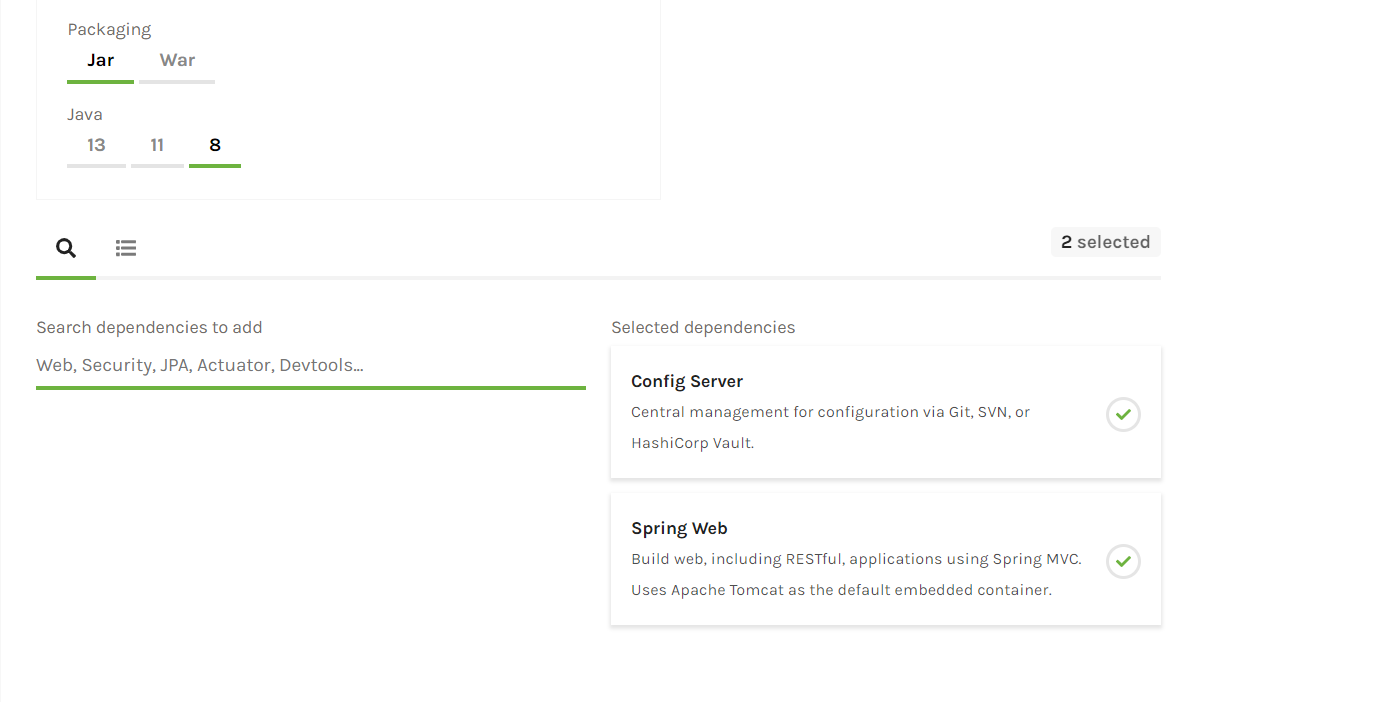
1. 使用Spring Cloud Config Server的原因。

对于一些简单的项目来说，我们一般都是直接把相关配置放在单独的配置文件中，以 properties 或者 yml 的格式出现，更省事儿的方式是直接放到 application.properties 或 application.yml 中。但是这样的方式有个明显的问题，那就是，当修改了配置之后，必须重启服务，否则配置无法生效。

使用Spring Cloud Config Server实现配置的集中管理与自动刷新(自动刷新需要依赖其它工具)。

2. 创建Spring Cloud Config Server步骤

⑴使用SpringBoot Initializr创建一个SpringBoot的应用程序，其中依赖中添加Config Server和Web。

接下来创建bootstrap.yml和application.yml，填写相关配置。

⑵不集成Eureka的配置中心

①bootstrap的内容

spring:

application:

name: config-server # 应用程序名

cloud:

config:

server:

git:

uri: git仓库在浏览器的地址

username: GitHub用户名

password: GitHub密码

default-label: master # git仓库分支

search-paths: config # 配置文件目录

application.yml的内容

②应用程序启动类添加注解@EnableConfigServer

server:

port: 8001 # 应用程序端口

③Spring Cloud Config的访问规则

/{application}/{profile}[/{label}]

