# Nacos搭建和feign使用

## 一、Nacos前言

下面讲述的过程中，可能会有些代码截图遗漏，或者配置漏讲解了，所以我先贴代码，保证大家能成功运行。

数据库连接信息大家根据自己的情况进行修改。

Nacos服务端有70M这么大，我就不贴了，官网可以下载。

大家一边看稳定，一边实操吧，祝大家学习顺利。

代码：



## 第一步：搭建nacos注册中心的服务端

其实网上很多教程的，我这里简单点吧

下载一个自己满意的nacos版本，我这里使用的是nacos1.3.1

前往 [https://github.com/alibaba/nacos/releases](https://links.jianshu.com/go?to=https%3A%2F%2Fgithub.com%2Falibaba%2Fnacos%2Freleases) 下载

解压后是这样的：



**bin**目录是启动用的，支持window启动或者放在linux下面启动。

Window启动，直接就是双击startup.cmd。

Linux启动，是执行sh startup.sh -m standalone 这个指令

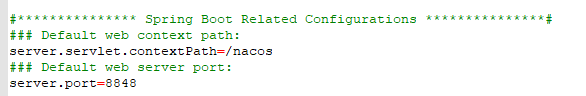
关闭的话，window直接关闭窗口就好了。Linux执行shutdown.sh或者kill掉进程



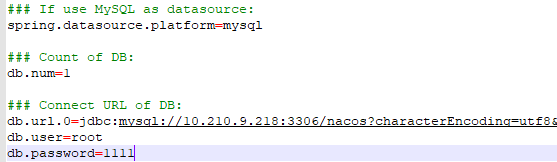
**conf**目录是nacos配置文档所在目录，可以修改nacos的默认路径，默认端口，是否连接mysql数据库。



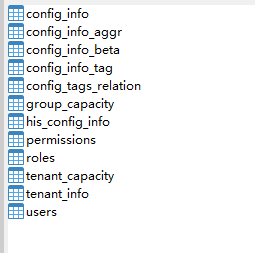
文件application.properties 可以修改默认路径和端口，如图：



Nacos默认使用一个本地存储的方式来存储数据。但如果打算搭建nacos集群，就需要把nacos集群的存储点指向同一个地方，因此nacos支持使用mysql作为自己的存储数据库，这样改改就好：



但连接的数据库，也需要创建对应的表，目录下有文档：**nacos-mysql.sql**，这个就是数据库初始化语句，执行即可。执行效果如图



还有一些其他的配置文档，比如cluster.conf.example是关于搭建nacos集群的，这是后话了。

根目录下还有其它的目录，暂时没什么用，那就先启动nacos吧。

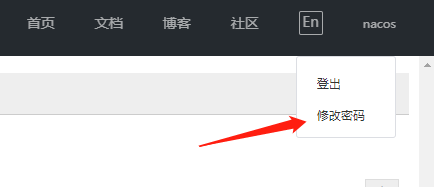
Window环境下，双击nacos\bin\startup.cmd

如图所示：  

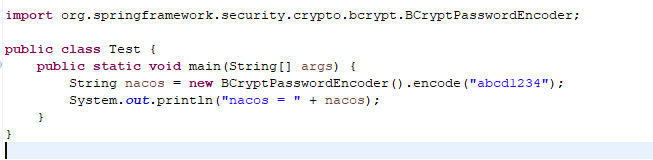

这样就启动成功了，之后在浏览器打开地址：<http://localhost:8848/nacos>

打开nacos的登录页面，一般默认的账号密码是nacos/nacos

这个密码也可以改的。2种方式。第一种是登录后，在页面上改。



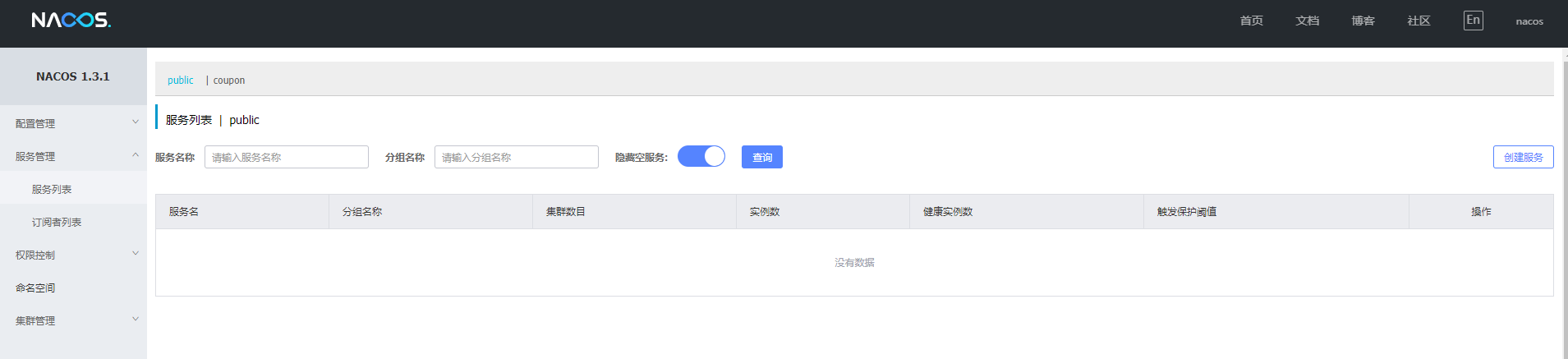
第二种，是直接数据库改，但需要配合对应的加密方式，如图：

当然可以顺便追加用户：

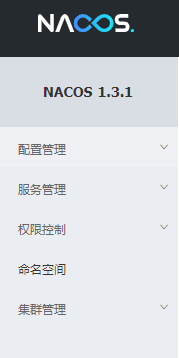
数据库中的users表，修改密码或者追加用户都可以的



登录后，可以看到这样的页面：



看左边的目录：



可以看到有这么多的菜单，我们现在先不管有什么作用，大家浏览一下即可。

说明：nacos除了有服务管理外，还有配置中心的功能，相当于springcloud-config的作用。为了更好的起到对比作用，我后续创建的项目，会把配置中心的特点也体现出来。

