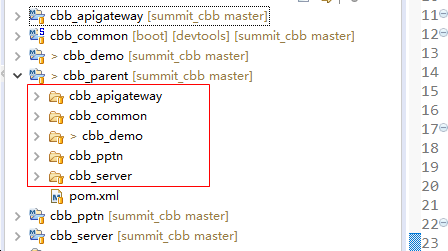
**共享组件工程搭建及请求流程**

**1.首先从华为云上clone 组件模块**

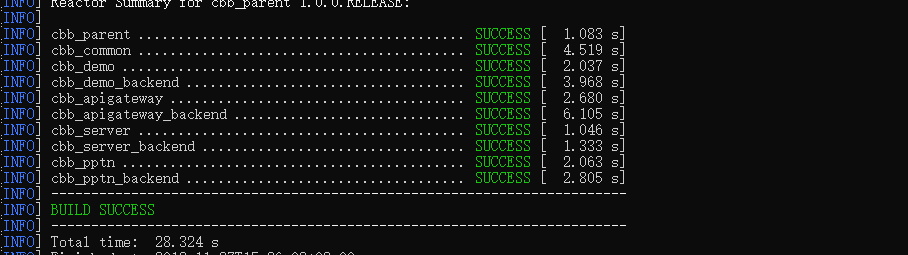
华为云访问地址:<https://devcloud.huaweicloud.com/codehub/project/7ebfb43fd288413495ccae1590babfe3/codehub/433269/home>

华为云git地址：<https://codehub.devcloud.huaweicloud.com/smgxzjpt_yeteng/summit_cbb.git>

组件结构如图所示，引入所有依赖：

进入到cbb\_parent目录,执行:

mvn package -Dmaven.test.skip=true ，对目前所有依赖组件打包



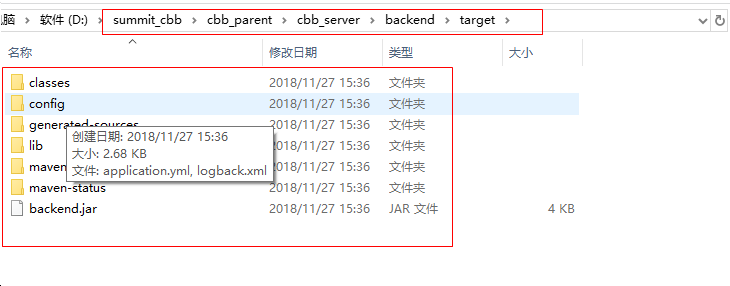
cbb\_register\_center：

对应springcloud eureka-**server** ,服务注册中心，当有一个新的组件**client**产生，

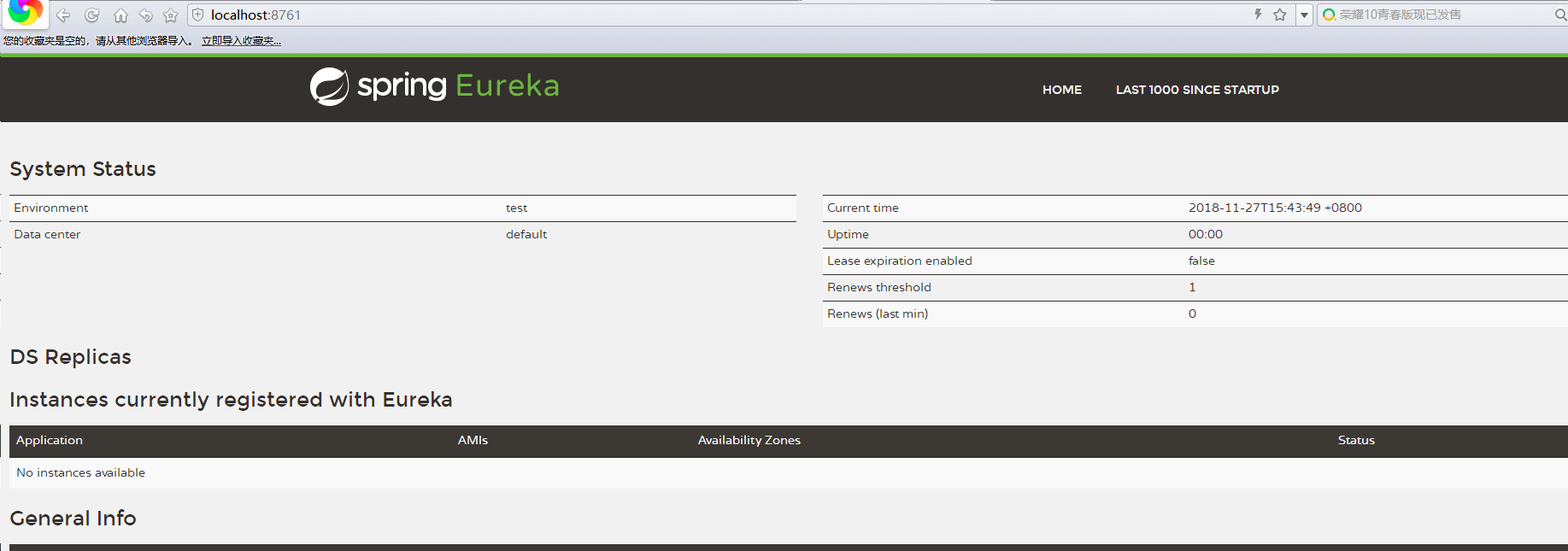
将会自动注册到注册中心上

组件服务发布 必须启动，作为注册中心和服务发现者

打包后的目录格式如下图所示：



进入到目录下 ，执行 java -jar backend.jar 执行启动cbb\_register\_center



启动服务后发现，没有任何组件服务注册到注册中心

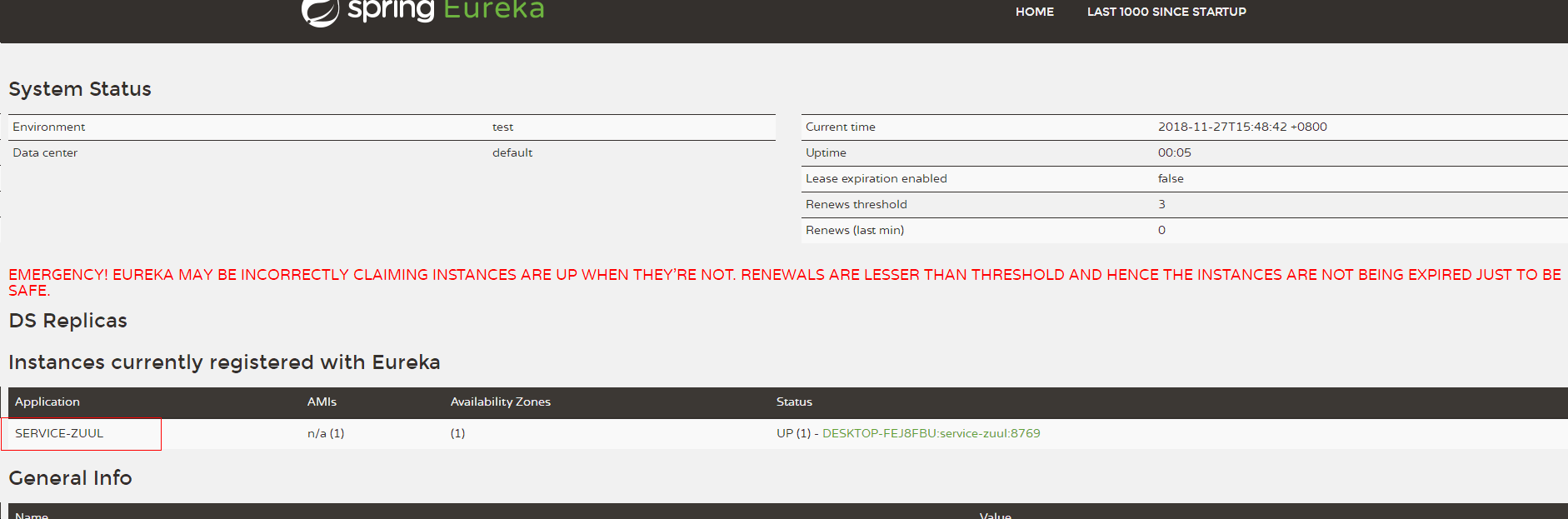