/{application}-{profile}.yml

/{label}/{application}-{profile}.yml

/{application}-{profile}.properties

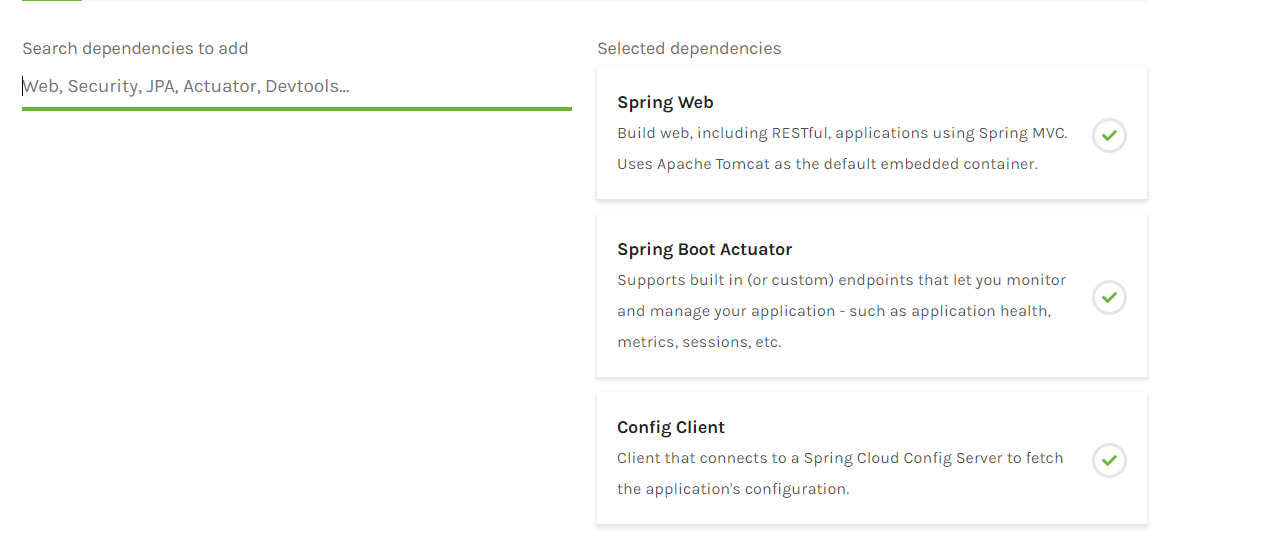
/{label}/{application}-{profile}.properties

{application} 就是应用名称，对应到配置文件上来，就是配置文件的名称部分，例如我上面创建的配置文件。

{profile} 就是配置文件的版本，我们的项目有开发版本、测试环境版本、生产环境版本，对应到配置文件上来就是以 application-{profile}.yml 加以区分，例如application-dev.yml、application-sit.yml、application-prod.yml。

{label} 表示 git 分支，默认是 master 分支，如果项目是以分支做区分也是可以的，那就可以通过不同的 label 来控制访问不同的配置文件了。

④客户端使用配置中心的配置

i. 使用SpringBoot Initializr创建一个SpringBoot的应用程序，其中依赖中添加Config Config，Web和Actuator。

ii. 创建bootstrap.yml。

bootstrap.yml

spring:

profiles:

active: dev

---

以上配置文件中，我们配置了两个环境。Spring Cloud Config Server的地址为http://localhost:8001，git仓库的分支为master，配置文件为service1-dev.yml和service1-prod.yml。

spring:

profiles: prod

application:

name: service1

cloud:

config:

uri: http://localhost:8001

lable: master

profile: prod

---

spring:

profiles: dev

application:

name: service1

cloud:

config:

uri: http://localhost:8001

label: master

profile: dev

iii. 代码中读取配置文件的配置，使用@Value和@ConfigurationProperties。

配置文件service1-dev.yml内容如下：

service1:

min: 1

max: 9

使用@Value读取的代码如下：

package com.longmao.service1.dto;

import lombok.Data;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

@Data

@Component

public class Service1Entity {

@Value("${service1.min}")

private int min;

