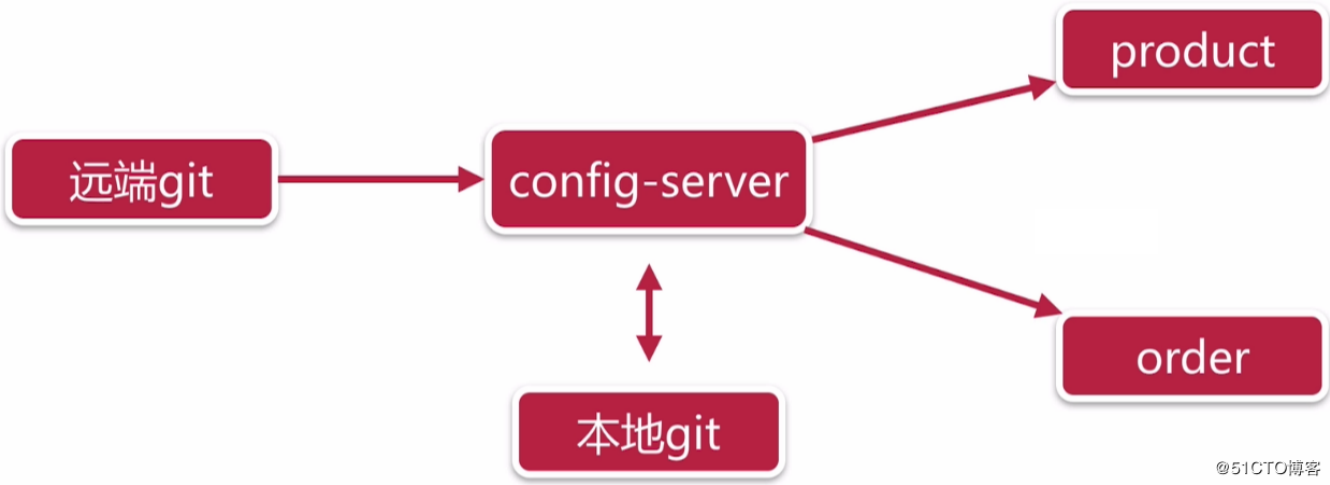
分布式配置中心

> 更多学习资料，一点课堂：www.yidiankt.com
>
> 一点课堂QQ群：984370849， QQ：2868289889
>
> 微信号：chengweixin9
>
> 免费公开课：https://ke.qq.com/course/394307

为什么需要分布式配置中心?

在分布式系统中，由于服务数量巨多，为了方便服务配置文件统一管理，所以需要分布式配置中心组件。在Spring Cloud中，有分布式配置中心组件spring cloud config ，它支持配置服务放在配置服务的内存中（即本地），也支持放在远程Git仓库中。在spring cloud config 组件中，分两个角色，一是config server，二是config client。



没有使用统一配置中心时，所存在的问题

- 配置文件分散在各个项目里，不方便维护
- 配置内容安全与权限，实际开发中，开发人员是不知道线上环境的配置的
- 更新配置后，项目需要重启

有哪些开源配置中心

1. spring-cloud/spring-cloud-config https://github.com/spring-cloud/spring-cloud-config spring出品，可以和spring cloud无缝配合
2. diamond https://github.com/takeseem/diamond
3. disconf https://github.com/knightliao/disconf
4. ctrip apollo https://github.com/ctripcorp/apollo/ Apollo（阿波罗）是携程框架部门研发的开源配置管理中心，具备规范的权限、流程治理等特性。

可用性与易用性

功能点	优先级	spring-cloud-config	ctrip apollo	disconf
单点故障	高	支持HA部署	支持HA部署	支持HA部署，高可用由zookeeper保证
多数据中心部署	高	支持	支持	支持
配置界面	中	无，需要通过git操作	统一界面	统一界面

快速入门（config-server）

1. 创建server-config模块项目
2. pom文件

```
<dependencies>
    <!--Spring Cloud Config 服务端依赖-->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
    </dependency>
</dependencies>
```

