# 分布式配置中心搭建

### 一．码云创建项目

#### 注册码云账号并登陆码云

#### **新建仓库**



#### **填写信息并创建**





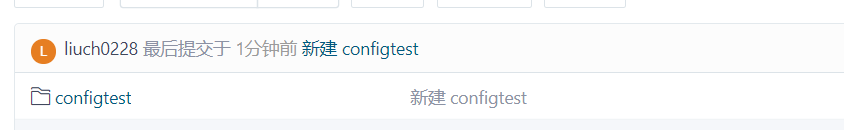
1. 新建存放配置文件的目录

新仓是空的，把仓库clone到本地，然后随便提交一个文件上去，然后新建文件夹——git环境上的文件根据项目进行区分，就是说，不要把不同项目的所有配置文件都放到同一个目录下



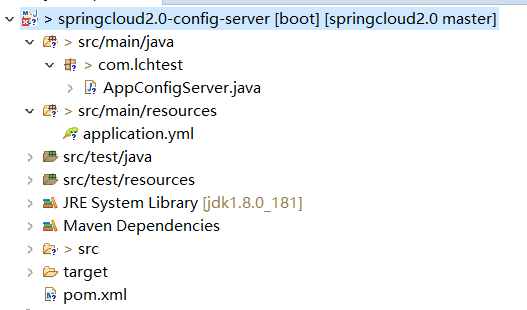
点击后输入文件夹名称然后提交：





### 二.搭建configserver

#### 创建一个configserver的maven项目



Pom依赖：

|  |
| --- |
| <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>com.lchtest</groupId>  <artifactId>springcloud2.0-config-server</artifactId>  <version>0.0.1-SNAPSHOT</version>    <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.3.RELEASE</version>  </parent>  <!-- 管理依赖 -->  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Finchley.RELEASE</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  <dependencies>  <!--spring-cloud 整合 config-server -->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-config-server</artifactId>  </dependency>  <!-- SpringBoot整合eureka客户端 -->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  </dependencies>  </project> |

#### **配置文件application.yml**

|  |
| --- |
| #服务注册到eureka地址  eureka:  client:  service-url:  defaultZone: http://localhost:8100/eureka  spring:  application:  #configserver在注册中心的应用名称  name: config-server  cloud:  config:  server:  git:  ###git环境地址  uri: https://gitee.com/liuch890228/distributed\_profile\_learning.git  ####搜索目录,自己在码云上的工程的仓库里面创建的目录名称  search-paths:  - configtest  #读取的配置的分支  label: master  # 码云创建仓库时选择的是公开，这里不用配置密码  #configserver服务的端口号  server:  port: 8888 |

#### **启动类**

启动类要加上**@EnableEurekaClient和@EnableConfigServer两个注解**

|  |
| --- |
| **package** com.lchtest;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.config.server.EnableConfigServer;  **import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;  /\*\*  \* 分布式配置中心的ConfigServer  \* **@author** pc  \* 1.@EnableConfigServer注解开启config server  \* 2.先启动eureka注册中心，再启动AppConfigServer  \* 3.如何把配置文件存放到git上:  \*/  @SpringBootApplication  @EnableEurekaClient  @EnableConfigServer  **public** **class** AppConfigServer {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(AppConfigServer.**class**, args);  }  } |