**cbb\_api\_gateway\_es**

组件的权限验证，统一网关组件。

所有的api请求都将通过该组件转发 ，如果需要权限验证，该组件为必须。

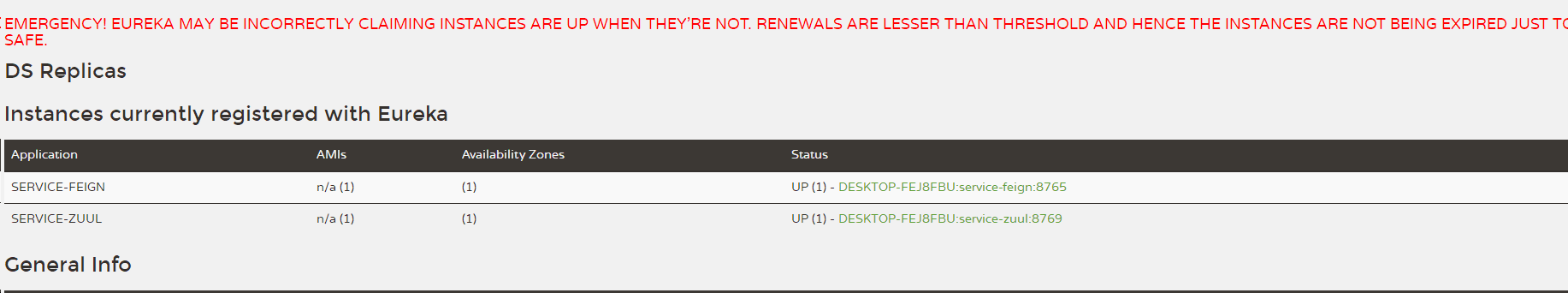
同样的方式启动，进入到目录下 ，执行 java -jar backend.jar 执行启动cbb\_apigateway\_es

此时发现已经有服务注册到注册中心上：

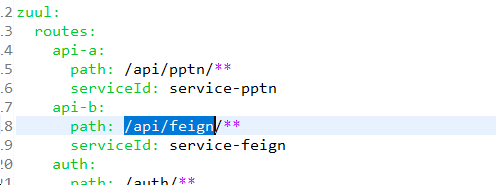


若想验证权限模块及统一网关模块是否有效 ，需要再启动一个组件模块进行验证： 启动cbb\_demo

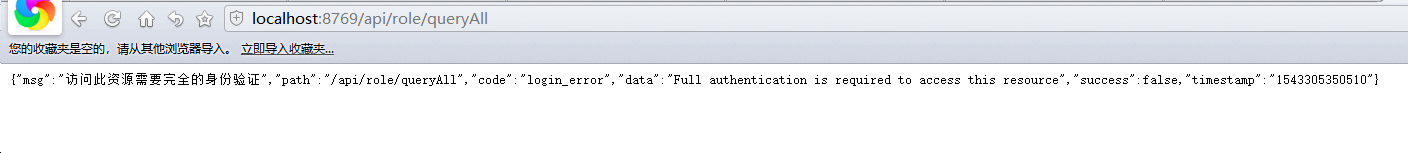
查看服务是否启动和被注册到注册中心



