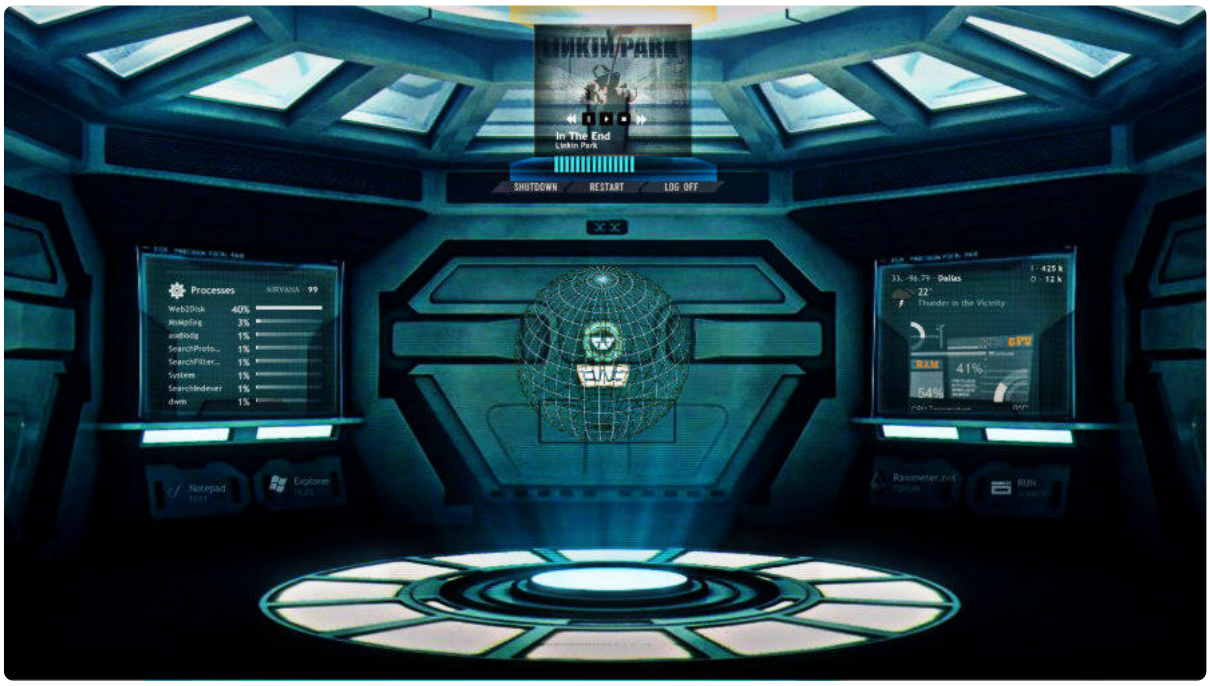
22 初识 Spring Cloud Config

更新时间：2019-07-09 09:53:05



> 能够生存下来的物种,并不是那些最强壮的,也不是那些最聪明的,而是那些对变化作出快速反应的。
>
> ——达尔文

通过前面的学习，相信大家对于 Spring Cloud 已经有了一个大致的认识了，应该感受到微服务和我们传统架构的项目最大区别在于微服务中要部署的项目数量非常多，每个服务的配置也非常繁杂。如果将这些配置文件都放在各个微服务之中，随着项目不断变大，配置文件会越来越多，越来越复杂，每一次项目上线估计运维工程师都要崩溃了，维护也非常不方便，因为要改的数据太多了。那么有没有一种办法可以让我们对这些配置文件进行统一管理呢？有！那就是我们今天要说的 Spring Cloud Config！

可能有人已经听说过 Spring Cloud Config，但分布式配置解决方案却不止 Spring Cloud Config，还有其它一些框架，例如 360 的 QConf 、淘宝的 diamond 、百度的 disconf 等都可以解决分布式配置中心问题。国外也有很多开源的配置中心例如 Apache Commons Configuration 、owner 、cfg4j 等等，但是 Spring Cloud Config 的功能更为强大，而且可以和 Spring 家族无缝结合，非常方便。

Spring Cloud Config 简介

Spring Cloud Config 项目是一个解决分布式系统的配置管理方案。它包含了 Client 和 Server 两个部分，Server 提供配置文件的存储，然后以接口的形式将配置文件的内容提供出去，Client 通过接口获取数据，然后依据此数据初始化自己的应用。Spring cloud 支持使用 Git 或者 Svn 存放配置文件，默认情况下使用 Git，考虑到目前企业开发的实际情况，我们这里主要通过 Git 来向大家演示 Spring Cloud Config 的用法。

