# 6.微服务框架（sringcloud）

## 6.1.微服务框架概述

springboot单独存在的意义不大，独立运行、自动配置，完全是为了微服务框架而创建的，springcloud框架必须基于springboot工程才可以实现

**什么是微服务**

当单体系统纵向拆分，拆分后的独立系统就叫微服务，每个微服务都专注一个功能领域，领域之间通过服务调用传递数据

**什么是微服务框架**

当微服务特别庞大的时候，就需要从管理、监控角度出发，对整个集群做到架构级别的技术整合-这种技术就称为微服务框架

**微服务框架技术**

常见的微服务框架有springcloud、dubbo（阿里，12年宣布停更，17年恢复更新）

springcloud:一种轻量级的，组件丰富全面的微服务框架技术，核心：注册发现组件eureka（不能由别的组件代替），实现的http的协议非常丰富，开发投入成本较低

dubbo：组件没有springcloud丰富，注册发现组件可以是eureka（http协议），redis，Zookeeper（rpc协议），使用rpc协议的技术，性能高，而http协议灵活度高

技术选型：

springcloud开发成本低，所有组件都是现成的，可以直接使用，更新比较及时，对应市面各种新技术做出更新

dubbo：性能高是dubbo重要优点，但是开发成本高

## 6.2.springcloud组件介绍

1.eureka

2.ribbon

3.zuul

4.feign

5.hystrix

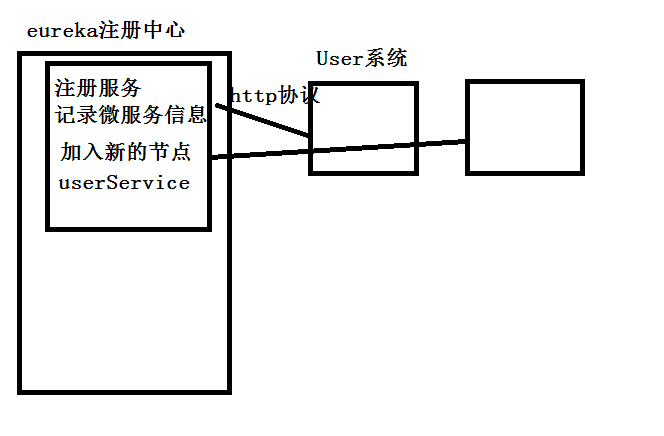
6.spring cloud config

## 6.3.eureka

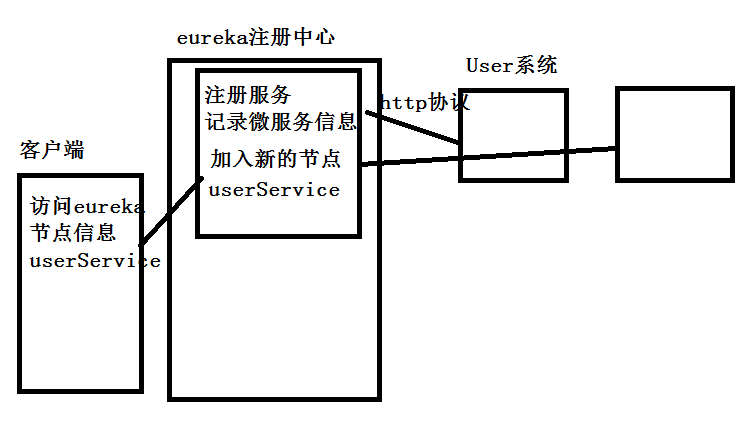
### 6.3.1.eureka介绍

是springcloud的核心组件（管理单位），完成当前所有集群微服务的注册与发现逻辑的核心进程，可以动态的处理所有的负载均衡访问时对集群server的管理工作

服务的注册：微服务节点作为真正的服务，功能的提供者，可以动态注册在eureka中心



服务的发现：客户端（eureka的客户端）访问eureka，动态获取某一个服务的真正提供者信息，从而发起访问调用



eureka组件的角色：

注册中心：称为eureka的服务端角色

注册在中心的微服务们：称为eureka的客户端角色

今日内容：

eureka

ribbon

zuul

easymall项目

### 6.3.2.搭建eureka注册中心

1.创建quickstart工程（eureka-server）

2.pom文件需要继承springboot

<parent>

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

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

<version>1.5.9.RELEASE</version>

</parent>

还需要引入springcloud资源，参考（课前资料\01-课前资料\06-springcloud\09-测试工程\eureka-server-center\pom.xml）的pom文件

加入dependencyManagement，注意版本需要改为Edgware.RELEASE

<dependencyManagement>

<dependencies>

<dependency>

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

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

<version>Edgware.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

引入eureka依赖

<dependency>

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

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

</dependency>

3.application.properties

端口号

server.port=8761

主机名，客户端通过此主机名访问eureka服务端，如果是localhost可以省略，如果是其他(www.eureka01.com)，则需要修改hosts或dns配合

eureka.instance.hostname=localhost

与eureka客户端角色相关的配置，注册中心eureka服务端，本身启动时，即具备服务端角色，也可以本身就是一个客户端，此配置就是eureka是否要作为客户端使用

eureka.client.registerWithEureka=false

eureka自身即不作为客户端注册，也不从自己的服务列表中发现服务

eureka.client.fetchRegistry=false

对外暴露的注册地址（接口）

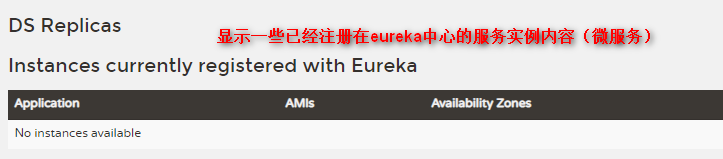
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

4.启动类

|  |
| --- |
| **package** cn.tedu;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  @SpringBootApplication  @EnableEurekaServer //启动eureka服务端注解  **public** **class** StarterEurekaServer {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterEurekaServer.**class**, args);  }  } |

5.启动eureka

localhost:8761



### 6.3.3.eureka客户端-所有微服务都可以是客户端

1.quickstart（eureka-client01）

2.pom文件引入依赖（课前资料\01-课前资料\06-springcloud\09-测试工程\eureka-client\pom.xml）

<parent>

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

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

<version>1.5.9.RELEASE</version>

</parent>

<dependencyManagement>

<dependencies>

<dependency>

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

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

<version>Edgware.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependency>

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

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

</dependency>

3.application.properties

端口号

server.port=9001

在向eureka注册时，携带此参数，并在eureka服务中心以服务名称存在

spring.application.name=service01

指向eureka注册中心暴露的接口

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

4.启动类

|  |
| --- |
| **package** cn.tedu;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;  @SpringBootApplication  @EnableEurekaClient  **public** **class** StarterEurekaClient01 {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterEurekaClient01.**class**, args);  }  } |

5.启动客户端，观察服务端页面的变化

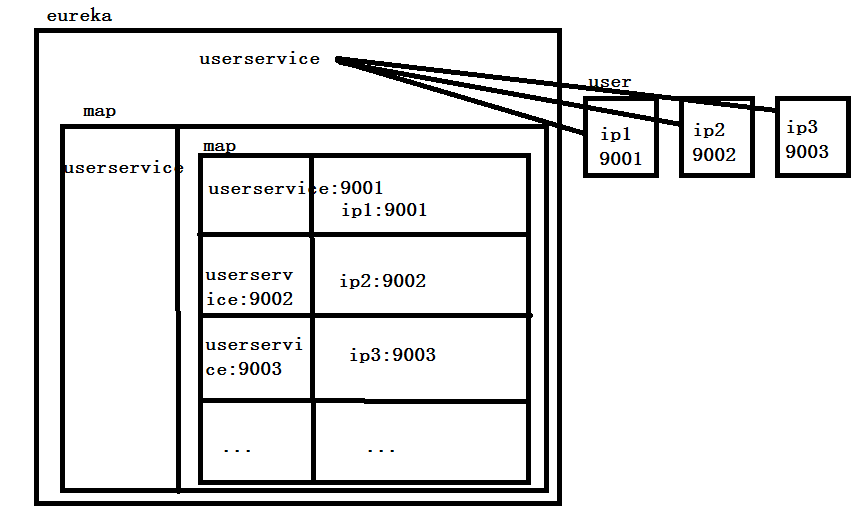
### 6.3.4.注册的逻辑

**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,每30秒发送一个心跳检测(续约),每60秒注册中心检测一次所有的服务续约,一旦发现超过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注册的信息结构**

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微服务心跳发送未成功，造成误删除给集群带来巨大伤害的机制