刚才可以看到，apigateway模块的端口号为8769，查看网关配置：



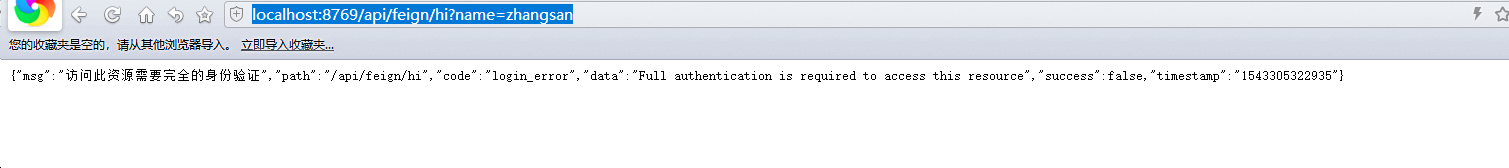
我们的cbb\_demo 访问路径为： path: /api/feign/\*\* 通过浏览器发送请求验证：[http://localhost:8769/api/role/queryAll](http://localhost:8769/api/role/queryAll" \t "_blank)



发现验证失败 没有权限访问，此api为apigateway本身提供的api方法

此时访问 cbb\_demo中的方法 ：

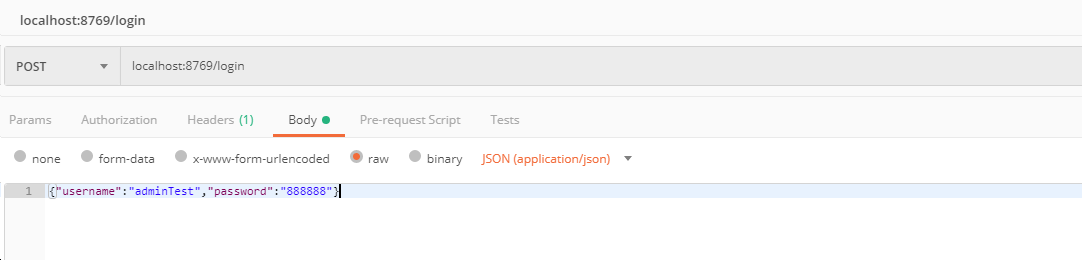
<http://localhost:8769/api/feign/hi?name=zhangsan>



