# springcloud开发框架

## 分布式开发与SpringCloud简介

### 分布式开发简介

对于分布式开发设计，最初就有了以下几点思考：

1. 分布式开发如何可以让代码更加安全
2. 分布式开发的时候如何进行有效的通信
3. 在进行分布式处理的时候到底如何进行程序的功能划分

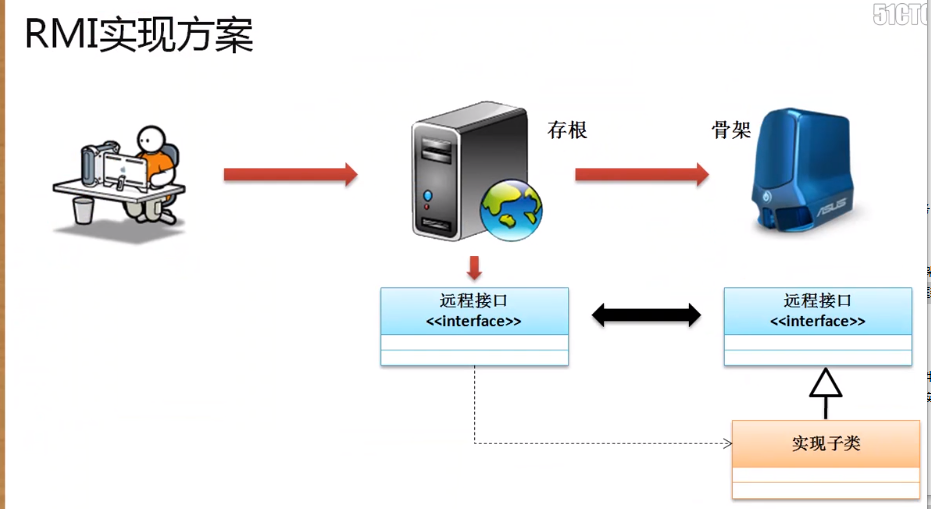
项目开发中分布式设计：

1. WEB集群：考虑到多用户并发访问的处理速度；
2. 业务中心：在进行一些庞大的项目设计过程中，应该有更加完善的业务处理，这样所有的客户端（服务器）直接调用这些业务中心的操作就可以完成具体的功能；
3. 数据库集群：解决了数据的存储问题，以及数据的分片管理。

对于分布式项目开发按照历史的发展经历过如下的一些技术：

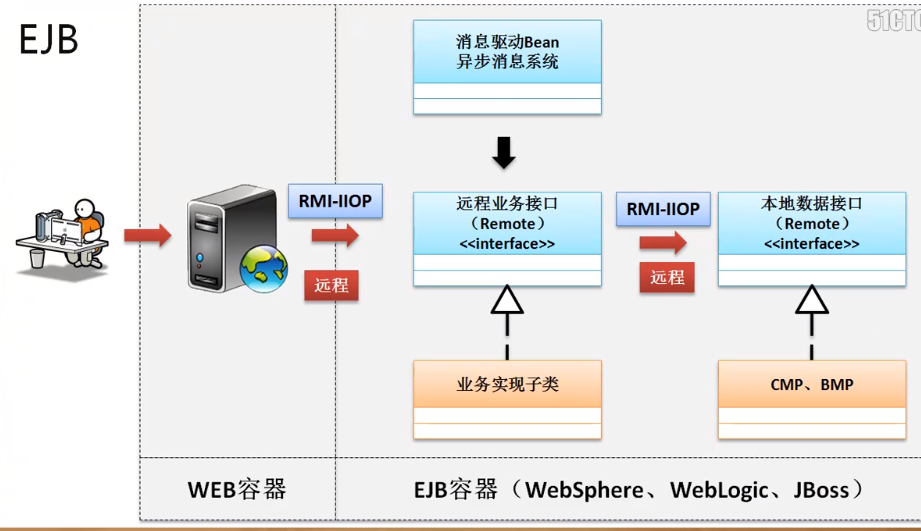
1. CORBA：公共对象请求代理架构，他是一种开发的标准，而且也是许多语音都支持的开发标准。
2. RMI(远程方法调用)：该技术是SUN提出来的，其是希望与CORBA进行市场竞争，很多公司不认可此技术；

