SpringBoot和微服务

1 SpringBoot的概述及优点

1.1 概述

随着 Spring 不断的发展，涉及的领域越来越多，项目整合开发需要配合各种各样的文件，慢慢变得不那么易用简单，违背了最初的理念，甚至人称配置地狱。Spring Boot 正是在这样的一个背景下被抽象出来的开发框架，目的为了让大家更容易的使用 Spring 、更容易的集成各种常用的中间件、开源软件；

Spring Boot 基于 Spring 开发，Spirng Boot 本身并不提供 Spring 框架的核心特性以及扩展功能，只是用于快速、敏捷地开发新一代基于 Spring 框架的应用程序。也就是说，它并不是用来替代 Spring 的解决方案，而是和 Spring 框架紧密结合用于提升 Spring 开发者体验的工具。Spring Boot 以**约定大于配置的核心思想**，默认帮我们进行了很多设置，多数 Spring Boot 应用只需要很少的 Spring 配置。同时它集成了大量常用的第三方库配置（例如 Redis、MongoDB、Jpa、RabbitMQ、Quartz 等等），Spring Boot 应用中这些第三方库几乎可以零配置的开箱即用，

简单来说就是SpringBoot其实不是什么新的框架，它默认配置了很多框架的使用方式，就像maven整合了所有的jar包，spring boot整合了所有的框架 。

1.2 优点

- 为所有Spring开发者更快的入门
- **开箱即用**，提供各种默认配置来简化项目配置
- 内嵌式容器简化Web项目
- 没有冗余代码生成和XML配置的要求

2 微服务

2.1 什么是微服务

微服务是一种**架构风格**，它要求我们在开发一个应用的时候，这个应用必须构建成一系列小服务的组合；可以通过http的方式（只是其中的一种方式）进行互通。要说微服务架构，先得说说过去我们的单体应用架构。

强调的是服务的大小，他关注的是某一个点，是具体解决某一个问题/提供落地对应服务的一个服务应用，狭义的看，可以看做是IDEA中的一个个微服务工程，或者Module。

```
IDEA工具里面使用Maven开发的一个个独立的小Module，它具体是使用SpringBoot开发的一个小模块，专业的事情交给专业的模块来做，一个模块就做着一件事情。
强调的是一个个的个体，每个个体完成一个具体的任务或者功能。
```

2.2 单体应用架构

所谓单体应用架构（all in one）是指，我们将一个应用的中的所有应用服务**都封装在一个应用中。**

无论是ERP、CRM或是其他什么系统，你都把数据库访问，web访问，等等各个功能**放到一个war包内。**

- 这样做的好处是，易于开发和测试；也十分方便部署；当需要扩展时，只需要将war复制多份，然后放到多个服务器上，再做个负载均衡就可以了。
- 单体应用架构的缺点是，哪怕我要修改一个非常小的地方，我都需要停掉整个服务，重新打包、部署这个应用war包。特别是对于一个大型应用，我们不可能吧所有内容都放在一个应用里面，我们如何维护、如何分工合作都是问题。

2.3 微服务架构

- 微服务架构是一种**架构模式**。它提倡将单一应用程序划分成一组小的服务，服务之间互相协调，互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相协作，每个服务分别围绕具体的业务进行构建，并且能够被独立的部署到生产环境中。另外，应尽量避免统一的、集中式的服务管理机制，对具体的一个服务而言，应根据上下文，选择合适的语言、工具对其进行构建。
- 所谓微服务架构，就是打破之前all in one的架构方式，**把每个功能元素独立出来**。把独立出来的功能元素的动态组合，需要的功能元素才去拿来组合，需要多一些时可以整合多个功能元素。所以微服务架构是对功能元素进行复制，而没有对整个应用进行复制。
- 这样做的好处是：
  - 节省了调用资源。
  - 每个功能元素的服务都是一个可替换的、可独立升级的软件代码。
- Martin Flower 于 2014 年 3 月 25 日写的《Microservices》，详细的阐述了什么是微服务。
  - 原文地址：http://martinfowler.com/articles/microservices.html

    翻译：https://www.cnblogs.com/liuning8023/p/4493156.html
- 微服务化的核心就是将传统的一站式应用，根据业务拆分成一个个的服务，彻底的解耦合，每一个服务提供单个业务功能的服务，一个服务做一件事情，从技术角度看就是一种小而独立的处理过程，类似进程的概念，能够自行单独启动或销毁，拥有自己独立的数据库。

2.4 如何构建微服务

一个大型系统的微服务架构，就像一个复杂交织的神经网络，每一个神经元就是一个功能元素，它们各自完成自己的功能，然后通过http相互请求调用。比如一个电商系统，查缓存、连数据库、浏览页面、结账、支付等服务都是一个个独立的功能服务，都被微化了，它们作为一个个微服务共同构建了一个庞大的系统。如果修改其中的一个功能，只需要更新升级其中一个功能服务单元即可。

