SpringCloud之七

分布式配置中心SpringCloudConfig

一、Spring Cloud Config 介绍

微服务常用的招式已经有了：

- 注册中心（eureka, consul, zk, etcd）
- 配置中心 (Spring Cloud Config, disconf)
- API网关 (Spring Cloud zuul, kong)
- 熔断器（hystrix）

那么还需要什么呢？当然就是一个行走江湖，说走就走的理由了，俗话说的号，师出有名，接下来要介绍的就是，SpringCloud的配置中心解决方案——Spring Cloud Config。

在官网上截图如下：

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，SVN等仓库。

在Spring Cloud Config组件中，分两个角色，一是**config server**，二是**config client**

Spring Cloud Config，主要有以下**特性**

- 配置集中管理
- 依赖于git支持版本回退
- 可以通过git 仓库的WebHook和消息队列来实现自动更新

- 配置加密解密
- 可以通过调用接口手动更新配置

其他配置中心技术还有就是国内的disconf, disconf 可以为各种业务平台提供统一的**配置管理服务。**

- 支持配置（配置项+配置文件）的分布式化管理
- 配置发布统一化
- 极简的使用方式（注解式编程 或 XML代码无代码侵入模式）
- 低侵入性或无侵入性、强兼容性
- 需要Spring编程环境

虽然有很多选择，如果说是Spring Cloud来构建微服务，那么Spring Cloud Config肯定是比较好的一个选择，集成非常方便。

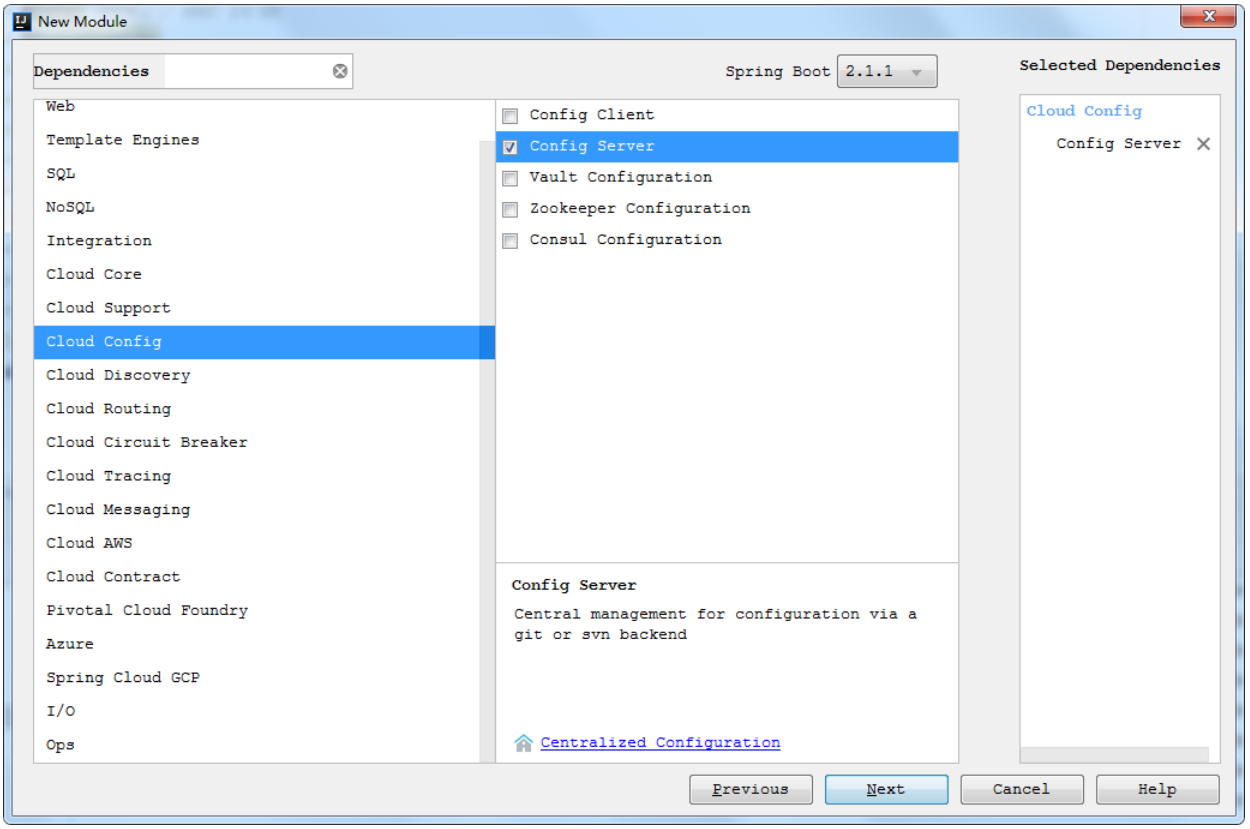
而且disconf虽然也是一个很好的作品，但是也是个人开源的，也不维护了，也许功能够用了。有时间得话，觉得可以个人再重新写一个，那样会更好。