当整个集群的15%以上同时检测超过90秒未发送心跳，将会开启保护机制，一旦开启，所有的注册信息，在eureka注册中心不会进行删除，默认情况下开启保护机制，可以在服务端添加配置关闭保护机制

eureka.server.enable-self-preservation=false

### 6.3.5.在测试案例中添加访问代码

浏览器可以访问client01微服务

|  |
| --- |
| **package** cn.tedu.controller;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  @RestController  **public** **class** HelloController {  //返回端口，标识响应的微服务节点  @Value("${server.port}")  **private** String port;  @RequestMapping("hi")  **public** String sayHi(String name){  **return** "hello "+name+" i am from "+port;  }  } |

将eureka-client01拷贝成eureka-client02

修改启动类名称为02，端口为9002

但是目前浏览器只能直接访问微服务，不能通过eureka注册中心访问，如果要通过注册中心访问，需要ribbon

## 6.4.ribbon

### 6.4.1.介绍

springcloud中，提供一个客户端组件，用来负载均衡的访问某个服务用的

实现内部服务调用另一个服务的客户端

ribbon启动之后，作为eureka的客户端，到eureka注册中心动态抓取服务提供者列表（30秒重新抓取一次），可以实现动态对服务负载均衡的访问（绕过了nginx，也绕过了静态页面，也无需nginx和后台服务进行绑定），ribbon需要依赖restTemplate，来发起http请求

### 6.4.2.搭建一个ribbon客户端

1.quickstart（ribbon-client）

2.pom文件导入springcloud、依赖eureka、依赖ribbon

<parent>

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

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

<version>1.5.9.RELEASE</version>

</parent>

<dependencyManagement>

<dependencies>

<dependency>

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

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

<version>Edgware.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-ribbon</artifactId>

</dependency>

3.application.properties

server.port=9004

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

spring.application.name=service02

4.启动类

需要加入eureka客户端注解，并加载一个实现ribbon负载均衡访问的restTemplate对象

|  |
| --- |
| **package** cn.tedu;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.client.loadbalancer.LoadBalanced;  **import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.web.client.RestTemplate;  @SpringBootApplication  @EnableEurekaClient  **public** **class** StarterRibbonClient {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterRibbonClient.**class**, args);  }  //生成具备ribbon拦截功能的restTemplate  @Bean  @LoadBalanced//此注解可以使restTemplate可以访问eureka中的服务名称  //并且实现负载均衡，默认是roundRobin(是IRule的实现类)轮询  **public** RestTemplate initRestTemplate(){  **return** **new** RestTemplate();  }  } |

5.测试类

实现通过ribbon访问eureka服务端并负载均衡方法服务端对应的所有微服务节点

浏览器访问ribbon的hello，返回的是微服务的响应内容

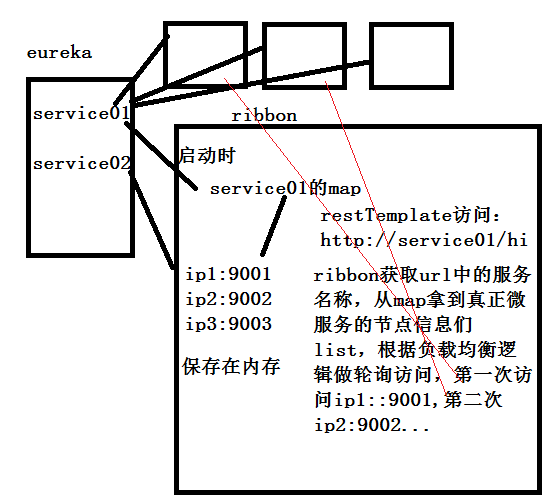
|  |
| --- |
| **package** cn.tedu.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  **import** org.springframework.web.client.RestTemplate;  @RestController  **public** **class** HelloController {  @Autowired  **private** RestTemplate restTemplate;  @RequestMapping("hello")  **public** String sayHello(String name){  //访问eureka注册中心的服务，动态负载均衡访问service01的所有节点的"hi"请求  String resultStr = restTemplate.getForObject(  "http://service01/hi?name="+name, String.**class**);  **return** "ribbon:"+resultStr;  }  } |

浏览器访问：http://localhost:9004/hello?name=zhangsanfeng

### 6.4.3.ribbon实现负载均衡逻辑

1.ribbon启动时，根据eureka客户端角色，到注册中心抓取所有的服务map，将抓取的数据保存在内存中的map

2.restTemplate发起请求之前做拦截和替换，根据请求地址中的服务名，找到map中的key，拿到微服务节点的信息list，根据负载均衡逻辑进行轮询访问（默认轮询）



**其他负载均衡IRule的实现类**

除了轮询，还提供了其他访问逻辑，例如随机，权重

IRule实现类：

RandomRule：实现拦截拼接后，随机选取节点的方式

WeightedResponseTimeRule：根据每个节点平均响应时间，做动态的权重值变化，响应速度越快，权重值越大

**随机访问案例**

在配置类中（启动类）利用@Bean创建一个IRule的实现类RandomRule，实现随机访问

|  |
| --- |
| //创建自定义负载均衡逻辑类：随机负载均衡  @Bean  **public** IRule myRule(){  **return** **new** RandomRule();  } |

由于springboot的存在，当程序启动时，就会加载此对象，并交给框架维护，当前环境会根据此类进行负载均衡逻辑（替换默认的轮询）

### 6.4.4.动态添加一个节点

将eureka-client01、02复制成03，修改启动类为03，端口9003，实现微服务集群的动态扩容（缩容）

动态添加（删除）无需其他任何修改，完全由服务器维护

### 6.4.5.将eureka服务端打包

在pom.xml中添加打包插件

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

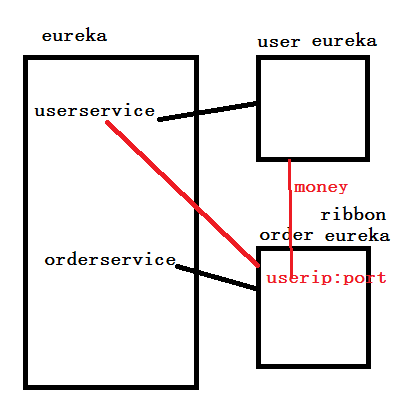
</build>

项目右键->run as->maven install

将打完的jar包（target下）拷贝到一个文件夹，然后在此文件夹下新建一个文件：eureka-start.bat，内容：

java -jar ./eureka-server-0.0.1-SNAPSHOT.jar

### 6.4.6.根据eureka和ribbon，修改order-user



**修改order系统**

1.注册中心用eureka-server

2.order的pom中添加springcloud相关依赖,edgware资源、eureka、ribbon

<dependencyManagement>

<dependencies>

<dependency>

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

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

<version>Edgware.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<!-- eureka客户端 -->

<dependency>

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

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

</dependency>

<!-- ribbon客户端 -->

<dependency>

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

<artifactId>spring-cloud-starter-ribbon</artifactId>

</dependency>

3.application.properties

其他不动，添加如下两行配置

注册到eureka服务中心的节点名称

spring.application.name=orderdemoservice

注册向eureka的地址

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

4.启动类

添加eureka客户端注解、restTemplate的ribbon的注解

|  |
| --- |
| **package** cn.tedu;  **import** org.mybatis.spring.annotation.MapperScan;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.client.loadbalancer.LoadBalanced;  **import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.web.client.RestTemplate;  @SpringBootApplication  @MapperScan("cn.tedu.mapper")  @EnableEurekaClient  **public** **class** StarterOrder {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterOrder.**class**, args);  }  @Bean  @LoadBalanced  **public** RestTemplate initRestTemplate(){  **return** **new** RestTemplate();  }  } |

5.OrderService发起请求修改

|  |
| --- |
| Integer success = rt.getForObject(  "http://userdemoservice/user/updatePoint?orderMoney="+  order.getOrderMoney(), Integer.**class**); |

**user系统修改**

1.eureka服务中心eureka-server

2.pom文件引入springcloud、eureka客户端（复制order的dependencyManagement和eureka客户端依赖）

3.application.properties

spring.application.name=userdemoservice

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

4.启动类

添加eureka客户端的注解@EnableEurekaClient

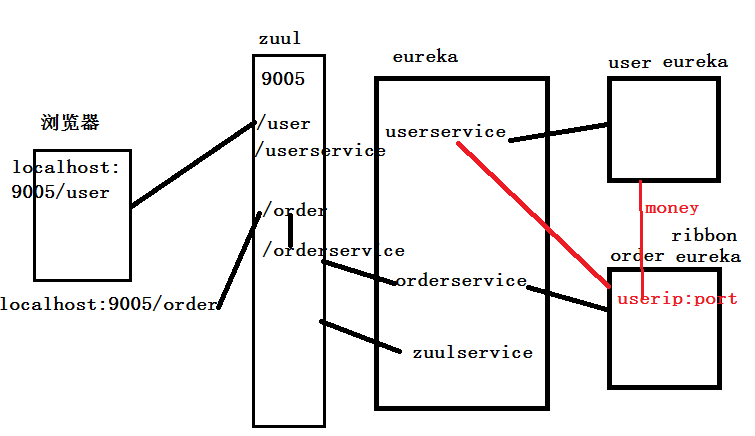
## 6.5.zuul网关

### 6.5.1.概述

整个微服务框架搭建完毕之后，集群唯一对外提供接口访问的角色，ribbon、eureka等不能将内部请求地址对外暴露，一旦暴露，会出现被攻击的危险

**zuul访问逻辑**

zuul底层实现了ribbon+restTemplate，当浏览器访问zuul对外暴露的接口时，zuul内部会通过接口绑定的eureka的服务名称，通过ribbon+restTemplate去访问真实的服务



