# SpringCloud

## 微服务架构

常见的微服务框架有Spring Cloud，**Dubbo**(2012停更，2017年恢复更，18年2月**Dubbo**开源送给Appache组织)

解决**SOA**各个子系统之间的相互调用和管理。

**SpringCloud Alibaba**

## Spring Cloud简介

### Spring Cloud 架构

计算机生成了可选文字:
自妈州洲曰d
乙
公必乙，
仑必亡，
公必亡j
，叫闷昨
乙
分份尔比
舀门
伪幽网勺
乙
白心乡巷
碍盔。
乙
七侧留匆
乙

物联网（ IoT ，Internet of things ）即“万物相连的互联网”。

APIGateway作为一个系统的后端总入口，承载着所有服务的组合路由转换等工作，一般也会把安全，限流，缓存，日志，监控，重试，熔断等放到 API 网关来做

Breaker dashboard 断路器仪表板，是熔断监控页面

Config deshboard 配置仪表版，在Spring Cloud Alibaba中，已经把Config deshboard 和ServiceRegistry整合在一起。

Distributed Tracing分布式跟踪 ，是一个关键的基础功能,通过分布式追踪技术,我们可以深入分析一次请求调用所执行的路径、性能消耗,帮助定位性能瓶颈点。

Message Brokers 消息中间件。Rabbit MQ Kafka

### SpringCloud网站

<https://cloud.spring.io/spring-cloud-static/Hoxton.SR5/reference/htmlsingle/>

英文困难的同学，也不耽误学习的

<https://www.bookstack.cn/read/spring-cloud-docs/docs-index.md>

### 组件概述

计算机生成了可选文字:
摘七。月…关止
子
月，命开发
l
l
匹口

* Feign是一个声明式的web服务客户端，让编写web服务客户端变得非常容易，只需创建一个接口并在接口上添加注解即可

## 关于SpringBoot和SpringCloud版本

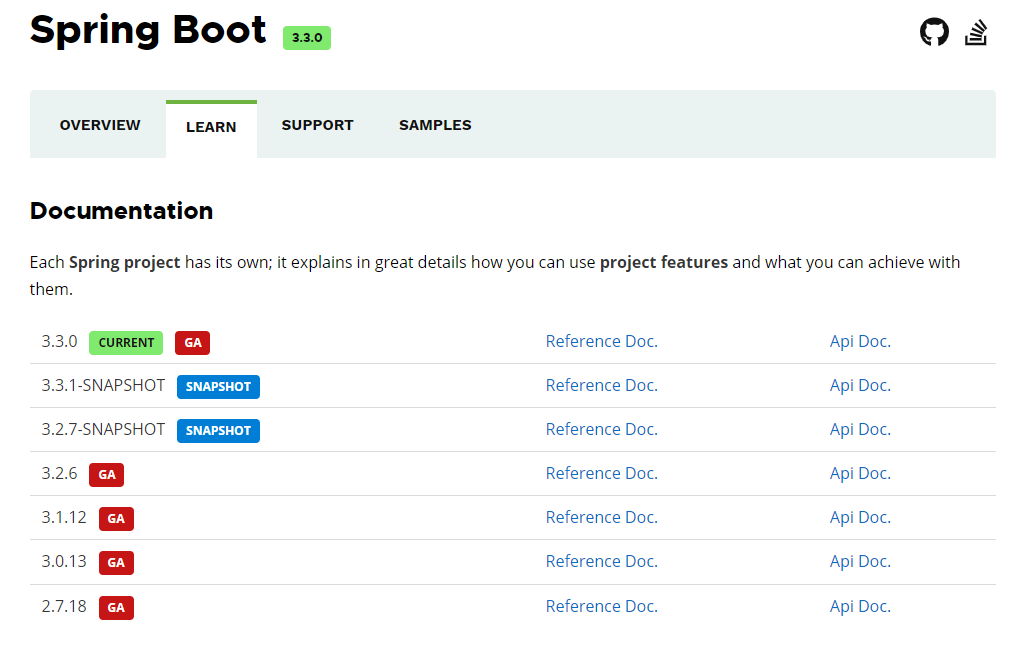
### Boot版本

**git源码地址：**

<https://github.com/spring-projects/spring-boot/releases/>

**springboot(截至2024.6.4)**

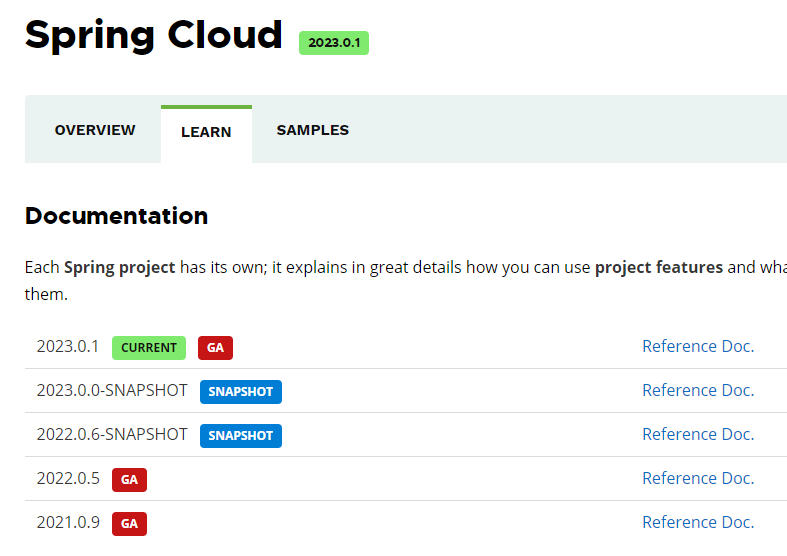
https://spring.io/projects/spring-boot#learn



### SpringCloud版本

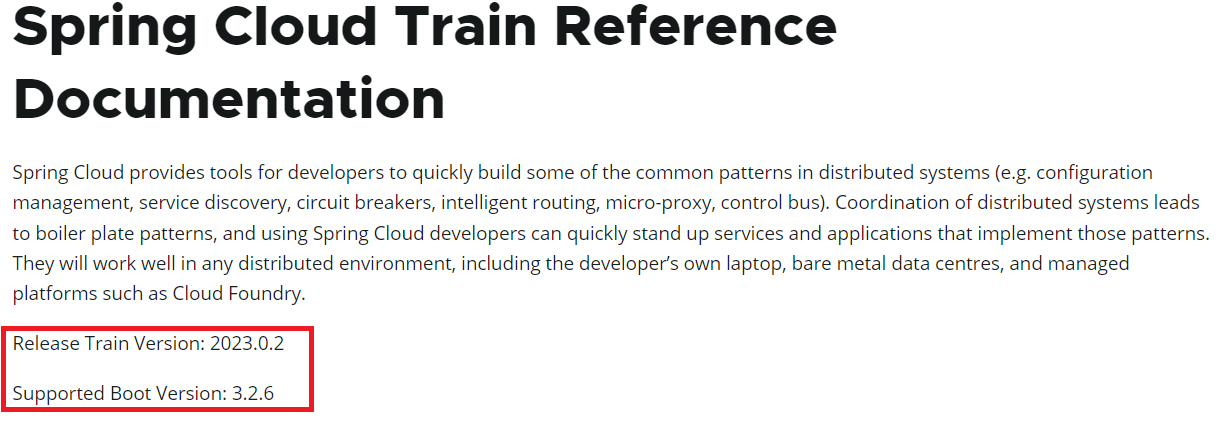
* git源码地址： <https://github.com/spring-projects/spring-cloud/wiki>
* 官网： <https://spring.io/projects/spring-cloud>

官网看SpringCloud(截至2024.6.4)



### SpringCloud和Springboot之间的依赖关系

https://docs.spring.io/spring-cloud-release/reference/index.html



详细的版本对应查看方法

<https://start.spring.io/actuator/info>

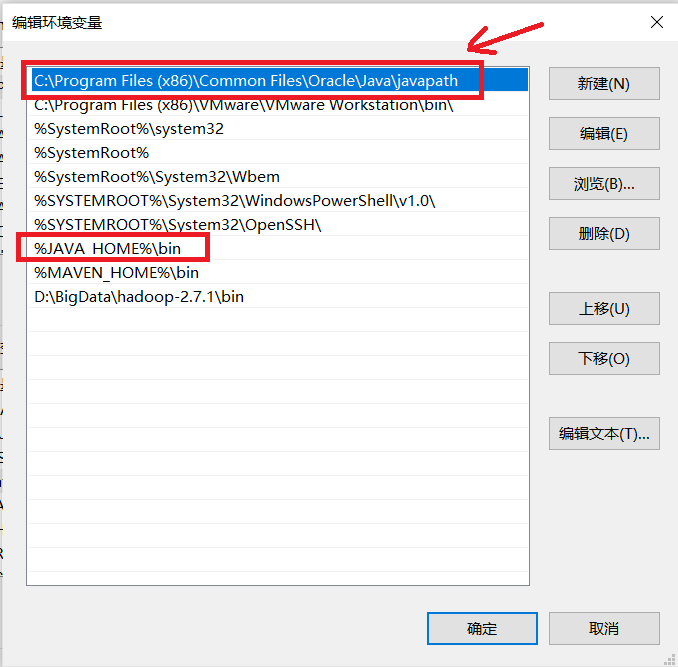
<https://cloud.spring.io/spring-cloud-static/Hoxton.SR1/reference/htmlsingle/>

计算机生成了可选文字:
SpringCloud
SpringCIoudprovidestoolsf0rdeveIoperstoquiCkIybuildsomeoftheCommonpatternsindistributedsystems(e.g.
Configurationmanagement,serviCedisCovery,CirCuitbreakers,intelligentrouting,miCro一proxy,Controlbus).Coordinationof
d1stributedSystemSIeadstoboilerplatepatterns,andusingSpringClouddevelopersCanquiCklystandupServiCeSand
applicationsthatimPIementthosepatterns.TheywiIIworkweIIinanydistributedenvironment,includingthedeveloper'sown
Iaptop,baremetaldataCentres,andmanagedplatformssuChasCloudFoundry.
ReleaseTrainVersion:HOXton.SRI
SupportedBootVersion:2.2.2.RELEASE

## SpringCloud入门案例-创建父工程

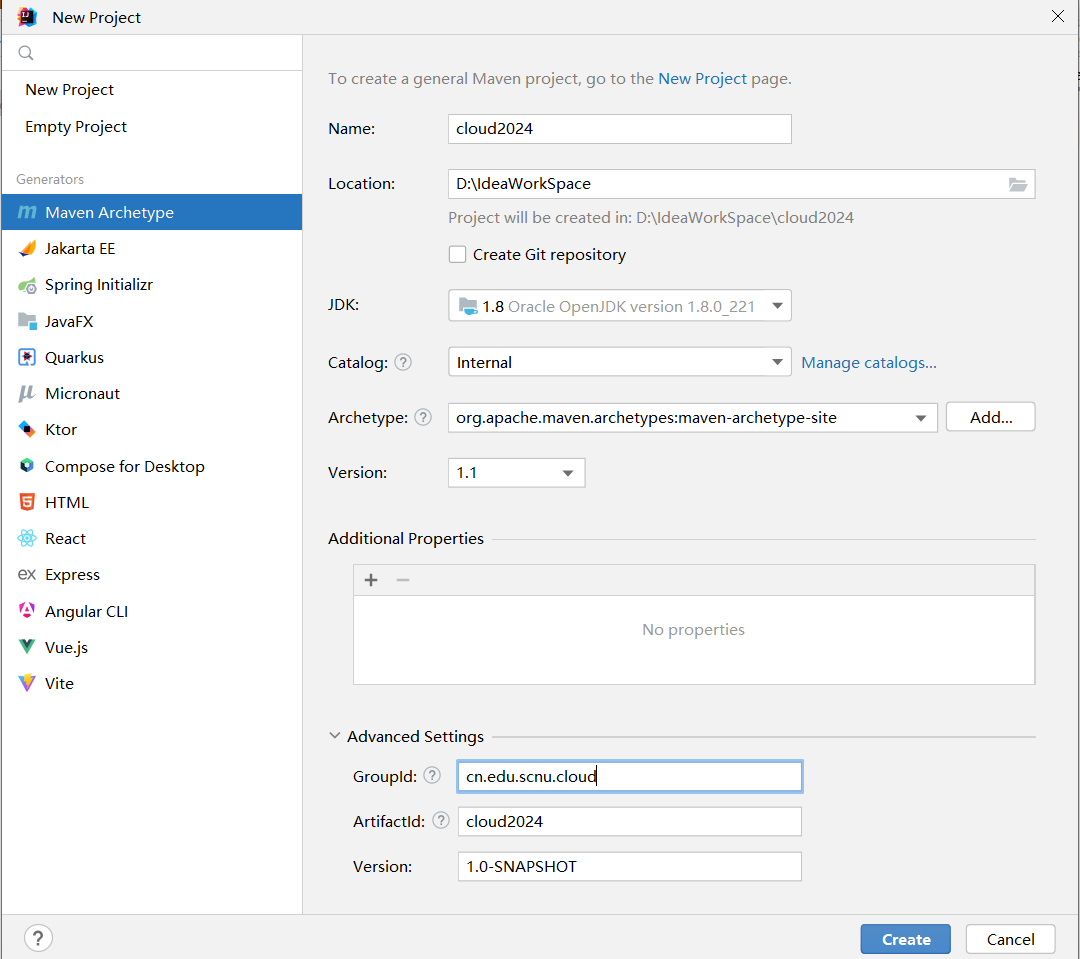
### SpringCloud（授课选择版本）

* + Spring Cloud
    - Hoxton.SR1
  + Spring Boot2
    - 2.2.2.RELEASE
  + Spring Cloud Alibaba
    - 2.1.0.RELEASE
  + mysql
    - 5.7及以上
  + JDK1.8
  + IDEA 2023.3.6