3. 创建启动类

```
@SpringBootApplication
@EnableConfigServer
public class ConfigServerApp {
    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApp.class, args);
    }
}
```

4. 配置文件
    - properties

```
spring.application.name=config-server
server.port=9100

spring.cloud.config.server.git.uri=https://gitee.com/yidiankt/config-server-test.git
spring.cloud.config.server.git.searchPaths=/**
spring.cloud.config.label=master
spring.cloud.config.server.git.username=
spring.cloud.config.server.git.password=
```

    - yaml

```yaml
server:
  port: 9100

spring:
  application:
    name: config-server
  cloud:
    config:
      server:
        git:
          uri: https://gitee.com/yidiankt/server-config.git
          search-paths: /**
          username: username
          password: password
        label: master
```

5. 创建码云配置仓库：https://gitee.com/
6. 创建测试的配置文件
    - 文件名-环境名.后缀
7. Config支持我们使用的请求的参数规则为：
    - /{应用名}/{环境名}[/{分支名}]
        - http://localhost:9100/config-server/dev
    - /{应用名}-{环境名}.yml
    - /{应用名}-{环境名}.properties
        - http://localhost:9100/config-server-dev.properties
    - /{分支名}/{应用名}-{环境名}.yml
    - /{分支名}/{应用名}-{环境名}.properties
        - http://localhost:9100/master/config-server-dev.properties

快速入门（config-client）

1. 新建模块项目config-client
2. pom文件

```xml
<dependencies>
    <!--Spring Cloud Config 客户端依赖-->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-config</artifactId>
    </dependency>
    <!-- web的依赖，必须加 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

3. 新建启动类

```
@SpringBootApplication
public class ConfigClientApp {

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

4. 创建测试controller

```
@RestController
public class TestController {

    @Value("${address}")
    private String test;

    /**
     * 返回配置文件中的值
     */
    @GetMapping("/value")
    @ResponseBody
    public String returnFormValue(){
        return test;
    }
}
```

5. 创建配置文件
    - bootstrap.properties

      ```
      # 需要和git上的文件名相同
      spring.application.name=hello-config
      spring.cloud.config.label=master
      spring.cloud.config.profile=dev
      spring.cloud.config.uri=http://localhost:9100/
      server.port=9200
      ```

    - bootstrap.yml

```yaml
spring:
  application:
    name: hello-config                          #指定了配置文件的应用名
  cloud:
    config:
      uri: http://localhost:9100/               #Config server的uri
      profile: dev                              #指定的环境
      label: master                             #指定分支
server:
  port: 9201
```