|- 在java里面提出了远程接口的概念，不过RMI的实现不太好。



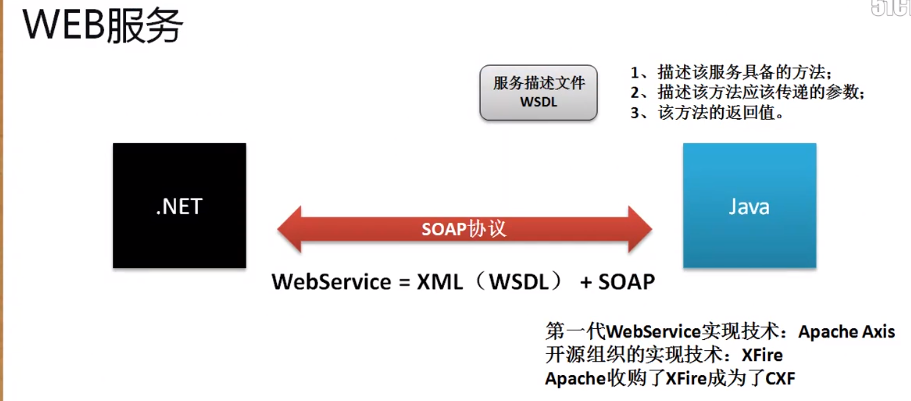
开发人员除了编写接口外，还要写存根和骨架。这比较麻烦。

1. 后来SUN事务设计师，RMI很好用（他们自己觉得），但是没有CORBA广泛，于是开始思考能否基于CORBA做一些更好的设计？于是产生了RMI-IIOP协议，其用在了EJB上。



EJB留给世界上的只是它的优秀理论，和糟糕的实现，这个理论被一些开源框架无限制的扩充与实现着。

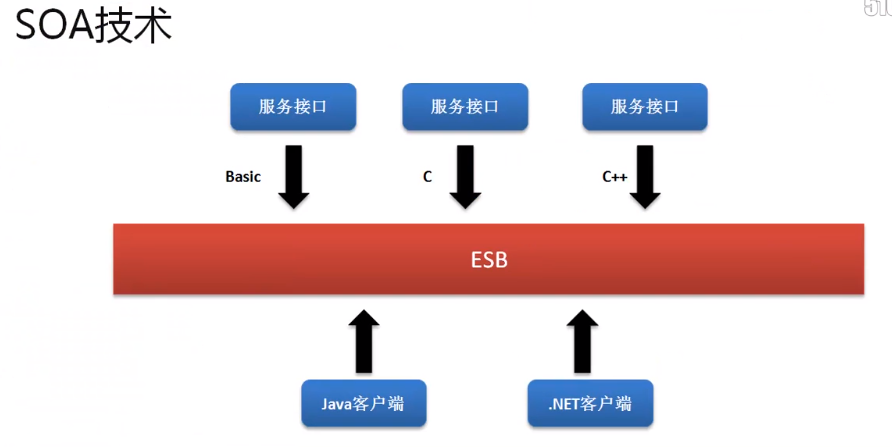
1. .NET出现，而后因为J#的问题，微软和SUN就彻底对立，后来整个行业就乱了，有两套企业系统架构，公司就面临选择，于是尴尬的局面出现了。选择谁？后来随着软件行业的继续推广，出现了XML标记语言，很多公司就觉得应该用XML作为数据交换的基础，整个社会著名的软件架构：WebService（web服务）登场了。



如果项目中要想使用WebService技术进行开发，择优如下问题：

* 速度太慢了，处理的速度不行；
* 如果要想采用远程接口的方式调用，则要利用开发工具生成一堆的工具类代码；

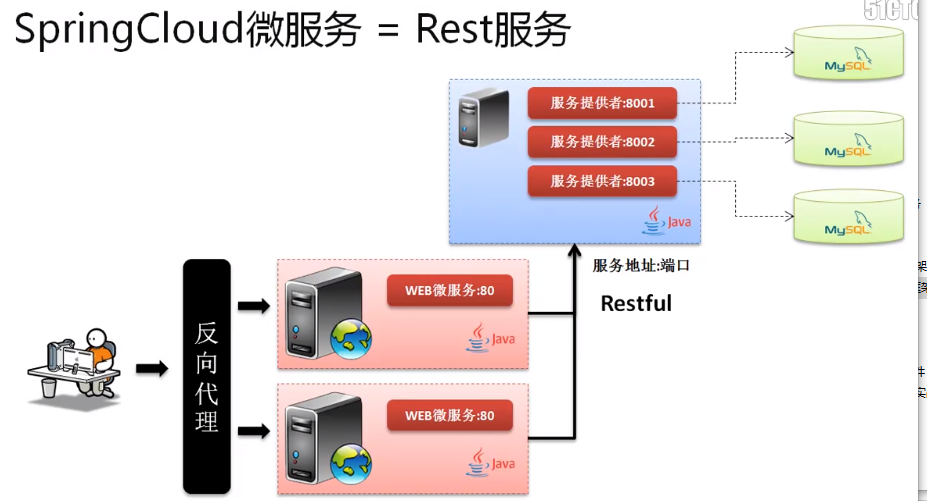
后来2005年的时候WebService在整个世界上继续发酵，形成了又一大核心神器：SOA(面向服务架构)，它提出了企业服务总线（ESB）的概念。



