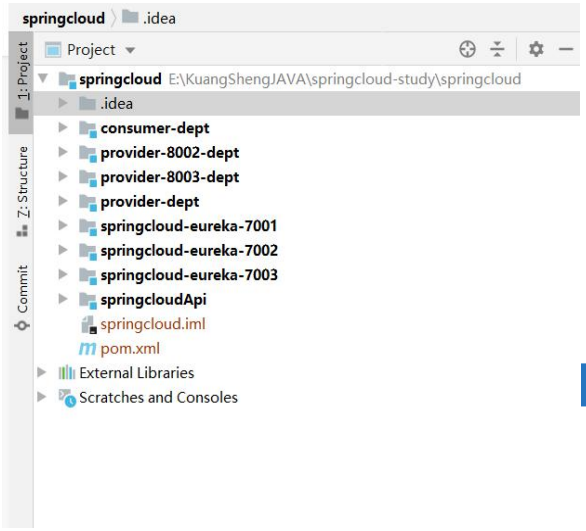
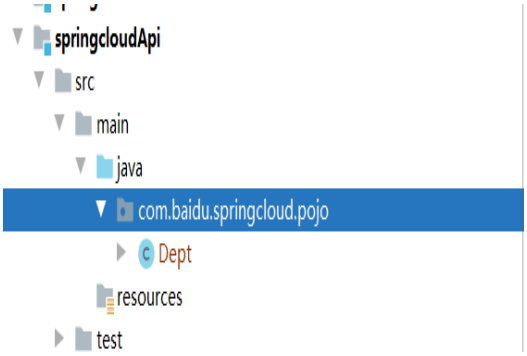
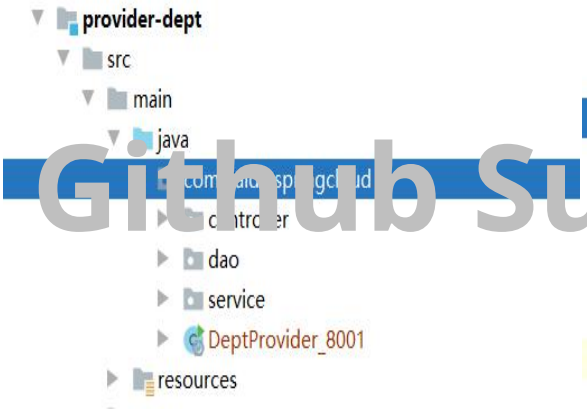
SpringCloud 操作记录

项目结构







Eureka 注册

DeptConsumerController.java

```java
@RestController
public class DeptConsumerController {
    //dubbo 和 zookeeper rpc时候是 reference引用接口
    //消费者不应该有service请求
    //找restful风格 此时可以发现有resttemplate模板 用于请求供我们直接调用就可以
    //把template注册到spring中，没有@bean 手动构造config
    @Autowired
    private RestTemplate restTemplate; //提供多种便捷访问远程http的方法 简单的restful服务模板
    //private static final String REST_URL_PREFIX="http://localhost:8001";
    //此时ribbon应该拿某一个服务来做请求
    private static final String REST_URL_PREFIX="http://SPRINGCLOUD-PROVIDER-DEPT";
    @RequestMapping("/consumer/dept/add")
    public boolean add( Dept dept){
        return  restTemplate.postForObject( url: REST_URL_PREFIX+"/dept/add", dept,Boolean.class);
    }
    @RequestMapping("/consumer/dept/list")
    public List<Dept> getList() { return restTemplate.getForObject( url: REST_URL_PREFIX+"/dept/list", List.class); }

    @RequestMapping("/consumer/dept/get/{id}")
    public Dept get(@PathVariable("id") Long id){
        //此时需要去localhost:8001/dept/get/{id} 拿
        return restTemplate.getForObject( url: REST_URL_PREFIX+"/dept/get/"+id,Dept.class);//返回dept类型
    }
```

Ribbon是用微服务名字访问

SpringCloud 拿取外部接口需要 RestTemplate 才可以

```java
package com.baidu.springcloud.config;

import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class ConfigBean {
    //@Configuration  相当于 applicationContext.xml

    //配置负载均衡
    @LoadBalanced  //Ribbon就行了 通过Irule可以 自定义轮询算法
    //配置bean
    @Bean
    public RestTemplate getRestTemplate() { return new RestTemplate(); }


}
```

DeptController.java

post请求json与obj转换

```java
//提供restful服务
@Autowired
DeptService deptService;
//获取配置信息
@Autowired
DiscoveryClient discoveryClient;
@PostMapping("/dept/add")
public boolean addDept(@RequestBody Dept dept) { return deptService.addDept(dept); }
@GetMapping("/dept/get/{id}")
public D ue P @PathVariable( Long id) { return dept rvic yDe ById( ); }
@Ge apping("/d t/l ")
pub c L t<Dep que Al { tur dep ervice quer ept ; }
//注册进来的微服务获取一些信息
// 个人用不到
// 协同需要
@RequestMapping("/dept/discovery ")
public Object discovery(){
    List<String> services = discoveryClient.getServices();
    System.out.println("discover=>"+services);
    //得到一个具体微服务信息
    List<ServiceInstance> instances = discoveryClient.getInstances( serviceId: "SPRINGCLOUD-PROVIDER-DEPT");//通过具体微服务名字拿到信息
    for (ServiceInstance instance : instances) {
        System.out.println(instance); //服务的端口，主机，等等内容
    }
    return this.discoveryClient;
```



```java
//启动类
@SpringBootApplication
@EnableEurekaClient
public class DeptProvider_8001 {
    public static void main(String[] args) {
        SpringApplication.run(DeptProvider_8001.class, args);
    }
}
```