这样，nacos注册中心的服务端就启动了，那么需要一些客户端，也就是服务提供者和服务消费者

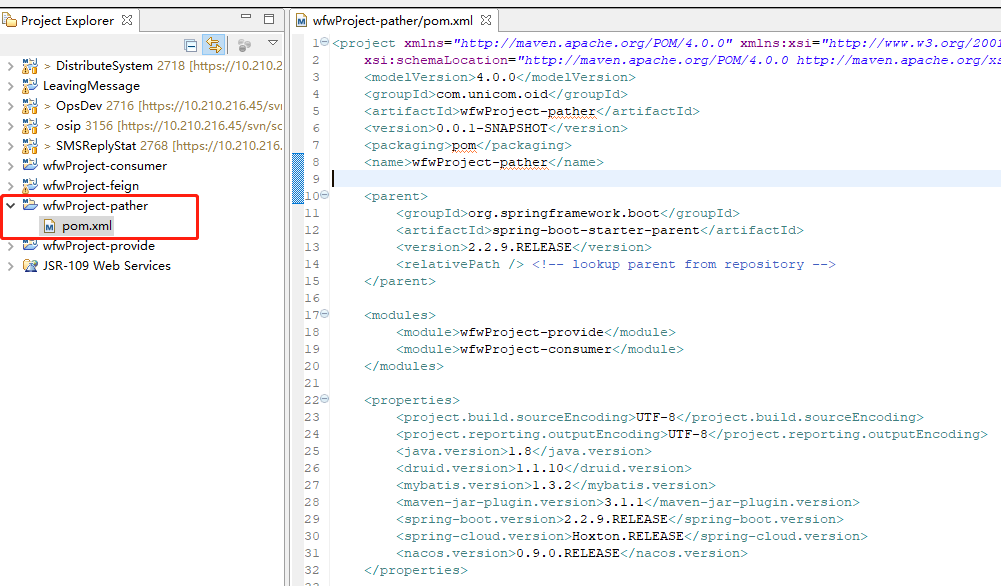
## 第二步：搭建父级项目pather

别问为啥，我打错字母了，应该是father，将错就错吧。。。

先搭建一个父级pom：



父级项目，就一个简单的pom文件

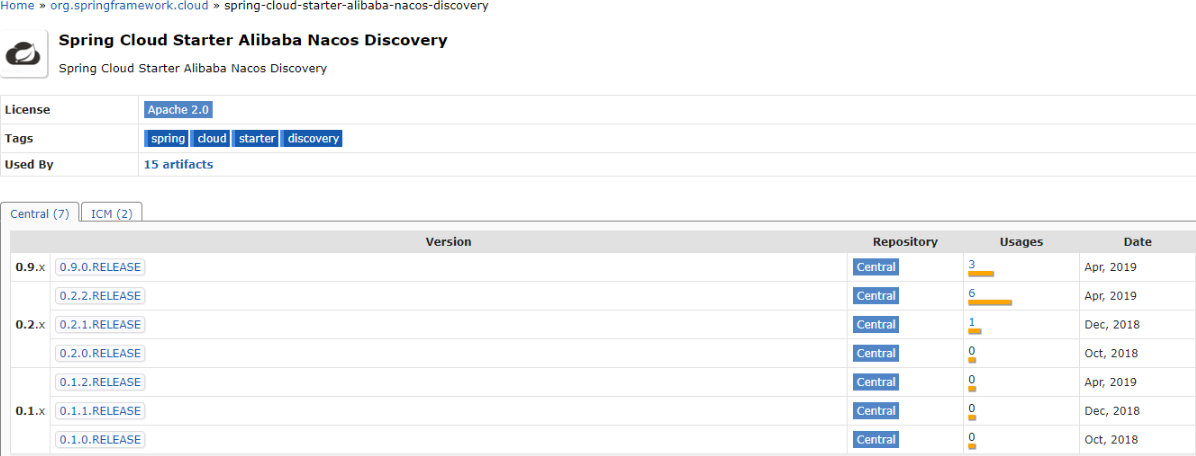


其中注意点：

1. springboot使用2.2.9.RELEASE。版本比较新，与时俱进。
2. springcloud的版本与springboot的版本有对应关系，使用Hoxton.RELEASE，对应关系如下图：



1. nacos的版本，nacos.version 需要使用 0.9.0.RELEASE ，虽然网上说springboot 1.x所对应的是0.1.0.RELEASE, springboot 2.x所对应的是0.2.0.RELEASE，但是实验后发现0.2.0.RELEASE不行。而尝试使用0.9.0.RELEASE却可以了.



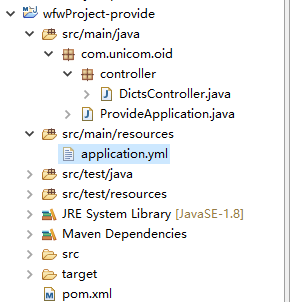
后来也在网上找到这么一个图，感觉关系还是蛮乱的。



1. feign相关的，将会在后续的章节讲到，这里使用的是2.2.2.RELEASE的feign
2. 观察父级的pom，会发现有2个子pom，分别是wfwProject-consumer和wfwProject-provide。下面就讲述如何创建。

## 第三步，搭建服务提供者wfwProject-provide

服务提供者的程序目录架构如下图，很精简：



其中pom文档：





其中，<artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>的版本，是受父级的<artifactId>spring-cloud-alibaba-dependencies</artifactId>控制的，所以版本是0.9.0.RELEASE

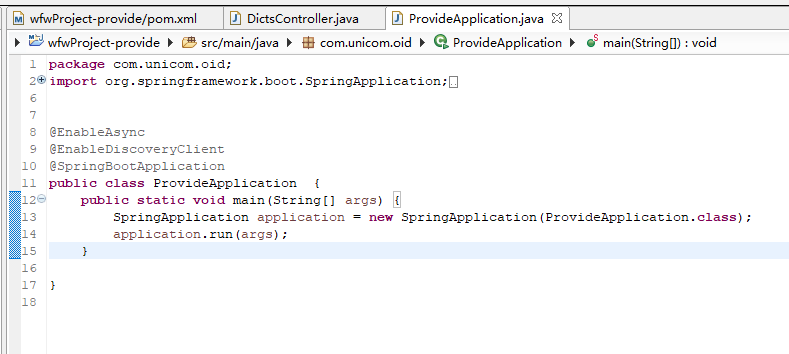
Controller里面的DictsController.java

说明：相当于创建了2个简单的服务，一个是不带参数的接口，一个是带了参数的接口。



ProvideApplication.java

说明：启动类，需要追加一个@EnableDiscoveryClient注解。相当于启动服务与消费者关系



application.yml

说明：nacos除了有服务与消费者管理外，还有配置中心的功能，相当于springcloud-config的作用。为了更好的起到对比作用，这里服务提供者不使用配置中心。

