

东软睿道内部公开

文件编号: D000-

Spring Cloud微服务架构

版本: 1.0.0

第8章 Config统一配置中心

东软睿道教育信息技术有限公司
(版权所有, 翻版必究)

Copyright © Neusoft Educational Information Technology Co., Ltd
All Rights Reserved



本章教学目标

- ✓ 了解Spring Cloud Config简介；
- ✓ 理解Spring Cloud Config架构图；
- ✓ 掌握配置文件推送到Github；
- ✓ 掌握Config Server编写；
- ✓ 掌握Config Client编写；
- ✓ 掌握Config版的eureka服务端编写；
- ✓ 掌握Config版的User微服务编写；

本章教学内容

节	知识点	掌握程度	难易程度	教学形式	对应在线微课
Spring Cloud Config简介	为什么要统一管理微服务配置	了解		线下	
	Spring Cloud Config简介	了解		线下	
	架构图	理解		线下	
编写Config Server	将配置文件推送到Github	掌握	难	线下	
	编写Config Server	掌握		线下	
	Config Server的端点	了解		线下	
	测试Config Server	掌握		线下	
编写Config Client	编写Config Client	掌握		线下	
	application和bootstrap配置文件的区别	掌握		线下	
	测试Config Client	掌握		线下	
Config配置实战	编写Git配置文件	掌握		线下	
	编写Config版的Eureka服务端	掌握		线下	
	编写Config版的User微服务	掌握		线下	

CONTENTS

目录

01

Spring Cloud Config简介

02

编写Config Server

03

编写Config Client

04

Config配置实战

为什么要统一管理微服务配置

- ❖ 对于传统的单体应用，常使用配置文件管理所有配置。
- ❖ 微服务意味着要将单体应用中的业务拆分成一个个子服务，每个服务的粒度相对较小，因此系统中会出现大量的服务。
- ❖ 每一个微服务自带着一个`application.properties`或者`application.yml`配置文件。
- ❖ 一个使用微服务架构的应用系统可能包含上百个微服务，因此一套集中式的、运行期间动态调整的、自动刷新的配置管理设施是必不可少的。
- ❖ Spring Cloud提供了Config Server来解决这个问题。

Spring Cloud Config简介

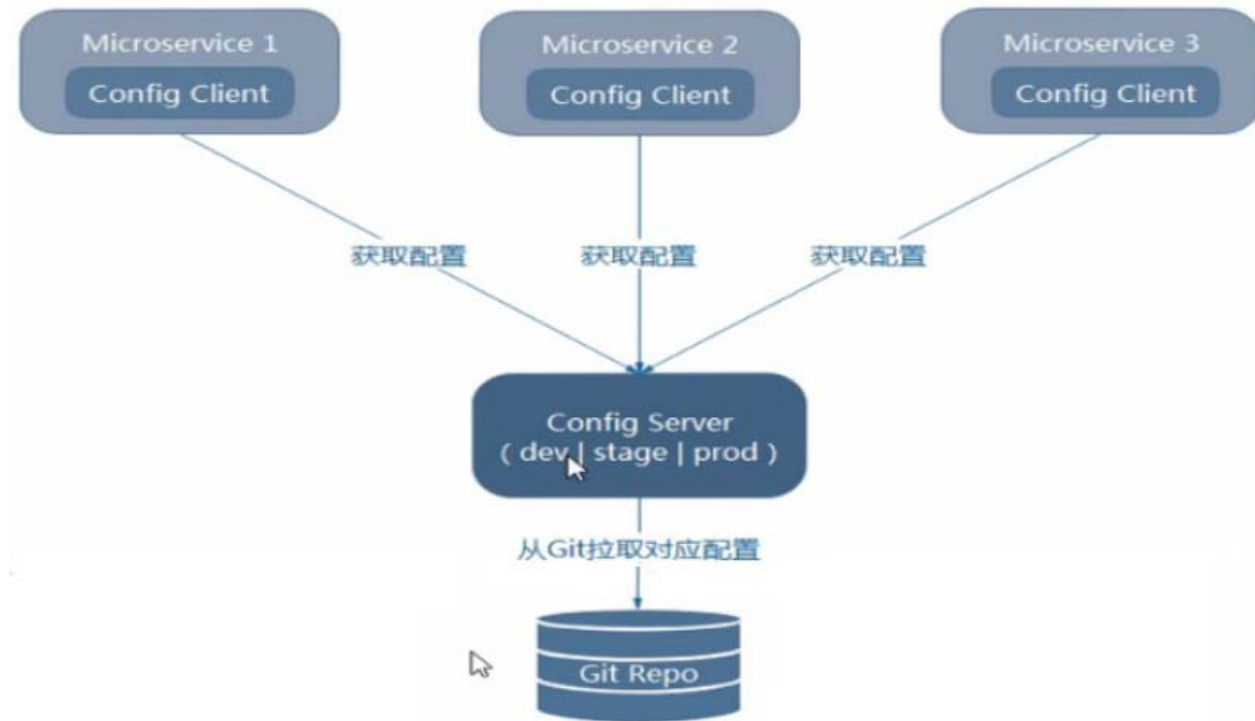
- ❖ Spring Cloud Config为微服务架构中的微服务提供集中化的外部配置支持，配置服务器为各个不同微服务应用的所有环境提供了一个中心化的外部配置。

Spring Cloud Config简介

- ❖ Spring Cloud Config包括Config Server和Config Client两部分。由于Config Server和Config Client都实现了对Spring Environment和PropertySource抽象的映射，因此，Spring Cloud非常适合Spring应用程序，当然也可与任何其他语言编写的应用程序配合使用。
- ❖ Config Server是一个可横向扩展、集中式的配置服务器，它用于集中管理应用程序各个环境下的配置，默认使用Git存储配置内容(也可使用Subversion、本地文件系统或Vault存储配置)，因此可以方便的实现对配置的版本控制与内容审计。
- ❖ Config Client 是Config Server的客户端，用于操作存储在Config Server中的配置属性。

