config配置中心介绍

为什么需要配置中心?

1、集中管理配置

2、不同环境不同配置

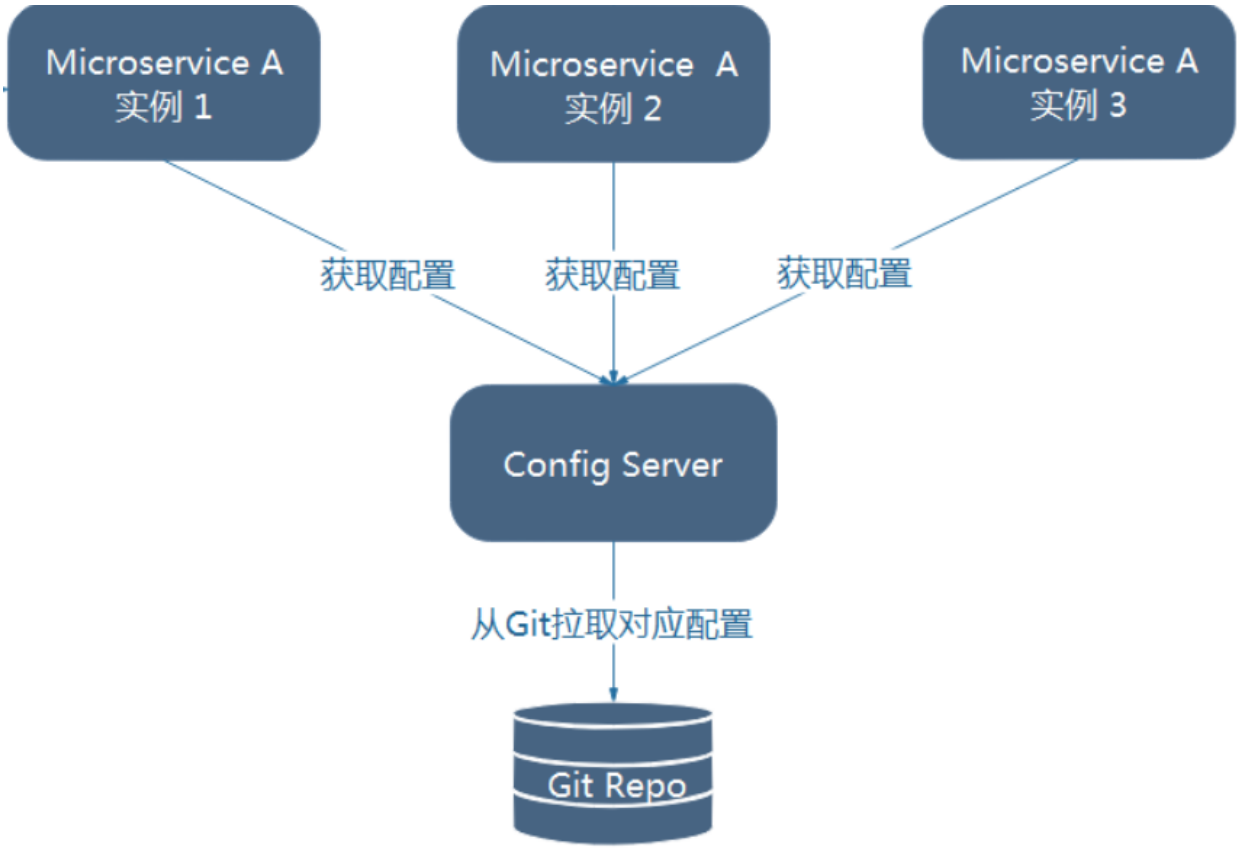
3、运行期间动态调整配置

4、自动刷新

Spring Cloud Config为分布式系统外部化配置提供了服务器端和客户端的支持，它包括Config Server和Config Client两部分。

Config Server是一个可横向扩展、集中式的配置服务器，它用于集中管理应用程序各个环境下的配置，默认使用Git存储配置内容(也可使用Subversion、本地文件系统或Vault存储配置),因此可以方便的实现对配置的版本控制与内容审计。

Config Client 是Config Server的客户端，用于操作存储在Config Server中的配置属性。



使用config实现配置中心服务端及客户端

首先新增git配置仓库中心，地址为：https://gitee.com/aaronrao/spring-cloud-config-test

在仓库里增加如下配置文件：

编写config配置中心服务端(使用git仓库存储)