配置内容大概是说明：

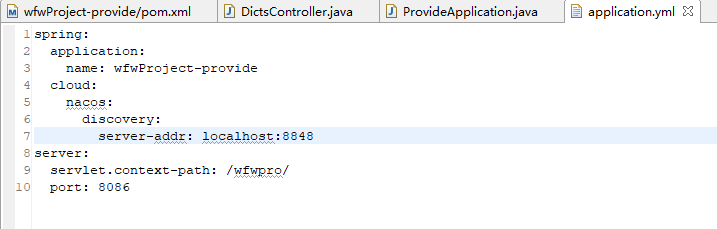
name为wfwProject-provide的系统

访问路径为/wfwpro/。

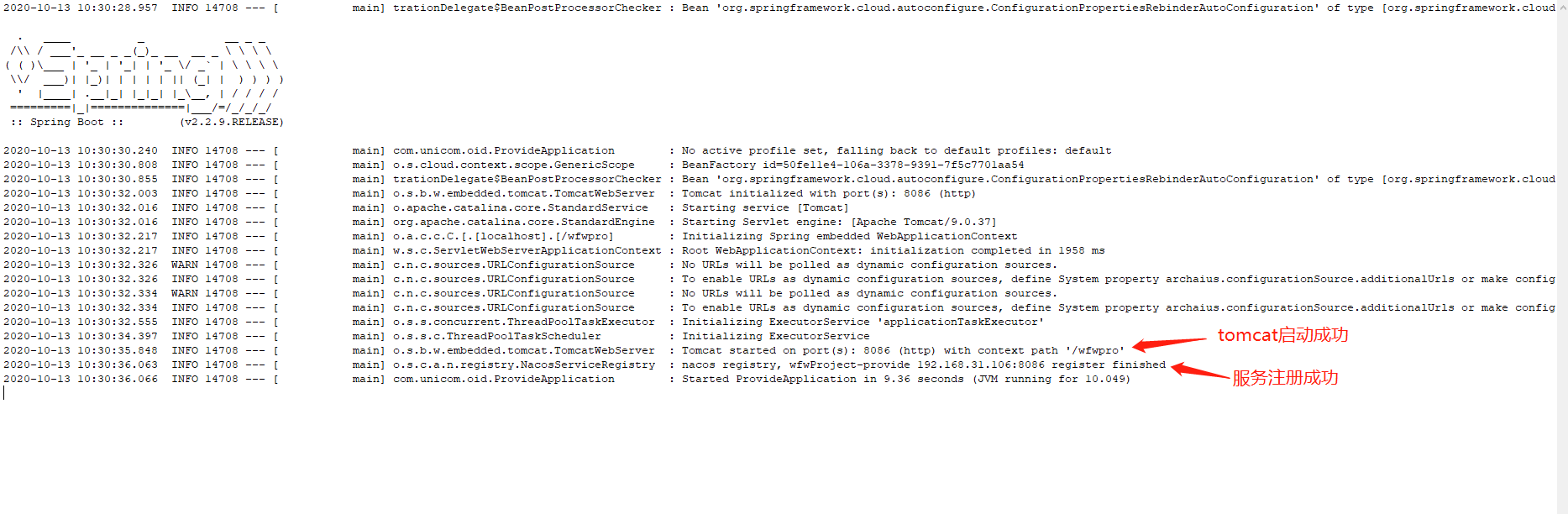
这里注意一下，springboot2.x版本开始，是servlet.context-path，而1.x版本是context-path

端口号使用8086

Nacos的地址使用localhost:8848。这个是第一步启动的nacos服务端的地址，意思是这个服务提供者，将会跟注册中心关联在一起。



启动程序wfwProject-provide



启动日志的最后几行中，  
Tomcat started on port(s): 8086 (http) with context path '/wfwpro'  
这一行说明的是程序启动成功，端口号是8086，目录是/wfwpro

nacos registry, wfwProject-provide 192.168.31.106:8086 register finished

这一行日志说明的是程序已经注册在注册中心，可以提供给其他消费者消费了。

这个时候打开网页的nacos注册中心，可以看到：

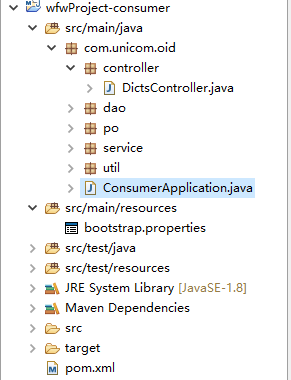


可以看到已经显示了名为：wfwProject-provide的服务者。

到这里，服务提供者已经创建成功，下面是创建消费者。

## 第四步，搭建服务消费者wfwProject-consumer

消费者的程序目录架构如下图，稍微多了一些东西，我拆开来讲：



先放pom.xml文档，大家先不用急着打开，后面慢慢讲：



### 一、Nacos配置中心的应用

看wfwProject-consumer的系统目录，你会发现，没有application.yml，这好像不合常理，实际上，nacos提供了配置中心的功能，也就是说，把配置文档，存放在nacos注册中心服务端上面。

使用nacos的配置中心，需要在pom引入

<dependency>

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

<artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>

</dependency>

先在resources目录下，创建一个bootstrap.properties文档，必须是这个名称





如果有这个配置文件，那么springboot启动的时候，将会去

spring.cloud.nacos.config.server-addr=localhost:8848 这个地址下的nacos注册中心，寻找对应的配置文件。配置文件是在这里编写的：



点击右上角的加号，可以新建配置，：



其中，**Data ID**对应bootstrap.properties文件中的

spring.application.name**+-+**spring.profiles.active**+**spring.cloud.nacos.config.file-extension