### 6.5.2.搭建zuul

1.quickstart（zuul-client）

2.pom引入springcloud、eureka、zuul，继承springboot

<parent>

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

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

<version>1.5.9.RELEASE</version>

</parent>

<dependencyManagement>

<dependencies>

<dependency>

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

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

<version>Edgware.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<!-- eureka客户端 -->

<dependency>

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

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

</dependency>

<!-- zuul客户端 -->

<dependency>

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

<artifactId>spring-cloud-starter-zuul</artifactId>

</dependency>

3.application.properties

server.port=9005

spring.application.name=zuulservice

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

zuul.routes.userroute.path=/user/\*\*

zuul.routes.userroute.serviceId=userdemoservice

zuul.routes.order.path=/order/\*\*

zuul.routes.order.serviceId=orderdemoservice

定义路由匹配规则

zuul.routes.user.path=/user/\*\*

定义与访问userroute相关服务配置的path映射地址/user/\*\*，只要访问路径以/user开头的所有url都可以匹配

一旦匹配成功，会寻找当前配置文件中的同名userroute.serviceId的配置，找到eureka中对应的服务名称，找到后，通过ribbon+restTemplate进行访问

url例如：

localhost:9005/user/queryOne?userId=1可以匹配

localhost:9005/user 可以

localhost:9005/use/hello 不可以

**匹配规则符号**

? ：匹配一个字符，例如/user/?，可以匹配/user/a、/user/x，不能匹配：/user/query、/user

\* ：匹配一级路径，例如/user/\*，可以匹配/user/query、/user/insert，不能匹配：/user/query/cart

\*\* : 匹配多级路径，例如/user/\*\*,只要以/user开头都可以匹配

4.启动类

|  |
| --- |
| **package** cn.tedu;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.netflix.zuul.EnableZuulProxy;  @SpringBootApplication  @EnableZuulProxy  **public** **class** StarterZuul {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterZuul.**class**, args);  }  } |

### 6.5.3.转发逻辑

浏览器访问http://localhost:9005/user/user/query/point?userId=1

起始：<http://localhost:9005/user/user/query/point?userId=1>

第二步：进入zuul，将访问zuul的ip:端口和匹配暴露的接口（/user）全部过滤，剩下：

/user/query/point?userId=1

第三步，拼接路由的eureka的服务名称

http://userdemoservice/user/query/point?userId=1

第四步：通过ribbon+restTemplate，去eureka中寻找同名服务名称，将userdemoservice翻译为微服务集群节点list信息（以127.0.0.1:8091为例），进行轮询访问

http://127.0.0.1:8091/user/query/point?userId=1

### 6.5.4.修改nginx

nginx匹配到后将location以及之前的内容全部过滤

假如：

location /user {

proxy\_pass http://127.0.0.1:9005/user;

浏览器：[www.ssm.com/user/query/point?userId=1](http://www.ssm.com/user/query/point?userId=1)

nginx过滤：http://127.0.0.1:9005/user/query/point?userId=1

zuul过滤：http://userdemoservice/query/point?userId=1

ribbon+restTemplate：http://127.0.0.1:8091/query/point?userId=1

所以上面少了一个user

nginx最终配置

server {

listen 80;

server\_name www.ssm.com;

location /user {

proxy\_pass http://127.0.0.1:9005/user/user;

add\_header 'Access-Control-Allow-Origin' '\*';

add\_header 'Access-Control-Allow-Credentials' 'true';

}

location /order {

proxy\_pass http://127.0.0.1:9005/order/order;

add\_header 'Access-Control-Allow-Origin' '\*';

add\_header 'Access-Control-Allow-Credentials' 'true';

}

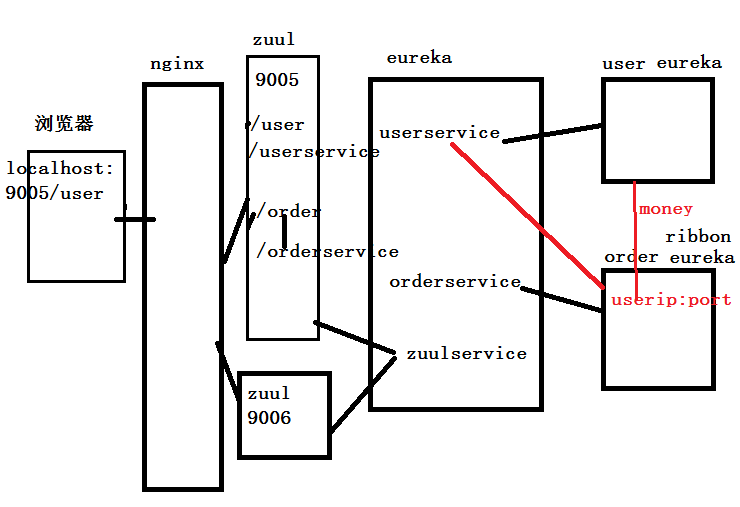
location /{

root easymall;

index index.html;

}

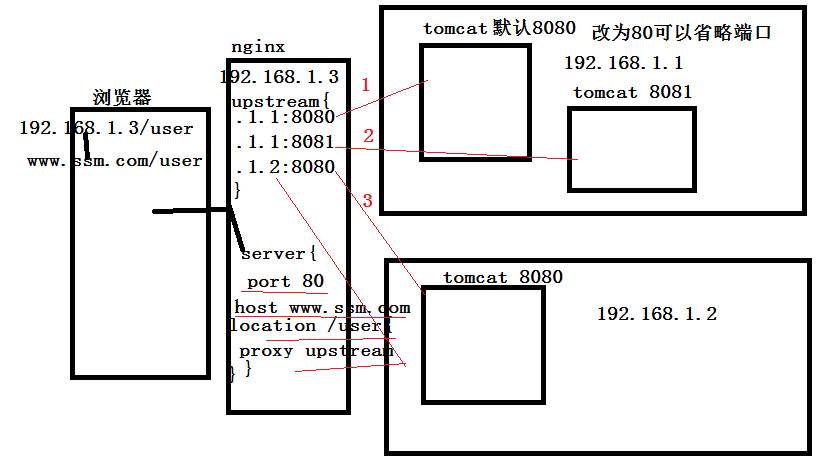
}



作业：order的转发逻辑剖析

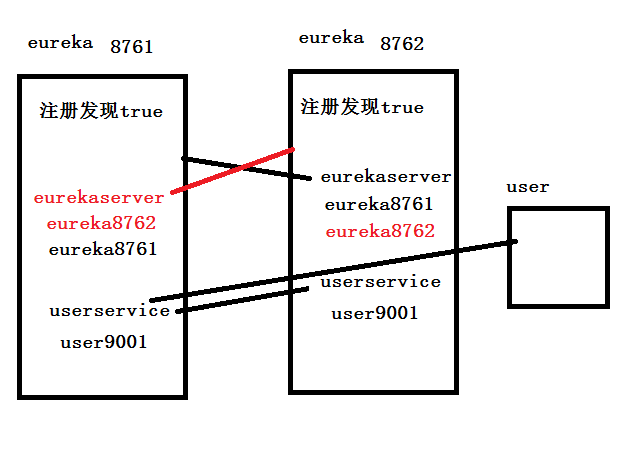
思考题：eureka高可用

配置两台eureka服务中心



## 6.6.eureka高可用集群

### 6.6.1.原理



### 6.6.2.搭建步骤

1.修改eureka-server（以前的）的配置文件

eureka.client.registerWithEureka=true

eureka.client.fetchRegistry=true

spring.application.name=eurekaserver

搭建高可用，服务端需要开启ip优先

eureka.instance.preferIpAddress=true

2.拷贝eureka-server项目为eureka-server02

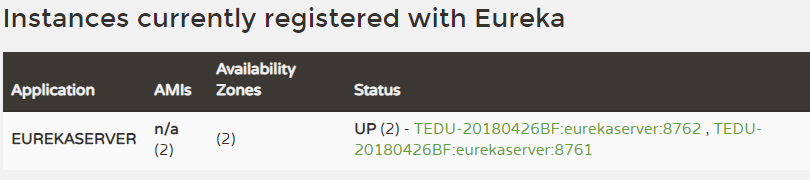
修改配置文件中端口为8762，

8761指向8762, 8762指向8761

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

3.分别启动eureka-server、eureka-server02

相互注册，相互发现



4.开启客户端

修改eureka-client01配置文件，向两个eureka服务中心注册

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka,http://localhost:8762/eureka

