标识： 密级

版本：001

微服务培训文档(第七讲)

二○一七年十二月

文档修改记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 修改内容描述 | 修改人 | 日期 | 备注 |
| 001 | config学习 | 张洪图 | 2017-12-10 | add |
| 002 | config项目搭建 | 张洪图 | 2017-12-11 | add |
| 003 | config联调配置 | 张洪图 | 2017-12-12 | add |
| 004 | config结束 | 张洪图 | 2017-12-12 | over |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

[讲解计划 4](#_Toc1919917590)

[1.1 config简介 4](#_Toc1827592793)

[1.2使用spring cloud config 5](#_Toc2140309229)

[S1:创建project 5](#_Toc1178408123)

[S2:创建eureka的module 7](#_Toc295727674)

[S3:创建config server module 11](#_Toc473979881)

[S4:创建luck-service的module 16](#_Toc977211857)

# 讲解计划

spring cloud基础入门、eureka服务发现

ribbon负载均衡、hystrix断路器、feign代理

zuul网关、config、bus、stream

说明：第一讲中,spring cloud基础讲解;在第二讲中,讲解注册中心eureka;在第三讲中,讲解负载均衡ribbon;在第四讲中,讲解断路器hystrix;在第五讲中,讲解代理feign;在第六讲中,讲解网关zuul;今天讲解的内容就是config.

zuul和feign内部都包含ribbon和hystrix,可以理解ribbon和hystrix是小组件,zuul和feign内部集成ribbon和hystrix的.然后在项目中可以直接使用zuul和feign就可以.

第七讲就是讲解config内容.

## 1.1 config简介

spring cloud config是spring cloud的全新项目,用来为分布式系统中的基础设施和微服务应用提供集中化的外部配置支持,它分为服务端与客户端两个部分.

服务端称为分布式配置中心,它是一个独立的微服务应用,用来连接配置仓库,并且为客户端提供配置信息,加密和解密信息等访问接口;

客户端则是微服务架构中的各个微服务应用或基础设施,他们通过制定的配置中心来管理应用资源与业务相关的配置内容,并且在启动的时候从配置中心获取和加载配置信息.[如果看不懂,那么代码来体验和查看功能]

spring cloud config实现了对服务端和客户端中环境变量和属性配置的抽象映射,因此它除了适用于spring构建的应用程序外,开可以在任何其他语言中的应用程序中使用.spring cloud config实现的配置中心默认采用git来存储配置信息,所以spring cloud构建的配合服务器,天然就支持对微服务应用配置信息的版本管理,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.它同样支持其他存储方式的支持,比如svn,或本地文件系统.

## 1.2使用spring cloud config

通过spring cloud构建config server, 使用spring cloud config的步骤如下:

1.在pom.xml中引入spring-cloud-server依赖;

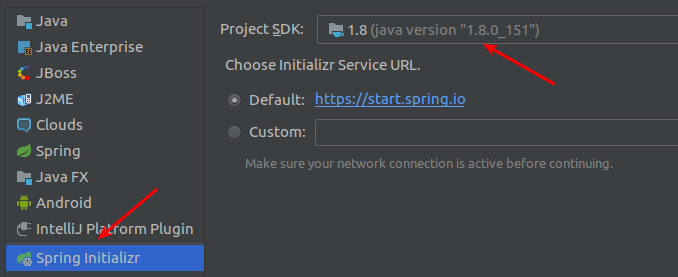
2.在程序主入口中添加@EnableConfigServer注解,开启Config Server;

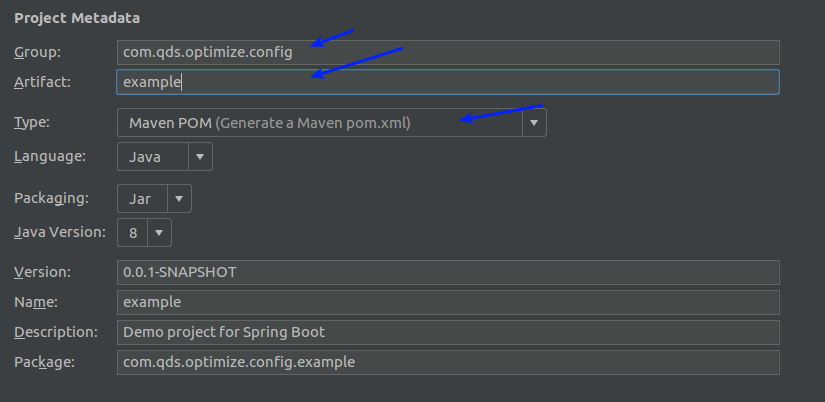
3.最后对yml文件进行git配置即可.

