

SpringCloud Config

一 简介

在分布式微服务中,我们有大量的微服务和服务器,因此会有大量的重复的配置文件,当我们的配置发生变化的时候,需要重新部署到所有的机器上面,非常繁琐,为了解决这个痛点,我们将配置文件都放入到配置中心中,配置中心为了方便服务配置文件统一管理,实时更新,所以需要分布式配置中心组件,它就是Spring Cloud Config.

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,实时更新,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。

Spring Cloud Config为分布式系统中的外部配置提供服务器和客户端支持。使用Config Server,您可以为所有环境中的应用程序管理其外部属性。它非常适合spring应用,也可以使用在其他语言的应用上。随着应用程序通过从开发到测试和生产的部署流程,您可以管理这些环境之间的配置,并确定应用程序具有迁移时需要运行的一切。服务器存储后端的默认实现使用git,因此它轻松支持标签版本的配置环境,以及可以访问用于管理内容的各种工具。

Spring Cloud Config服务端特性

- HTTP, 为外部配置提供基于资源的API (键值对, 或者等价的YAML内容)
- 属性值的加密和解密 (对称加密和非对称加密)
- 通过使用@EnableConfigServer在Spring boot应用中非常简单的嵌入。

Config客户端的特性 (特指Spring应用)

- 绑定Config服务端, 并使用远程的属性源初始化Spring环境。
- 属性值的加密和解密 (对称加密和非对称加密)

springcloud config 的原理很简单,就是将不同服务的不同文件按照自己定义的规则放到不同的位置下,然后通过配置告诉 config server,让 server 可以在我们指定的参数下找到对应的位置下单文件,然后返回给我们的每个 config client, client 就是我们的不同的服务,这样设置后,以后我们只需要关注的是将文件修改后提交到对应的位置即可,不用再重新打包

二 Config server

在 SpringCloud 中,建议我们通过 Git 或者 SVN 来保存文件,这个我们可以进行版本化控制,本案例中使用gitee来保存我们的配置文件

2.1 创建文件仓库

在 git 上面创建了一个仓库 spring-cloud-config 用于存放我们的配置文件,示例仓库地址为 <https://gitee.com/mrjackiechan/spring-cloud-config/>

2.2 创建 config server 项目

2.2.1 pom依赖文件

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

2.2.2 application.yml

我们需要告诉 server 我们的 git 仓库在什么位置

```
server:
  port: 40000
spring:
  application:
    name: 24Configserver
  cloud:
    config:
      server:
        git:
          uri: https://gitee.com/mrjackiechan/spring-cloud-config #git仓库地址
          # username: #如果有用户名和密码
          # password:
```

2.2.3 主程序

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

/**
 * Created by jackiechan on 2019-09-02 11:24
 *
 * @Author jackiechan
 */
@SpringBootApplication
@EnableConfigServer //开启 configserver
public class ConfigServerStartApp {