然后启动客户端

### 6.6.3.CAP理论（eureka与zookeeper选型）

C:Consistency，数据一致性

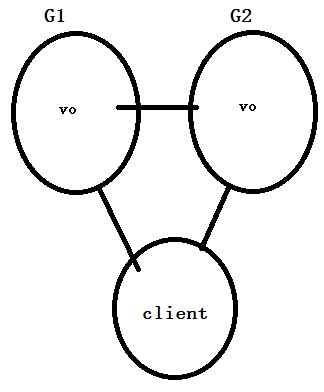
A:Availability，可用性，系统响应速度

P：Partition tolerance，分区容错性

**Partition tolerance**

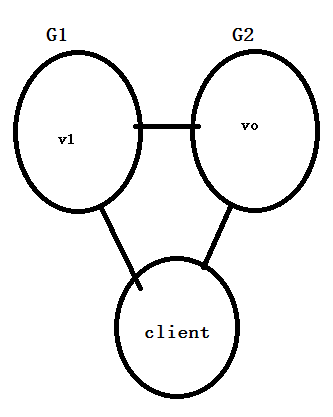
分区容错性：大多数分布式系统，都分布在多个子网络，每个子网络就叫一个区，分区容错意思是：区之间通信可能失败，例如一个区在中国，一个区在美国，它们之间可能会无法通信

如下图，G1和G2是两个区的服务器，G1向G2发送一条消息，G2可能无法收到，系统设计时，必须考虑这种情况，但是一般情况下，分区容错无法避免，所以，我们认为在CAP理论中，P总是成立



**Consistency**

一致性：用户写和读的内容保持一致，例如：client写的是v1，读的就必须是v1，这就是数据一致



问题是，由于client可能想G1写数据，但是在G2读数据，由于G1和G2网络通信失败，造成数据不一致，这就不满足一致性

**Availability**

可用性：即只要服务器收到用户的请求，立刻做出回应，例如client无论向G1还是G2发起请求，只要服务器收到请求，立刻将数据相应给client

**Consistency和Availability的矛盾**

一致性和可用性不能同时成立，原因很简单，因为通信可能失败（分区容错性一定成立）

如果保证G1和G2数据一致性，那么客户端在向G1写数据时，G1必须将数据同步到G2，只有数据同步，才能保证数据的一致性，但是，在同步时，网络可能出现异常，此时client向G2发起读的请求，

如果要保持数据一致性，G2必须等到数据同步后才能响应给client，这样就不能保证可用性

如果要保证可用性，则G2需要立刻向client响应，但此时并未同步到G1的数据，那么响应的数据与client之前写的数据不一致

结论：CAP不能在分布式结构中同时存在，一般只能做到cp或者ap

**eureka和zookeeper选型**

强一致性（偏向cp）：任意时间，查看分布式结构的任意节点，数据都是一致的

弱一致性（最终一致性）（偏向ap）：任意时间，整体一致性未必一致，但是只要过了一段时间（毫秒、微秒）,整体一致性存在，可用性更高

eureka：偏向ap（对一致性要求较低，响应要求较高）

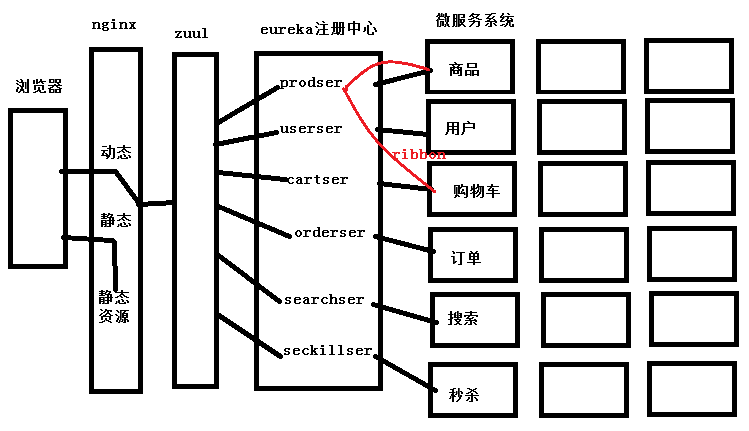
zookeeper:偏向cp，一致性要求较高，响应相对较低

## 6.7.easymall项目结构

### 6.7.1.easymall介绍

模拟目前市面比较常见的电商系统，有商品、用户、订单、购物车、搜索、秒杀相关功能，经过纵向拆分，利用springcloud框架搭建一个高可用，高并发，高性能的系统

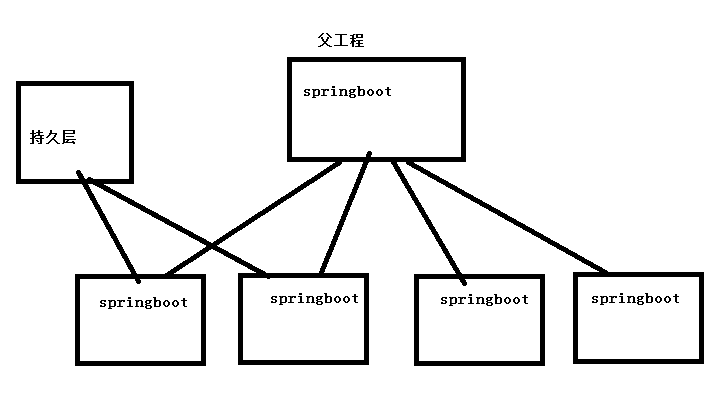
**架构**



### 6.7.2.easymall的项目搭建

**parent工程**

eureka使用之前的eureka-server



1.quickstart（easymll-parent）

2.修改pom文件（参考课前资料\01-课前资料\06-springcloud\07-easymall项目\springcloud-parent）

|  |
| --- |
| <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.tedu</groupId>  <artifactId>easymall-parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>pom</packaging>  <name>easymall-parent</name>  <url>http://maven.apache.org</url>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.9.RELEASE</version>  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  <version>3.3.2</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-ribbon</artifactId>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Edgware.RELEASE</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  </project> |

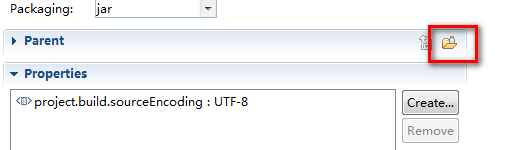
**持久层依赖工程**

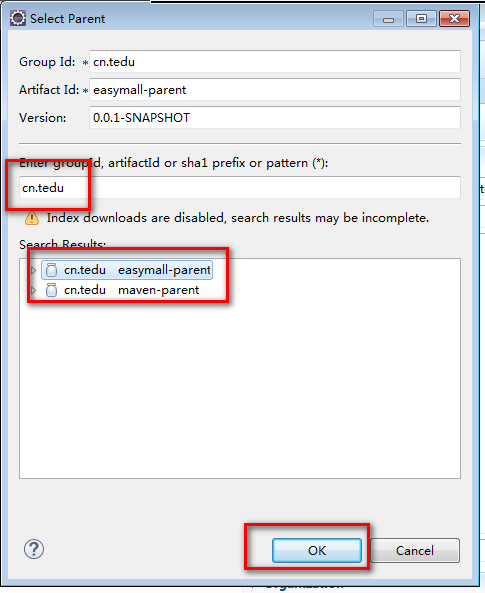
作用：需要用到持久层的工程可以依赖此工程

1.quickstart（easymall-common-repository）

2.pom.xml

继承easymall-parent，jdbc\mysql\mybatis





四个依赖：

<!-- jdbc -->

<dependency>

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

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

</dependency>

<!-- mysql -->

<dependency>

<groupId>mysql</groupId>

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

</dependency>

<!-- mybatis -->

<dependency>

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

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

<version>1.3.0</version>

</dependency>

<!-- 德鲁伊连接池 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.0.14</version>

</dependency>

3.application.properties（此工程不用，子工程配置）

如果在依赖工程中配置数据库datasource相关信息，数据会写死，所以在子工程配置相关信息

4.自定义属性配置类

在程序启动时，加载此类，将配置信息读取到此类中初始化，并最终交给环境，配置到该配置的地方

|  |
| --- |
| **package** cn.tedu.repository.config;  **import** org.springframework.boot.context.properties.ConfigurationProperties;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.context.annotation.Primary;  @Configuration  @ConfigurationProperties("spring.datasource")  **public** **class** DataSourceConfig {  **private** String driverClassName;  **private** String url;  **private** String username;  **private** String password;  **private** Integer maxActive;//最大连接数50  **private** Integer maxIdle;//最大空闲10  **private** Integer minIdle;//最小空闲5  **private** Integer initialSize;//初始连接数6  @Bean  @Primary //当容器中存在多个同类对象，优先使用有@Primary的  **public** DataSourceConfig initDruidDataSource(){  DataSourceConfig dsc = **new** DataSourceConfig();  dsc.setDriverClassName(driverClassName);  dsc.setUrl(url);  dsc.setUsername(username);  dsc.setPassword(password);  dsc.setMaxActive(maxActive);  dsc.setMaxIdle(maxIdle);  dsc.setMinIdle(minIdle);  dsc.setInitialSize(initialSize);  **return** dsc;  }  get&set  } |