### 创建父级工程

#### New Project



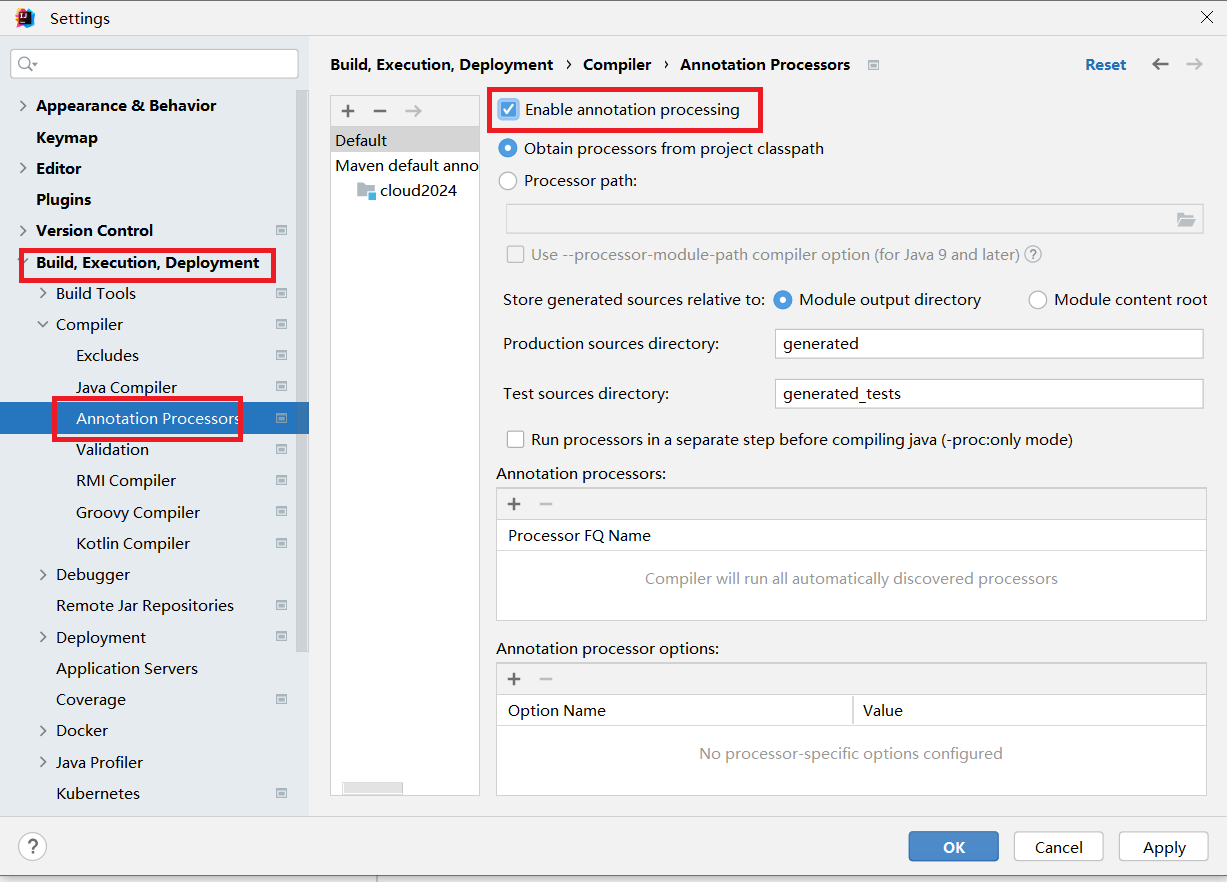
#### Maven选版本



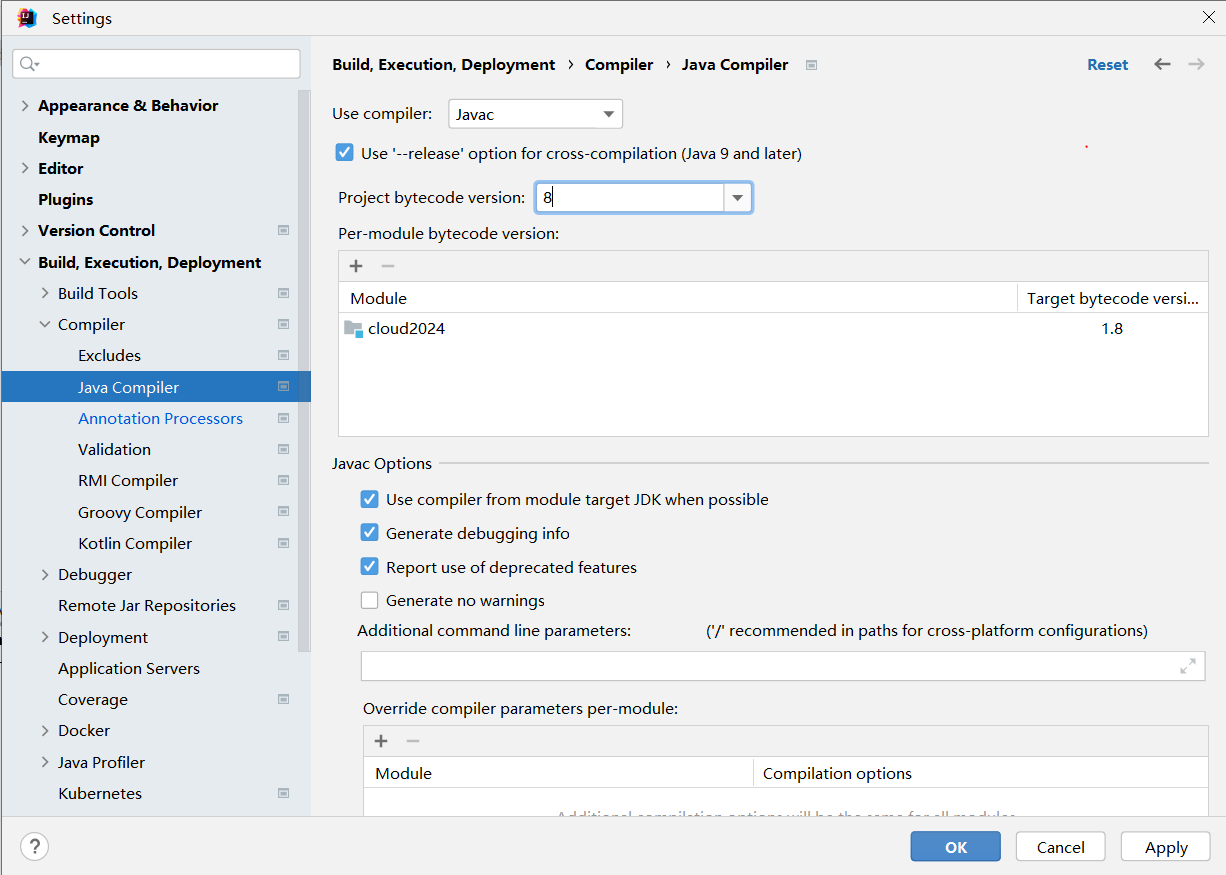
#### 字符编码

计算机生成了可选文字:
口setting,
匡
KeymaP
l"Ed'torl
X
〔ditor卜rileEncodings
嗜Forcurrentproject
Reset
》General
＋一／
path▲
Font
》Co!orScheme
Encoding
》
CodeStyle
Inspections
FileandCodeTemplates
FileEncodings
Encoding,arenotcon6gured
咱
一而丽雨示一
FIIeType,
AndroidLayoutEditor
>Copyright
InlayHints
AndroidDataBinding
Duplicates
Emm鱿
GUIDesigner
lmages
Intentions
>LanguageInjections
Spelling
TeX'tMateBundles
T000
TochangeencodingIntelliJIDEAusesforafile,adirectory,ortheentireproject,additspethifneces幼ryandthen
凭lectencQdingfromtheencoding11眺．Built一infileencoding(e.g.JSP.HTMLorXML)overridesencodingyou
specifyhere.Ifnotspecified,filesanddirectoriesinheritencodingSettingsfromthe闪rentdirectoryorfromthe
ProjectEncoding
Pro叶rtiesFilese.pro叶rties)
Defaultencodingforpropertie、6ies:UTF一8一口Transparentnative一to一a、ciiconversion
BOMfDFne侧丫UTF一8files
CreateUTF一8石les:withNOBOM
旧EAwillNOTaddUTF一8BOMtoeve叮created石leinUTF一8encoding
O
Cancel
…Apply…

#### 注解生效激活

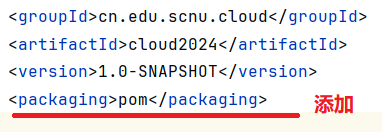


#### java编译版本



### 父工程POM

在父工程的pom文件添加packaging类型为pom



*<?*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.edu.scnu.cloud</groupId>

<artifactId>cloud2024</artifactId>

<version>1.0-SNAPSHOT</version>

**<packaging>pom</packaging>**

*<!-- 统一管理jar包版本 -->*

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

<junit.version>4.12</junit.version>

<log4j.version>1.2.17</log4j.version>

<lombok.version>1.16.18</lombok.version>

<mysql.version>5.1.47</mysql.version>

<druid.version>1.1.16</druid.version>

<mybatis.spring.boot.version>1.3.0</mybatis.spring.boot.version>

</properties>

*<!-- 子模块继承之后，提供作用：锁定版本+子modlue不用写groupId和version -->*

<dependencyManagement>

<dependencies>

*<!--spring boot 2.2.2-->*

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-dependencies</artifactId>

<version>2.2.2.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

*<!--spring cloud Hoxton.SR1-->*

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Hoxton.SR1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

*<!--spring cloud alibaba 2.1.0.RELEASE-->*

<dependency>

<groupId>com.alibaba.cloud</groupId>

<artifactId>spring-cloud-alibaba-dependencies</artifactId>

<version>2.1.0.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>${mysql.version}</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>${druid.version}</version>

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>${mybatis.spring.boot.version}</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>${log4j.version}</version>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>${lombok.version}</version>

<optional>true</optional>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<fork>true</fork>