根据我当前的配置文档，所以**Data ID**为wfwProject-consumer-dev.yaml

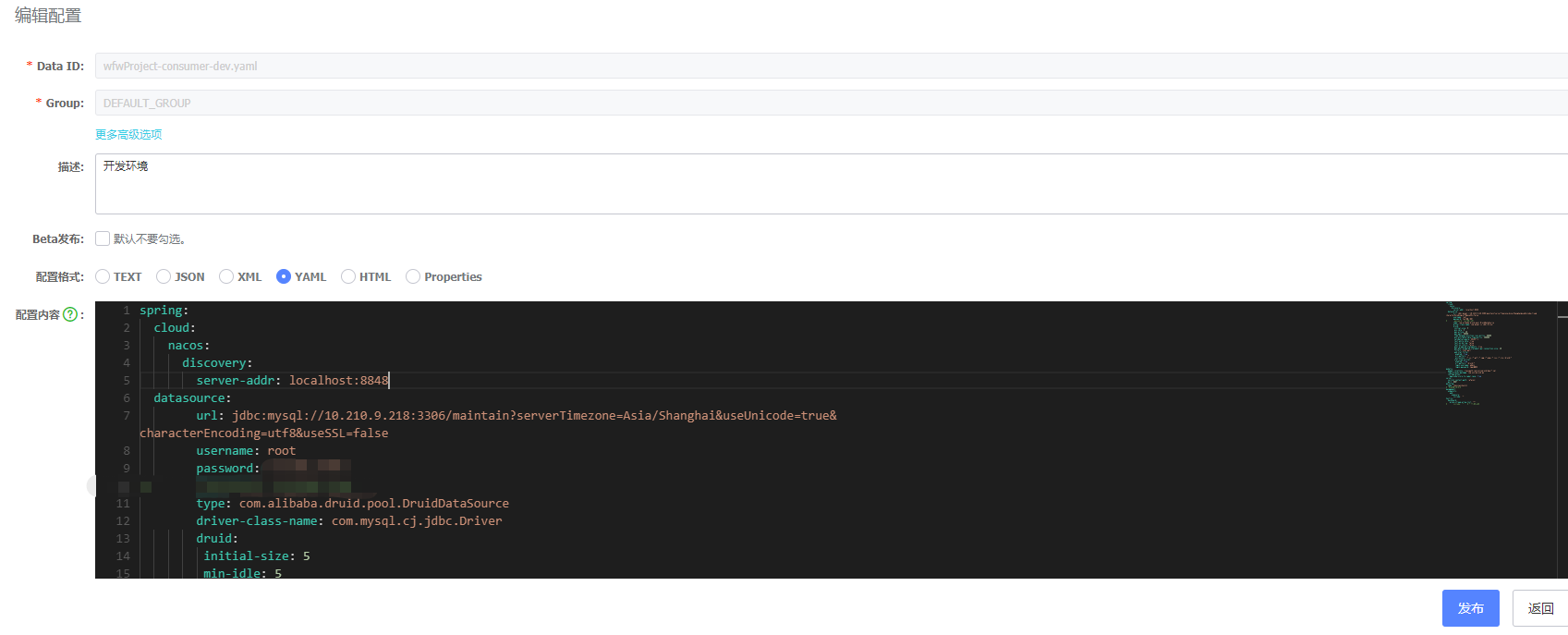
其中spring.application.name为配置文件的主名称。

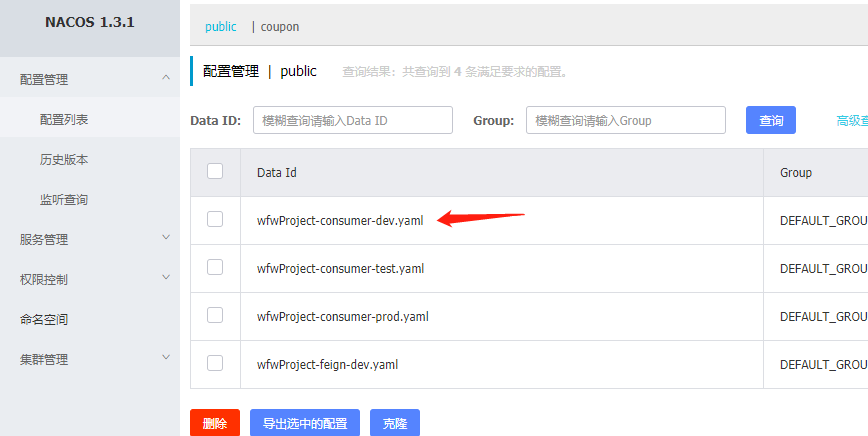
spring.profiles.active为追加的中间名，可以用于区别开发，测试，生产环境的配置文档。如果不填写，则默认不追加这个中间名。

spring.cloud.nacos.config.file-extension为整个配置文件的后缀名，nacos支持很多种，但如果不指定的话，就默认使用**Properties**的样式。我这里选择了yaml。

那么springboot启动的时候，就会找nacos注册中心上的一个名为wfwProject-consumer-dev.yaml的配置文档。

而这个配置文档，需要把平时的application.yml的内容，写上去，最终的效果如下图：





顺便补充一下，springboot 2.x使用mysql的驱动，需要写这个：  
driver-class-name: com.mysql.cj.jdbc.Driver

wfwProject-consumer-dev.yaml配置文件中，

有写nacos的地址server-addr: localhost:8848

还有程序的启动配置：

server:

servlet.context-path: /wfwcon/

port: 8087

总结一下，就是springboot启动的时候，读取bootstrap.properties文档，通过这个文档，读取nacos注册中心服务端上的wfwProject-consumer-dev.yaml文件作为springboot系统的启动配置文件。这样的好处就是，如果在集群环境下，多个系统只需要管理一个配置文件。另外还有修改配置可以自动生效到各个集群应用（有局限的刷新，后面会说到）

### 二、启动消费者程序

先看wfwProject-consumer的启动类，这里使用了多种注解：  
@EnableHystrix 和 @EnableCircuitBreaker 这2个注解为使用了熔断机制，后面章节解析。

@EnableFeignClients 这个注解是关于feign的，feign打算后面再讲，所以把注解屏蔽掉。

这里也对RestTemplate进行注册，后续使用restTemplate来作为服务的调用类。

而上面也追加了@LoadBalanced，这个注解是启动了负载均衡的效果。实际上nacos是包含了ribbon这个负载均衡器。当自己的服务有多个时，将会使用ribbon进行负载均衡的去调用服务者。



下面我们来启动消费者的程序，看日志，启动成功。



这里的日志，对比服务者启动，多出了：

Loading nacos data, dataId: 'wfwProject-consumer-dev.yaml', group: 'DEFAULT\_GROUP'

说明是加载了nacos的配置中心，group 为DEFAULT\_GROUP ，dataId为wfwProject-consumer-dev.yaml的配置。

启动成功后，再去看nacos的注册中心的服务列表，发现已经有2个程序注册在上面了。



这里大家还会发现，这里没有使用@EnableDiscoveryClient注解，但实际上也会把自己注册到nacos，经查询得知，是因为springcloud版本问题，Edgeware.RELEASE以上的版本，已经是默认添加这个注解了。