**公用工具的工程**

在系统开发过程中，很多系统都可能用到同一个工具类，这种也叫共享资源，可以单独提取到一个工具工程，其他工程依赖此工程即可（pojo，加密，对象转换）

1.quickstart（easymall-common-resources）

2.pom.xml继承easymall-parent

3.拷贝工具类资源

课前资料\01-课前资料\06-springcloud\07-easymall项目\springboot-boot-common-easymall-pojo\src\main

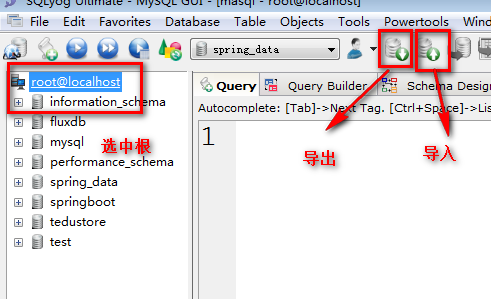
将main文件夹拷贝

打开eclipse中的navigator视图，拷贝到src下，提示是否覆盖，yes to all

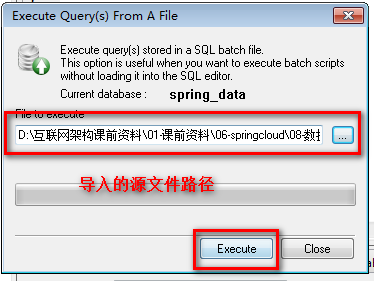
根据编译报错调整内容（此处无报错）

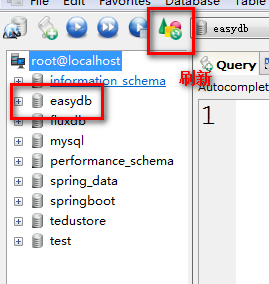
## 6.8.开发第一个功能系统-商品系统

### 6.8.1.数据库准备



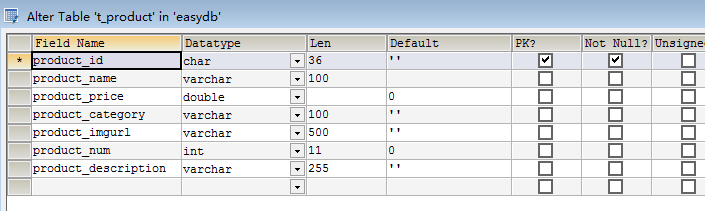
导入路径：课前资料\01-课前资料\06-springcloud\08-数据库\easydb.sql





如果数据库版本低于5.5，可能导入失败

### 6.8.2.商品表t\_product结构



product\_id：商品id

product\_name：商品名

product\_price：价格

product\_category：种类

product\_imgurl：图片路径

图片路径：

<http://image.jt.com/upload/5/e/d/5/4/5/e/b/5f0d34dc-157f-49ba-ad39-1b28927ba6ae_1005714.jpg>

图片作为数据能存入数据库的，字段类型为：blob，但是如果将blob的数据存入数据库，会破坏数据库的索引结构，在查询数据时，会绕过索引，遍历全表，严重影响效率，所以所有的blob数据都只存储路径，这里交给nginx，这里存在本机上，由于图片资源量特别大，这些资源无法存入一台服务器，需要分布式存储

product\_num：库存

product\_description：描述详情

### 6.8.3.搭建商品系统

1.quickstart（easymall-microservice-product）

2.pom.xml

springboot、eureka、ribbon

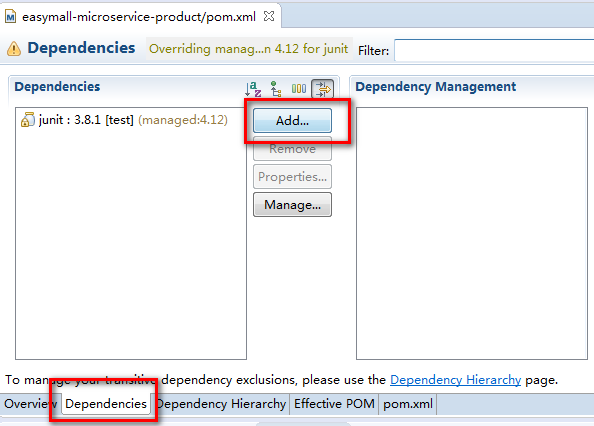
继承easymall-parent

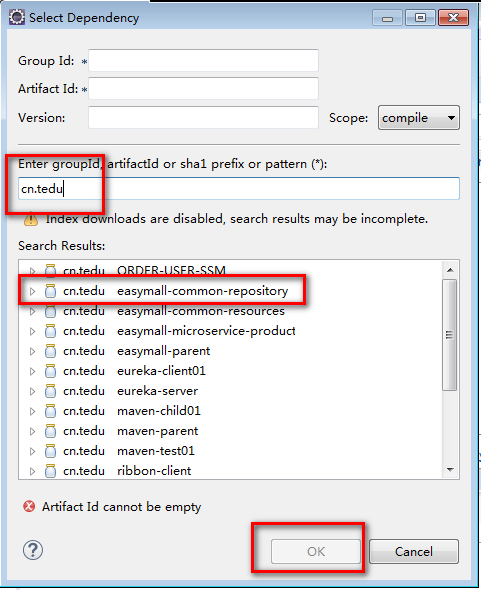
jdbc、mysql、mybatis、连接池

依赖repository

工具类

依赖Resource





3.application.properties

创建src/main/resources下创建application.properties

端口、datasource（德鲁伊连接池）、mybatis、eureka

|  |
| --- |
| server.port=10001  spring.datasource.driverClassName=com.mysql.jdbc.Driver  spring.datasource.url=jdbc:mysql:///easydb  spring.datasource.username=root  spring.datasource.password=root  spring.datasource.type=com.alibaba.druid.pool.DruidDataSource  spring.datasource.initialSize=5  spring.datasource.maxActive=50  spring.datasource.maxIdle=10  spring.datasource.minIdle=5  mybatis.typeAliasesPackage=com.jt.common.pojo  mybatis.mapperLocations=classpath:mapper/\*.xml  mybatis.configuration.mapUnderscoreToCamelCase=true  mybatis.configuration.cacheEnables=false  spring.application.name=productservice  eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka |

4.启动类

需要springboot注解、eureka注解、mapperscan注解

同时，初始化ribbon的restTemplate

|  |
| --- |
| **package** cn.tedu;  **import** org.mybatis.spring.annotation.MapperScan;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.client.loadbalancer.LoadBalanced;  **import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.web.client.RestTemplate;  @SpringBootApplication  @EnableEurekaClient  @MapperScan("cn.tedu.product.mapper")  **public** **class** StarterProductCenter {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterProductCenter.**class**, args);  }  //ribbon的restTemplate  @Bean  @LoadBalanced  **public** RestTemplate initRestTemplateProduct(){  **return** **new** RestTemplate();  }  } |

### 6.8.4.功能实现

分页查询接口文件

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/products/pageManage?page=1&rows=1 |
| 后台接收 | /product/manage/pageManage?page=1&rows=1 |
| 请求方式 | Get |
| 请求参数 | Get提交参数 Integer page,Integer rows |
| 返回数据 | 根据查询结果封装2个数据到EasyUIResult对象中:  Integer total:查询的总条数;  List<Product> rows:查询分页的数据结果; |

**ProductController**

|  |
| --- |
| **package** cn.tedu.product.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  **import** com.jt.common.vo.EasyUIResult;  **import** cn.tedu.product.service.ProductService;  @RestController  **public** **class** ProductController {  @Autowired  **private** ProductService productService;  @RequestMapping("/product/manage/pageManage")  **public** EasyUIResult productPageQuery(Integer page,Integer rows){  EasyUIResult result = productService.productPageQuery(page,rows);  **return** result;  }  } |

