Spring Cloud

提供了基于微服务的分布式软件架构解决方案。

什么是微服务?

在传统的开发模式下，绝大部分的 Web 应用都是采用单体架构的风格来进行构建的，这意味着 Web 应用程序是作为单个可部署的软件制品进行交付的，所有的接口、业务逻辑、持久层都被打包成一个 Web 应用，并且部署在一台服务器上。

这种开发模式会带来诸多不便，大多数情况下，一个应用程序交由多个团队来协同开发的，每个团队负责各自不同的模块，并且会有自己定制组件来服务对应的客户。随着应用程序的规模和复杂度不断增长，多个团队协同开发一个单体应用会变得越来越复杂，假设某个团队需要修改接口，那么其他团队与之对应的接口也需要修改，同时整个应用程序都需要重新构建、测试、部署。

为了解决这一问题，微服务应用而生，微服务是一个小型的松耦合分布式服务，可以将一个大型的应用程序分解为若干个具有严格职责定义的组件，并且这些组件是便于管理的，可以与轻量级机制（通常是 HTTP 的 API）进行通信。简单理解就是分解应用程序的功能，把一个大型服务拆分成很多小服务，使它们彼此完全独立，并且可以互相通信，这样拆分之后，每个团队都可以彼此独立地开发自己的相关代码，并且进行测试和部署。

微服务的优点

- 独立性，各个服务的开发、测试、部署相互独立，比如用户服务可以拆分成一个独立的服务，而它的开发不需要依赖于其他的服务，如果用户量很大，我们可以很容易的对其进行负载。
- 敏捷性，当一个新需求出现时，特别是在一个庞大的项目系统中，你得去考虑各方面问题，兼容性，和其他业务模块的整合等等，而使用微服务则可以直接跳过这些费时又烧脑的环节。
- 实现更加灵活，传统的项目开发中，基本上一个大型项目就是基于同一种语言来开发，这种方式对于项目开发就会有很多限制，而使用微服务进行拆分之后，各个服务之间就消除了这种限制，只需要保证每个微服务对外提供的接口可用，至于使用什么语句，什么框架来开发通通不用关心。

微服务的不足

- 拆解难度，微服务的拆分是基于业务的，不是随心所欲，想怎么拆就怎么拆，那么问题来了，由谁来拆，怎么拆？这就给多团队协作沟通带来了很多挑战。
- 沟通成本，当服务调用方需要使用某服务的接口时，首先需要找到服务的提供方，通常在一个大公司中，这种场景往往是跨部门的，沟通成本可想而知，同时，如果服务的提供方需要对某个接口进行修改，也得通知各个服务调用方。
- 数据一致性，由于微服务之间是相互独立的，它们的数据也是独立的，这就会带来一个问题，当调用多个服务接口来进行操作时，如何保证各个服务数据的一致性，这即是问题，也是难点。

微服务通信

我们将大型的应用进行拆分之后，各个微服务之间需要相互调用，需要相互通信，此时微服务已经被拆分成多个独立的进程，部署在不同的服务器上，不同进程进程之间的通信需要使用远程调用机制，RPC（远程过程调用）。

通信方式：

- 一对一 / 一对多
- 同步 / 异步

常用的 RPC 框架

- Dubbo

Dubbo 是阿里的开源 RPC 框架，目前已经捐献给了 Apache。

用户、饭店、点餐平台。

- Motan

Motan 是新浪开源的 RPC 框架。

- Thrift

Thrift 由 Facebook 开发，后来捐献给了 Apache。

- gRPC

gRPC 是 Google 开源的 RPC 框架。

- Spring Cloud

Spring Cloud 是 Spring 全家桶中的重要一员，实现微服务的治理框架，当 Spring 框架已经成为 Java 的行业标准时，Spring Cloud 是微服务架构中一个十分优秀的实现方案。Spring Cloud 是完全基于 Spring Boot，可以快速搭建一个微服务应用。