那么 Spring Cloud Config 都有哪些功能呢？

- 提供服务端和客户端支持
- 集中管理各环境的配置文件
- 配置文件修改之后，可以快速生效

- 因为配置文件通过 Git 或者 Svn 进行管理，所以配置文件天然具备版本回退等功能
- 支持大的并发查询
- 支持多种开发语言

好了，说了这么多，可能有人已经迫不及待想要试一试 Spring Cloud Config了，那么接下来我们就来看看这个 Spring Cloud Config 到底要怎么使用！

准备工作

学习Spring Cloud Config 需要大家掌握 Git 的基本操作，因为本专栏以 Spring Cloud Config 为主，因此如果大家对 Git 的操作还不太擅长的话，建议参考下面的教程：

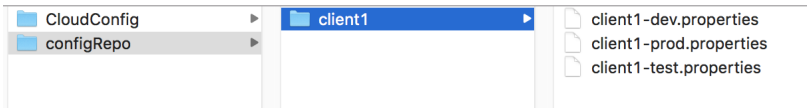
Git教程

接下来的操作默认大家已经掌握了 Git的基本用法了。

首先我们需要创建一个本地 Git 仓库，这个仓库中存放的就是我们的配置文件，具体操作步骤如下：

- 创建文件

首先创建一个名为 configRepo 的文件夹，文件夹中存放一个名为 client1 的文件夹，client1 中存放三个文件，如下图：



假设这三个文件就是用户微服务在不同环境下（如图分别表示开发环境、生产环境、测试环境以及默认配置）的配置文件，三个文件的内容分别如下：

client1-dev.properties

```
javaboy=dev
```

client1-test.properties

```
javaboy=test
```

client1-prod.properties

```
javaboy=prod
```

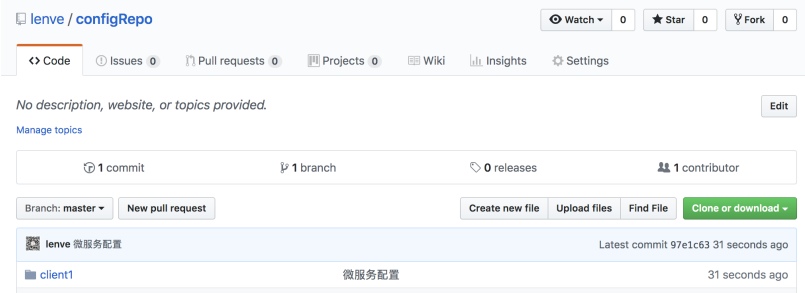
- 仓库初始化

初始化本地仓库，然后在 GitHub 上创建一个名为 configRepo 的仓库，将本地仓库中的数据上传到远程仓库，操作命令如下：

```
git init
git add .
git commit -m "微服务配置"
git remote add origin git@github.com:lenve/configRepo.git
git push -u origin master
```

注意：这里我们需要将 **client1** 目录保存到 **GitHub** 远程仓库（即上面的命令是在 **configRepo** 目录下执行的），**configRepo** 目录下的不同文件夹中保存不同微服务的配置文件（这只是我们暂时的配置文件规划，学习完本文读者也可以自己来规划相关配置文件位置）。

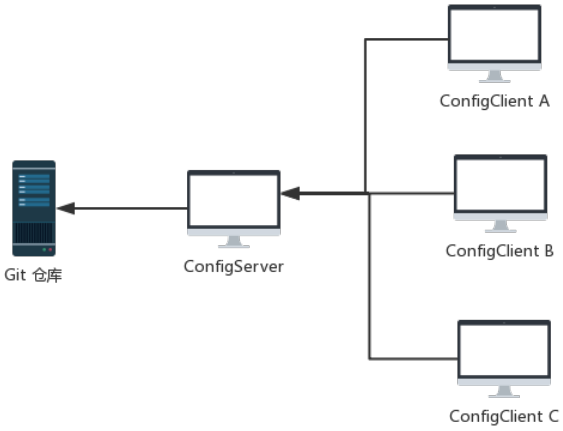
上传到 GitHub 之后，GitHub 上目录结构如下：



将本地仓库中的数据同步到远程仓库之后，接下来我们需要搭建一个 Spring Cloud Config Server ，通过这个微服务提供的接口来访问存储在 Git 仓库中的配置文件。

