使用统一配置中心的理由

1 方便维护

2 配置内容安全与权限

3 更新配置项目需要重启

RabbitMq消息队列（实现配置文件的实时更新）

原理：

Config-server

订单服务

Config-client

商品服务

Config-client

本地git

远端git

Configserver从远端git上将配置拉取下来，放到本地git上，同时如果远端git不能拉取配置了，就从本地git上加载。然后不同的服务就可以从config-server上拉取配置了。

为了保证Server端的高可用，可以多启动几个server端，因为寻找可用是通过名字寻找，所以在同一服务名下有多个实例，一个挂掉还可以有其他的替代。就可以保证高可用性。

**Server端**

本身也是一个微服务，需要注册到Eureka上。

使用方式：加上这个依赖

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

然后在启动类上加上注解 @EnableConfigServer

Server需要用bootstrap.yml，这个配置文件会在启动项目之前加载。

配置项demo

**spring:  
 application:  
 name:** config  
 **cloud:  
 config:  
 server:  
 git:  
 basedir:   
 uri:** https://github.com/836219171/config-repo  
 **username:** 836219171  
 **password:** qinliang110

在git上写好配置后可以用这个路径测试一下配置文件格式对不对

http://localhost:8090/order-a.yml

/{name}-{profiles}.yml 默认使用master分支

/{label}/{name}-{profiles}.yml

name:服务名

profiles:环境

label:分支(branch)

**Client端：**

Client端由于配置文件要先于启动类启动拉取配置文件，所以要用bootstrap.yml文件.

Client端的使用只需要加上依赖，不需要在启动类加注解

按图配置好相应文件即可

**spring:  
 application:  
 name:** order  
 **cloud:  
 config:  
 discovery:  
 enabled:** true  
 **service-id:** CONFIG  
 **profile:** dev

远端配置文件上order.yml上的会和order-x.yml上合并，所以把通用配置写在order.yml上比较合理。

**实时更新配置文件（Rabbitmq）**

1 加入依赖

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

2 windows下有一个 路径为192.168.99.100:15672

Linux下有一个 没有路径，默认安装的。暂时没看

Windows下打开后默认用户名和密码为guest，在配置文件中指定ip和端口即可自动注册

3 需要server端暴露接口 bus-refresh

**management:  
 endpoints:  
 web:  
 exposure:  
 include:** "\*"

4 用postman访问这个接口http://localhost:8090/actuator/bus-refresh(这一步可以交由git完成)

5 别忘记在controller上加@RefreshScope注解

这个注解一般用在config文件上，config是用于接收经常变的属性的文件对象，里面有动态刷新的属性，在类上加这个注解。然后在别的地方直接注入使用.

6 自动触发bus-refresh接口暂时的解决方案是用gitee上的webhock

Springcloud提供了一个路由，专门为了触发bus-refresh用的。域名后缀加monitor，下面选json

7 更新代码后大概会有10秒钟左右的延迟

码云上的配置

**spring:  
 application:  
 name:** config  
 **cloud:  
 config:  
 server:  
 git:  
 uri:** https://gitee.com/qinliang11/config-repo.git  
 **username:** 836219171@qq.com  
 **password:** qinliang110  
 **basedir:** E:\git\gitproject\springcloudconfig

Git的配置

**spring:  
 application:  
 name:** config  
 **cloud:  
 config:  
 server:  
 git:  
 uri:** https://github.com/836219171/config-repo  
 **username:** 836219171  
 **password:** qinliang110  
 **basedir:** E:\git\gitproject\springcloudconfig

**记录一下浪费了两个小时的大坑**

原因是不能自动更新配置，

网上找了好多办法都行不通，后来仔细阅读官方文档之后，发现这一句。注意标红处。

但是在我的2.10build的版本上 github还是不行，gitee可以，所以配置还是换成

9. Push Notifications and Spring Cloud Bus

Many source code repository providers (such as Github, Gitlab, Gitee, or Bitbucket) notify you of changes in a repository through a webhook. You can configure the webhook through the provider’s user interface as a URL and a set of events in which you are interested. For instance, Github uses a POST to the webhook with a JSON body containing a list of commits and a header (X-Github-Event) set to push. **If you add a dependency on the spring-cloud-config-monitor library and activate** the Spring Cloud Bus in your Config Server, then a /monitor endpoint is enabled.

When the webhook is activated, the Config Server sends a RefreshRemoteApplicationEvent targeted at the applications it thinks might have changed. The change detection can be strategized. However, by default, it looks for changes in files that match the application name (for example, foo.properties is targeted at the foo application, while application.properties is targeted at all applications). The strategy to use when you want to override the behavior is PropertyPathNotificationExtractor, which accepts the request headers and body as parameters and returns a list of file paths that changed.

The default configuration works out of the box with Github, Gitlab, Gitee, or Bitbucket. In addition to the JSON notifications from Github, Gitlab, Gitee, or Bitbucket, you can trigger a change notification by POSTing to /monitor with form-encoded body parameters in the pattern of path={name}. Doing so broadcasts to applications matching the {name} pattern (which can contain wildcards).

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

MQ应用场景

1 异步处理

例如用户登录后，将短信服务，积分服务等去队列里获取资源再执行。

2 流量削峰

常见于秒杀场景，流量暴增的时候在应用前端加入消息队列，超过峰值的时候抛弃用户请求

3 日志处理

Kalka日志处理，分布式日志处理

4 应用解耦

队列监听端口

1 使用已有的队列

@RabbitListener(“myQueue”)

2可以实现如果没有队列，可以创建一个新队列

@RabbitListener(queuesToDeclare = @Queue("myQueue"))

3 和exchange路由绑定

@RabbitListener(bindings = @QueueBinding(  
 value = @Queue("myQueue"),  
 exchange = @Exchange("myExchange")  
))