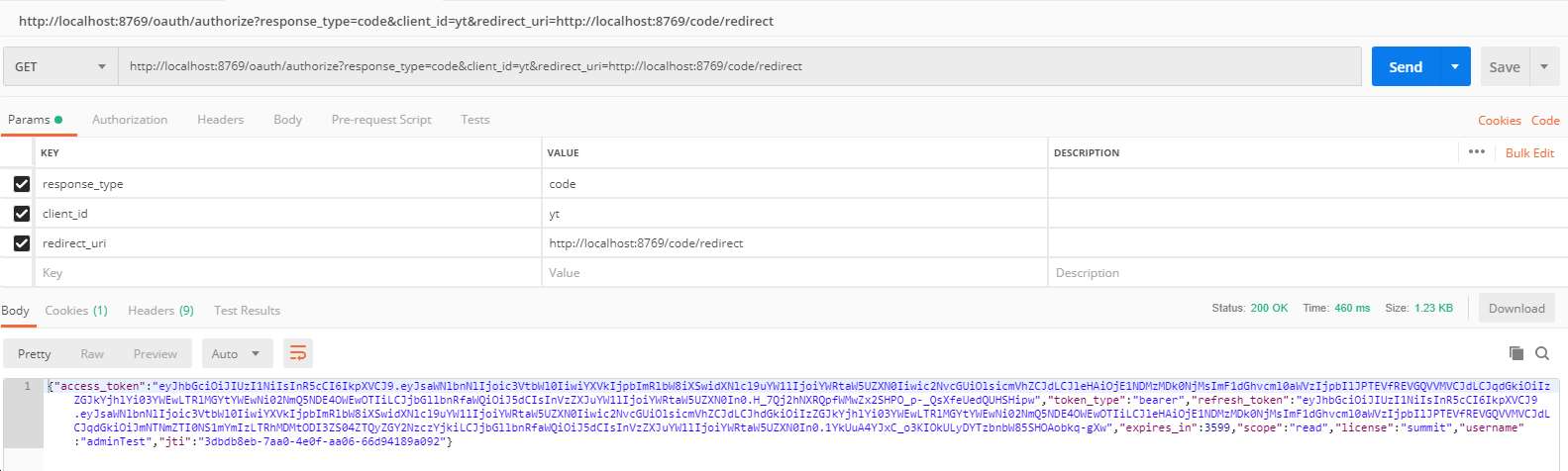
发现效果一样，都无权限访问，而统一网关及权限验证起到作用

开始验证权限 ：

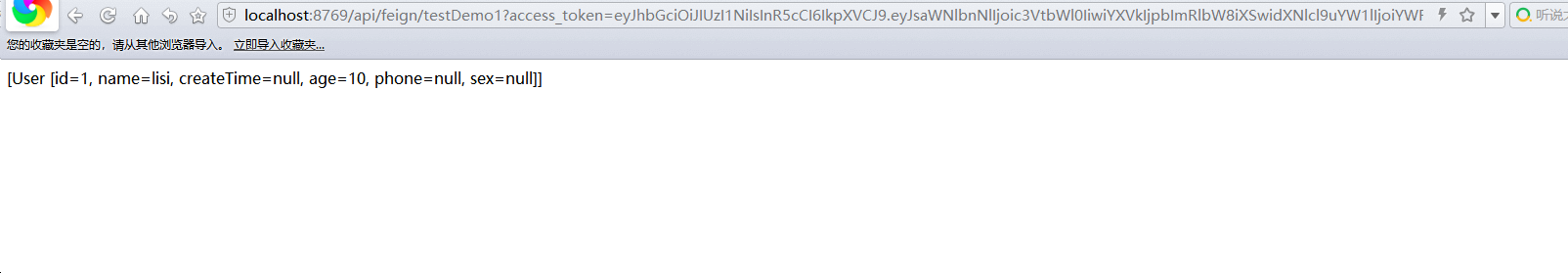
登录认证 ：

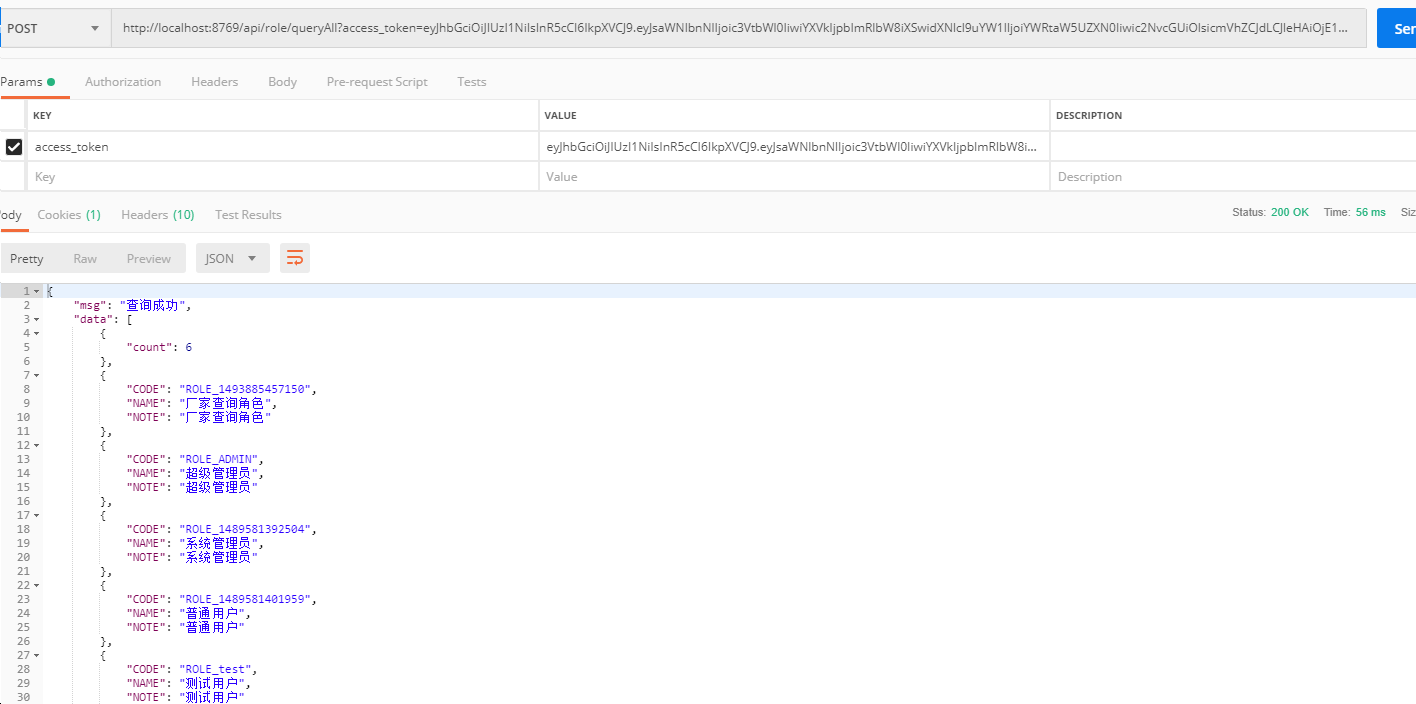


获取token:



此时使用获取后的token值 再次请求





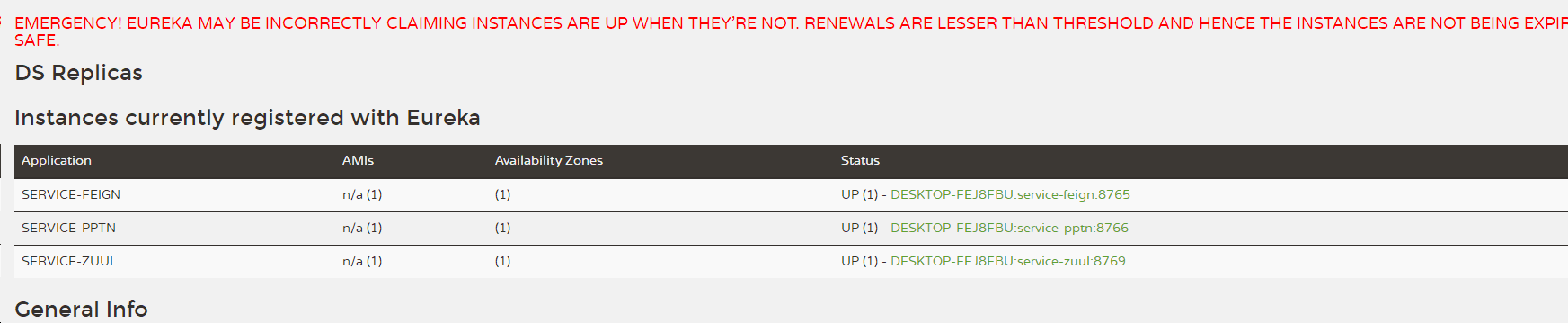
权限验证，统一网关实现。

**组件内功能： 消费服务 ，负载均衡，熔断器： feign**

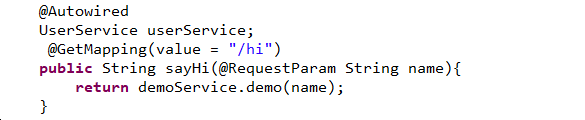
**消费服务，负载均衡：**

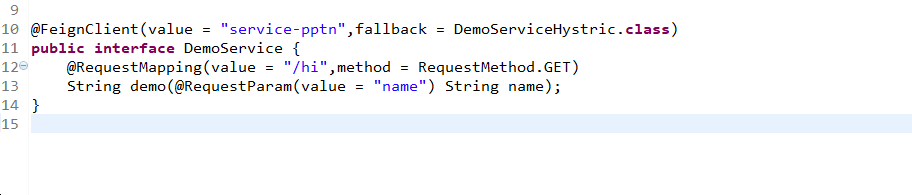
指我们在组件之间通过rest请求相互调用，平时任何一个rest请求都会写一大堆代码去发送请求 ，而springcloud 通过feign可以帮助我们简化流程，实现组件之间的通信及负载均衡。

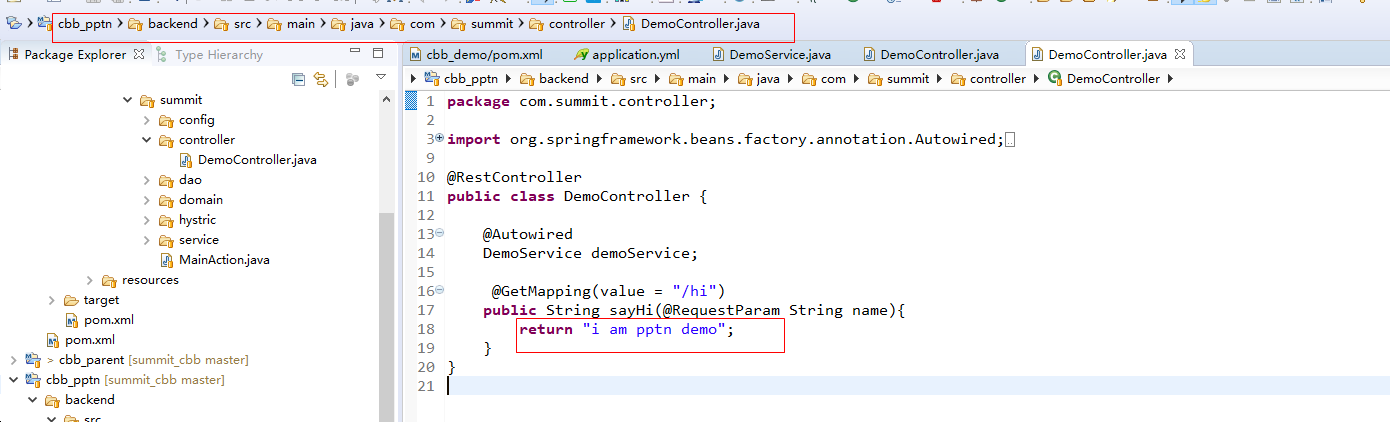
开启两个组件服务 ： cbb\_demo ,cbb\_pptn 若不需要权限验证可先关闭apigateway服务.