微服务架构的核心由三部分组成：服务提供者，服务消费者，注册中心。

每个微服务（服务提供者，服务消费者）在启动时，将自己的信息存储在注册中心，服务消费者可以从注册中心查询服务提供者的网络信息，并通过此信息来调用服务提供者的接口。

各个微服务与注册中心通过心跳机制完成通信，每间隔一段时间就会汇报一次，如果注册中心长时间无法与某个微服务进行通信，就会自动销毁该服务。当某个微服务的网络信息发生变化时，会重新注册。同时微服务也可以通过客户端缓存将要调用的服务地址保存起来，并且通过定时任务更新的方式来保证服务的时效性，这样就可以减轻注册中心的压力，如果注册中心出现问题，也不会影响服务之间的调用。

注册中心的核心模块

- 服务注册表：用来存储各个微服务的信息，注册中心提供 API 来查询和管理各个微服务。
- 服务注册：微服务在启动时，将自己的信息保存在注册中心。
- 服务发现：查询需要调用的微服务的网络信息，IP、端口。
- 服务检查：通过心跳机制与完成注册的各个微服务进行通信，如果发现某个微服务长时间无法访问，则销毁该服务，

Spring Cloud 具体是通过 Eureka 来实现注册中心的。

Eureka 是一个 REST 风格的基础服务，它是由 Netflix 提供，Spring Cloud Eureka 是 Spring Cloud Netfilx 的一个组件。

Spring Cloud Eureka 由两部分组成：

- Eureka Server 注册中心
- Eureka Client 服务注册（服务提供者，服务消费者）

Eureka Server 是提供服务注册与发现功能的服务端，需要注册的服务通过 Eureka Client 连接到 Eureka Server 中完成注册，并通过心跳连接来实现对已经注册的各个服务的状态监控。

Eureka Server 就相当于一个点餐平台，它已经为我们实现了买家和卖家的注册管理功能，并且还具备管理买家和卖家信息的功能。

- 创建父工程，pom.xml如下

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.southwind</groupId>
    <artifactId>myspringcloudtest</artifactId>
    <version>1.0-SNAPSHOT</version>

    <!-- 引入Spring Boot依赖 -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.7.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- 解决JDK9以上版本没有JAXB的冲突 -->
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.3.0</version>
        </dependency>
        <dependency>
            <groupId>com.sun.xml.bind</groupId>
            <artifactId>jaxb-impl</artifactId>
            <version>2.3.0</version>
        </dependency>
        <dependency>
            <groupId>com.sun.xml.bind</groupId>
            <artifactId>jaxb-core</artifactId>
```

```
            <version>2.3.0</version>
        </dependency>
        <dependency>
            <groupId>javax.activation</groupId>
            <artifactId>activation</artifactId>
            <version>1.1.1</version>
        </dependency>
    </dependencies>

    <!-- 引入Spring Cloud依赖，管理各个组件 -->
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Finchley.SR2</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

</project>
```

- 在父工程下创建 Module，实现 Eureka Server 注册中心，pom.xml 如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>myspringcloudtest</artifactId>
        <groupId>com.southwind</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>eurekaserver</artifactId>

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

</project>
```

- 创建配置文件 application.yml，添加 Eureka Server （注册中心）相关配置。

```yaml
server:
  port: 8761
eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
    service-url:
      defaultZone: http://localhost:8761/eureka/
```

`server.port`：当前 Eureka Server 的服务端口。

`eureka.client.register-with-eureka`：是否将当前的 Eureka Server 服务作为客户端进行注册。

`eureka.client.fetch-registry`：是否获取其他 Eureka Server 服务的数据。

`eureka.client.service-url.defaultZone`：注册中心的访问地址。

- 创建启动类。

```java
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class,args);
    }
}
```

`@SpringBootApplication`：声明该类是 Spring Boot 服务的入口。

`@EnableEurekaServer`：声明该类是一个 Eureka Server 微服务，提供服务注册，服务发现的功能，即注册中心。