见示例：10-ms-config-server

添加依赖，并在启动类上增加注解@EnableConfigServer

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

配置文件application.yml如下：

```
server:
  port: 8080
spring:
  application:
    name: microservice-config-server
  cloud:
    config:
      server:
        git:
          uri: https://gitee.com/aaronrao/spring-cloud-config-test.git      # 配置Git仓库的地址
          username:                                                         # Git仓库的账号
          password:                                                         # Git仓库的密码
```

启动项目，访问地址：http://localhost:8080/ms-config/dev，得到整个项目的配置信息


```
{
    "name": "ms-config",
    "profiles": ["dev"],
    "label": null,
    "version": "91da94890ab02bfc9ad4fc2905c7351739293661",
    "state": null,
    "propertySources": [{
        "name": "https://gitee.com/aaronrao/spring-cloud-config-test.git/ms-config-dev.properties",
        "source": {
            "profile": "dev-1.0"
        }
    }, {
        "name": "https://gitee.com/aaronrao/spring-cloud-config-test.git/ms-config.properties",
        "source": {
            "profile": "default-1.0"
        }
    }]
}
```


访问地址：http://localhost:8080/ms-config-dev.properties，得到配置文件内容

```
profile: dev-1.0
```

配置文件映射规则如下：

```
/{application}/{profile}[/{label}]
/{application}-{profile}.yml
/{label}/{application}-{profile}.yml
/{application}-{profile}.properties
/{label}/{application}-{profile}.properties
```

以上端点都可以映射到{application}-{profile}.properties这个配置文件，{application}表示微服务的名称，{label}对应Git仓库的分支，默认是 master

编写config配置中心客户端

见示例：10-ms-config-client

添加依赖

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

除了默认的application.yml配置文件，还需增加一个bootstrap.yml的配置文件，内容如下：

```
spring:
  application:
    name: ms-config    # 对应config server所获取的配置文件的{application}
  cloud:
    config:
      uri: http://localhost:8080/
      profile: dev            # profile对应config server所获取的配置文件中的{profile}
      label: master           # 指定Git仓库的分支，对应config server所获取的配置文件的{label}
```

spring cloud有一个“引导上下文"的概念，这是主应用程序的父上下文。引导上下文负责从配置服务器加载配置属性，以及解密外部配置文件中的属性。和主应用程序加载application.*(yml或 properties)中的属性不同，引导上下文加载(bootstrap.*)中的属性。配置在 bootstrap.*中的属性有更高的优先级，因此默认情况下它们不能被本地配置覆盖。

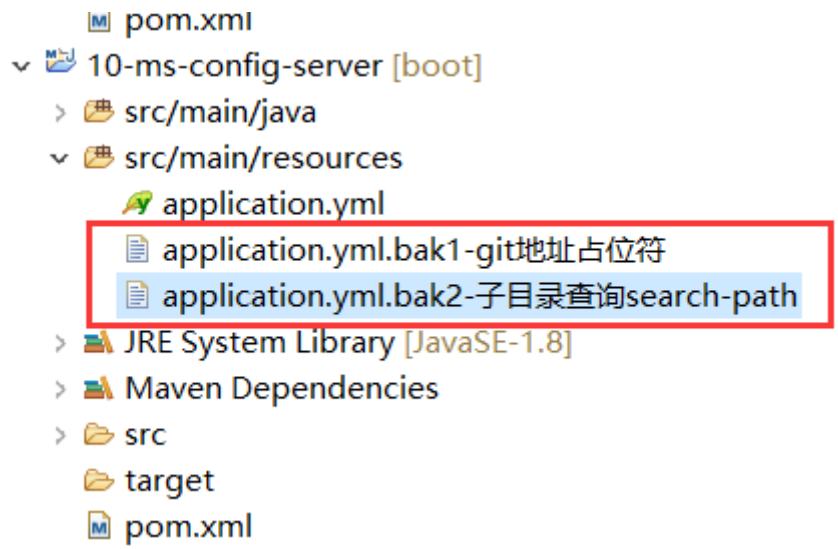
启动项目，访问地址：http://localhost:8081/profile，得到config配置中心的配置文件内容如下

dev-1.0

config配置中心服务端配置详解

见示例：10-ms-config-server

各种配置方法见项目配置文件，如下



配置信息的加解密安全处理

前文是在 Git仓库中明文存储配置属性的。很多场景下，对于某些敏感的配置内容（例如数据库账号、密码等），应当加密存储。 config server为配置内容的加密与解密提供了支持。

