# 第一章：微服务架构概述

## 微服务的优点：

1. 易于开发和维护：一个微服务一个特定的业务功能，开发和维护单个微服务相对简单，整个应用由若干个微服务构建而成。单个微服务启动较快。局部修改容易部署。
2. 技术栈不受限，结合项目业务和团队特点合理选择技术栈。
3. 按需伸缩，根据需求，实现细粒度的扩展。

## 微服务架构面临的挑战

1. 运维要求提高，必须保证几百个服务的正常运行与协作
2. 分布式固有的复杂性：系统容错、网络延迟、分布式事务
3. 接口挑战成本高，如果修改某一个微服务的API，可能所有使用了该接口的微服务都需要调整
4. 重复劳动，某些服务可能使用到相同的功能，但这个功能还未达到分解为一个微服务的程度，则各个服务都需开发这个功能，代码重复。

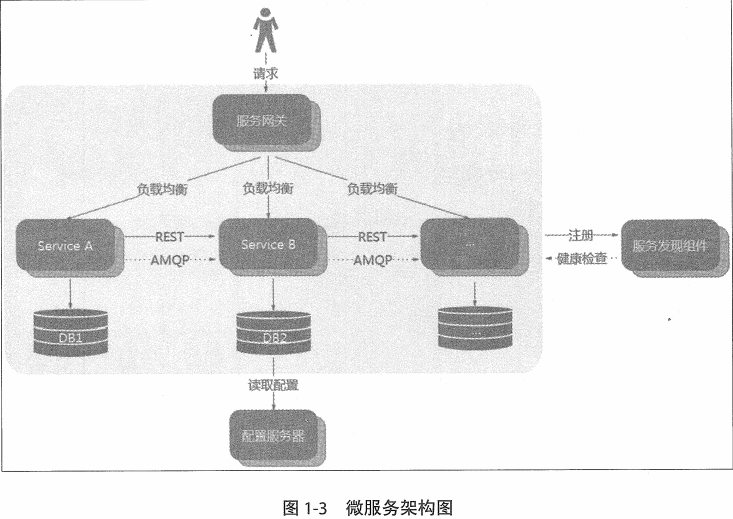
## 微服务的设计原则

1. 单一职责原则：同一个单元应只关注整个系统功能中单独的一部分。
2. 服务自治原则：每个微服务应具备独立的业务能力、依赖与运行环境。
3. 轻量级通信机制：微服务之间应该通过轻量级的通信机制进行交互。可以跨语言跨平台，例如REST协议。而相反Java的RMI协议不符合要求因其绑定Java语言。常用协议有：REST、AMQP、STOMP、MQTT等。
4. 微服务粒度：如何根据实际情况使用合理的粒度划分微服务。

## 实现微服务架构

选择使用SpringCloud作为微服务开发框架，具备开箱即用的生产特性，可大大提升开发效率；为微服务架构提供了完整的解决方案。其他开发框架或方案有：Dubbo、Dropwizard、Armda等。

架构图：



# 第二章：微服务开发框架SpringCloud

## SpringCloud简介

在SpringBoot基础上构建的，用于快速构建分布式系统的通用模式的工具集。

SpringCloud有一下特点：

1. 约定优于配置
2. 适用各种环境
3. 隐藏了组件的复杂性，提供声明式、无XML的配置方式
4. 轻量级组建：Eureka、Zuul等
5. 为微服务架构提供非常完整的支持：配置管理、服务发现、断路器、微服务网关等/
6. 选型丰富，可以按需灵活挑选。

# 第三章：开始使用SpringCloud实战微服务

使用JDK1.8 + SpringBoot 1.4.3 RELESE + Maven 3.3.9构建项目。

使用微服务构建的是分布式系统，微服务之间通过网络进行通信。

1. 开始搭建项目，Maven项目，pom依赖如下：

<?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.gjsyoung</groupId>  
 <artifactId>SpringCloudTest</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <!-- 引入SpringBoot -->  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.4.3.RELEASE</version>  
 </parent>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <java.version>1.8</java.version>  
 </properties>  
  
 <dependencies>  
 <!-- SpringMVC -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <!-- Jpa -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 <version>1.5.15.RELEASE</version>  
 </dependency>  
 <dependency>  
 <groupId>org.hibernate.javax.persistence</groupId>  
 <artifactId>hibernate-jpa-2.1-api</artifactId>  
 <version>1.0.0.Final</version>  
 </dependency>  
  
 <!-- Mysql数据库 -->  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 </dependency>  
  
  
 </dependencies>  
  
 <!-- 引入SpringCloud的依赖 -->  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>Camden.SR4</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <!-- 添加SpringBoot的Maven插件 -->  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
</project>

1. 创建用户实体类、Repository、Controller

@Entity  
public class User {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
  
 @Column  
 private String username;  
  
 @Column  
 private String name;  
  
 @Column  
 private Integer age;  
  
 @Column  
 private BigDecimal balance;  
}

public interface UserRepository extends JpaRepository<User,Long> {}

@RestController  
public class UserController {  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @GetMapping("/{id}")  
 public User findById(@PathVariable Long id){  
 User one = userRepository.findOne(id);  
 return one;  
 }  
  
}

@GetMapping是Spring4.3提供的新注解。相当于@RequestMapping（Method = GET）

1. 编写SpringBoot的启动类

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

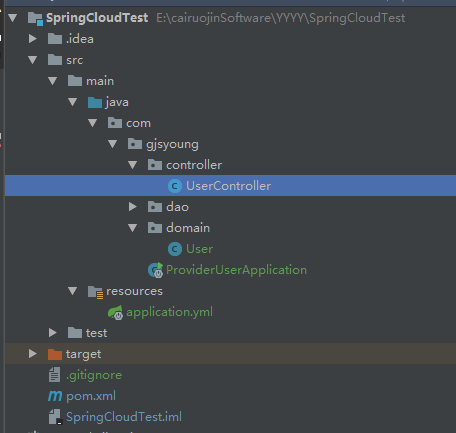
@SpringBootApplication是组合注解，整合了@Configuration、@EnableAutoConfiguration和@ComponentScan注解，并开启了SpringBoot程序的组件扫描和自动配置功能。

1. 编写配置文件application.yml

**server:  
 port:** 8080  
  
**logging:  
 com.gjsyoung.SpringCloudTest:** DEBUG  
  
  
**spring:***# profiles:  
# active: dev* **jpa:  
 generate-ddl:** false  
 **show-sql:** true  
 **hibernate:  
 ddl-auto:** none  
  
 **datasource:** *# 数据库地址* **url:** jdbc:mysql://111.230.196.33:3306/eiaproject?useSSL=false  
 **username:** cairuojin  
 **password:**   
 **driver-class-name:** com.mysql.jdbc.Driver

yml的文件格式是YAML（Yet Another Markup Language）编写的文件格式，SpringBoot和SpringCloud支持使用prop或者yml作配置文件。

此时即可测试SpringBoot项目搭建是否成功，当前项目文件如下：



同理编写一个服务的提供者，通过RestTemplte调用用户微服务的API。根据ID查询结果。