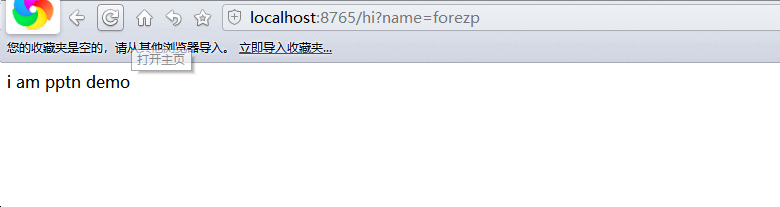


通用代码我们可以知道，通过调用 demo中的hi 接口即将请求发送到pptn的hi接口中









可以直接调用pptn中的 hi 接口。

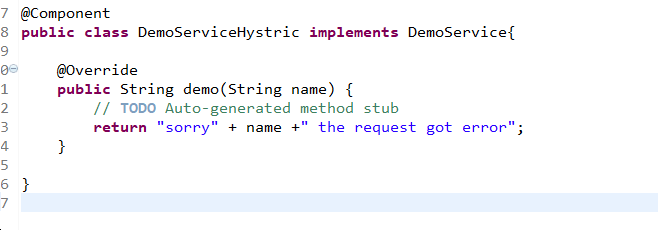
**熔断器：**

当我们访问某个组件中的请求时，若由于某些原因（eg:服务未起），

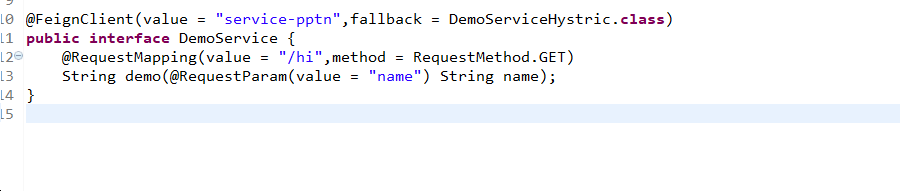
长时间访问不到，若此时执行多次访问，很有可能造成服务雪崩。

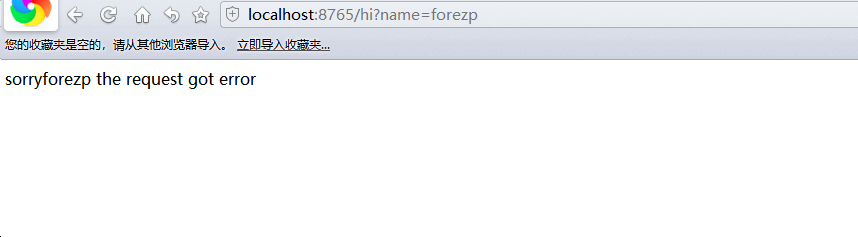
此时,熔断器就起着很重要的作用。

首先自定义熔断器：



service接口上定义执行fallback的熔断器:



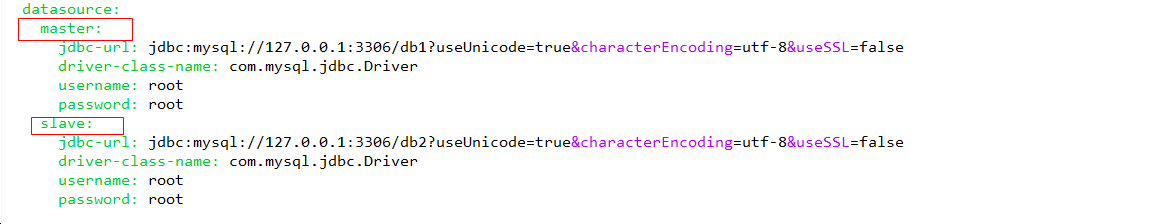
此时，关闭pptn 服务，再次访问，执行service方法 ,实现熔断功能:

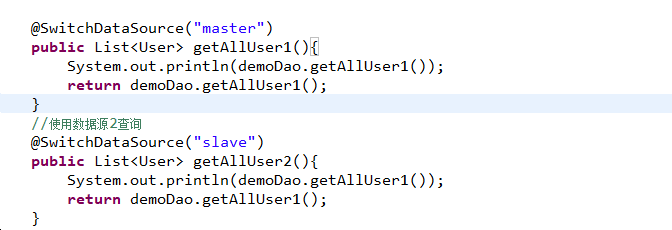
**自定义多数据源**

在我们多个组件相互调用的过程中，可能会分别访问不同数据源的情况。

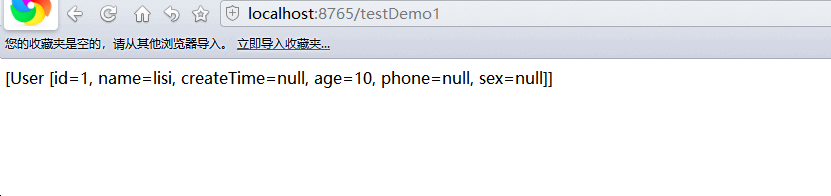
通过**spring** 拦截机制 实现了一套自定义多数据源 ，实现结果如下：

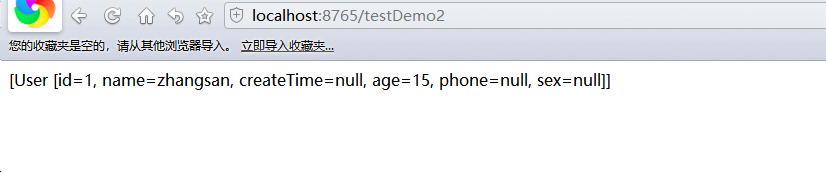
通过配置文件 注入两个dataSource链接 ，通过自定义注解加载连接并实现查询：