(5)SOA提出的服务总线又成为了新一代分布式的讨论需求，人们发现如果将所有的服务统一管理起来，就成为了服务总线，所有的开发者直接调用里面的服务就完成一些功能。后来又不断出现了许多的RPC开发技术，其中具有代表性的dubbo(阿里提供)开发技术。不过这个时候成长最快的就是Rest,因为JSON的广泛应用，如同最初的XML技术造就了WebService一样，JSON造就了Rest服务，人们认为我们的操作应该更加简单一些，Rest还是一个未成型的标准，随着Rest的广泛认可，Spring终于得到了一个新机会：利用Rest进行RPC技术实现，这样的操作速度很快，而且占用额网络带宽要少。在SpringCloud中真正的将整个Rest作为了RPC实现技术,并且这一技术已经开始出现有行业统一之势。SpringBoot也依照与SpringCloud的开发技术，可以实现项目的打包分布以及单独运行，这一点符合当前云时代的开发要求。

### SpringCloud开发

SpringCloud整体核心架构只有一点：Rest服务，也就是说在整个SpringCloud配置过程之中，所有的配置处理都是围绕着Rest完成的，在整个Rest处理之中，一定要有两个：服务的提供者（provider）、服务的消费者（consumer），所以SpringCloud的基础结构就如下所示：