Config+Bus : 实现动态刷新

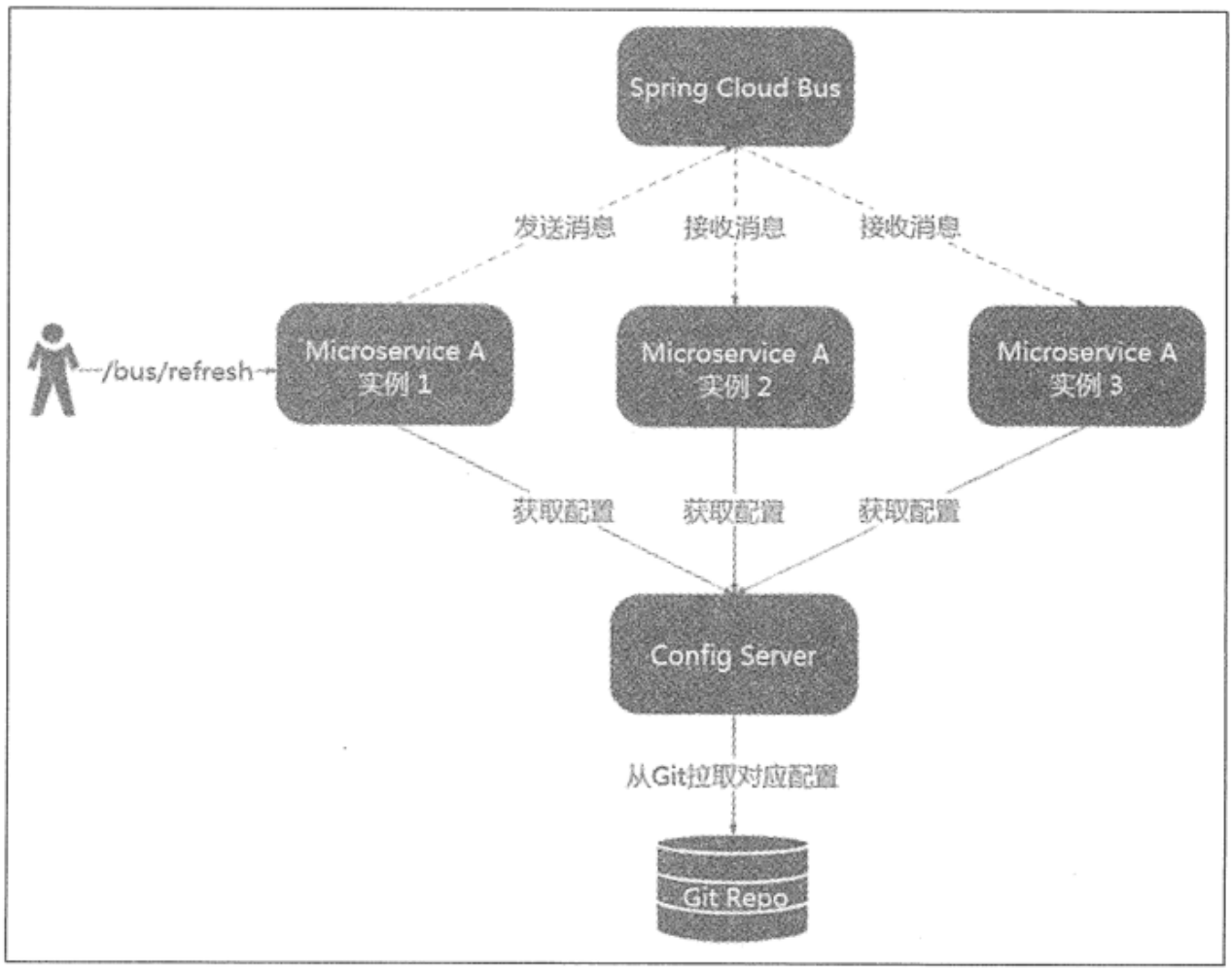


图 9-2　使用 Spring Cloud Bus 的架构图

1. config-client添加依赖

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

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

2. 添加注解@RefreshScope

```
@RestController
//开启更新功能
@RefreshScope
public class TestController {
```