架构图

- ❖ 所有的微服务都指向Config Server。
- ❖ 各个微服务在启动时，会请求Config Server以获取所需要的配置属性，然后缓存这些属性以提高性能。



CONTENTS

目录

01

Spring Cloud Config简介

02

编写Config Server

03

编写Config Client

04

Config配置实战

将配置文件推送到Github

❖ 实现步骤

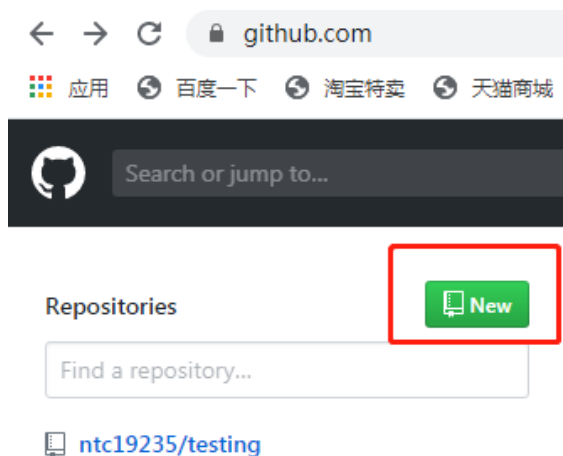
- ▶ 在GitHub上新建一个名为springcloud-config的新Repository
- ▶ 本地硬盘中clone git仓库
- ▶ 在本地仓库里面新建一些配置文件

```
microservice-foo.properties  
microservice-foo-dev.properties  
microservice-foo-test.properties  
microservice-foo-production.properties
```

- ▶ 将配置文件推送到Github上

将配置文件推送到Github

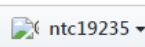
❖ 在GitHub上新建一个名为springcloud-config的新Repository



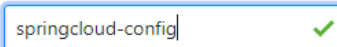
Create a new repository

A repository contains all project files, including the revision history. Already have a repository? [Import a repository.](#)

Owner



Repository name *



Great repository names springcloud-config is available. inspiration? How about...

Description (optional)

☒ Public

Anyone can see this repository. You choose who can commit.

☐ Private

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ Initialize this repository with a README

This will let you immediately clone the repository to your computer.

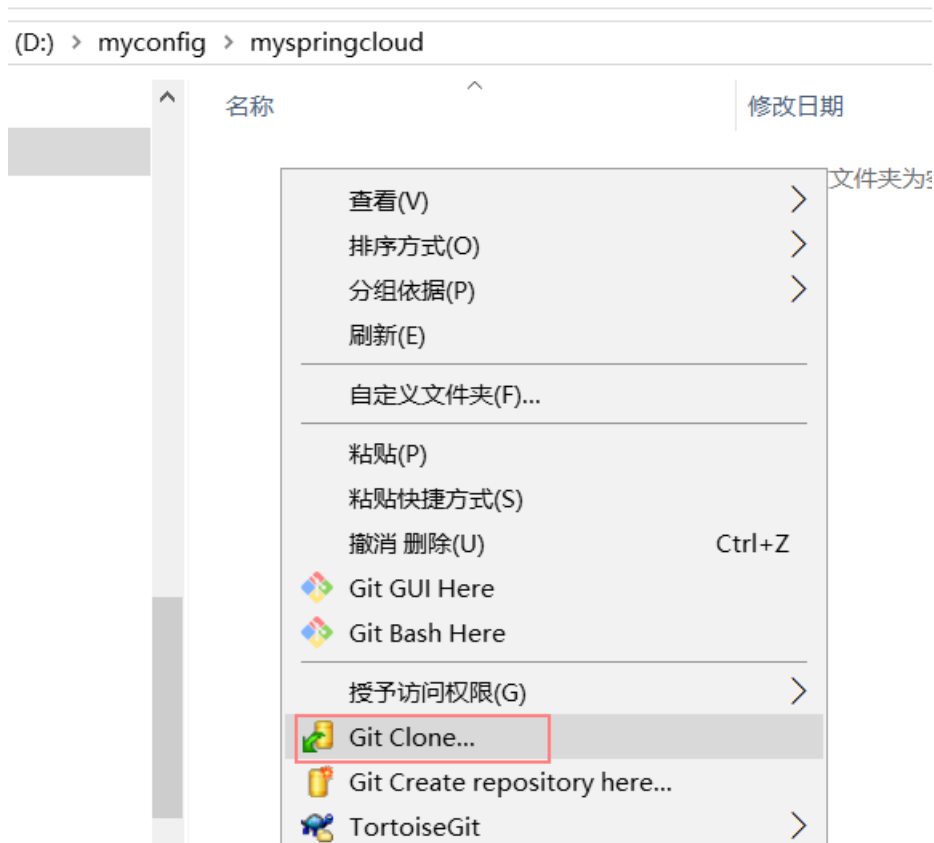
Add .gitignore: None ▾

Add a license: None ▾ ⓘ

Create repository

将配置文件推送到Github

❖ 本地硬盘目录上clone git仓库



将配置文件推送到Github

- ❖ 在本地仓库D:\myconfig\myspringcloud\mscloud-config里面新建几个配置文件：

```
microservice-foo.properties  
microservice-foo-dev.properties  
microservice-foo-test.properties  
microservice-foo-production.properties
```

- ▶ 文件内容分别是：

```
profile=default-1.0  
profile=dev-1.0  
profile=test-1.0  
profile=production-1.0
```

将配置文件推送到Github

- ❖ 将上一步的配置文件推送到Github上
 - ▶ 文件浏览器上，选择上传文件，点右键菜单的TortoiseGit中的Add
 - ▶ 然后选择Commit
 - ▶ 文件浏览器，返回上一层目录
 - ▶ 在文件目录上，点右键，选择TortoiseGit中的Push
 - ▶ 然后选择OK

将配置文件推送到Github

❖ 将上一步的配置文件推送到Github上

g > myspringcloud > mscloud-config

名称

.git

microservice-foo.properties

microservice-foo-dev.pro

microservice-foo-produ

microservice-foo-test.pro

ntc19235 / springcloud-config

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

No description, website, or topics provided.