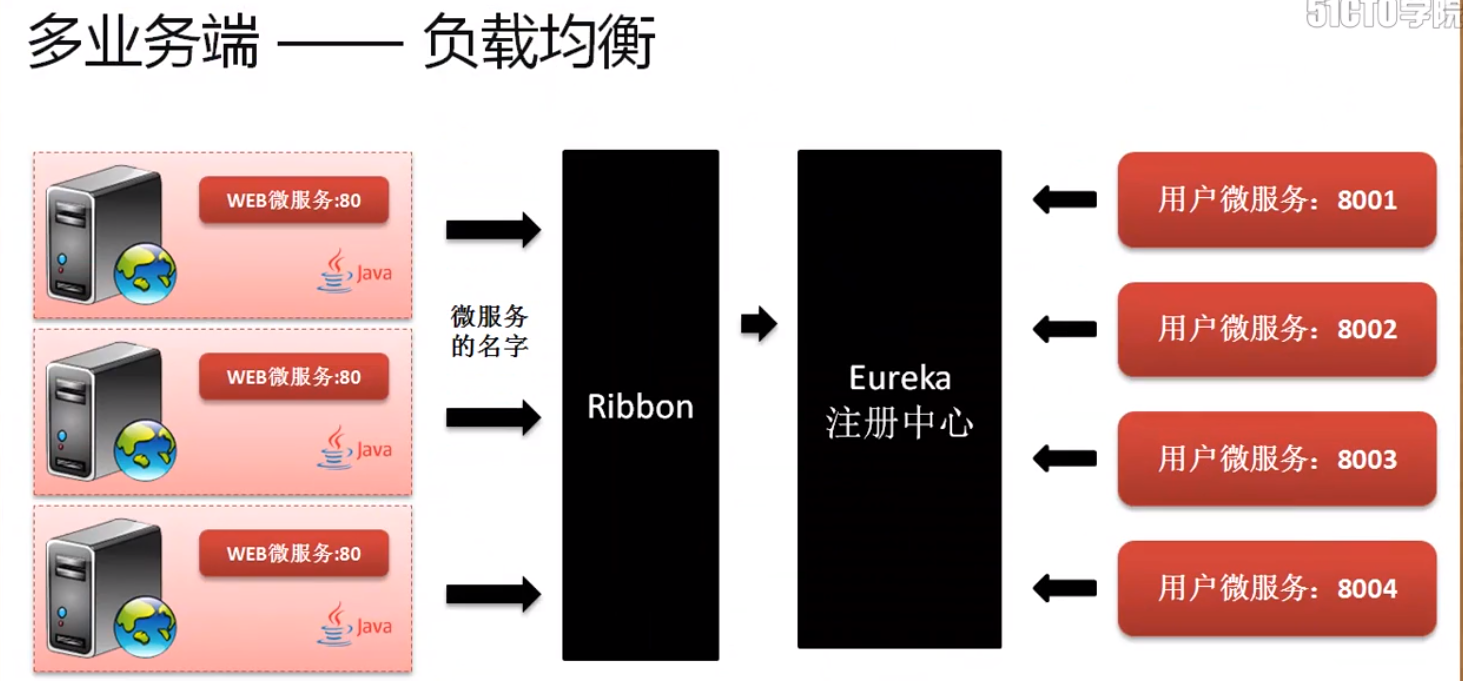




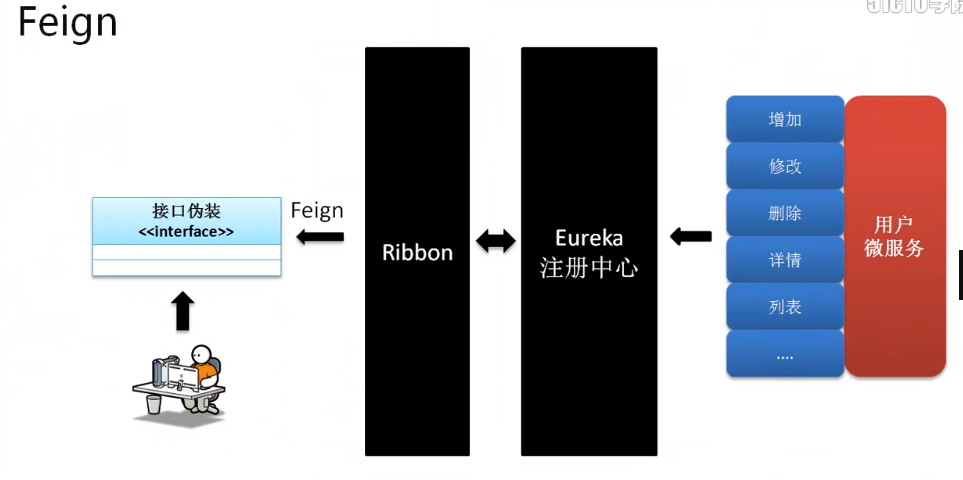
既然SpringCloud的核心是Restful结构，那么如果想要更好的去使用Rest微服务，还需要考虑如下几个问题：

1. 所有的微服务地址一定会非常多，所以为了统一管理这些地址信息，也为了可以及时的告诉用户哪些服务不可用，所以应该准备一个分布式的注册中心，并且该注册中心应该支持HA机制。
2. 为高速并且方便的进行所有服务的注册操作，在SpringCloud里面提供有一个Eureka的注册中心（虽然spring支持Zookeeper和Consul,但是性能都很差）。

对于整个的WEB端架构（SpringBoot实现）可以轻松过方便的进行WEB程序的编写，而后利用Nginx或Apache实现负载均衡，但是WEB端出现了负载均衡，那么业务端呢？应该也提供有多个业务端进行负载均衡。那么这个时候就需要将所有需要参与到负载均衡的业务端在Eureka中进行注册。