<addResources>true</addResources>

</configuration>

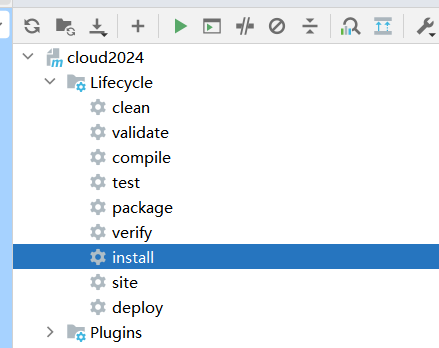
</plugin>

</plugins>

</build>

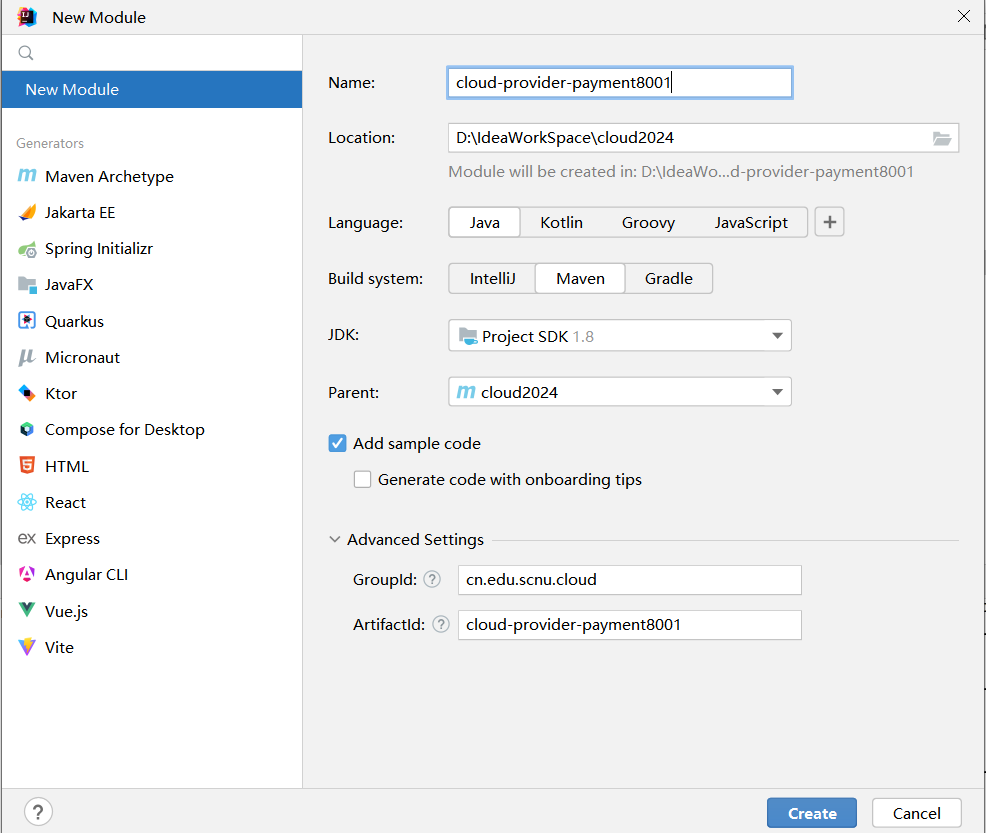
</project>

### 父工程创建完成执行mvn:install

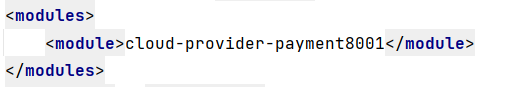


## 入门案例-创建微服务-【服务提供者】

### 建cloud-provider-payment8001



创建完成后回到父工程查看pom文件变化，可以看到增加了聚合模块



### 改POM文件

*<?*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>

<parent>

<groupId>cn.edu.scnu.cloud</groupId>

<artifactId>cloud2024</artifactId>

<version>1.0-SNAPSHOT</version>

</parent>

<artifactId>cloud-provider-payment8001</artifactId>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid-spring-boot-starter</artifactId>

<version>1.1.10</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

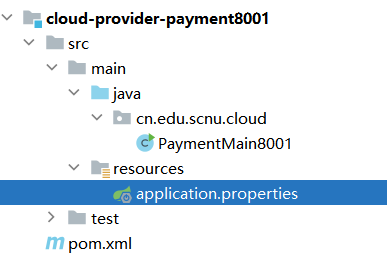
</dependency>

</dependencies>

</project>

### 编写配置文件

在resource下创建application.properties



server.port = 8001

spring.application.name=cloud-payment-service

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

spring.datasource.driverClassName=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql:///cloud2024?useUnicode=true&characterEncoding=utf-8&useSSL=false

spring.datasource.username=root

spring.datasource.password=12345678

*#mybatis config*

mybatis.typeAliasesPackage=cn.edu.scnu.cloud.entity

mybatis.mapperLocations=classpath:mapper/\*.xml

### 启动类

修改Main类的文件名和类名，修改为PaymentMain8001 ，启动类如下

|  |
| --- |
| package cn.edu.scnu.cloud;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class PaymentMain8001 {  public static void main(String[] args) {  SpringApplication.*run*(PaymentMain8001.class,args);  }  } |

### 业务类

#### 建表SQL

|  |
| --- |
| CREATE DATABASE IF NOT EXISTS cloud2024 DEFAULT CHARACTER SET utf8 ;  USE cloud2024 ;  DROP TABLE IF EXISTS payment ;  CREATE TABLE payment (  id BIGINT (20) NOT NULL AUTO\_INCREMENT COMMENT 'ID',  SERIAL VARCHAR (300) DEFAULT NULL,  PRIMARY KEY (id)  ) ENGINE = INNODB AUTO\_INCREMENT = 33 DEFAULT CHARSET = utf8 ;  INSERT INTO payment (id, SERIAL) VALUES(31, 'a001'),(32, 'a002') ;  Select \* from payment; |

#### Entity

##### 主实体Payment

**package cn.edu.scnu.cloud.entity;**

**import lombok.AllArgsConstructor;**

**import lombok.Data;**

**import lombok.NoArgsConstructor;**

**import java.io.Serializable;**

**@Data**

**@AllArgsConstructor**

**@NoArgsConstructor**

**public class Payment implements Serializable {**

**private Long id;**

**private String serial;**

**}**

##### 实体类CommonResult（Json封装）

**package cn.edu.scnu.cloud.entity;**

**import lombok.AllArgsConstructor;**

**import lombok.Data;**

**import lombok.NoArgsConstructor;**

**import java.io.Serializable;**

**@Data**

**@AllArgsConstructor**

**@NoArgsConstructor**

**public class CommonResult <T> implements Serializable {**

**private Integer code;**

**private String message;**

**private T data;**

**public CommonResult(Integer code,String message){**

**this(code,message,null);*//如果这行报错，请安装lombok插件***

**}**

**}**

##### lombok

<https://www.projectlombok.org/>

@Data：提供getter/setter

@NoArgsConstructor, 无参构造器 @RequiredArgsConstructor @AllArgsConstructor 全参数构造器

@EqualsAndHashCode：提供equals和hashCode方法

@Getter/@Setter

@Slf4j 内置log对象，直接调用日志方法输出日志

#### Dao

##### 接口PaymentDao

**package** cn.edu.scnu.cloud.dao;

**import** cn.edu.scnu.cloud.entity.Payment;

**import** org.apache.ibatis.annotations.Mapper;

**import** org.apache.ibatis.annotations.Param;

**import** org.springframework.stereotype.Component;

@Component *//代替@Repository声明bean*

@Mapper *//mybatis提供的，等价：@MapperScan("cn.edu.scnu.cloud.dao")*

*//@Repository //spring提供的。在此，只是为了声明bean对象*

**public interface** PaymentDao {

**public int** create(Payment payment);

**public** Payment getPaymentById(@Param(**"id"**) Long id);

}

##### mybatis的映射文件

src\main\resources\mapper\PaymentMapper.xml

*<?***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.edu.scnu.cloud.dao.PaymentDao"**>

<**insert id="create" useGeneratedKeys="true" keyProperty="id"**>

insert into payment(serial) values(#{serial});

</**insert**>

<**resultMap id="BaseResultMap" type="cn.edu.scnu.cloud.entity.Payment"**>

<**id column="id" property="id" jdbcType="BIGINT"**></**id**>

<**result column="serial" property="serial" jdbcType="VARCHAR"**></**result**>

</**resultMap**>

<**select id="getPaymentById" parameterType="Long" resultMap="BaseResultMap"**>

select \* from payment where id=#{id}

</**select**>

</**mapper**>

#### Service

##### 接口PaymentService

**package** cn.edu.scnu.cloud.service;

**import** cn.edu.scnu.cloud.entity.Payment;

**public interface** PaymentService {

**public int** create(Payment payment); *//写*

**public** Payment getPaymentById(Long id); *//读取*

}

##### 实现类PaymentServiceImpl

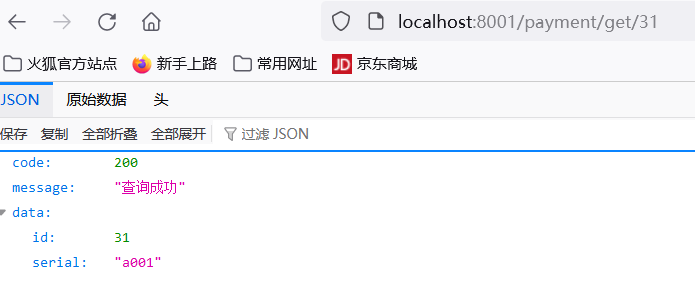
|  |
| --- |
| package cn.edu.scnu.cloud.service;  import cn.edu.scnu.cloud.dao.PaymentDao;  import cn.edu.scnu.cloud.entity.Payment;  import org.springframework.stereotype.Service;  import javax.annotation.Resource;  @Service  public class PaymentServiceImpl implements PaymentService{  @Resource  *//@Autowired*  private PaymentDao paymentDao;  public int create(Payment payment){  return paymentDao.create(payment);  }  public Payment getPaymentById( Long id){  return paymentDao.getPaymentById(id);  }  } |

#### Controller

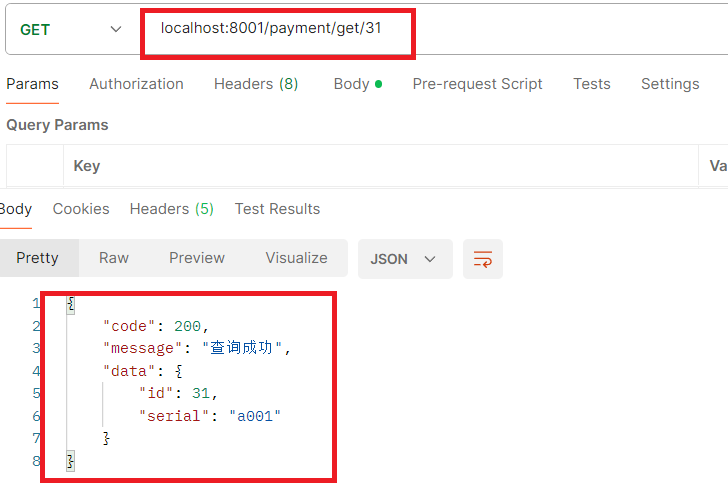
|  |
| --- |
| package cn.edu.scnu.cloud.controller;  import cn.edu.scnu.cloud.entity.CommonResult;  import cn.edu.scnu.cloud.entity.Payment;  import cn.edu.scnu.cloud.service.PaymentService;  import lombok.extern.slf4j.Slf4j;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.PostMapping;  import org.springframework.web.bind.annotation.RestController;  import javax.annotation.Resource;  @RestController  @Slf4j  public class PaymentController {  @Resource  private PaymentService paymentService;  @PostMapping(value = "/payment/create")  public CommonResult<Payment> create(Payment payment){ *//埋雷*  int result = paymentService.create(payment);  *log*.info("\*\*\*\*\*插入结果："+result);  if (result>0){ *//成功*  return new CommonResult(200,"插入数据库成功",result);  }else {  return new CommonResult(444,"插入数据库失败",null);  }  }  @GetMapping(value = "/payment/get/{id}")  public CommonResult<Payment> getPaymentById(@PathVariable("id") Long id){  Payment payment = paymentService.getPaymentById(id);  *log*.info("\*\*\*\*\*查询结果："+payment);  if (payment!=null){ *//说明有数据，能查询成功*  return new CommonResult(200,"查询成功",payment);  }else {  return new CommonResult(444,"没有对应记录，查询ID："+id,null);  }  }  } |

### 测试

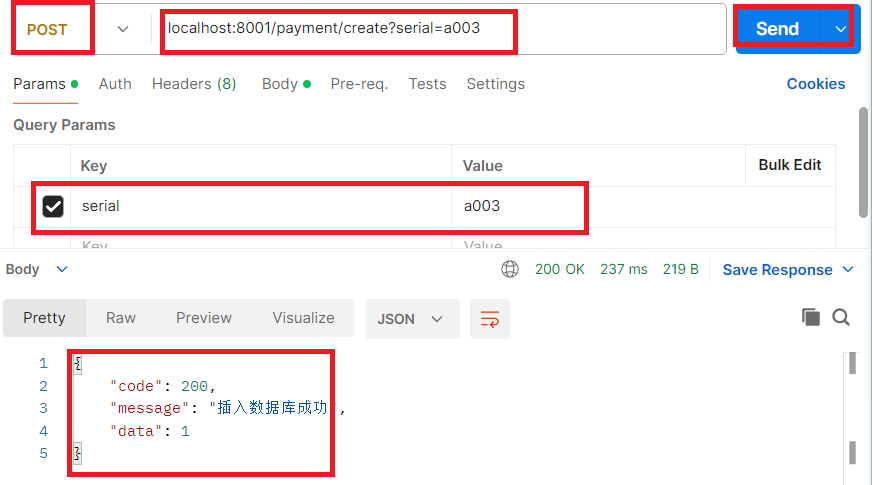
#### 浏览器测试get



#### postman测试get请求



#### postman测试post请求



### 开发步骤-小总结

1. 创建module
2. 修改POM
3. 编写配置文件
4. 启动类
5. 业务类

### 热部署Devtools

#### Adding devtools to your project

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <scope>runtime</scope>  <optional>true</optional>  </dependency> |