但是这种庞大的系统架构给部署和运维带来很大的难度。于是，spring为我们带来了构建大型分布式微服务的全套、全程产品：

- 构建一个个功能独立的微服务应用单元，可以使用springboot，可以帮我们快速构建一个应用；
- 大型分布式网络服务的调用，这部分由spring cloud来完成，实现分布式；
- 在分布式中间，进行流式数据计算、批处理，我们有spring cloud data flow。
- spring为我们想清楚了整个从开始构建应用到大型分布式应用全流程方案。

2.5 微服务优缺点

2.5.1 优点

- 单一职责原则
- 每个服务足够内聚，足够小，代码容易理解，这样能聚焦一个指定的业务功能或业务需求
- 开发简单，开发效率高，一个服务可能就是专一的只干一件事
- 微服务能够被小团队单独开发，这个小团队是2-5人的开发人员组成
- 微服务是松耦合的，是由功能意义的服务，无论是在开发阶段或者部署阶段都是独立的
- 微服务能使用不同的语言开发
- 易于和第三方集成，微服务允许容易且灵活的方式集成自动部署，通过持续集成工具，如jenkins，Hudson，bamboo等

- 微服务易于被一个开发人员理解、修改和维护，这样小团队能够更关注自己的工作成果。无需通过合作才能体现价值
- 微服务允许利用融合最新技术
- 微服务只是业务逻辑的代码，不会和HTML、CSS或其他界面混合
- 每个微服务都有自己的存储能力，可以有自己的数据库，也可以有统一数据库

2.5.2 缺点

- 开发人员要处理分布式系统的复杂性
- 多服务运维难度，随着服务的增加，运维的压力也变大
- 系统部署依赖
- 服务间通信成本
- 数据一致性
- 系统集成测试
- 性能监控

2.6 技术栈

落地技术	微服务条目
服务开发	SpringBoot，Spring，Spring MVC
服务配置与管理	NetFlix公司的Archaius、阿里的Diamond等
服务注册与发现	Eureka、Consul、Zookeeper等
服务调用	Rest、RPC、GRP C
服务熔断器	Hystrix、Envoy等
负载均衡	Ribbon、Nginx等
服务接口调用（客户端调用服务的简化工具）	Feign等
消息队列	Kafka、RabbitMQ、ActiveMQ等
服务配置中心管理	SpringCloudConfig、Cherf等
服务路由（API网关）	Zuul等
服务监控	Zabbix、Nagios、Metrics、Spectator等
全链路追踪	Zipkin、Brave、Kubernetes等
服务部署	Docker、OpenStack、Kubernetes等
数据流操作开发包	SpringCloud Stream(封装与Redis、Rabbit、Kafka等发送接收消息)
事件消息总线	Spring Cloud Bus

2.7 为什么选择SpringCloud作为微服务架构

2.7.1 选型依据

- 整体解决方案和框架成熟度
- 社区热度
- 可维护性

- 学习曲线

2.7.2 当前各大IT公司用的微服务架构有哪些?

- 阿里Dobbo/HSF
- 京东JSF
- 新浪微博Motan
- 当当网DubboX

2.7.3 各微服务框架对比

	Netflix/Spring Cloud	Motan	gRPc	Thrift	Dubbo/DubboX
功能定位	完整的微服务框架	RPC框架，但整合了ZK或Consul，实现集群环境的基本的服务注册/发现	RPC框架	RPC框架	服务框架
支持Rest	是 Ribbon支持多种可插拔的序列化选择	否	否	否	否
支持RPC	否	是（Hession2）	是	是	是
支持多语言	是（Rest形式）	否	是	是	是
服务注册/发现	是（Eureka） Eureka服务注册表，Karyon服务端框架支持服务自注册和健康检查	是（Zookeeper/consul）	否	否	是
负载均衡	是（服务端zuul+客户端Ribbon） Zuul-服务，动态路由 云端负载均衡Eureka（针对中间层服务器）	是（客户端）	否	否	是（客户端）
配置服务	Netflix Archaius Spring Cloud Config Server集中配置	是（zookeeper提供）	否	否	否
服务调用链监控	是（zuul） Zuul提供边缘服务，API网关	否	否	否	否
高可用/容错	是（服务端Hystrix+客户端Ribbon）	是（客户端）	否	否	是（客户端）
典型应用案例	Netflix	Sina	Gooogle	Fackbook	
社区活跃程度	高	一般	高	一般	2017.8重新启动
学习难度	中等	低	高	高	低

	Netflix/Spring Cloud	Motan	gRPc	Thrift	Dubbo/DubboX
文档丰富度	高	一般	一般	一般	高
其他	Spring Cloud Bus为我们的应用程序带来了更多管理端点	支持降级	Netflix内部在开发集成gRPC	IDL定义	实践的公司比较多