring Cloud 客户端 Feign 集成 Nacos 中心

内容简介：

一、重构调用端 Feign 的项目代码

二、重构调用端 Feign 的项目代码

这节课的话咱们讲如何改造我们的之前的 Spring Cloud 项目，让他和 Feign 客户端进行集成，实现微服务的一个调用，要完成服务的注册，还要完成客户端的一个数据中心 Nacos 的一个对接，接下来我们要完成调用工作，之前我们实现了这样一个调用链，现在的话就需要我们同样去改造我们的整个项目。包括代码的实现，这里面其实主要就是一个依赖，参考我们之前的代码、依赖、改配置，通过我们的整个项目。

一、重构调用端 Feign 的项目代码

1. Java Spring Cloud 微服务调用端 Feign

- POM
- <dependency>
- <groupId>org.springframework.cloud</groupId> •

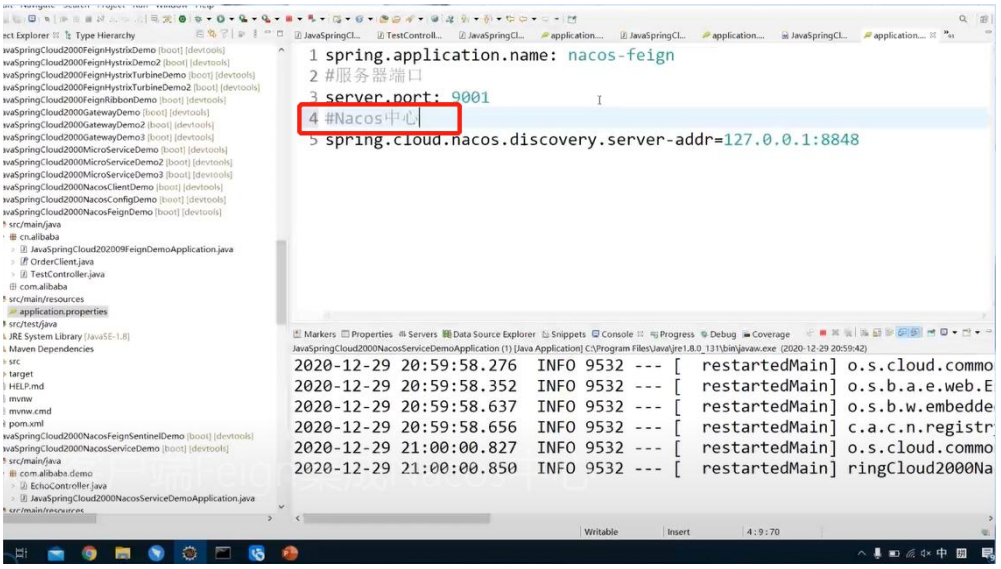
<artifactId>spring-cloud-starter-alibaba-nacosdiscovery</artifactId>

- </dependency>
- **配置**
- server.port=8080
- spring.application.name=microservice-caller
- spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
- 代码 REST API

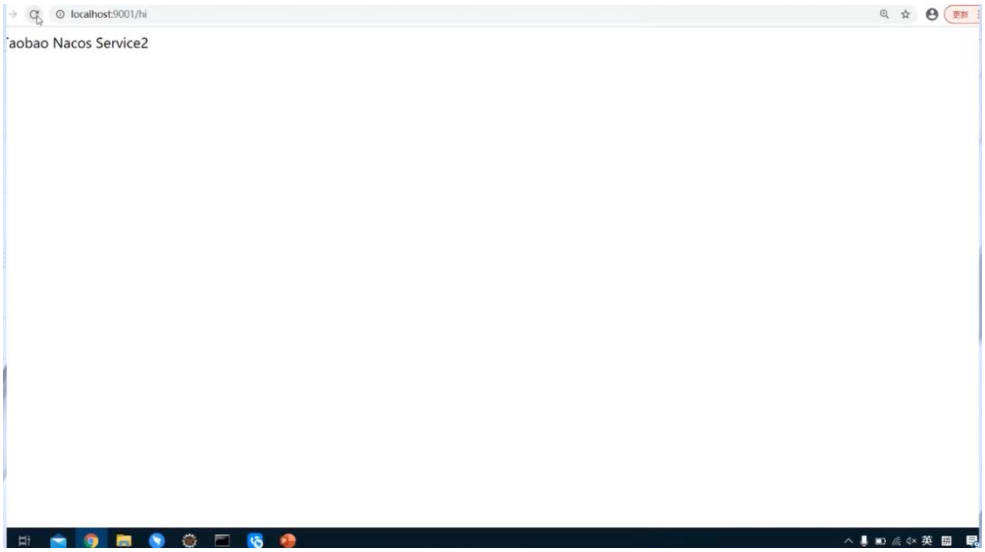
后面的话像我们还讲熔断，包括统一配置都在讲，这里面的话这 POM 与配置是非常重要的，当然我们现在的话给大家演示一下整个过程，主要还是以实战为主。

实战：Feign Nacos

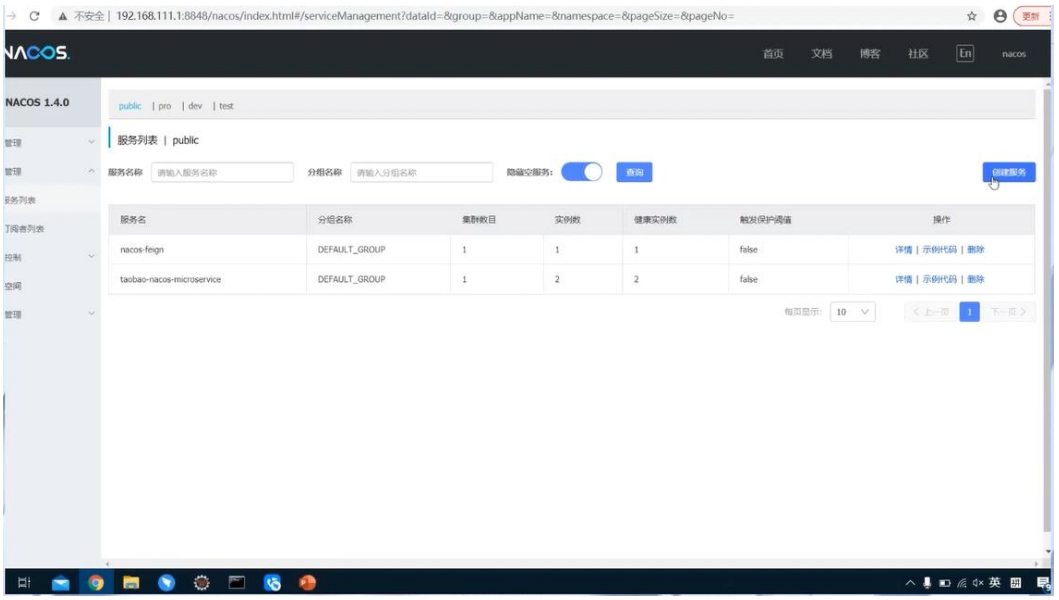
我现在看一下咱们客户端是 Nacos Feign ，Feign 的话是看我们的端口咱们来演示一下看看怎么调。



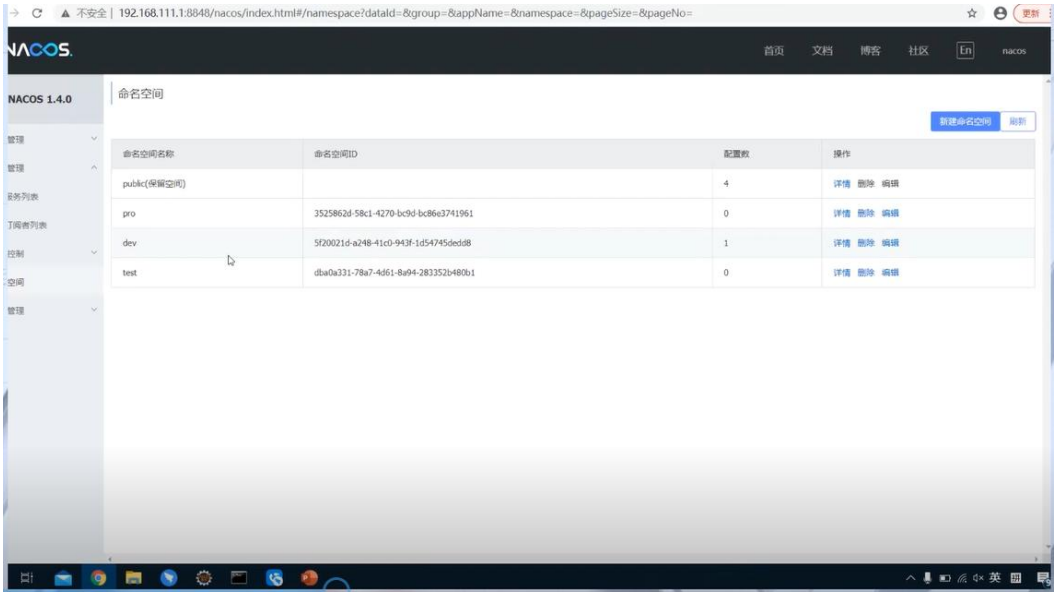
1、添加 Nacos 中心，查看服务器端口，服务器端口为 9001。



2、打开 9001，点 hi，然后查看负载均衡在一二，它执行的模式的是轮巡，我们是通过 Feign 的客户端我们来调谁调后端的服务。

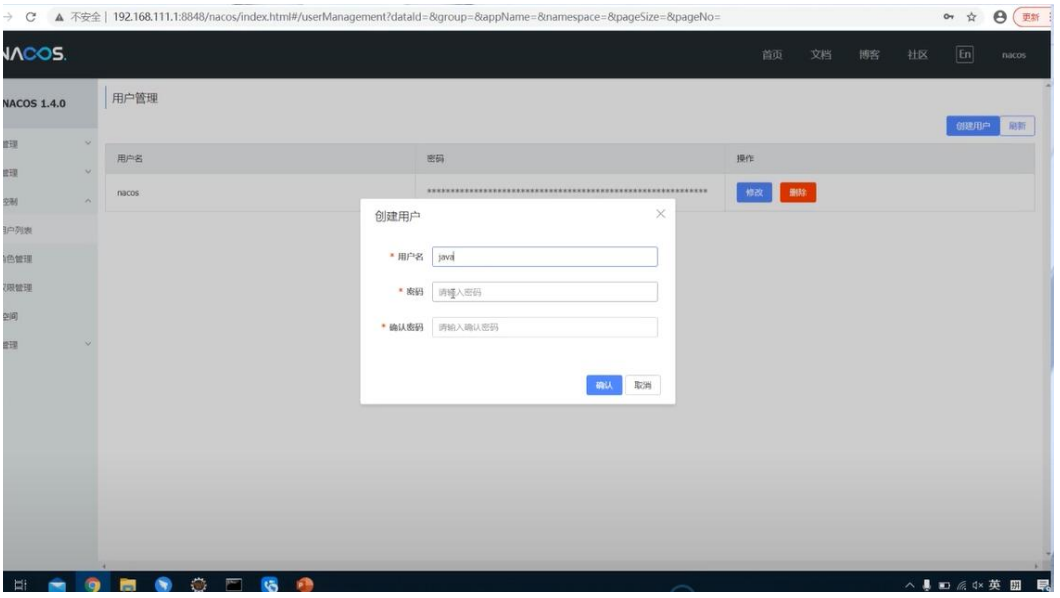


大家可以看一下咱们这个注册中心，注册中心的话可以动态下线，你可以把服务给删掉。这是我们说比 URL 做的比较好的地方，隐藏空服务而且我们说各种搜索，当然你也可以手动录入服务。