### 三、测试服务消费者调用服务提供者

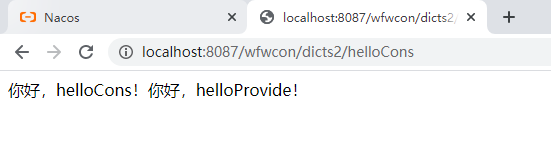
查看wfwProject-consumer的controller下的DictsController.java

这里尝试去调用服务提供者提供的接口，服务的地址的格式为：  
<http://服务名/系统名/接口名/>接口方法



那么我们尝试调用一下:http://localhost:8087/wfwcon/dicts2/helloCons

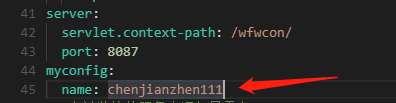
结果是：



这里已经成功的使用消费者去调用服务提供者。

### 四、配置中心读取配置信息

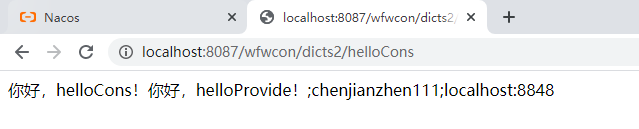
我们在wfwProject-consumer-dev.yaml 追加这个配置，之后重新点击发布



如图修改一下DictsController.java，重启程序：



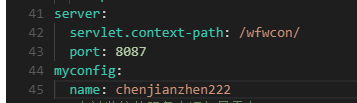
重新调用<http://localhost:8087/wfwcon/dicts2/helloCons>



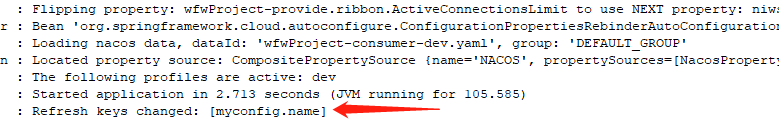
可以发现，可以成功读取配置信息中的myconfig.name和spring.cloud.nacos.discovery.server-addr

这两个配置。

之后我们再去nacos页面上修改一下配置，并点击发布：



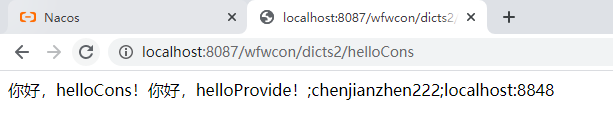
在不重启wfwProject-consumer的前提，如果看程序的控制台输出，可以发现



刷新了一个key为myconfig.name的值。

不重启wfwProject-consumer，我们再调用

<http://localhost:8087/wfwcon/dicts2/helloCons>



这里说明了，哪怕我不重启程序，只要我改了配置中心，程序也会自动刷新，获取最新的配置。

**但我试验过，部分配置是无法刷新的，如数据库连接信息，程序的端口号，程序的路径，nacos的配置地址等等**。

因此动态刷新也并不是万能的，这个大家可以自己尝试一下。

## 二、Nacos补充说明

以上的代码，可能有疏漏，请大家谅解。

Nacos是支持集群的，这里我就不搭建了，下面的参考文献中也有说到。（实际上是我搭建还是有点小问题，我就不误导大家了，我琢磨一下）

参考文献：

<https://www.jianshu.com/p/d38e6d5ebfcb>

<https://blog.csdn.net/zjcjava/article/details/88316190>

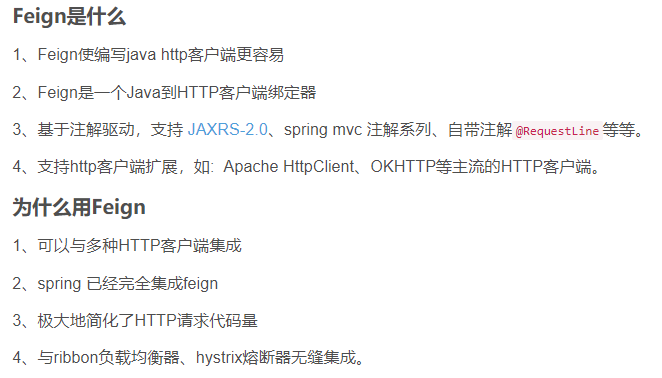
<https://segmentfault.com/q/1010000021044375>

