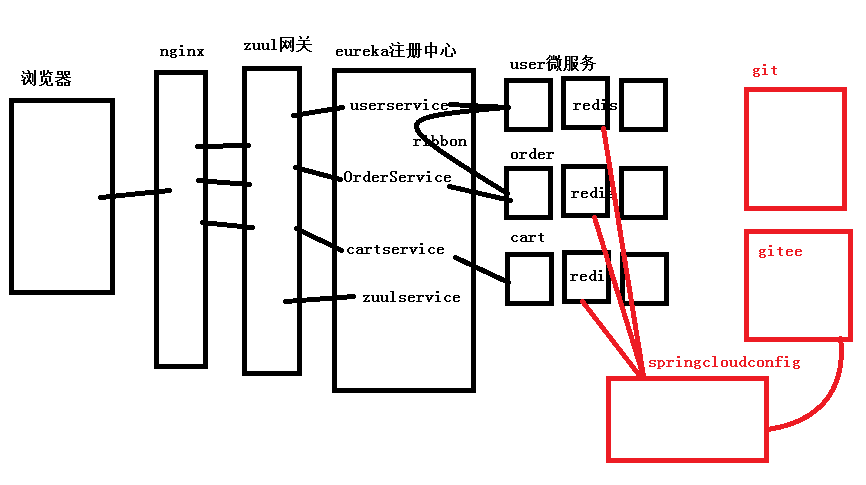
# 8.SpringCloudConfig

## 8.1.SpringCloudConfig介绍

springcloudconfig是springcloud提供的一个共享配置的组件，可以实现从远程库读取配置文件，交给连接访问，根据读取文件的结构和规则，客户端可以选择使用那些配置。

**SpringCloudConfig架构**



## 8.2.springcloudconfig搭建

### 8.2.1.服务端系统搭建

1.quickstart(config-server)

2.pom继承springboot，依赖springcloud，导入config-server的依赖

<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-config-server</artifactId>

</dependency>

3.配置文件（指向gitee）



端口号

server.port=11000

指向自己的gitee仓库地址

spring.cloud.config.server.git.uri=https://gitee.com/zhangqi23/emconfig

一个仓库中可以有很多个文件夹，可以通过searchPaths指定访问哪个文件夹

spring.cloud.cinfig.server.git.searchPaths=emtest

指定分支为master（如果不指定，默认master）

spring.cloud.config.label=master

如果gitee是私有的，需要提供用户名密码

#spring.cloud.config.server.git.username=zhangsan

#spring.cloud.config.server.git.password=123456

4.启动类

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

### 8.2.2.测试

1.读取配置文件规则

/{application}/{profile}/[{label}]

/[{label}]/{application}-{profile}.properties\.yml

例如：demo01-test.properites

application：就是文件名的前缀，这里就是demo01

profile：是文件名的后缀，这里就是test

label：是分支名称，不加默认master

浏览器测试访问：

<http://localhost:11000/demo01/test>

<http://localhost:11000/demo01-test.properties>

2.如果前后缀非常复杂

如：demo01-version01-test01.properties

localhost:11000/demo01/version01-test01

或者localhost:11000/demo01-version01/test01

### 8.2.3.客户端系统搭建

1.quickstart（config-client）

2.pom依赖

springboot、springcloud、config-client

<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-config</artifactId>

</dependency>

<dependency>

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

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

</dependency>

3.配置文件

配置文件的名称：bootstrap.properties

作用：连接config-server的配置信息需要在这里配置

和application.properites区别：

1.可以共存，也可以只有bootstrap.properties

2.相同内容不能在两个文件重复配置

端口

server.port=11100

注册到eureka的服务名称，同时，如果不指定，也可以作为读取配置文件的前缀:demo01\demo02（application）

spring.application.name=configclient

指定分支

spring.cloud.config.label=master

配置中心的uri（目前写死了）

spring.cloud.config.uri=http://localhost:11000

读取配置文件的前缀，如果不指定，由前面的application配置决定，只能指定一个

spring.cloud.config.name=demo01

读取配置文件的后缀，可以多个

spring.cloud.config.profile=test,test02

4.启动类

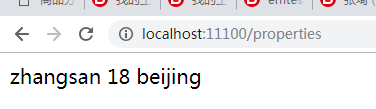
|  |
| --- |
| **package** cn.tedu;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  **public** **class** StarterConfigClient {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterConfigClient.**class**,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