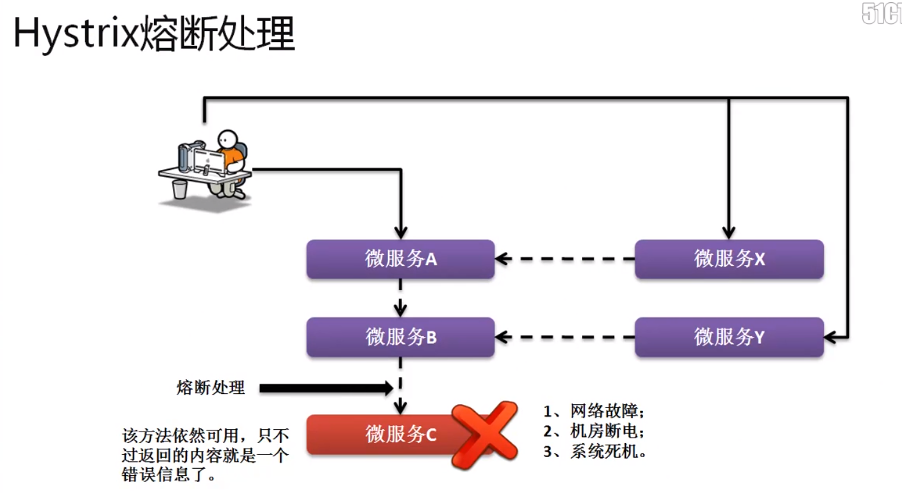


在客户端进行Rest架构调用的时候，往往都需要有一个调用地址，即使现在使用了Eureka作为注册中心，它也需要有一个明确的调用地址，可是所有的操作都利用调用地址的方式来处理，那就比较麻烦，而持续开发者最方便应用的工具是接口，所以现在就希望可以将所有的rest服务的内容以接口的形式调用，所以它又提供了一个feign技术，利用此技术可以伪造接口实现。

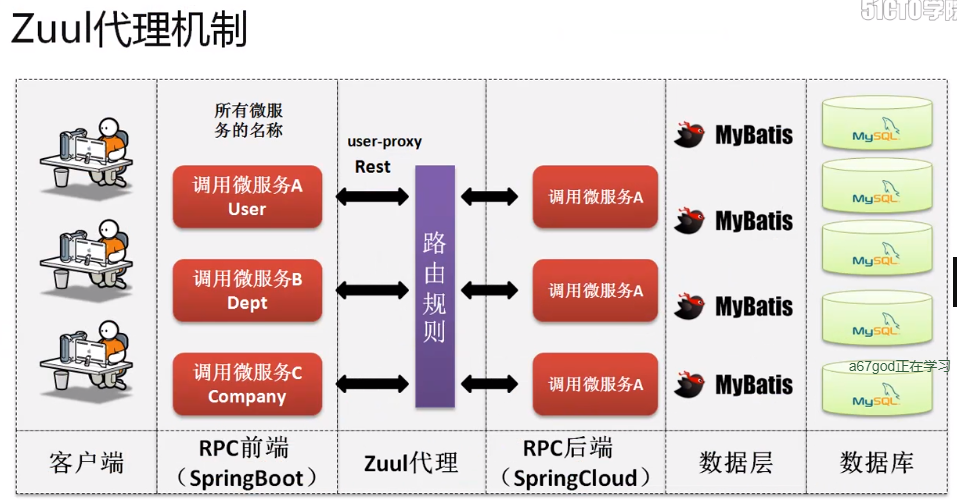


1. 在微架构设计的时候由于牵扯到的问题还是属于RPC，所以必须考虑熔断处理机制，实际上所谓的熔断就像生活中使用的保险丝一样，有了保险丝，在一些设备出现故障之后依然可以保护家庭电器可以正常使用，如果现在有若干个微服务，并且这些微服务之间允许互相调用，例如：A微服务调用了B的微服务、B的微服务又调用了C的微服务。

如果在实际的项目设计过程中没有处理好熔断机制，那么就会产生雪崩效应，所以为了防止这样的问题出现，SpringCloud里面提供有一个Hystrix熔断处理机制，以保证某一个微服务即使出现可以正常使用。

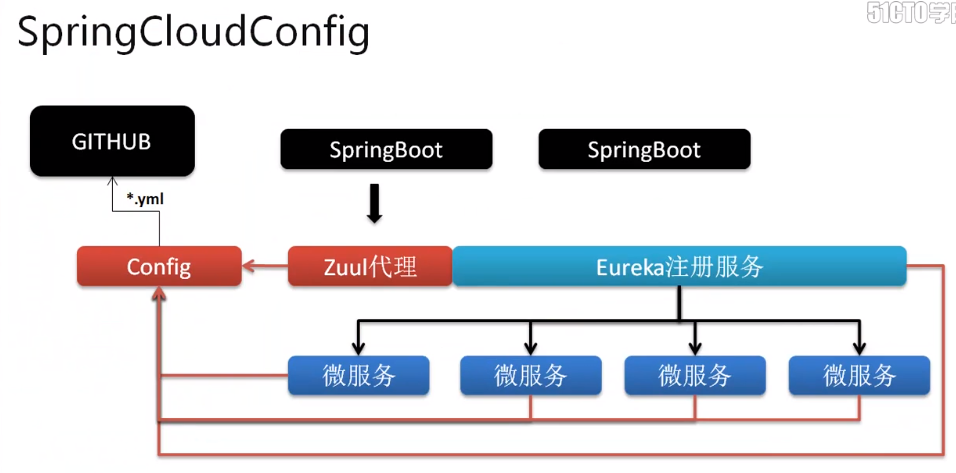


1. 在进行为服务访问的时候还有一点是非常可怕的。



通过Zuul的代理用户只需要知道指定的路由的路径就可以访问指定的微服务的信息，这样更好的体现了java中“Key=value”的设计思想，所有的微服务进行代理之后也可以更加合理的进行名称的隐藏。