#### Adding plugin to your pom.xml

|  |
| --- |
| 下一段配置黏贴到父工程当中的pom里（已经添加）  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>  <fork>true</fork>  <addResources>true</addResources>  </configuration>  </plugin>  </plugins>  </build> |

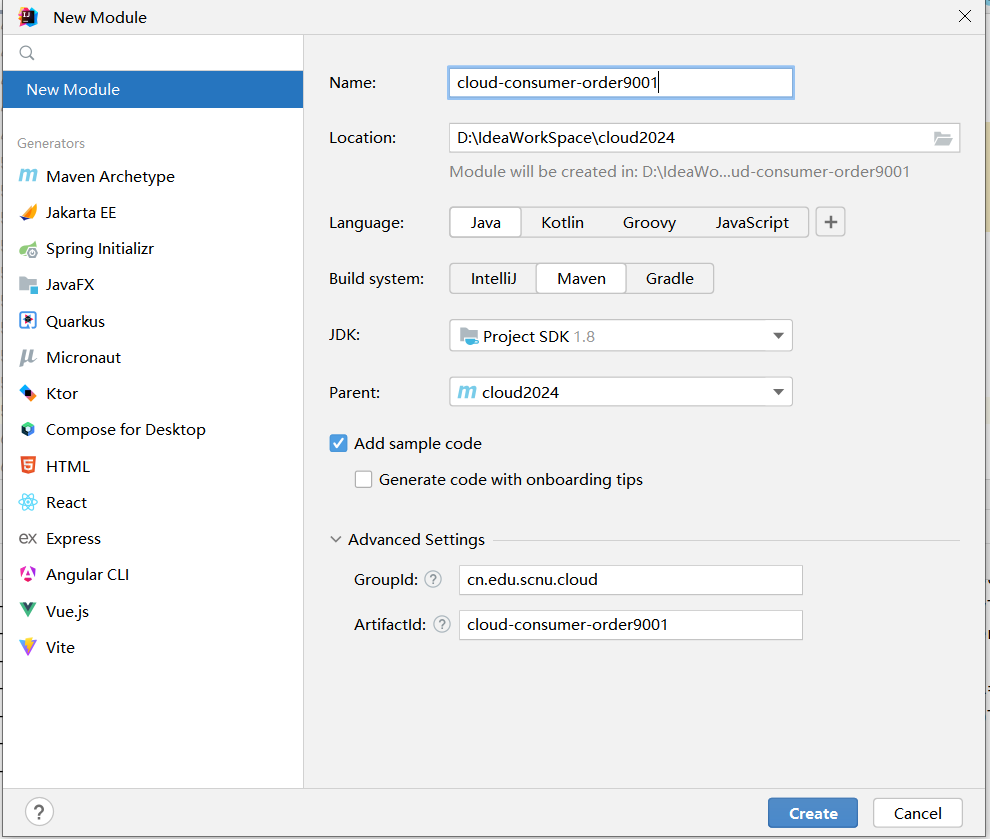
#### Enabling automatic build

计算机生成了可选文字:
口settings
X
APPearance&Behavior
KeymaP
》Editor
plugins
VersionContrO!
Build,EXecution,DePloyment
BuildTools
,uild．欧ecution.Deployment》compiler「cl。：日r·。ntproject
Resourcepatterns:!?"．〕ava:!?".fo饥；!?".cla33;!?*.groovy:!?".3cala;!?*.flex:!?"．枕；竺？".cl〕；
Use:toseparatepattern'and里tonegatea闪tternAcceptedwildcard':？一exactlyoneSymbol二食
一zeroormore卿mbols;I一闪thse闪rato‘产V一anynumberofdirectorie,；侧乞～,:
心户曰岔恕m）一re蛇ricttosourcerootswiththespecifiedname
口CIearoutputdirecto印onrebuild
口Addruntlmea:,ertion,fornotnull一annotatedmethod,andparameter,con石gureannotatlon：…
口Autom。ti。。llyshow6rsterrorineditor
Displaynoti6cationonbuildcompletion
Debugger
RemoteJarRepositorie,
。Deployment
ArquillianContainers
ApplicationServers
Cloud'
…口”u""p『ojectau,oma,iCa"y
｝回Compileindependentmodulesinparallel
回Rebuildmoduleondependen守change
Buildprocess卜eapsize(Mbytes):
S卜aredbulldproce、、VMoption、：
U,er一Iocalbuildproces、VMoption、（override、Sharedoptions):
(onlyworkswhilenotrunning/debugging)
(mayrequirelarger卜eapsize)
7oo

#### 重启IDEA

## 微服务架构编码构建-Rest微服务-【服务消费者】

### 建cloud-consumer-order9001



### POM

*<?*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>

<parent>

<groupId>cn.edu.scnu.cloud</groupId>

<artifactId>cloud2024</artifactId>

<version>1.0-SNAPSHOT</version>

</parent>

<artifactId>cloud-consumer-order9001</artifactId>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

</project>

### application.properties

server.port = 9001

spring.application.name=cloud-consumer-order9001

### 主启动

package cn.edu.scnu.cloud;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class OrderMain9001 {

public static void main(String[] args) {

SpringApplication.*run*(OrderMain9001.class,args);

}

}

### 业务类

#### 创建entity

(将cloud-provider-payment8001工程下的entity包下的两个实体类复制过来)

#### RestTemplate

* RestTemplate么介绍

RestTemplate提供了多种便捷访问远程Http服务的方法，是一种简单便捷的访问Restful服务模板类，是Spring 提供的用于访问Rest服务的客户端模板工具集

* 官网及使用

官网地址： <https://docs.spring.io/spring-framework/docs/5.2.2.RELEASE/javadoc-api/org/springframework/web/client/RestTemplate.html>

使用RestTemplate访问Restful接口非常的简单粗暴无脑。（url，requestMap，ResponseBean.class）这三个参数分别代表REST请求地址、请求参数、Http响应转换被转换成的对象类型。

#### config配置类

ApplicationContextConfig

package cn.edu.scnu.cloud.config;

import org.springframework.boot.SpringBootConfiguration;

import org.springframework.context.annotation.Bean;

import org.springframework.web.client.RestTemplate;

@SpringBootConfiguration

public class ApplicationContextConfig {

@Bean

*//@LoadBalanced*

public RestTemplate getRestTemplate(){

return new RestTemplate();

}

}

#### 创建controller

package cn.edu.scnu.cloud.controller;

import cn.edu.scnu.cloud.entity.CommonResult;

import cn.edu.scnu.cloud.entity.Payment;

import lombok.extern.slf4j.Slf4j;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.client.RestTemplate;

import javax.annotation.Resource;

@RestController

@Slf4j

public class OrderController {

public static final String *PAYMENT\_URL* = "http://localhost:8001";

@Resource

private RestTemplate restTemplate;

@PostMapping("/consumer/payment/create")

public CommonResult<Payment> create(Payment payment){

return restTemplate.postForObject(*PAYMENT\_URL*+"/payment/create",payment, CommonResult.class); *//写操作*

}

@GetMapping("/consumer/payment/get/{id}")

public CommonResult<Payment> getPayment(@PathVariable("id") Long id){

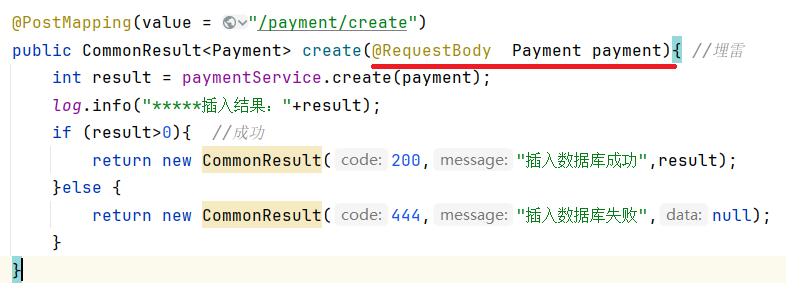
return restTemplate.getForObject(*PAYMENT\_URL*+"/payment/get/"+id,CommonResult.class);

}

}

### 测试

1. 先启动cloud-provider-payment8001
2. 再启动cloud-consumer-order9001
3. 因为RestTemplate远程调用，传的payment对象属于json格式数据，所以要加上@RequestBody注解，否则，虽然显示插入数据成功，但是数据没存入数据库。

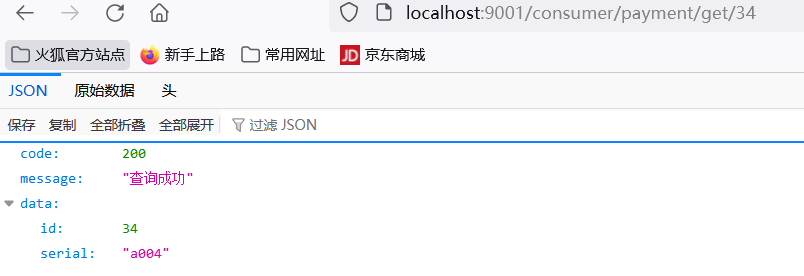


在Postman中输入json数据



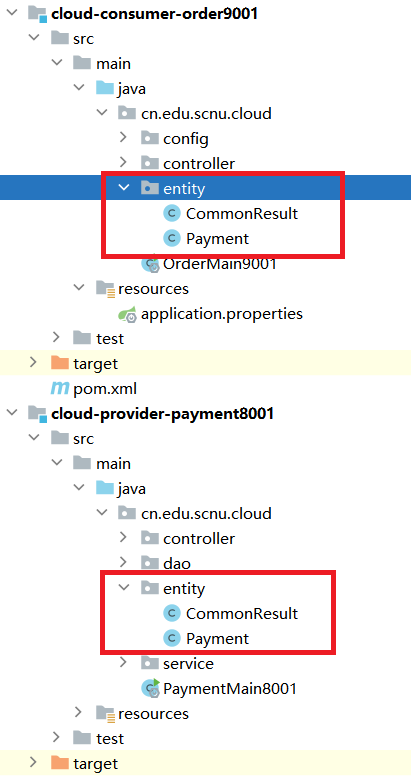
1. 查看数据

[http://localhost:9001/consumer/payment/get/34](http://localhost/consumer/payment/get/32)

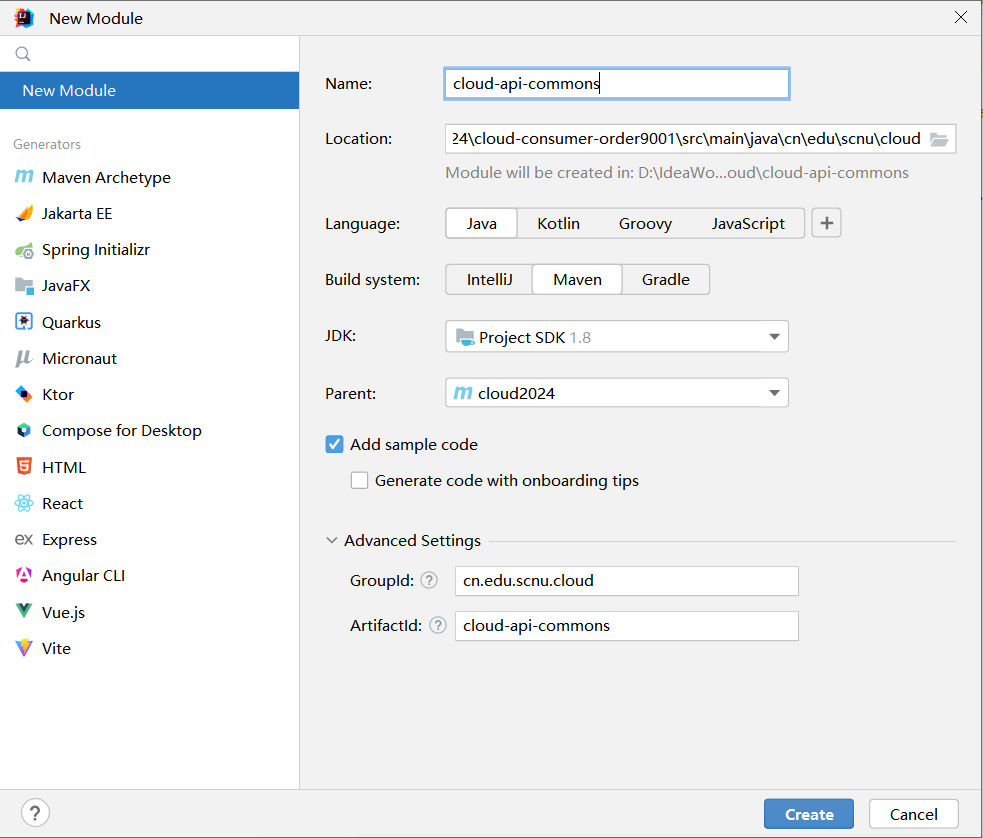


## 工程重构

### 观察问题



### 新建：cloud-api-commons



### POM

*<?*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>

<parent>

<groupId>cn.edu.scnu.cloud</groupId>

<artifactId>cloud2024</artifactId>

<version>1.0-SNAPSHOT</version>

</parent>

<artifactId>cloud-api-commons</artifactId>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>cn.hutool</groupId>