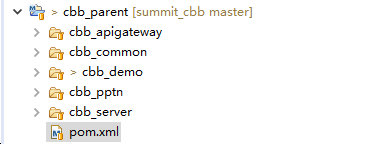
通过调用不同的api ，访问不同的数据库：



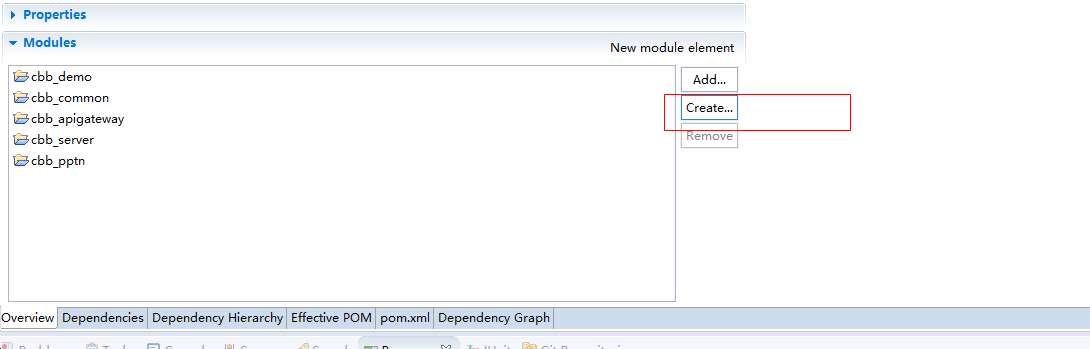


**如何开发一个新的组件？**

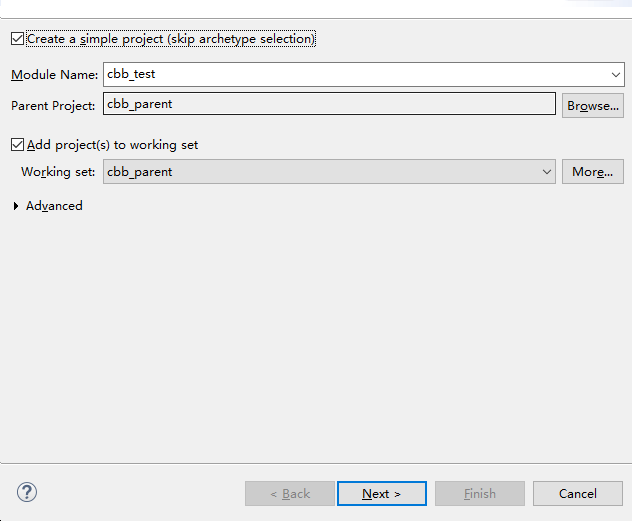
首先我们需要有一个共享组件工程架构:



基于cbb\_parent 新建maven\_module eg:cbb\_test

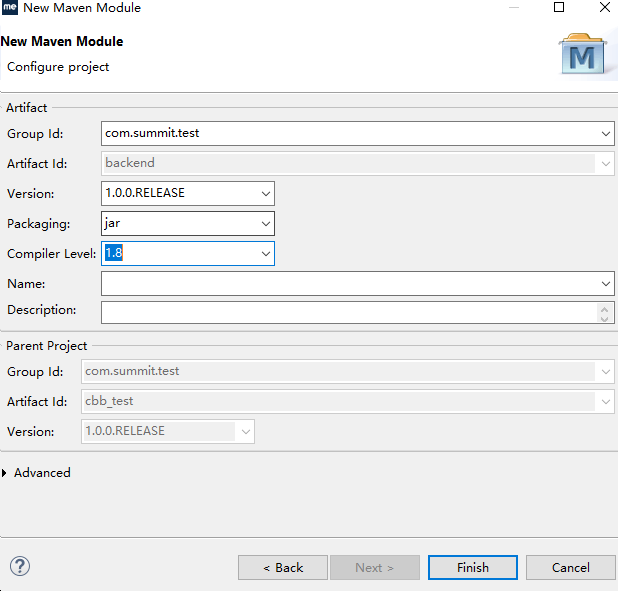


修改 group\_id 及 jdk版本,packaging 选择pom



创建完成后，基于cbb\_test新建maven\_module ：backend(名字固定)

修改 jdk版本,packaging 选择 jar



工程构建完成后 ： 将cbb\_demo中的结构及文件复制copy即可，若要实现自身业务，即在该组件下实现业务代码及引入相应的jar包