二、构建Config Server

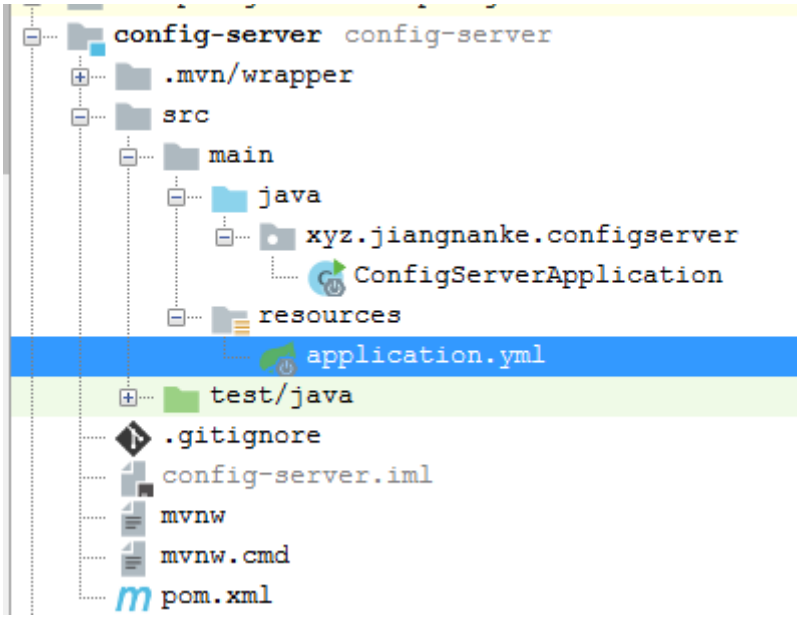
2.1、创建项目

创建一个Config server的spring-boot项目，我将之命名为 config-server。

如图：



创建好后，如图结构：



2.2、配置环境

配置pom.xml文件，当然也要修改main项目的pom.xml文件，添加上子模块（这里不多介绍），如下是config-server的pom.xml文件内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>xyz.jiangnanke</groupId>
 <artifactId>config-server</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <name>config-server</name>
 <description>Demo project for Spring Boot</description>

 <parent>
 <groupId>xyz.jiangnanke</groupId>
 <artifactId>main</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 </parent>

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

```
 </dependency>
 </dependencies>

</project>
```