5.测试

在启动类添加Controller的测试内容

|  |
| --- |
| **package** cn.tedu;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  @SpringBootApplication  @RestController  **public** **class** StarterConfigClient {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterConfigClient.**class**, args);  }  @Value("${name}")  **private** String name;  @Value("${age}")  **private** String age;  @Value("${location}")  **private** String location;  @RequestMapping("properties")  **public** String getProperties(){  **return** name+" "+age+" "+location;  }  } |

浏览器测试



gitee:





### 8.2.4.高可用的服务中心

问题：配置的服务中心只有一个，万一挂掉，则无法读取配置信息

解决方式：将配置中心注册到eureka注册中，作为一个微服务集群

客户端需要从eureka获取配置中心的节点信息

**（1）搭建高可用的配置中心（config-server）**

1.pom文件引入eureka资源

<dependency>

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

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

</dependency>

2.配置文件

加入eureka的配置文件

spring.application.name=configserver

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

3.启动类

@EnableEurekaClient

**（2）修改客户端**

1.pom依赖eureka

<dependency>

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

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

</dependency>

2.配置文件

注释掉之前的uri内容

#spring.cloud.config.uri=http://localhost:11000

注册到eureka

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

开启发现功能

spring.cloud.config.discovery.enabled=true

拿到配置中心在eureka的注册信息，底层通过ribbon+restTemplate获取的

spring.cloud.config.discovery.serviceId=configserver

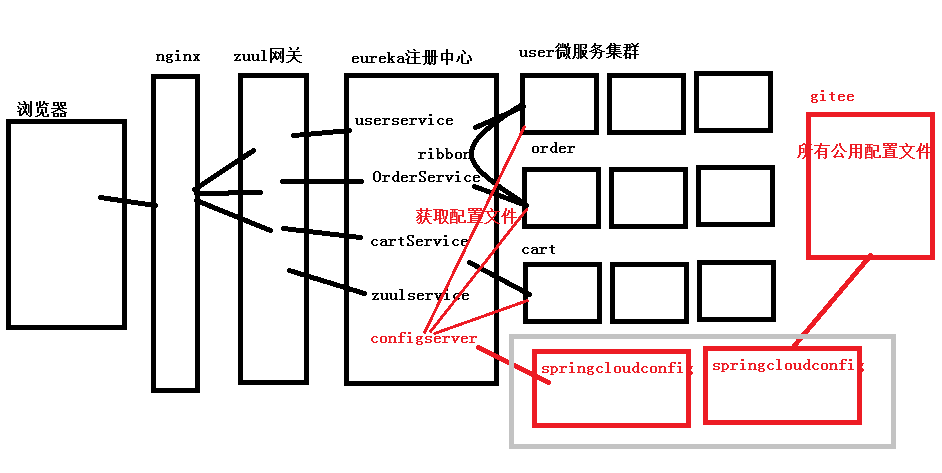
3.启动类

添加eureka注解

@EnableEurekaClient

## 8.3.easymall引入springcloudconfig

### 8.3.1.架构



### 8.3.2.将用户系统整合springcloudconfig

前提：gitee上准备配置文件、configserver使用已有的

gitee库：easymall，文件夹：emconfig，配置文件：

datasource.properties

**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

redis.properties

**spring.redis.cluster.nodes**=192.168.171.138:8000,192.168.171.138:8001

**spring.redis.cluster.maxTotal**=100

**spring.redis.cluster.maxIdle**=10

**spring.redis.cluster.minIdle**=5

1.pom引入springcloudconfig依赖

<dependency>

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

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

</dependency>

2.配置文件

新建bootstrap.properties

server.port=10003

spring.application.name=userservice

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

spring.cloud.config.label=master

spring.cloud.config.name=redis,datasource

spring.cloud.config.discovery.enabled=true

spring.cloud.config.discovery.serviceId=configserver

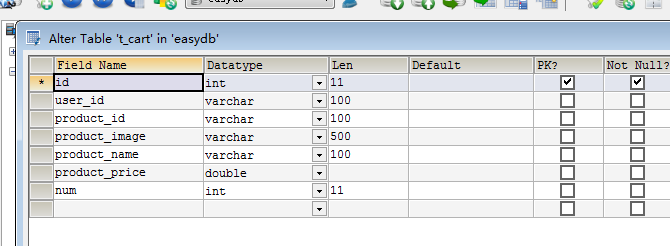
3.测试

启动nginx、eureka、zuul、configserver、user系统、RedisCluster

## 8.4.购物车系统

### 8.4.1.数据库表格

t\_cart表格



id：主键自增

user\_id:用户id，复合主键

product\_id:商品id，复合主键

product\_image:商品图片url（冗余字段，反范式设计）

product\_name:商品名称（冗余字段，反范式设计）

product\_price:商品价格（冗余字段，反范式设计）

num:购买数量

### 8.4.2.搭建购物车系统

1.quickstart(easymall-microservice-cart)

2.pom继承easymall-parent，依赖repository、resources、redis、[config]

继承一个，依赖三个，通过视图完成