1. 在SpringBoot学习的时候一直强调一个问题：“零配置”，本质上是希望不编写任何配置文件，但是事实上这一点并没有完全的实现，因为在整个设计里面，依然会提供有application.yml（application.properties）配置文件。在微服务的创建过程中，一定会有成百上千个微服务的信息出现，这些配置文件的管理就成为了问题，例如：某一天机房变更，那么所有的Ip地址都有可能发生变化，这样对于程序的维护是不方便的，为了解决这样的问题，在SpringCloud设计的时候提供有一个SpringCloudConfig的程序组件，利用这个组件就可以直接基于GIT或SVN来进行配置文件的管理。



在整体设计上SpringCLoud更好的实现了RPC的架构设计，而且使用了Rest作为通讯的基础，这一点是它的成功之处，同时由于大量的使用了netflix公司的产品技术，所以这些技术也有可靠的保证。

## 搭建Rest服务

对于Rest基础架构实现处理时SpringCloud核心所在，其基本的操作形式在SpringBoot中已经有了基本的认识。

### 配置项目开发环境

对于现在的项目创建一个：microcloud的整体父pom项目，然后创建三个子模块：microcloud-api模块，作为公共的信息导入模块、microcloud-provider-dept-8001作为服务提供者，负责使用mybatis与数据库的交互、microcloud-consumer-80作为微服务调用的客户端使用。

1. 创建一个新的maven项目：microcloud
2. 【microcloud】修改pom.xml文件，主要追加Springcloud与SpringBoot两个开发包的依赖关系；

注意：SpringBoot中针对于依赖包的版本并不是像传统那样采用数字的形式定义的，而是使用一序列的英国的地铁站或城市的名字来定义的。

*<?***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**>cn.ym.com</**groupId**>  
 <**artifactId**>microcloud</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
  
 <**properties**>  
 <**jdk.version**>1.8</**jdk.version**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 </**properties**>  
 <**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**> *<!-- 进行SpringCloud依赖包的导入处理 -->* <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-dependencies</**artifactId**>  
 <**version**>Dalston.SR5</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 <**dependency**> *<!-- SpringCloud离不开SpringBoot，所以必须要配置此依赖包 -->* <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-dependencies</**artifactId**>  
 <**version**>1.5.14.RELEASE</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 </**dependencies**>  
 </**dependencyManagement**>  
 <**build**>  
 <**finalName**>microcloud</**finalName**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-compiler-plugin</**artifactId**>  
 <**configuration**>  
 <**source**>${jdk.version}</**source**>*<!-- 源代码使用的开发版本 -->* <**target**>${jdk.version}</**target**>*<!-- 需要生成的目标class文件的编译版本 -->* <**encoding**>${project.build.sourceEncoding}</**encoding**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
</**project**>

1. 【microcloud-api】主要功能是提供有公共的处理类，本次预计建立一个Dept数据表（实体）：

**public class** Dept **implements** Serializable {  
 **private** Long **deptno**;  
 **private** String **dname**;  
 **private** String **loc**;  
  
 **public** Long getDeptno() {  
 **return deptno**;  
 }  
  
 **public void** setDeptno(Long deptno) {  
 **this**.**deptno** = deptno;  
 }  
  
 **public** String getDname() {  
 **return dname**;  
 }  
  
 **public void** setDname(String dname) {  
 **this**.**dname** = dname;  
 }  
  
 **public** String getLoc() {  
 **return loc**;  
 }  
  
 **public void** setLoc(String loc) {  
 **this**.**loc** = loc;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Dept [deptno="** + **deptno** + **", dname="** + **dname** + **", loc="** + **loc** + **"]"**;  
 }  
}

1. 【microcloud-provider-dept-8001】创建一个Rest提供者的项目模块，在整个模块里面主要定义数据库的脚本：

DROP DATABASE IF EXISTS "yangmi8001" ;