<artifactId>hutool-all</artifactId>

<version>5.1.0</version>

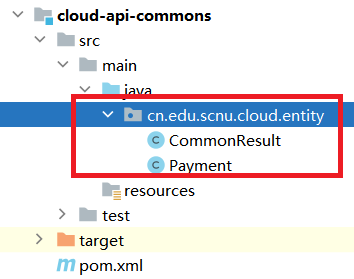
</dependency>

</dependencies>

</project>

### entity

复制 Payment实体和 CommonResult通用封装类 到cloud-api-commons

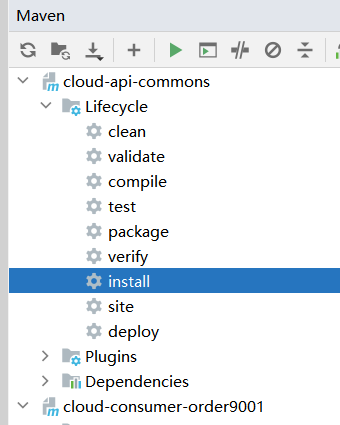


### 删除主方法

删除cloud-api-commons项目的主方法。

### maven命令clean install

在mavan窗口执行clean命令，然后执行install命令。



### 订单9001和支付8001分别改造

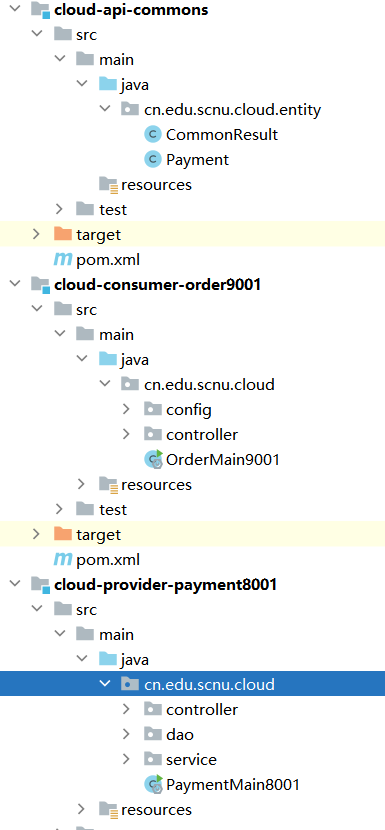
删除8001和9001各自的原有entity包以及包中的两个实体类。

在cloud-provider-payment8001和cloud-consumer-order9001项目中添加以下依赖，依赖于cloud-api-commons公共项目。

分别在 cloud-provider-payment8001和cloud-cunsumer-orde9001项目的pom文件添加下面依赖。

|  |
| --- |
| <dependency>  <groupId>cn.edu.scnu.cloud</groupId>  <artifactId>cloud-api-commons</artifactId>  <version>${project.version}</version>  </dependency> |

### 目前工程样图



### 测试

经过整合后的项目，运行结果跟前面一样

# Eureka服务注册与发现

## Eureka基础知识

### 什么是服务治理

在传统的RPC远程调用框架中，管理服务与服务之间依赖关系比较复杂、所以需要进行**服务治理**，通过服务治理管理服务与服务之间依赖关联，以实现服务调用，负载均衡、容错等，实现服务发现与注册。

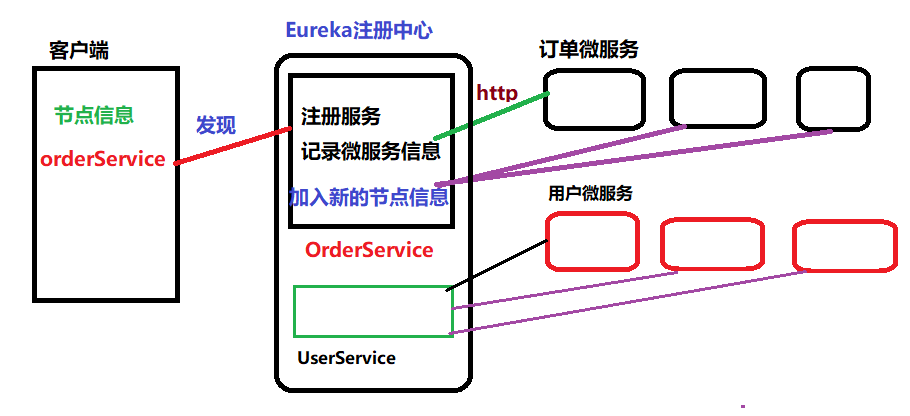
SpringCloud封装了Netflix公司开发的Eureka模块来实现服务治理。

Eureka是SpringCloud 的微服务框架的核心组件(管理单位)，是完成当前所有集群微服务的注册与发现的核心进程，可以动态的处理所有负载均衡访问的集群（serverList）的管理工作;

### 什么是服务注册

Eureka采用了CS的设计架构，Eureka Server作为**服务注册**功能的服务器，它是服务注册中心。

系统中的服务提供者，使用Eureka的客户端连接到Eureka Server，动态注册在Eureka 注册中心。



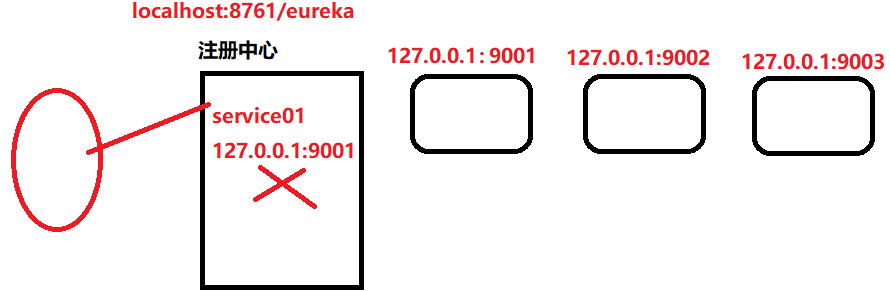
### 什么是服务发现

* **服务的发现**:系统中的服务消费者，具备访问Eureka 的能力,动态的获取某个一个服务提供者信息,从而发起访问调用;

### Eureka注册逻辑

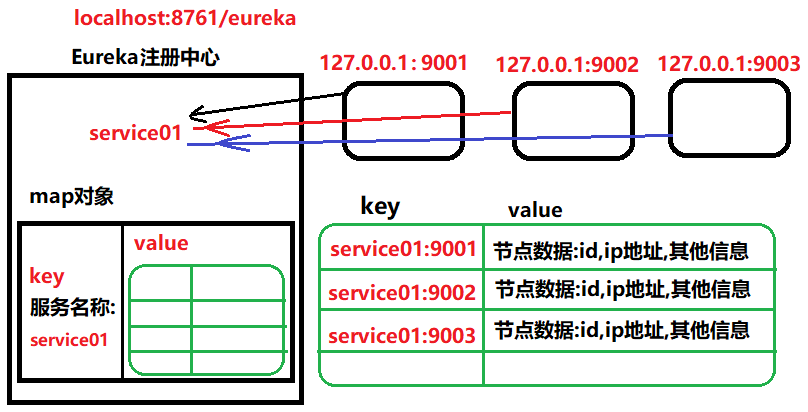
* **续约,和剔除**

启动的每一个eureka客户端进程.都会根据配置到Eureka 注册中心进行注册,注册之后, Eureka 客户端（服务）每30秒给注册中心发送一个心跳检测(续约) ,注册中心每60秒检测一次所有的服务续约,一旦发现服务超过90秒没有发送心跳检测的话,将会把服务对应的实例节点剔除;



* **Eureka注册信息结构**

每一个服务的提供者注册到服务端时,访问调用localhost:8761/eureka发送请求.并且携带一些信息,Eureka服务端将这些信息保管在一个双层map中。

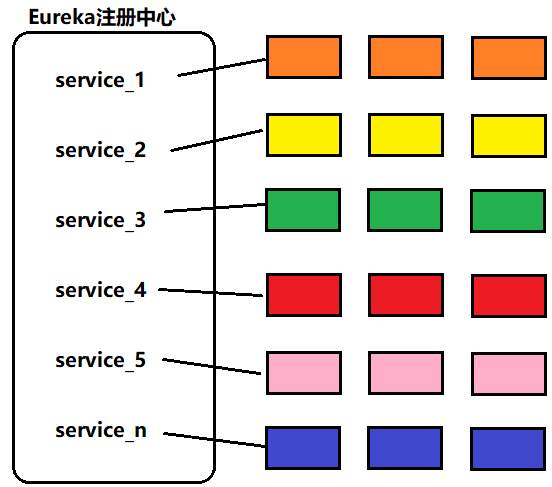


* **保护机制**

Eureka 注册中心.提供一种保护微服务集群注册服务信息的机制,防止由于网络波动导致大量的注册心跳检测发送未成功,造成的误剔除,给集群,给微服务带来的巨大伤害,所出现的一种机制--保护机制

当整个集群15%以上同一时间检测到超过90秒未发送心跳.将会开启保护机制,一旦开启,所有的服务注册信息,在eureka中心不会进行任何剔除;默认情况保护机制是开启的。

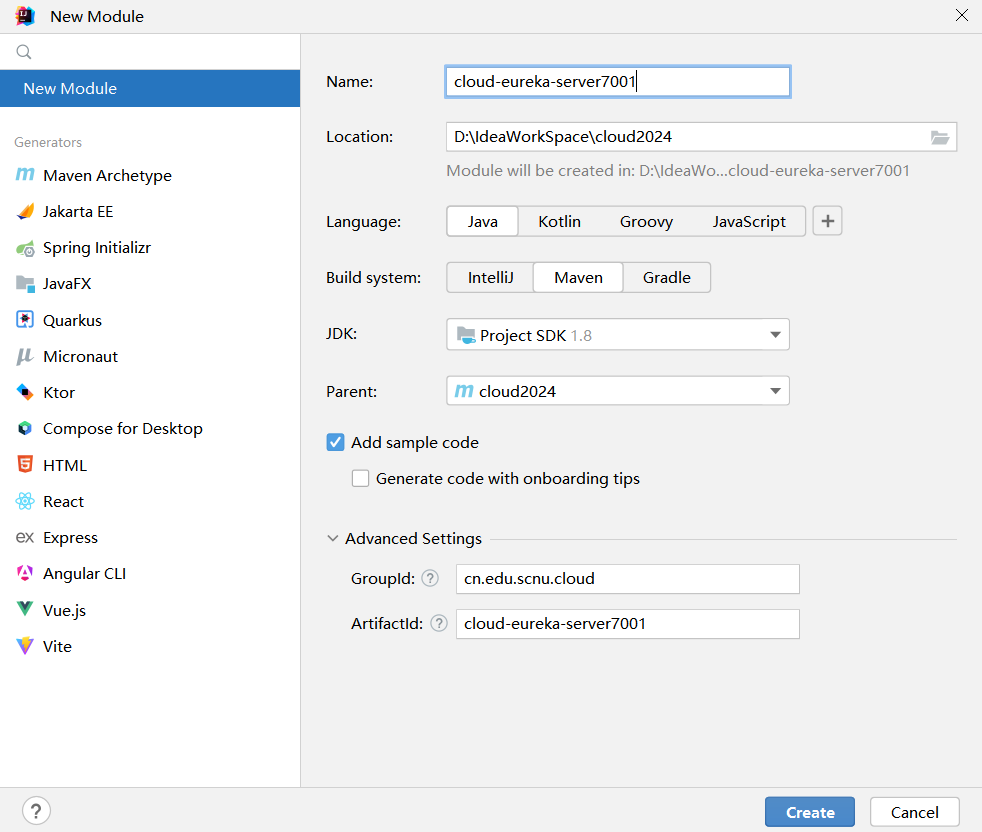
eureka.server.enable-self-preservation=false