安装JCE

config server的加解密功能依赖Java Cryptography Extension（JCE）

Java8 JCE下载地址：

http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html

下载JCE并解压，将其中的jar包覆盖到JDK/jre/lib/security目录中

对称加密

config server提供了加密与解密的接口，分别是/encrypt与/decrypt

见示例：10-ms-config-server-encryption

添加依赖

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

增加一个配置文件bootstrap.yml，配置对称加密的密钥

```
encrypt:
  key: tuling  # 设置对称密钥
```

运行项目，验证加解密

加密：curl http://localhost:8080/encrypt -d mySecret

解密：curl http://localhost:8080/decrypt -d 767552394ee539201214199d55b0a91b1485d959b667eeabad29a101b9c1c61e

在git仓库增加配置文件ms-config-encryption-dev.yml，内容如下：

```
spring:
  datasource:
    username: dbuser
    password: '{cipher}767552394ee539201214199d55b0a91b1485d959b667eeabad29a101b9c1c61e'
```

访问地址：http://localhost:8080/ms-config-encryption-dev.yml，得到密钥原文

```
spring:
  datasource:
    password: mySecret
    username: dbuser
```

说明 config server能自动解密配置内容。一些场景下，想要让 config server直接返回密文本身，而并非解密后的内容，可设置 **spring.cloud.config.server.encrypt.enabled=false**，这时可由 ConfigClient自行解密。

配置信息手动刷新

很多场景下，需要在运行期间动态调整配置。如果配置发生了修改，微服务要如何实现配置的刷新呢?

见示例：10-ms-config-client-refresh

添加依赖如下，其中spring-boot-starter-actuator提供了/refresh端点，用于配置的刷新

```
<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>
```

在Controller上添加注解@RefreshScope，添加这个注解的类会在配置更改时得到特殊的处理

```java
@RestController
@RefreshScope
public class ConfigClientController {
  @Value("${profile}")
  private String profile;

  @GetMapping("/profile")
  public String hello() {
    return this.profile;
  }
}
```

1、启动项目(启动两个，一个端口8081，一个端口8082)，访问地址：http://localhost:8081/profile，得到结果：dev-1.0，访问地址：http://localhost:8082/profile，得到结果：dev-1.0