CREATE DATABASE yangmi8001 CHARACTER SET UTF8 ;

USE yangmi8001 ;

CREATE TABLE dept (

deptno BIGINT AUTO\_INCREMENT ,

dname VARCHAR(50) ,

loc VARCHAR(50) ,

CONSTRAINT pk\_deptno PRIMARY KEY(deptno)

) ;

INSERT INTO dept(dname,loc) VALUES ('开发部',database()) ;

INSERT INTO dept(dname,loc) VALUES ('财务部',database()) ;

INSERT INTO dept(dname,loc) VALUES ('市场部',database()) ;

INSERT INTO dept(dname,loc) VALUES ('后勤部',database()) ;

INSERT INTO dept(dname,loc) VALUES ('公关部',database()) ;

### 创建部门微服务

所谓微服务的核心本质就是JSON的传输，那么既然现在要求用mybatis进行数据库操作，所以应该在项目里面配置Druid数据库连接池，而后对外进行项目的发布。

1. 【microcloud-provider-dept-8001】修改pom.xml配置文件，追加相关依赖：

*<?***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**>microcloud</**artifactId**>  
 <**groupId**>cn.ym.com</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>microcloud-provider-dept-8001</**artifactId**>  
  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 </**properties**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>cn.ym.com</**groupId**>  
 <**artifactId**>microcloud-api</**artifactId**>

<**version**>1.0-SNAPSHOT</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>druid</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>ch.qos.logback</**groupId**>  
 <**artifactId**>logback-core</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.mybatis.spring.boot</**groupId**>  
 <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-jetty</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>springloaded</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-devtools</**artifactId**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

同时修改父项目pom.xml：

<**dependencies**>  
 <**dependency**>  
 <**groupId**>cn.ym.com</**groupId**>  
 <**artifactId**>microcloud-api</**artifactId**>  
 <**version**>1.0</**version**>  
 </**dependency**>  
  
 <**dependency**> *<!-- 进行SpringCloud依赖包的导入处理 -->* <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-dependencies</**artifactId**>  
 <**version**>Dalston.SR1</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 <**dependency**> *<!-- SpringCloud离不开SpringBoot，所以必须要配置此依赖包 -->* <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-dependencies</**artifactId**>  
 <**version**>1.5.4.RELEASE</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>5.0.4</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>druid</**artifactId**>  
 <**version**>1.0.31</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.mybatis.spring.boot</**groupId**>  
 <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  
 <**version**>1.3.0</**version**>  
 </**dependency**>  
</**dependencies**>

1. 【microcloud-provider-dept-8001】创建一个IDeptDAO数据库操作接口，这个接口里面将提供有三个数据操作方法：

@Mapper  
**public interface** IDeptDAO {  
 **public boolean** doCreate(Dept vo);  
  
 **public** Dept findById(Long id);  
  
 **public** List<Dept> findAll();  
}

1. 【microcloud-provider-dept-8001】修改application.yml配置文件，追加mybatis和服务的相关信息：

**server:  
 port:** 8001  
**mybatis:  
 config-location:** classpath:mybatis/mybatis.config.xml *# mybatis配置文件所在路径* **type-aliases-package:** cn.ym.com.vo *# 定义所有操作类的别名所在包* **mapper-locations:** *# 所有的mapper映射文件* - classpath:mybatis/mapper/\*\*/\*.xml  
**spring:  
 datasource:  
 type:** com.alibaba.druid.pool.DruidDataSource *# 配置当前要使用的数据源的操作类型* **driver-class-name:** org.gjt.mm.mysql.Driver *# 配置MySQL的驱动程序类* **url:** jdbc:mysql://localhost:3306/yangmi8001 *# 数据库连接地址* **username:** root *# 数据库用户名* **password:** 12345 *# 数据库连接密码* **dbcp2:** *# 进行数据库连接池的配置* **min-idle:** 5 *# 数据库连接池的最小维持连接数* **initial-size:** 5 *# 初始化提供的连接数* **max-total:** 5 *# 最大的连接数* **max-wait-millis:** 200 *# 等待连接获取的最大超时时间*

1. 【microcloud-provider-dept-8001】定义src/main/resources/mybatis/mybatis.config.xml文件：

