

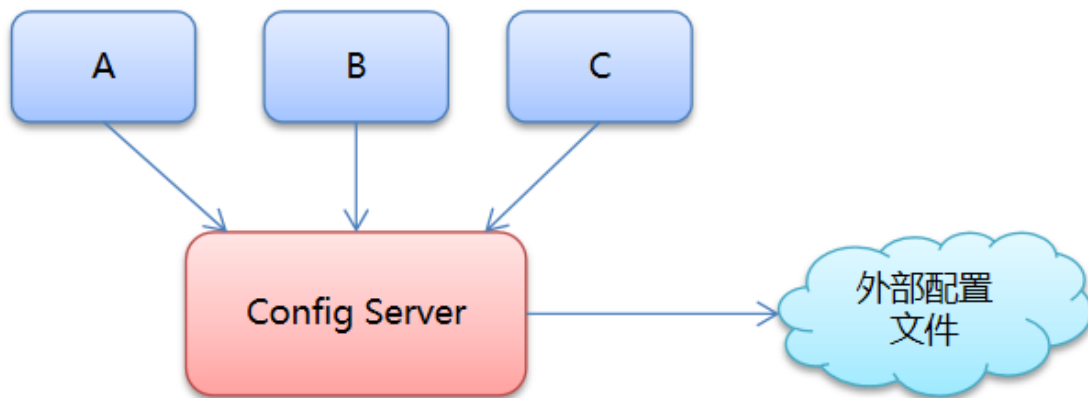
Config分布式配置中心

概述

Spring Cloud Config 解决了在分布式场景下多环境配置文件的管理和维护。

好处：

- 集中管理配置文件
- 不同环境不同配置，动态化的配置更新
- 配置信息改变时，不需要重启即可更新配置信息到服务



config server:

1. 使用gitee创建远程仓库，上传配置文件
2. 搭建 config server 模块
3. 导入 config-server 依赖
4. 编写配置，设置 gitee 远程仓库地址
5. 测试访问远程配置文件

config client:

1. 导入 starter-config 依赖
2. 配置config server 地址，读取配置文件名称等信息
3. 获取配置值
4. 启动测试

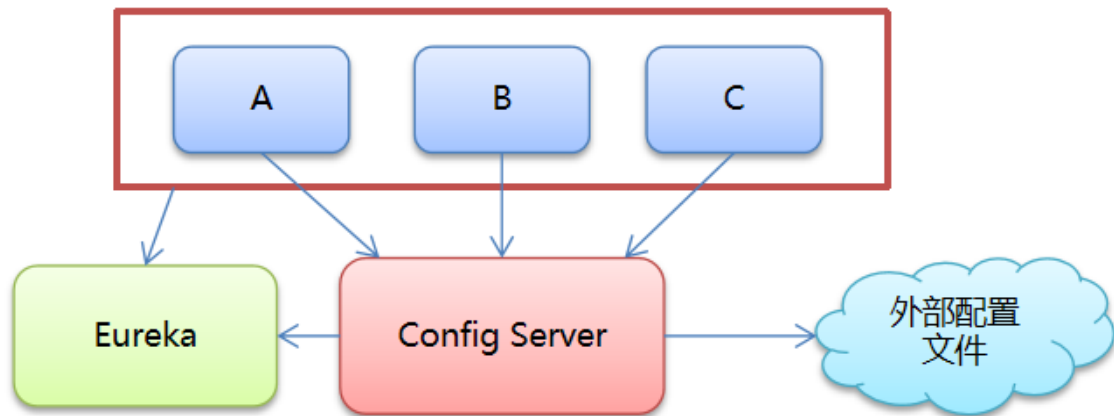
Config 客户端刷新

1. 在 config 客户端引入 actuator 依赖
2. 获取配置信息类上，添加 @RefreshScope 注解
3. 添加配置
 - management.endpoints.web.exposure.include: refresh
4. 使用curl工具发送post请求
curl -X POST <http://localhost:8001/actuator/refresh>

Config 集成 Eureka

```
#config-client配置:
spring:
  cloud:
    config:
      discovery:
        enabled: true
        service-id: config-server

#config-server配置:
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
```