然后再application启动类添加上@EnableConfigServer注解来启动config服务。

代码如下

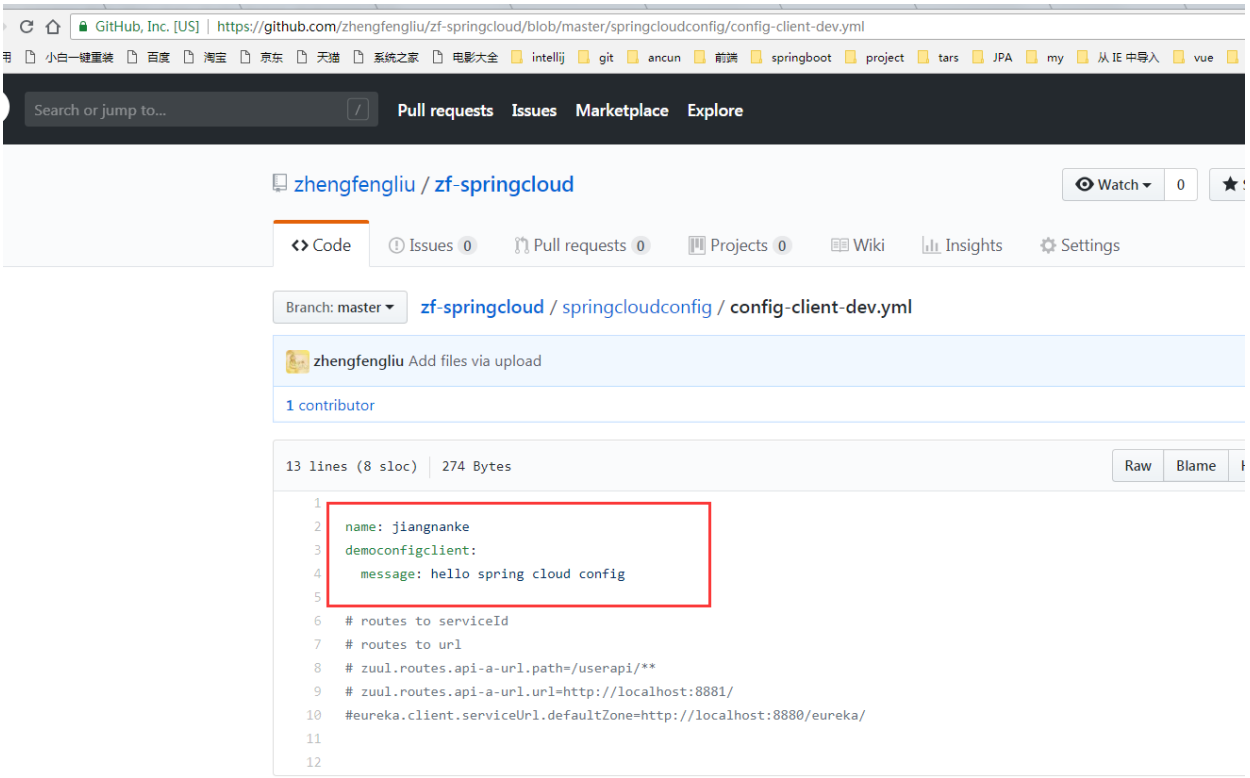
```
package xyz.jiangnanke.configserver;

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

@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {

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

}

```

再在application.yml配置文件添加如下内容：

```
spring:
 application:
 name: config-server
 cloud:
 config:
 server:
 git:
# 配置git仓库地址
 uri: https://github.com/zhengfengliu/zf-springcloud/
 # 配置仓库路径
 search-paths: springcloudconfig
 # 访问git仓库的用户名,密码
 username: zhengfeng
 password: zhengfeng
 # 配置仓库的分支
 label: master
```

```
17 # 如果Git仓库为公开仓库，可以不填写用户名和密码，如果是私有仓库需要填写，本例子是公开仓库，放心使用。
18
19 server:
20  port: 10089
21
```

然后再配置git里面添加上：

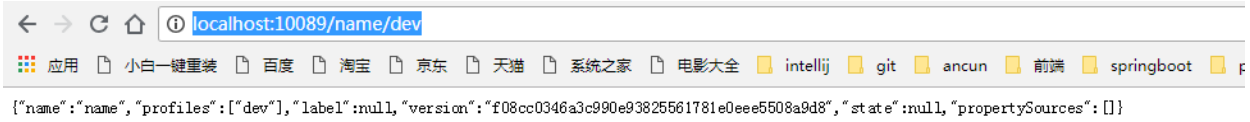


2.3、启动运行

启动项目，由于远程仓库git上config-client-dev.yml上有内容，接下来访问：

http://localhost:10089/name/dev

得到结果如下图：



证明配置服务中心可以从远程程序获取配置信息。

当然**http请求地址和资源文件映射**如下:

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

/{application}-{profile}.yml

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

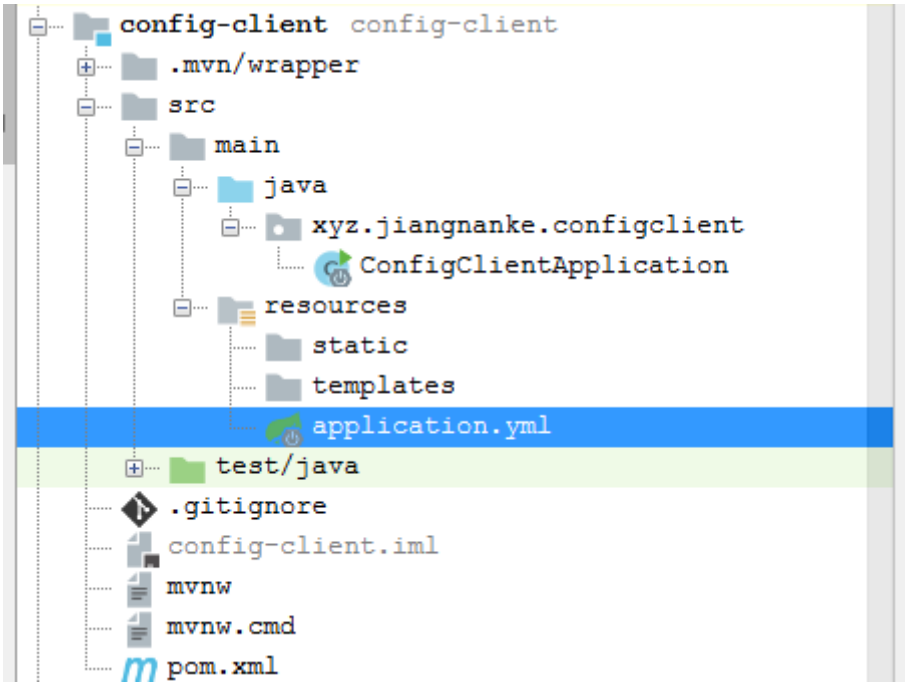
/{application}-{profile}.properties

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

三、构建Config Client

3.1、创建ConfigClient项目

创建一个Config Client的Spring-boot项目，我将之命名为config-client。如下：



其pom.xml配置文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>xyz.jiangnanke</groupId>
 <artifactId>config-client</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <name>config-client</name>
 <description>Demo project for Spring Boot</description>

 <parent>
 <groupId>xyz.jiangnanke</groupId>
 <artifactId>main</artifactId>
```

```
<version>0.0.1-SNAPSHOT</version>
</parent>

<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>

</project>
```

3.2、配置环境

首先配置application.yml文件，如下：

```
spring:
 application:
 name: config-client
 cloud:
 config:
# 指明远程仓库的分支
 label: master
# spring.cloud.config.profile
# dev开发环境配置文件
 # test测试环境
 # pro正式环境
  profile: dev
 # 指明配置服务中心的网址。
  uri: http://localhost:10089/
 server:
  port: 10090
```

接下来创建一个程序入口类LoginController.java。具体代码如下：

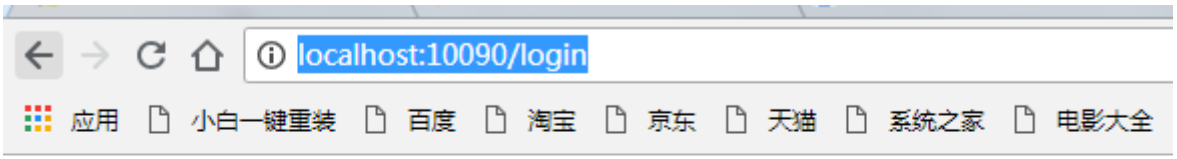
```
package xyz.jiangnanke.configclient.controller;

```

```java
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @Auther: zhengfeng
 * @Date: 2019\1\7 0007 17:32
 * @Description: 返回从配置中心读取的name变量的值
 */
@RestController
public class LoginController {

 @Value("${name}")
 String name;