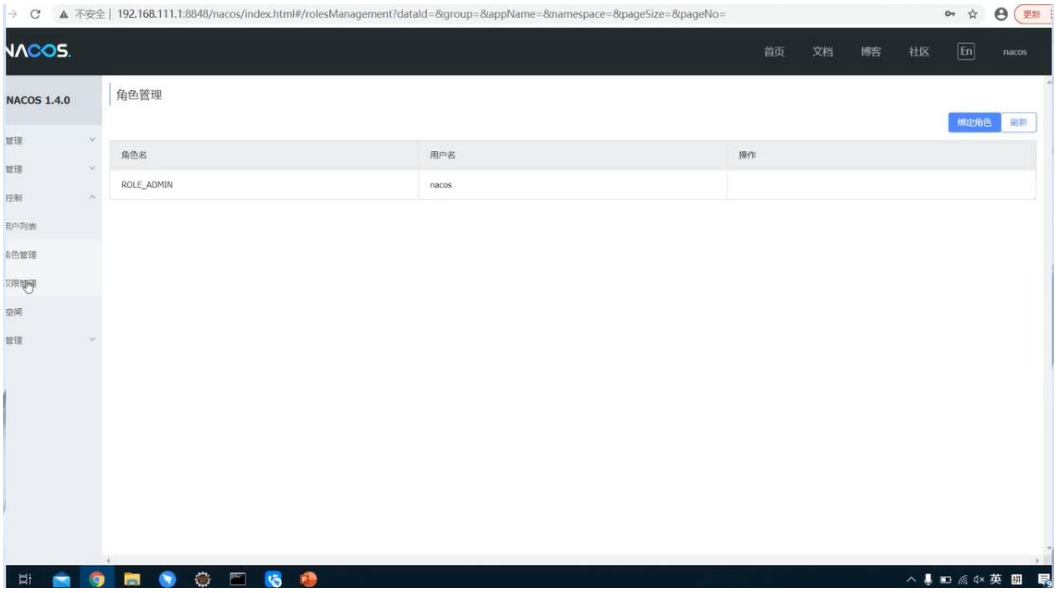
这里面还可以创建命名空间，命名空间作用是做后期开发发布，后期可以通过命名空间来管理生产、测试、开发每个不同命名空间可以保存自己的资源，可以定义不同的配置然后给不同的空间去使用。

用户管理：


192.168.111.1:8848/nacos/index.html#/userManagement?dataId=&group=&appName=&namespace=&pageSize=&pageNo=
NACOS
首页
文档
博客
社区
nacos
NACOS 1.4.0
用户管理
创建用户
刷新
用户名
密码
操作
nacos
修改
删除
创建用户
* 用户名
java
* 密码
请输入密码
* 确认密码
请输入确认密码
确认
取消

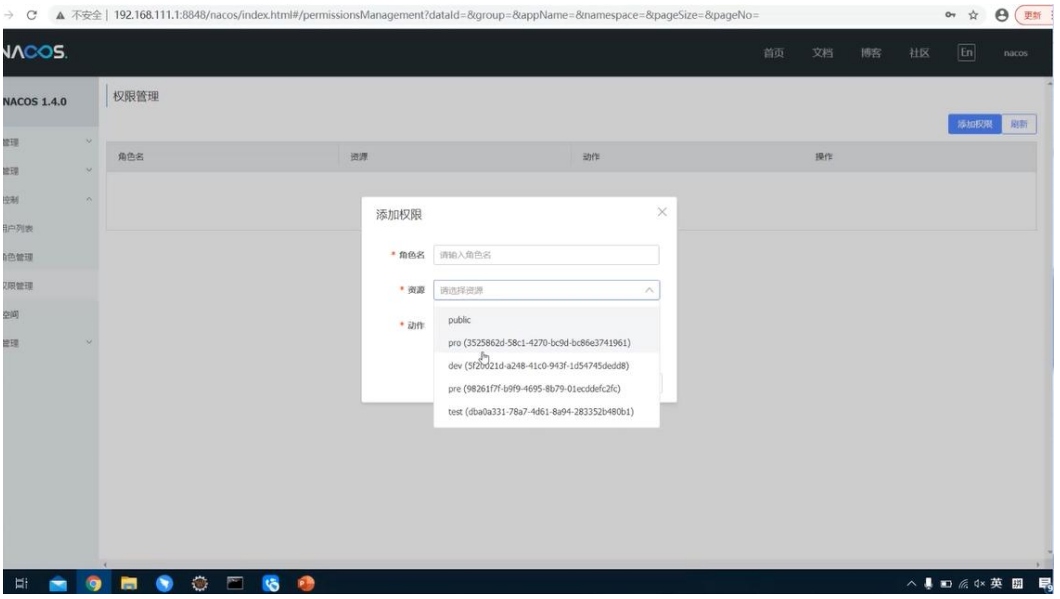
可以添加用户。

角色管理：


192.168.111.1:8848/nacos/index.html#/rolesManagement?dataId=&group=&appName=&namespace=&pageSize=&pageNo=
NACOS
首页
文档
博客
社区
nacos
NACOS 1.4.0
角色管理
绑定角色
刷新
角色名
用户名
操作
ROLE_ADMIN
nacos

可以自定义添加用户角色。

权限管理：



可以给用户开通指定权限。

二、重构调用端 Feign 的项目代码

1. Java Spring Cloud 微服务调用

- POM
- <dependency>
- <groupId>org.springframework.cloud</groupId>

• <artifactId>spring-cloud-starter-alibaba-nacosdiscovery</artifactId> • </dependency>

• 配置

• server.port=8080

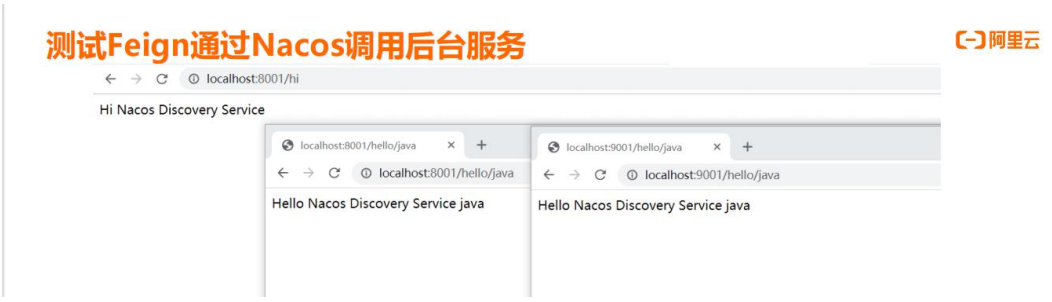
• spring.application.name=microservice-caller

• spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848

• 代码 REST API

2. 验证 API

3. 测试 Feign 通过 Nacos 调用后台服务



3.5 Spring Cloud 使用 Nacos 作为微服务统一配置中心

内容简介：

一、Nacos 作为 Spring Cloud 配置中心

二、微服务集成 Nacos 配置中心

三、Nacos 配置中心其他功能

一、Nacos 作为 Spring Cloud 配置中心

1. Nacos 配置中心

阿里开源 Nacos 的另外一个功能，统一配置中心，是对于方式集群大规模集群非常重要的一个功能点，阿里开源 Nacos，实际是集注册中心和配置中心为一身，是一个非常全能的工具。

Nacos 启动阶段配置文件稍微特殊一点，有一个 bootstrap 文件 ，是加载普通配置之前执行的一个动作，优先启动的一个文件系统，会优先读取文件中的一些配置数据。

1）Spring Cloud Alibaba Nacos Config；

2）Nacos 提供用于存储配置和其他元数据的 key/value 存储；

3）为分布式系统中的外部化配置提供服务器端和客户端支持；

4）使用 Spring Cloud Alibaba Nacos Config；

5）可以在 Nacos Server 集中管理 Spring Cloud 的外部属性配置；

6）Alibaba Nacos Config 是 Config Server 和 Client 的替代方案；

7）客户端和服务器上的概念与 Spring Environment 和 PropertySource 有着一致的抽象；

8）bootstrap 启动阶段，配置被加载到 Spring 环境中；

9）当应用程序通过部署管道从开发到测试再到生产时；

10）可以管理不同环境 dev\test\pro 的配置；

11）并确保应用程序具有迁移时需要运行的所有内容。

2. Nacos 配置服务

1）Nacos 也支持配置服务；

2）统一微服务集群的配置信息；

3）支持配置的动态更新；

4）可支持 profile 粒度的配置；

5）支持自定义 namespace 的配置，做资源管理配置，数据可到开发测试，生产环境有了命名空间，里面找对应的一个配置数据，进行推送下发，主要便于实现配置的更新和推送管理；