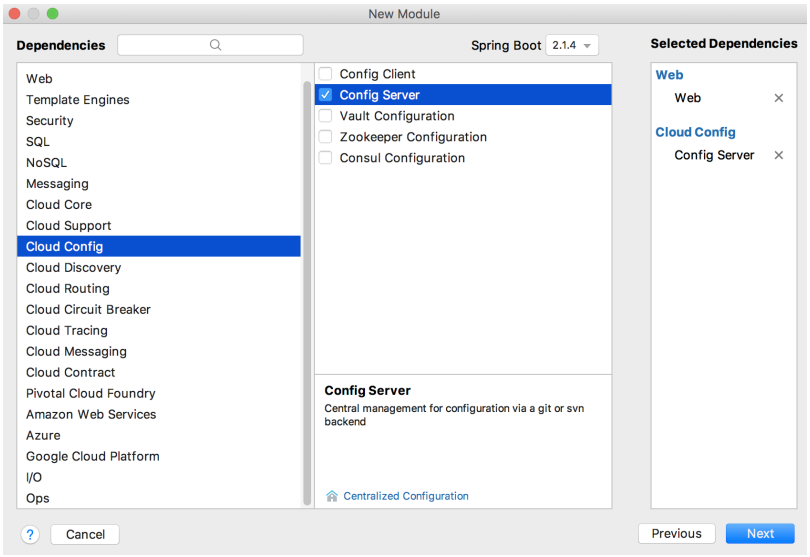
Config Server

首先我们来看下面一张简单的 Spring Cloud Config 工作流程图：



ConfigServer 从 Git 仓库获取到配置文件，然后 ConfigClient（就是我们前文中和大家分享的一个一个的微服务）再从 ConfigServer 中获取各自的配置文件，大致就是这样一个工作流程。

接下来我们首先创建一个名为 CloudConfig 的 Maven 父工程，这个父工程是空的，然后在父工程中创建一个名为 confin_server 的 Spring Boot 工程，创建时候添加 Web 依赖和 Config Server 依赖，如下图：



创建完成后的 pom.xml 文件内容如下（注意，这里暂时不需要服务注册中心）：

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

创建完成后，接下来首先在项目启动类上添加 `@EnableConfigServer` 注解表示开启配置中心服务端，如下：

```
@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {

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

}
```

然后在 application.properties 中添加仓库相关配置，如下：

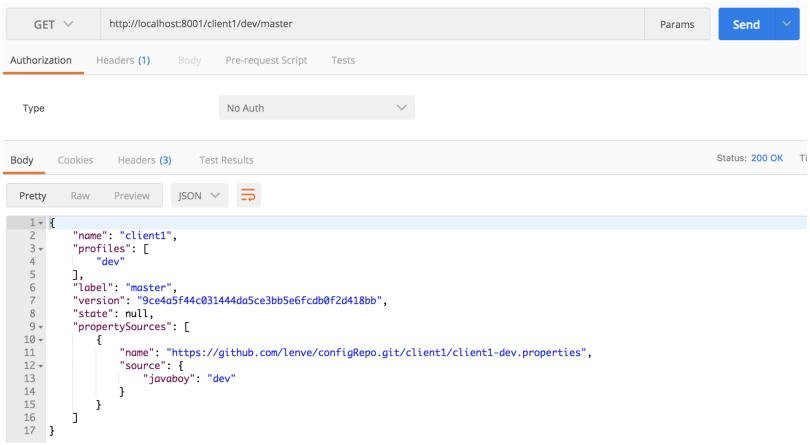
```
spring.application.name=config-server
server.port=8001
spring.cloud.config.server.git.uri=https://github.com/lenve/configRepo.git
spring.cloud.config.server.git.search-paths=client1
spring.cloud.config.server.git.username=wangsong0210@gmail.com
spring.cloud.config.server.git.password=
```

各项配置含义如下：

1. 前两行表示服务的名称和端口号，这个比较容易，不需多说；
2. 接下来 git.uri 表示配置的 GitHub 仓库地址，数据将从这个地方加载；
3. search-paths 表示仓库中配置文件的地址，由于 ConfigServer 是 Git 仓库和微服务之间的一个桥梁，不同微服务的仓库在 Git 仓库中放在不同的目录下，这里是指具体的目录，有人会说这样把地址写死了没有任何意义，这个问题一会和大家详细分析；
4. 接下来两个配置就是自己的 GitHub 的用户名和密码，但是这两个配置并非必须的，如果你的仓库就是公开的，那么可以不必配置，实际生产环境中，仓库往往是私有的，因此这里就需要配置用户密码来获取仓库的访问权