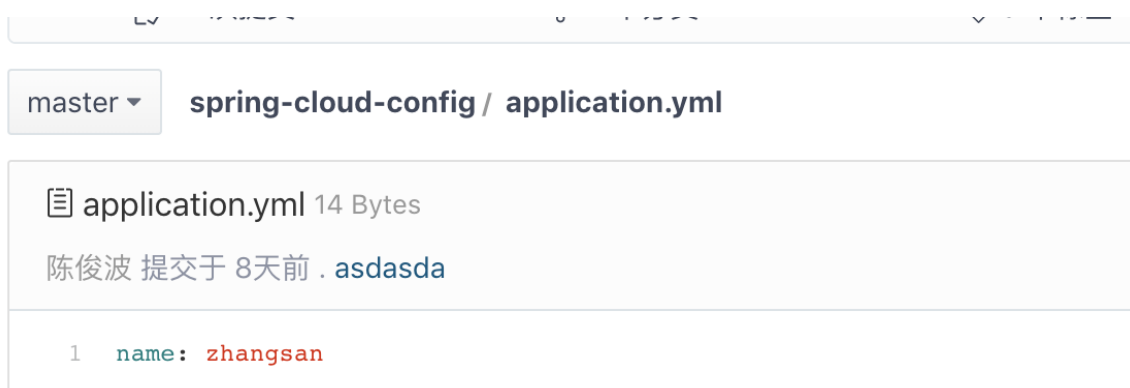
    public static void main (String[] args){
        SpringApplication.run(ConfigServerStartApp.class,args);
    }
}
```

3 启动程序

4 上传配置文件并测试

4.1 上传文件

在 git 仓库中上传了一个 application.yml 文件, properties 文件也可以



4.2 访问

此处我们先使用直接访问的方式测试,并没有使用 configclient

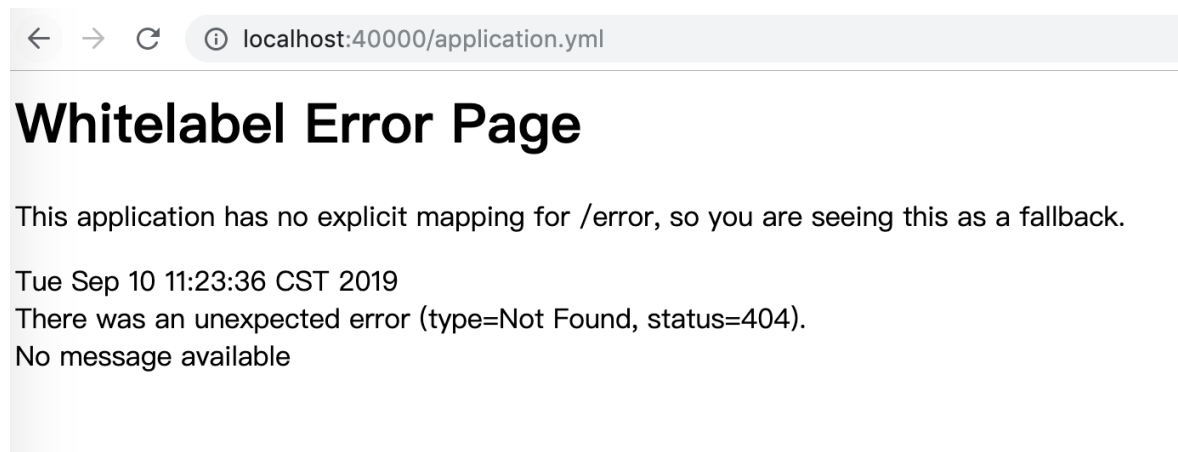
官方文档中给我们列举的通过 config server 访问 git 仓库的方式

The HTTP service has resources in the following form:

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

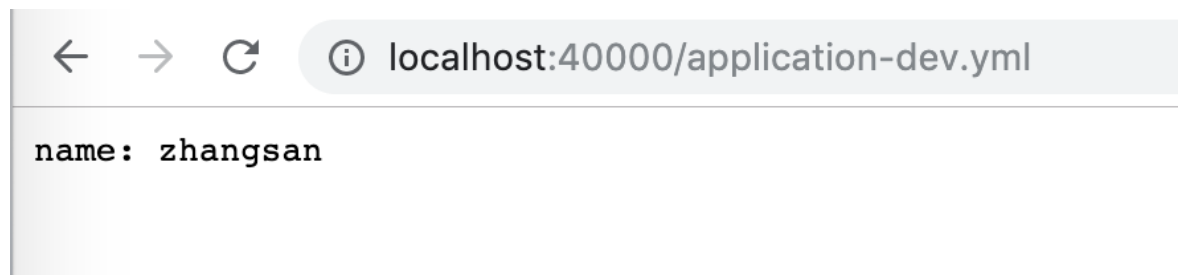
其中application代表应用的名字,profile代表我们的配置文件名字,label代表是分支git 默认是 master,以上方式的默认意思我们的文件名字要匹配上面的application-profile.yml或者application-profile.properties格式,当使用/{label}/{application}-{profile}.yml或者/{label}/{application}-{profile}.properties的时候 label 默认可以忽略,实际访问就是通过访问对应的文件名进行访问

但是当我们访问 <http://localhost:40000/application.yml> 的时候发现出现了404问题



原因是我们以上的访问路径不匹配 `config` 给我们的规则, 所以出现了 404

修改访问地址为 <http://localhost:40000/application-dev.yml>, 我们发现可以获取数据, 此处的 `dev` 随便写的



实际上上面的地址并没有找到任何内容, `config` 有个默认规则就是当我们访问的地址无法找到内容的时候会查找仓库目录下是否有 `application.yml` 或者 `application.properties`, 有的话获取里面多内容并返回, 所以此处的数据实际上是默认数据,

4.3 上传新文件

重新在仓库中上传了一个文件 `cacheservice-dev.yml`

spring-cloud-config

📄 17 次提交

🔗 1 个分支

🏷 0 个标签

master ▾

spring-cloud-config / `cacheservice-dev.yml`

📄 cacheservice-dev.yml 22 Bytes

陈俊波 提交于 8天前 · dasdasd

```
1 redis: redis.qfjava.cn
```

4.4 访问

访问 <http://localhost:40000/cacheservice-dev.yml>

← → ↻ ⓘ localhost:40000/cacheservice-dev.yml