6）支持自定义 Group 的配置，主要是分组去管理服务的；

7）支持自定义扩展的 Data Id 配置；

8）配置的优先级；

9）可以通过 Spring 的${spring.profiles.active} 配置多套环境；

10）微服务项目需要 spring-cloud-starter-alibaba-nacos-config；

11）支持两种配置文件格式:

- bootstrap.properties
- bootstrap.yml

二、微服务集成 Nacos 配置中心

1. Nacos 配置中心实战

- 启动 Nacos 服务
- Nacos 添加配置参数
- 修改客户端，连接 Nacos 配置中心
- 注意：

  版本 2.1.x.RELEASE 对应的是 Spring Boot 2.1.x 版本。

  版本 2.0.x.RELEASE 对应的是 Spring Boot 2.0.x 版本。

  版本 1.5.x.RELEASE 对应的是 Spring Boot 1.5.x 版本。

2. 客户端配置

- <dependency>
- <groupId>com.alibaba.cloud</groupId>
- <artifactId>spring-cloud-starter-alibaba-nacosconfig</artifactId>
- </dependency>

3. 微服务集成 Nacos 配置中心

- bootstrap.propertie 配置 Nacos Server 地址
- spring.application.name=nacos-config
- spring.cloud.nacos.config.server-addr=127.0.0.1:8848

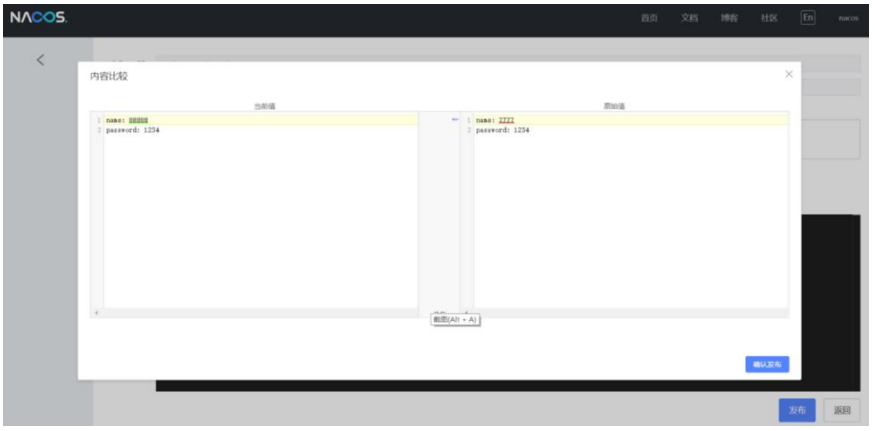
4. Nacos 支持动态刷新配置

通过配置 spring.cloud.nacos.config.refresh.enabled=false 来关闭动态刷新。

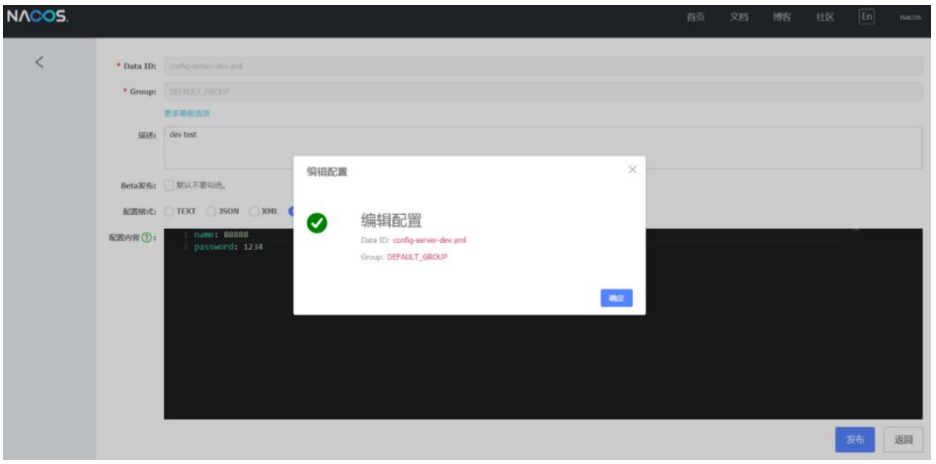
动态更新，启动的更新本质上是自动会刷新配置，一边改完以后，另一边会迅速读到通过线程读取，线程去远端去进行拉取操作。

原始的 APP 协议，本身并不是双工通信，是单向的。

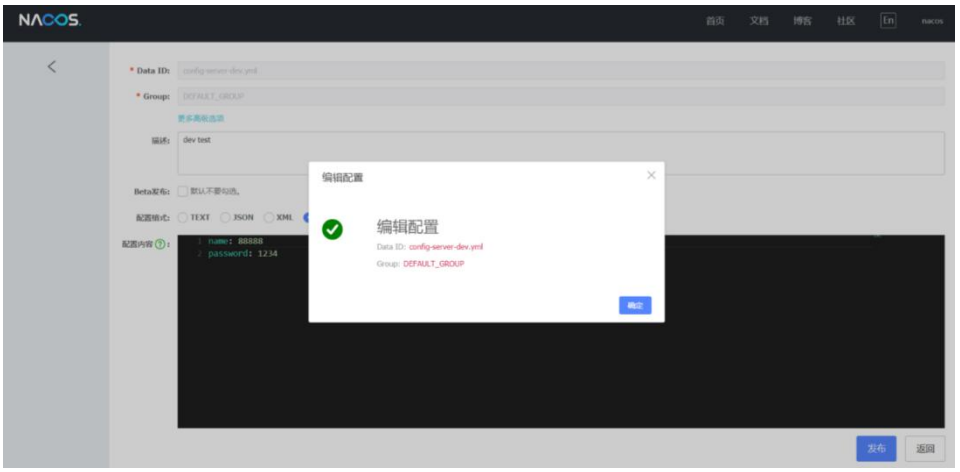
5. 更新配置实战效果



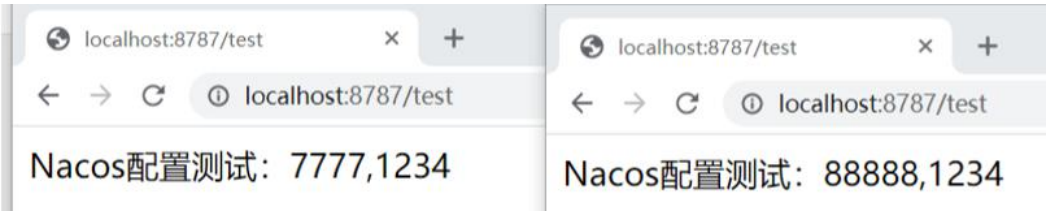
6. Spring Cloud 配置参数变



7. Spring Cloud 配置参数变化



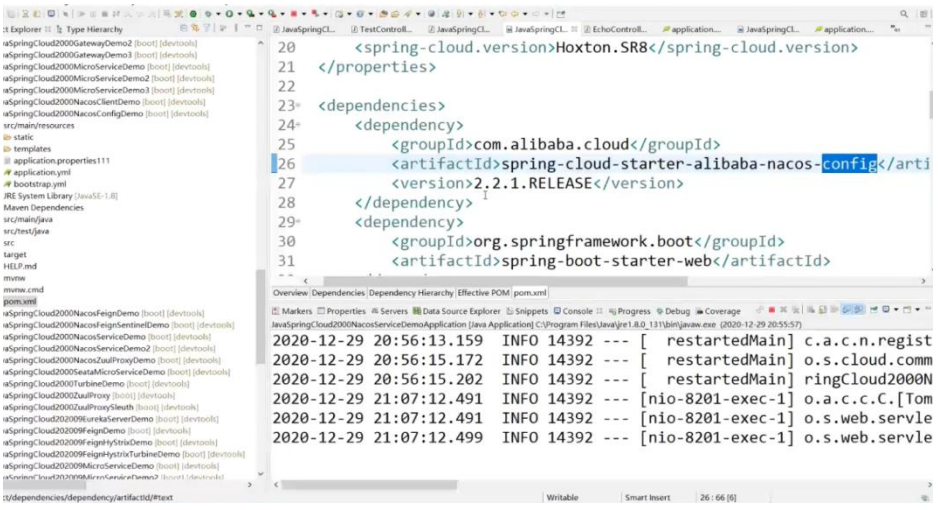
8. Spring Cloud 配置参数变化



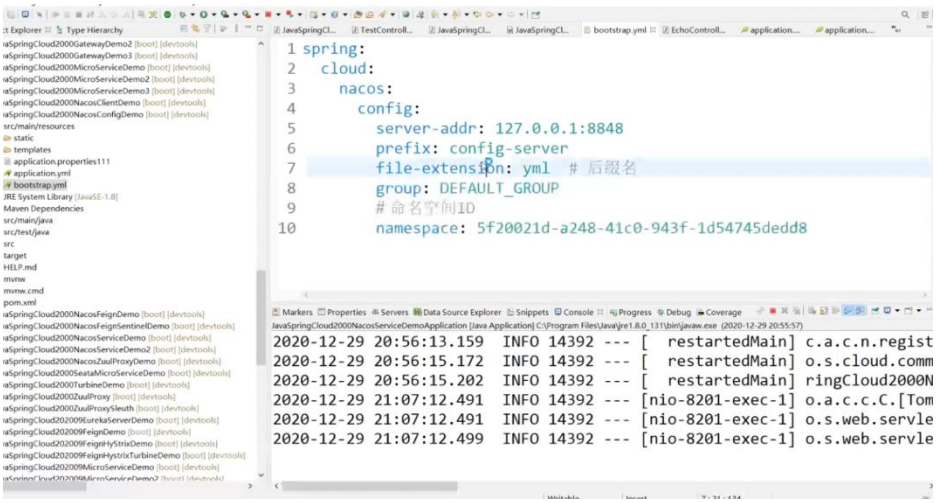
三、Nacos 配置中心其他功能

实战案例

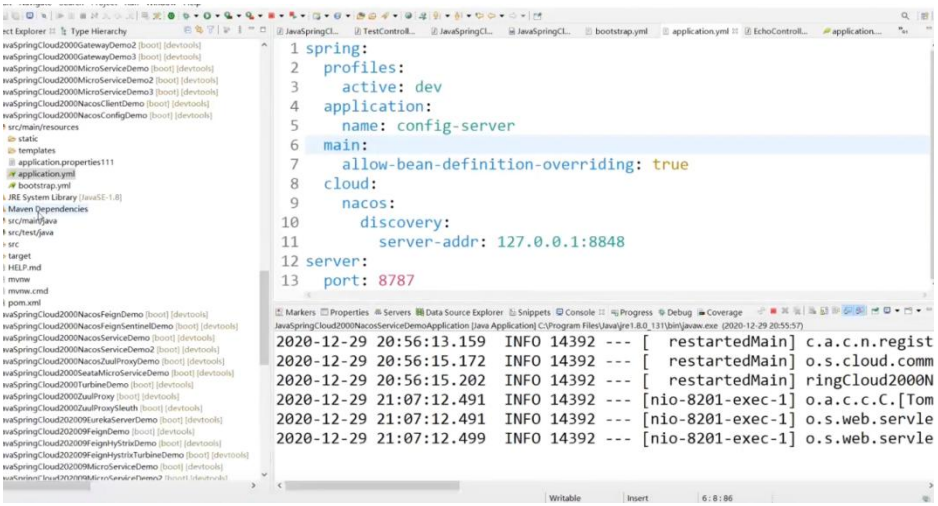
配置服务可以是普通的微服务项目，普通微服务项目改造 config 依赖，因为要实现配置，如简单的配置推送拉取，这是 config 独有的。