使用@ConfigurationProperties读取的代码如下：

package com.longmao.service1.dto;

import lombok.Data;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

@Data

@Component

@ConfigurationProperties(prefix = "service1")

public class Service1Entity {

private int min;

private int max;

}

@Value("${service1.max}")

private int max;

}

iv. application.yml的内容：

server:

port: 8003

management:

endpoint:

shutdown:

enabled: false

endpoints:

web:

exposure:

include: "\*"

service1:

min: 0

max: 0

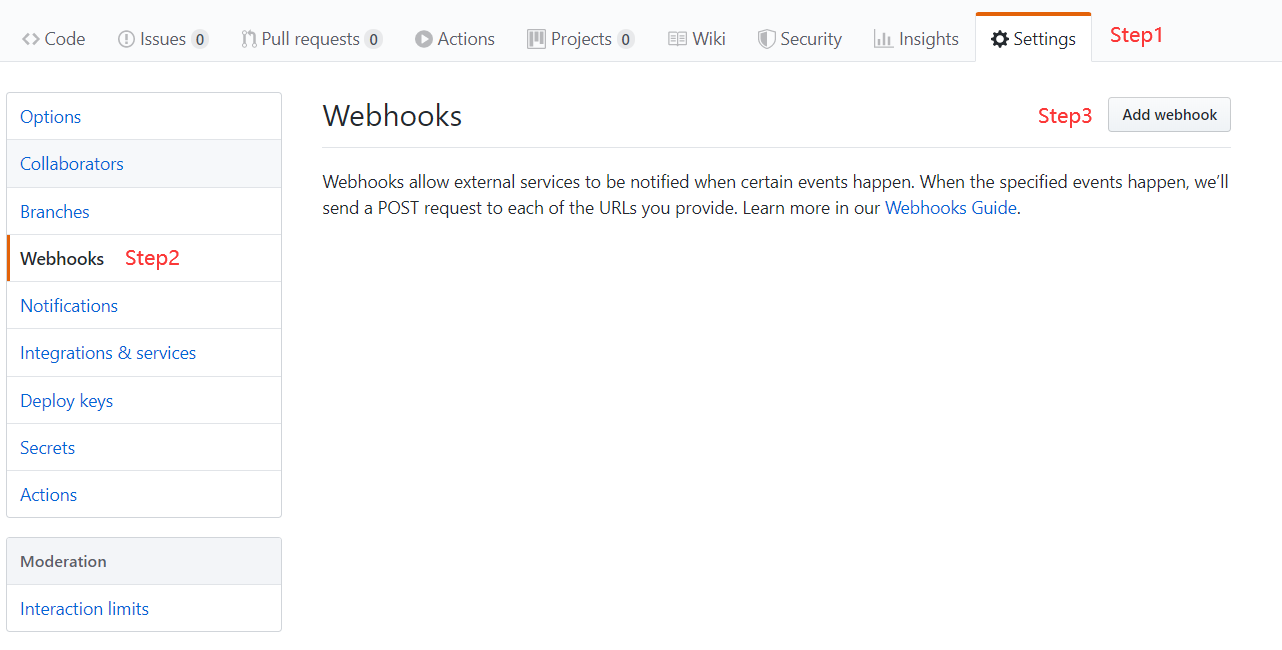
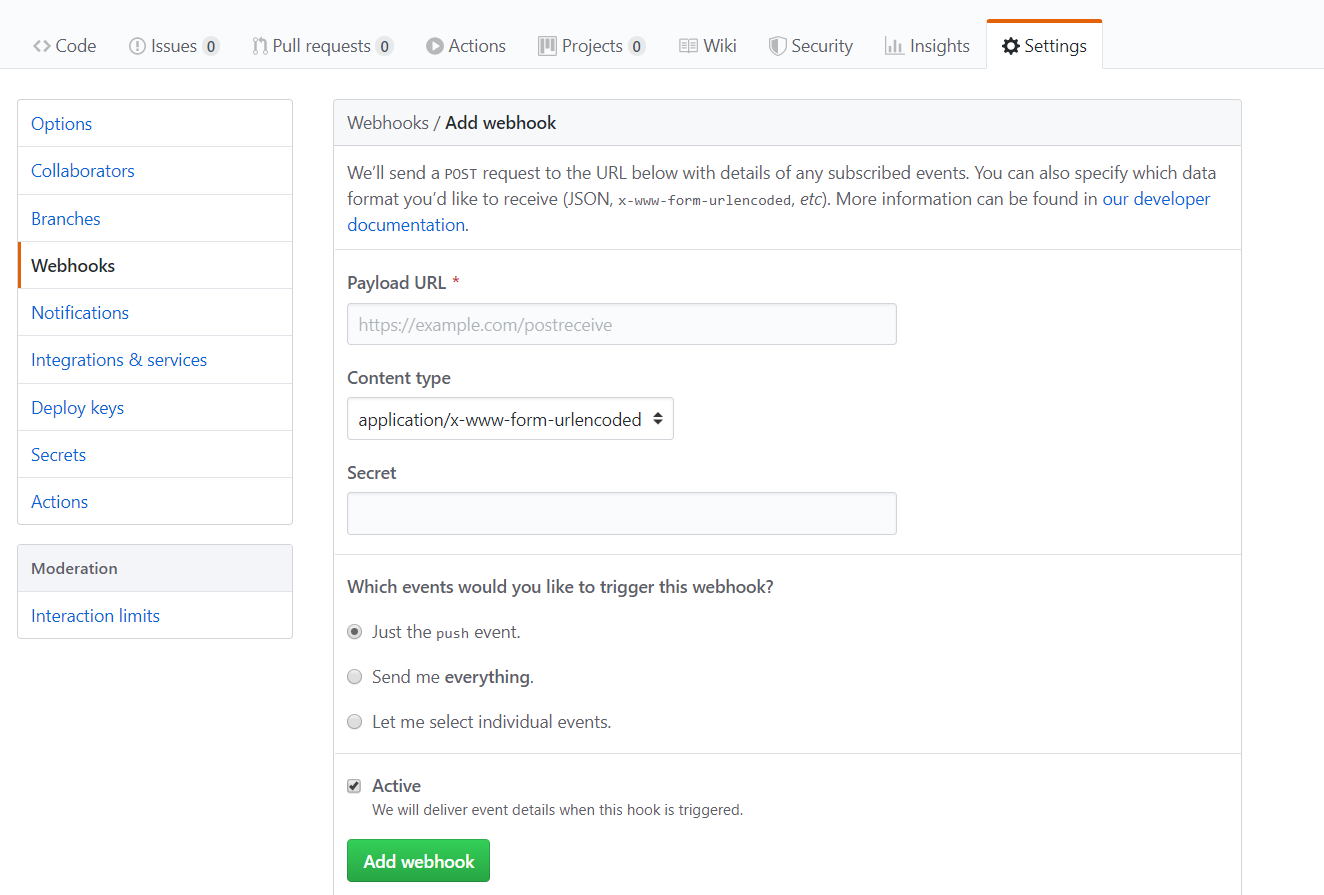
其中 management 是关于 actuator 相关的，接下来自动刷新配置的时候需要使用。service1部分是当无法读取配置中心的配置时，使用此配置，以免项目无法启动。

