# 案例1 完成案例工程的项目搭建

## 1.1 父工程 demo-parent

在Idea中新建module，命名为demo-parent

pom.xml：

   <!--父工程打包方式-->  
   <packaging>pom</packaging>  
​  
   <!--spring boot 父启动器依赖-->  
   <parent>  
       <groupId>org.springframework.boot</groupId>  
       <artifactId>spring-boot-starter-parent</artifactId>  
       <version>2.1.6.RELEASE</version>  
   </parent>  
​  
   <dependencies>  
       <!--web依赖-->  
       <dependency>  
           <groupId>org.springframework.boot</groupId>  
           <artifactId>spring-boot-starter-web</artifactId>  
       </dependency>  
       <!--日志依赖-->  
       <dependency>  
           <groupId>org.springframework.boot</groupId>  
           <artifactId>spring-boot-starter-logging</artifactId>  
       </dependency>  
       <!--测试依赖-->  
       <dependency>  
           <groupId>org.springframework.boot</groupId>  
           <artifactId>spring-boot-starter-test</artifactId>  
           <scope>test</scope>  
       </dependency>  
       <!--lombok工具-->  
       <dependency>  
           <groupId>org.projectlombok</groupId>  
           <artifactId>lombok</artifactId>  
           <version>1.18.4</version>  
           <scope>provided</scope>  
       </dependency>  
       <!-- Actuator可以帮助你监控和管理Spring Boot应用-->  
       <dependency>  
           <groupId>org.springframework.boot</groupId>  
           <artifactId>spring-boot-starter-actuator</artifactId>  
       </dependency>  
       <!--热部署-->  
       <dependency>  
           <groupId>org.springframework.boot</groupId>  
           <artifactId>spring-boot-devtools</artifactId>  
           <optional>true</optional>  
       </dependency>  
​  
   </dependencies>  
​  
   <build>  
       <plugins>  
           <!--编译插件-->  
           <plugin>  
               <groupId>org.apache.maven.plugins</groupId>  
               <artifactId>maven-compiler-plugin</artifactId>  
               <configuration>  
                   <source>11</source>  
                   <target>11</target>  
                   <encoding>utf-8</encoding>  
               </configuration>  
           </plugin>  
           <!--打包插件-->  
           <plugin>  
               <groupId>org.springframework.boot</groupId>  
               <artifactId>spring-boot-maven-plugin</artifactId>  
               <executions>  
                   <execution>  
                       <goals>  
                           <goal>repackage</goal>  
                       </goals>  
                   </execution>  
               </executions>  
           </plugin>  
       </plugins>  
   </build>

## 1.2 公共组件微服务

1) 在公共组件微服务中引入数据库驱动及mybatis-plus

<?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>demo-parent</artifactId>  
       <groupId>com.demo</groupId>  
       <version>1.0-SNAPSHOT</version>  
   </parent>  
   <modelVersion>4.0.0</modelVersion>  
​  
   <artifactId>demo-service-common</artifactId>  
​  
   <dependencies>  
       <dependency>  
               <groupId>com.baomidou</groupId>  
               <artifactId>mybatis-plus-boot-starter</artifactId>  
               <version>3.3.2</version>  
       </dependency>  
         
<!--pojo持久化使用-->  
       <dependency>  
           <groupId>javax.persistence</groupId>  
           <artifactId>javax.persistence-api</artifactId>  
           <version>2.2</version>  
       </dependency>  
​  
       <dependency>  
           <groupId>mysql</groupId>  
           <artifactId>mysql-connector-java</artifactId>  
           <scope>runtime</scope>  
       </dependency>  
   </dependencies>  
​  
</project>

2) 生成数据库实体类：com.demo.common.pojo.Products

package com.demo.common.pojo;  
​  
import lombok.Data;  
​  
import javax.persistence.Id;  
import javax.persistence.Table;  
​  
@Data  
@Table(name = "products")  
public class Products {  
​  
 @Id  
 private Integer id;  
 private String name;  
 private double price;  
 private String flag;  
 private String goodsDesc;  
 private String images;  
 private long goodsStock;  
 private String goodsType;  
​  
}  
​