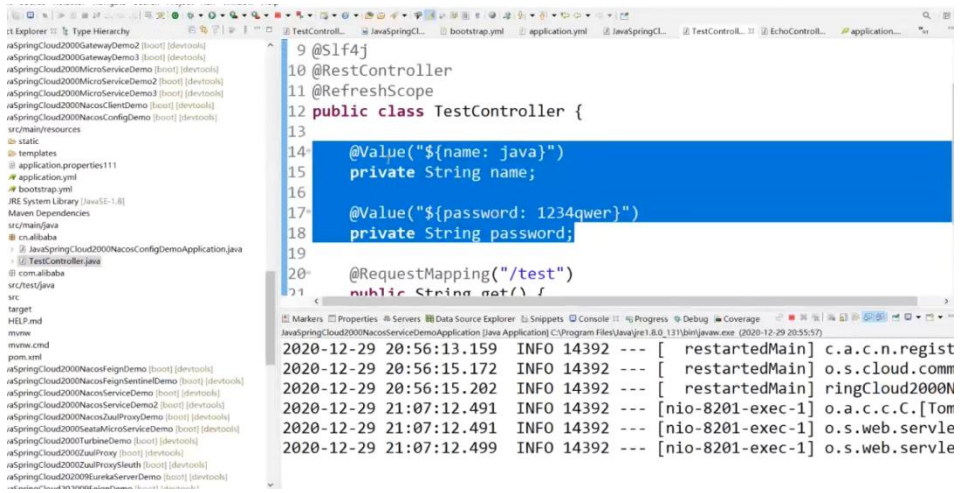
配置文件定义就按照下图规则来做：



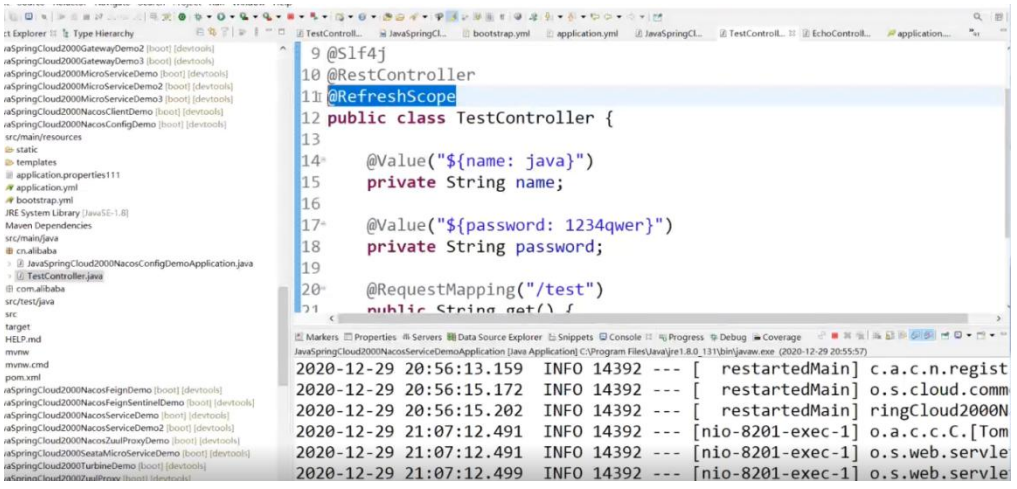
关键的配置，选的活动状态是 dev，dev 模式就是开发环境，要保证和配置中心要统一，这里端口稍微改了，8787，通过这样一个配置，读取 Nacos 配置中心最新的配置数据，然后在本地做一个展示。



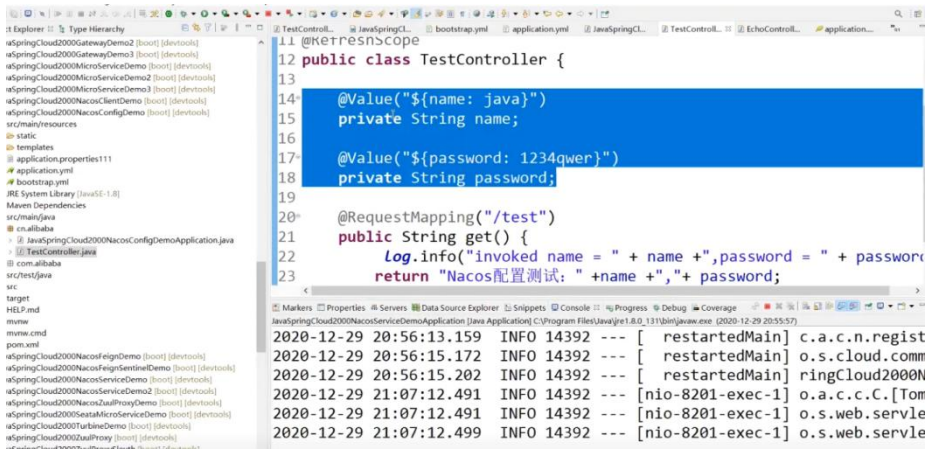
本地展示，通过拟定了一个简单动态的变量来测试，主要是定义了一个用户名，一个密码，两个变量，通过围绕注解，让它自动去加载最新的变量值。



启用注解，refresh 主要作用自动去背后线程去拉取注册中心最新的值。

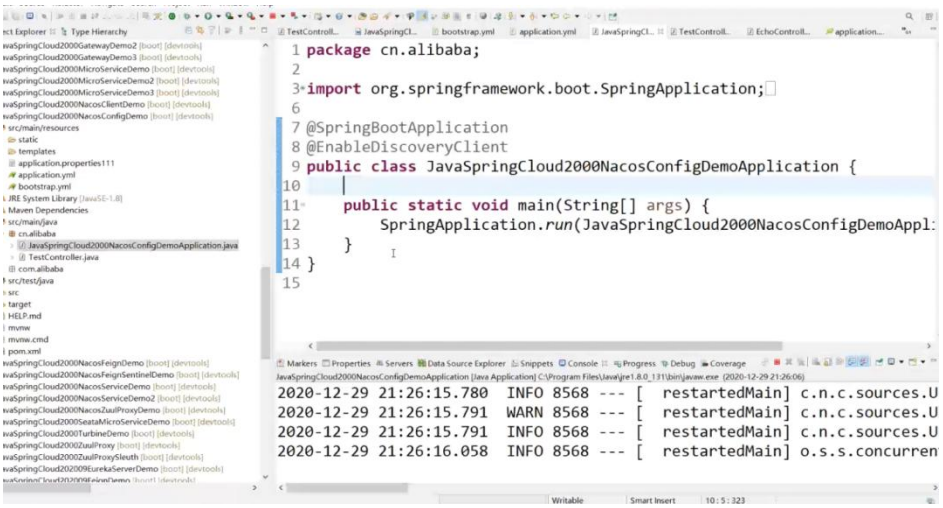


默认值，一个是 Java，一个是 1234qwer。



通过接口来调用，返回最新的用户密码，所以这里面如果之前用户和密码是 Java 和 1234qwer 的话，正常配置中心推送完新的配置后，重新查询，就应该返回新的值。

正常加载 bootstreap 再加载 Applcation



1. 支持 profile 粒度的配置

- spring-cloud-starter-alibaba-nacos-config 在加载配置的时 候，不仅仅加载了以 dataid 为${spring.application.name}.${file-extension:properties} 为前 缀的基础配置。
- 还加载了 dataid 为 ${spring.application.name}${profile}.${file-extension:properties} 的基础配置。
- 在日常开发中如果遇到多套环境下的不同配置，可以通过 Spring 提供的 ${spring.profiles.active} 这个配置项来配置。
- spring.profiles.active=dev
- spring.profiles.active=test
- spring.profiles.active=pro

2. 支持自定义 namespace 的配置

- 用于进行租户粒度的配置隔离。不同的命名空间下，可以存在相同的 Group 或 Data ID 的配置。Namespace 的常用场景之一是不同环境的配 置的区分隔离，例如开发测试环境和生产环境的资源（如配置、服务）隔离 等。
- 在没有明确指定 ${spring.cloud.nacos.config.namespace} 配置的情况下， 默认使用的是 Nacos 上 Public 这个 namespae。如果需要使用自定义的命 名空间，可以通过以下配置来实现：
- spring.cloud.nacos.config.namespace=b3404bc0-d7dc-4855-b519570ed34b62d7
- 该配置必须放在 bootstrap.properties 文件中。此外 spring.cloud.nacos.config.namespace 的值是 namespace 对应的 id， id 值可以在 Nacos 的控制台获取。并且在添加配置时注意不要选择其他的 namespae，否则将会导致读取不到正确的配置。