```
name: zhangsan
redis: redis.qfjava.cn
```

发现上面返回的结果,除了`cacheservice-dev.yml`中的内容外还有 `application.yml` 中的内容,是因为这个`application.yml`默认配置文件中的内容总是会被获取,但是如果目标文件和默认文件中有重复的属性,的时候只会返回目标文件的,如果默认文件中有目标文件中没有的属性就会一起返回这些默认属性,

5 分析

经过上面的测试我们可以发现,实际上我们上传到文件的名称要和访问的路径名称保持一致,所以我们的主要规则就是怎么定义文件的名称,既要符合规则,又要保证不同项目的唯一性,所以 **cloud** 建议我们使用应用程序名称-文件属性级别,如 `userservice-dev.yml` `orderservice-test.yml` 等

三 Config Client

上面我们是直接通过浏览器进行测试的,实际开发中我们需要通过 client 进行访问获取数据

3.1 pom中依赖

```
<dependencies>

    <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>
</dependencies>
```

3.2 controller

此 controller 中我们有一个变量 `name` 通过 spring 注入值,然后有一个方法获取数据,实际开发中我们肯定需要在配置文件中给这个属性赋值,此处我们期望通过我们的 config server 从 git 中获取数据

```

/**
 * Created by jackiechan on 2019-09-02 11:33
 *
 * @Author jackiechan
 */
@RestController
public class TestController {
    @Value("${spring.name}")
    private String name;

    @RequestMapping("/test")
    public String getName() {
        return name;
    }
}

```

3.3 bootstrap.yml

因为application.yml是项目启动后在初始化数据的时候加载的,但是我们在初始化数据的时候必须已经从 server 中获取数据,所以我們必須在初始化之前去加载数据, springboot 提供了一个 bootstrap.yml 用于在初始化之前进行解析,其加载顺序会比application.yml早,所以我们可以将部分配置放到这里面来

```

spring:
  cloud:
    config:
      uri: http://localhost:40000 #配置 configserver 的地址,不是仓库地址
  application:
    name: testconfig #我们的 config client 查找文件的方式是 根据应用的名字加上active的值来匹配的,比如此处代表要找的文件叫testconfig-pro.yml
  profiles:
    active: pro

```

3.4 application.yml

```

server:
  port: 41000

```

3.5 主程序

```

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * Created by jackiechan on 2019-09-02 11:35
 *

```

```

* @Author jackiechan
*/
@SpringBootApplication
public class ConfigClientStartApp {
    public static void main (String[] args){
        SpringApplication.run(ConfigClientStartApp.class,args);
    }
}

```

3.6 上传文件

在 git 仓库中上传 testconfig-pro.yml

```

spring:
  name: gkfdgjgkhgdfhglhjd fhjsdfhkskj6666666666

```

3.7 启动测试

看启动日志可以看到从 <http://localhost:40000> 上拉取数据, 并且项目没有报错找不到数据,说明找到了数据,访问我们上面的 <http://localhost:41000/test> 发现返回值就是我们从 git上面获取到的值