**ProductService**

|  |
| --- |
| **package** cn.tedu.product.service;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** com.jt.common.pojo.Product;  **import** com.jt.common.vo.EasyUIResult;  @Service  **public** **class** ProductService {  @Autowired  **private** ProductMapper productMapper;  **public** EasyUIResult productPageQuery(Integer page, Integer rows) {  //准备封装数据的EasyUIResult  EasyUIResult result = **new** EasyUIResult();  //利用mapper查询总条数  Integer total = productMapper.queryTotal();  //确定起始位置和查的条数  Integer start = (page-1)\*rows;  //利用mapper根据起始位置和每页条数查询商品list  List<Product> list = productMapper.queryByPage(start,rows);  //将数据封装到EasyUIResult并返回  result.setTotal(total);  result.setRows(list);  **return** result;  }  } |

**ProductMapper**

|  |
| --- |
| **package** cn.tedu.product.mapper;  **import** java.util.List;  **import** com.jt.common.pojo.Product;  **public** **interface** ProductMapper {  Integer queryTotal();  //如果参数大于两个，映射文件无法直接通过变量名拿到此参数  //可以通过@Param将指定的变量名传给映射文件  List<Product> queryByPage(@Param("start")Integer start, @Param("rows")Integer rows);  } |

**ProductMapper.xml**

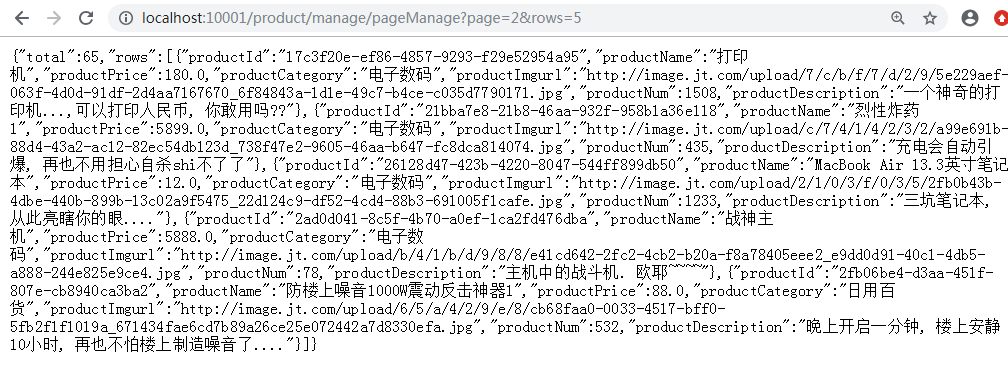
去其他工程中拷贝一个mapper目录，修改xml名称、namespace名称

|  |
| --- |
| <?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.tedu.product.mapper.ProductMapper"*>  <select id=*"queryTotal"* resultType=*"Integer"*>  select count(\*) from t\_product;  </select>  <select id=*"queryByPage"* resultType=*"Product"*>  select \* from t\_product limit #{start},#{rows};  </select>  </mapper> |

启动测试：

启动eureka、启动product工程

浏览器访问：<http://localhost:10001/product/manage/pageManage?page=2&rows=5>



### 6.8.5.页面整合

配置hosts文件127.0.0.1 www.easymall.com

根据js请求，在nginx中做location匹配

js访问路径：<http://www.easymall.com/products/pageManage>

nginx配置修改：

location /products {

proxy\_pass http://127.0.0.1:9005/zuul-product/product/manage;

add\_header 'Access-Control-Allow-Credentials' 'true';

add\_header 'Access-Control-Allow-Origin' '\*';

}

### 6.8.6.配置zuul工程

复制之前的zuul，改名为easymal-micro-zuul

修改配置文件

server.port=9005

spring.application.name=zuulservice

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

zuul.routes.zuul-product.path=/zuul-product/\*\*

zuul.routes.zuul-product.serviceId=productservice

### 6.8.7.显示图片

图片路径：

http://image.jt.com/upload/2/6/4/a/a/5/2/3/ee6c796a-6333-4cd5-a06e-271d876aac8c\_589577.jpg

d盘下创建一个文件夹名为upload

将课前资料\追加文件\02-SSM框架\06-图片资源\upload

将此处的upload文件夹拷贝到d:/upload下

修改hosts文件：127.0.0.1 image.jt.com

启动eureka、zuul、product、nginx，浏览器访问www.easymall.com，点击全部商品

作业一：完成所有商品的显示

作业二：路径的流转过程捋顺了

浏览器请求：http://www.easymall.com/products/pageManage?page=1&rows=5

### 6.8.8.访问逻辑：

1.js起始url：（1）<http://www.easymall.com/products/pageManage>

2.进入nginx，拦截端口：80，拦截域名www.easymall.com，location拦截为/products

会将http://www.easymall.com/products/拦截到并去掉，只剩/pageManage

拼接proxy\_pass（http://127.0.0.1:9005/zuul-product/product/manage）的配置内容：

（2）http://127.0.0.1:9005/zuul-product/product/manage/pageManage

3.通过ip、端口找到zuul

zuul的配置：

zuul.routes.zuul-product.path=/zuul-product/\*\*

zuul.routes.zuul-product.serviceId=productservice

对外暴露的接口为/zuul-product/\*\*即拦截以/zuul-product开头的请求

第二步的转发请求满足zuul暴露的接口，找到和接口绑定的eureka中的微服务名称

再次拼接，去掉和暴露接口匹配的内容，即去掉127.0.0.1:9005/zuul-product，只剩product/manage/pageManage

拼接zuul.routes.zuul-product.serviceId=productservice的value值

（3）http://productservice/product/manage/pageManage

进入eureka

通过ribbon+restTemplate，访问eureka的微服务名称为productservice的节点，拿到eureka保存的双层map，拿到真正微服务的ip:端口的list，进行轮询访问

例如product微服务（其中一个节点）的ip:127.0.0.1，端口：10001

将微服务名称替换为真实的ip+端口，访问真实节点

（4）<http://127.0.0.1:10001/product/manage/pageManage>

和真实节点的ip、端口、controller拦截内容保持一致

## 6.9.单个商品查询

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/products/item/{productId} |
| 后台接收 | /product/manage/item/{productId} |
| 请求方式 | Get |
| 请求参数 | 路径中携带的参数 String productid |
| 返回数据 | 返回查询的Product对象,在ajax中解析使用 |

### 6.9.1.路径携带参数问题

springmvc可以将路径中的参数进行解析，通过@PathVariable

例如/page/hello/5

想要路径中的hello和5的值，在controller接收时，”/page/{变量名}/{变量名}”

按照顺序@PathVariable注解获取变量对应的值

例如在ProductController中添加测试代码：

@RequestMapping("page/{name}/{num}")

**public** String sayHello(@PathVariable String name,@PathVariable Integer num){

**return** "第一个参数为："+name+",第二个为："+num;

}

然后启动eureka和product项目，浏览器访问：

<http://localhost:10001/page/zhangsan/7>

页面效果：



### 6.9.2.单一商品查询步骤

ProductController

|  |
| --- |
| //在类上加入@RequestMapping("/product/manage/")，相当于加了前缀  @RequestMapping("/item/{productId}")  **public** Product queryById(@PathVariable String productIed){  **return** productService.queryById(productIed);  } |

ProductService

|  |
| --- |
| **public** Product queryById(String productIed) {  //**TODO** 1.去redis查  //2.去数据库  //每次去数据库查，非常耗费资源，尤其是热门商品，去之前，先去redis查  Product prod = productMapper.queryById(productIed);  //**TODO** 3.将数据放入redis  **return** prod;  } |

ProductMapper

|  |
| --- |
| Product queryById(String productIed); |

ProductMapper.xml

|  |
| --- |
| <select id=*"queryById"* resultType=*"Product"*>  select \* from t\_product where product\_id=#{productId};  </select> |

页面整合（无需改动）

### 6.9.3.pojo分类

所有的封装类可以统称为pojo

1.entity：严格按照和数据库字段一对一匹配的封装类

2.domain：具有一定业务意义的封装类，例如Pruduct类中处理封装数据库字段外，还添加一个List<Product>属性

3.dto：将要从页面传递给服务器的数据格式对象，不一定和最后保存到数据库的表一一对应

4.vo：将要从后台传给前端页面的数据

## 6.10.商品新增

（暂无图片上传）

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/products/save |
| 后台接收 | /product/manage/save |
| 请求方式 | Post |
| 请求参数 | Product product对象接参 缺少id |
| 返回数据 | 返回SysResult对象的json,其结构:  Integer status; 200表示成功,其他表示失败  String msg;成功返回 “ok”,失败返回其他信息  Object data;根据需求携带其他数据 |
| 备注 | 返回的SysResult对象是一个标准的和ajax对话的后台vo对象,它可以将当前的操作结果通过status传递给ajax,可以将操作中出现任何后台的消息,例如错误信息,以msg传递给前台,也可以将很多其他数据封装到data传递给前台使用 |