3. 支持自定义 Group 的配置

- 在没有明确指定 ${spring.cloud.nacos.config.group} 配置的情 况下， 默认使用的是 DEFAULT_GROUP 。
- 如果需要自定义自己的 Group，可以通过以下配置来实现：
- spring.cloud.nacos.config.group=DEVELOP_GROUP
- 该配置必须放在 bootstrap.properties 文件中。并且在添加配 置时 Group 的值一定要和 spring.cloud.nacos.config.group 的配置值一致。

4. 支持自定义扩展的 Data Id 配置

Spring Cloud Alibaba Nacos Config 从 0.2.1 版本后，可支持自定义 Data Id 的配置。关于这部分详 细的设计可参考 这里。 一个完整的配置案例如下所示:

- spring.application.name=opensource-service-provider
- spring.cloud.nacos.config.server-addr=127.0.0.1:8848
- # config external configuration
- # 1、Data Id 在默认的组 DEFAULT_GROUP,不支持配置的动态刷新
- spring.cloud.nacos.config.extension-configs[0].data-id=ext-config-common01.properties
- # 2、Data Id 不在默认的组，不支持动态刷新
- spring.cloud.nacos.config.extension-configs[1].data-id=ext-config-common02.properties
- spring.cloud.nacos.config.extension-configs[1].group=GLOBALE_GROUP

- # 3、Data Id 既不在默认的组，也支持动态刷新
- spring.cloud.nacos.config.extension-configs[2].data-id=ext-config-common03.properties
- spring.cloud.nacos.config.extension-configs[2].group=REFRESH_GROUP
- spring.cloud.nacos.config.extension-configs[2].refresh=true

5. 配置的优先级

- Spring Cloud Alibaba Nacos Config 目前提供了三种配置能力 从 Nacos 拉取相关的配置。
- A: 通过 spring.cloud.nacos.config.shared-configs[n].data-id 支持多个共享 Data Id 的配置。
- B: 通过 spring.cloud.nacos.config.extensionconfigs[n].data-id 的方式支持多个扩展 Data Id 的配置。
- C: 通过内部相关规则(应用名、应用名+ Profile)自动生成相关的 Data Id 配置
- 当三种方式共同使用时，他们的一个优先级关系是:A < B < C。

3.6 Spring Cloud 实战集成 Sentinel 熔断限流

内容简介：

一、阿里巴巴 Sentinel 熔断限流工具

二、阿里巴巴 Sentinel 分布式架构

三、Sentinel 实战 Spring Cloud

四、开发 Sentinel 微服务项目

五、测试 Sentinel 微服务熔断限流

一、阿里巴巴 Sentinel 熔断限流工具

1. Alibaba 微服务组件 Sentinel

1）Sentinel：分布式系统的流量防卫兵。

2）Sentinel：哨兵，流量控制、熔断降级、系统负载保护等多维度保护服务的稳定性。

Sentinel 本身的意思，我们说叫分布式系统的流量防卫兵。其次的话基于某个设置条件来做实现熔断功能，实现降低保护系统的一个可用性。

2. Sentinel 新特性

1）丰富的应用场景：Sentinel 承接了阿里巴巴近 10 年的双十一大促流量的核心场景，例如秒杀（即突发流量控制在系统容量可以承受的范围）、消息削峰填谷、集群流量控制、实时熔断下游不可用应用等。

2）完备的实时监控：Sentinel 同时提供实时的监控功能。您可以在控制台中看到接入应用的单台机器秒级数据，甚至 500 台以下规模的集群的汇总运行情况。

3）广泛的开源生态：Sentinel 提供开箱即用的与其它开源框架/库的整合模块，例如与 Spring Cloud、Dubbo、gRPC 的整合。您只需要引入相应的依赖并进行简单的配置即可快速地接入 Sentinel。

4）完善的 SPI 扩展点：Sentinel 提供简单易用、完善的 SPI 扩展接口。您可以通过实现扩展接口来快速地定制逻辑。例如定制规则管理、适配动态数据源等。

3. Sentinel 对比 Hystrix

对比内容	Sentinel	Hystrix
隔离策略	信号量隔离	线程池隔离/信号量隔离
熔断降级策略	基于响应时间或失败比率	基于失败比率
实时指标实现	滑动窗口	滑动窗口（基于 RxJava）
规则配置	支持多种数据源	支持多种数据源
扩展性	多个扩展点	插件的形式
基于注解的支持	支持	支持
限流	基于 QPS，支持基于调用关系的限流	不支持
流量整形	支持慢启动、匀速器模式	不支持
系统负载保护	支持	不支持
控制台	开箱即用，可配置规则、查看秒级监控、机器发现等	不完善
常见框架的适配	Servlet、Spring Cloud、Dubbo、gRPC 等	Servlet、Spring Cloud Netflix

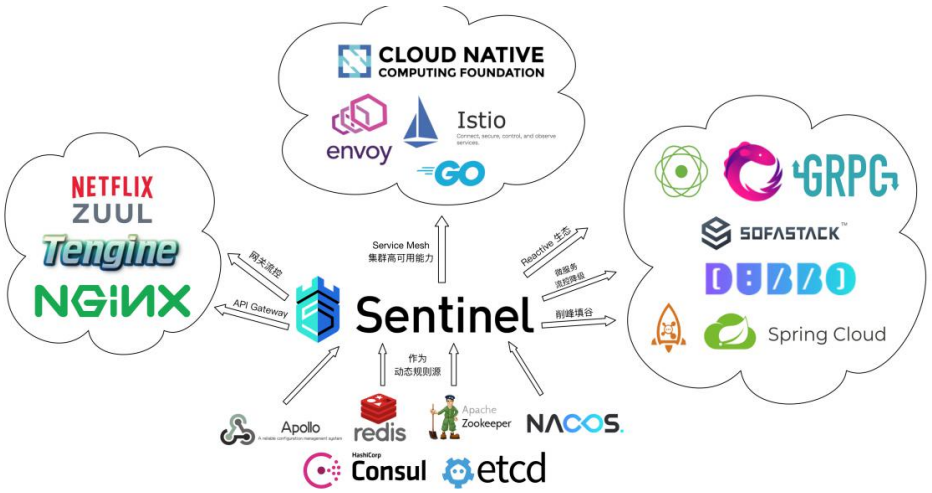
Hystrix 对比 Sentinel 属于完败，图表里面提到了限流基于调用关系限流它不支持慢启动它也不支持附带保护，这里面的话它也有管理界面，还是管理页面相对来说比较简单。URL 的管理页面简单还是随时随地，咱们讲这个原因并不是他做不好，是因为麦飞公司和皮波罗公司是一出现这个叫隔阂。他不愿意花大量的时间改在开源项目上，因为这个事情自己没得到什么好处。

二、阿里巴巴 Sentinel 分布式架构

1. Sentinel 的逻辑架构

监控是熔断限流得一个基础，这里面大家支持的技术，比较新的就是 service mesh，service mesh 就叫服务网格，更偏底层一点。 Sentinel 并不是要取代 service cloud，他们不是取代 Dubbo，也不取代 gRPC，是它的有效补充，大家工作的层次不一样，设备变形更偏底层一点。

2. Sentinel 的开源生态



篇底层的网络调度，比如网络，虽然有些的包括级的工作，它是概念是重复的，但是两个功能并不冲突，它生态比较完善，对接的各种不同的分布式场景。当然那个也可以集成它的实际是来者不拒包容的心态对接所有的技术。

三、Sentinel 实战 Spring Cloud

1. 改造 Spring Cloud 微服务

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.alibaba.cloud/spring-cloud-starter-alibaba-r
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
        <version>2.2.3.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
```

四、开发 Sentinel 微服务项目

1. POM

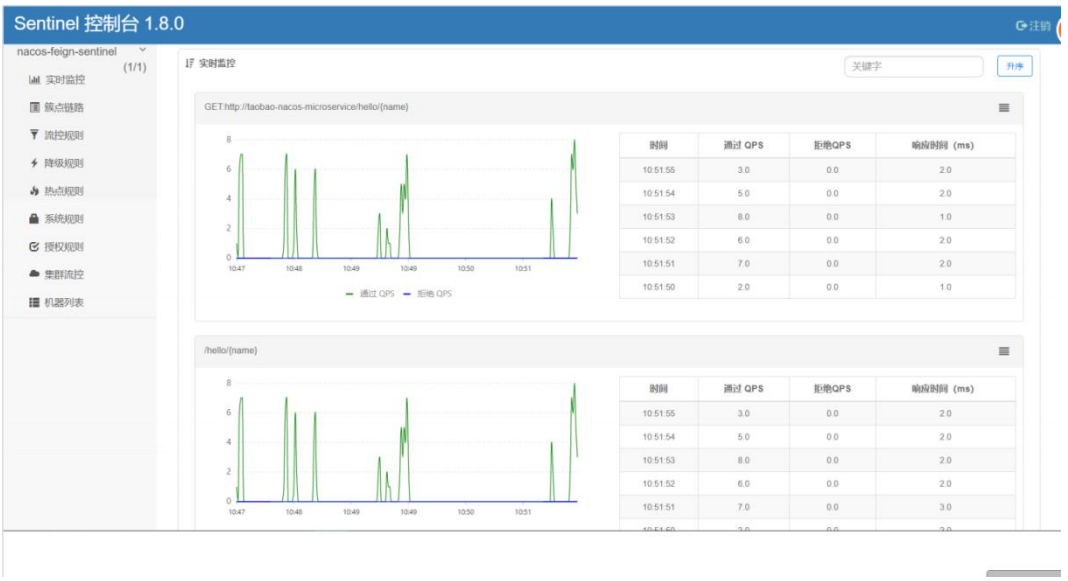
```
<dependency>
<groupId>com.alibaba.cloud</groupId>
<artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
<version>2.2.3.RELEASE</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.alibaba.cloud/spring-cloud-starter-
alibaba-nacos-discovery -->
<dependency>
<groupId>com.alibaba.cloud</groupId>
<artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
<version>2.2.3.RELEASE</version>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