<https://www.cnblogs.com/yelao/p/10741948.html>

<https://blog.csdn.net/m0_46159525/article/details/107386988> 这个是关于集群的文章

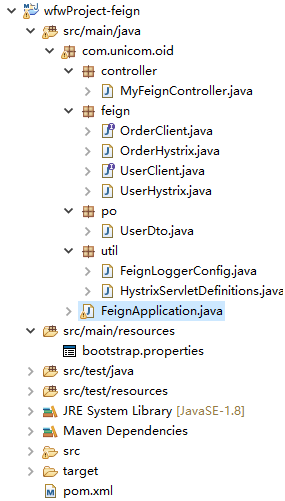
## 三、Feign前言

Feign是对服务之间的调用的一种优化，直接贴图：



## 第一步，搭建服务消费者wfwProject-feign

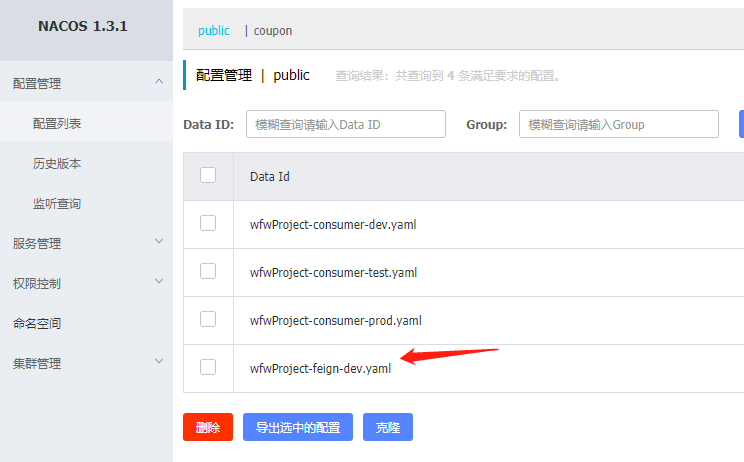
feign的程序目录架构如下图，比上面的系统又多了一些东西，我继续拆开来讲：



老套路，使用nacos的配置中心：



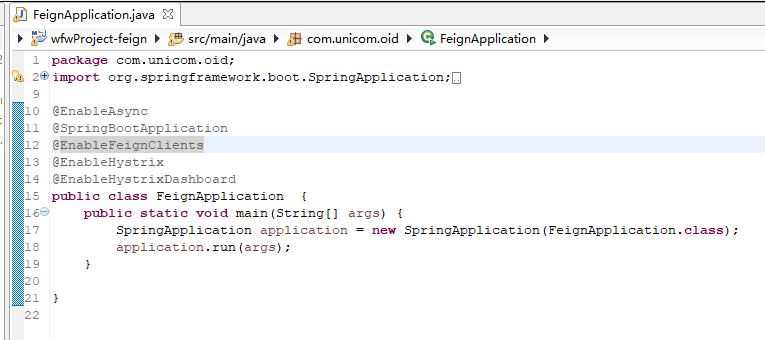
所以读取的是nacos注册中心的wfwProject-feign-dev.yaml 配置文档。



其中配置文件的关键信息有：



启动类：

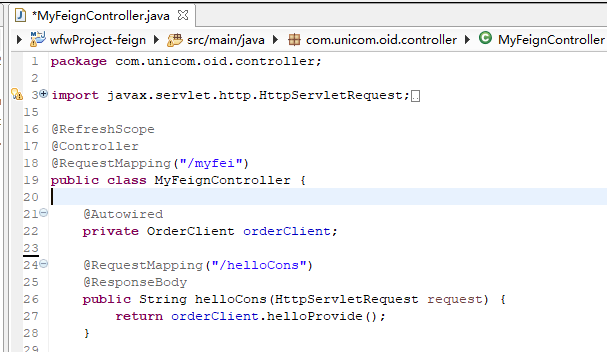


启动类里面，需要添加@EnableFeignClients这个注解，才可以启动feign。

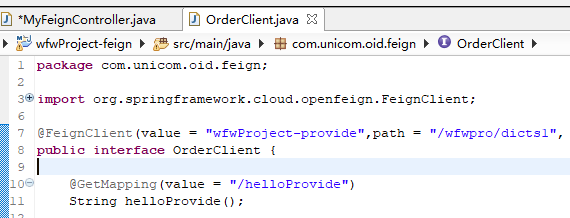
@EnableHystrix和@EnableHystrixDashboard都是启动熔断机制和熔断监控。

## 第三步，使用feign调用服务者提供的接口

打开MyFeignController.java



打开OrderClient.java：



打开地址：

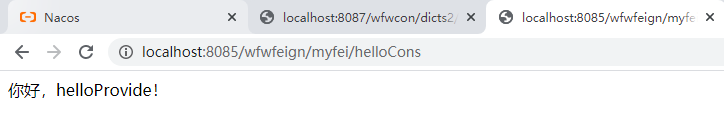
<http://localhost:8085/wfwfeign/myfei/helloCons>

调用顺序为：  
1、controller 的helloCons接口，调用orderClient.helloProvide()

2、OrderClient.java这个类，实际上追加了@FeignClient注解，参数value为服务名，path为接口名。

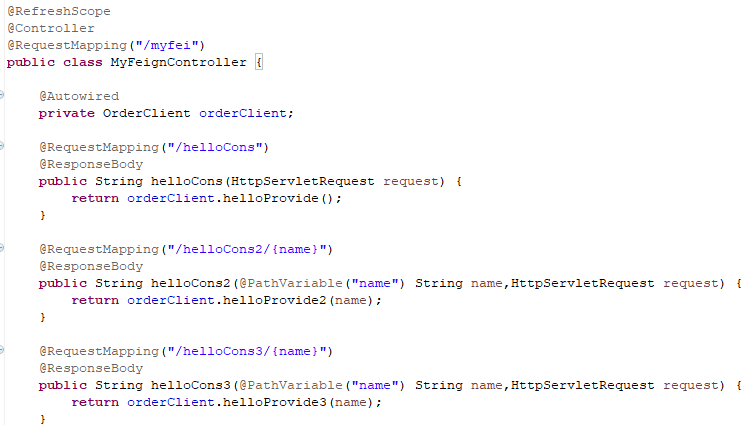
3、因此转换调用服务http://wfwProject-provide/wfwpro/dicts1/helloProvide