v. 实现自动刷新(@Value不会刷新，@ConfigurationProperties会刷新)

Spring Cloud Config 在项目启动时加载配置内容这一机制，导致了它存在一个缺陷，修改配置文件内容后，不会自动刷新。例如我们上面的项目，当服务已经启动的时候，去修改Github上的配置文件内容，这时候，再次刷新页面，对不起，还是旧的配置内容，新内容不会主动刷新过来。

@RefreshScope 注解并结合 actuator实现配置的刷新，在读取的配置的类上加上@RefreshScope注解，即上面使用Service1Entity类加上注解。

接下来发送POST http://localhost:8001/actuator/refresh这个接口实现刷新(可以用Postman发送接口，地址换成自己的地址)。

此处也可以使用Github的Webhook，Payload URL填写上面的地址。

⑶集成Eureka的配置中心

i. Spring Cloud Config Server端加入依赖Eureka Client，在application.yml中加入Eureka服务器的地址，这里为http://localhost:8761/eureka。

修改后的application.yml的内容：

server:

port: 8001

eureka:

client:

service-url:

default-zone: http://localhost:8761/eureka

在应用程序启动类中加上注解@EnableEurekaClient或者@EnableDiscoveryClient 。

ii. 客户端使用Eureka

pom中引入Eureka Client依赖，修改bootstrap.yml，添加Eureka客户端配置，修改环境配置(service-id为Spring Cloud Config Server的应用程序名)。修改后的bootstrap.yml如下：

spring:

profiles:

active: dev

eureka:

client:

service-url:

register-with-eureka: true

fetch-registry: true

default-zone: http://localhost:8761/eureka

---

spring:

profiles: prod

application:

name: service1

cloud:

config:

lable: master

profile: prod

discovery:

enabled: true

service-id: config-server

---

spring:

profiles: dev

application:

name: service1

⑷使用Spring Cloud Bus刷新多个使用Spring Cloud Config Server的客户端。

cloud:

config:

label: master

profile: dev

discovery:

enabled: true

service-id: config-server

如果只有一个 client 端的话，那我们用 webhook ，设置手动刷新都不算太费事，但是如果端比较多的话呢，一个一个去手动刷新未免有点复杂。这样的话，我们可以借助 Spring Cloud Bus 的广播功能，让 client 端都订阅配置更新事件，当配置更新时，触发其中一个端的更新事件，Spring Cloud Bus 就把此事件广播到其他订阅端，以此来达到批量更新。

Spring Cloud Bus 核心原理其实就是利用消息队列做广播，所以要先有个消息队列，目前官方支持 RabbitMQ 和 kafka。

这里用的是 RabbitMQ， 所以先要搭一套 RabbitMQ 环境。

使用消息队列的步骤：

①在Client端添加Bus依赖

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-bus-amqp</artifactId>

</dependency>

②在配置文件中加入如下配置：

spring:

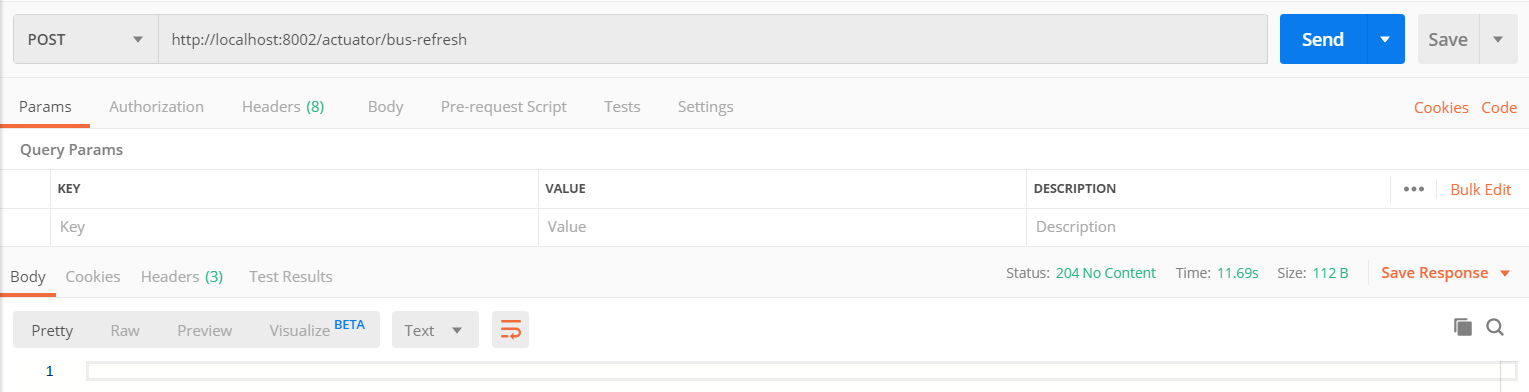
rabbitmq:

host: localhost

port: 5672

username: guest

password: guest

③启动多个Client端，修改其中一个Client端在Github的配置之后，使用POST方式提交/actuator/bus-refresh接口实现多个端配置的刷新。