2. 配置

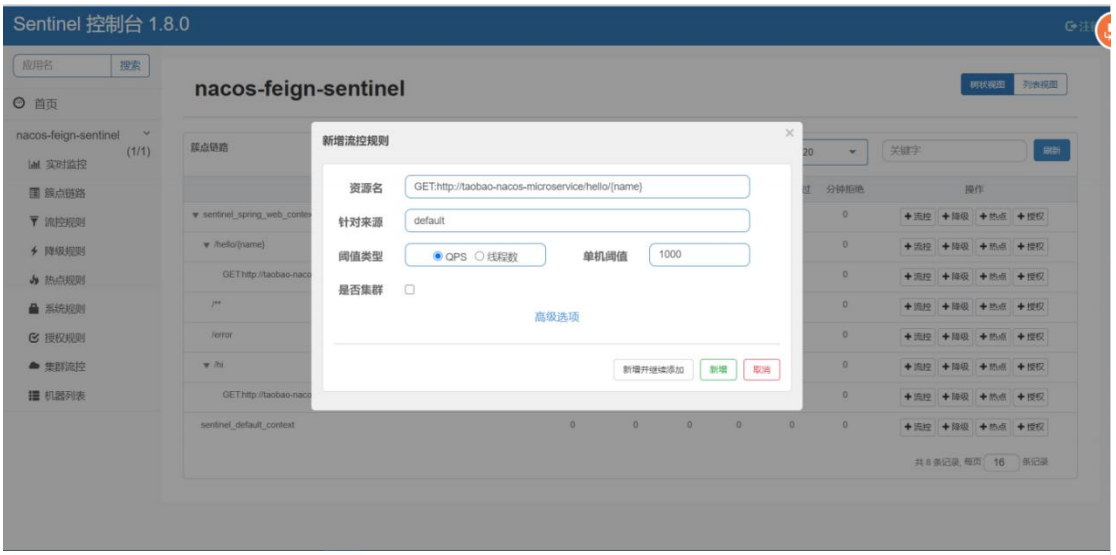
```
spring.application.name: nacos-feign-sentinel
#服务器端口
server.port: 9001
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
#sentinel
spring.cloud.sentinel.transport.dashboard=127.0.0.1:8080
feign.sentinel.enabled=true
#监控数据源要暴露地址
management.endpoints.web.exposure.include=*
```

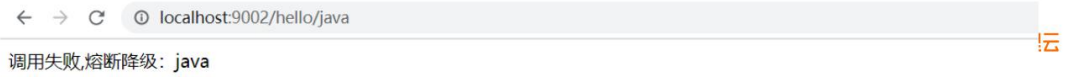
标准端的话，改成 9002。用 9001 容易出错，下一步启动一下咱们通过 902 这个端口的话，这个程序我们来调结合了熔断的这样的一个配置来实现垄断操作。9002 和 9001 的差别就在于 9001 的调动端实际是没有启用垄断这地方已经有了我们 nacos-feign 的这样一个程序来看，实时监控，用几次会出现请求次数峰值，实际跟 high school 的很像，只不过它的统计也一样，最高 3 次 7 次。

五、测试 Sentinel 微服务熔断限流



这里面差别是这两个都可以调一个是通过 hi 来调的，一个通过 hello 来调节。但现在的话咱们要熔断的话可以在直接在线进行配置，Sentinel 做这一点做的比较好，很方便，我们比如说在 high 上我们加个流控，降级也可以。





3.7 Spring Cloud 网关 Zuul 集成 Nacos 注册中心

内容简介：

一、网关 Zuul 集成 Nacos 中心

各位同学大家好，欢迎继续收看 Spring Cloud 的微服务架构实战系列课程，这一节课的话我们来讲一下 Spring Cloud 的网关如何集成 Nacos 注册中心，咱们来一起看一下如何来做，Nacos 功能非常强大，咱们前面也实战练习了好多次了，现在需要把整体 Spring Cloud 的微服架构做升级改造。这里包括整个的微服务，从基础开发到配置优化，再到底层原理的设计模式，还有现在使用最新框架改造的过程。