## 1.3 商品微服务

商品微服务是服务提供者，页面静态化微服务是服务的消费者

创建商品微服务demo-service-product，继承demo-parent

1）在商品微服务的pom文件中，引入公共组件坐标

   <dependencies>  
       <dependency>  
           <groupId>com.demo</groupId>  
           <artifactId>demo-service-common</artifactId>  
           <version>1.0-SNAPSHOT</version>  
       </dependency>  
   </dependencies>

2）在yml文件中配置端口、应用名、数据库连接等信息

server:  
port: 9000  # 后期该微服务多实例，9000（10个以内）  
Spring:  
application:  
  name: demo-service-product  
datasource:  
  driver-class-name: com.mysql.jdbc.Driver  
  url: jdbc:mysql://localhost:3306/demodb?useUnicode=true&characterEncoding=utf8&serverTimezone=UTC  
  username: root  
  password: 123456

3) Mapper接口开发

package com.demo.product.mapper;  
​  
​  
import com.baomidou.mybatisplus.core.mapper.BaseMapper;  
import com.baomidou.mybatisplus.core.mapper.Mapper;  
import com.demo.common.pojo.Products;  
​  
/\*\*  
\* 现在使用的Mybatis-plus组件，该组件是Mybatis的加强版  
\* 能够与SpringBoot进行非常友好的整合，对比Mybatis框架只有使用便捷的改变  
\* 没有具体功能的改变  
\* 具体使用：让具体的Mapper接口继承BaseMapper即可  
\*/  
public interface ProductMapper extends BaseMapper<Products> {  
​  
}  
​

4) serive层开发

package com.demo.product.service.impl;  
​  
import com.demo.common.pojo.Products;  
import com.demo.product.mapper.ProductMapper;  
import com.demo.product.service.ProductService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
​  
@Service  
public class ProductServiceImpl implements ProductService {  
​  
   @Autowired  
   private ProductMapper productMapper;  
/\*\*  
\* 根据商品ID查询商品对象  
\*/  
   @Override  
   public Products findById(Integer productId) {  
       return productMapper.selectById(productId);  
  }  
}  
​

5) controller层开发

package com.demo.product.controller;  
​  
import com.demo.common.pojo.Products;  
import com.demo.product.service.ProductService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
​  
@RestController  
@RequestMapping("/product")  
public class ProductController {  
​  
   @Autowired  
   private ProductService productService;  
​  
   @RequestMapping("/query/{id}")  
   public Products query(@PathVariable Integer id){  
       return productService.findById(id);  
  }  
​  
}  
​

6) 启动类

package com.demo.product;  
​  
import org.mybatis.spring.annotation.MapperScan;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
​  
@SpringBootApplication  
@MapperScan("com.demo.product.mapper")  
public class ProductApplication {  
   public static void main(String[] args) {  
       SpringApplication.run(ProductApplication.class,args);  
  }  
}

## 1.4 页面静态化微服务

1） 在pom文件中，引入公共组件依赖

<?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>demo-parent</artifactId>  
       <groupId>com.demo</groupId>  
       <version>1.0-SNAPSHOT</version>  
   </parent>  
   <modelVersion>4.0.0</modelVersion>  
​  
   <artifactId>demo-service-page</artifactId>  
​  
   <dependencies>  
       <dependency>  
           <groupId>com.demo</groupId>  
           <artifactId>demo-service-common</artifactId>  
           <version>1.0-SNAPSHOT</version>  
       </dependency>  
   </dependencies>  
​  
</project>

2）在yml文件中配置端口、应用名、数据库连接等信息

server:  
port: 9100   # 后期该微服务多实例，端口从9100递增（10个以内）  
Spring:  
application:  
  name: demo-service-page  
datasource:  
  driver-class-name: com.mysql.jdbc.Driver  
  url: jdbc:mysql://localhost:3306/demodb?useUnicode=true&characterEncoding=utf8&serverTimezone=UTC  
  username: root  
  password: 123456