Config 集成 Eureka

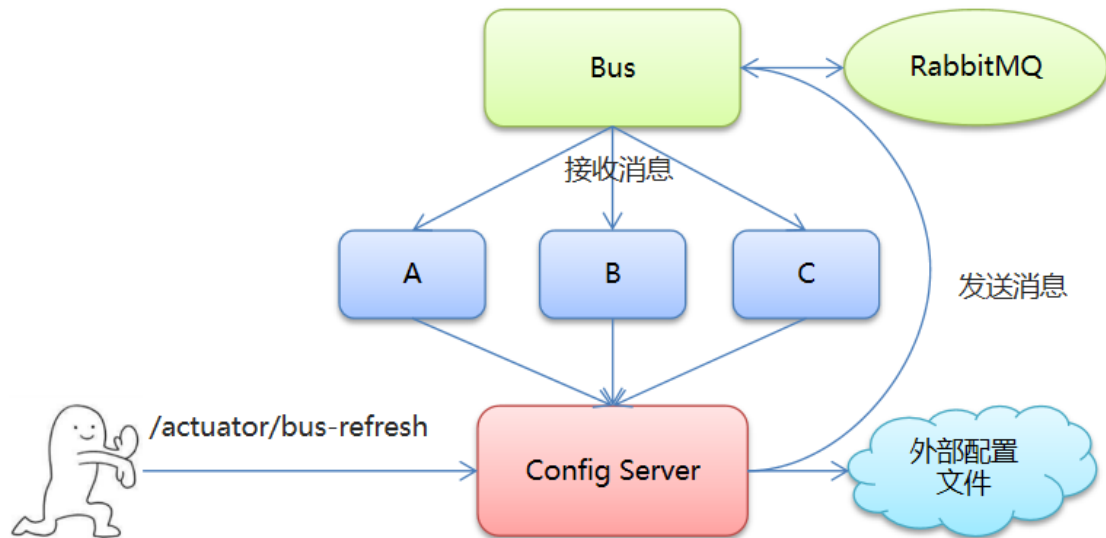
```
spring:
  cloud:
    config:
      discovery:
        enabled: true
        service-id: config-server

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
```