并维持心跳连接。这样系统的维护人员可以通过Eureka Server来监控系统中各个微服务是否正常运行。

## 单机Eureka构建步骤

#### 建Module：cloud-eureka-server7001



#### POM

*<?*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>

<parent>

<groupId>cn.edu.scnu.cloud</groupId>

<artifactId>cloud2024</artifactId>

<version>1.0-SNAPSHOT</version>

</parent>

<artifactId>cloud-eureka-server7001</artifactId>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

<dependency>

<groupId>cn.edu.scnu</groupId>

<artifactId>cloud-api-commons</artifactId>

<version>${project.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

</dependency>

</dependencies>

</project>

#### 添加配置文件application.yml

server:

port: 7001

eureka:

instance:

hostname: localhost

client:

register-with-eureka: false

fetchRegistry: false

service-url:

defaultZone: http://localhost:7001/eureka

#### 启动类加注解

package cn.edu.scnu.cloud;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer *//声明当前的微服务是注册中心*

public class EurekaMain7001 {

public static void main(String[] args) {

SpringApplication.*run*(EurekaMain7001.class,args);

}

}

#### 测试

<http://localhost:7001/>

### 服务提供者

**EurekaClient端:将cloud-provider-payment8001注册进EurekaServer成为服务提供者provider**

#### POM

修改Module：cloud-provider-payment8001,在POM这添加依赖

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

#### application配置文件中添加

eureka.client.register-with-eureka=true

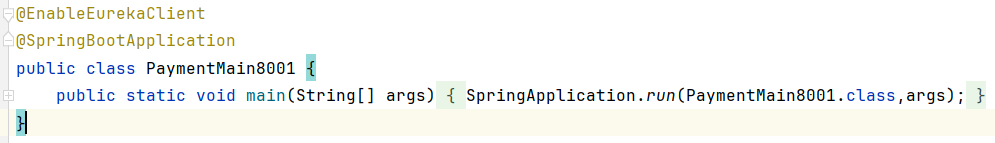
eureka.client.fetchRegistry=true

eureka.client.service-url.defaultZone= http://localhost:7001/eureka

#### 主启动

在启动方法前面加上注解

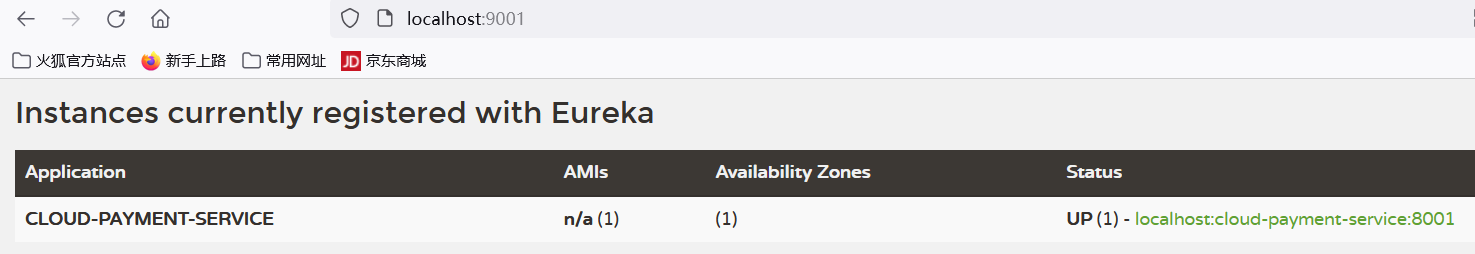
@EnableEurekaClient



#### 测试

先启动EurekaServer,再启动cloud-provider-payment8001

<http://localhost:7001/>



### 服务消费者

**EurekaClient端:将cloud-consumer-order9001注册进EurekaServer成为服务消费者consumer**

#### POM

建Module：cloud-consumer-order80，修改pom

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

#### 写全局配置文件

eureka.client.register-with-eureka=true

eureka.client.fetchRegistry=true

eureka.client.service-url.defaultZone= http://localhost:7001/eureka

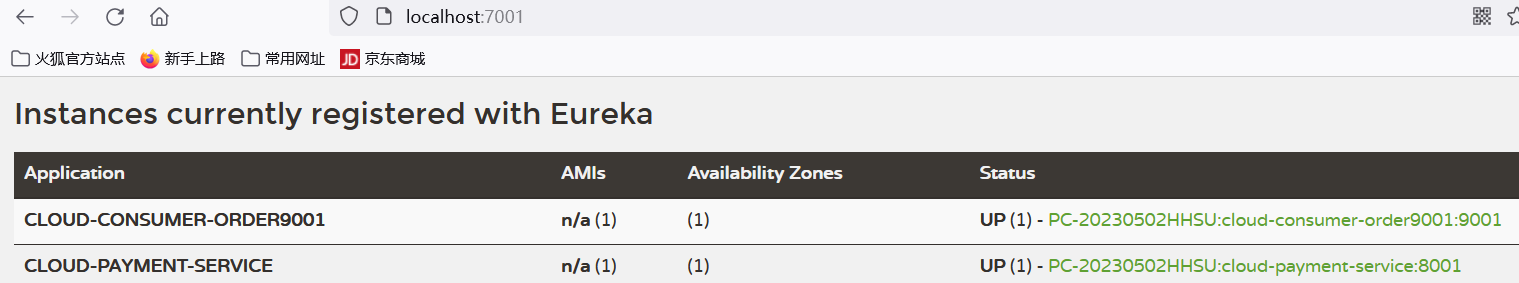
#### 主启动

@EnableEurekaClient

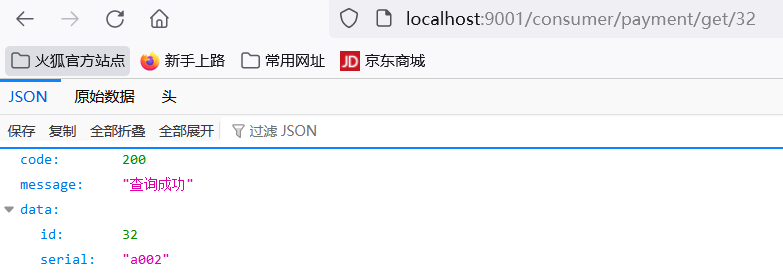
#### 测试

1. 先要启动EurekaServer，7001服务
2. 再先后启动服务提供者8001服务和服务消费者9001服务
3. eureka服务器

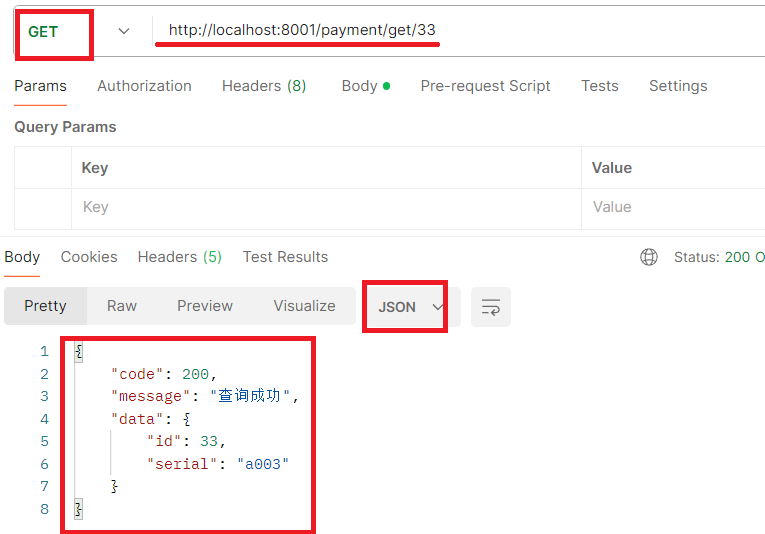
<http://localhost:7001/>



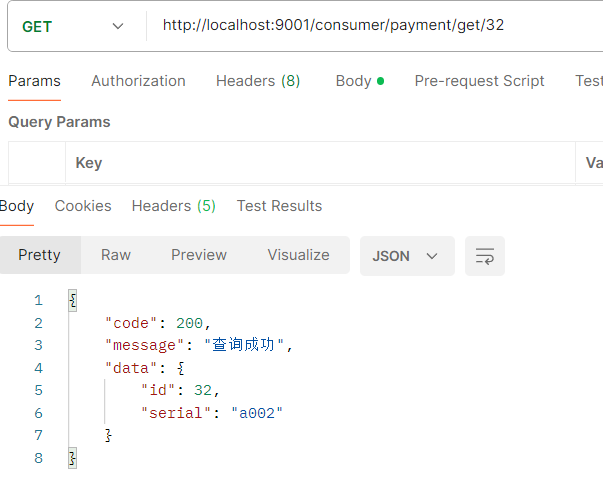
1. 测试查询：[http://localhost:9001/consumer/payment/get/32](http://localhost/consumer/payment/get/31)



1. 测试添加：postman测试添加



1. 测试9001服务



# Ribbon负载均衡

## 概述

### Ribbon是什么

Spring Cloud Ribbon是基于Netflix Ribbon实现的一套客户端负载均衡的工具。

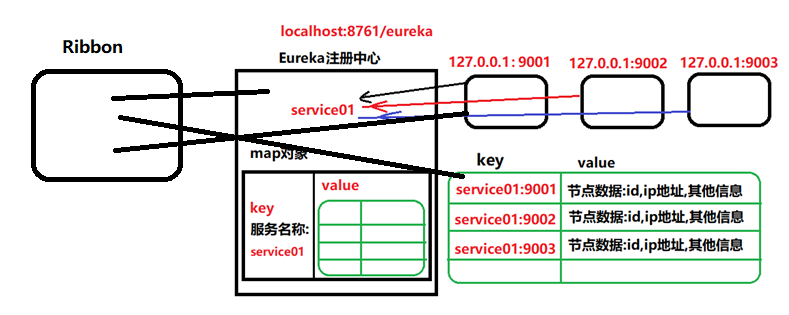
简单的说，Ribbon是Netflix发布的开源项目，主要功能是提供客户端的软件负载均衡算法和服务调用。

Ribbon客户端组件提供一系列完善的配置项，如：连接超时，重试等。

Ribbon启动之后,作为Eureka的客户端，到Eureka注册中心实现**抓取存放在服务端的服务提供者列表list**(每30秒重新抓取一次),这样就可以实现动态的对服务负载均衡访问。

绕过了nginx的静态文件配置.修改环境重启nginx;

这个客户端的访问服务的功能，需要依赖RestTemplate，发起http请求，请求的是服务名称)



### 官网资料

<https://github.com/Netflix/ribbon/wiki/Getting-Started>

* 未来替换方案
  + Spring Cloud LoadBalancer

### Ribbon能干嘛

#### LB（负载均衡）

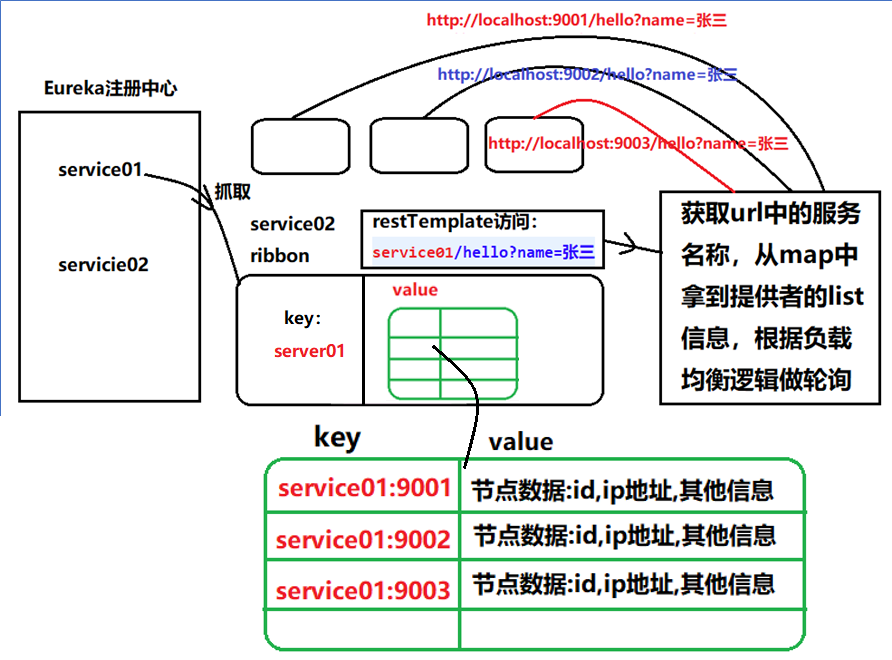
1. 简单的说就是将用户的请求平均分配到多个服务器上，从而达到系统的(高可用)。
2. 常见的负载均衡有软件Nginx，LVS，硬件F5等。
3. **Ribbon VS Nginx区别**：
   * + - Nginx是服务器负载均衡,是主机间的负载均衡，客户端所有请求都会交给Nginx，然后，由nginx实现转发请求。即负载均衡是由服务器端完成的。
       - Ribbon工程内部的负载均衡，在调用微服务接口时候，会在注册中心上获取注册信息服务列表之后缓存到JVM本地，从而在本地实现RPC远程服务调用。
4. 集中式LB
   * + - 即在服务的消费方和提供方之间使用独立的LB设施（可以是硬件，如F5，也可以是软件，如Nginx）,由该设施负责把访问请求通过某种策略转发至服务的提供方；
5. 进程内LB
   * + - 将LB逻辑集成到消费方，消费方从服务注册中心获知有哪些地址可用，然后自己再从这些地址中选择出一个合适的服务器。
       - Ribbon就属于进程内LB，它只是一个类库，集成于消费方进程，消费方通过它来获取到服务提供方的地址。