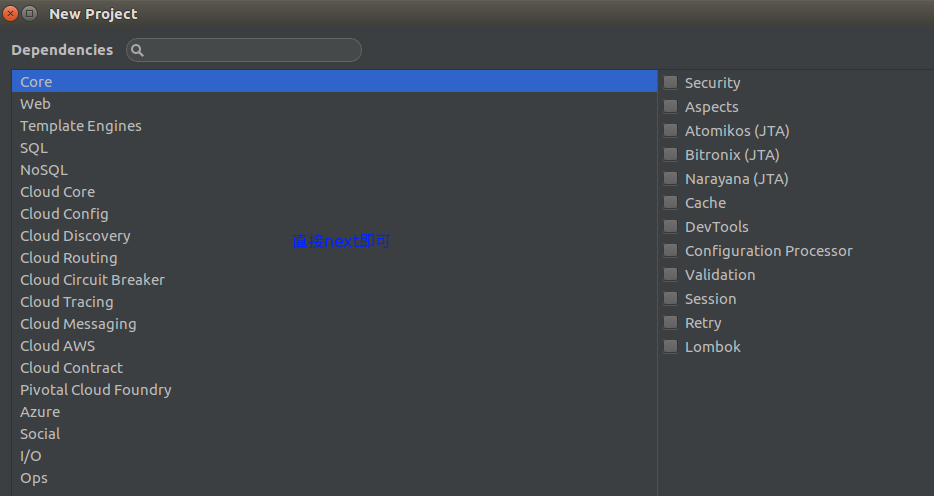
配置中心验证:

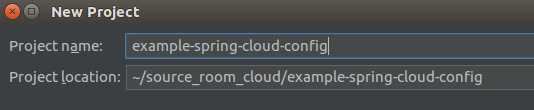
首先编写4个配置文件,然后里面有spring.profiles.active属性,然后设置成不同的数据,然后根据规则去进行验证

