配置中心| Config、Apollo

在SpringCloud项目中，每个服务都有自身独立的配置，这些配置文件分布在各自的服务器上，当我们需要修改某个服务的配置时，寻找及修改起来将十分麻烦，因此可以借助配置中心来统一管理所有服务的配置文件。

SpringCloud Config

SpringCloud Config通过Config Server来将配置文件放在Git仓库中进行统一管理，因此需要拥有一个Git仓库，并且配置文件名称要遵循【应用名称-开发环境.yml（或.properties）】格式。

搭建 Config 服务

创建Config服务，并引入Config依赖：

XML

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

激活@EnableConfigServer：

Java

```java
@SpringBootApplication
@EnableConfigServer//启动ConfigServer
public class Application{

  public static void main(String[] args){
    SpringApplication.run(Application.class,args);
  }

}
```

配置Config Server：

YAML

```yaml
server:
 port: 10000 #服务端口
spring:
 application:
   name: config-server #指定服务名
 cloud:
   config:
     server:
       git:
         #请求的git服务器仓库https地址
         uri: https://gitee.com/panda_99/cloud-config.git
```

查看git上的yml文件：http://localhost:10000/xxx.yml

客户端配置

配置文件要交给Config Server管理的服务引入依赖：

XML

```xml
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

服务原有配置文件被放在Git仓库中，因此需要新建一个bootstrap.yml来指定服务配置信息的来源：

YAML

```yaml
# bootstrap.yml: 指明配置文件通过config server获取
spring:
  cloud:
    config:
      name: img5              #应用名称，需要对应git配置文件名的前半部分(img5-dev.yml)
      profile: dev            #开发环境，需要对应git配置文件名的后半部分(img5-dev.yml)
      label:
        uri: http://localhost:10000  #config-server的请求地址
```

动态刷新配置

我们知道修改配置文件后需要重启项目才能生效，服务配置交由Config管理后，我们可以在Git上修改配置信息，并使其配置动态刷新。

- 手动请求刷新：post

Config Server引入健康检查：

XML

```xml
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Config Server开启动态刷新：@RefreshScope

Java

```java
@RefreshScope  //开启动态刷新
@RestController
@RequestMapping("/img")
public class ImgController {

   //测试配置刷新效果
   @Value("${nametest}")
   private  String nametest;

   @RequestMapping("/test")
   public String testpram(){
       return nametest;
   }

}
```

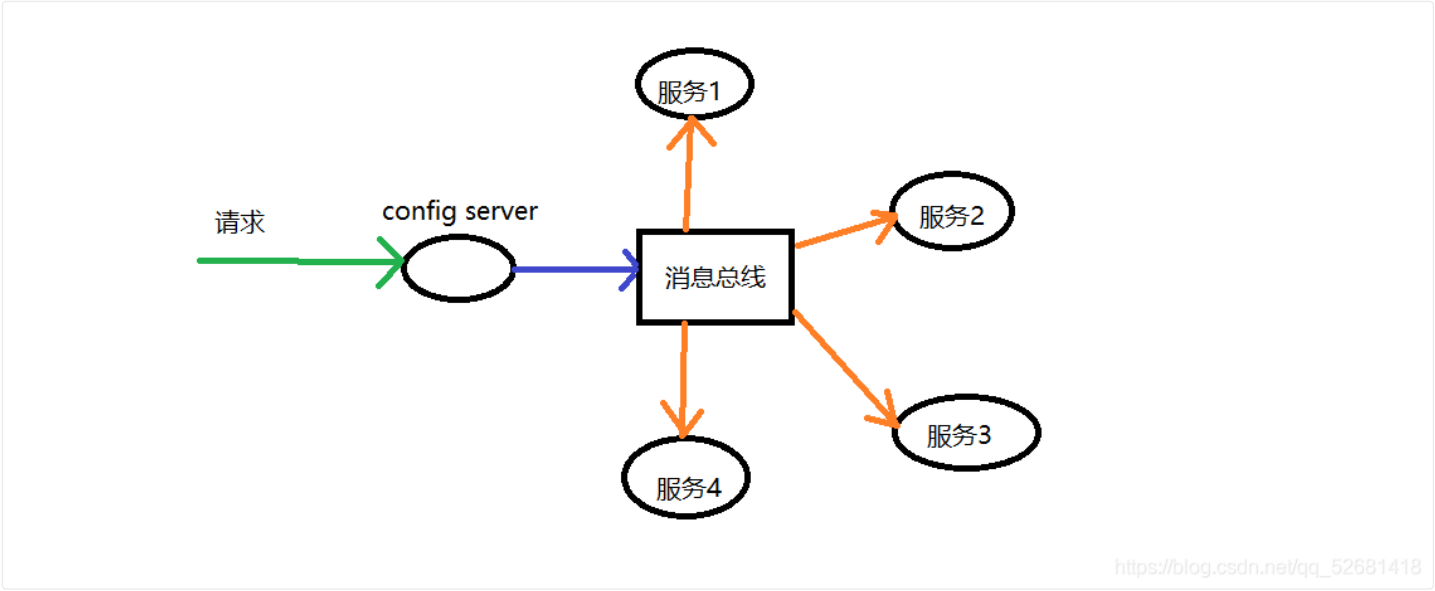
客户端bootstrap.yml添加配置：重启服务后生效，每次修改并发送Post请求后就会变为最新配置。

