spring cloud config

这里主要是记录一些难点，具体步骤见书籍《spring cloud in action》

1.spring cloud config 包括两个子工程

一是config Server

二是config client

所以分两步：

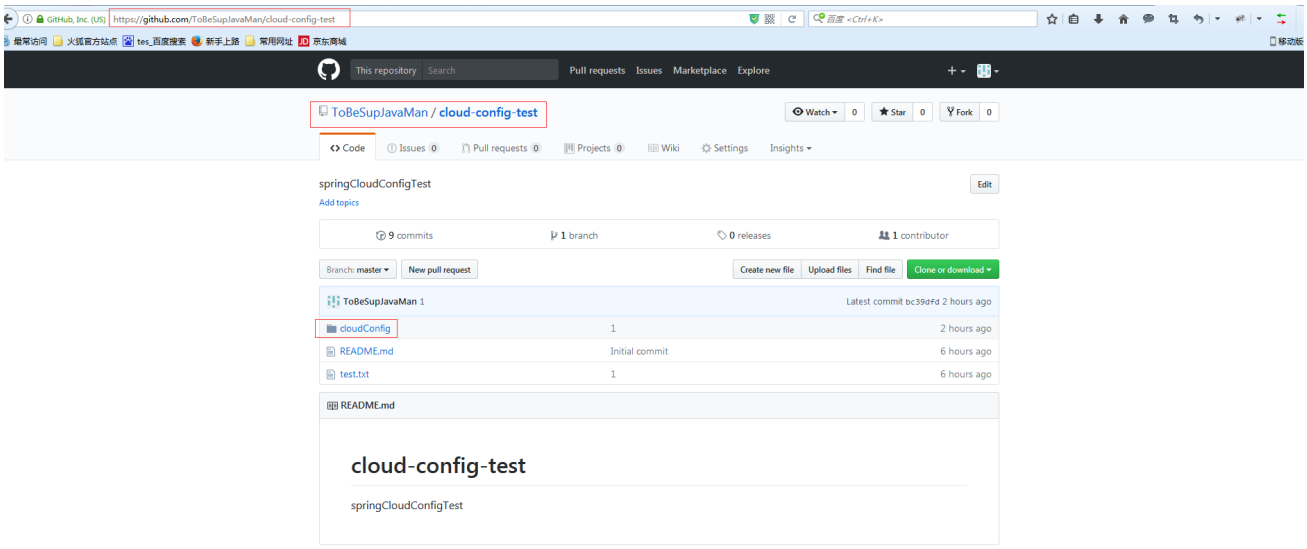
configServer中首先添加依赖，主程序注解，配置文件

在配置文件中

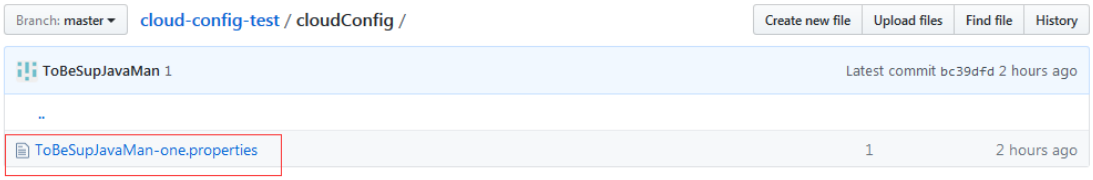
```
spring.application.name=cloud-eruaka-configer
server.port=1119
#https://github.com/ToBeSupJavaMan/cloud-config-test.git也可以是ToBeSupJavaMan为Git上项目的名字，cloud-config-test为仓库名
spring.cloud.config.server.git.uri=https://github.com/ToBeSupJavaMan/cloud-config-test
##在以上目录下的相对路径，如有两层文件夹则用cloudConfig/...写到properties或yml文件上层文件夹为止
spring.cloud.config.server.git.search-paths=cloudConfig
spring.cloud.config.server.git.username=ToBeSupJavaMan
spring.cloud.config.server.git.password=a66666555555
```

红框中内容需要明确，

git上的表现如下



在cloudConfig文件夹中：



即我们的配置文件

以上可以看出

application.properties配置文件中的spring.cloud.config.server.git.uri

其实就是访问仓库的地址

而search-paths只不过是相对uri的配置文件的相对路径的上一级

另：

在git上的配置文件命名方式也是有习惯的

一般是通过git项目名-profile名.properties或.yml(这里profile是uml中的术语是Profiles将stereotypes(版型)、tagged values(标记值)和constraints(约束)应用于具体的模型元素比如类、属性、操作和活动。一个Profile对象就是一系列为特定领域(比如，航空航天、保健、金融)或平台(J2EE、.NET)自定义的UML集合。反正我没怎么明白)