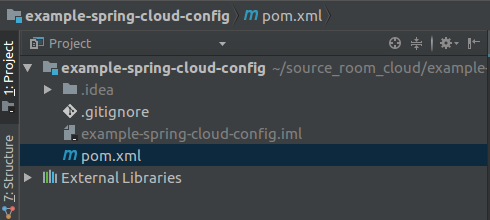
### S1:创建project





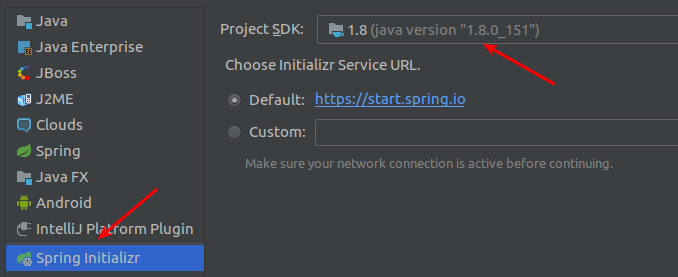


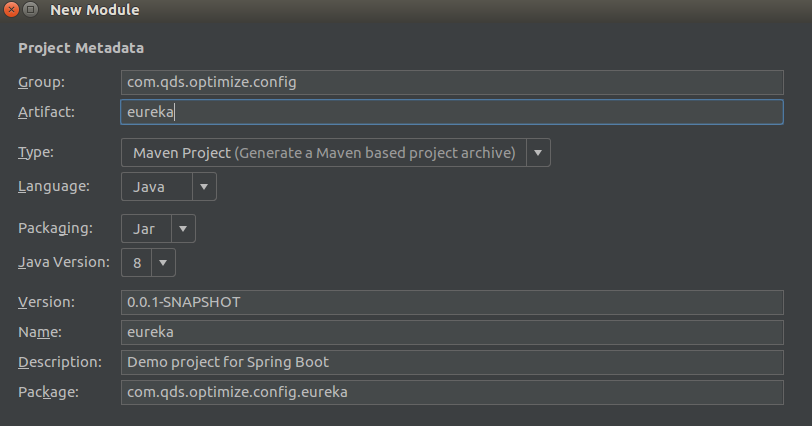


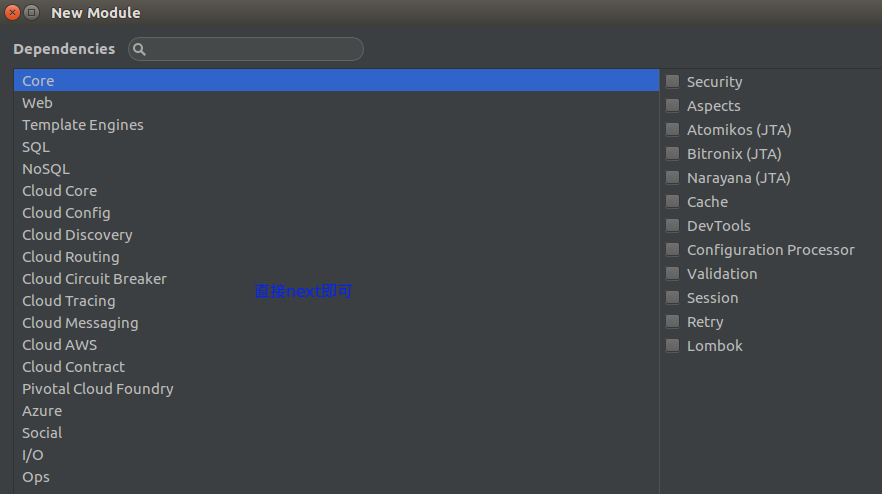


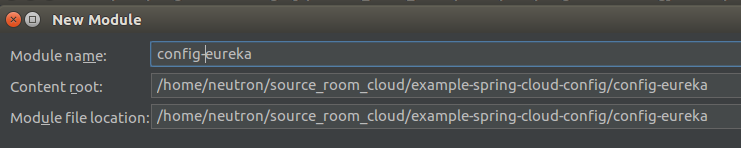
每个人的项目路径自己处理,我重新处理主要是为了不同的知识点放置在不同的路径下面,方便大家查看练习.

### S2:创建eureka的module









注意:一定要对spring cloud的项目的包结构有个清醒的认识,因为spring cloud会按照一定包的规范去查找类的,如果包的命名乱七八糟的,那么很可能很多类是无法扫描的到的.

因为是config的demo,所以都以config开头,因此module name设置为config-name.请参考之前的module创建,来配置入口类,配置文件内容.

只有在2017年2月版本后的内容才会出现Dashboard界面,之前的版本是没有Dashboard的操作界面的.

查看IDEA的about内容如下:

IntelliJ IDEA 2017.2.5

Build #IU-172.4343.14, built on September 26, 2017

Licensed to lan yu

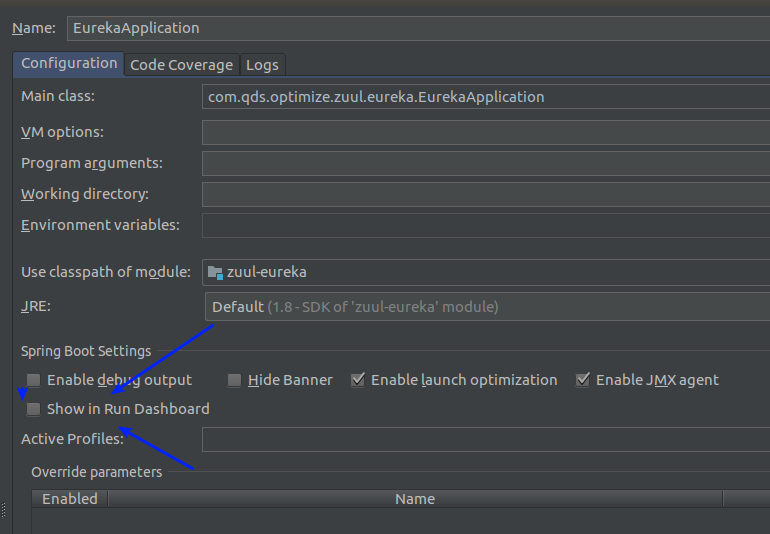
Subscription is active until October 14, 2018

