Spring Cloud学习文档

# 相关网站

官网：<https://spring.io/projects/spring-cloud>

中文网：<https://springcloud.cc/>

当当书籍目录：<http://product.dangdang.com/25309061.html>

中文文档：<https://springcloud.cc/spring-cloud-dalston.html>

# 简介

Spring Cloud为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智能路由，微代理，控制总线）。分布式系统的协调导致了样板模式, 使用Spring Cloud开发人员可以快速地支持实现这些模式的服务和应用程序。他们将在任何分布式环境中运行良好，包括开发人员自己的笔记本电脑，裸机数据中心，以及Cloud Foundry等托管平台。

版本：Dalston.RELEASE

# 特性

Spring Cloud专注于提供良好的开箱即用经验的典型用例和可扩展性机制覆盖。

* 分布式/版本化配置
* 服务注册和发现
* 路由
* service - to - service调用
* 负载均衡
* 断路器
* 分布式消息传递

# [服务注册](https://www.cnblogs.com/chry/p/7248947.html)

<https://www.cnblogs.com/chry/p/7248947.html>

Spring Cloud官网的定义比较抽象，我们可以从简单的东西开始。Spring Cloud是基于Spring Boot的， 最适合用于管理Spring Boot创建的各个微服务应用。要管理分布式环境下的各个Spring Boot微服务，必然存在服务的注册问题。所以我们先从服务的注册谈起。既然是注册，必然有个管理注册中心的服务器，各个在Spring Cloud管理下的Spring Boot应用就是需要注册的client。

Spring Cloud使用erureka [jʊ'ri:kə] server,  然后所有需要访问配置文件的应用都作为一个erureka client注册上去。eureka是一个高可用的组件，它没有后端缓存，每一个实例注册之后需要向注册中心发送心跳，在默认情况下erureka server也是一个eureka client ,必须要指定一个 server。

**2.  创建Eureka Server**

1）.创建一个Maven工程helloworld.eureka.server， pom.xml内容如下：

<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>org.lys</groupId>

<artifactId>helloworld.eureka.server</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>helloworld.eureka.server</name>

<url>http://maven.apache.org</url>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.3.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!--eureka server -->

<dependency>

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

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-eureka-server</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-config</artifactId>

</dependency>

<!-- spring boot test-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RC1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

</project>

2）. 用Spring Boot创建一个服务类EurekaServerApplication，需要一个注解@EnableEurekaServer加在springboot工程的启动类上

**package** org.lys.helloworld.eureka.server;

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@EnableEurekaServer

@SpringBootApplication

**public** **class** EurekaServerApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(EurekaServerApplication.**class**, args);

}

}

3).eureka server的配置文件application.yml，其中registerWithEureka：false和fetchRegistry：false表明自己是一个eureka server

server:

port: 8761

eureka:

instance:

hostname: localhost

client:

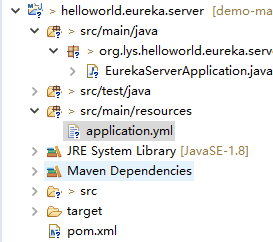
registerWithEureka: false

fetchRegistry: false

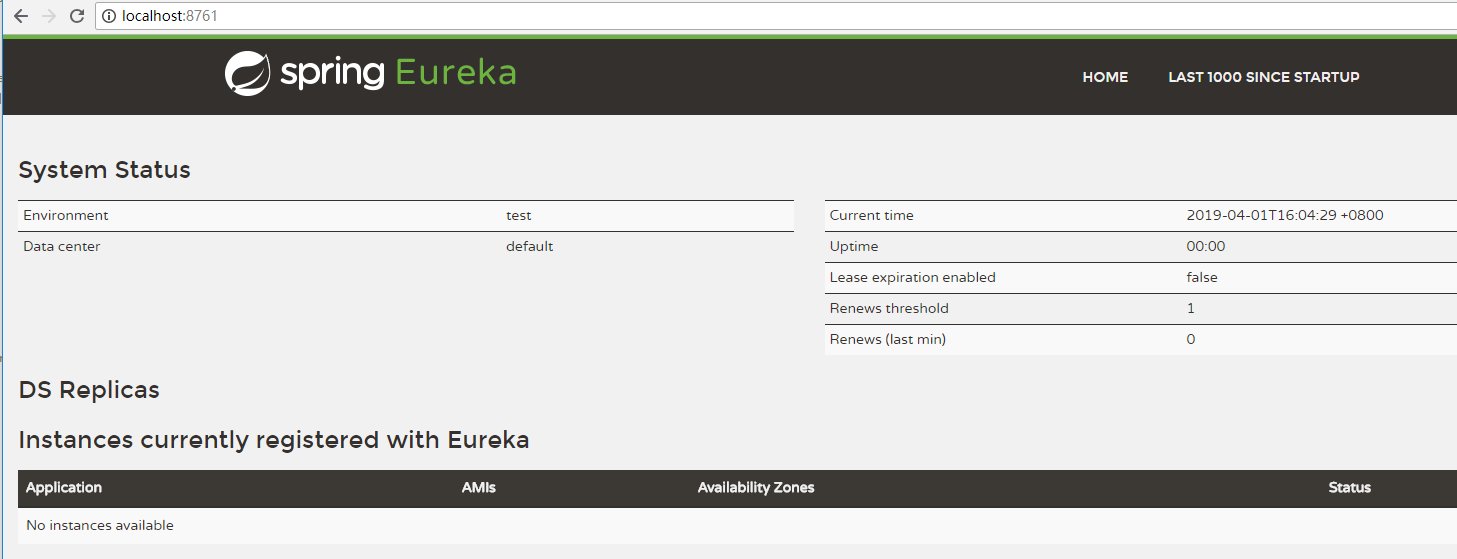
serviceUrl:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

4) eureka server的工程结构如下



5）启动eureka server，然后访问http://localhost:8761, 界面如下， "No instances available" 表示无client注册



**3.  创建Eureka Client**

1). 创建一个Maven工程helloworld.eureka.client， pom.xml内容如下：

<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>org.lys</groupId>

<artifactId>helloworld.eureka.client</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>helloworld.eureka.client</name>

<url>http://maven.apache.org</url>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.3.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RC1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

</project>

2).  创建主类EurekaClientApplication

使用@EnableEurekaClient注解表明是client

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

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@EnableEurekaClient

@RestController

**public** **class** EurekaClientApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(EurekaClientApplication.**class**, args);

}

@Value("${server.port}")

String port;

@RequestMapping("/")

**public** String home() {

**return** "hello world from port " + port;

}

}

3) eureka client的配置文件application.yml

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka/

server:

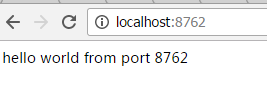
port: 8762

spring:

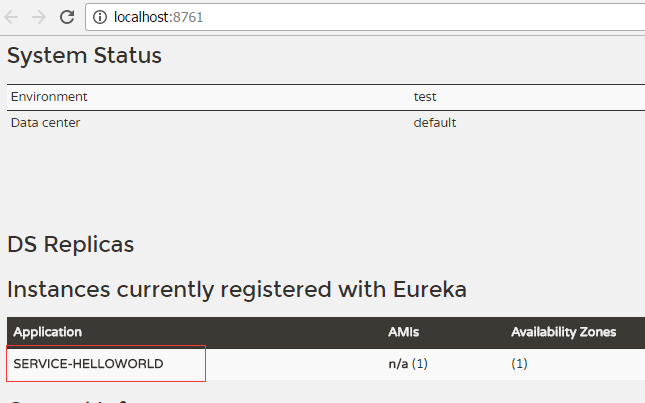
application:

name: service-helloworld

4). Client启动后， 可以访问<http://localhost:8762>



5). 再次访问服务器端口, 可以看到Service Helloworld已经自动注册到之前的server中http://localhost:8761/



# [配置管理](https://www.cnblogs.com/chry/p/7250584.html)

<http://www.cnblogs.com/chry/p/7250584.html>

使用Config Server，您可以在所有环境中管理应用程序的外部属性。客户端和服务器上的概念映射与Spring *Environment*和*PropertySource*抽象相同，因此它们与Spring应用程序非常契合，但可以与任何以任何语言运行的应用程序一起使用。随着应用程序通过从开发人员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。很容易添加替代实现，并使用Spring配置将其插入。