限。

配置完成后，启动 config_server 项目，启动成功之后，我们就可以在浏览器中输入 `http://localhost:8001/client1/dev/master` 来访问配置文件了，访问返回结果如下：



可以看到，这里返回了配置文件的所有信息，同时，我们观察 config_server 的控制台，可以看到输出了如下一行日志：

```
Adding property source: file:/var/folders/wx/8j_v819j5n5cgdj3_ymf7pjh0000gn/T/config-repo-2130359132534713O1/client1/client1-dev.properties
```

进入这里提到的目录，发现 config_server 已经将远程 git 仓库中的内容 clone 下来了：

事实上，仓库中的配置文件会被转换成web接口，访问可以参照以下的规则：

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

这里，各个占位符含义分别如下：

- {application} 表示配置文件的文件名
- {profile} 表示配置文件的 profile ，例如我们上文提到的 test、dev 以及 prod
- {label} 表示配置文件的 git 分支

此时我们尝试修改本地仓库的配置文件，例如修改 client1-dev.properties 文件内容如下：

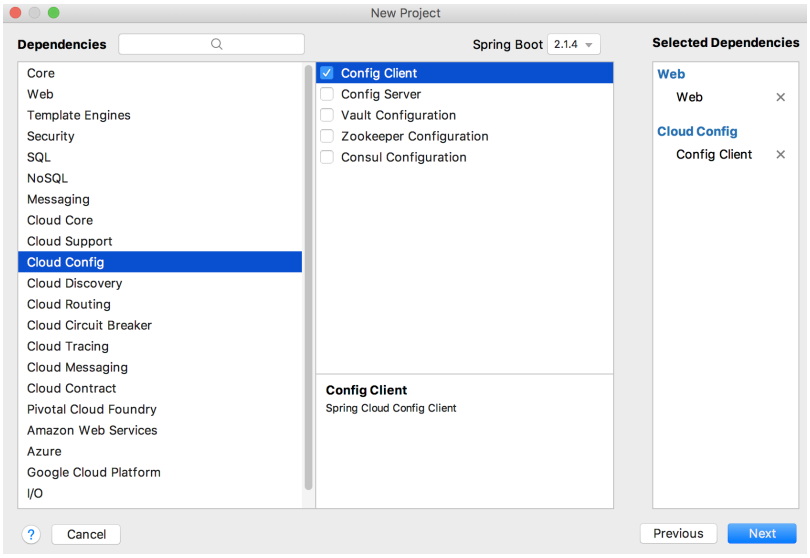
```
javaboy=hello dev
```

然后提交到 Git 仓库，再通过 ConfigServer 进行访问，发现访问到的数据已经发生了变化，说明 ConfigServer 中的数据可以实时更新。

好了，通过上面的配置，我们的 ConfigServer 就算是配置成功了，接下来我们就来看看如何在 ConfigClient 中来访问 ConfigServer 中提供的配置数据。

Config Client

首先我们在父工程 CloudConfig 中创建一个子工程，叫做 config_client。创建时，添加 Config Client 依赖，如下图：



创建成功后，pom.xml 文件中的依赖如下：

```
<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>
```

创建成功后，添加配置文件，注意配置文件添加在 bootstrap.properties 文件中，这时需要大家手动在 classpath 下添加 bootstrap.properties 配置文件，相对于 application.properties 文件，bootstrap.properties 文件加载时机更早，适合于目前的场景，bootstrap.properties 中的配置内容如下：

```
spring.application.name=client1
server.port=8002
spring.cloud.config.profile=dev
spring.cloud.config.label=master
spring.cloud.config.uri=http://localhost:8001/
```