Manage topics

1 commit

1 branch

0 packages

0 releases

1 contributor

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

ntc19235 create

Latest commit fa95c40 1 hour ago

microservice-foo-dev.properties

create

1 hour ago

microservice-foo-production.properties

create

1 hour ago

microservice-foo-test.properties

create

1 hour ago

microservice-foo.properties

create

1 hour ago

Help people interested in this repository understand your project by adding a README.

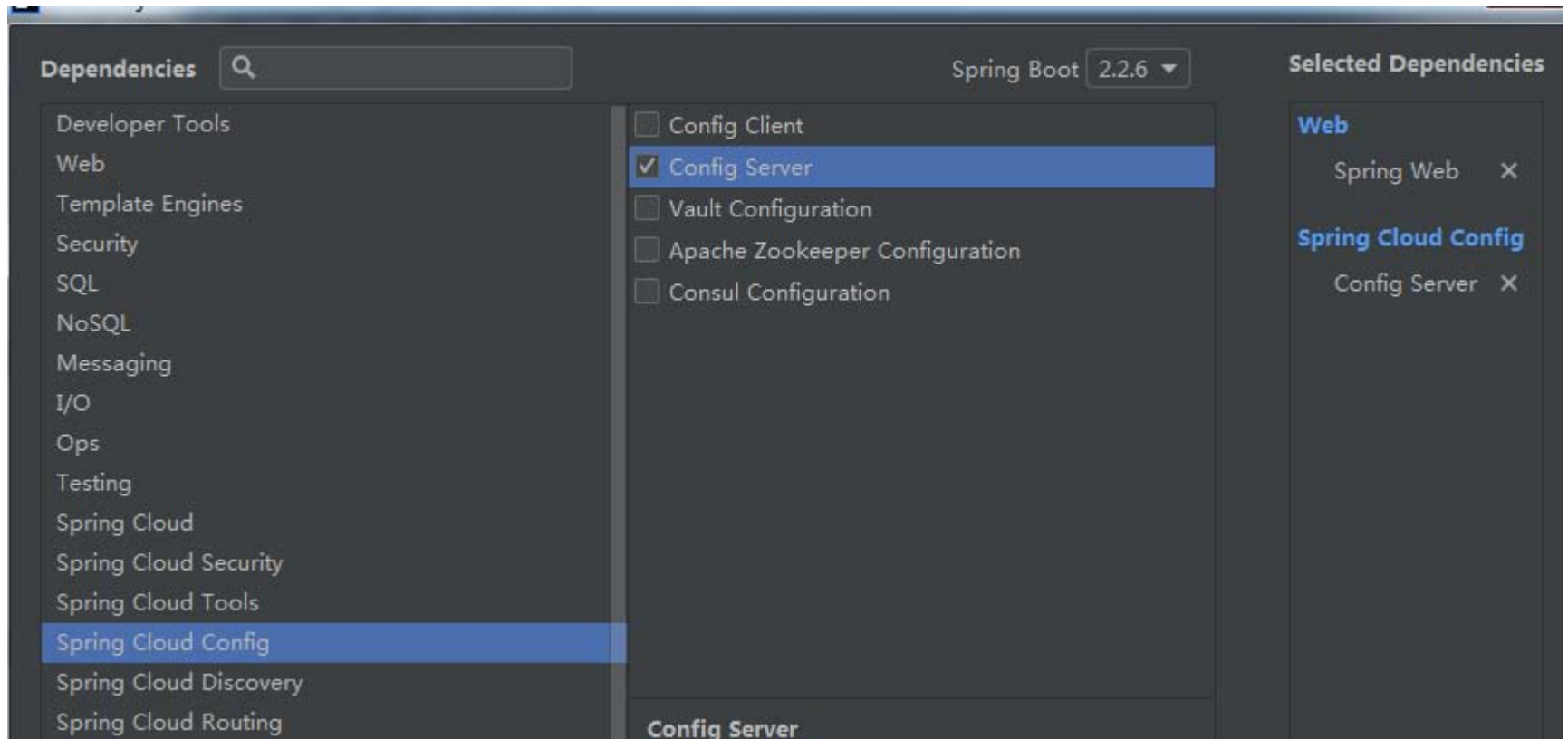
Add a README

编写Config Server

- ❖ 新建工程config，7755端口，即为Cloud的配置中心模块
 - ▶ 创建工程
 - ▶ 全局配置文件application.yml
 - ▶ 主启动类添加@EnableConfigServer
 - ▶ 修改hosts文件，增加映射

编写Config Server

❖ 新建工程



编写Config Server

❖ 全局配置文件

spring:

application:

name: config-server

cloud:

config:

server:

git:

uri: [https://github.com/ntc19235/springcloud-](https://github.com/ntc19235/springcloud-config.git)