```

2019-09-10 16:29:35.939 INFO 20621 --- [          main]
c.c.c.ConfigServicePropertySourceLocator : Fetching config from server at :
http://localhost:40000
2019-09-10 16:29:37.990 INFO 20621 --- [          main]
c.c.c.ConfigServicePropertySourceLocator : Located environment: name=testconfig,
profiles=[pro], label=null, version=f3465c0c2dd451ef24f2ffb03d64225edc0d7c96,
state=null
2019-09-10 16:29:37.991 INFO 20621 --- [          main]
b.c.PropertySourceBootstrapConfiguration : Located property source:
CompositePropertySource {name='configService', propertySources=[MapPropertySource
{name='configClient'}, MapPropertySource {name='https://gitee.com/mrjackiechan/spring-
cloud-config/testconfig-pro.yml'}, MapPropertySource
{name='https://gitee.com/mrjackiechan/spring-cloud-config/application.yml'}}}
2019-09-10 16:29:37.998 INFO 20621 --- [          main]
c.q.m.c.c.c.ConfigClientStartApp          : The following profiles are active: pro
2019-09-10 16:29:38.017 INFO 20621 --- [          main]
ConfigServletWebServerApplicationContext : Refreshing

```

← → ↻ ⓘ localhost:41000/test

gkfdgjgkhgdfhglhjd fhjsdfhkskj6666666666

3.8 分析

只要我们仓库中的文件名字和我们 `client` 中的 `application.name` 的值-`profiles.active` 的值一样,那么我们就可以获取到,只要保证这个规范 后面就可以获取数据

四 其他访问方式

此处不再使用 `client` 进行访问获取数据,而是直接通过浏览器请求测试的方式来演示,使用方式只有 `server` 的配置文件不同,所以下面只配置配置文件

4.1 搜索路径的方式

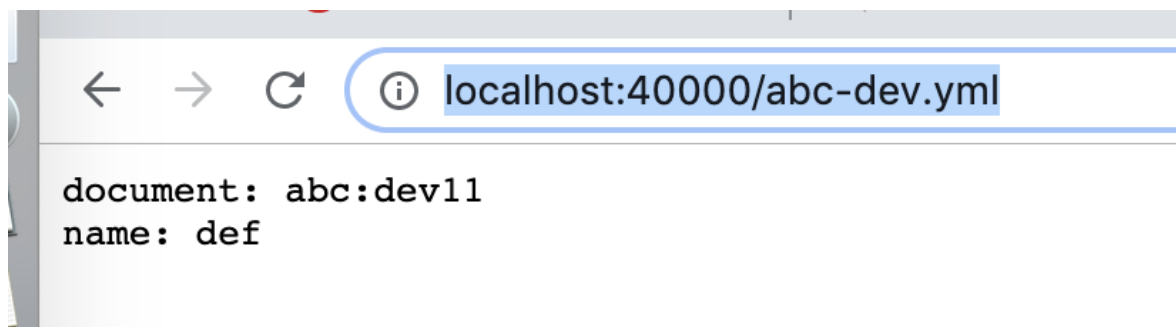
```
server:
  port: 40000
spring:
  application:
    name: 28cloud-config-server
  cloud:
    config:
      server:
        git: #配置configserver从什么地方加载配置文件
          uri: https://gitee.com/mrjackiechan/spring-cloud-config/
          search-paths:
            - abc #指的是请求以abc-开头的文件的时候会转达abc这个目录并且加载指定名字的文件,比如
                  我们的文件在 abc 目录下叫 abc-dev.yml 的文件,那么当我们访问 abc-dev.yml 的时候会被转到当前目录并
                  查找 abc-dev.yml,注意 abc 目录下的文件必须都是以 abc 开头的,这样我们可以通过不同的目录来区分不同的
                  微服务
            - def
```

在仓库中创建了一个 `abc` 目录,并在里面上传了 `abc-dev.yml` 文件



启动测试

启动后访问 <http://localhost:40000/abc-dev.yml>



4.2 模式匹配模式

在搜索模式下,我们同的服务可以将文件放在一个仓库下的不同目录下,我们还有一种方式是不同的服务可以放在不同的仓库下

```
server:
  port: 40000
spring:
  application:
    name: 28cloud-config-server
cloud:
  config:
    server:
      git: #配置configserver从什么地方加载配置文件
        uri: https://gitee.com/mrjackiechan/spring-cloud-config #默认地址
      repos: #配置不通的项目的不同仓库
        simple:
          uri: https://gitee.com/mrjackiechan/simple #以simple开头的地址会转到以下仓库
        special:
          pattern: special*/dev*,special*/test* #代表以 special开头并且匹配当前规则的地址会去这个仓库,否则会去默认仓库
          uri: https://gitee.com/mrjackiechan/special
```

上传文件

在不同的仓库中上传不同的文件



📄 11 次提交

🔗 1 个分支

🔒 0

master ▾

simple / simple-dev.properties properties 和 yml 一样

📄 simple-dev.properties 33 Bytes

陈俊波 提交于 2月前 · 更新 simple-dev.properties

```
1 abc=zhangsan
2 profile=simple-dev-7
```

 陈俊波 / special  代码

🔔 Issues 0

🔗 Pull Requests 0

📄

暂无描述

📄 3 次提交

🔗 1 个分支

master ▾

special / special-dev.yml

📄 special-dev.yml 11 Bytes

陈俊波 提交于 6分钟前 · dadsa

```
1 special:dev
```

</> 陈俊波 / special 

</> 代码

🔔 Issues 0

🔗 Pull Requests 0

📎 附件 0

📄

暂无描述

📄 3 次提交

🔗 1 个分支

🔖 0

master ▾

special / special-pro.yml

📄 special-pro.yml 11 Bytes

陈俊波 提交于 6分钟前 · dasdasd

1 special:pro

访问测试

访问 <http://localhost:40000/simple-dev.yml> 和 <http://localhost:40000/simple-dasda.yml> 发现获取到的 simple 仓库中的内容

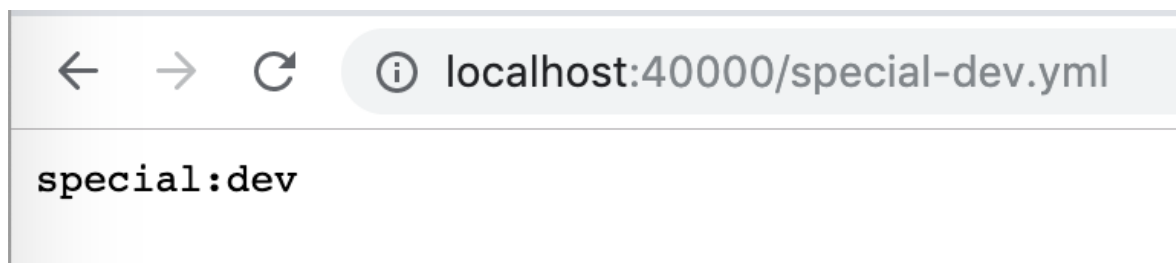
← → ↻ ⓘ localhost:40000/simple-dev.yml