3. 启动rabbitmq消息队列
4. 修改配置文件

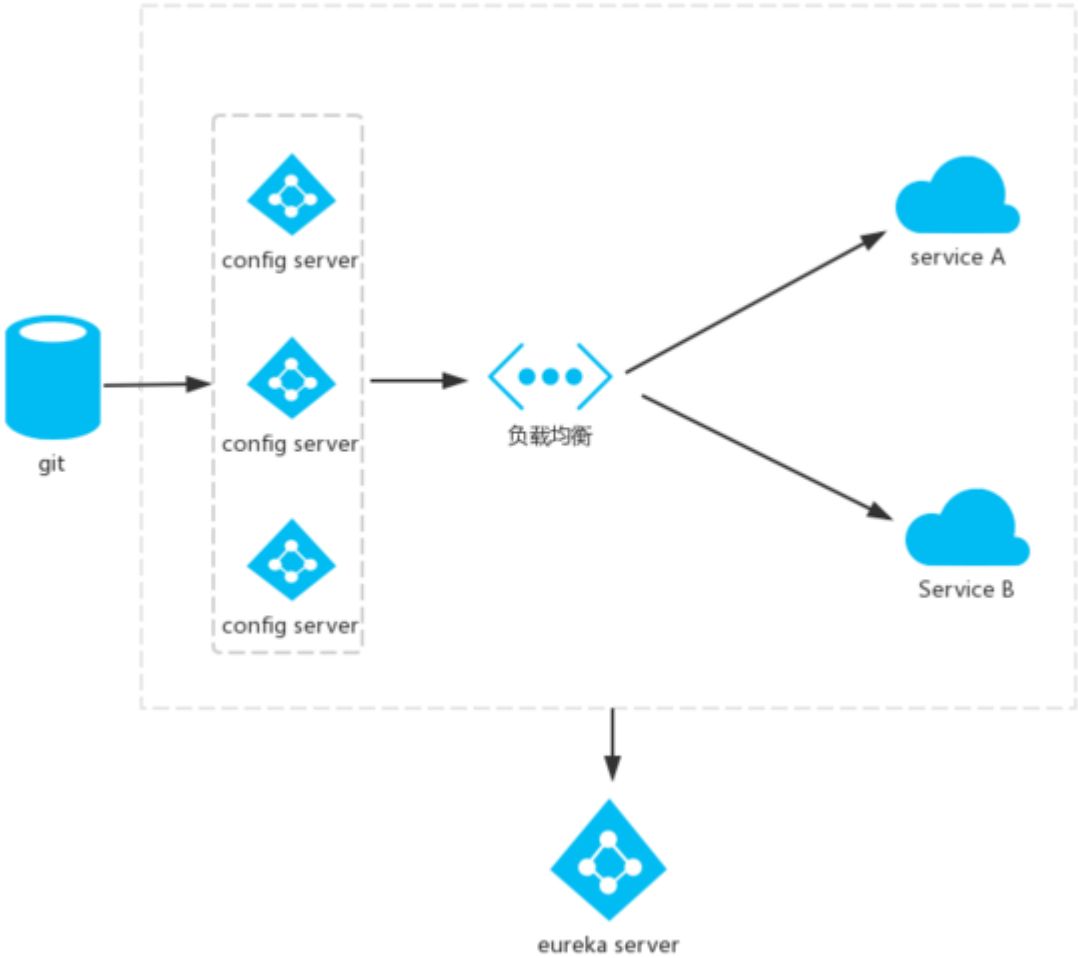
```
spring:
  rabbitmq:
    host: 192.168.57.101
    port: 5672
    username: guest
    password: guest
  application:
    name: hello-config                    #指定了配置文件的应用名
  cloud:
    config:
      uri: http://localhost:9100/         #Config server的uri
      profile: dev                        #指定的环境
      label: master                       #指定分支
server:
  port: 9201

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

5. 本次测试地址：POST请求
    - http://localhost:9201/actuator/bus-refresh

高可用的分布式配置中心

1. 架构图



> 在微服务架构中，宗旨就是要实现服务的高可用，配置中心也必须满足这个条件；所以我们将配置中心也注册到注册中心，这样，服务端就可以以调用服务的方式来访问配置中心的配置了。

2. 修改config-server

- pom依赖

```
<!-- 改造高可用配置中心 -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

- 修改配置文件

```
server:
  port: 9101

spring:
  application:
    name: cfg-server-test
  cloud:
```

```
    config:
      server:
        git:
          uri: https://gitee.com/yidiankt/config-server-test.git
          search-paths: /**
          username:
          password:
      label: master


eureka:
  client:
    service-url:
          defaultZone: http://127.0.0.1:8888/eureka
```

- 修改启动类

```
@EnableEurekaClient
```

3. 修改config-client
   - pom依赖

```
<!-- 改造高可用配置中心 -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

   - 配置文件

```
spring:
  rabbitmq:
    host: 192.168.57.101
    port: 5672
    username: guest
    password: guest
  application:
    name: hello-config                      #指定了配置文件的应用名
  cloud:
    config:
      ### uri: http://localhost:9100/
      profile: dev                          #指定的环境
      label: master                         #指定分支
      discovery:
        enabled: true
        # 使用服务名
        service-id: cfg-server-test

server:
  port: 9201
```

```yaml
management:
  endpoints:
    web:
      exposure:
        include: bus-refresh

eureka:
  client:
    service-url:
          defaultZone: http://127.0.0.1:8888/eureka
```