# 案例1：Config分布式配置构建

发的

1）登录GitHub，创建项目demo-config

2）上传yml配置文件，命名规则如下：

{application}-{proﬁle}.yml 或者 {application}-{proﬁle}.properties

其中，application为应用名称，proﬁle指的是环境（用于区分开发环境，测试环境、生产环境等）

示例：demo-service-page-dev.yml、demo-service-page-test.yml、demo-service-page-prod.yml

3）构建Config Server统一配置中心

新建SpringBoot工程，引入依赖坐标（需要注册自己到Eureka）

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
   <parent>  
       <artifactId>demo-parent</artifactId>  
       <groupId>com.demo</groupId>  
       <version>1.0-SNAPSHOT</version>  
   </parent>  
   <modelVersion>4.0.0</modelVersion>  
​  
   <artifactId>demo-cloud-config</artifactId>  
​  
   <dependencies>  
       <!--eureka client 客户端依赖引入-->  
       <dependency>  
           <groupId>org.springframework.cloud</groupId>  
           <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
       </dependency>  
       <!--config配置中心服务端-->  
       <dependency>  
           <groupId>org.springframework.cloud</groupId>  
           <artifactId>spring-cloud-config-server</artifactId>  
       </dependency>  
   </dependencies>  
​  
</project>

配置启动类，使用注解@EnableConfigServer开启配置中心服务器功能：

package com.demo.config;  
​  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
import org.springframework.cloud.config.server.EnableConfigServer;  
​  
import java.io.ObjectInputFilter;  
​  
@SpringBootApplication  
@EnableConfigServer // 开启配置服务器功能  
@EnableDiscoveryClient  
public class ConfigApplication {  
   public static void main(String[] args) {  
       SpringApplication.run(ConfigApplication.class, args);  
  }  
}  
​

application.yml配置：

server:  
port: 9400  
#注册到Eureka服务中心  
eureka:  
client:  
  service-url:  
    defaultZone: http://demoCloudEurekaServerA:9200/eureka,http://demoCloudEurekaServerB:9201/eureka  
instance:  
  prefer-ip-address: true  
  instance-id: ${spring.cloud.client.ip-address}:${spring.application.name}:${server.port}:@project.version@  
spring:  
application:  
  name: demo-service-config  
cloud:  
  config:  
    server:  
      git:  
        uri: https://github.com/jetwu-do/demo-config.git  #配置git服务地址  
        username: gitwu #配置git用户名  
        password: 123456 #配置git密码  
        search-paths:  
          - demo-config  
     # 读取分支  
    label: master  
​

测试：

<http://127.0.0.1:9600/master/application-dev.yml>

master：分支名称

application-dev.yml：文件名称

4）构建Client客户端（在已有页面静态化微服务基础上）

案例实现：在demo-service-page微服务中动态获取config server的配置信息

已有工程中添加依赖坐标

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-config-client</artifactId>  
</dependency>

application.yml修改为bootstrap.yml配置文件

bootstrap.yml是系统级别的，优先级比application.yml高，应用启动时会检查这个配置文件，在这个配置文件中指定配置中心的服务地址，会自动拉取所有应用配置并且启用。

（主要是把与统一配置中心连接的配置信息放到bootstrap.yml）

注意：需要统一读取的配置信息，从配置中心获取

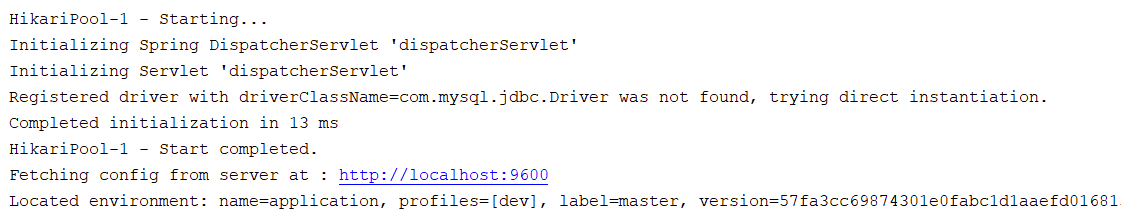
bootstrap.yml（部分）

server:  
port: 9100   # 后期该微服务多实例，端口从9100递增（10个以内）  
Spring:  
application:  
  name: demo-service-page  
datasource:  
  driver-class-name: com.mysql.jdbc.Driver  
  url: jdbc:mysql://localhost:3306/demodb?useUnicode=true&characterEncoding=utf8&serverTimezone=UTC  
  username: root  
  password: 123456  
cloud:  
  config:  
     #config客户端配置,和ConfigServer通信，并告知ConfigServer希望获取的配置信息在哪个文件中  
    name: application  
    profile: dev   #后缀名称  
    label: master  #分支名称  
    uri: http://localhost:9400   #ConfigServer配置中心地址  
​

ConfigController：

package com.demo.page.controller;  
​  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
​  
@RestController  
@RequestMapping("/config")  
public class ConfigClientController {  
​  
   @Value("${mysql.user}")  
   private String mysqlUser;  
​  
   @Value("${person.name}")  
   private String personName;  
​  
   @RequestMapping("/remote")  
   public String findRemoteConfig() {  
       return mysqlUser + " " + personName;  
  }  
​  
}