```
abc: zhangsan
profile: simple-dev-7
```

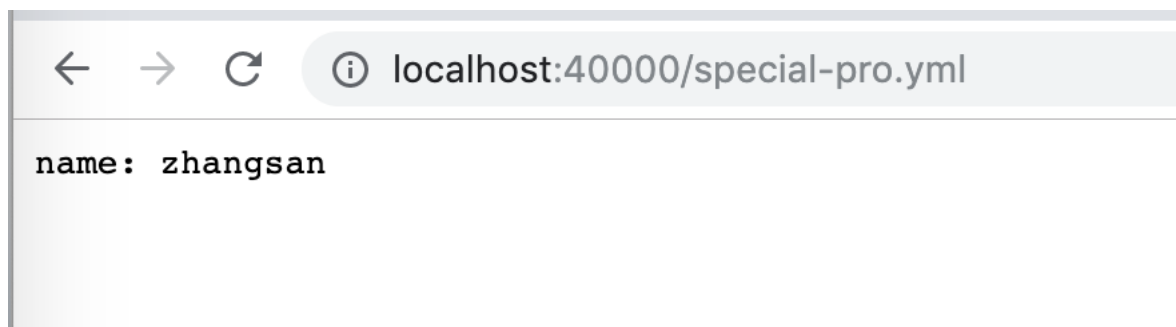
← → ↻ ⓘ localhost:40000/simple-dasda.yml

```
simple:dasda
```

访问 <http://localhost:40000/special-dev.yml> 发现获取到的是 special 仓库中的对应内容



访问 <http://localhost:40000/special-pro.yml> 发现获取到的是默认仓库中默认文件的内容,因为 special-pro.yml 不符合 special 仓库的匹配要求



4.3 通配符模式

上面的方式我们需要给每个服务单独配置仓库,我们的微服务特别多的时候,并且会随时添加,这样 server 就需要不断修改,所以我们可以使用通配符的方式,也就是我们定制一个规则,比如服务的名字就是仓库的名字,这样子以服务名字开头的地址会自动分发到对应的仓库中

yml 文件

```
server:
  port: 40000
spring:
  application:
    name: 28cloud-config-server
  cloud:
    config:
      server:
        git: #配置configserver从什么地方加载配置文件
          uri: https://gitee.com/mrjackiechan/{application} #application取的是
application的名字
```

以上方式和模式匹配模式是一样的访问方式,只不过由手动配置对应的仓库变成了自动匹配对应的仓库

五 整合Eureka

在上面的 config client 例子中,我们的 config server 地址是写死的单机地址,如果 server 宕机的话我们就无法获取数据了,所以我们的 server 必须是集群保证高可用,如果是集群的话,我们就需要注册中心来帮我们进行服务发现和负载均衡,所以我们需要将 server 和 client 都注册到 eureka 上面,client中配置好 server 在 eureka 中的名字

5.1 server 整合 eureka

5.1.1 pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
    </dependency>
<!--
如果访问 server 不需要密码就不用添加这个依赖
-->

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

    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
</dependencies>
```

5.1.2 application.yml

```
server:
  port: 40000
spring:
  application:
    name: 31cloud-config-server-atuh-eureka
  security: #配置访问 config server 的账号和密码,如果不需要就不用配置
    user:
      password: config
      name: cloud #默认是user
  cloud:
    config:
      server:
        git: #配置configserver从什么地方加载配置文件
          uri: https://gitee.com/mrjackiechan/{application}

eureka: #注册中心的地址
  client:
    service-url:
      defaultZone: http://zhangsan:abc@localhost:10000/eureka #curl风格
```

5.1.3 主程序

```
@SpringBootApplication
@EnableEurekaClient //开启服务发现
@EnableConfigServer //开启配置 server
public class ConfigServerStartApp {
    public static void main (String[] args){
        SpringApplication.run(ConfigServerStartApp.class,args);
    }
}
```