 @RequestMapping(value = "/login")
 public String login(){
 return name;
 }
}
```

3.3、启动运行

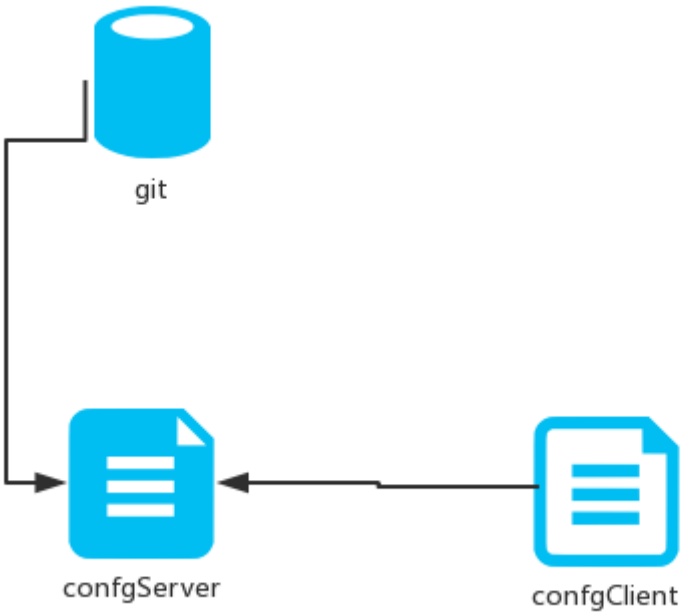
启动运行之后，访问网址：http://localhost:10090/login

得到结果，如图：

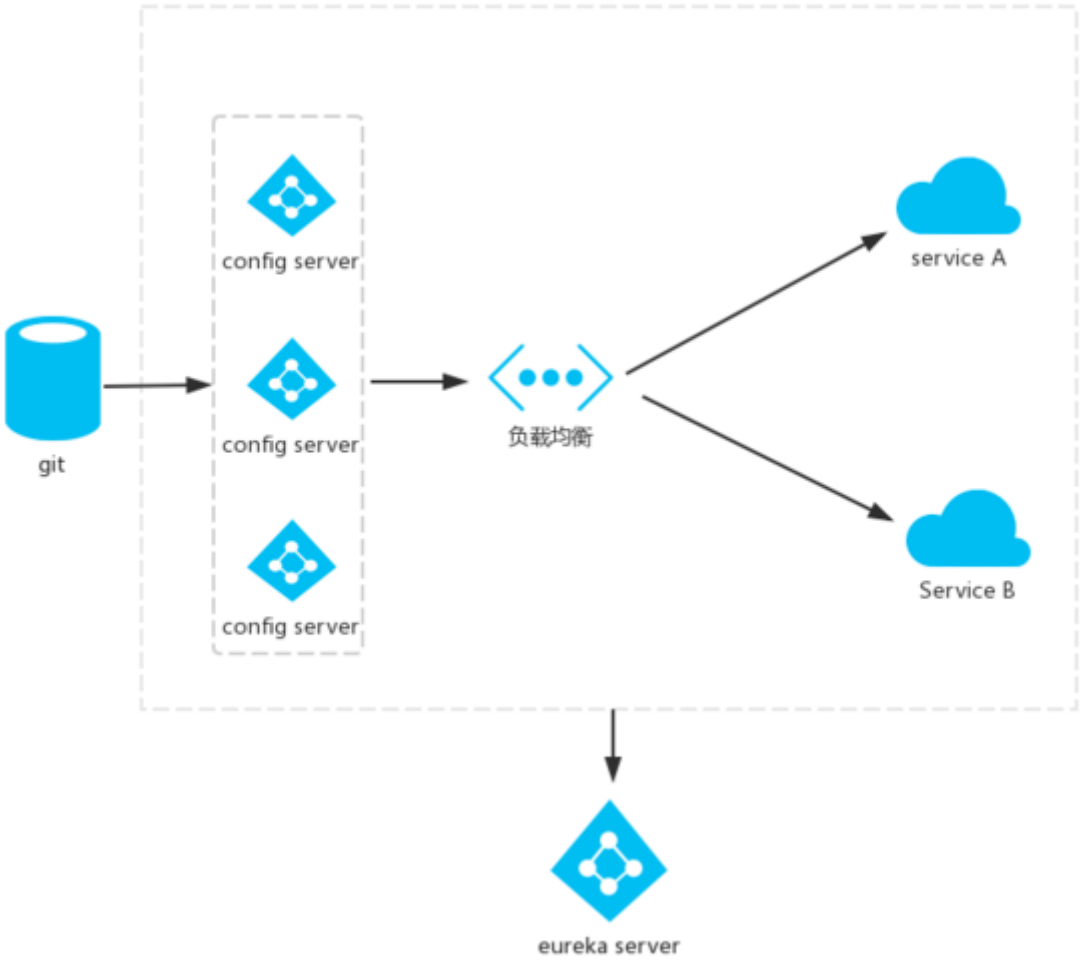


jiangnanke

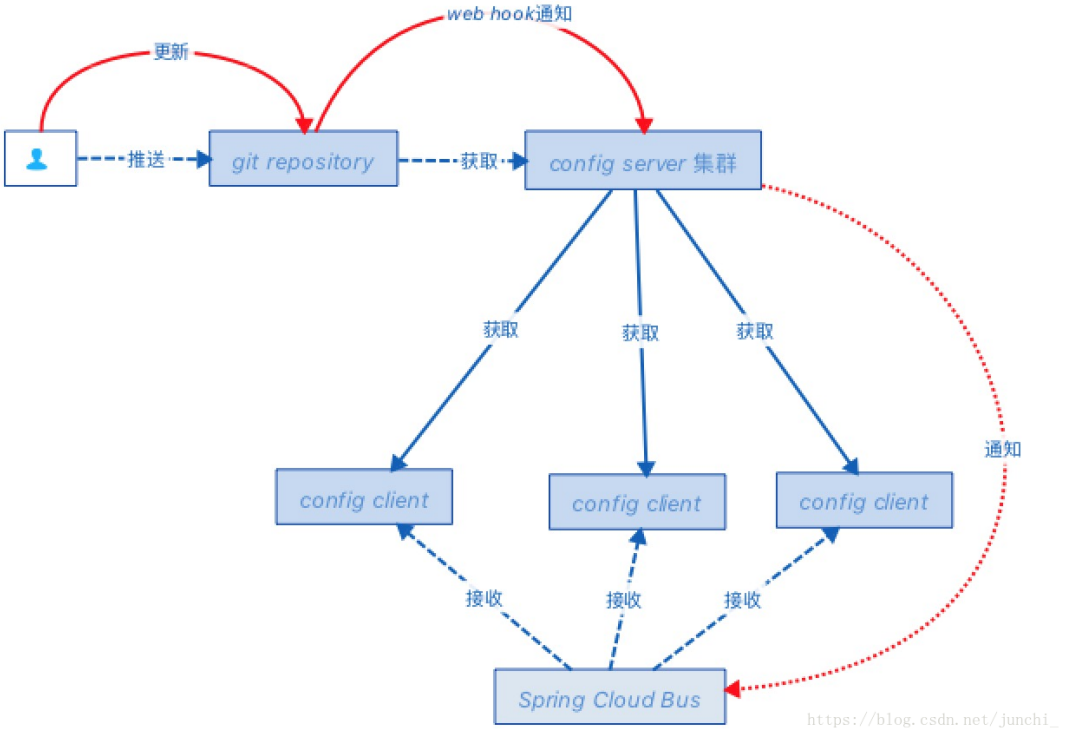
这就说明，config-client从config-server获取了 name 的属性，而config-server是从git仓库读取的,如图：