[config.git](https://github.com/ntc19235/springcloud-config.git)

label: master

server:

port: 7755



编写Config Server

- ❖ 主启动类
 - ▶ 添加@EnableConfigServer

编写Config Server

- ❖ 修改hosts
 - ▶ 添加映射

```
C:\Windows\System32\drivers\etc
```

- ▶ 127.0.0.1 config.com

Config Server的端点

- ❖ 可以使用Config Server的端点获取配置文件的内容。端点与配置文件的映射规则如下：

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

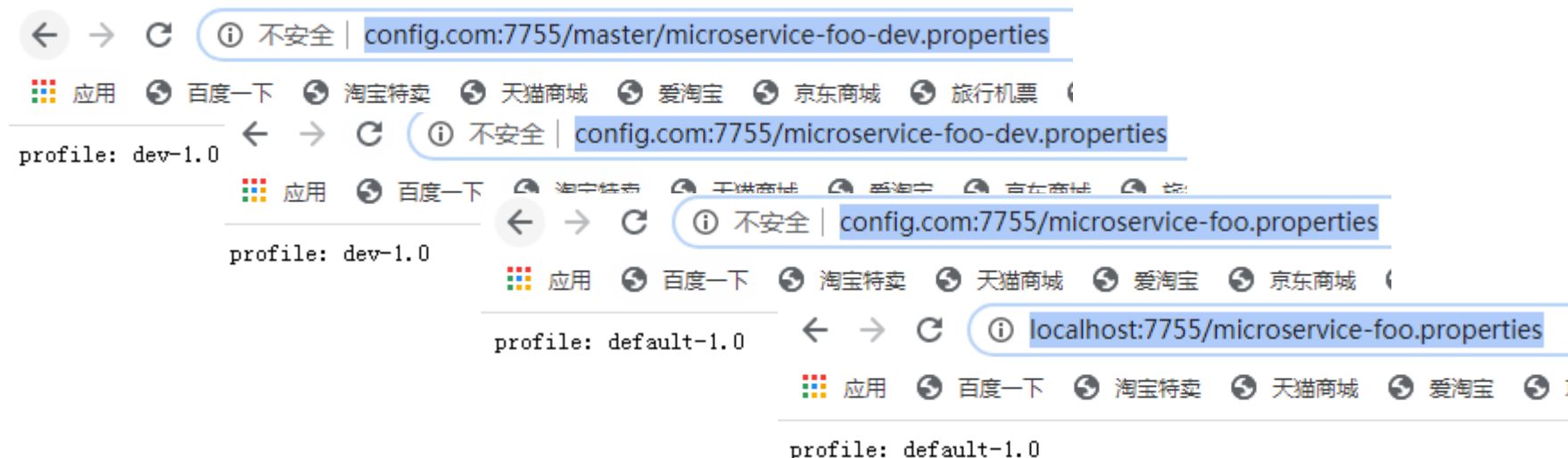
- ❖ 以上端点都可以映射到 {application} - {profile} . properties 这个配置文件， {application} 表示微服务的名称， {label} 对应Git仓库的分支，默认是master。

Config Server的端点

- ❖ 按照以上规则，可以使用以下URL访问到Git仓库master分支的 `microservice-foo-dev.properties`：
 - ▶ `http://localhost:7755/microservice-foo/dev`
 - ▶ `http://localhost:7755/microservice-foo-dev.properties`
 - ▶ `http://localhost:7755/master/microservice-foo-dev.properties`

测试Config Server

- ❖ 测试通过Config微服务从GitHub上获取配置内容
 - ▶ 启动mscloud-config-server-7755
 - ▶ <http://config.com:7755/master/microservice-foo-dev.properties>
 - ▶ <http://config.com:7755/microservice-foo-dev.properties>
 - ▶ <http://config.com:7755/microservice-foo.properties>
 - ▶ <http://localhost:7755/microservice-foo.properties>



CONTENTS

目录

01

Spring Cloud Config简介

02

编写Config Server

03

编写Config Client

04

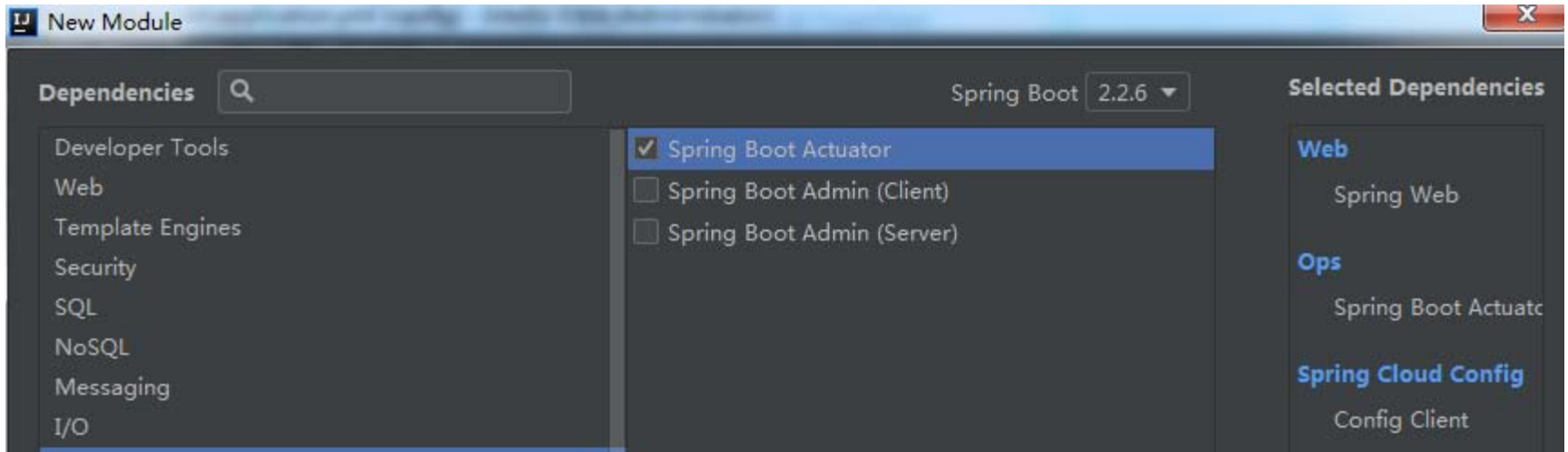
Config配置实战

编写Config Client

- ❖ 新建Module模块config-client, 端口7766
 - ▶ 新建模块
 - ▶ 编写全局配置文件application.yml
 - ▶ 创建配置文件bootstrap.properties
 - ▶ 编写主启动类
 - ▶ 修改hosts文件, 增加映射
 - ▶ 编写controller