5.2 client 整合 eureka

因为我们的 client 就是我们的服务,本身就在 eureka上, 所以配置和之前一样,只需要额外添加配置告诉 client 我们的 server 在 eureka 中的注册 id

5.2.1 pom.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.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

5.2.2 bootstrap.yml

注意 和configserver 相关的配置和依赖性配置必须放在 bootstrap.yml 中,其他非依赖性的配置可以放在 application.yml中,如端口等

```
#这个文件是启动的时候就加载, 优先级高于application.yml
spring:
  application:
    name: simple #这个就是我们的名字, 当前值和下面的 profile 组合起来代表我们的访问路径, 比如此处访问的就是 simple-dev.yml
  cloud:
    config: #配置configserver的地址
    profile: dev
    label: master
    discovery: ##服务发现
```

```

    enabled: true ##允许服务发现
    service-id: 31cloud-config-server-atuh-eureka #配置configserver在eureka上面的名字
    #配置config server 的用户名和密码,如果 server 那边没配置,就不需要下面的配置
    username: cloud
    password: config
eureka: #注册中心的地址,必须配置在这个文件里面
  client:
    service-url:
      defaultZone: http://zhagsan:abc@localhost:10000/eureka #curl风格

```

application.yml

```

server:
  port: 19245

```

5.2.3 controller

和上面的客户端一样,想要获取spring.aaaa这个属性

```

@RestController
@RefreshScope//当当前类找到某些内容发生变化的时候,会刷新属性,(必须是RefreshScope知道属性发生变化)
public class ClientController {
    @Value("${spring.aaaa}")
    private String aaaa;

    @GetMapping("/aaa")
    public String getAaaa() {
        return aaaa;
    }
}

```

5.2.4 主程序

```

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

```

5.2.5 修改 git 中的配置文件

在我们的 simple 仓库中有一个 simple-dev.properties 文件,我们的项目访问的就是这个文件,在里面把我们的内容放进去

master ▾

simple / simple-dev.properties

simple-dev.properties 50 Bytes

陈俊波 提交于 刚刚 · fsdfs

```
1 abc=zhangsan
2 profile=simple-dev-7
3 spring.aaaa=lisi
```

5.2.6 启动测试

启动我们的项目,按照流程,我们的 client 会向 configserver 发起请求获取数据,请求的地址 simple-dev.yml(properties),然后我们的 config server 会根据它的配置去我们的 git 中找对应的文件

启动后访问我们的 <http://localhost:19245/aaa> 可以获取到我们的数据



不安全 | 192.168.1.147:19245/aaa

lisi

六 数据刷新

经过上面的配置,我们的服务已经实现了从 git 中获取数据的功能,但是测试后发现,当我们到数据后,如果修改了 git 里面的内容,我们期望的是服务这边也会变,但是经过实际测试,修改了 git 后我们的服务中的数据并不会变,但是我们通过 config server 再访问发现 configserver 中的数据已经变化的

暂无描述

📄 14 次提交

🔗 1 个分支

master ▼

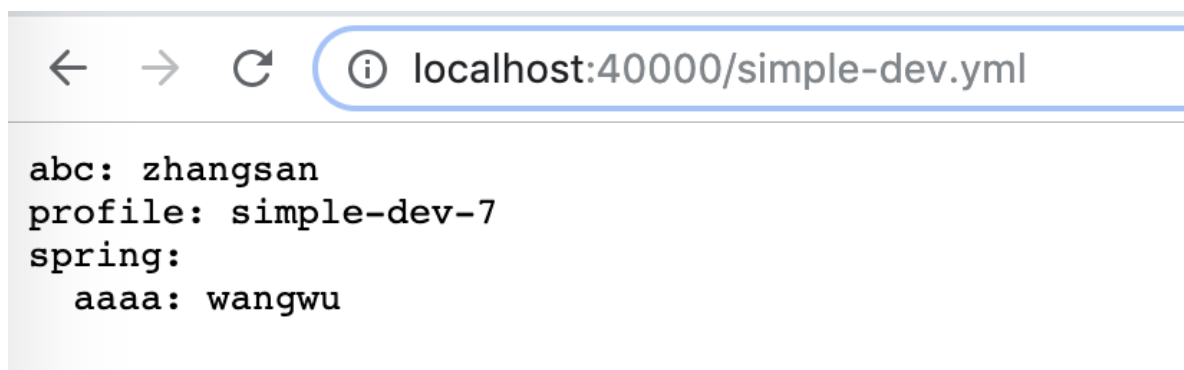
simple / simple-dev.properties

📄 simple-dev.properties 52 Bytes

陈俊波 提交于 刚刚 · fsdfs

```
1 abc=zhangsan
2 profile=simple-dev-7
3 spring.aaaa=wangwu 修改为王五
```