加引用改配置，这是最重要的。

通过实战完成几个事项：

1）Zuul 是否上线

2）通过 Zuul 能不能后台的微服务

3）路由规则是否有变化

老版本改造注意事项：

1）Nacos 中心运行正常

2）要有熔断，因为有的时候要在 Zuul 代理，或者做熔断

3）要和 Nacos 进行集成，需要用到 Nacos 引用代码

4）Nacos 中心运行正常

5）控制面板也是需要打开

如果是从 0 开始改造，需要注意版本问题，因为过高的版本可能会不支持。Nacos 和 Zuul 的兼容性会出现问题。

Nacos 引用

```
server.port=8201
spring.application.name=taobao-nacos-microservice
#Nacos
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

修改 Zuul 的配置代码

如果出错需要注意什么：

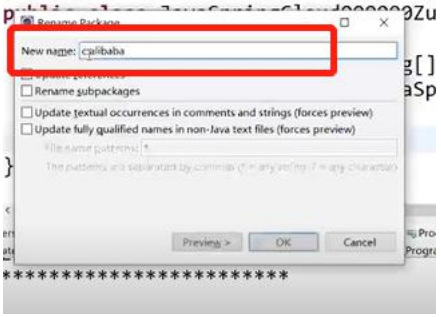
1）making one of the beans of @primary

处理方法：删除配置文件中的，不需要的引用，案例中需要删除下图代码：

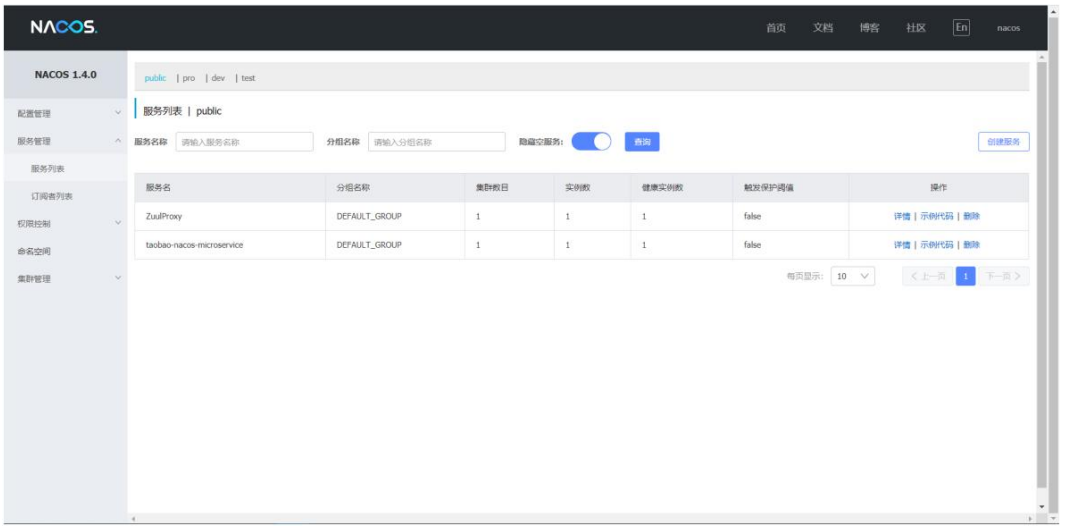
```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</
    </dependency>
```

2）Consier defining a bean of type

处理办法：扫描默认包时文件名重复，修改文件名即可

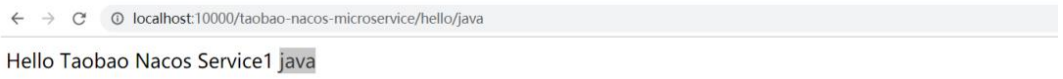


Nacos 中心查看微服务



如果不报错，就可以在 Nacos 注册中心看到 Zuul 上线。

用过网关调用：网关+服务偏移+服务方法名，效果如果图：



我们改造成了把我们的整个数据中心和网关代理来给集中起来，并做了升级改造。大家一定要做实战练习，必须练习对大家强制要求，不练习的话是掌握不了这些知识。

3.8 Spring Cloud Alibaba Seata 分布式事务

内容简介：

一、Seata 微服务分布式事务框架

二、Seata 微服务分布式架构图

三、Seata 微服务分布式事务框架实战

各位同学大家好，欢迎继续收看 Spring Cloud 微服务架构实战系列课程，这一节课我们来讲一下如何去结合 Spring Cloud 去使用阿里巴巴的微服务分布式框架（Seata），Seata 是阿里巴巴一个非常重要的微服务的解决方案。Seata 可以认为是强势支持不同的微服务框架中的事务性的一个机制。

一、Seata 微服务分布式事务框架

1. Spring Cloud Seata

1）Seata 阿里巴巴开源的分布式事务框架，原名 Fescar

2）Simple Extensible Autonomous Transaction Architecture

3）是一套一站式分布式事务解决方案。

4）2007 开始，蚂蚁金服自主研发分布式事务分布式事务中间件 XTS(eXtended Transaction Service)，在内部广泛应用并解决金融核心场景下跨数据库、跨服务数据一致性问题，最终以 DTX (Distributed Transactione Xtended)的云产品化展现，并依托蚂蚁金融云对外输出。

5）与此同时，阿里巴巴中间件团队发布 TXC (Taobao Transaction Constructor)，为集团提供分布式事务服务，于 2016 年对 TXC 进行产品化改造，形成 GTS（Global ransaction Service），依托阿里云对外输出。

6）前身是 2019 年 1 月，阿里巴巴中间件团队发起了开源项目 Fescar（Fast& EaSy Commit And Rollback），和社区一起共建开源分布式事务解决方案。

7）2019 年 5 月两个项目合并。

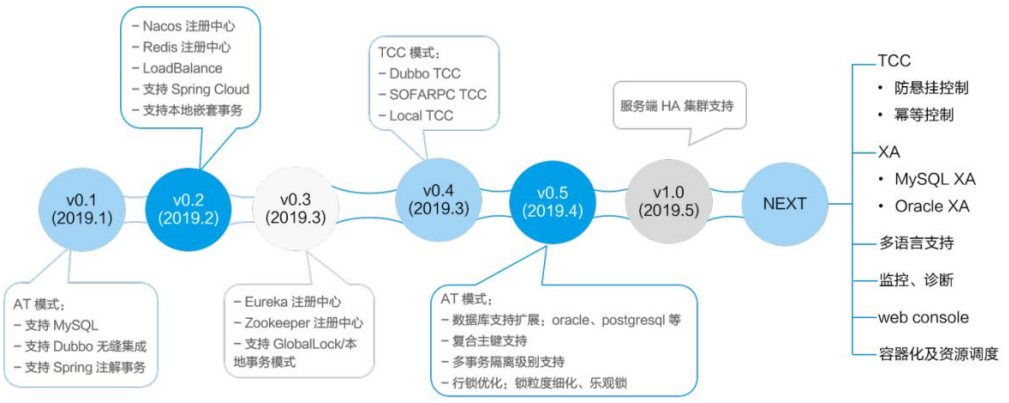
8）Seata 提供了 AT、TCC、SAGA 和 XA 事务模式，为用户打造一站式的布式解决方案。

9）https://github.com/seata/

10）支持 Dubbo、Spring Cloud 微服务分布式事务。

微服务本身是很难去支持分布式事务，它的一个很大的弱点，原因是在于它的协议，默认使用的是 AGV 协议，Rs 协议本身并不支持事物特性。作为企业级开发的话，有很多重要的要求，但是 Spring Cloud 不支持。目前些场景下会用到基于微服务下的事务机制。微服务架构中那些事物绝大部分，都是弱事务机制，并没有实现真正意义上的事务的 Sid 特性。

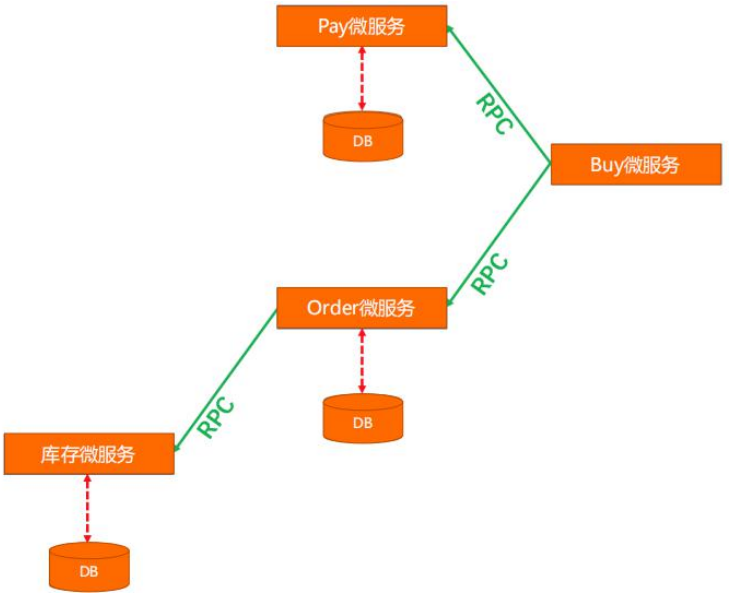
Seata 早期叫 fast（fast and easy），是阿里巴巴的集团的那两个项目后面合并了，白了就是更简单更好用的这样的一个分布式事务框架。不仅支持开放的 Double，包括其他的一些事务。



阿里 Seata 发展路线图

阿里 Seata 发展在不断迭代演化过来的，在微服务架构当中，可以不断的去切换不同的组件，还且可以不断的去升级改造。

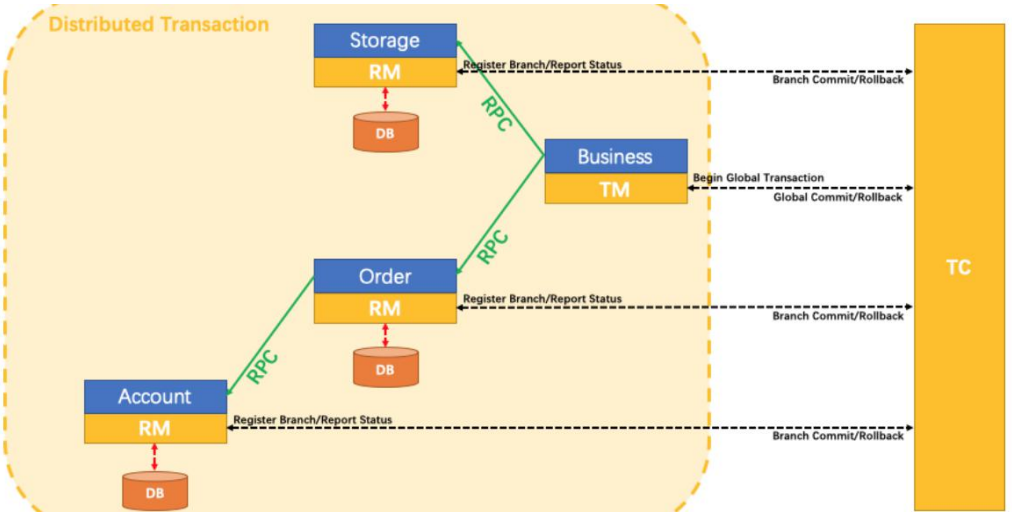
二、Seata 微服务分布式架构图



电商交易场景（无事务分布式架构图）

微服务架构中大部分做的都是消息补偿，比如说搞个消息中间件，更新一下对面的服务或者搞个定时任务等，这种机制用的比较多。真正使用分布式事务的比较少，主要是考虑到性能问题。

模拟下场景：比如用户要购买过程中，用户生成了一个订单，系统就生成订单服务，快递服务等，如果在抢购过程还要考虑库存问题，比如库存只有 5 条，但是订单购买了 10 条，要做判断是否成功，这个中间可能需要不同的数据来进行判断。还有就是数据的同步，比如同步积分，同步优惠券等一系列问题。这个在实际上微服务架构中这是很难实现的，目前阿里的 Seata 开源之后，就给我们在微服务架构提供一个非常重要的解决方案。



Seata 分布式事务架构图

每个技术框架、每个方案是在特定场景下使用的，并不强迫大家去使用，我们去了解它的原理，能够为我们以后架构设计能够多个选择。让大家能够更合理的进行技术选型，我们说很重要的原则，在做架构设计时，要综合考虑，包括成本，需求等等问题。

三、Seata 微服务分布式事务框架实战

Seata分布式事务改造

- 简单的注解
- @GlobalTransactional
- 直接标注在事务方法上即可
- @GlobalTransactional
- public void purchase(String userId, String commodityCode, int orderCount) {
- ……
- }

现在框架的机制大部分是基于补偿机制，它虽然模拟的早期分布式两提交协议，但不是真正意义上，没有强制去锁库，强制去搞分布式的一个所谓的概念，实际是尝试插入数据，然后在删除的。实际上是通过逻辑上的代码去模拟分布式事务。这样子的一个差别。其中有一个比较重要的注意的 global section，叫全局事务。这设计模式，像提交协议包括一些阶梯的分布式框架，一定会有跟实物，事务的协调器，还有资源管理器这些角色参与进来，这个事务属于强事务，会严重影响系统性能。现在这些高等化系统绝大部分在事务这一块的话是选择了就是柔性事务。

Seata实战依赖

```
<dependency>
    <groupId>io.seata</groupId>
    <artifactId>seata-all</artifactId>
    <version>${seata.version}</version>
</dependency>
```

代码需要加入 Seata 的依赖，可以直接去但是也要考虑一个版本的兼容性问题，阿里贡献的版本和实际 Spring Cloud 有存在 1~2 个版本的差异，需提前做测试下。

Seata实战依赖