编写Config Client

❖ 新建Module模块mscloud-config-client-7766



编写Config Client

❖ 创建bootstrap.properties配置文件

#需要从github上读取的资源名称，注意没有后缀名
`spring.cloud.config.name=microservice-foo`

#配置服务中心

`spring.cloud.config.uri=http://config.com:7755/`

#配置文件所在分支

`spring.cloud.config.label=master`

#配置文件所指环境

`spring.cloud.config.profile=test`

编写Config Client

❖ bootstrap.properties配置文件

- ▶ `spring.cloud.config.name` 对应Config Server所获取的配置文件中的{application}
- ▶ `spring.cloud.config.uri` 指定Config Server的地址
- ▶ `spring.cloud.config.label` 指定Git仓库的分支，对应于Config Server所获取的配置文件的{label}
- ▶ `spring.cloud.config.profile` profile对应Config Server所获取的配置文件中的{profile}

编写Config Client

- ❖ application.yml全局配置文件
 - ▶ spring:
 - application:
 - name: config-client
 - server:
 - port: 7766

编写Config Client

❖ 主启动类

- ▶ 创建一个基本的Spring Boot启动类
- ▶ 只添加@SpringBootApplication注解

编写Config Client

- ❖ 修改hosts
 - ▶ 添加映射

```
C:\Windows\System32\drivers\etc
```

```
127.0.0.1    client-config.com
```

编写Config Client

❖ 编写Controller

```
@RestController
public class ConfigClientController {
    @Value("${profile}")
    private String profile;

    @GetMapping("/profile")
    public String hello() {
        return this.profile;
    }
}
```


application和bootstrap配置文件的区别

❖ 首先都是属于配置文件，功能一样。主要是区别于如下：

▶ 1、application和bootstrap的加载顺序

- ★ Bootstrap.yml (bootstrap.properties) 在application.yml (application.properties) 之前加载，用于应用程序上下文的引导阶段

▶ 2、典型场景

- ★ 1. 当使用 Spring Cloud Config Server的时候，应该在 bootstrap.application里面指定 spring.cloud.config.name 和 spring.cloud.config.uri
- 2. 一些加密/解密的信息

▶ 3、属性覆盖问题

- ★ 启动上下文时，Spring Cloud会创建一个Bootstrap Context，作为Spring应用的Application Context的父上下文。初始化的时候，Bootstrap Context负责从外部源加载配置属性并解析配置。这两个上下文共享一个从外部获取的Environment。Bootstrap属性有高优先级，默认情况下，它们不会被本地配置覆盖。
- ★ Bootstrap context和Application Context有着不同的约定，所以新增了一个bootstrap.yml文件，而不是使用application.yml (或者 application.properties)。保证Bootstrap Context和Application Context配置的分离。

测试Config Client

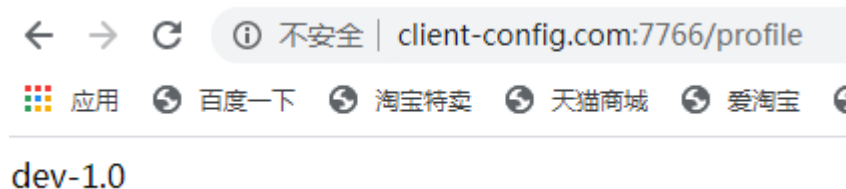
- ❖ 测试通过Config微服务从GitHub上获取配置内容
 - ▶ 启动config, 端口7755
 - ▶ 启动config-client, 端口7766
 - ▶ 测试 `http://config.com:7755/microservice-foo/dev.properties`



A screenshot of a web browser window. The address bar shows the URL `config.com:7755/microservice-foo/dev.properties`. The page content displays a JSON object representing configuration data for a microservice.

```
{
  "name": "microservice-foo",
  "profiles": [
    {
      "name": "dev.properties",
      "label": null,
      "version": "fa95c40ffdf2146297171b45f2a2cf9d0f08db38",
      "state": null,
      "propertySources": [
        {
          "name": "https://github.com/ntcl9235/springcloud-config.git/microservice-foo.properties",
          "source": {
            "profile": "default-1.0"
          }
        }
      ]
    }
  ]
}
```

- ▶ 测试: `http://client-config.com:7766/profile`



A screenshot of a web browser window. The address bar shows the URL `client-config.com:7766/profile`. The page content displays the text `dev-1.0`.

dev-1.0

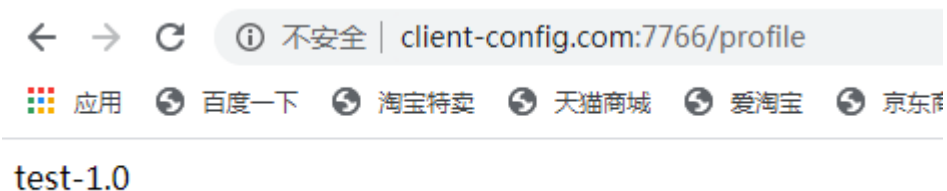
测试Config Client

❖ 修改bootstrap.properties配置文件的profile属性，再次测试



```
bootstrap.properties
1 #需要从github上读取的资源名称，注意没有后缀名
2 spring.cloud.config.name=microservice-foo
3
4 #配置服务中心
5 spring.cloud.config.uri=http://config.com:7755/
6
7 #配置文件所在分支
8 spring.cloud.config.label=master
9 #配置文件所指环境
10 spring.cloud.config.profile=test
11
```