通过configserver 访问发现已经变化



刷新客户端发现还是之前的lisi



要想变为王五,得重启 client 才可以获取到最新数据, config server每次都是去 git 上获取最新数据,但是 client 从 server 获取后会缓存到本地,所以不会再去重新获取数据,导致数据出现不一致的情况

6.1 手动刷新

我们每次提交数据后,都应该要刷新一下 client,让它重新从 server 上面获取一次数据

为了让我们刷新数据,springcloud 给我们提供了一个注解叫@RefreshScope ,被这个注解修饰的类,当数据发生变化的时候会刷新重新获取数据

2.9 Refresh Scope

When there is a configuration change, a Spring `@Bean` that is marked as `@RefreshScope` gets special treatment. This feature addresses the problem of stateful beans that only get their configuration injected when they are initialized. For instance, if a `DataSource` has open connections when the database URL is changed via the `Environment`, you probably want the holders of those connections to be able to complete what they are doing. Then, the next time something borrows a connection from the pool, it gets one with the new URL.

Sometimes, it might even be mandatory to apply the `@RefreshScope` annotation on some beans which can be only initialized once. If a bean is "immutable", you will have to either annotate the bean with `@RefreshScope` or specify the classname under the property key `spring.cloud.refresh.extra-refreshable`.

`Refresh` scope beans are lazy proxies that initialize when they are used (that is, when a method is called), and the scope acts as a cache of initialized values. To force a bean to re-initialize on the next method call, you must invalidate its cache entry.

The `RefreshScope` is a bean in the context and has a public `refreshAll()` method to `refresh` all beans in the scope by clearing the target cache. The `/refresh` endpoint exposes this functionality (over HTTP or JMX). To `refresh` an individual bean by name, there is also a `refresh(String)` method.

上面的意思就是@RefreshScope可以给我们刷新数据,它给我们提供了一个接口地址,当我们访问这个地址的时候,会触发刷新操作,那就是说,我们只要提交数据到 `git`,然后请求刷新的地址,这样数据就刷新了

6.1.1 修改 controller

对于 client 来说我们只需要将会发生数据变化的 controller 添加@RefreshScope注解即可

```
@RestController
@RefreshScope//当前类找到某些内容发生变化的时候,会刷新属性,(必须是RefreshScope知道属性发生变化)
public class ClientController {
    @Value("${spring.aaaa}")
    private String aaaa;

    @GetMapping("/aaa")
    public String getAaaa() {
        return aaaa;
    }
}
```

6.1.2 修改数据访问刷新地址

在 spring cloud E 版本以及之前,刷新地址为当前项目服务器地址下的/refresh地址

F 版本开始,地址被修改为/actuator/refresh,但是这个地址默认是不允许访问的,要想访问这个地址,我们需要添加一个依赖包和修改配置文件允许访问这个地址

§ 2.11 Endpoints

For a Spring Boot Actuator application, some additional management endpoints are available. You can use:

- `POST` to `/actuator/env` to update the `Environment` and rebind `@ConfigurationProperties` and log levels.
- `/actuator/refresh` to re-load the boot strap context and `refresh` the `@RefreshScope` beans. F 版本修改为当前地址
- `/actuator/restart` to close the `ApplicationContext` and restart it (disabled by default).
- `/actuator/pause` and `/actuator/resume` for calling the `Lifecycle` methods (`stop()` and `start()` on the `ApplicationContext`).

6.1.3 添加依赖

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

6.1.4 修改配置文件

我们 springcloud 的所以管理地址都是通过上面的依赖来进行映射的,但是默认情况下都不允许访问,我们需要在 application.yml中进行配置

```
management:
  endpoints:
    web:
      exposure:
        include: '*' #允许访问所有管理地址,不然无法访问刷新地址,如果指向访问某个管理地址,写具体地址就行,比如 '/actuator/refresh'
```

6.1.5 测试

启动我们的项目后,访问我们的 controller,发现获取到的最新数据,然后修改我们的 git 数据内容,然后刷新,发现没有变化

这时候我们通过 **POST** 请求访问我们的 client 下面的 <http://localhost:14295/actuator/refresh> 然后我们再回来刷新 controller 页面,发现数据变化为最新的

6.2 自动刷新

上面我们实现了刷新功能,但是经过测试发现,上面的方式只能刷新单个服务器,我们的 client 都是集群,如果要刷新所有的 client,我们需要手动挨个访问所有的 client 才可以变化过来,这样的效率非常慢,而且所有的 client 到底有多少服务器我们也不清楚,所以我们启动的时候能自动刷新

springcloud 支持我们使用 事件的方式通知所有客户端刷新,可以使用 MQ 来完成这个功能,我们只要让所有的 client 整合 MQ,然后通过发送 MQ 消息来实现通知所有客户端刷新