#### **码云新建配置文件**

点击这个目录名称，

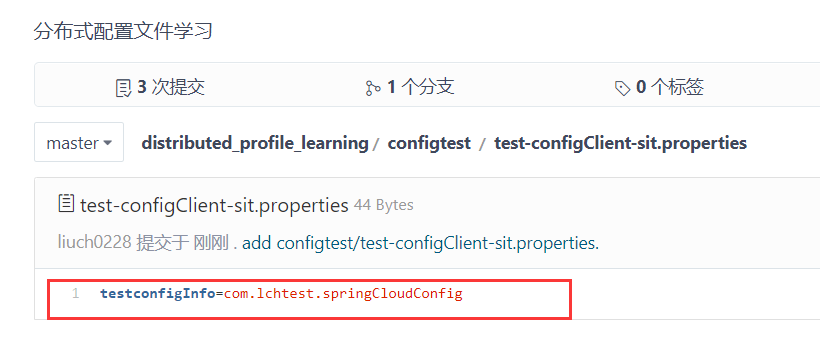




如 上图，点击新建文件，创建 一个配置文件test-configClient-sit.properties,输入配置并提交



提交结果如下：



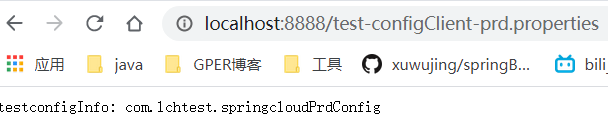
同理创建一个test-configClient-prd.properties配置文件，配置项相同，值不同：