- ▶ 启动config，端口7755
- ▶ 启动config-client，端口7766
- ▶ 测试 <http://client-config.com:7766/profile>



CONTENTS

目录

01

Spring Cloud Config简介

02

编写Config Server

03

编写Config Client

04

Config配置实战

Config配置实战

- ❖ 此时，Config服务端配置配置OK且测试通过，我们可以用Config + GitHub进行配置修改并获得内容。
- ❖ 现在，我们做一个Eureka服务 + 一个User访问的微服务，将两个微服务的配置统一由GitHub获得，实现统一配置分布式管理，完成多环境的变更。

编写Git配置文件

- ❖ 在本地仓库里面新建配置文件，分别对应Eureka服务和User访问的服务的配置文件，然后推送到Github上
 - ▶ `mscloud-config-eureka-client-dev.properties`
 - ▶ `mscloud-config-eureka-client-test.properties`
 - ▶ `mscloud-config-user-client-dev.properties`
 - ▶ `mscloud-config-user-client-test.properties`

 mscloud-config-eureka-client-dev.properties	config file	38 minutes ago
 mscloud-config-eureka-client-test.properties	config file	38 minutes ago
 mscloud-config-user-client-dev.properties	config file	38 minutes ago
 mscloud-config-user-client-test.properties	config file	38 minutes ago

编写Git配置文件

❖ mscloud-config-eureka-client-dev.properties内容

#配置服务名称及端口

spring.application.name=regist-center

server.port=7001

#—————服务注册中心配置—————

#服务注册中心实例的主机名

eureka.instance.hostname=eureka.com

#是否向服务注册中心注册自己

eureka.client.register-with-eureka=false

#是否检索服务

eureka.client.fetch-registry=false

#服务注册中心的配置内容，指定服务注册中心的位置

eureka.client.serviceUrl.defaultZone=http://eureka.com:7001/eureka/

#读取对等节点服务器复制的超时的时间，单位为毫秒，默认为200

eureka.server.peer-node-read-timeout-ms=5000

编写Git配置文件

❖ msccloud-config-eureka-client-test.properties内容

#配置服务名称及端口

spring.application.name=regist-center

server.port=7002

#—————服务注册中心配置—————

#服务注册中心实例的主机名

eureka.instance.hostname=eureka2.com

#是否向服务注册中心注册自己

eureka.client.register-with-eureka=false

#是否检索服务

eureka.client.fetch-registry=false

#服务注册中心的配置内容，指定服务注册中心的位置

eureka.client.serviceUrl.defaultZone=http://eureka2.com:7002/eureka/

#读取对等节点服务器复制的超时的时间，单位为毫秒，默认为200

eureka.server.peer-node-read-timeout-ms=5000

编写Git配置文件

❖ mscloud-config-user-client-dev.properties内容

```
server.port=8001
# url
spring.datasource.url=jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8&serverTimezone=GMT%2B8
# username
spring.datasource.username=root
# password
spring.datasource.password=123456
# driver
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#配置服务名称
spring.application.name=provider
#客户端注册进eureka服务列表内
eureka.client.service-url.defaultZone=http://localhost:7001/eureka/
eureka.instance.instance-id: user-provider
#访问路径可以显示IP地址
eureka.instance.prefer-ip-address: true

info.app.name: provider
info.company.name: neusoft
info.build.artifactId: $project.artifactId$
info.build.version: $project.version$
```

编写Git配置文件

❖ mscloud-config-user-client-test.properties内容

```
server.port=8002
```

```
# url
```

```
spring.datasource.url=jdbc:mysql://localhost:3306/mybatis2?characterEncoding=utf-8&serverTimezone=GMT%2B8
```

```
# username
```

```
spring.datasource.username=root
```

```
# password
```

```
spring.datasource.password=123456
```

```
# driver
```

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

```
#配置服务名称
```

```
spring.application.name=provider
```

```
#客户端注册进eureka服务列表内
```

```
eureka.client.service-url.defaultZone=http://localhost:7002/eureka/
```

```
eureka.instance.instance-id: user-provider2
```

```
#访问路径可以显示IP地址
```

```
eureka.instance.prefer-ip-address: true
```

```
info.app.name: provider2
```

```
info.company.name: neusoft
```

```
info.build.artifactId: $project.artifactId$
```

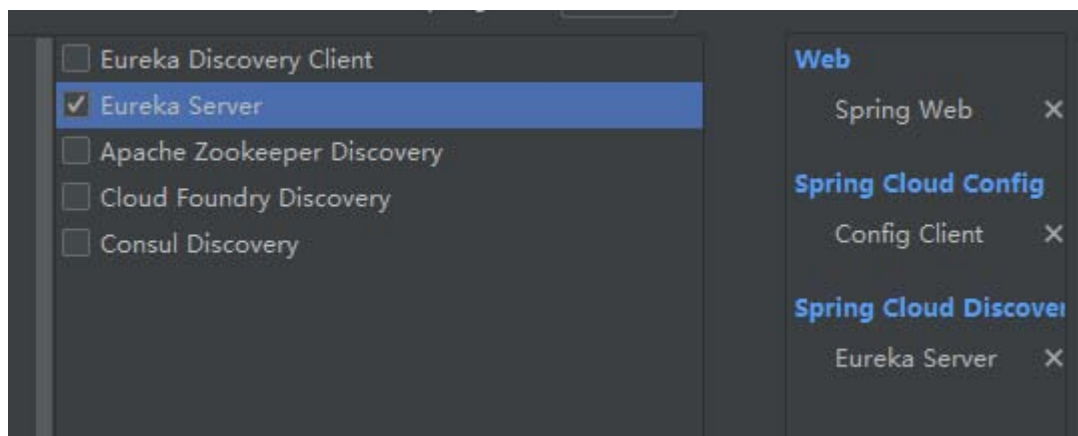
```
info.build.version: $project.version$
```