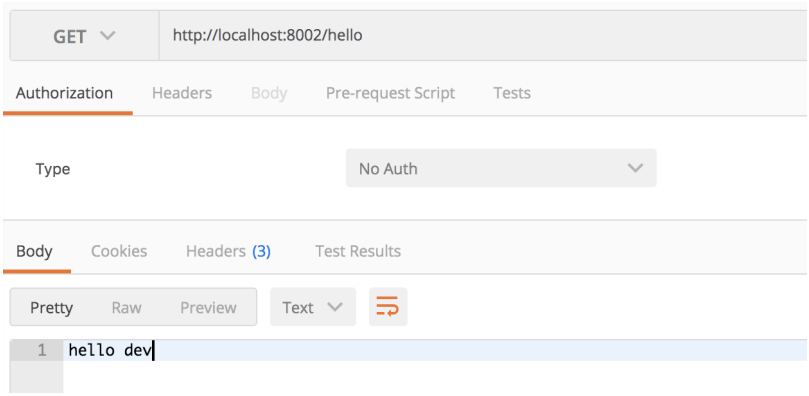
配置含义如下：

1. 前面两行是指服务的名字和端口号，这个比较简单，无需多说；
2. 大家还记得 config_server 中的访问地址问题吧。上文我们提到的访问规则是 `/{application}/{profile}[/{label}]` ，当 Config Client 去访问 Config Server 时， `spring.application.name` 、 `spring.cloud.config.profile` 以及 `spring.cloud.config.label` 的值分别对应上面三个占位符，所以这三个地方不能乱写，要根据实际情况来，特别是 `spring.application.name` 不能写错，因为之前我们定义名称时都是可以随意取的，这里不可以；
3. spring.cloud.config.uri 则表示 config_server 的地址。

配置完成后，当我们的 config_client 项目启动时，配置文件会被自动加载到项目中，我们就可以像使用普通配置文件一样来使用仓库中的配置文件，我们写一个简单的 HelloController 来测试下，如下：

```
@RestController
public class HelloController {
    @Value("${javaboy}")
    String hello;
    @GetMapping("/hello")
    public String hello() {
        return hello;
    }
}
```

配置完成后，启动 config_client ，访问 `/hello` 接口，结果如下：



成功访问到我们需要的配置文件，如果想访问 app-prod.properties ，只需要修改 bootstrap.properties 中 `spring.cloud.config.profile` 的值为 prod 即可。

配置细节

由于配置细节较多，这里主要和大家说一下关于路径的配置细节，其它的细节将在本章后面的小节中和大家介绍。

前面和大家说过，`spring.application.name` 、`spring.cloud.config.profile` 以及 `spring.cloud.config.label` 的值分别对应 `{application}` 、`{profile}` 以及 `{label}` 三个占位符，即在 config_server 中，我们可以通过 `{application}` 、`{profile}` 以及 `{label}` 分别访问到 config_client 中对应的 `spring.application.name` 、`spring.cloud.config.profile` 以及 `spring.cloud.config.label` 的值。利用这种特性我们就可以实现对 `config_server` 中 search-paths 属性的动态配置，例如我们可以将 search-paths 的值和 `config_client` 的 `spring.application.name` 绑定在一起，这样就可以实现动态修改配置文件的文件夹了。例如我们上文提到的配置方案，我们现在可以直接修改 config_server 的配置文件，如下：

```
spring.application.name=config-server
server.port=8001
spring.cloud.config.server.git.uri=https://github.com/lenve/configRepo.git
spring.cloud.config.server.git.search-paths={application}
spring.cloud.config.server.git.username=wangsong0210@gmail.com
spring.cloud.config.server.git.password=
```

这里我们主要是修改了 search-paths 的值，将之用一个 `{application}` 占位符代替，然后重启 config_server ，发现运行效果和前面的一样。

如此之后，我们就可以通过灵活使用 `{application}` 、`{profile}` 以及 `{label}` 三个占位符，来动态地从 client 中控制 server 所访问的仓库了。读者可以根据自己的实际情况，把这三个占位符玩的更加精彩。

本地配置

上文我们提到的配置是在 Git 仓库中完成的，Spring Cloud Config 也提供本地存储配置的方式，例如我们可以添加如下将配置，表示让 config_server 在 classpath 下查找配置文件：

```
spring.profiles.active=native
```

也可以添加如下配置，通过绝对路径来定位配置文件位置：

```
spring.cloud.config.server.native.searchLocations=file:E:/properties/
```

这些方式，大家作为一个了解即可，不必深究，项目中，我们一般还是以 Git 仓库中的配置为主。

小结

本文主要向读者介绍了 Spring Cloud Config 的一个基本使用，涉及到 Spring Cloud Config Server 和 Spring Cloud Config Client 的搭建，其中 Config Client 就是我们一个一个的微服务，整个过程并不复杂，不过还有很多细节没有讲到，例如 config_server 的安全管理、配置文件的加密等，这些我们将在下篇文章和大家介绍。

21 Gateway 中 Predicate 和 Filter 的用法

23 Spring Cloud Config 中配置文件的加密与解密

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示

目前暂无任何讨论