### 6.10.1.功能编写

ProductController

|  |
| --- |
| @RequestMapping("/product/manage/save")  **public** SysResult productSave(Product product){  **try** {  productService.productSave(product);  //没有异常，返回200，ok  **return** SysResult.*ok*();  } **catch** (Exception e) {  e.printStackTrace();  //有异常，返回除200的数，其他信息  **return** SysResult.*build*(201, e.getMessage(), **null**);  }  } |

ProductService

|  |
| --- |
| **public** **void** productSave(Product product) {  //补充productId  product.setProductId(UUIDUtil.*getUUID*());  productMapper.productSave(product);  } |

ProductMapper

|  |
| --- |
| **void** productSave(Product product); |

ProductMapper.xml

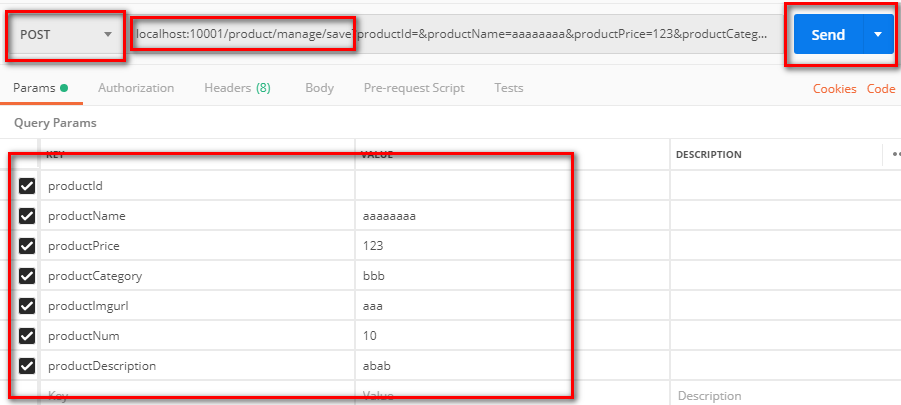
|  |
| --- |
| <insert id=*"productSave"* parameterType=*"Product"*>  insert into t\_product(  product\_id,  product\_name,  product\_price,  product\_category,  product\_imgurl,  product\_num,  product\_description) values(  #{productId},  #{productName},  #{productPrice},  #{productCategory},  #{productImgurl},  #{productNum},  #{productDescription});  </insert> |

### 6.10.2.页面整合

无需改动，只要同一个微服务系统，zuul一般无需改动，nginx转发到zuul配置

### 6.10.3.测试

请求为post请求，浏览器url地址只能拼接get请求，一般可以利用其它插件软件发起post请求，可以使用课前资料-windows环境-Postman-win64-7.0.9-Setup插件完成



## 6.11.商品更新

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/products/update |
| 后台接收 | /product/manage/update |
| 请求方式 | Post |
| 请求参数 | Product product |
| 返回数据 | 返回SysResult对象的json,其结构:  Integer status; 200表示成功,其他表示失败  String msg;成功返回 “ok”,失败返回其他信息  Object data;根据需求携带其他数据 |

ProductController

|  |
| --- |
| @RequestMapping("/update")  **public** SysResult productUpdate(Product product){  **try** {  productService.productUpdate(product);  **return** SysResult.*ok*();  } **catch** (Exception e) {  e.printStackTrace();  //有异常，返回除200的数，其他信息  **return** SysResult.*build*(201, e.getMessage(), **null**);  }  } |

ProductService

|  |
| --- |
| **public** **void** productUpdate(Product product) {  productMapper.productUpdate(product);  } |

ProductMapper

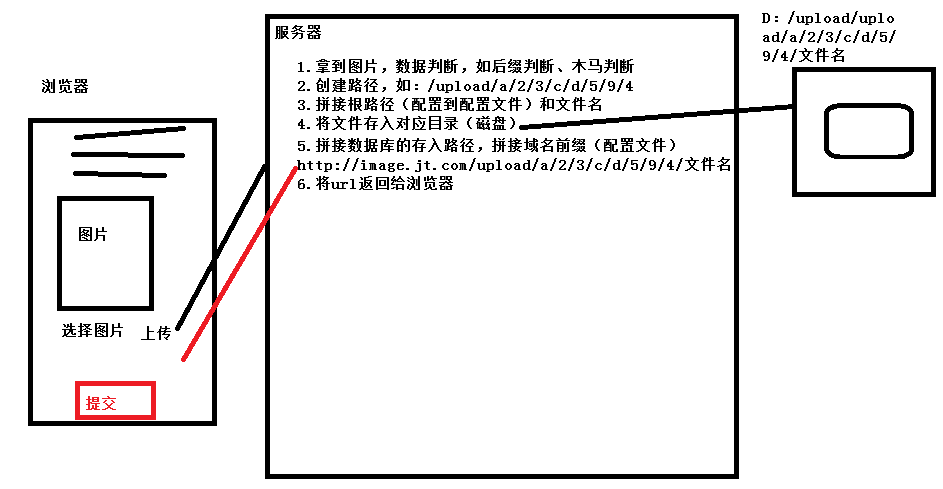
|  |
| --- |
| **void** productUpdate(Product product); |

ProductMapper.xml

|  |
| --- |
| <update id=*"productUpdate"* parameterType=*"Product"*>  update t\_product set  product\_name=#{productName},  product\_price=#{productPrice},  product\_category=#{productCategory},  product\_imgurl=#{productImgurl},  product\_num=#{productNum},  product\_description=#{productDescription}  where product\_id=#{productId};  </update> |

## 6.12.图片上传

### 6.12.1.上传流程



### 6.12.2.搭建图片工程

1.quickstart（easymall-microservice-pic）

2.pom.xml引入：继承easymall-parent，依赖工具项目easymall-resources

3.配置文件application.properties

server.port=10002

spring.application.name=picservice

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

pic.pathDirPrefix=D://upload/

pic.urlPrefix=http://image.jt.com/

4.启动类

|  |
| --- |
| **package** cn.tedu;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;  @SpringBootApplication  @EnableEurekaClient  **public** **class** StarterPicCenter {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterPicCenter.**class**, args);  }  } |

### 6.12.3.图片上传功能实现

|  |  |
| --- | --- |
| Js请求地址 | <http://www.easymall.com/uploadImg> |
| 后台接收地址 | /pic/upload |
| 请求方式 | Post |
| 请求参数 | MultipartFile pic |
| 返回数据 | PicUploadResult对象的json,结构是:  String url 生成的访问图片路径  Integer error 上传成功和失败的标志 0表示无错误 1表示有错误 |

PicController

|  |
| --- |
| **package** cn.tedu.pic.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  **import** org.springframework.web.multipart.MultipartFile;  **import** com.jt.common.vo.PicUploadResult;  @RestController  **public** **class** PicController {  @Autowired  **private** PicService picService;  @RequestMapping("/pic/upload")  **public** PicUploadResult picUpload(MultipartFile pic){  PicUploadResult result = picService.picUpload(pic);  **return** result;  }  } |

PicService

|  |
| --- |
| **package** cn.tedu.pic.service;  **import** java.awt.image.BufferedImage;  **import** java.io.File;  **import** java.io.IOException;  **import** javax.imageio.ImageIO;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.stereotype.Service;  **import** org.springframework.web.multipart.MultipartFile;  **import** com.jt.common.utils.UUIDUtil;  **import** com.jt.common.utils.UploadUtil;  **import** com.jt.common.vo.PicUploadResult;  @Service  **public** **class** PicService {  //获取配置文件的路径资源  @Value("${pic.pathDirPrefix}")  **private** String pathDirPrefix;  @Value("${pic.urlPrefix}")  **private** String urlPrefix;  **public** PicUploadResult picUpload(MultipartFile pic) {  PicUploadResult result = **new** PicUploadResult();  //1.图片合法判断  //--后缀判断，获取图片名称  String originalFilename = pic.getOriginalFilename();  //截取后缀，判断是否以.png\.jpg\.gif结尾  String extName = originalFilename.substring(  originalFilename.lastIndexOf("."));  **boolean** matches = extName.matches(".(jpg|png|gif|JPG|PNG|GIF)$");  //如果后缀不匹配，则直接return，只有匹配成功，才能继续  **if**(!matches){  result.setError(1);//代表错误  **return** result;  }  //木马判断  **try** {  BufferedImage bufferImage = ImageIO.*read*(pic.getInputStream());  bufferImage.getWidth();  bufferImage.getHeight();  } **catch** (IOException e) {  //不满足图片属性  e.printStackTrace();  result.setError(1);//代表错误  **return** result;  }  //2.创建路径  //引入UploadUtil工具类，生成图片路径，如：/upload/a/b/c/1/2/3/4/5/  String dir = "/"+UploadUtil.*getUploadPath*(  originalFilename, "upload")+"/";  //3.拼接根路径和文件名(配置文件读取的根路径)  //相当于D://upload/upload/a/b/c/1/2/3/4/5/  String pathDir = pathDirPrefix+dir;  //磁盘创建目录（如果没有）  File file = **new** File(pathDir);  **if**(!file.exists()){  file.mkdirs();  }  //4.将文件存入对应的目录  //避免文件名重复（加uuid）  String filename = UUIDUtil.*getUUID*()+originalFilename;  //流输出文件  **try** {  pic.transferTo(**new** File(pathDir+filename));  } **catch** (Exception e) {  //存储失败，返回  e.printStackTrace();  result.setError(1);//代表错误  **return** result;  }  //5.拼接域名  String imageUrl = urlPrefix+dir+filename;  //6.将imageUrl封装到返回对象中  result.setUrl(imageUrl);  **return** result;  }  } |