编写Config版的Eureka服务端

- ❖ 新建工程config_test, 端口7001
 - ▶ 新建工程
 - ▶ 编写全局配置文件application.yml
 - ▶ 创建配置文件bootstrap.properties
 - ▶ 编写主启动类
 - ▶ 测试

编写Config版的Eureka服务端

❖ 新建工程config_test, 端口7001



编写Config版的Eureka服务端

- ❖ 编写全局配置文件application.yml
 - ▶ 根据config工程里的config_client，端口7766改写

```
spring:  
  application:  
    name: regist-center
```

编写Config版的Eureka服务端

❖ 创建配置文件bootstrap.properties

- ▶ 根据config工程里的config_client，端口7766改写

#需要从github上读取的资源名称，注意没有后缀名
spring.cloud.config.name=mscloud-config-eureka-client

#配置服务中心
spring.cloud.config.uri=http://config.com:7755/

#配置文件所在分支
spring.cloud.config.label=master
#配置文件所指环境
#spring.cloud.config.profile=dev
spring.cloud.config.profile=test

编写Config版的Eureka服务端

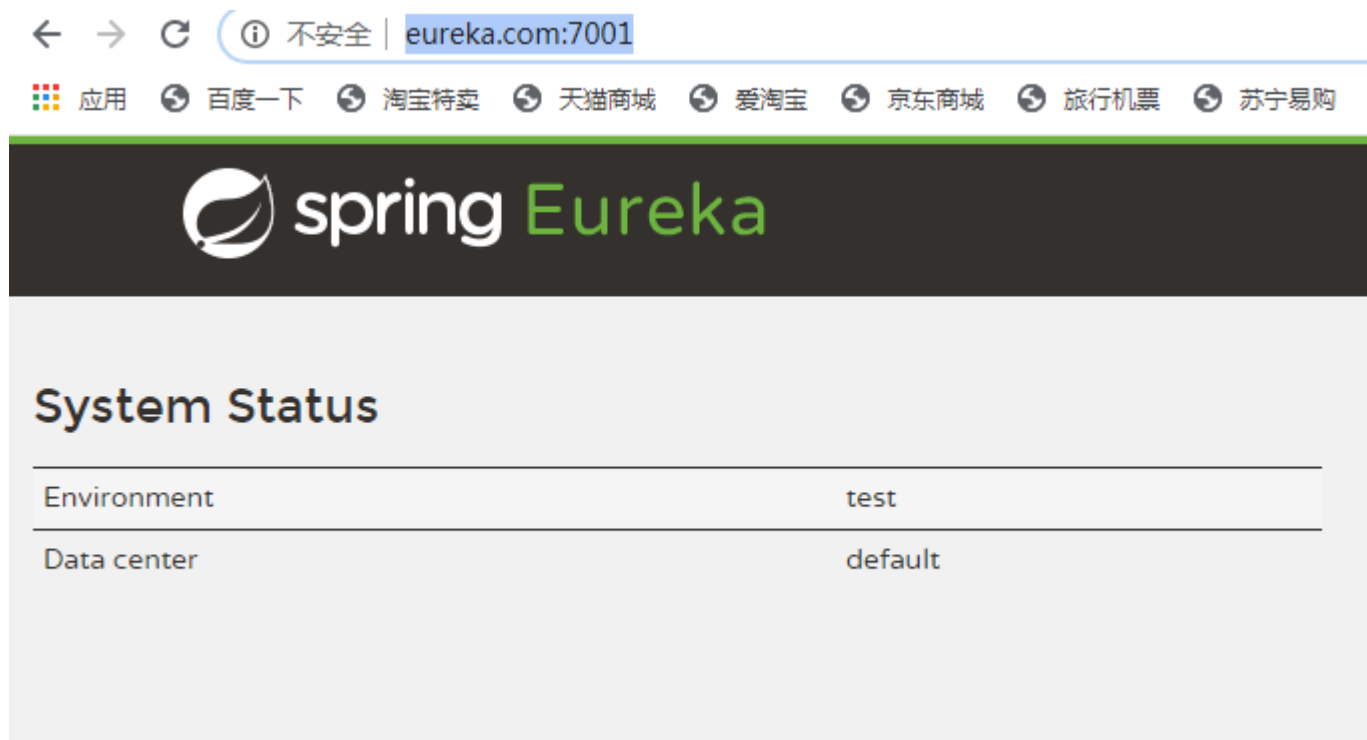
❖ 编写主启动类

添加@EnableEurekaServer注解

编写Config版的Eureka服务端

❖ 测试

- ▶ 先启动config, 端口7755微服务
- ▶ 再启动config_test, 7001微服务 (bootstrap里配置为dev环境)
- ▶ 浏览器 <http://eureka.com:7001/>



编写Config版的Eureka服务端

❖ 测试

- ▶ 修改bootstrap中的spring.cloud.config.profile=test
- ▶ 重启config_test工程
- ▶ 浏览器 <http://eureka2.com:7002/>