3.配置文件application.properties\[bootstrap.properties]

|  |
| --- |
| server.port=10004  #RedisCluster  spring.redis.cluster.nodes=192.168.171.138:8000,192.168.171.138:8001  spring.redis.cluster.maxTotal=100  spring.redis.cluster.maxIdle=10  spring.redis.cluster.minIdle=5  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=cartservice  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.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.cart.mapper")  **public** **class** StarterCartCenter {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterCartCenter.**class**, args);  }  @Bean  @LoadBalanced  **public** RestTemplate initCartRestTemplate(){  **return** **new** RestTemplate();  }  } |

### 8.4.3.购物车查询

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/cart/query?userId=\*\* |
| 后台接收 | /cart/manage/query?userId=\*\* |
| 请求方式 | Get |
| 请求参数 | String userId |
| 返回数据 | 返回List<Cart>数据 |

CartController

|  |
| --- |
| **package** cn.tedu.cart.controller;  **import** java.util.List;  **import** org.apache.commons.lang.StringUtils;  **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.pojo.Cart;  @RestController  @RequestMapping("/cart/manage/")  **public** **class** CartController {  @Autowired  **private** CartService cartService;  @RequestMapping("/query")  **public** List<Cart> queryCart(String userId){  //判断userId是否为空  **if**(StringUtils.*isNotEmpty*(userId)){//不空  List<Cart> list = cartService.queryCart(userId);  **return** list;  }**else**{  **return** **null**;  }  }  } |

CartService

|  |
| --- |
| **package** cn.tedu.cart.service;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** com.jt.common.pojo.Cart;  **import** cn.tedu.cart.mapper.CartMapper;  @Service  **public** **class** CartService {  @Autowired  **private** CartMapper cartMapper;  **public** List<Cart> queryCart(String userId) {  **return** cartMapper.queryCart(userId);  }  } |

CartMapper

|  |
| --- |
| **package** cn.tedu.cart.mapper;  **import** java.util.List;  **import** com.jt.common.pojo.Cart;  **public** **interface** CartMapper {  List<Cart> queryCart(String userId);  } |

CartMapper.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.cart.mapper.CartMapper"*>  <select id=*"queryCart"* resultType=*"Cart"*>  select \* from t\_cart where user\_id=#{userId};  </select>  </mapper> |

### 8.4.4.页面整合

**（1）nginx**

location /cart {

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

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

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

}

**（2）zuul**

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

zuul.routes.zuul-cart.serviceId=cartservice

启动测试

### 8.4.5.新增功能

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/cart/save |
| 后台接收 | /cart/manage/save |
| 请求方式 | post |
| 请求参数 | Cart cart,缺少商品name,商品价钱,商品类型 |
| 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; |
| 备注 | 需要调用商品查询功能将其他三个字段补齐,并且需要逻辑判断新增购物车是更新num还是新增 |