### 6.12.4.前端页面整合

zuul路由规则：

zuul.routes.zuul-pic.path=/zuul-pic/\*\*

zuul.routes.zuul-pic.serviceId=picservice

nginx转发规则

location /uploadImg{

proxy\_pass http://127.0.0.1:9005/zuul-pic/pic/upload;

add\_header 'Access-Control-Allow-Credentials' 'true';

add\_header 'Access-Control-Allow-Origin' '\*';

}

### 6.12.5.测试

启动（重启）eureka、zuul、product、pic、nginx

访问www.easymall.com后台新增商品

## 6.13.用户系统

### 6.13.1.系统搭建

1.quickstart（easymall-microservice-user）

2.pom.xml继承easymall-parent，依赖repository、resources

3.application.properties

端口、eureka、datasource、mybatis

server.port=10003

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

spring.datasource.url=jdbc:mysql:///easydb

spring.datasource.username=root

spring.datasource.password=root

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

spring.datasource.initialSize=5

spring.datasource.maxActive=50

spring.datasource.maxIdle=10

spring.datasource.minIdle=5

mybatis.typeAliasesPackage=com.jt.common.pojo

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

mybatis.configuration.mapUnderscoreToCamelCase=true

mybatis.configuration.cacheEnables=false

spring.application.name=userservice

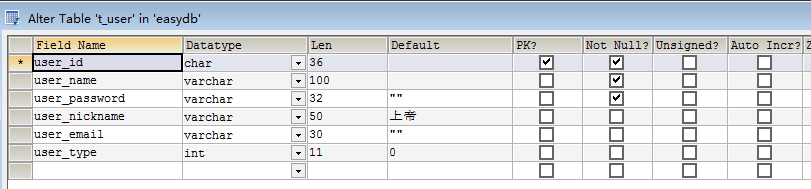
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

4.启动类

|  |
| --- |
| **package** cn.tedu;  **import** org.mybatis.spring.annotation.MapperScan;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;  @SpringBootApplication  @EnableEurekaClient  @MapperScan("cn.tedu.user.mapper")  **public** **class** StarterUserCenter {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterUserCenter.**class**, args);  }  } |

### 6.13.2.功能开发-用户名校验

**数据库t\_user表**



user\_id：用户id，uuid

user\_name：用户名

user\_password：密码（加密）

user\_nickname：昵称

user\_email：email

user\_type：用户类型，默认0，类似level

**用户名校验开发逻辑**

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/user/checkUserName |
| 后台接收 | /user/manage/checkUserName |
| 请求方式 | Post |
| 请求参数 | String userName |
| 返回数据 | 返回SysResult对象的json,其结构:  Integer status; 200表示用户名可用,其他表示不可用  String msg;成功返回 “ok”,失败返回其他信息  Object data;根据需求携带其他数据 |

UserController

|  |
| --- |
| **package** cn.tedu.user.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  **import** com.jt.common.vo.SysResult;  @RestController  @RequestMapping("/user/manage/")  **public** **class** UserController {  @Autowired  **private** UserService userService;  @RequestMapping("/checkUserName")  **public** SysResult checkUsername(String userName){  Integer exist = userService.checkUsername(userName);  **if**(exist == 0){//数据库无此用户名，用户名可以使用  **return** SysResult.*ok*();  }**else**{//用户名已存在  **return** SysResult.*build*(201,"用户名已存在", **null**);  }  }  } |

UserService

|  |
| --- |
| **package** cn.tedu.user.service;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  @Service  **public** **class** UserService {  @Autowired  **private** UserMapper userMapper;  **public** Integer checkUsername(String userName) {  **return** userMapper.queryUsername(userName);  }  } |

UserMapper

|  |
| --- |
| **package** cn.tedu.user.mapper;  **public** **interface** UserMapper {  Integer queryUsername(String userName);  } |

UserMapper.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.tedu.user.mapper.UserMapper"*>  <select id=*"queryUsername"* resultType=*"int"*>  select count(\*) from t\_user where user\_name=#{userName};  </select>  </mapper> |

**页面整合**

zuul网关

zuul.routes.zuul-user.path=/zuul-user/\*\*

zuul.routes.zuul-user.serviceId=userservice

nginx整合

location /user {

proxy\_pass http://127.0.0.1:9005/zuul-user/user/manage;

add\_header 'Access-Control-Allow-Credentials' 'true';

add\_header 'Access-Control-Allow-Origin' '\*';

}

**测试**

启动eureka、zuul、nginx、user

### 6.13.3.注册功能

**功能实现**

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/user/save |
| 后台接收 | /user/manage/save |
| 请求方式 | Post |
| 请求参数 | User user 缺少用户id,password需要加密 |
| 返回数据 | 返回SysResult对象的json,其结构:  Integer status; 200表示成功,其他表示失败  String msg;成功返回 “ok”,失败返回其他信息  Object data;根据需求携带其他数据 |

UserController

|  |
| --- |
| @RequestMapping("/save")  **public** SysResult userSave(User user){  **try** {  userService.userSave(user);  **return** SysResult.*ok*();  } **catch** (Exception e) {  e.printStackTrace();  **return** SysResult.*build*(201,e.getMessage(), **null**);  }  } |

UserService

|  |
| --- |
| **public** **void** userSave(User user) {  //生成userId-uuid  user.setUserId(UUIDUtil.*getUUID*());  //对密码加密  user.setUserPassword(MD5Util.*md5*(user.getUserPassword()));  //交给数据库存  userMapper.addUser(user);  } |

UserMapper

|  |
| --- |
| **void** addUser(User user); |

UserMapper.xml

|  |
| --- |
| <insert id=*"addUser"* parameterType=*"User"*>  insert into t\_user(  user\_id,  user\_name,  user\_password,  user\_nickname,  user\_email,  user\_type) values (  #{userId},  #{userName},  #{userPassword},  #{userNickname},  #{userEmail},  #{userType});  </insert> |

**页面整合（不用动）**

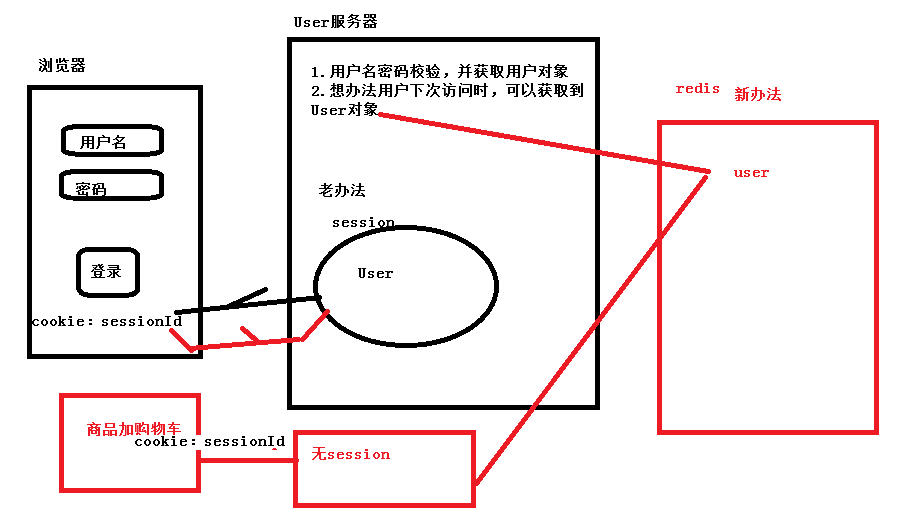
由于第一次整合，后续user项目无需整合

**测试**

直接启动测试

### 6.13.4.登录功能

**登录逻辑**



学完redis才能实现，暂停代码，进入redis