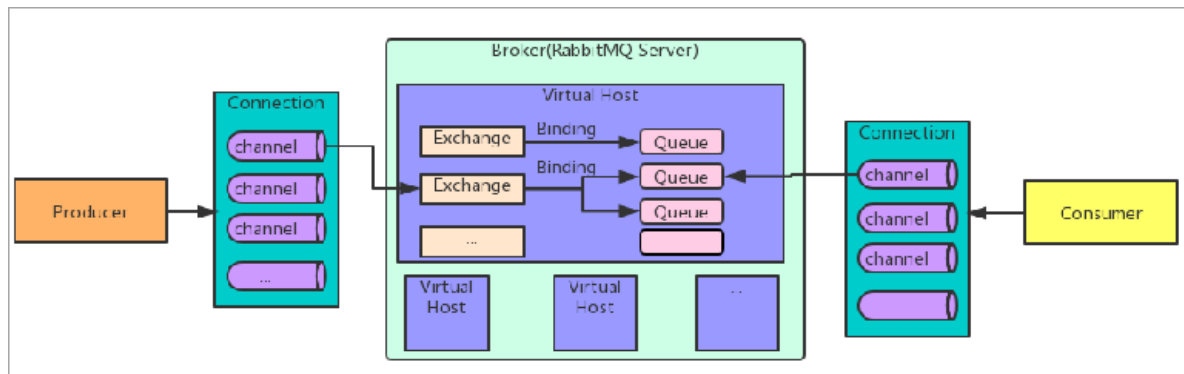
Bus 消息总线

Bus 概述

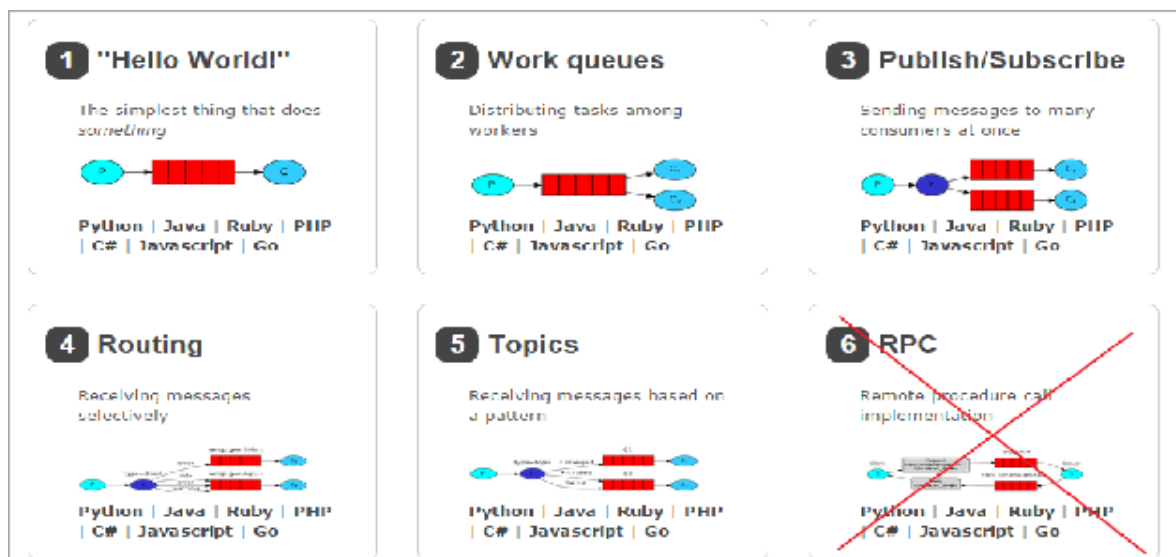
- Spring Cloud Bus 是用轻量的消息中间件将分布式的节点连接起来，可以用于广播配置文件的更改或者服务的监控管理。关键的思想就是，消息总线可以为微服务做监控，也可以实现应用程序之间相通信。
- Spring Cloud Bus 可选的消息中间件包括 RabbitMQ 和 Kafka 。



RabbitMQ 回顾



RabbitMQ 提供了 6 种工作模式：简单模式、work queues、Publish/Subscribe 发布与订阅模式、Routing 路由模式、Topics 主题模式、RPC 远程调用模式（远程调用，不太算 MQ；暂不作介绍）。



RabbitMQ Windows安装

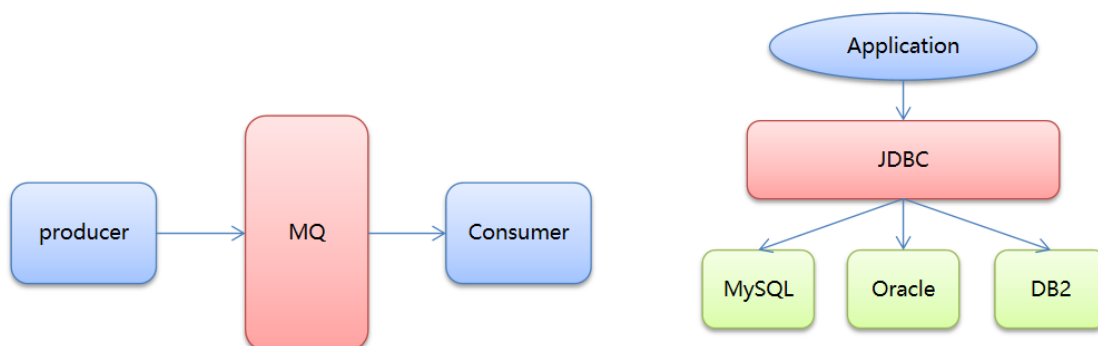
Bus 快速入门

1. 分别在 config-server 和 config-client 中引入 bus 依赖: bus-amqp
2. 分别在 config-server 和 config-client 中配置 RabbitMQ
3. 在 config-server 中设置暴露监控断点: bus-refresh
4. 启动测试