config-server在微服务中的定位：



config-client在微服务中的定位：

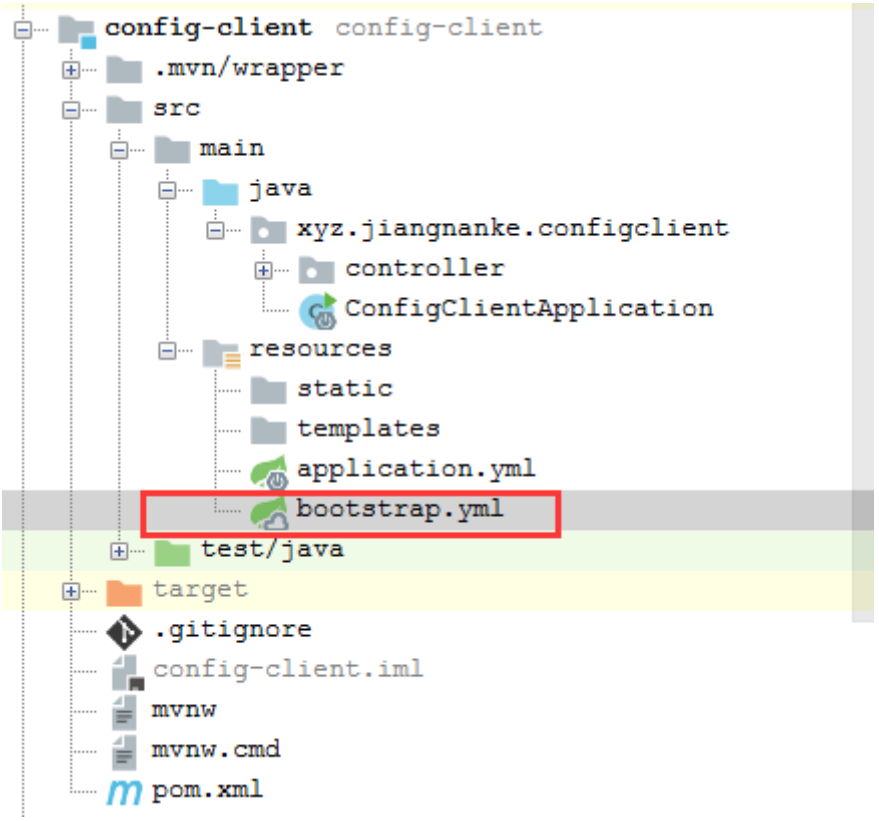


主要实现需要以下几个步骤：

1. 创建git仓库。用于存放配置文件。
2. 安装kafka，zookeeper。用于分布式项目统一刷新。
3. 创建ConfigServer服务，提供项目统一获取配置的中心服务。可配多个。
4. 创建ConfigClient服务，作为客户端。
5. 利用gitlab的webhook功能，每次push操作，发送post请求，通知刷新配置。

附录

