**SpringCloudConfig**

**1.概括**

**1.1介绍**

Spring cloud config是springcloud为共享配置的使用提供的一个组件,可以实现从远程库读取配置文件,交给连接访问他的所有客户端使用,根据读取文件的结构和规则,客户端可以选择使用哪些配置文件.

**1.2加入config配置中心的微服务集群结构**

图示

描述已自动生成

Git的私有资源是收费的，而gitee是免费的

**2.搭建一个config工程**

**2.1搭建配置中心**

* quickstart工程

**config-server**

* pom依赖

1. 基于springboot的parent

<parent>

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

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

<version>1.5.9.RELEASE</version>

</parent>

2）导入springcloud资源

<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>

3) config配置中心的依赖内容

<dependency>

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

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

</dependency>

* **properties指向gitee**

在config-server创建application.properties配置文件

手机屏幕截图

描述已自动生成

gitee上的仓库

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

在application.properties文件中配置:

图形用户界面, 文本

描述已自动生成

* 1. 仓库地址

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

如果要在同一个仓库下管理多个文件夹,可以在config配置中心指定serchPaths属性,例如下面的配置：

spring.cloud.config.server.git.searchPaths=/**xutest**

表示从仓库根目录读取一个叫**xutest**的文件夹中的内容

指定分支master(默认就是master)

spring.cloud.config.label=master

如果读取的资源是private还要提供2个参数 username password

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

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

* 启动类

**package** cn.edu.scnu;

**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);

}}

**2.2通过配置中心访问gitee资源的方式**

* **读取配置文件的结构**

通过配置中心读取,提供多个接口读取properties/yml

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

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

例如：demo01-test.properties

application:可以是一个微服务的名称,配置中心对应一个文件名称的前缀 demo01

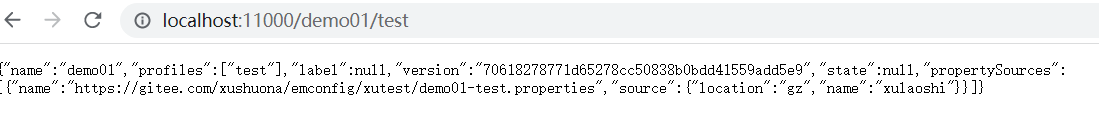
profile:是定义一个微服务可以访问的多个位置文件的后缀,test

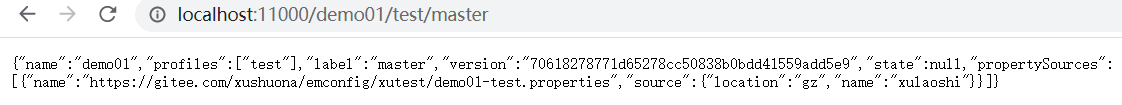
label:分支名称 master

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

* **测试访问(demo01-test.properties)**

localhost:11000/demo01/test/[master]





localhost:11000/demo01-test.properties

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

* **当文件前后缀复杂时也可以读取文件并且解析不同结构**

创建一个文件demo02-version1-test01.properties

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 文本

描述已自动生成

图形用户界面, 文本

描述已自动生成

访问过的文件都被缓存到：

C:\Users\LENOVO\AppData\Local\Temp\config-repo-4944820678932051570\xutest

图形用户界面, 文本, 应用程序

描述已自动生成

**2.3搭建配置中心的客户端工程**

* **quickstart**

config-client

* **pom.xml修改**
  + 基于springboot的parent

<parent>

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

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

<version>1.5.9.RELEASE</version>

</parent>

* + 引入cloud资源

<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>

* + 依赖config客户端资源

<dependency>

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

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

</dependency>

* + 依赖starter-web

当前工程启动一个web应用

<dependency>

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

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

</dependency>

* 启动类和普通springboot启动类一模一样

文本

描述已自动生成

* **配置文件名称**

properties或者yml配置文件名称不能使用application,用bootstrap , bootstrap可以和application配置文件共存,必须注意内容不能重复,可以让bootstrap专门管理连接访问配置中心的各种内容,application配置其他内容;一旦冲突，bootstrap虽然会覆盖application，但不建议冲突。

创建bootstrap.properties配置文件

图片包含 文本

描述已自动生成

指定application读取名称，这里指定为configclient表示读取gitee配置中心中以configclient为前缀的配置文件。目前gitee中没有该前缀的配置文件。

spring.application.name=configclient

如果不单独指定application的读取名称,服务名称就是读取的配置文件的application(demo01,demo02)