我们可以通过

- /{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，其中{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，就可以访问这个 url：http://localhost:7001/didispace/prod/config-label-test，并获得如下返回信息：

例如

这时说明,配置文件可以被访问到,也就可以来写第二步 client了

第二 spring cloud config client

该工程没什么难度,唯一有意思的是其配置文件bootstrap.properties,只能写成这样是因为该配置文件依赖于configserver的配置以及加载,根据书的第二章可以知道配置文件的加载顺序

1. 在命令行中传入的参数。
2. SPRING_APPLICATION_JSON 中的属性。SPRING_APPLICATION_JSON 是以 JSON 格式配置在系统环境变量中的内容。
3. java:comp/env 中的 JNDI 属性。
4. Java 的系统属性，可以通过 System.getProperties() 获得的内容。
5. 操作系统的环境变量。
6. 通过 random.*配置的随机属性。

7. 位于当前应用 jar 包之外，针对不同{profile}环境的配置文件内容，例如 application-{profile}.properties 或是 YAML 定义的配置文件。
8. 位于当前应用 jar 包之内，针对不同{profile}环境的配置文件内容，例如 application-{profile}.properties 或是 YAML 定义的配置文件。
9. 位于当前应用 jar 包之外的 application.properties 和 YAML 配置内容。
10. 位于当前应用 jar 包之内的 application.properties 和 YAML 配置内容。
11. 在@Configuration 注解修改的类中，通过@PropertySource 注解定义的属性。
12. 应用默认属性，使用 SpringApplication.setDefaultProperties 定义的内容。

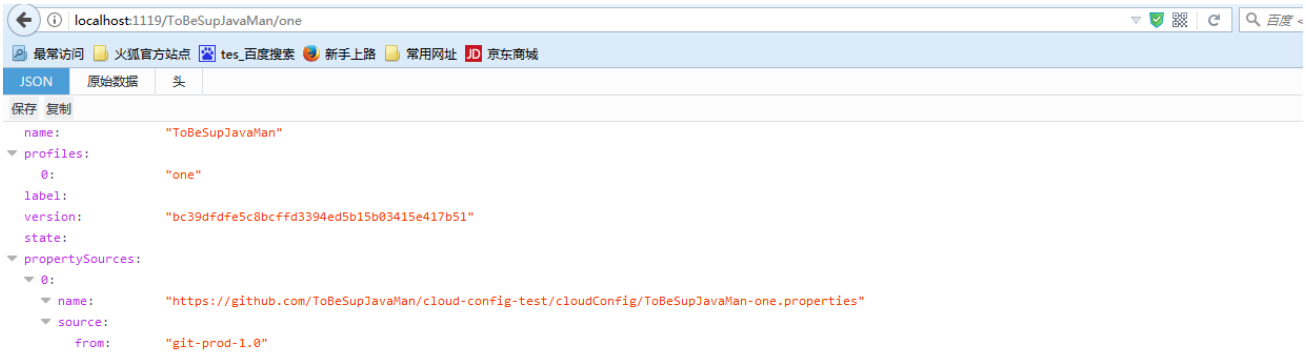
优先级按上面的顺序由高到低，数字越小优先级越高。

可以看到，其中第 7 项和第 9 项都是从应用 jar 包之外读取配置文件，所以，实现外部化配置的原理就是从此切入，为其指定外部配置文件的加载位置来取代 jar 包之内的配置内容。通过这样的实现，我们的工程在配置中就变得非常干净，只需在本地放置开发需要的配置即可，而不用关心其他环境的配置，由其对应环境的负责人去维护即可。

内容应如下:

```
spring.application.name=ToBeSupJavaMan
spring.cloud.config.profile=one
spring.cloud.config.label=master
spring.cloud.config.uri=http://localhost:1119/
server.port=1120
```

可以对比我们在测试配置文件能否别加载的时候访问后得到的结果



这里label是默认的也可以加上去访问.这时这个profile才与项目其他内容有了联系所以也不能乱写.

第三.服务化配置中心

步骤按书中即可.

困难点:

当通过/refresh去访问时报"Full authentication is required to access this resource."的401错误

该错误显示访问资源需要完全身份验证,这是访问的信息有了权限的限制,具体为什么,哪里限制了权限,我觉得应该是spring的springSecurity导致的因为,刷新的信息其实是我们本地clone的git的资源,git上资源已经改变,这时本地却没有,所以需要刷新(重新clone)一下,这时需要访问并改变本地资源也可以说是项目或spring的资源因为本身该资源就是spring自动生成的,spring有其完整的权限,所以该错误信息应该能是spring造成的,这时需要在config-client中的

bootstrap中配置management.security.enabled=false即可.