一定要注意，config-client的配置信息是放在，bootstrap.yml文件中的或者bootstrap.properties文件中。不然启动会报错。修改之后的项目结构如图：



四、改造spring cloud config 与eureka结合

4.1、改造config-server项目

4.1.1、添加依赖

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

4.1.2、添加环境配置

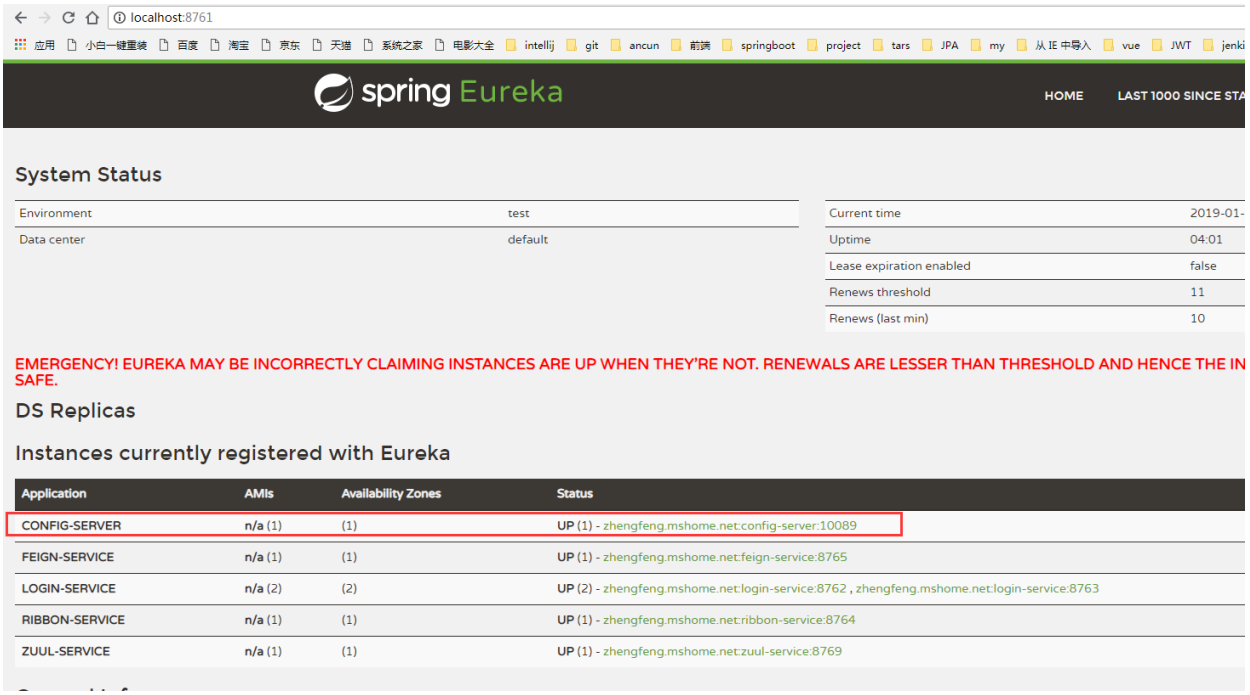
```
eureka:
 client:
 serviceUrl:
 defaultZone: http://localhost:8761/eureka/
```

4.1.3、添加启动配置