3) 编写PageController，在PageController中调用商品微服务对应的URL

package com.demo.page.controller;  
​  
import com.demo.common.pojo.Products;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
import org.springframework.web.client.RestTemplate;  
​  
@RestController  
@RequestMapping("/page")  
public class PageController {  
​  
   @Autowired  
   private RestTemplate restTemplate;  
​  
   @GetMapping("/getData/{id}")  
   public Products findDataById(@PathVariable Integer id){  
       Products products = restTemplate.getForObject("http://localhost:9000/product/query/"+id, Products.class);  
       System.out.println("从demo-service-product获得product对象:"+products);  
       return products;  
  }  
​  
}

4) 编写启动类，注入RestTemplate

package com.demo.page;  
​  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.annotation.Bean;  
import org.springframework.web.client.RestTemplate;  
​  
​  
@SpringBootApplication  
public class PageApplication {  
​  
   public static void main(String[] args) {  
       SpringApplication.run(PageApplication.class,args);  
  }  
​  
   @Bean  
   public RestTemplate restTemplate(){  
       return new RestTemplate();  
  }  
​  
}

# 案例2：搭建单例Eureka Server服务注册中心，改造案例工程

1、搭建Eureka Server服务 demo-cloud-eureka

demo-parent中引入Spring Cloud 依赖

Spring Cloud 是一个综合的项目，下面有很多子项目，比如eureka子项目

   <dependencyManagement>  
       <dependencies>  
           <dependency>  
               <groupId>org.springframework.cloud</groupId>  
               <artifactId>spring-cloud-dependencies</artifactId>  
               <version>Greenwich.RELEASE</version>  
               <type>pom</type>  
               <scope>import</scope>  
           </dependency>  
       </dependencies>  
   </dependencyManagement>

2、demo-cloud-eureka工程pom.xml中引入依赖

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

注意：在父工程的pom文件中手动引入jaxb的jar，因为Jdk9之后默认没有加载该模块，Eureka Server使用到，所以需要手动导入，否则Eureka Server服务无法启动

父工程：

<!--引入Jaxb，开始-->  
<dependency>  
  <groupId>com.sun.xml.bind</groupId>  
  <artifactId>jaxb-core</artifactId>  
  <version>2.2.11</version>  
</dependency>  
<dependency>  
  <groupId>javax.xml.bind</groupId>  
  <artifactId>jaxb-api</artifactId>  
</dependency>  
<dependency>  
  <groupId>com.sun.xml.bind</groupId>  
  <artifactId>jaxb-impl</artifactId>  
  <version>2.2.11</version>  
</dependency>  
<dependency>  
  <groupId>org.glassfish.jaxb</groupId>  
  <artifactId>jaxb-runtime</artifactId>  
  <version>2.2.10-b140310.1920</version>  
</dependency>  
<dependency>  
  <groupId>javax.activation</groupId>  
  <artifactId>activation</artifactId>  
  <version>1.1.1</version>  
</dependency>  
<!--引入Jaxb，结束-->

3、在yml文件中配置Eureka server服务端口，服务名等信息

 #Eureka server服务端口  
server:  
port: 9200  
spring:  
application:  
  name: demo-cloud-eureka-server # 应用名称，会在Eureka中作为服务的id标识（serviceId）  
eureka:  
instance:  
  hostname: localhost  
client:  
  service-url: # 客户端与EurekaServer交互的地址，如果是集群，也需要写其它Server的地址  
    defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/  
  register-with-eureka: false  # 自己就是服务不需要注册自己  
  fetch-registry: false #自己就是服务不需要从Eureka Server获取服务信息,默认为true，置为false