下面的master可一个不给定，因为接口里面的master可加也可以不加

spring.cloud.config.label=master

连接配置中心的uri(不是高可用,只能连接访问一个具体的配置中心)

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

可以不使用服务名称作为配置文件的application前缀直接配置想要的前缀

spring.cloud.config.name=demo01

spring.cloud.config.profile=test,test2

图形用户界面, 文本, 应用程序

描述已自动生成

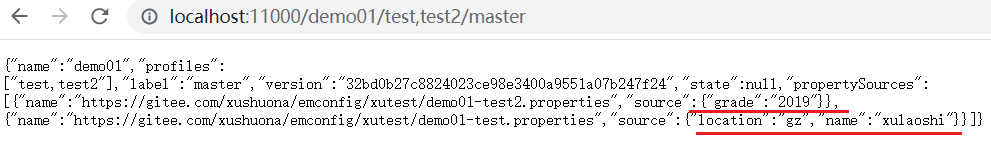
当前配置的客户端,读取逻辑

启动时连接config配置中心,根据客户端配置内容,读取demo01-test.properties demo01-test2.properties

调用接口localhost:**11000**/demo01/test,test2/master

图形用户界面, 文本, 应用程序

描述已自动生成



* **验证config客户端能否读取到想要的key-value数据**

@Value("${location}")

@Value("${name}")

在启动类中添加Controller，进行测试

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

**3.搭建高可用的配置中心**

实现逻辑:配置中心注册在eureka注册中心,形成服务的提供者角色,这个服务只做为其他服务提供高可用的访问获取git中properties的配置文件资源;

配置中心的客户端工程需要通过从eureka抓取配置中心服务的列表高可用高并发的访问配置中心服务;

**3.1修改配置中心的配置文件和pom**

配置中心具备eureka的客户端角色

* 添加pom的eureka客户端依赖资源

<dependency>

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

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

</dependency>

* properties中添加注册的eureka中心地址

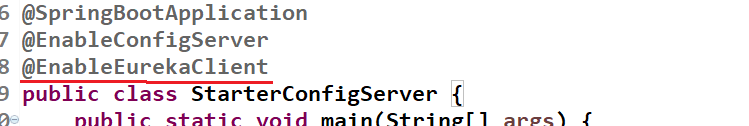
#eureka config server

spring.application.name=configserver

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

* 启动类添加注解形成eureka客户端进程

@EnableEurekaClient



**3.2修改配置客户端工程**

* eureka的客户端角色

添加eureka的依赖

<dependency>

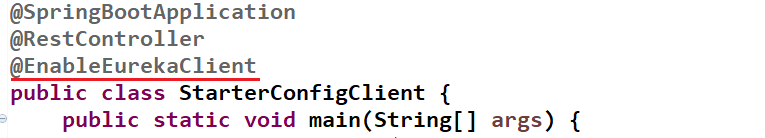
<groupId>org.springframework.cloud</groupId>

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

</dependency>

* 启动类中添加eureka客户端注解

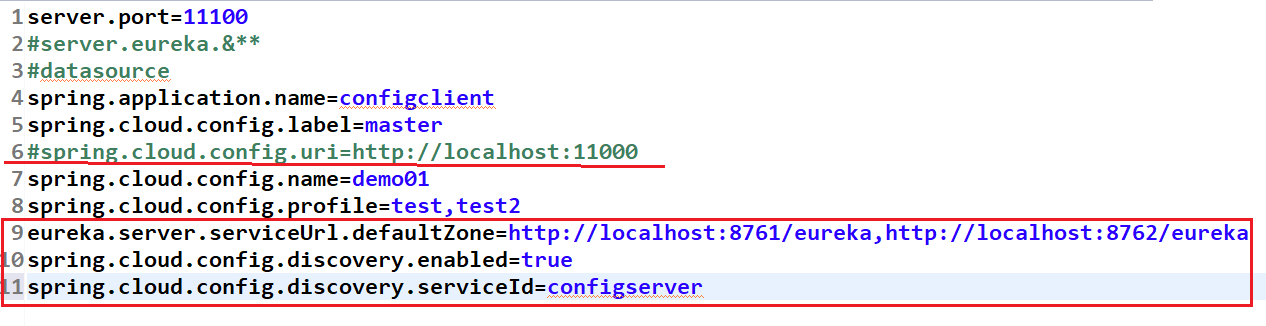
@EnableEurekaClient



* properties修改内容

修改bootstrap.properties

注释掉以单个实例访问的uri配置内容



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

spring.cloud.config.discovery.enabled=true

spring.cloud.config.discovery.serviceId=configserver

上面的配置表示开启通过服务发现的方式找到配置中心的服务名称,进行调用底层restTemplate+ribbon <http://配置中心服务名称/application/profile>