Stream 消息驱动

Stream 概述

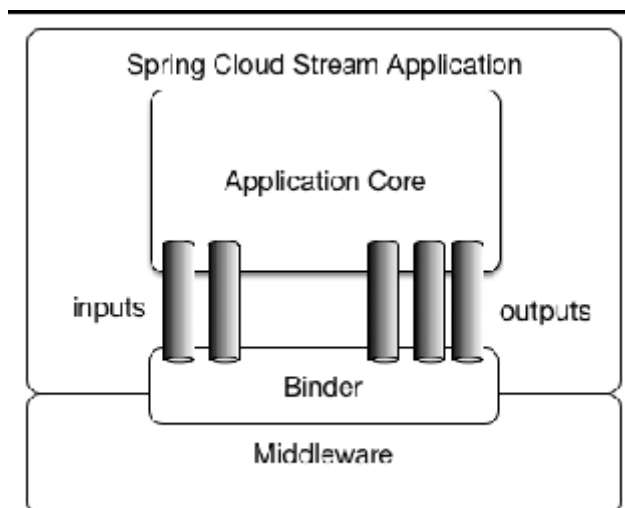
- Spring Cloud Stream 是一个构建消息驱动微服务应用的框架。
- Stream 解决了开发人员无感知的使用消息中间件的问题，因为Stream对消息中间件的进一步封装，可以做到代码层面对中间件的无感知，甚至于动态的切换中间件，使得微服务开发的高度解耦，服务可以关注更多自己的业务流程。
- Spring Cloud Stream目前支持两种消息中间件RabbitMQ和Kafka



Stream 组件

- Spring Cloud Stream 构建的应用程序与消息中间件之间是通过绑定器 Binder 相关联的。绑定器对于应用程序而言起到了隔离作用，它使得不同消息中间件的实现细节对应用程序来说是透明的。
- binding 是我们通过配置把应用和spring cloud stream 的 binder 绑定在一起
- output: 发送消息 Channel，内置 Source接口

- input: 接收消息 Channel, 内置 Sink接口



Stream 消息生产者

1. 创建消息生产者模块, 引入依赖 starter-stream-rabbit
2. 编写配置, 定义 binder, 和 bindings
3. 定义消息发送业务类。添加 @EnableBinding(Source.class), 注入 MessageChannel output, 完成消息发送
4. 编写启动类, 测试

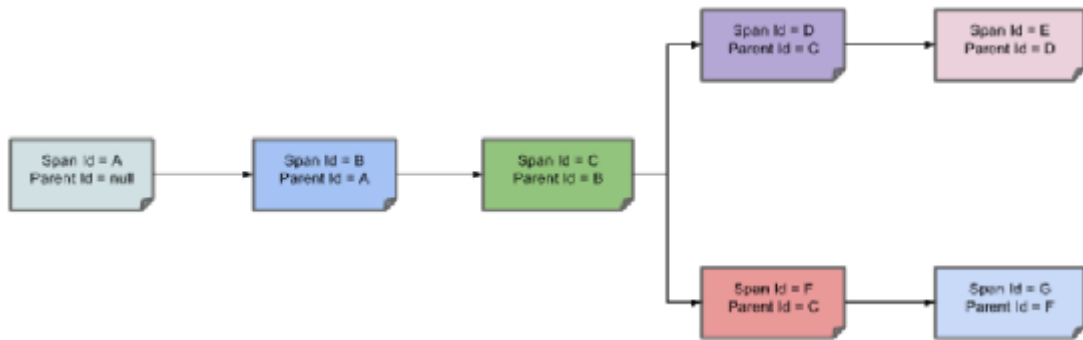
Stream 消息消费者

1. 创建消息消费者模块, 引入依赖 starter-stream-rabbit
2. 编写配置, 定义 binder, 和 bindings
3. 定义消息接收业务类。添加 @EnableBinding(Sink.class), 使用 @StreamListener(Sink.INPUT), 完成消息接收。
4. 编写启动类, 测试

Sleuth+Zipkin 链路追踪

概述

- Spring Cloud Sleuth 其实是一个工具, 它在整个分布式系统中能跟踪一个用户请求的过程, 捕获这些跟踪数据, 就能构建微服务的整个调用链的视图, 这是调试和监控微服务的关键工具。
 - 耗时分析
 - 可视化错误
 - 链路优化
- Zipkin 是 Twitter 的一个开源项目, 它致力于收集服务的定时数据, 以解决微服务架构中的延迟问题, 包括数据的收集、存储、查找和展现。



快速入门

1. 安装启动zipkin。java -jar zipkin.jar
2. 访问zipkin web界面。 <http://localhost:9411/>
3. 在服务提供方和消费方分别引入 sleuth 和 zipkin 依赖
4. 分别配置服务提供方和消费方。
5. 启动，测试