*<?***xml version="1.0" encoding="UTF-8"** *?>***<!DOCTYPE configuration   
 PUBLIC "-//mybatis.org//DTD Config 3.0//EN"   
 "http://mybatis.org/dtd/mybatis-3-config.dtd"*>***<**configuration**> *<!-- 进行Mybatis的相应的环境的属性定义 -->* <**settings**> *<!-- 在本项目之中开启二级缓存 -->* <**setting name="cacheEnabled" value="true"**/>  
 </**settings**>  
</**configuration**>

5、【microcloud-provider-dept-8001】创建

src/main/resources/mybatis/mapper/cn/ym/com/Dept.xml配置文件：

*<?***xml version="1.0" encoding="UTF-8"***?>***<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"   
"http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>***<**mapper namespace="cn.ym.com.dao.IDeptDAO"**>  
 <**select id="findById" resultType="Dept" parameterType="long"**>  
 SELECT deptno,dname,loc FROM dept WHERE deptno=#{deptno} ;  
 </**select**>  
 <**select id="findAll" resultType="Dept"**>  
 SELECT deptno,dname,loc FROM dept ;  
 </**select**>  
 <**insert id="doCreate" parameterType="Dept"**>  
 INSERT INTO dept(dname,loc) VALUES (#{dname},database()) ;  
 </**insert**>  
</**mapper**>

6、【microcloud-provider-dept-8001】建立IDeptService接口，做业务实现：

**package** cn.ym.com.service;  
  
**import** cn.ym.com.vo.Dept;  
  
**import** java.util.List;  
  
**public interface** IDeptService {  
 **public** Dept get(**long** id);  
  
 **public boolean** add(Dept dept);  
  
 **public** List<Dept> list();  
}

**package** cn.ym.com.service.impl;  
  
**import** cn.ym.com.dao.IDeptDAO;  
**import** cn.ym.com.service.IDeptService;  
**import** cn.ym.com.vo.Dept;  
**import** org.springframework.stereotype.Service;  
**import** javax.annotation.Resource;  
**import** java.util.List;  
  
@Service  
**public class** DeptServiceImpl **implements** IDeptService {  
 @Resource  
 **private** IDeptDAO **deptDAO**;  
  
 @Override  
 **public** Dept get(**long** id) {  
 **return this**.**deptDAO**.findById(id);  
 }  
  
 @Override  
 **public boolean** add(Dept dept) {  
 **return this**.**deptDAO**.doCreate(dept);  
 }  
  
 @Override  
 **public** List<Dept> list() {  
 **return this**.**deptDAO**.findAll();  
 }  
  
}

7、【microcloud-provider-dept-8001】定义程序启动类：

@SpringBootApplication  
**public class** Dept\_8001\_StartSpringCloudApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(Dept\_8001\_StartSpringCloudApplication.**class**, args);  
 }  
}

8、【microcloud-provider-dept-8001】编写业务接口测试类：

1. 【microcloud-provider-dept-8001】建立DeptRest服务类（控制器）：

@RestController  
**public class** DeptController {  
 @Resource  
 **private** IDeptService **deptService**;  
  
 @RequestMapping(value = **"/dept/get/{id}"**, method = RequestMethod.***GET***)  
 **public** Object get(@PathVariable(**"id"**) **long** id) {  
 **return this**.**deptService**.get(id);  
 }  
  
 @RequestMapping(value = **"/dept/add"**, method = RequestMethod.***GET***)  
 **public** Object add(@RequestBody Dept dept) {  
 **return this**.**deptService**.add(dept);  
 }  
  
 @RequestMapping(value = **"/dept/list"**, method = RequestMethod.***GET***)  
 **public** Object list() {  
 **return this**.**deptService**.list();  
 }  
}

1. 修改本机host文件，追加一个映射路径：

127.0.0.1 dept-8001.com

1. 【microcloud-provider-dept-8001】观察Rest服务能否正常提供：
2. 调用get操作

http://dept-8001.com:8001/dept/get/1

1. 调用list操作

http://dept-8001.com:8001/dept/list

1. 调用add操作

http://dept-8001.com:8001/dept/add?deptno=100&dname=%E7%BE%8E%E5%A5%B3%E9%83%A8&loc=%E5%A4%A9%E7%95%8C

### 创建服务消费端

## SpringSecutity安全认证

## Eureka服务注册于发现框

## Ribbion负载均衡

## Feign接口转换

## Hystrix熔断机制

## Zuul路由访问

## 上传微服务

## SpringCloudConfig基础配置

## SpringCloudConfig高级配置

## SpringCloudStream简介

## SpringCloudSleuth