JRE: 1.8.0\_152-release-915-b12 amd64

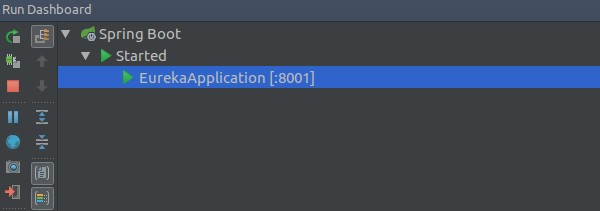
JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o

Linux 4.10.0-38-generic

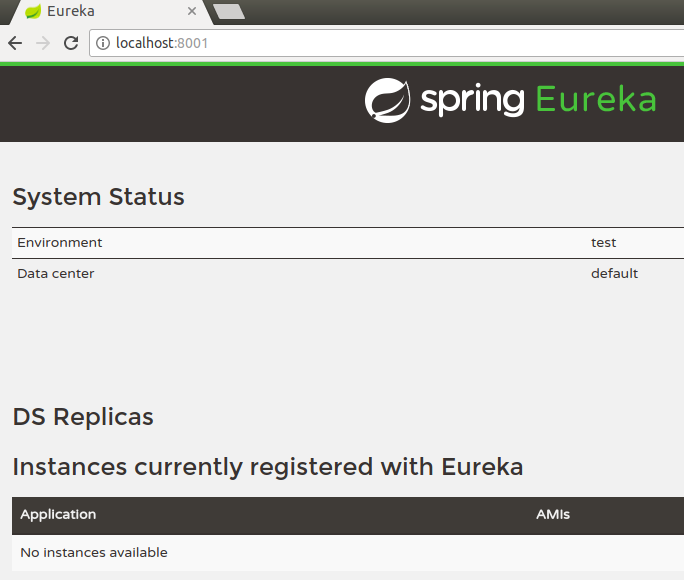
如何出现Dashboard内容.勾选某个module的执行,然后勾选show in Run Dashboard就会出现Dashboard.



然后运行eureka的module



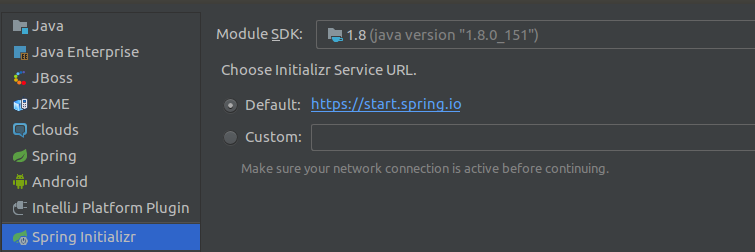
注册中心界面如下:

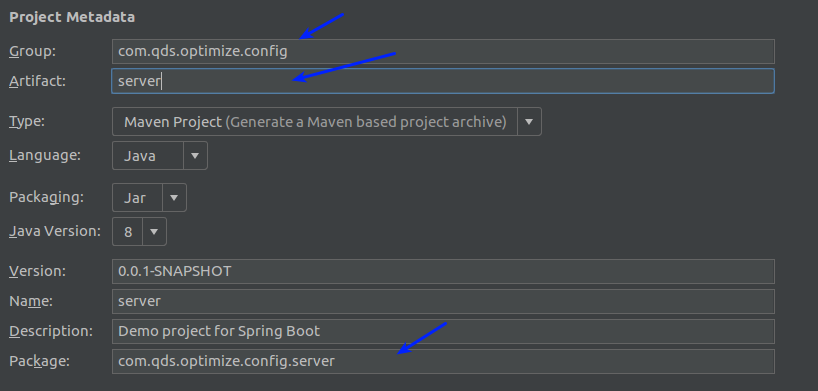


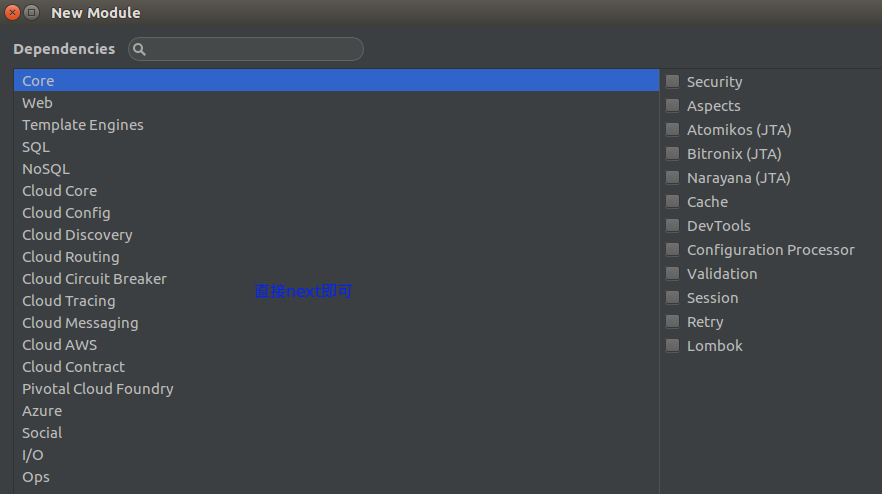
至此注册中心处理完成.

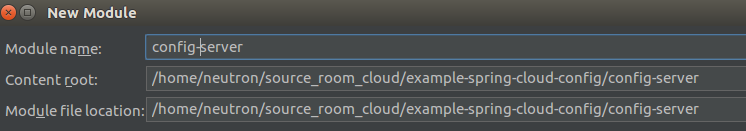
### S3:创建config server module

创建配置中心的server

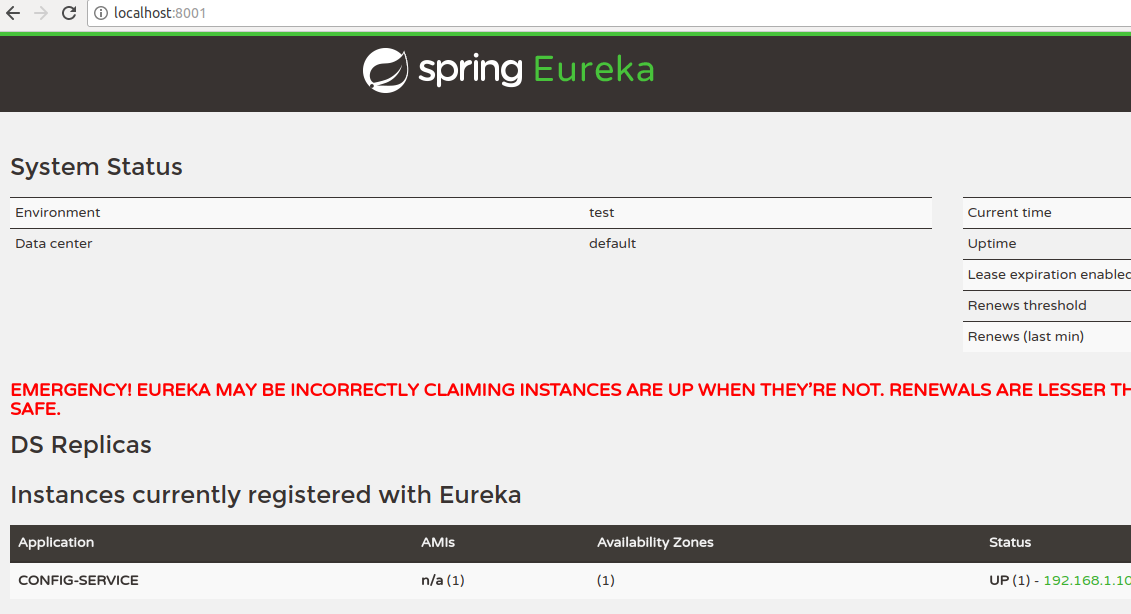








处理config server 的module的pom.xml,yml文件,入口类内容,查看注册中心如下:



根据配置内容,修改pom.xml,yml,入口类如下:

pom.xml如下:

<!-- 配置服务中心必须引入spring-cloud-config-server依赖 -->

<dependency>

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

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

</dependency>

入口类:

/\*\*

\* 配置中心server

\*

\* @author neutron

\*/

// 开启spring cloud config server的配置中心

@EnableConfigServer

@EnableEurekaClient

@SpringBootApplication

public class ServerApplication {

public static void main(String[] args) {

SpringApplication.run(ServerApplication.class, args);

}

}

配置文件内容:

spring:

application:

name: config-service

cloud:

config:

server:

git:

# 以uri作为前缀

uri: https://github.com/zhtzyh2012/example-spring-cloud-config-files

# 相对地址,uri下面的配置文件,后缀名称

# searchPaths: #相对路径 #user-service, hello-service, luck-service

searchPaths: luck-service-url

profiles:

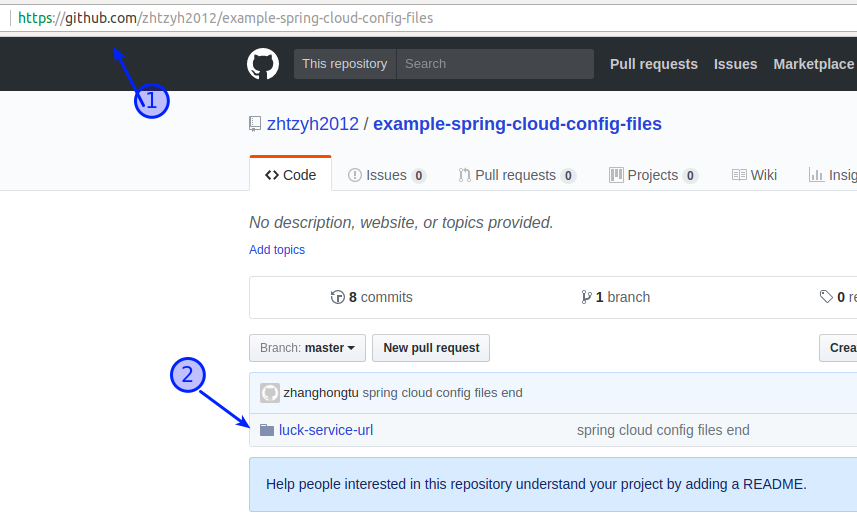
active: $spring.profiles.active$

server:

port: 8002

context-path: /

访问git上的地址uri的路径,上面是我的github地址,案例demo而已,即https://github.com/zhtzyh2012/example-spring-cloud-config-files



其中1就是uri,即项目地址.

相对地址呢,searchPaths的内容是文件夹的路径,图示2的内容,相对路径可以放多个,使用逗号进行间隔.

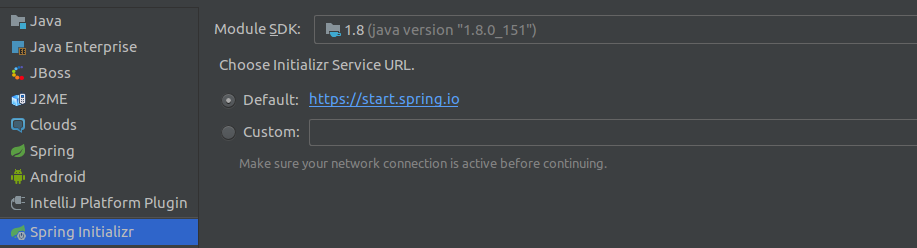
参考spring cloud config的中文文档地址学习:

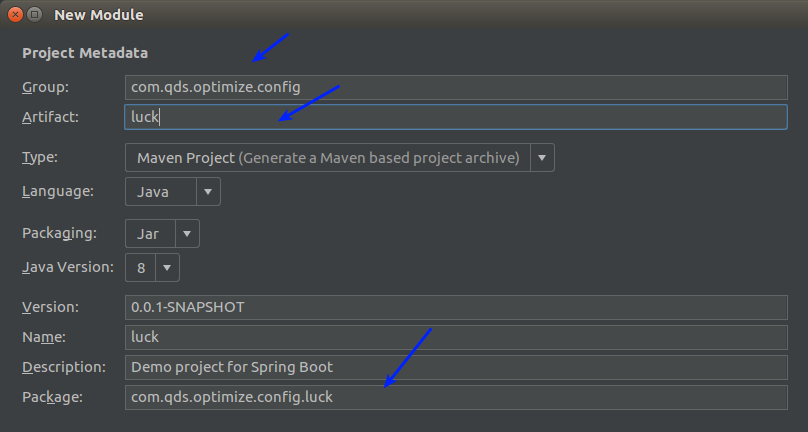
<https://springcloud.cc/spring-cloud-config-zhcn.html>

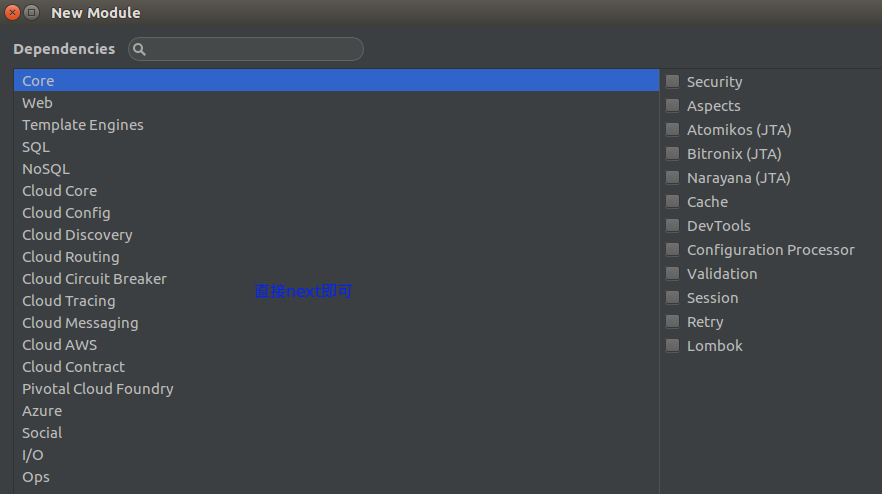
有非常多的配置方式,可以酌情使用,目前我们采用的方式就是不同的项目使用不同的文件夹,然后在对应的文件夹下创建不同环境的配置文件,这样一目了然.

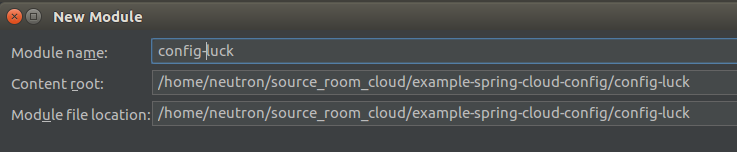
### S4:创建luck-service的module

创建正常的微服务,比如user,或者pay的微服务模块,那么luck-service其实和user等微服务的作用等同.









创建完成luck的服务后,修改pom.xml,yml文件,入口类配置

I1.处理入口类内容:

package com.qds.optimize.config.luck;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.ComponentScan;

/\*\*

\* spring cloud client

\*

\* @author neutron

\*/

@ComponentScan("com.qds.optimize.config.luck.\*\*")

@SpringBootApplication

public class LuckApplication {

public static void main(String[] args) {

SpringApplication.run(LuckApplication.class, args);

}

}

因为是单纯的spring boot项目需要进行必要的扫描,加入到spring容器中管理.