4、编写启动类，声明当前服务为Eureka注册中心

package com.demo.eureka;  
​  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  
​  
@SpringBootApplication  
//// 声明本项目是一个Eureka服务  
@EnableEurekaServer  
public class EurekaApplication {  
   public static void main(String[] args) {  
       SpringApplication.run(EurekaApplication.class,args);  
  }  
}

5、启动后访问<http://127.0.0.1:9200>

6、商品微服务和页面静态化微服务注册到Eureka

pom文件中添加Eureka Client依赖

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

yml配置Eureka服务端信息

eureka:  
client:  
  serviceUrl: # eureka server的路径  
    defaultZone: http://localhost:9200/eureka/   
instance:  
   #使用ip注册，否则会使用主机名注册了（此处考虑到对老版本的兼容，新版本经过实验都是ip）  
  prefer-ip-address: true  
   #自定义实例显示格式，加上版本号，便于多版本管理，注意是ip-address，早期版本是ipAddress  
  instance-id: ${spring.cloud.client.ip-address}:${spring.application.name}:${server.port}:@project.version@

修改启动类

@SpringBootApplication  
@EnableDiscoveryClient //@EnableEurekaClient  
public class PageApplication {  
​  
   public static void main(String[] args) {  
       SpringApplication.run(PageApplication.class,args);  
  }  
​  
   @Bean  
   public RestTemplate restTemplate(){  
       return new RestTemplate();  
  }  
​  
}

# 案例3：搭建Eureka Server 高可用集群，改造案例工程

由于是在个人计算机中进行测试很难模拟多主机的情况，Eureka配置server集群时需要执行host地址。 所以需要修改个人电脑中host地址:

win10操作系统下：C:\Windows\System32\drivers\etc\host

127.0.0.1 demoCloudEurekaServerA  
127.0.0.1 demoCloudEurekaServerB

将demo-cloud-eureka复制一份为demo-cloud-eureka9201

1、修改 demo-cloud-eureka-server 工程中的yml配置文件

9200:

 #Eureka server服务端口  
server:  
port: 9200  
spring:  
application:  
  name: demo-cloud-eureka-server # 应用名称，会在Eureka中作为服务的id标识（serviceId）  
eureka:  
instance:  
  hostname: demoCloudEurekaServerA  
client:  
  register-with-eureka: true  
  fetch-registry: true  
  serviceUrl:  
    defaultZone: http://demoCloudEurekaServerB:9201/eureka

9201:

 #Eureka server服务端口  
server:  
port: 9201  
spring:  
application:  
  name: demo-cloud-eureka-server # 应用名称，会在Eureka中作为服务的id标识（serviceId）  
eureka:  
instance:  
  hostname: demoCloudEurekaServerB  
client:  
  register-with-eureka: true  
  fetch-registry: true  
  serviceUrl:  
    defaultZone: http://demoCloudEurekaServerA:9200/eureka

商品微服务：

server:  
port: 9000  # 后期该微服务多实例，9000（10个以内）  
Spring:  
application:  
  name: demo-service-product  
datasource:  
  driver-class-name: com.mysql.jdbc.Driver  
  url: jdbc:mysql://localhost:3306/demodb?useUnicode=true&characterEncoding=utf8&serverTimezone=UTC  
  username: root  
  password: 123456  
​  
eureka:  
client:  
  serviceUrl: # eureka server的路径  
    defaultZone: http://democloudeurekaservera:9200/eureka/,http://democloudeurekaserverb:9201/eureka/ #把 eureka 集群中的所有 url 都填写了进来，也可以只写一台，因为各个 eureka server 可以同步注册表  
instance:  
   #使用ip注册，否则会使用主机名注册了（此处考虑到对老版本的兼容，新版本经过实验都是ip）  
  prefer-ip-address: true  
   #自定义实例显示格式，加上版本号，便于多版本管理，注意是ip-address，早期版本是ipAddress  
  instance-id: ${spring.cloud.client.ip-address}:${spring.application.name}:${server.port}:@project.version@