调用结果：

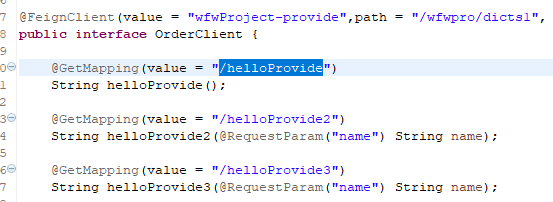


调用成功了，但我们还需要更多的实验，追加接口：

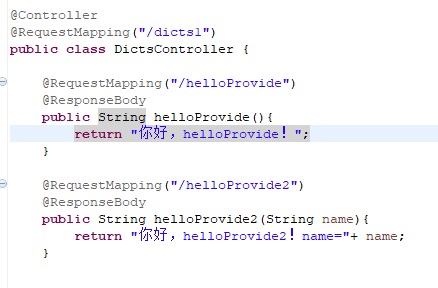
MyFeignController.java



OrderClient.java：

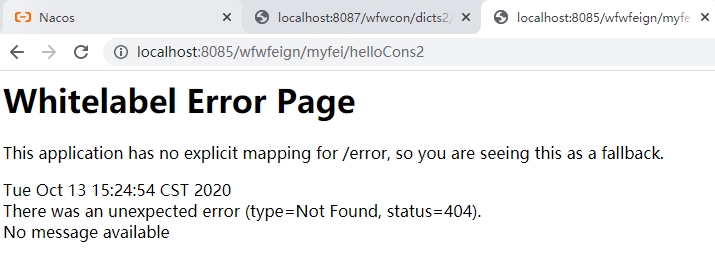


消费提供者的controller：

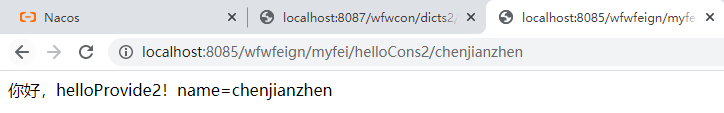


**先尝试调用第二个接口：**

<http://localhost:8085/wfwfeign/myfei/helloCons2>



发现报错，也就是说必须需要带上参数，再来：

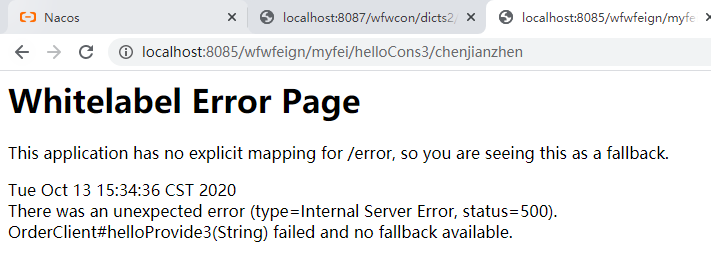


成功调用，参数传递成功。

**尝试调用第三个接口：**

<http://localhost:8085/wfwfeign/myfei/helloCons3/chenjianzhen>

由于服务提供者并没有<http://wfwProject-provide/wfwpro/dicts1/helloProvide3这个接口，那么程序肯定会报404>错误的：

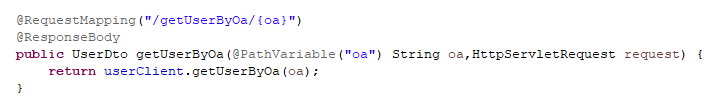


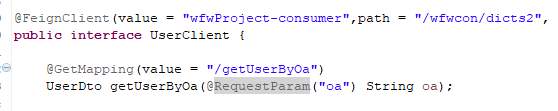


如何防止这种不友好的报错呢，后面的章节会说到熔断机制，就是报错后，会有友好的提示。

接着，我们打算调用后，返回的是一个json对象值，而不是一个字符串。

追加第四个接口：

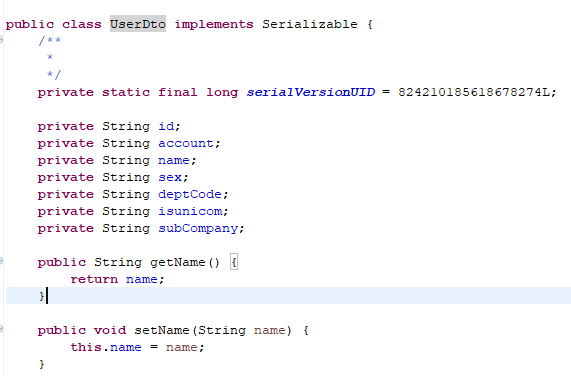




接口使用的服务提供者为wfwProject-consumer，编写对应的controller的接口方法



双方都创建一个UserDto.java，并序列化，添加get set方法



**尝试调用第四个接口：**

<http://localhost:8085/wfwfeign/myfei/getUserByOa/chenjianzhen>



发现可以返回整个对象。

Feign其实还支持很多种请求协议，请求体，返回的格式，这里就不一一说明，后面的参考文献会有提供给大家参考的。

## 第四步，启用熔断机制Hystrix和监控Dashboard

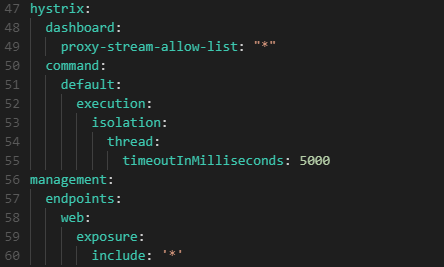
**其实Hystrix和HystrixDashboard是两个东西，但我放在一起讲吧。**

必须满足以下的条件，才可以启用：

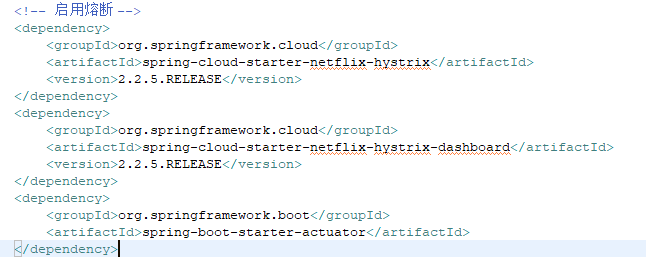
1、启动类里面，添加@EnableHystrix和@EnableHystrixDashboard注解。

2、bootstrap.properties文件里面，添加多一行：feign.hystrix.enabled=true

3、配置文档wfwProject-feign-dev.yaml： （这个配置是用于熔断监控的）



4、pom文件引入对应的依赖：（第一个是启用hystrix，后面两个是熔断监控的）



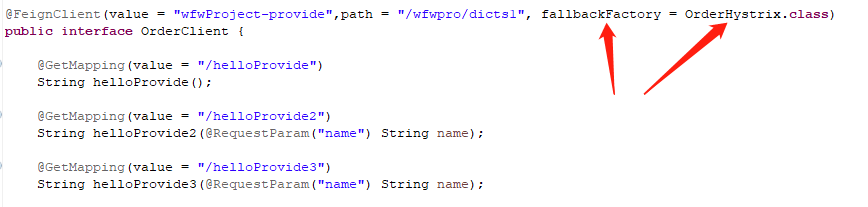
5、添加配置HystrixServletDefinitions.java （这个是用于熔断监控的）



好了，做了以上的准备后，我们回想一下刚才调用**第三个接口**，因为消费者那边没有提供，就会出现这个报404错误。又或者如果服务者提供的接口报错导致的500错误，都可能会引起消费者的程序异常，那么如何解决，可以追加熔断机制。

网上有好几种熔断机制的玩法，但我经过对比，我个人比较喜欢这种：

在@FeignClient注解中绑定错误处理类：



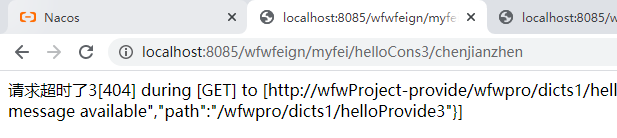
这个OrderHystrix.java继承FallbackFactory，并可以这样写：

这样每个方法调用后，如果发生异常，就可以在这里捕抓并处理。

**再尝试调用第三个接口：**

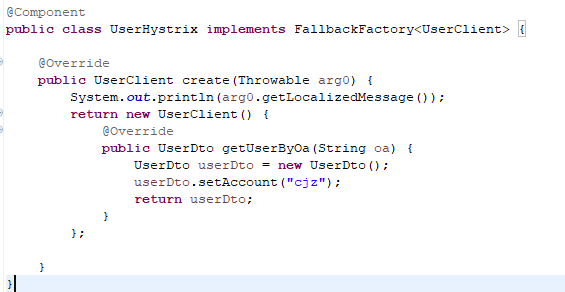
<http://localhost:8085/wfwfeign/myfei/helloCons3/chenjianzhen>