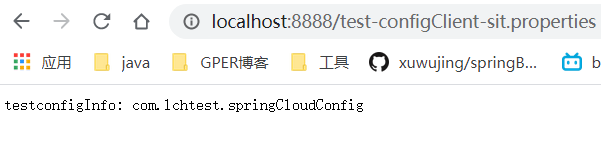


#### **访问配置文件**

依次启动eurekaServer和configServer，在浏览器输入访问

<http://localhost:8888/test-configClient-prd.properties>，可以读取到配置信息





### 三．Config客户端搭建

#### 创建configclient工程

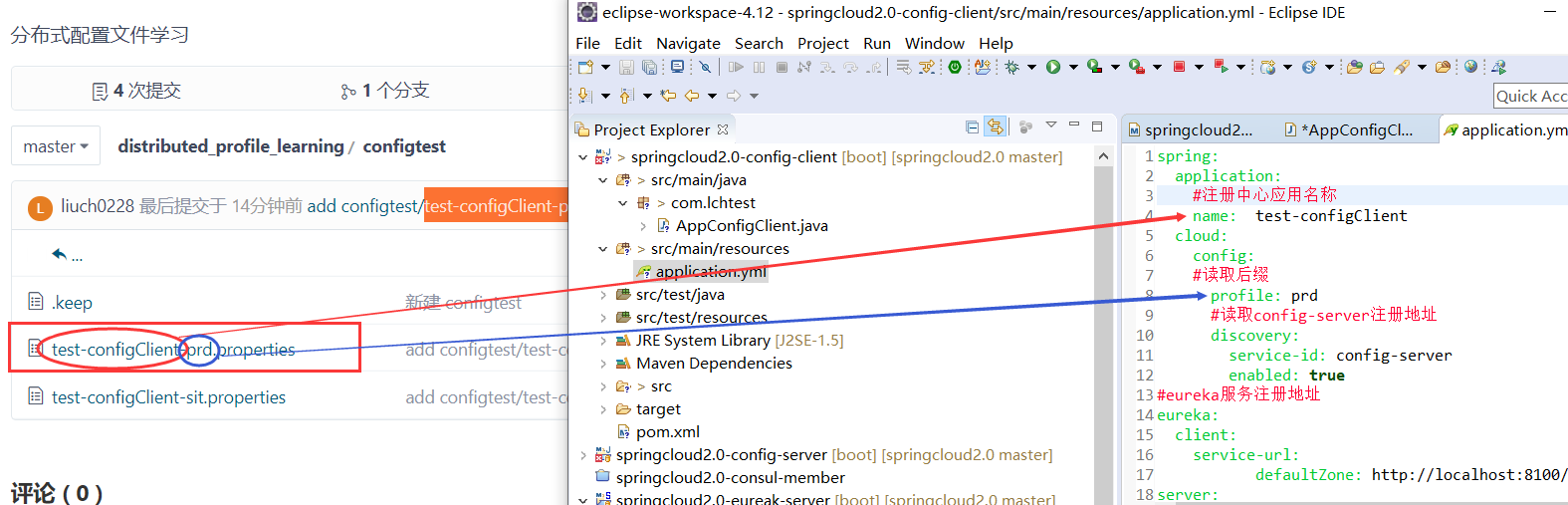
Pom依赖：

|  |
| --- |
| <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>com.lchtest</groupId>  <artifactId>springcloud2.0-config-client</artifactId>  <version>0.0.1-SNAPSHOT</version>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.3.RELEASE</version>  </parent>  <!-- 管理依赖 -->  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Finchley.RELEASE</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  <dependencies>  <!-- SpringBoot整合Web组件 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>    <!-- SpringBoot整合springcloud config客户端 -->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-config-client</artifactId>  </dependency>  <!-- SpringBoot整合eureka客户端 -->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  </dependencies>  </project> |

#### **配置文件：**

|  |
| --- |
| spring:  application:  #注册中心应用名称  name: test-configClient  cloud:  config:  #读取的版本，通过后缀确定是开发环境or测试环境or生产环境  profile: prd  discovery:  #读取config-server在注册中心上注册的别名  service-id: config-server  #开启读取权限  enabled: **true**  #eureka服务注册地址  eureka:  client:  service-url:  defaultZone: http://localhost:8100/eureka  server:  port: 8882 |

配置文件中，需要注意spring.application.name 值是码云上面的配置文件名称中的服务名，Spring.cloud.config.profile的值是码云上面配置文件的开发/测试/生产环境标识，如下图所示：



#### **启动类：**

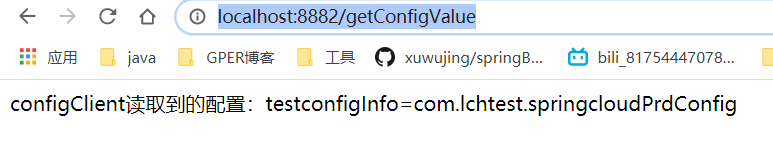
|  |
| --- |
| **package** com.lchtest;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;  @SpringBootApplication  @EnableEurekaClient  **public** **class** AppConfigClient {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(AppConfigClient.**class**, args);  }  } |

测试用的controller

|  |
| --- |
| **package** com.lchtest.controller;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  @RestController  **public** **class** TestController {  //从configserver读取配置  @Value("${testconfigInfo}")  **private** String testconfigInfo;    @RequestMapping("/getConfigValue")  **public** String getConfigValue() {  **return** "configClient读取到的配置：testconfigInfo=" + testconfigInfo;  }  } |

在启动eurekaServer和configServer项目的前提下，启动configClient项目，浏览器访问

<http://localhost:8882/getConfigValue> ，结果如下，顺利读取到配置项的值：



### 刷新配置文件

修改码云配置文件值



浏览器再去请求<http://localhost:8882/getConfigValue>，发现配置项还是原来的值

1 默认情况下，修改了配置文件的内容，再去调用访问 /getConfigValue接口，是不能获取到最新的配置项的值的，并且configclient也会有缓存，除非重启configclient服务；

1. SpringCloud分布式配置中心可以采用手动刷新或者自动刷新，两者都不需要重启服务器

手动刷新——需要人工调用接口，读取最新配置文件(监控中心)

自动刷新——消息总线进行实时通知(springbus)

在实际使用过程中，不建议使用自动刷新功能，对性能有影响，建议每次修改了配置之后，人工调用/actuator/refresh接口进行刷新

如何实现手动刷新：

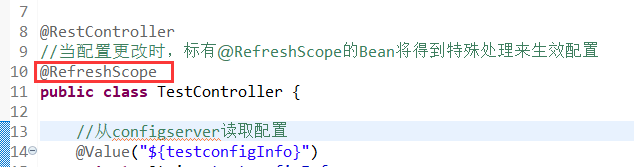
(1)Configclient工程pom加入actuator依赖：

|  |
| --- |
| <!-- SpringBoot整合actuator监控中心 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency> |