以上是Spring Cloud官网对配置服务的描述， 简单阐述一下我的理解。比如我们要搭建一个网站，需要配置数据库连接，指定数据库服务器的IP地址，数据库名称，用户名和口令等信息。通常的方法， 我们可以在一个配置文件中定义这些信息，或者开发一个页面专门配置这些东西。只有一个web服务器的时候, 很方便。但假如需要搭建同多台服务器时，当然可以每台服务器做同样配置，但维护和同步会很麻烦。我理解的配置服务至少有两种不同场景：

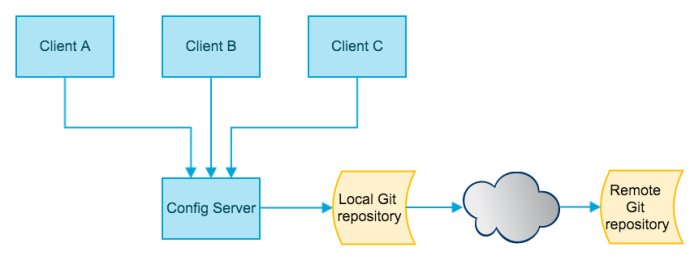
1）.  多个客户使用同一配置： 比如，多台服务器组成的集群，假如后端使用同一数据库，那么每台服务器都是用相同的配置。

2）.  不同客户使用不同的配置： 比如典型的场景是，开发，测试，生产使用相同的系统，但使用不同的数据库

如果有个统一的根本配置，是不是就很方便，一个可行的办法是，把这些配置文件放到一个共享存储（比如网络共享盘）中。这样只需要在共享存储修改一个或多个配置文件就可以了。但共享文件的方式受到具体布署环境的限制，很多时候很难达到多台Web服务器共享同一个存储硬盘。

共享盘的缺点是资源定位比较困难，Spring Cloud的解决方案是， 将这些配置文件放到版本管理服务器里面，Spring Cloud缺省配置使用GIT中。所有Web服务均从GIT中获取这些配置文件。由于GIT服务器与具体Web服务器之间不需要共享存储， 只要网络可达就行，从而可以实现Web服务于配置信息的存放位置的解耦。

Spring Cloud统一控制应用和GIT服务的交互，应用只需要按照Spring Cloud的规范配置GIT的URL即可。 使用GIT后，场景2和场景1的区别仅仅是，场景2中不同的client使用不同版本的配置文件，但应用但访问的文件看起来是会是同一个。Spring Cloud的配置服务结构入下图



下面我们继续上一节的例子[Spring Cloud 入门之一. 服务注册](http://www.cnblogs.com/chry/p/7248947.html) 继续展开, 让“Hello World”从配置文件helloworld.properties读出，内容格式如下

hello=Hello World

其中关键字hello的值“Hello World”，就是我们要输出的内容。

**一. 创建config Server**

 1.  创建Config Server：helloworld.config.server， maven工程里面配置spring-cloud-config-server

<dependency>

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

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

完整配置如下：

<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>org.lys</groupId>

<artifactId>helloworld.config.server</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>helloworld.config.server</name>

<url>http://maven.apache.org</url>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.3.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!--eureka server -->

<dependency>

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

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-eureka-server</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

<!-- spring boot test-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RC1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

</project>

2. 创建Config Server，它也是一个Spring Boot应用，@EnableConfigServer注解说明了一个Config Server。同样我们使用@EnableEurekaClient将它注册到服务中心。

**package** org.lys.helloworld.config.server;

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.cloud.config.server.EnableConfigServer;

//@EnableEurekaClient

@EnableConfigServer

@SpringBootApplication

**public** **class** ConfigServerApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(ConfigServerApplication.**class**, args);