页面静态化微服务：

server:  
port: 9100   # 后期该微服务多实例，端口从9100递增（10个以内）  
Spring:  
application:  
  name: demo-service-page  
datasource:  
  driver-class-name: com.mysql.jdbc.Driver  
  url: jdbc:mysql://localhost:3306/demodb?useUnicode=true&characterEncoding=utf8&serverTimezone=UTC  
  username: root  
  password: 123456  
​  
eureka:  
client:  
  serviceUrl: # eureka server的路径  
    defaultZone: http://democloudeurekaservera:9200/eureka/,http://democloudeurekaserverb:9201/eureka/ #把 eureka 集群中的所有 url 都填写了进来，也可以只写一台，因为各个 eureka server 可以同步注册表  
instance:  
   #使用ip注册，否则会使用主机名注册了（此处考虑到对老版本的兼容，新版本经过实验都是ip）  
  prefer-ip-address: true  
   #自定义实例显示格式，加上版本号，便于多版本管理，注意是ip-address，早期版本是ipAddress  
  instance-id: ${spring.cloud.client.ip-address}:${spring.application.name}:${server.port}:@project.version@

服务消费者调用服务提供者

改造页面静态化微服务：之前是直接通过RestTemplate写死URL进行调用，现在通过Eureka方式进行调用。

@RestController  
@RequestMapping("/page")  
public class PageController {  
   @Autowired  
   private RestTemplate restTemplate;  
   @Autowired  
   private DiscoveryClient discoveryClient;  
   @RequestMapping("/getData/{id}")  
   public Products findDataById(@PathVariable Integer id){  
       //1.获得Eureka中注册的demo-service-product实例集合  
       List<ServiceInstance> instances = discoveryClient.getInstances("demo-service-product");  
       //2.获得实例集合中的第一个  
       ServiceInstance instance = instances.get(0);  
       //3.根据实例信息拼接IP地址  
       String host = instance.getHost();  
       int port = instance.getPort();  
       String url = "http://"+host+":"+port+"/product/query/"+id;  
       //4.调用  
       Products products = restTemplate.getForObject(url, Products.class);  
       System.out.println("从demo-service-product获得product对象:"+products);  
       return products;  
  }  
}

# 案例4：演示自定义元数据的定义与获取

instance:  
   #使用ip注册，否则会使用主机名注册了（此处考虑到对老版本的兼容，新版本经过实验都是ip）  
  prefer-ip-address: true  
   #自定义实例显示格式，加上版本号，便于多版本管理，注意是ip-address，早期版本是ipAddress  
  instance-id: ${spring.cloud.client.ip-address}:${spring.application.name}:${server.port}:@project.version@  
  metadata-map:  
    ip: 192.168.200.128  
    port: 10000  
    user: demo  
    pwd: 123456

我们可以在程序中可以使用DiscoveryClient 获取指定微服务的所有元数据信息，在任何想注册中心注册的微服务都可以获取。

package com.demo.page.controller;  
​  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.cloud.client.ServiceInstance;  
import org.springframework.cloud.client.discovery.DiscoveryClient;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
​  
import java.util.List;  
import java.util.Map;  
import java.util.Set;  
​  
@RestController  
@RequestMapping("/metadata")  
public class MetadataController {  
​  
   @Autowired  
   private DiscoveryClient discoveryClient;  
​  
   @RequestMapping("show")  
   public String showMetadata(){  
       String result = "";  
       List<ServiceInstance> instances = discoveryClient.getInstances("demo-service-page");  
       for (ServiceInstance instance:instances) {  
           //获取服务元数据  
           Map<String, String> metadata = instance.getMetadata();  
           Set<Map.Entry<String, String>> entries = metadata.entrySet();  
           for (Map.Entry<String,String> entry : entries){  
               String key = entry.getKey();  
               String value = entry.getValue();  
               result+="key:"+key+",value:"+value;  
          }  
      }  
       return result;  
  }  
​  
}

debug模式下查看元数据