启动：

图形用户界面, 表格

中度可信度描述已自动生成

为了避免缓存，可以清除缓存，也可以在另外一个浏览器访问（火狐）

图形用户界面, 文本, 应用程序

描述已自动生成

**4.微服务集群结构的变化**

围绕着eureka注册中心.服务的调用,服务间的调用展开的所有逻辑

图示

描述已自动生成

**5.将用户系统,商品系统整理到config配置中心作为其客户端**

**5.1在git准备好配置文件**

* 利用profile切分

common-datasource.properites

common-redis-sentinel.properties

common-redis-cluster.properties

userservice-dev.properties

* 利用application切分

datasource-dev.properties

redis-sentinelt-dev.properties

redis-cluster-dev.properties

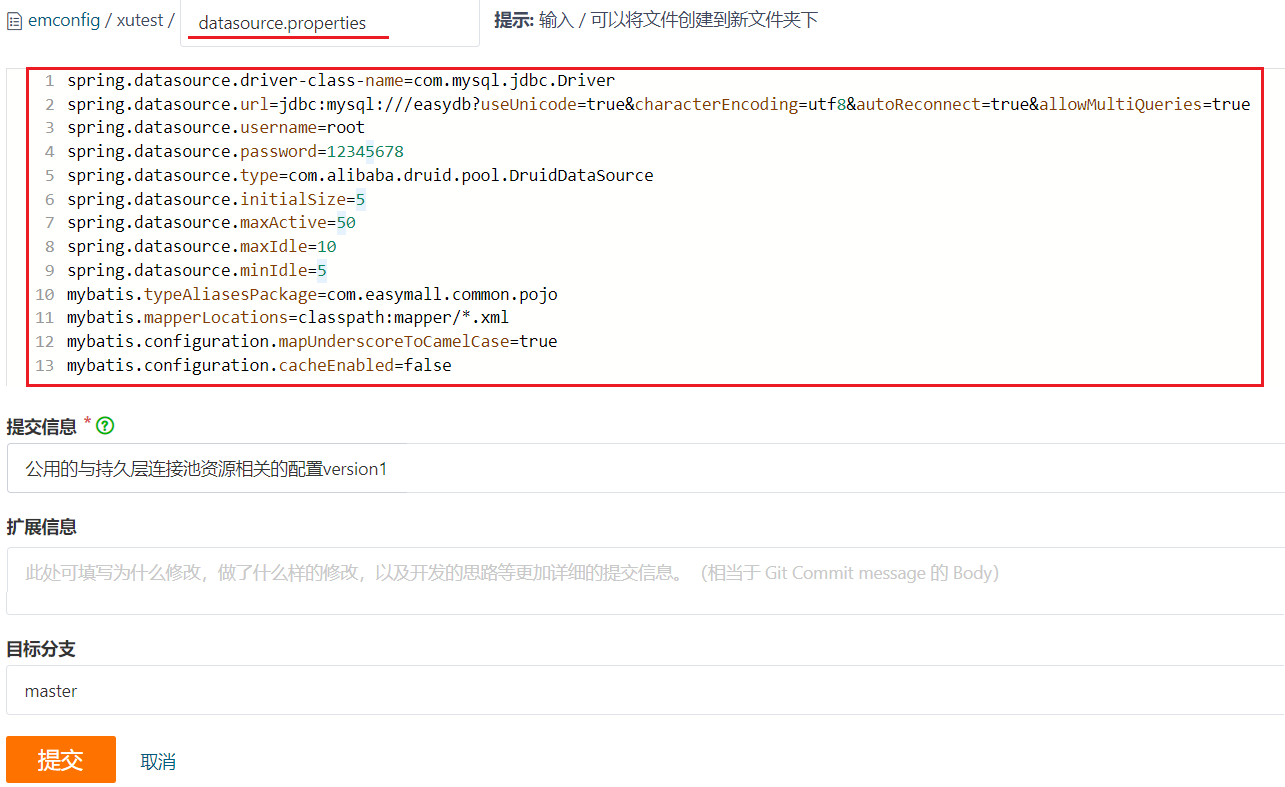
* 空profile

datasource.properties

redis.properties

* 由于我们系统配置文件很简单，我们使用空profile，到gitee网站去创建配置文件datasource.properties和redis.properties