}

}

3. Config server的配置文件application.yml , 注意配置文件的url是GIT服务器的仓库地址， searchPaths配置文件所在的文件夹在仓库中的路径, 在server端不需要指定具体配置文件名， 因为具体的配置文件是什么有应用（也就是client）决定。

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka/

server:

port: 8888

spring:

cloud:

config:

server:

git:

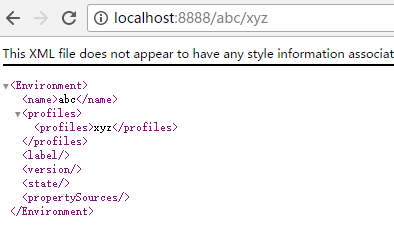
uri: https://git.oschina.net/chrywhy/test

searchPaths: spring-cloud/helloworldConfig

application:

name: config-server

4. 启动config server后，访问http://localhost:8888/abc/xyz, 可见如下响应。这个是输出是并没有包括具体配置文件的内容， 这个响应说明，config server可以正常访问我们配置在application.yml中的GIT服务



这个URL是啥意思， 需要解释一下。我们从输出就可以看到 abc 就是application的名字，xyz是profile的名字， 注意这里的abc, xyz均是随便输入的名字， 并不需要真实存在，config server这个REST接口返回的只是应用名为abc, profile名为xyz时，GIT配置环境的结构。

config server提供的REST接口，Spring Cloud官方文档提供了几个可选URL可以是如下几个：

1. /{application}/{profile}[/{label}]
2. /{application}-{profile}.yml
3. /{label}/{application}-{profile}.yml
4. /{application}-{profile}.properties
5. /{label}/{application}-{profile}.properties

比如 第三个格式，如果我们在GIT版本库中有一个配置文件 spring-cloud/helloworldConfig/config-client-dev.properties. 那么访问http://localhost:8888/config-client-dev.properties就可以显示配置文件内容。这个例子中， application的名字是"config-client"(也是下面我们即将创建的client), profile名字是dev, 文件后缀是.properties



本例由于配置了eureka服务中心，所以这个config server作为一个eureka client注册到了 eureka server中， 可以从http://localhost:8761看到我们启动的config server, 如果不需要注册到服务中心， 也可把这个配置去掉

  二**. 创建config client:** **helloworld.config.client**

1.  创建maven工程， pom.xml如下：

<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>org.lys.demo</groupId>

<artifactId>helloworld.config.client</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>helloworld.config.client</name>

<url>http://maven.apache.org</url>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.3.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<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-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RC1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

</project>

2. 创建一个spring boot应用作为client

**package** org.lys.demo.helloworld.config.client;

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

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@RestController

**public** **class** ConfigClientApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(ConfigClientApplication.**class**, args);

}

@Value("${hello}")

String hello;

@RequestMapping(value = "/hello")

**public** String hello(){

**return** hello;

}

}

这个应用非常简单，就是从Config Server中获取配置项hello的值，Client Server向Config Server提交REST请求后，Config Server将访问GIT服务器，并将取得的配置项hello的值返回给client.

3. Config client需要一个应用配置文件， 定义config Server的URL，以及要访问的GIT具体分支。这个配置文件是bootstrap.yml (或者bootstrap.properties)

spring:

application:

name: config-client

cloud:

config:

label: master

profile: dev

uri: http://localhost:8888/

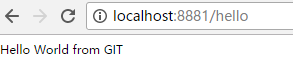
server:

port: 8881

这个配置定义了应用的名字是config-client(这就是将要用于组装前面Config Server一节中题到的application), profile采用dev, GIT分支用master。url是config server的地址。那么问题来了，我们似乎没定义配置文件名， 那配置文件名是什么呢？ 这点又体现了约定优于配置的思路， 这里Spring Cloud约定， 应用的配置文件名以如下方式组成：{application}-{profile}.properties（或者{application}-{profile}.yml）。比如我们这个应用的配置文件就是config-client-dev.properties. 所以只需要在GIT的中创建配置文件spring-cloud/helloworldConfig/config-client-dev.properties就可以了， 内容如下：

hello=Hello World from GIT

 4. 启动config-client应用后， 可以访问http://localhost:8881/hello, 可以看到，应用本身并没有直接配置hello的具体内容， 也没指定具体配置文件，所欲这些都由spring cloud框架提交给config server了。