注意：前端页面只传过来userId和productId和num，缺少其他数据，需要通过ribbon+restTemplate访问商品系统拿到数据

购物车添加注意：

如果购物车没有当前商品，直接新增，如果有，只增加购买数量

CartController

|  |
| --- |
| @RequestMapping("/save")  **public** SysResult cartSave(Cart cart){  **try** {  cartService.cartSave(cart);  **return** SysResult.*ok*();  } **catch** (Exception e) {  e.printStackTrace();  **return** SysResult.*build*(201,e.getMessage(), **null**);  }  } |

CartService

|  |
| --- |
| @Autowired  **private** RestTemplate restTemplate;  **public** **void** cartSave(Cart cart) {  //判断购物车是否已经存在当前商品  Cart exisCart = cartMapper.queryOne(cart);  **if**(exisCart!=**null**){//已存在  //修改购买数量，将原有数量和新数量相加  **int** num = exisCart.getNum()+cart.getNum();  //设置到老的对象中  exisCart.setNum(num);  //将对象交给mapper进行数量的修改  cartMapper.updateNum(exisCart);  }**else**{//不存在  //在加入购物车之前，补充缺失数据，通过restTemplate  Product product = restTemplate.getForObject(  "http://productservice/product/manage/item/"  +cart.getProductId(), Product.**class**);  //将获取的product信息设置到购物车cart对象中  cart.setProductImage(product.getProductImgurl());  cart.setProductPrice(product.getProductPrice());  cart.setProductName(product.getProductName());  cartMapper.saveCart(cart);  }  } |

CartMapper

|  |
| --- |
| Cart queryOne(Cart cart);  **void** updateNum(Cart exisCart);  **void** saveCart(Cart cart); |

CartMapper.xml

|  |
| --- |
| <select id=*"queryOne"* resultType=*"Cart"*>  select \* from t\_cart where user\_id=#{userId} and  product\_id=#{productId};  </select>  <update id=*"updateNum"* parameterType=*"Cart"*>  update t\_cart set num=#{num} where user\_id=#{userId} and  product\_id=#{productId};  </update>  <insert id=*"saveCart"* parameterType=*"Cart"*>  insert into t\_cart(  user\_id,product\_id,product\_image,product\_name,product\_price,num)  values(#{userId},  #{productId},  #{productImage},  #{productName},  #{productPrice},  #{num});  </insert> |

### 8.4.6.修改num

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/cart/update |
| 后台接收 | /cart/manage/update |
| 请求方式 | get |
| 请求参数 | Cart cart,具有三个属性userId,productid,num |
| 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; |

CartController

|  |
| --- |
| //更新购买数量  @RequestMapping("/update")  **public** SysResult updateNum(Cart cart){  **try** {  cartService.updateNum(cart);  **return** SysResult.*ok*();  } **catch** (Exception e) {  e.printStackTrace();  **return** SysResult.*build*(201,e.getMessage(), **null**);  }  } |

CartService

|  |
| --- |
| **public** **void** updateNum(Cart cart) {  cartMapper.updateNum(cart);  } |

其他无需改动

### 8.4.7.购物车删除

|  |  |
| --- | --- |
| js请求地址 | http://www.easymall.com/cart/delete |
| 后台接收 | /cart/manage/delete |
| 请求方式 | get |
| 请求参数 | Cart cart,具有两个属性userId,productid |
| 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; |

CartController

|  |
| --- |
| //删除商品  @RequestMapping("/delete")  **public** SysResult deleteCart(Cart cart){  **try** {  cartService.deleteCart(cart);  **return** SysResult.*ok*();  } **catch** (Exception e) {  e.printStackTrace();  **return** SysResult.*build*(201,e.getMessage(), **null**);  }  } |

CartService

|  |
| --- |
| **public** **void** deleteCart(Cart cart) {  cartMapper.deleteCart(cart);  } |

CartMapper

|  |
| --- |
| **void** deleteCart(Cart cart); |

CartMapper.xml

|  |
| --- |
| <delete id=*"deleteCart"* parameterType=*"Cart"*>  delete from t\_cart where user\_id=#{userId} and  product\_id=#{productId};  </delete> |

测试

### 8.4.8.游客购物车

数据存储方式：

一：数据库，并非重要数据，没必要存数据库，浪费资源

二：redis，游客加入购物车时，生成key，交给cookie，下次检查cookie内容并通过cookie的值去redis中查询

三：cookie保存游客购物车的所有商品内容