处理yml文件等内容.spring cloud config的约定,如果使用config配置,那么bootstrap.yml文件会优先于application.yml执行,同时建议使用bootstrap.yml,同时可以并行使用application.yml,其中bootstrap.yml作为连接配置使用,application.yml作为spring cloud的配置使用.

II2.application.yml内容如下:

spring:

application:

name: luck-service

profiles:

active: $spring.profiles.active$

server:

port: 8003

bootstrap.yml

spring:

cloud:

config:

# 以uri作为前缀,表示配置中心的位置,即config server的访问路径

uri: http://localhost:8002

# 查询对应的文件名称

name: service-luck

# 分支名称, 默认分支名称是master

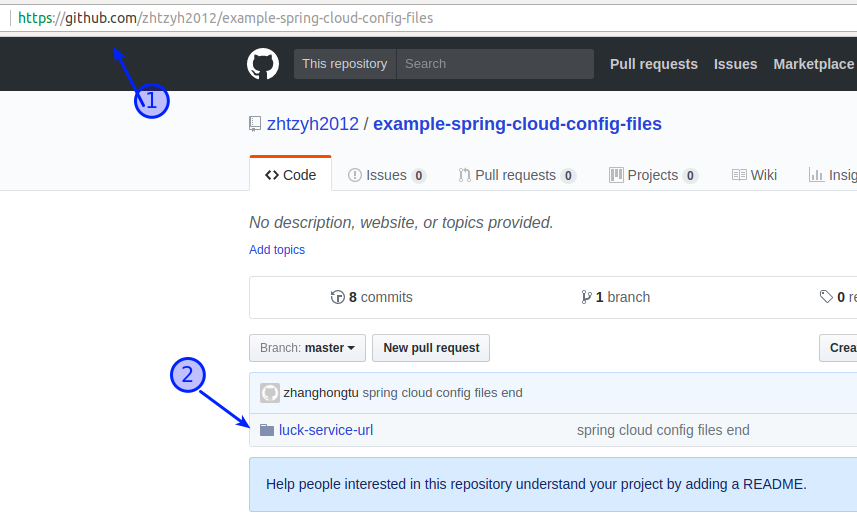
# label: master

# 什么环境,获取spring.profiles.active的变量值

# 如果想获取application.yml中的变量值,使用下面的形式${}

profile: ${spring.profiles.active}

如何定位到git上的文件呢?根据uri去找到配置中心的地址,根据搜索路径,即1和2的内容



然后根据上面的配置内容,即

spring.cloud.config.name

spring.cloud.config.label

spring.cloud.config.profile

去定位配置文件内容,默认的查询规则如下:

所以我们查询的是service-luck-dev.yml,通常label可以省略,默认使用master分支.



III3:现在就是验证性配置生效

package com.qds.optimize.config.luck.web;

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

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

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

/\*\*

\* 使用对外的服务

\*

\* @author neutron

\*/

@RestController

public class LuckApi {

@Value("${hello.name}")

private String luck;

@RequestMapping(value = "/luck")

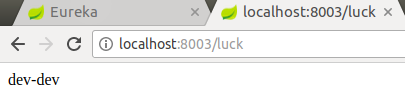
public String luck() {

return luck;

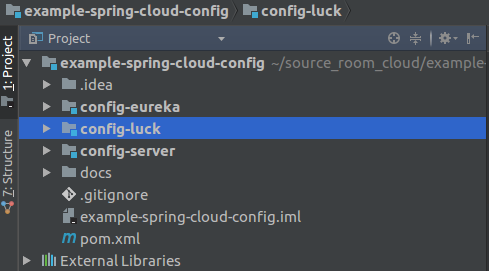
}

}

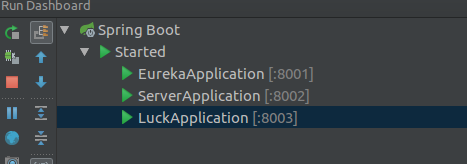
III4:浏览器验证



最后的项目结构如下:



启动的Dashboard如下:



方法的含义和内容,请参考代码中的案例.

不启动项目更新配置,可以生效的内容,稍后添加.其中bus和stream暂时不讲解,以后抽空处理.