2、修改git仓库里的配置文件ms-config-dev.properties的内容为：

```
profile=dev-1.0-change
```

3、再次访问地址：http://localhost:8081/profile，得到结果还是dev-1.0，说明配置尚未刷新

4、发送post请求：http://localhost:8081/refresh，返回结果："profile"，表示profile这个配置属性已被刷新

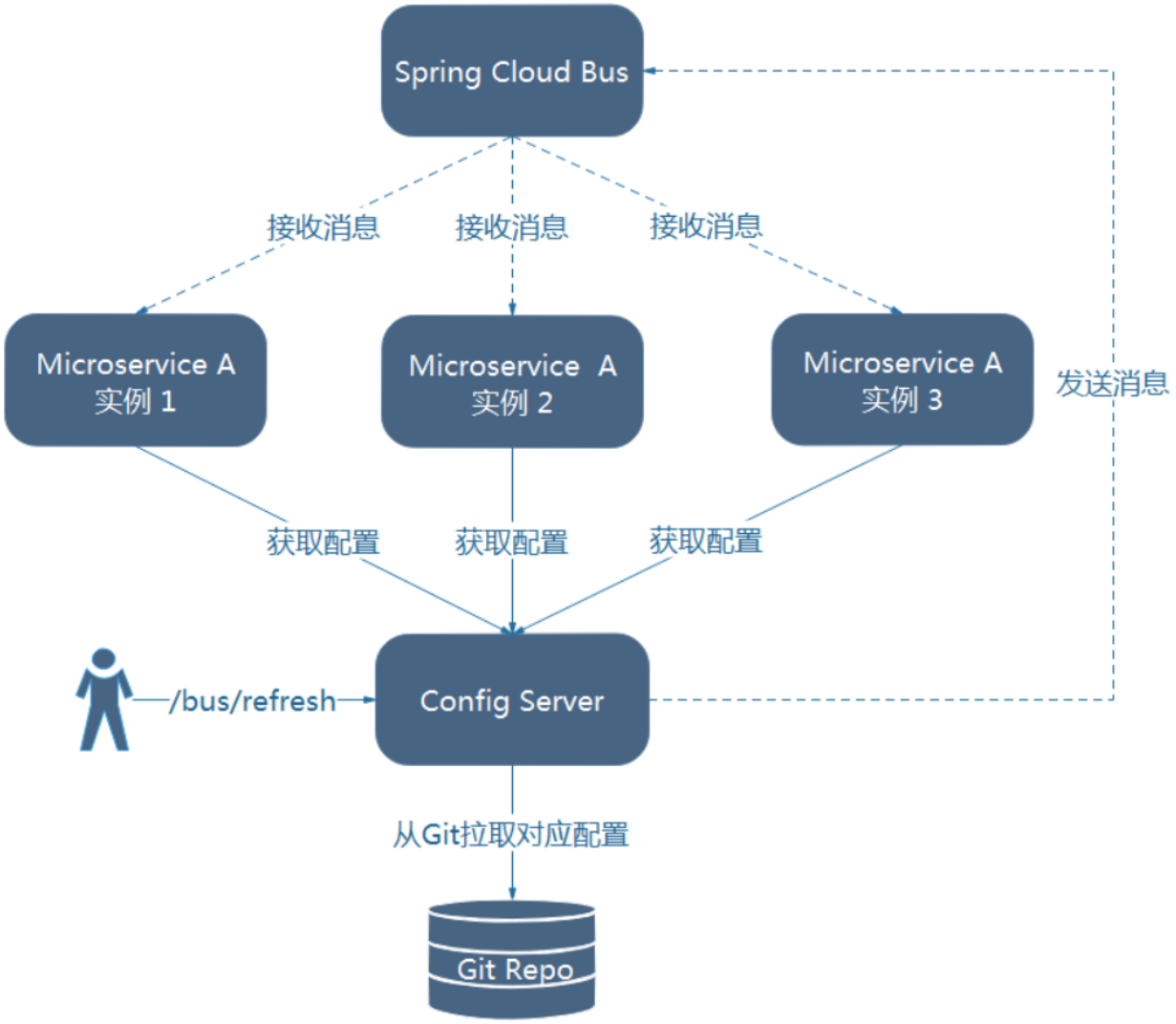
5、再次访问http://localhost:8081/profile，得到结果：dev-1.0-change，说明属性已刷新

6、再次访问http://localhost:8082/profile，得到结果：dev-1.0，说明8082的服务并没有刷新，还需再次手动刷新才能更新配置

配置信息自动刷新

前文讨论了使用/refresh端点手动刷新配置，但如果所有微服务节点的配置都需要手动去刷新，工作量可想而知。不仅如此，随着系统的不断扩张，会越来越难以维护。因此，实现配置的自动刷新是很有必要的，**Spring Cloud Bus**就可以实现配置的自动刷新。

Spring Cloud Bus使用轻量级的消息代理（例如 RabbitMQ、 Kafka等）连接分布式系统的节点，这样就可以广播传播状态的更改（例如配置的更新）或者其他的管理指令。可将Spring Cloud Bus想象成一个分布式的Spring Boot Actuator。



见示例：10-ms-config-server-refresh-cloud-bus和10-ms-config-client-refresh-cloud-bus

服务端示例：10-ms-config-server-refresh-cloud-bus

添加依赖

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

配置文件增加rabbitmq的配置：

```yaml
server:
  port: 8080
management:
  security:
    enabled: false  #关掉安全认证
spring:
  application:
    name: microservice-config-server
  cloud:
    config:
      server:
        git:
          uri: https://gitee.com/aaronrao/spring-cloud-config-test.git      # 配置Git仓库的地址
          username:                                                         # Git仓库的账号
          password:                                                         # Git仓库的密码
    bus:
      trace:
        enabled: true     # 开启cloud bus的跟踪
  rabbitmq:
    host: localhost
    port: 5672
    username: guest
    password: guest
```

客户端示例：10-ms-config-client-refresh-cloud-bus

添加依赖

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

配置文件增加rabbitmq的配置：

```yaml
spring:
  application:
    name: ms-config    # 对应config server所获取的配置文件的{application}
  cloud:
    config:
      uri: http://localhost:8080/
      profile: dev            # profile对应config server所获取的配置文件中的{profile}
      label: master           # 指定Git仓库的分支，对应config server所获取的配置文件的{label}
  rabbitmq:
    host: localhost
    port: 5672
    username: guest
    password: guest
```