返回如下信息：  
请求超时了3[404] during [GET] to [http://wfwProject-provide/wfwpro/dicts1/helloProvide3?name=chenjianzhen] [OrderClient#helloProvide3(String)]: [{"timestamp":"2020-10-13T07:58:24.598+0000","status":404,"error":"Not Found","message":"No message available","path":"/wfwpro/dicts1/helloProvide3"}]



那我们就可以根据不同的报错，做出不同的处理。

又比如getUserByOa可以给一个初始值



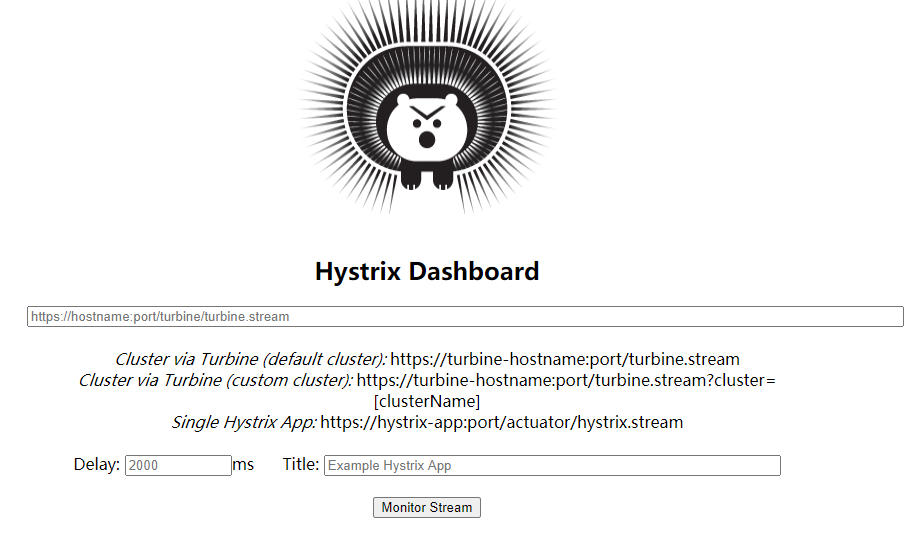
## 第五步，查看监控Dashboard

1. 尝试打开地址：<http://localhost:8085/wfwfeign/hystrix.stream>

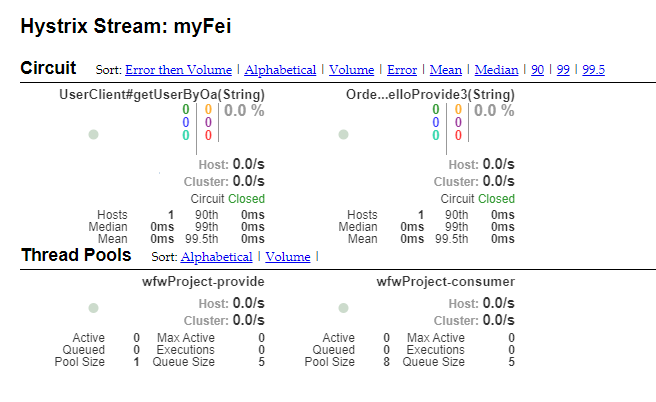
你会发现有很多刷出来的东西。如果出现了，说明你成功启动监控面板Dashboard。可以关闭掉这个页面了。

1. 再尝试打开地址：<http://localhost:8085/wfwfeign/hystrix>

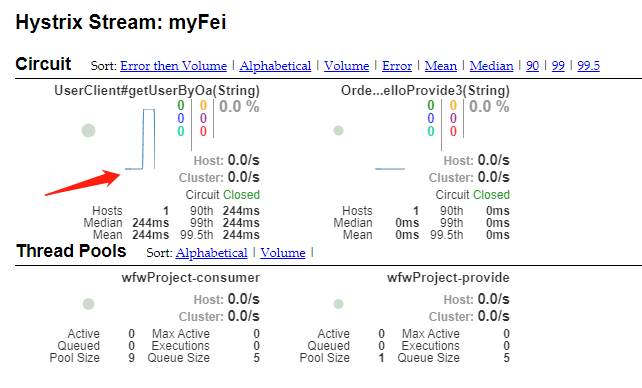
这一次，你会看见一个这样的页面：



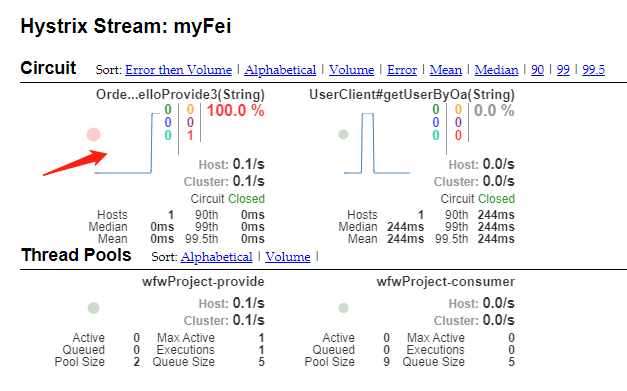
1. 第一个输入框，输入需要监控的服务：http://localhost:8085/wfwCon/hystrix.stream
2. 下面还有2个输入框，第一个是延时多少毫秒监控一次，默认2000毫秒。第二个是监控的标题，我们填写myFei吧，点击最下面的按钮，看到这样的一个页面：



1. 因为没有调用过任何借口，所以这里没有任何信息。现在我们尝试调用一次**第四个接口**。再观察一下：



1. 再尝试调用**第三个接口**，再观察一下：



1. 不同的颜色，分别对应：



到这里为止，监控的作用也有了。

## 四、Feign后言

1、Feign有很多种调用方式，具体查询网上其他参考文献：

<https://www.cnblogs.com/duanxz/p/7054928.html>

2、熔断机制，还有蛮多可以学习的，可以参考网上其他参考文献。

3、熔断监控面板，实际上，监控我也是弄通而已，怎么去看，怎么去用，怎样深层次的应用，还有待学习。

参考文献：

<https://blog.csdn.net/sun_shaoping>