```
package xyz.jiangnanke.configserver;

import org.springframework.boot.SpringApplication;
```

```java
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableConfigServer
 @EnableEurekaClient
 public class ConfigServerApplication {

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

 }

```

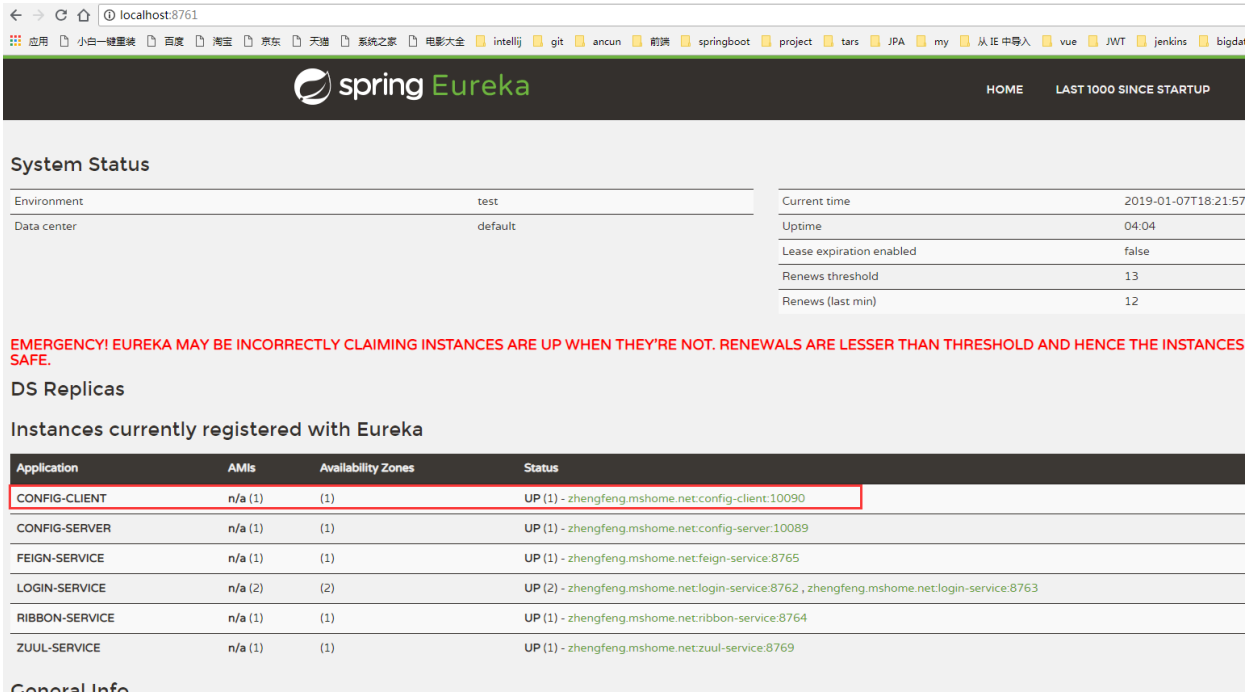
启动之后如图：



4.2、改造config-client项目

基本上如config-server一样修改

启动后如图：



当然如果可以的话，其实一开始创建项目就直接加上这些配置，就不用现在继续改造这个过程了。

参考资料

https://blog.csdn.net/liqi_q/article/details/81158002

https://springcloud.cc/spring-cloud-config.html

https://springcloud.cc/

https://blog.csdn.net/forezp/article/details/81041028

https://github.com/zhengfengliu/zf-springcloud

http://cxytiandi.com/blog/detail/12466

https://blog.csdn.net/forezp/article/details/81041045