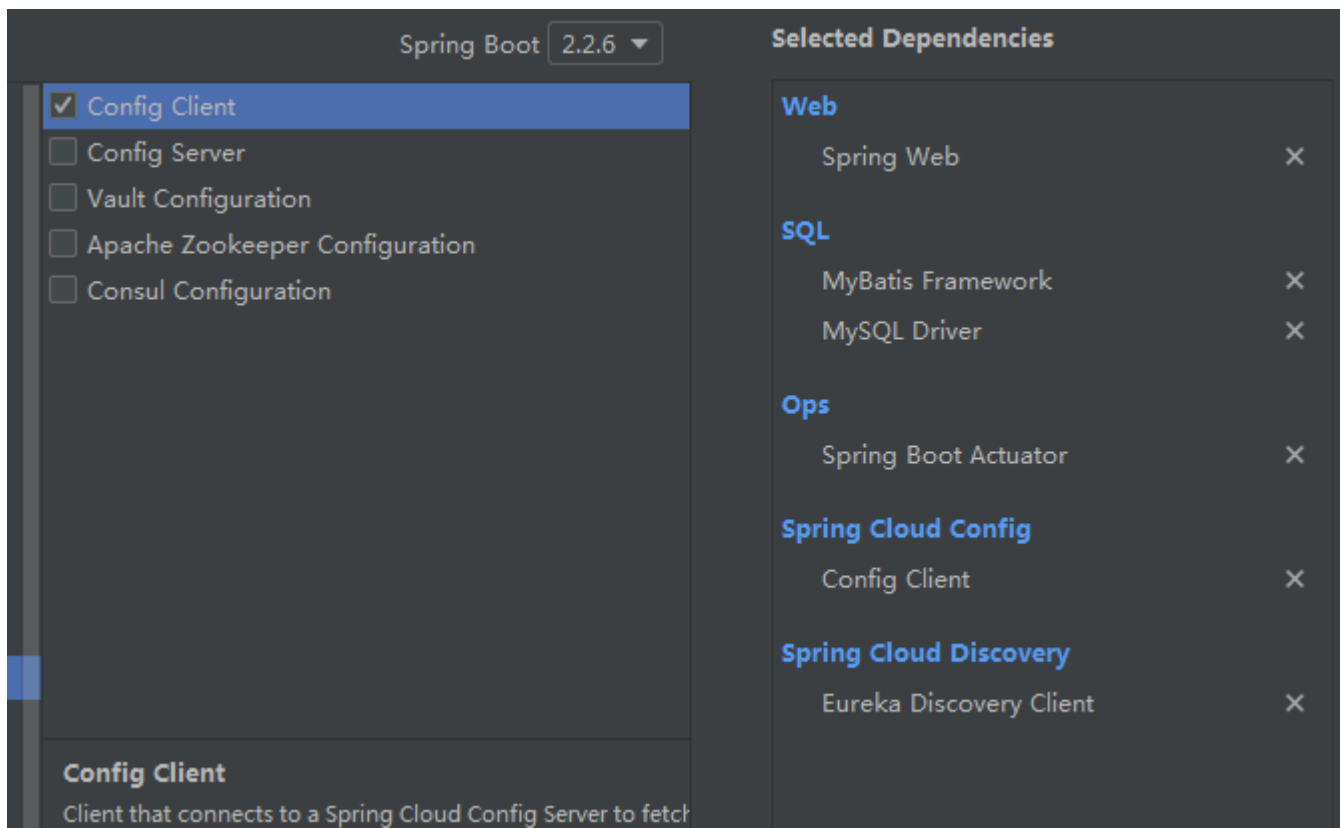


编写Config版的用户微服务

- ❖ 新建Module模块provider，端口8001
 - ▶ 新建模块
 - ▶ 编写全局配置文件application.yml
 - ▶ 创建配置文件bootstrap.properties
 - ▶ 修改主启动类
 - ▶ 拷贝其他业务逻辑代码
 - ▶ 测试

编写Config版的用户微服务

- ❖ 新建Module模块provider，端口8001
 - ▶ 根据第三章eureka工程的provider模块改写，另加入config client依赖



编写Config版的user微服务

❖ 编写全局配置文件application.yml

```
spring:
```

```
  application:
```

```
    name: provider
```

编写Config版的user微服务

❖ 创建配置文件bootstrap.properties

#需要从github上读取的资源名称，注意没有后缀名
spring.cloud.config.name=mscloud-config-user-client

#配置服务中心地址
spring.cloud.config.uri=http://config.com:7755/

#配置文件所在分支
spring.cloud.config.label=master
#配置文件所指环境
spring.cloud.config.profile=dev

编写Config版的用户微服务

❖ 编写主启动类

@SpringBootApplication

@EnableEurekaClient

@MapperScan("edu.neu.springcloud.repository")

@EnableDiscoveryClient

编写Config版的user微服务

- ❖ 拷贝其他业务逻辑代码
 - ▶ 参考源码

编写Config版的user微服务

❖ 测试

- ▶ 先启动config, 端口7755服务
- ▶ 启动config_test, 7001微服务 (确定 `spring.cloud.config.profile=dev`)
- ▶ 启动provider, 端口8001微服务
- ▶ 浏览器 `http://eureka.com:7001/`
- ▶ 浏览器 `http://localhost:8001/user/findUserId/1`

← → ↻ ① 不安全 | eureka.com:7001

应用 百度一下 淘宝特卖 天猫商城 爱淘宝 京东商城 旅行机票 苏宁易购 京东打折 梯度下降法的推导... 6.5 矩阵的运算及...

System Status

Environment	test	Current time	2020-04-18T10:00:17.000Z
Data center	default	Uptime	00:17
		Lease expiration enabled	false
		Renews threshold	3
		Renews (last min)	1

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THE THRESHOLD AND THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
PROVIDER	n/a (1)	(1)	UP (1) - user-provider

← → ↻ ① localhost:8001/user/findUserId/1

应用 百度一下 淘宝特卖 天猫商城 爱淘宝 京东商城 旅行机票 苏宁易购

```
{"id":1,"loginName":"user111","username":"张三","password":"123456","dbSource":"mybatis"}
```


本章重点总结

- ❖ 了解Spring Cloud Config简介；
- ❖ 理解Spring Cloud Config架构图；
- ❖ 掌握配置文件推送到Github；
- ❖ 掌握Config Server编写；
- ❖ 掌握Config Client编写；
- ❖ 掌握Config版的Eureka服务端编写；
- ❖ 掌握Config版的User微服务编写；

课后作业【必做任务】

- ❖ 1、独立完成课件中的示例