查看启动日志：



测试：

<http://127.0.0.1:9100/config/remote>

# 案例2：Config配置手动刷新

1）Client客户端添加依赖springboot-starter-actuator（已添加）

2）Client客户端bootstrap.yml中添加配置（暴露通信端点）

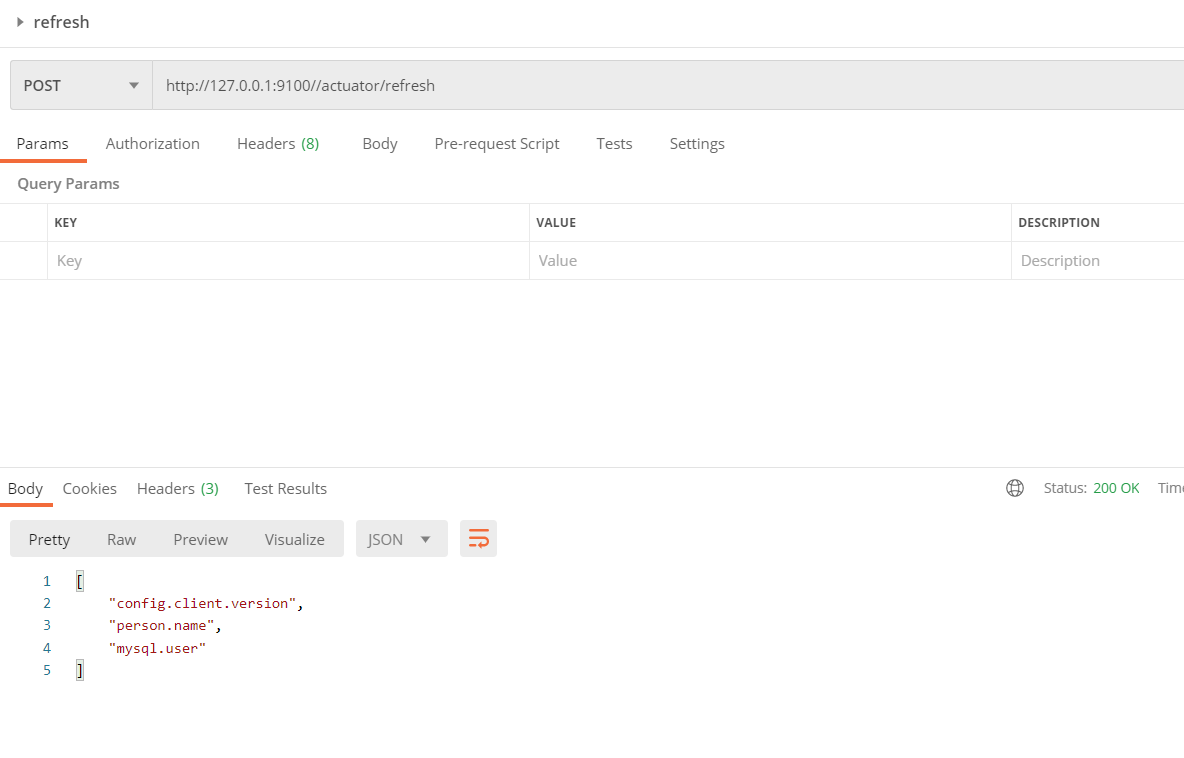
management:  
endpoints:  
web:  
exposure:  
include: refresh  
  
#也可以暴露所有的端口  
management:  
endpoints:  
  web:  
    exposure:  
      include: "\*"

3）Client客户端使用到配置信息的类上添加@RefreshScope

package com.demo.page.controller;  
​  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.cloud.context.config.annotation.RefreshScope;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
​  
@RestController  
@RequestMapping("/config")  
@RefreshScope  //手动刷新  
public class ConfigClientController {  
​  
   @Value("${mysql.user}")  
   private String mysqlUser;  
​  
   @Value("${person.name}")  
   private String personName;  
​  
   @RequestMapping("/remote")  
   public String findRemoteConfig() {  
       return mysqlUser + " " + personName;  
  }  
​  
}  
​

4）手动向Client客户端发起POST请求，

<http://localhost:9100/actuator/refresh>，刷新配置信息



# 案例3：Config配置自动刷新

1）Config Server服务端和客户端添加消息总线支持

<dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-starter-bus-amqp</artifactId>  
</dependency>

2）Config Server和客户端添加配置

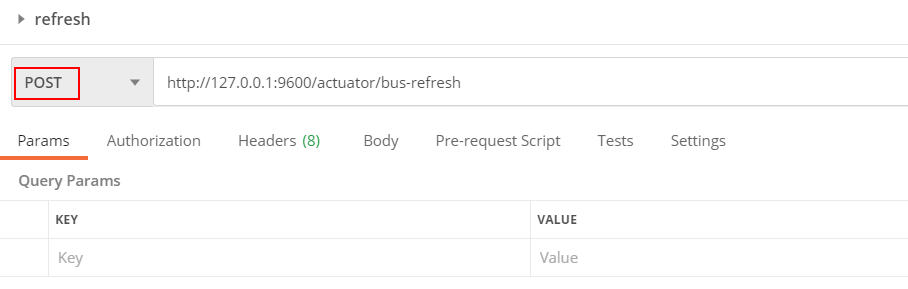
spring:  
rabbitmq:  
host: 192.169.200.128   
port: 5672   
username: guest   
password: guest

3）Config Server微服务暴露端口

management:  
endpoints:  
web:  
exposure:  
include: bus-refresh  
  
#建议暴露所有的端口  
management:  
endpoints:  
  web:  
    exposure:  
      include: "\*"

4）重启各个服务，更改配置之后，向配置中心服务端发送post请求，各个客户端配置即可自动刷新

<http://127.0.0.1:9400/actuator/bus-refresh>



5）Config Client测试

http://localhost:9100/config/remote

在广播模式下实现了一次请求，处处更新，如果我只想定向更新呢？

在发起刷新请求的时候http://localhost:9006/actuator/bus-refresh/demo-service-page:9100

即为最后面跟上要定向刷新的实例的 服务名:端口号即可