#### Ribbon的作用

Ribbon=负载均衡+RestTemplate调用

### Ribbon+RestTemplate实现负载均衡的原理

* ribbon客户端启动时,会根据eureka客户端角色到注册中心抓取所有的服务列表.将抓取的数据保存在内存的map
* restTemplate在发起请求之前做拦截.替换,根据请求地址中的服务名称,找到map中key,拿到提供者list,根据负载均衡逻辑(默认轮询)进行轮询的负载均衡访问



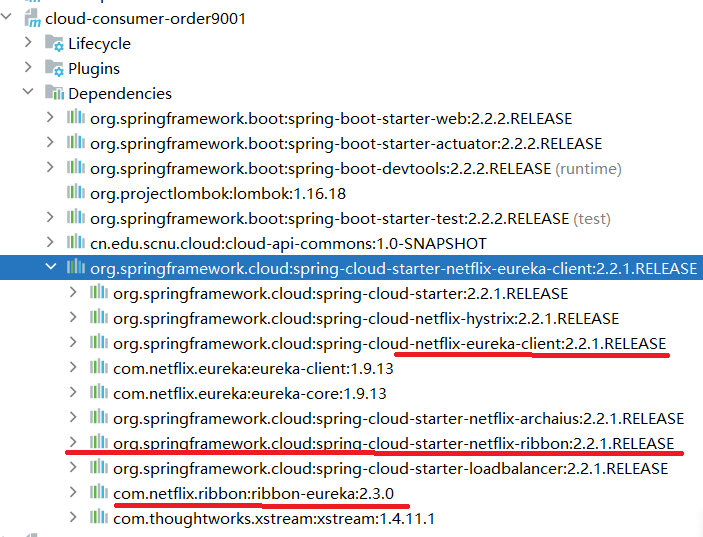
## Ribbon负载均衡案例

### POM

这里不需要添加依赖。

|  |
| --- |
| <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>  </dependency> |

**注意：不需要手动引用，因为eureka-client 自带Ribbon**



## Ribbon核心组件Irule

计算机生成了可选文字:
．。IRule.java
28。工
publicinterfaceIRule{
,/*
*chooseonealiveserverfromlb.allserversor
*lb.upserversaccordingtokey
*
*@returnchoosenServerobject.NULL15returnedifnone
*server15available
*/
,4
publicServerchoose(Objectkey);
：落
publicvoidsetLoadBalancer(ILoadBalancerlb);
.4
publicILoadBalancergetLoadBalancer();
}
9012气j4Lj678901飞Z
23333333333444

### IRule

IRule根据**特定算法**从服务列表中选取一个要访问的服务

1. com.netflix.loadbalancer.RoundRobinRule 轮询，默认策略。
2. com.netflix.loadbalancer.RandomRule 随机
3. com.netflix.loadbalancer.RetryRule 先按照RoundRobinRule的策略获取服务，如果获取服务失败则在指定时间内会进行重试，获取可用的服务
4. WeightedResponseTimeRule 对RoundRobinRule的扩展，响应速度越快的实例选择权重越大，越容易被选择
5. BestAvailableRule 会先过滤掉由于多次访问故障而处于断路器跳闸状态的服务，然后选择一个并发量最小的服务
6. AvailabilityFilteringRule 先过滤掉故障实例，再选择并发较小的实例
7. ZoneAvoidanceRule 默认规则，复合判断server所在区域的性能和server的可用性选择服务器

### 如何替换

* 修改cloud-consumer-order9001
* 注意配置细节

官方文档明确给出警告：

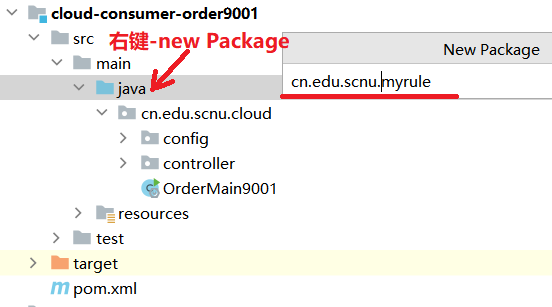
[https://cloud.spring.io/spring-cloud-static/Hoxton.SR1/reference/htmlsingle/#customizing-the-ribbon-client](https://cloud.spring.io/spring-cloud-static/Hoxton.SR1/reference/htmlsingle/" \l "customizing-the-ribbon-client)

计算机生成了可选文字:
5.7.2.Customizingthe只ibbonClient
YouCanC0nfigureSomebitsofaRibbonClientbyusingexternaIPropertiesin<client>.ribbon.*,whiChissimilartousingthe
NetflixAPIsnatively,exCeptthatyouCanuSeSpringBootConfigurationfileS.ThenativeoptionsCanbeinspeCtedasstatiCfieIds
incommonclientcon于igKey(partofribbon一Core).
SpringCloudalsoIetsyOutakefuIIControIOftheCIientbydeCIaringadditionaIConfigurati0n(ontopofthe
Ribbonclientconfiguration)using@Ribbonclient,asshowninthefollOWingexample:
「一一一一一一一一一一一一一一一〕AvA
匹onfiguration
@Ribbonclient(naoe="cust.",configuration=customConfiguration.class)
pub二主二classTestconfiguration{
｝肾、、＿,/／、、、、＿
,,
,,
,,
,,
,,
,,
Inthiscase,theclient15composedfromthecomPonentsalreadyinRibbonclientcon于iguration,togetherwithanyin
customcon+iguration(wheretheIattergenerallyoverridestheformer).
之还
Thecustomcon于igurationclassmustbea@configurationclass,buttakecarethatit15notina@componentscanfor
themainapPlicationcontext.otherwise,it15sharedbyallthe@Ribbonclients.Ifyouuse@componentscan(or
@SpringBootApplication),youneedtotakestepstoavoiditbeinginCluded(forinstanCe,youCanputitinaseparate,
non一overlapPingpaCkageorspeCifythepaCkagestoSCanexpliCitlyinthe@Componentscan).

这个自定义配置类不能放在@ComponentScan所扫描的当前包下以及子包下，否则我们自定义的**这个配置类**就会被所有的Ribbon客户端所共享，达不到特殊化订制的目的了。

### 新建package（注意：包的位置）

cn.edu.scnu.myrule



### 在上面包下新建MySelfRule规则类

package cn.edu.scnu.myrule;

import com.netflix.loadbalancer.IRule;

import com.netflix.loadbalancer.RandomRule;

import org.springframework.beans.factory.annotation.Configurable;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class MySelfRule {

@Bean

public IRule myRule(){

return new RandomRule();*//定义为随机*

}

}

### 启动类添加@RibbonClient

package cn.edu.scnu.cloud;

import cn.edu.scnu.myrule.MySelfRule;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

import org.springframework.cloud.netflix.ribbon.RibbonClient;

@RibbonClient(name = "CLOUD-PAYMENT-SERVICE",configuration = MySelfRule.class)

@EnableEurekaClient

@SpringBootApplication

public class OrderMain9001 {

public static void main(String[] args) {

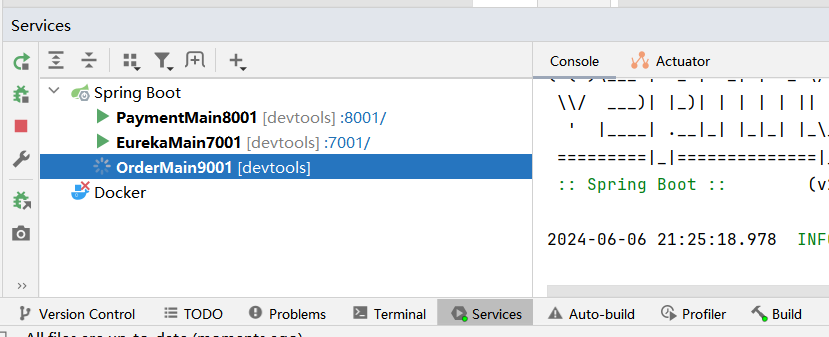
SpringApplication.*run*(OrderMain9001.class,args);

}

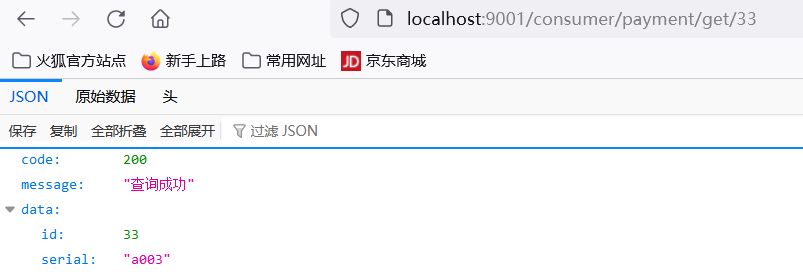
}

## 测试

先启动Eureka，再启动8001，再启动9001（右键点击启动类—run)，或者在service选卡



[http://localhost:9001/consumer/payment/get/31](http://localhost/consumer/payment/get/31)



# OpenFeign服务接口调用

## 概述

### OpenFeign是什么

* Feign是一个声明式的web服务客户端（发出请求，如前面的RestTemplate)，有了Feign使编写web服务客户端变得非常容易，只需创建一个接口并在接口上添加注解即可
* SpringCloud对Feign进行了封装，使其支持了SpringMVC标准注解和HttpMessageConverters。Feign可以与Eureka和Ribbon组合使用以支持负载均衡。

[https://cloud.spring.io/spring-cloud-static/Hoxton.SR1/reference/htmlsingle/#spring-cloud-openfeign](https://cloud.spring.io/spring-cloud-static/Hoxton.SR1/reference/htmlsingle/" \l "spring-cloud-openfeign)

<https://github.com/spring-cloud/spring-cloud-openfeign>

### Feign能做什么

* Feign能干什么？

Feign旨在 使 编写**Java Http客户端**变得更容易。

前面在使用Ribbon+RestTemplate时，利用RestTemplate对Http请求的封装处理，形成了一套模板化的调用方法，RestTemplate是一个类，是基于类的，不是基于接口的，这样编写的代码耦合度比较强，而Feign是基于接口的。

在Feign的实现下，我们只需创建一个接口并使用注解的方式来配置它（以前是DAO接口上面标注Mapper注解，现在是一个微服务接口上面标注一个Feign注解即可），即可完成对服务提供方的接口绑定，简化了使用Spring Cloud Ribbon时，自动封装服务调用客户端的开发量。

* Feign集成了 Ribbon

利用Ribbon维护了**服务提供者**的服务列表信息，并且通过轮询实现了客户端的负载均衡。而与Ribbon不同的是，通过Feign只需要定义服务绑定接口且以声明式的方法，**优雅而简单**的实现了服务调用。

* Feign和OpenFeign两者区别

计算机生成了可选文字:
Feign
Feign是SPringCloud组件中的一个轻量级RESTful的HTTP服务客户端
Feign内置了Ribbon，用来做客户端负蜀的衡，去调用服务注册中心的
服务。Feign的使用方式是：使用Feign的注解定义接口，调用这个接口
，就可以调用服务注册中心的服务工
<dependency>
<groupld>org·Springframework.cloud</groupld>
<artifactld>spring一cloud一starter一feign</artifactld>
</dependency>
OPenFeign
OpenFeign是SPringCloud在Feign的勤出上支持7SpringMvC的注解
，如＠RequesMapPing等等．OpenFeign的＠FeignClient可以解析
5pringMvC的＠RequestMaPPing注解下的接口，并通过动态代理的方
式产生实现类，实现类中做负载均衡并调用其他服务．
<dependency>
<groupld>org.Springframework.cloud</groupld>
<artifactld>spring一cloud一starter一openfeign</artifactld>
</dependency>