**

5.  配置的更新

至此，spring cloud的配置管理简单示例已经完成，但client 不能自动感知服务端的变化。 比如，我们修改了GIT中的文件内容，但无论如何刷新client端的页面，都不能反映配置的变化

# [配置自动刷新](https://www.cnblogs.com/chry/p/7260778.html)

<http://www.cnblogs.com/chry/p/7260778.html>

之前讲的配置管理， 只有在应用启动时会读取到GIT的内容， 之后只要应用不重启，GIT中文件的修改，应用无法感知， 即使重启Config Server也不行。

比如上一单元（Spring Cloud 入门教程(二)： 配置管理）中的Hello World 应用，手动更新GIT中配置文件config-client-dev.properties的内容（别忘了用GIT push到服务器）

hello=Hello World from GIT version 1

刷新 http://localhost:8881/hello，页面内容仍然和之前一样，并没有反映GIT中最新改变， 重启config-server也一样，没有任何变化。要让客户端应用感知到这个变哈， Spring Cloud提供了解决方案是，客户端用POST请求/refresh方法就可以刷新配置内容。

**1. 让客户端支持/refresh方法**

要让/refresh生效，客户端需要增加一些代码支持：

1). 首先，在pom.xml中添加以下依赖。spring-boot-starter-actuator是一套监控的功能，可以监控程序在运行时状态，其中就包括/refresh的功能。

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

并修改bootstrap.yml

spring:

application:

name: config-client

cloud:

config:

label: master

profile: dev

uri: http://localhost:8888/

server:

port: 8881

management:

security:

enabled: false

增加红色部分，springboot 1.5.X 以上默认开通了安全认证，所以需要在配置文件添加以下配置management.security.enabled=false

2). 其次，开启refresh机制， 需要给加载变量的类上面加载@RefreshScope注解，其它代码可不做任何改变，那么在客户端执行/refresh的时候就会更新此类下面的变量值，包括通过config client从GIT获取的配置。

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

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.context.config.annotation.RefreshScope;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@RestController

@RefreshScope

public class ConfigClientApplication {

public static void main(String[] args) {

SpringApplication.run(ConfigClientApplication.class, args);

}

@Value("${hello}")

String hello;

@RequestMapping(value = "/hello")

public String hello(){

return hello;

}

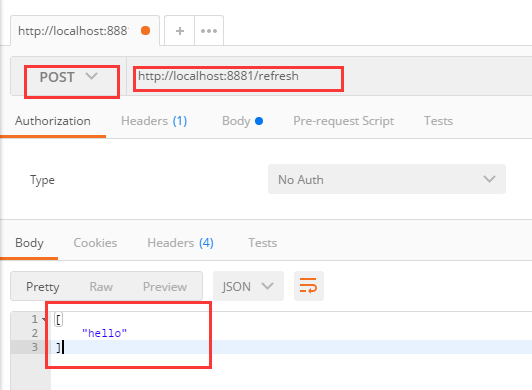
}

3). 启动应用， 查看http://localhost:8881/hello

4). 再次修改config-client-dev.properties的内容

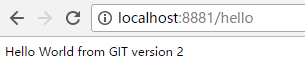
hello=Hello World from GIT version 2

5). 用postman发送POST请求：http://localhost:8881/refresh



可以从POST的结果看到，此次refresh刷新的配置变量有hello

6). 再次访问http://localhost/hello，可见到配置已经被刷新



**2. 通过Webhook自动触发/refresh方法刷新配置**

以上每当GIT中配置文件被修改，仍然需要我们主动调用/refresh方法（手动调用或者写代码调用）， 有没有办法让GIT中配置有改动就自动触发客户端的refresh机制呢？ 答案是：有。可以通过GIT提供的githook来监听push命令，如果项目中使用了Jenkins等持续集成工具（也是利用githook来监听的），就可以监听事件处理中直接调用/refresh方法就可以了。