运行项目(运行一个config server和两个config client)，修改git仓库里的配置文件，然后用post方式请求地址：http://localhost:8080/bus/refresh，如果返回成功，则config的所有客户端的配置都会动态刷新，详细演示参看视频

config的安全认证

见示例：10-ms-config-server-authenticating和10-ms-config-client-authenticating

config服务端：10-ms-config-server-authenticating

添加依赖：

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-server</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

application.yml配置如下：

```
server:
  port: 8080
spring:
  application:
    name: microservice-config-server
  cloud:
    config:
      server:
        git:
          uri: https://gitee.com/aaronrao/spring-cloud-config-test.git      # 配置Git仓库的地址
          username:                                                         # Git仓库的账号
          password:                                                         # Git仓库的密码
security:
  basic:
    enabled: true                # 开启基于HTTP basic的认证
  user:
    name: zhuge                  # 配置登录的账号是user
    password: 123456         # 配置登录的密码是password123
```

config客户端：10-ms-config-client-authenticating

在配置文件里bootstrap.yml里增加如下配置：

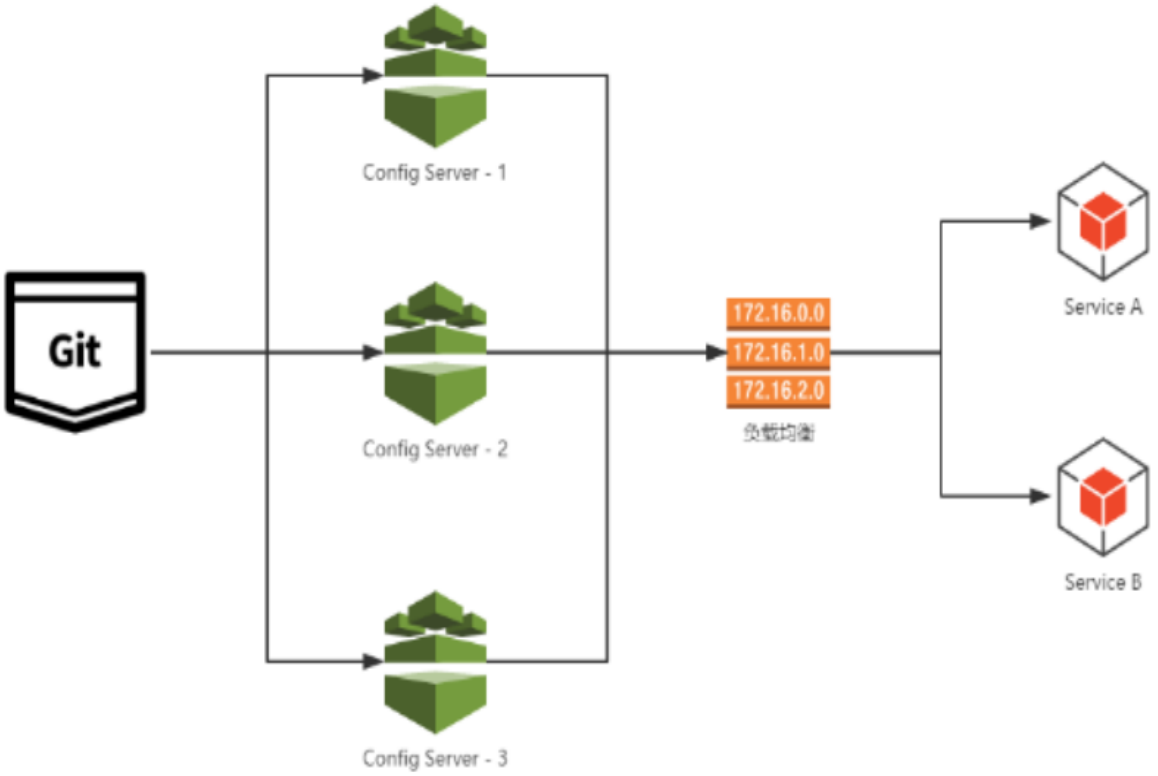
```
spring:
  application:
    name: ms-config    # 对应config server所获取的配置文件的{application}
  cloud:
    config:
      uri: http://localhost:8080/
      username: zhuge
      password: 123456
      profile: dev             # profile对应config server所获取的配置文件中的{profile}
      label: master            # 指定Git仓库的分支，对应config server所获取的配置文件的{label}
```

config与eureka配合使用

见示例：10-ms-config-server-eureka和10-ms-config-client-eureka

config配置中心的高可用

1、config server未注册到eureka上的情况，通过负载均衡器来实现



2、config server注册到eureka上的情况，client端也注册到eureka上，则已经实现高可用