YAML

```yaml
#开启动态刷新的请求路径端点
management:
 endpoints:
   web:
     exposure:
       include: refresh #暴露的请求路径
```

手动刷新需要对修改的服务发送POST请求，这样无疑是一件麻烦事。

- 消息总线Bus、RabbitMQ：消息总线可以在接收到一个消息后给其他所有服务发送消息，因此向服务总线发送消息，总线再向其它服务发送消息，便可以同时请求所有服务。



服务端、客户端添加依赖：

```xml
<!--config客户端：此模块配置将存在码云-->
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<!--Bus-->
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-bus</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-stream-binder-rabbit</artifactId>
</dependency>
```

服务端、客户端（修改Git仓库中配置文件）添加配置：

```yaml
#消息总线配置
rabbitmq:
   host: 127.0.0.1
   port: 5672
   username: guest
   password: guest
```

发送post请求到：http://localhost:10000/actuator/bus-refresh ，即可刷新全部配置。

配置中心高可用、Config 集群

Config Server管理所有服务的配置文件，影响巨大，因此我们需要搭建Config Server集群来保证其可靠性。

可以将Config Server注册到注册中心，其他服务通过注册中心来获取配置：

```yaml
# 修改客户端配置：bootstrap.yml，指明配置文件通过config server获取
spring:
 cloud:
   config:
     name: cloudimg1  #应用名称，需要对应git配置文件名的前半部分
     profile: dev     #开发环境，需要对应git配置文件名的后半部分
     label:
     #uri: http://localhost:10000  # config-server的请求地址
     #通过注册中心获取config server配置
     discovery:
       enabled: true #开启服务发现
       service-id: config-server
#开启动态刷新的请求路径端点
```

```
management:
 endpoints:
   web:
     exposure:
       include: refresh
```

Apollo

由携程研发出的分布式配置中心框架，具有不发送请求进行实时刷新、版本回滚、灰度发布等优点。

环境要求：java（服务端1.8+、客户端1.7+）、mysql5.6.5+

搭建 Apollo 服务

- Apollo Server下载地址：https://github.com/nobodyiam/apollo-build-scripts
- 数据库创建子库：ApolloPortalDB 、ApolloConfigDB

AppIloConfigDB.sql 17.02 KB

ApolloPortalDB.sql 21.15 KB

- 修改启动脚本demo.sh，配置数据库连接，并启动：

.properties
```
#apollo config db info
apollo_config_db_url=jdbc:mysql://localhost:3306/ApolloConfigDB?characterEncoding=utf8
apollo_config_db_username=用户名
apollo_config_db_password=密码（如果没有密码，留空即可）

# apollo portal db info
apollo_portal_db_url=jdbc:mysql://localhost:3306/ApolloPortalDB?characterEncoding=utf8
apollo_portal_db_username=用户名
apollo_portal_db_password=密码（如果没有密码，留空即可）
```

- 启动脚本会在本地启动3个服务，分别使用8070, 8080, 8090端口，请确保这3个端口当前没有被使用:

Ada
```
./demo.sh start
```

访问Apollo配置中心http://localhost:8070 ，默认用户名密码apollo/admin.

客户端配置

引入依赖：

XML
```
<dependency>
  <groupId>com.ctrip.framework.apollo</groupId>
  <artifactId>apollo-client</artifactId>
  <version>1.1.0</version>
</dependency>
```

Apollo配置：

YAML
```
#注入默认 application namespace
apollo:
```