**Spring Cloud构建微服务架构（四）分布式配置中心**

Spring Cloud Config为服务端和客户端提供了分布式系统的外部化配置支持。配置服务器为各应用的所有环境提供了一个中心化的外部配置。它实现了对服务端和客户端对Spring Environment和PropertySource抽象的映射，所以它除了适用于Spring构建的应用程序，也可以在任何其他语言运行的应用程序中使用。作为一个应用可以通过部署管道来进行测试或者投入生产，我们可以分别为这些环境创建配置，并且在需要迁移环境的时候获取对应环境的配置来运行。

配置服务器默认采用git来存储配置信息，这样就有助于对环境配置进行版本管理，并且可以通过git客户端工具来方便的管理和访问配置内容。当然他也提供本地化文件系统的存储方式，下面从这两方面介绍如何使用分布式配置来存储微服务应用多环境的配置内容。

**构建Config Server**

通过Spring Cloud构建一个Config Server，非常简单，只需要三步：

* pom.xml中引入spring-cloud-config-server依赖，完整依赖配置如下：

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.3.5.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-config-server</artifactId>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Brixton.RELEASE</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement> |

* 创建Spring Boot的程序主类，并添加@EnableConfigServer注解，开启Config Server

|  |
| --- |
| @EnableConfigServer  @SpringBootApplication  **public** **class** **Application** {  **public** **static** **void** **main**(String[] args) {  **new** SpringApplicationBuilder(Application.class).web(**true**).run(args);  }  } |

* application.properties中配置服务信息以及git信息，例如：

|  |
| --- |
| spring.application.name=config-server  server.port=7001  # git管理配置  spring.cloud.config.server.git.uri=http://git.oschina.net/didispace/SpringBoot-Learning/  spring.cloud.config.server.git.searchPaths=Chapter9-1-4/config-repo  spring.cloud.config.server.git.username=username  spring.cloud.config.server.git.password=password |

* spring.cloud.config.server.git.uri：配置git仓库位置
* spring.cloud.config.server.git.searchPaths：配置仓库路径下的相对搜索位置，可以配置多个
* spring.cloud.config.server.git.username：访问git仓库的用户名
* spring.cloud.config.server.git.password：访问git仓库的用户密码

到这里，使用一个通过Spring Cloud Config实现，并使用git管理内容的配置中心已经完成了，启动该应用，成功后开始下面的内容。

**Spring Cloud Config也提供本地存储配置的方式。我们只需要设置属性spring.profiles.active=native，Config Server会默认从应用的src/main/resource目录下检索配置文件。也可以通过spring.cloud.config.server.native.searchLocations=file:F:/properties/属性来指定配置文件的位置。虽然Spring Cloud Config提供了这样的功能，但是为了支持更好的管理内容和版本控制的功能，还是推荐使用git的方式。**

**服务端验证**

为了验证上面完成的配置服务器，在git 项目下创建了一个config-repo目录作为配置仓库，并根据不同环境新建了下面四个配置文件：

* didispace.properties
* didispace-dev.properties
* didispace-test.properties
* didispace-prod.properties

其中设置了一个from属性，为每个配置文件分别设置了不同的值，如：

* from=git-default-1.0
* from=git-dev-1.0
* from=git-test-1.0
* from=git-prod-1.0

为了测试版本控制，在master中，我们都加入1.0的后缀，同时创建一个config-label-test分支，并将各配置文件中的值用2.0作为后缀。

完成了这些准备工作之后，我们就可以通过浏览器或POSTMAN等工具直接来访问到我们的配置内容了。

URL与配置文件的映射关系如下：

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

上面的url会映射{application}-{profile}.properties对应的配置文件，{label}对应git上不同的分支，默认为master。

我们可以尝试构造不同的url来访问不同的配置内容，比如：要访问config-label-test分支，didispace应用的prod环境，可以通过这个url：[http://localhost:7001/didispace/prod/config-label-test](http://localhost:7001/didispace/prod/config-label-test" \t "_blank)

|  |
| --- |
| {  "name": "didispace",  "profiles": [  "prod"  ],  "label": "config-label-test",  "version": "19de8a25575a7054a34230f74a22aa7f5575a9d1",  "propertySources": [  {  "name": "http://git.oschina.net/didispace/SpringBoot-Learning/Chapter9-1-4/config-repo/didispace-prod.properties",  "source": {  "from": "git-prod-2.0"  }  },  {  "name": "http://git.oschina.net/didispace/SpringBoot-Learning/Chapter9-1-4/config-repo/didispace.properties",  "source": {  "from": "git-default-2.0"  }  }  ]  } |

**微服务端映射配置**

在完成并验证了配置服务中心之后，下面看看我们如何在微服务应用中获取配置信息。

* 创建一个Spring Boot应用，在pom.xml中引入spring-cloud-starter-config依赖，完整依赖关系如下：

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.3.5.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </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>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Brixton.RELEASE</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement> |

* 创建最基本的Spring Boot启动主类

|  |
| --- |
| @SpringBootApplication  **public** **class** **Application** {  **public** **static** **void** **main**(String[] args) {  **new** SpringApplicationBuilder(Application.class).web(**true**).run(args);  }  } |

* 创建bootstrap.properties配置，来指定config server，例如：

|  |
| --- |
| spring.application.name=didispace  spring.cloud.config.profile=dev  spring.cloud.config.label=master  spring.cloud.config.uri=http://localhost:7001/  server.port=7002 |

* spring.application.name：对应前配置文件中的{application}部分
* spring.cloud.config.profile：对应前配置文件中的{profile}部分
* spring.cloud.config.label：对应前配置文件的git分支
* spring.cloud.config.uri：配置中心的地址

**这里需要格外注意：上面这些属性必须配置在bootstrap.properties中，config部分内容才能被正确加载。因为config的相关配置会先于application.properties，而bootstrap.properties的加载也是先于application.properties。**

* 创建一个Rest Api来返回配置中心的from属性，具体如下：

|  |
| --- |
| @RefreshScope  @RestController  **class** **TestController** {  @Value("${from}")  **private** String from;  @RequestMapping("/from")  **public** String **from**() {  **return** **this**.from;  }  } |

通过@Value("${from}")绑定配置服务中配置的from属性。

启动该应用，并访问：<http://localhost:7002/from> ，我们就可以根据配置内容输出对应环境的from内容了。