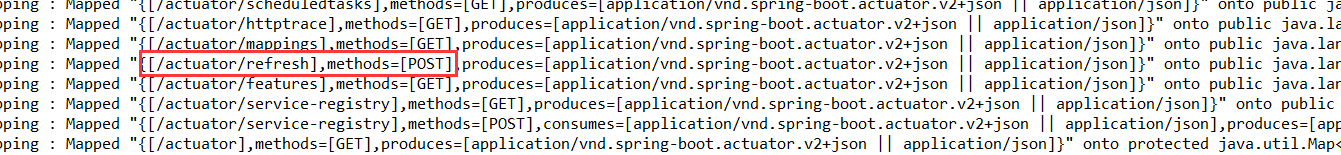
1. 配置文件增加配置项：

|  |
| --- |
| #开启所有端点  management:  endpoints:  web:  exposure:  include: "\*" |

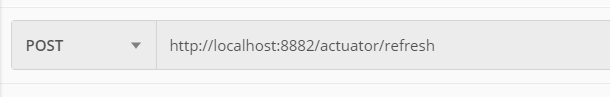
(3)controller中加上@RefshScope注解



重启启动AppConfigClient服务，可以发现启动时有一个接口/actuator/refresh，只要调用这个接口，就可以手动刷新配置文件了，需要注意的是这个接口是post请求