```
<dependency>
<groupId>com.alibaba.cloud</groupId>
<artifactId>spring-cloud-starter-alibaba-seata</artifactId>
<version>2.2.1.RELEASE</version>
</dependency>
```

这里面也提供了一个统一的简化依赖包，但是也把配置放到文件里面，就直接去拿所有的关联包，咱们看看代码主要给大家是做一个扩展，Seata 支持的几种模式，都是模拟分布式事务的场景，实际项目大家使用一定要慎重，本身事务会有更多的交易通信，协调工作，还有影响到系统。

Seata 提供了 AT、TCC、SAGA 和 XA 事务模式，在使用的过程一定注意下。

3.9 Spring Cloud Gateway 微服务新网关实战

内容简介：

一、Spring Cloud Gateway 网关

二、Spring Cloud Gateway 网关架构

三、实战 Spring Cloud Gateway

Gateway 本身框架不断进行迭代，除了官方共建以外，社区贡献了许多技术组件，像麦飞公司贡献了 Spring Cloud Gateway 早期的微服务的许多核心组件。对于整个 Java 的微服务发展做出了非常巨大的贡献，但是官方披露公司一直希望能够进行云原生企业化，包括有可能会走一些收费路线，麦飞贡献了很多代码，但是实际没得到多少好处。

一、Spring Cloud Gateway 网关

1. Spring Cloud Gateway 新特性

1）Built on Spring Framework 5, Project

2）Reactor and Spring Boot 2.0

3）Able to match routes on any request

4）attribute.

5）Predicates and filters are specific to routes.

6）Hystrix Circuit Breaker integration.

7）Spring Cloud DiscoveryClient integration

8）Easy to write Predicates and Filters

9）Request Rate Limiting

10）Path Rewriting

主要特点是你现在用 Spring Cloud Gateway Spring Framework 在 5 年以上。继 Spring boot2.0 往上的版本它里面有一些编程语法。这边有个词叫魏池，魏池的话实际上我们说主要是做或者说自定义的一些代码的扩展，主要我们比如说做过滤器或者说路由的一些定自定义的开发工作另外也自己继承 Hystrix 的组件，也有说叫 Discoveryclient 贡献服务的客户端，所以很多技术组件还依然存在，只不过有些是官方在后续的更新中，我们知道 2020 以后可能就不再默认集成。

2. Spring Cloud Gateway 新特性

1）基于 Spring 5，Project Reactor 和 Spring Boot 2.0 构建。

2）能够匹配任何请求路由。

3）特定路由专用 Predicate 谓词和过滤器 Filter 特。

4）集成 Hystrix 断路器。

5）集成 Spring Cloud DiscoveryClient。

6）易于编写谓词和过滤器。

7）请求速率限制。

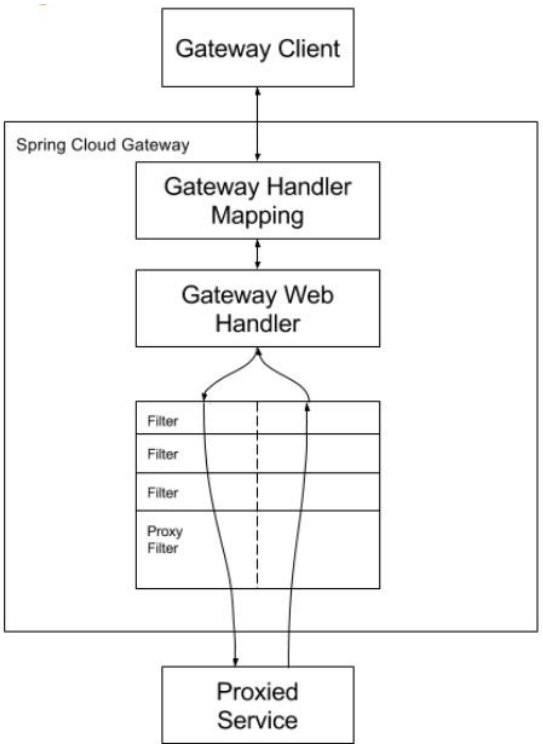
8）路径重写。

9）非阻塞 API，响应式 API，高并发。

这里面比较有意思的是当细节已经有了，之前梳理里面也已经集成了服务，重要的是我们实际想变成异步的非阻塞 Api，其实对后期要高并发吞吐量会有新的改善，也不是绝对的，要做压测才能够体现出来技术差异，出了一个结构，大家发现里面其实有很多相似点，比如这里面其实也有请求处理。

二、Spring Cloud Gateway 网关架构

1. Spring Cloud Gateway 架构图



普通的架构师会模仿架构就可以，但是高级或专家架构师不仅要能开发，还能做优化，还要做一些架构关键底层的设计扩展，这是对于高级岗位的要求。

2. Gateway 核心组件

Route 是网关的基础元素，由 ID、目标 URI、断网、过滤器组成。当请求到达网关时，由 Gateway Handler Mapping 通过断言进行路由匹配（Mapping），当断言为真时，匹配到路由。

Predicate 是 Java 8 Function Predicate。输入类型是 Spring Framework SeverWebExchange。匹配 HTTP 请求，例如请求头或者请求参数。简单来说它就是匹配条件做路由。

Filter 是 Gateway 中的过滤器，在请求发出前后进行处理。

3. Route Predicate Factories 模式

1）After Route Predicate Factory
2）Before Route Predicate Factory
3）Between Route Predicate Factory
4）Cookie Route Predicate Factory
5）Header Route Predicate Factory
6）Host Route Predicate Factory
7）Method Route Predicate Factory

8）Path Route Predicate Factory

9）Query Route Predicate Factory

10）RemoteAddr Route Predicate Factory

三、实战 Spring Cloud Gateway

1. 依赖包

- <dependency>
- <groupId>org.springframework.cloud</groupId> • <artifactId>spring-cloud-star tergateway</artifactId> • </dependency>

我们做一个 Spring Cloud Gateway 项目，要加入必要的依赖，一般必要的依赖包有了，要改一下配置，这个和我们以往开发过程不太一样，因为有些参数方式不一样,我们实验的话基本上基于 Spring Cloud Gateway2。实际有一些问题是要启用路由才行，直接改完配置后，加注解启动就可以，这边还要改配，不改配的话，你可能介入以后请求无法进行转发的。核心依赖包就是 artifactId。

2. 配置文件

- spring.cloud.gateway.enabled=false

配置文件里面要启用一个开关，你可以提供给他们，也可以关闭它。我们要显示启动，这里面等于 false 的话就等于关闭，稍微注意一下。

3. 代码配置路由

代码配置路由 阿里云

```java
import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class JavaSpringCloud2000GatewayDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(JavaSpringCloud2000GatewayDemoApplication.class, args);
    }

    @Bean
    public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {
        return builder.routes()
                .route(r -> r.path("/163/**").uri("http://www.163.com"))
                .build();
    }
}
```

配置路由 阿里云

```
spring.application.name=Gateway
#服务器端口
eureka.instance.hostname=localhost
server.port=10000
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
#打开查询
eureka.client.fetch-registry=true
eureka.client.register-with-eureka=true
#gateway
spring.cloud.gateway.enabled=true
spring.cloud.gateway.discovery.locator.enabled=true
spring.cloud.gateway.discovery.locator.lower-case-service-id=true
#spring.cloud.gateway.routes[0].id=MicroService
#spring.cloud.gateway.routes[0].uri=lb://MicroService
#spring.cloud.gateway.routes[0].predicates[0]=Path=/test/**
```

前面的配置文件的方式和我们之前租的很像，有自己的名字，如果我要注册到注册中心进行挂接的话，主要是 eureka。

3.10 Spring Cloud Gateway 实战接入 Nacos 服务

内容简介：

一、Spring Cloud Gateway 网关

二、Spring Cloud Gateway 实战集成 Nacos

各位同学大家好，咱们继续讲解 Spring Cloud 微服务架构实战系列课程，继续实战 Gateway 项，Gateway 是官方出品的网管工具，接入 Nacos 注册中心进行集成。目前 Nacos 国内很多大型公司在用，作为一个分布式数据中心，能够做注册和服务发现治理，还可以做统一配置服务。

Gateway 为什么要和 Nacos 做集成，因为这两 2 个都非常优秀，是黄金搭档，Nacos 本身不仅支持 Spring Cloud，还支持 double，GO 语言等一些分布式框架来做统一的注册和治理中心。

一、Spring Cloud Gateway 网关

Spring Cloud Gateway 新特性

1）基于 Spring 5，Project Reactor 和 Spring Boot 2.0 构建。

2）能够匹配任何请求路由。

3）特定路由专用 Predicate 谓词和过滤器 Filter 特。

4）集成 Hystrix 断路器。

5）集成 Spring Cloud DiscoveryClient。

6）易于编写谓词和过滤器。

7）请求速率限制。

8）路径重写。

9）非阻塞 API，响应式 API，高并发。

Spring Cloud Gateway 可以和集成微服务的项目直接改造进行集成，但是和 Nacos 进行搭配需要注意下，因为 Nacos 有集成模式和单点模式。关于如何进行操作可以回看之前讲解过的 Nacos 实战。如果起不来可能是环境、配置文件等出现问题。

二、Spring Cloud Gateway 实战集成 Nacos

改造Spring Cloud Gateway网关

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.alibaba.cloud/spring-cloud-starter-alibaba-
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
        <version>2.2.3.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
```

改造 Spring Cloud Gateway 具体怎么操作？简单一点的话，就是之间的 Gateway 项目进行重构，升级，去支持 Nacos 服务。之前的微服也需要进行重构，去支持 Nacos 中心服务。只有 2 个对接成功之后，才可以做后续的工作。

Nacos 本身开发客户端的依赖，客户端项目去集成它，需要具备和 Nacos 通信的功能。现在 Spring Cloud 微服架构，不仅可以用麦飞的，Spring Cloud 官方的，还有阿里巴巴的，所以架构设计有个多个选择， 而且各个框架在不断的迭代。有更多的优秀的解决方案去落地你的项目。

改造Spring Cloud Gateway网关配置

```
#gateway
spring.cloud.gateway.enabled=true
spring.cloud.gateway.discovery.locator.enabled=true
spring.cloud.gateway.discovery.locator.lower-case-service-id=true
#Nacos
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

Spring Cloud Gateway 在集成 Nacos 时，可能出现一些错误，目前还是存在 Bug 的，后续官方会进行修复，需要注意下相关的配置。

改造Spring Cloud微服务

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.alibaba.cloud/spring-cloud-starter-alibaba-
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
        <version>2.2.3.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
```

改造Spring Cloud微服务配置

- #Nacos
- spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848

大家把微服务项目改造完成以后挂接到 Gateway 上进行一下测试，流程给跑通，进行测试下。

```
<!-- https://mvnrepository.com/artifact/com.alibaba.cloud/spring
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery
        <version>2.2.3.RELEASE</version>
    </dependency>
```

项目进行改造需要把依赖加进去，是 Nacos 中心的一个对接。

启动 Nacos 服务在 win10 、Linux、MAC 脚本会不太一样。目前截图界面是单点模式，集群模式可以在生成模式进行。

本地可以弄一个 Nacos 配置文件，这样子配置数据不会丢失，即使重启了也不会丢失。

```
Field autoServiceRegistration in org.springframework.cloud.client.servic
        - eurekaAutoServiceRegistration: defined by method 'eurekaAutoSe
        - nacosAutoServiceRegistration: defined by method 'nacosAutoServ
```

图（1）

上图（1）显示重启是出现一个 Bug，自动服务注册的时候出现 2 个，由于这个项目是从早期的 Spring Cloud 移植过来的，只要删除下图（2）依赖，在重启就可以正常运行。

```
26          </dependency>
27          <dependency>
28              <groupId>org.springframework.cloud</groupId>
29              <artifactId>spring-cloud-starter-netflix-eureka-client</a
30          </dependency>
```

图（2）

如配置正常是可以在 Nacos 查看 Spring Cloud Gateway，如见图 3

图 3

后续的微服务改造也是同理的，配置文件进行修改，增加 Nacos 依赖。

希望大家在实战过程中去思考，之前讲的那些设计模式和原理。如：Gateway 如何监控后台的微服务，谁来执行，多长时间拉一次服务列表，在调用的过程中有没有 High Strikes 这种现成词的概念，有没有链接池的概念，这里面都会影响到后面的性能问题。

底层的机制中，Europe 默认是 30 秒，实际淘汰一个服务实际是 90 秒，Europe 本身也有服务过期的机制，过期了会从服务列表中删除掉。

微服务架构师本身不像传统的一些简单的架构只需要做三层或者五层的改动，而是更多的不同框架，并且考虑问题需要综合全面的。微服务架构里面有许多许多的框架和方案，并且每个框架方案的话都需要你自己实际去实战配代码。微服务架构含金量十足，作为分布式架构里面，是一个巅峰，里面包含了几十种设计模式、几十种框架，而且还在不断的迭代进行更新。这里面就需要我们在不断的去学习，逼着大家不断的去进步，是个考验，是个挑战，但是同时也是一个很好的机遇。

扫一扫

免费领取同步课程

钉钉扫一扫

进入官方答疑群

开发者学院【Alibaba Java 技术图谱】

更多好课免费学

阿里云开发者“藏经阁”

海量电子书免费下载