## OpenFeign的使用

### 接口+注解

微服务调用接口+@FeignClient

### 新建Module：cloud-consumer-feign-order9001

### POM

注意：openFeign也是自带bibbon

*<?***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**>

<**parent**>

<**groupId**>cn.edu.scnu.cloud</**groupId**>

<**artifactId**>cloud2024</**artifactId**>

<**version**>1.0-SNAPSHOT</**version**>

</**parent**>

<**artifactId**>cloud-consumer-feign-order9001</**artifactId**>

<**dependencies**>

*<!--openfeign-->*

<**dependency**>

<**groupId**>org.springframework.cloud</**groupId**>

<**artifactId**>spring-cloud-starter-**openfeign**</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework.cloud</**groupId**>

<**artifactId**>spring-cloud-starter-netflix-eureka-client</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>cn.edu.scnu.cloud</**groupId**>

<**artifactId**>cloud-api-commons</**artifactId**>

<**version**>${project.version}</**version**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-web</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-actuator</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-devtools</**artifactId**>

<**scope**>runtime</**scope**>

<**optional**>true</**optional**>

</**dependency**>

<**dependency**>

<**groupId**>org.projectlombok</**groupId**>

<**artifactId**>lombok</**artifactId**>

<**optional**>true</**optional**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-test</**artifactId**>

<**scope**>test</**scope**>

</**dependency**>

</**dependencies**>

</**project**>

### application.properties

**server.port** = **9001**

**spring.application.name**=**cloud-consumer-feign-order9001**

**eureka.client.register-with-eureka**=**true**

**eureka.client.fetchRegistry**=**true**

**eureka.client.service-url.defaultZone**= **http://localhost:7001/eureka**

### 主启动类

**package** cn.edu.scnu.cloud;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;

**import** org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication

@EnableEurekaClient

@EnableFeignClients

**public class** OrderFeignMain9001 {

**public static void** main(String[] args) {

SpringApplication.*run*(OrderFeignMain9001.**class**,args);

}

}

### 业务类

#### 业务逻辑接口+@FeignClient配置调用provider服务

#### 新建PaymentFeignService接口并新增注解@FeignClient

创建cn.edu.scnu.cloud.service，在该包中创建PaymentFeignService接口，并在该接口添加@FeignClient主角

**package** cn.edu.scnu.cloud.service;

**import** cn.edu.scnu.cloud.entity.CommonResult;

**import** cn.edu.scnu.cloud.entity.Payment;

**import** feign.Param;

**import** org.springframework.cloud.openfeign.FeignClient;

**import** org.springframework.stereotype.Component;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PathVariable;

@Component

@FeignClient(value = **"CLOUD-PAYMENT-SERVICE"**)

**public interface** PaymentFeignService {

@**GetMapping**(value = **"/payment/get/{id}"**)

**public** CommonResult getPaymentById(@PathVariable(**"id"**) Long id);

}

#### 控制层Controller

**package** cn.edu.scnu.cloud.controller;

**import** cn.edu.scnu.cloud.entity.CommonResult;

**import** cn.edu.scnu.cloud.entity.Payment;

**import** cn.edu.scnu.cloud.service.PaymentFeignService;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RestController;

**import** javax.annotation.Resource;

**@RestController**

**public class** OrderFeignController {

@Resource

**private** PaymentFeignService **paymentFeignService**; *//调用远程的微服接口*

@GetMapping(value = **"/consumer/payment/get/{id}"**)

**public** CommonResult<Payment> getPaymentById(@PathVariable(**"id"**) Long id){

**return paymentFeignService**.getPaymentById(id);

}

}

### 测试

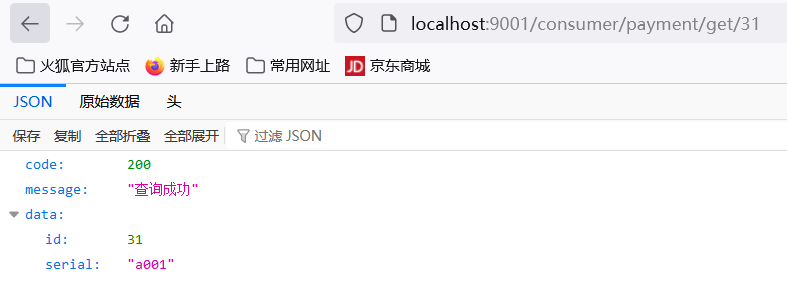
1. 先启动Eureka7001
2. 再启动2个微服务8001/8002(将8001复制一份，打开后改端口号）

先修改8001的PaymentController，添加@Value注解，修改get方法。

@Value("${server.port}")  
private String port;

@GetMapping(value = "/payment/get/{id}")  
public CommonResult<Payment> getPaymentById(@PathVariable("id") Long id){  
 Payment payment = paymentService.getPaymentById(id);  
  
 *log*.info("\*\*\*\*\*查询结果："+payment);  
 if (payment!=null){ *//说明有数据，能查询成功* return new CommonResult(200,"查询成功~~~~~~,端口号："+port,payment);  
 }else {  
 return new CommonResult(444,"没有对应记录，查询ID："+id+"，端口号："+port,null);  
 }  
}

1. 启动OpenFeign微服务：cloud-consumer-feign-order9001
2. [http://localhost:9001/consumer/payment/get/31](http://localhost/consumer/payment/get/31)



1. Feign自带负载均衡配置项

### 小总结

计算机生成了可选文字:
@FeignClient(Value=
",CLOUD一PAYMENT一SERVICE"'
pU0llCIntert.Ce尸ayment卜elgnberVICe
｛一
InstanCescurrentlyregis'
邱etMapping.(value="'/payment/get/{id}"')
commonResult<Payment>_getPaymentByld(@Pathvariable(""id")Longid）二
全＿\
，仑n,rOll竺．java…
CLOUO一pAYMEN不SERVICE
nla(2
paymentControllergetpaymentByldo
Genor日1Info
邱et州apping(value="/payment/get/{id}’乍
publiccommonResult<Payment>getPaymentByld(@Pathvariable("id"")Longid)
total一avail一memory
Paymentpayment=paymentservice.getPaymentByld(id）二
乙09.1nfo('"****＊六询结果：{}"',payment);
enVIFOnlTlent
if(payment
retUrn
}else{
retUrn
}
num一of一cpus
new
nuii){
commonResult(
code:200,,1．。、、dge;”六询成功”'+""\t服务端11:""+serverport,payment）二
new
commonResult(code:444,me、sage:'”没有对应记求，六询10:""+id,data:null);
}
内浑曲，饮”八MIS

## OpenFeign超时控制

### 超时设置，故意设置超时演示出错情况

#### 服务提供方8001故意写暂停程序

@Value(**"${server.port}"**)

**private** String **port**;

@GetMapping(value = **"/payment/feign/timeout"**)

**public** String paymentFeignTimeout(){

**try** {

TimeUnit.***SECONDS***.sleep(3); *//单位秒*

}**catch** (Exception e) {e.printStackTrace();}

**return port**;

}

#### 服务消费方9001的接口添加超时方法

在cloud-consumer-feign-order9001項目的PaymentFeignService接口中添加超時方法

@GetMapping(value = **"/payment/feign/timeout"**)

**public** String paymentFeignTimeout();

#### 服务消费方9001的控制器添加超时方法OrderFeignController

在cloud-consumer-feign-order9001項目的OrderFeignController控制器中添加超時方法

@GetMapping(value = **"/consumer/payment/feign/timeout"**)

**public** String paymentFeignTimeout(){

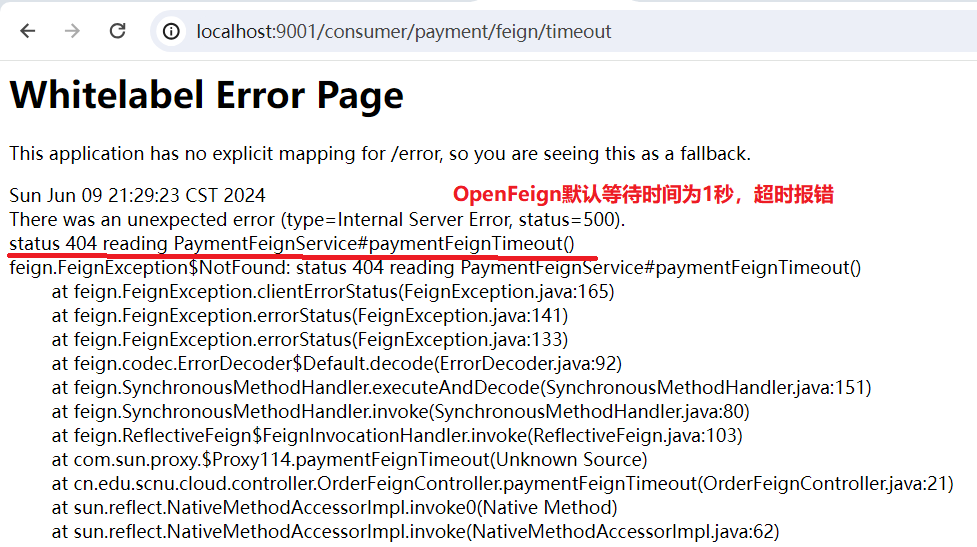
**return paymentFeignService**.paymentFeignTimeout();

}

#### 测试

[http://localhost:9001/consumer/payment/feign/timeout](http://localhost/consumer/payment/feign/timeout)

错误页面，OpenFeign默认等待一秒钟，超过后报错



### 超时原因

默认Feign客户端只等待一秒钟，但是，服务端处理需要超过1秒钟，导致Feign客户端不想等待了，直接报错。

为了避免这样的情况，我们可以设置Feign客户端的超时控制，也即Ribbon的超时时间，因为Feign集成了Ribbon进行负载均衡。

### 在application配置文件中开启OpenFeign客户端超时控制

Feign设置超时时间

使用Feign调用接口分两层，ribbon的调用和hystrix的调用，所以ribbon的超时时间和Hystrix的超时时间的结合就是Feign的超时时间

在cloud-consumer-feign-order9001项目的application配置文件中做超时配置

**ribbon.ReadTimeout**=**6000**

**ribbon.ConnectTimeout**=**6000**

**ribbon.MaxAutoRetries**=**1 #同一台实例最大重试次数,不包括首次调用**

**ribbon.MaxAutoRetriesNextServer**=**1**  **#重试负载均衡其他的实例最大重试次数,不包括首次调用**

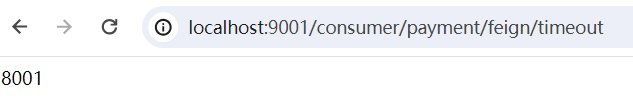
**ribbon.OkToRetryOnAllOperations**=**false #是否所有操作都重试**

*#hystrix的超时时间*

**hystrix.command.default.execution.timeout.enabled**=**true**

**hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds**=**9000**

再次测试，结果：



一般都是使用用Hystrix来解决超时问题，很少使用Feign，我们知道Feign有此功能即可。

**注意：**

默认情况下,GET方式请求无论是连接异常还是读取异常,都会进行重试

非GET方式请求,只有连接异常时,才会进行重试

## OpenFeign日志打印功能

### 日志打印功能

### 是什么

Feign提供了日志打印功能，我们可以通过配置来调整日志级别，从而了解Feign中Http请求的细节。说白了就是对Feign接口的调用情况进行监控和输出。

### 日志级别

NONE：默认的，不显示任何日志

BASIC：仅记录请求方法、RUL、响应状态码及执行时间

HEADERS：除了BASIC中定义的信息之外，还有请求和响应的头信息

FULL：除了HEADERS中定义的信息之外，还有请求和响应的正文及元数据.

### 配置日志bean

**package** com.atguigu.springcloud.config;

**import** feign.Logger;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

@Configuration

**public class** FeignConfig {

@Bean

**public** Logger.Level feignLoggerLevel(){

**return** Logger.Level.***FULL***;

}

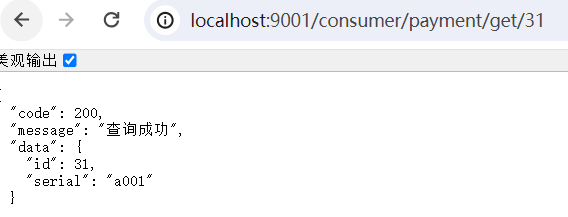
}

### applciation文件开启日志的Feign客户端

**logging.level.cn.edu.scnu.cloud.service.PaymentFeignService**=**debug**

### 后台日志查看

[http://localhost:9001/consumer/payment/get/31](http://localhost/consumer/payment/get/31)



在9001项目控制台可以查看到纤细的日志信息