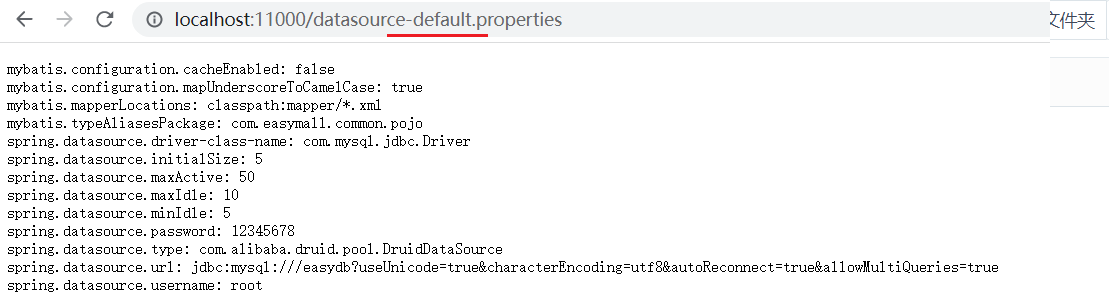


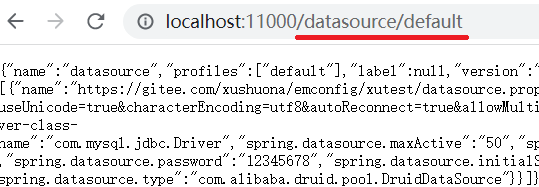




* 启动配置中心启动eureka

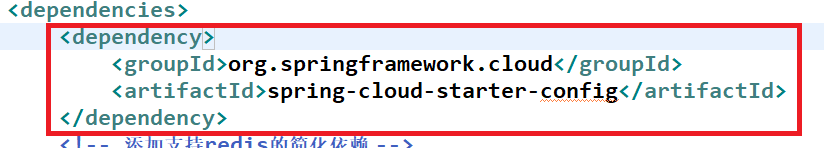
看看配置文件能否正常读取出来：默认要加上**-default**





**5.2修改商品系统的内容**

* + pom添加config客户端依赖



<dependency>

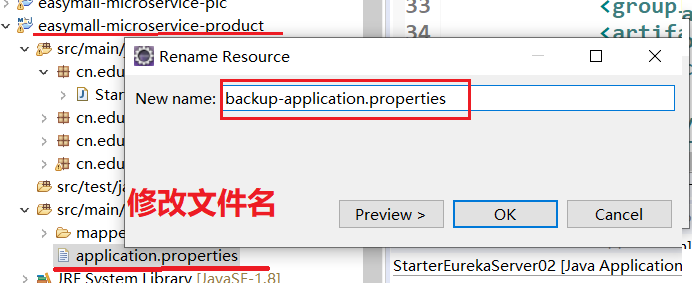
<groupId>org.springframework.cloud</groupId>

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

</dependency>

* properties文件-->bootstrap.properties

先备份application.properties,再创建bootstrap.properties



文本

描述已自动生成

创建bootstrap.properties:



server.port=10002

spring.application.name=productservice

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

#config client

spring.cloud.config.label=master

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

spring.cloud.config.name=redis,datasource

#eureka client

spring.cloud.config.discovery.enabled=true

spring.cloud.config.discovery.serviceId=configserver

* 测试修改后的商品系统

启动：分别启动EurekaServer01/ EurekaServer02/zuul/product/config-server/

图形用户界面, 文本, 应用程序

描述已自动生成

查看商品，能正常访问

图形用户界面, 网站

描述已自动生成

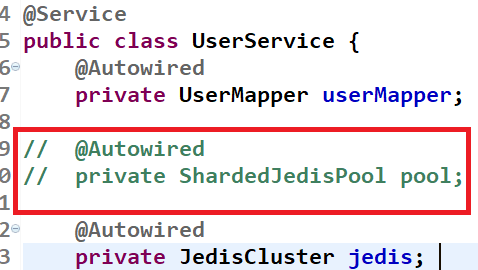
**5.3 修改user用户系统(作业)**

作业：按照前面修改商品系统内容的步骤修改User系统

**注意：**除了按照上面修改之外，还要将ShardedJedisPoolConfig.java文件备份，然后删除，否则会因为读取不到pool的相关配置信息而报错。



UserService也有修改，注释掉ShardedJedisPool：



修改后登录：

图形用户界面, 文本

描述已自动生成