通过postman向接口/actuator/refresh发送刷新请求：



在浏览器再去访问，可以看到配置项的值被刷新了



以下是调用/actuator/refresh接口时的日志

|  |
| --- |
| 2020-02-14 18:42:32.872 INFO 3940 --- [nio-8882-exec-6] s.c.a.AnnotationConfigApplicationContext : **Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@2137c555**: startup date [Fri Feb 14 18:42:32 CST 2020]; root of context hierarchy  2020-02-14 18:42:32.911 INFO 3940 --- [nio-8882-exec-6] f.a.AutowiredAnnotationBeanPostProcessor : JSR-330 'javax.inject.Inject' annotation found and supported for autowiring  2020-02-14 18:42:32.912 INFO 3940 --- [nio-8882-exec-6] **trationDelegate$BeanPostProcessorChecker : Bean 'configurationPropertiesRebinderAutoConfiguration' of type** [org.springframework.cloud.autoconfigure.ConfigurationPropertiesRebinderAutoConfiguration$$EnhancerBySpringCGLIB$$b75434c9] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)  2020-02-14 18:42:34.276 INFO 3940 --- [nio-8882-exec-6] c.c.c.ConfigServicePropertySourceLocator : **Fetching config from server at : http://localhost:8888**  2020-02-14 18:42:36.828 INFO 3940 --- [nio-8882-exec-6] c.c.c.ConfigServicePropertySourceLocator : **Located environment: name=test-configClient, profiles=[prd], label=null, version=6916f55374ae7c14b2869248d3e37cbfae7f8240**, state=null  2020-02-14 18:42:36.828 INFO 3940 --- [nio-8882-exec-6] b.c.PropertySourceBootstrapConfiguration : **Located property source: CompositePropertySource {name='configService', propertySources=[MapPropertySource {name='configClient'}, MapPropertySource {name='https://gitee.com/liuch890228/distributed\_profile\_learning.git/configtest/test-configClient-prd.properties'}]}**  2020-02-14 18:42:36.831 INFO 3940 --- [nio-8882-exec-6] o.s.boot.SpringApplication : No active profile set, falling back to default profiles: default  2020-02-14 18:42:36.838 INFO 3940 --- [nio-8882-exec-6] s.c.a.AnnotationConfigApplicationContext : Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@79bc52cc: startup date [Fri Feb 14 18:42:36 CST 2020]; parent: org.springframework.context.annotation.AnnotationConfigApplicationContext@2137c555  2020-02-14 18:42:36.842 INFO 3940 --- [nio-8882-exec-6] f.a.AutowiredAnnotationBeanPostProcessor : JSR-330 'javax.inject.Inject' annotation found and supported for autowiring  2020-02-14 18:42:36.850 INFO 3940 --- [nio-8882-exec-6] o.s.boot.SpringApplication : Started application in 5.4 seconds (JVM running for 708.189)  2020-02-14 18:42:36.851 INFO 3940 --- [nio-8882-exec-6] s.c.a.AnnotationConfigApplicationContext : Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@79bc52cc: startup date [Fri Feb 14 18:42:36 CST 2020]; parent: org.springframework.context.annotation.AnnotationConfigApplicationContext@2137c555  2020-02-14 18:42:36.853 INFO 3940 --- [nio-8882-exec-6] s.c.a.AnnotationConfigApplicationContext : Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@2137c555: startup date [Fri Feb 14 18:42:32 CST 2020]; root of context hierarchy  2020-02-14 18:42:37.031 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Shutting down DiscoveryClient ...  2020-02-14 18:42:37.033 INFO 3940 --- [nio-8882-exec-6] o.s.c.n.eureka.InstanceInfoFactory : Setting initial instance status as: STARTING  2020-02-14 18:42:40.040 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Unregistering ...  2020-02-14 18:42:40.053 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : DiscoveryClient\_TEST-CONFIGCLIENT/192.168.61.1:test-configClient:8882 - deregister status: 200  2020-02-14 18:42:40.074 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Completed shut down of DiscoveryClient  2020-02-14 18:42:40.079 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Initializing Eureka in region us-east-1  2020-02-14 18:42:40.091 INFO 3940 --- [nio-8882-exec-6] c.n.d.provider.DiscoveryJerseyProvider : Using JSON encoding codec LegacyJacksonJson  2020-02-14 18:42:40.092 INFO 3940 --- [nio-8882-exec-6] c.n.d.provider.DiscoveryJerseyProvider : Using JSON decoding codec LegacyJacksonJson  2020-02-14 18:42:40.092 INFO 3940 --- [nio-8882-exec-6] c.n.d.provider.DiscoveryJerseyProvider : Using XML encoding codec XStreamXml  2020-02-14 18:42:40.092 INFO 3940 --- [nio-8882-exec-6] c.n.d.provider.DiscoveryJerseyProvider : Using XML decoding codec XStreamXml  2020-02-14 18:42:40.255 INFO 3940 --- [nio-8882-exec-6] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration  2020-02-14 18:42:40.258 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Disable delta property : false  2020-02-14 18:42:40.258 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Single vip registry refresh property : null  2020-02-14 18:42:40.258 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Force full registry fetch : false  2020-02-14 18:42:40.258 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Application is null : false  2020-02-14 18:42:40.258 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Registered Applications size is zero : true  2020-02-14 18:42:40.258 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Application version is -1: true  2020-02-14 18:42:40.259 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Getting all instance registry info from the eureka server  2020-02-14 18:42:40.267 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : The response status is 200  2020-02-14 18:42:40.269 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Starting heartbeat executor: renew interval is: 30  2020-02-14 18:42:40.271 INFO 3940 --- [nio-8882-exec-6] c.n.discovery.InstanceInfoReplicator : InstanceInfoReplicator onDemand update allowed rate per min is 4  2020-02-14 18:42:40.273 INFO 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Discovery Client initialized at timestamp 1581676960273 with initial instances count: 3  2020-02-14 18:42:40.274 INFO 3940 --- [nio-8882-exec-6] o.s.c.n.e.s.EurekaServiceRegistry : Unregistering application test-configClient with eureka with status DOWN  2020-02-14 18:42:40.274 INFO 3940 --- [nio-8882-exec-6] o.s.c.n.e.s.EurekaServiceRegistry : Registering application test-configClient with eureka with status UP  2020-02-14 18:42:40.274 WARN 3940 --- [nio-8882-exec-6] com.netflix.discovery.DiscoveryClient : Saw local status change event StatusChangeEvent [timestamp=1581676960274, current=UP, previous=DOWN]  2020-02-14 18:42:40.276 INFO 3940 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient\_TEST-CONFIGCLIENT/192.168.61.1:test-configClient:8882: registering service...  2020-02-14 18:42:40.295 INFO 3940 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient\_TEST-CONFIGCLIENT/192.168.61.1:test-configClient:8882 - registration status: 204 |