6.2.1 pom 依赖

我们只需要给 client 添加 mq 依赖即可

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

6.2.2 配置 MQ 地址

mq 的配置可以写在 bootstrap 也可以写在 application 中,因为它是程序运行后用来接收或者发送消息的,演示中放到了 bootstrap.yml 中

```
#这个文件是启动时候就加载,优先级高于application.yml
spring:
```

```

application:
  name: simple
rabbitmq: #配置 MQ 的地址,用于首发自动更新的消息
  host: rabbitmq.qfjava.cn
  port: 8800
  username: guest
  password: guest
cloud:
  config: #配置configserver的地址
  profile: dev
  label: master
  discovery:
    enabled: true
    service-id: 31cloud-config-server-atuh-eureka #配置configserver在eureka上面的名字
    #配置config server 的用户名和密码
    username: cloud
    password: config
management:
  endpoints:
    web:
      exposure:
        include: '*' #允许访问所有管理地址,不然无法访问刷新地址,如果指向访问某个管理地址,写具体地址就行,比如 '/actuator/bus-refresh'
eureka: #注册中心的地址
  client:
    service-url:
      defaultZone: http://zhangsan:abc@localhost:10000/eureka #curl风格

```

6.2.3 访问地址

在自动刷新的方式下,访问地址变成了/actuator/bus-refresh,这个地址也需要导入actuator和配置上面的内容

6.2.4 测试

启动多个client,然后访问数据,可以获取到我们的数据,然后修改 git 中的数据,再通过 POST 来访问任意一个 client 的服务器来访问/actuator/bus-refresh这个地址,然后刷新所有 client 的 controller 发现数据已经变成最新的了

6.2.5 自动刷新

通过 mq 可以实现自动刷新,但是我们仍然要手动访问/actuator/bus-refresh地址,实际开发中,我们的服务器都是可以通过公网访问的,在 git 中给我们提供了一个 webhooks,可以在我们提交代码后自动请求我们指定的地址来刷新数据

陈俊波 / simple

Unwatch1Star0Fork0

代码

Issues0

Pull Requests0

附件0

Wiki0

统计

DevOps

服务

管理

暂无描述

14次提交1个分支0个标签0个发行版1位贡献者

master

+ Pull Request

+ Issue

文件

Web IDE

挂件

克隆/下载

陈俊波 最后提交于 3小时前 fsdfs

README.en.md

Initial commit

2月前

README.md

Initial commit

2月前

simple-dasda.yml

dasda

5小时前

simple-dev.properties

fsdfs

3小时前

README.md

陈俊波 / simple

Unwatch1Star0Fork0

代码

Issues0

Pull Requests0

附件0

Wiki0

统计

DevOps

服务

管理

- 仓库设置
- 基本设置
- 转移仓库
- 清空仓库
- 存储库 GC
- 删除仓库
- 代码审查设置
- 仓库成员管理
- 部署公钥管理
- 集成与服务
- WebHooks
- 只读文件管理New

基本设置

仓库名称

simple

路径 (请注意: 仓库路径即仓库访问 URL 地址, 更改仓库路径将导致原克隆地址不可用)

https://gitee.com/mrjackiechan/ simple

介绍(可选)

用简短的语言来描述一下吧

主页(可选)

输入一个有效的 http 链接

语言(可选)

请选择语言

状态

正在中

代码Issues0Pull Requests0附件0Wiki0统计DevOps服务管理

- 仓库设置
- 仓库成员管理
- 部署公钥管理
- 集成与服务
- WebHooks
- 只读文件管理New

WebHooks 设置

添加

每次您 push 代码后, 都会给远程 HTTP URL 发送一个 POST 请求 更多说明 »

WebHook 增加对钉钉的支持 更多说明 »

当前仓库没有使用 Webhook

添加 WebHook

URL：超时时长为5秒，长时间的操作建议您异步进行

密码：请求 URL 时会带上该密码，防止 URL 被恶意请求

URL：

POST 地址

公网地址

密码：

请求密钥

选择事件：

- ☒ Push 仓库推送代码、推送 / 删除分支
- ☐ Tag Push 新建 / 删除 tag
- ☐ Issue 新建任务、变更任务状态、更改任务指派人员
- ☐ Pull Request 新建、更新、合并、关闭 Pull Request
- ☐ 评论 评论仓库、任务、Pull Request、Commit

☒ 激活 （激活后事件触发时将发送请求）

添加

上图是码云的方式,其他的网站只是位置不一样,功能是一样的