提供者是客户端

```java
@SpringBootApplication
@EnableEurekaClient
public class DeptConsumer_80 {
    public static void main(String[] args) { SpringApplication.run(DeptConsumer_80.class, args); }
}
```

application.yaml

```yaml
server:
  port: 80
#Eureka
eureka:
  client:
    register-with-eureka: false
    service-url:
      defaultZone: http://127.0.0.1:7001/eureka/
```

三个注册中心的地址（集群负载均衡才配置）

消费者也要从数据中拿数据

springcloud > provider-dept > src > main > resources > application.yml

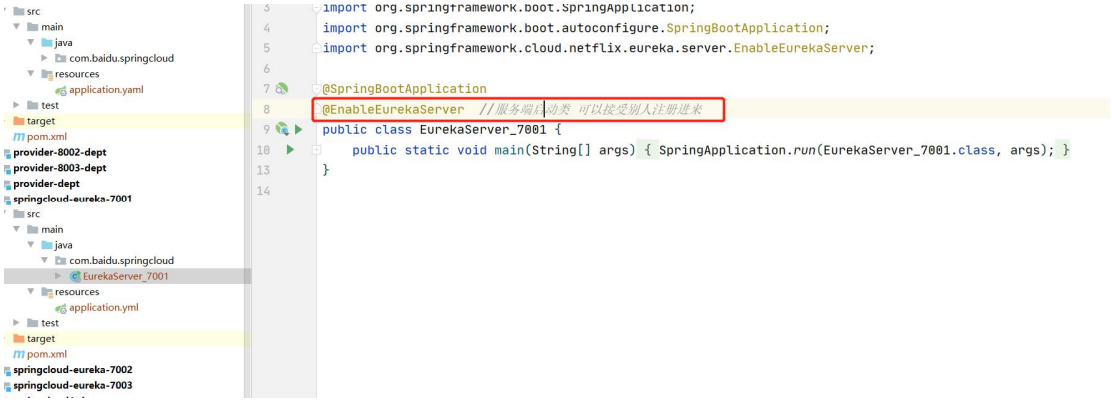
application.yml

```yaml
server:
  port: 8001
mybatis:
  type-aliases-package: com.baidu.springcloud.pojo
  #核心配置文件
  config-location: classpath:mybatis/mybatis-config.xml
  mapper-locations: classpath:mybatis/mapper/*.xml

#spring配置
spring:
  application:
    name: sp...
  datasource:
    ...
    password: root
    driver-class-name: org.gjt.mm.mysql.Driver
    url: jdbc:mysql://192.168.1.100:3306/dbsuki?useUnicode=true&characterEncoding=utf-8
    type: com.alibaba.druid.pool.DruidDataSource
eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
      defaultZone: http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/,http://eureka7001.com:7001/eureka/
```


application.yml

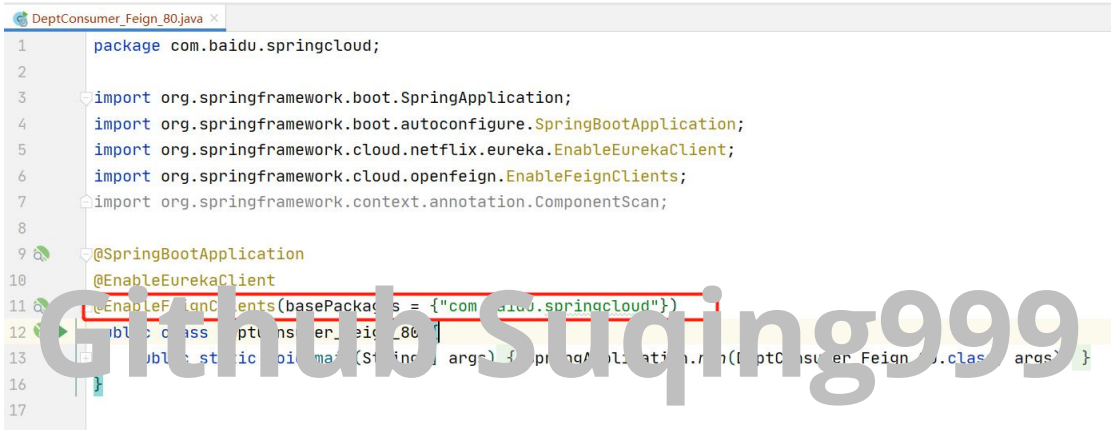
```yaml
server:
  port: 7001

#配置eureka
eureka:
  instance:
    hostname: eureka7001.com
  client:
    registerWithEureka: false
    fetchRegistry: false
    service-url:
      defaultZone: http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
      #单机
      #defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
      #集群（关联其他两台机器）
```

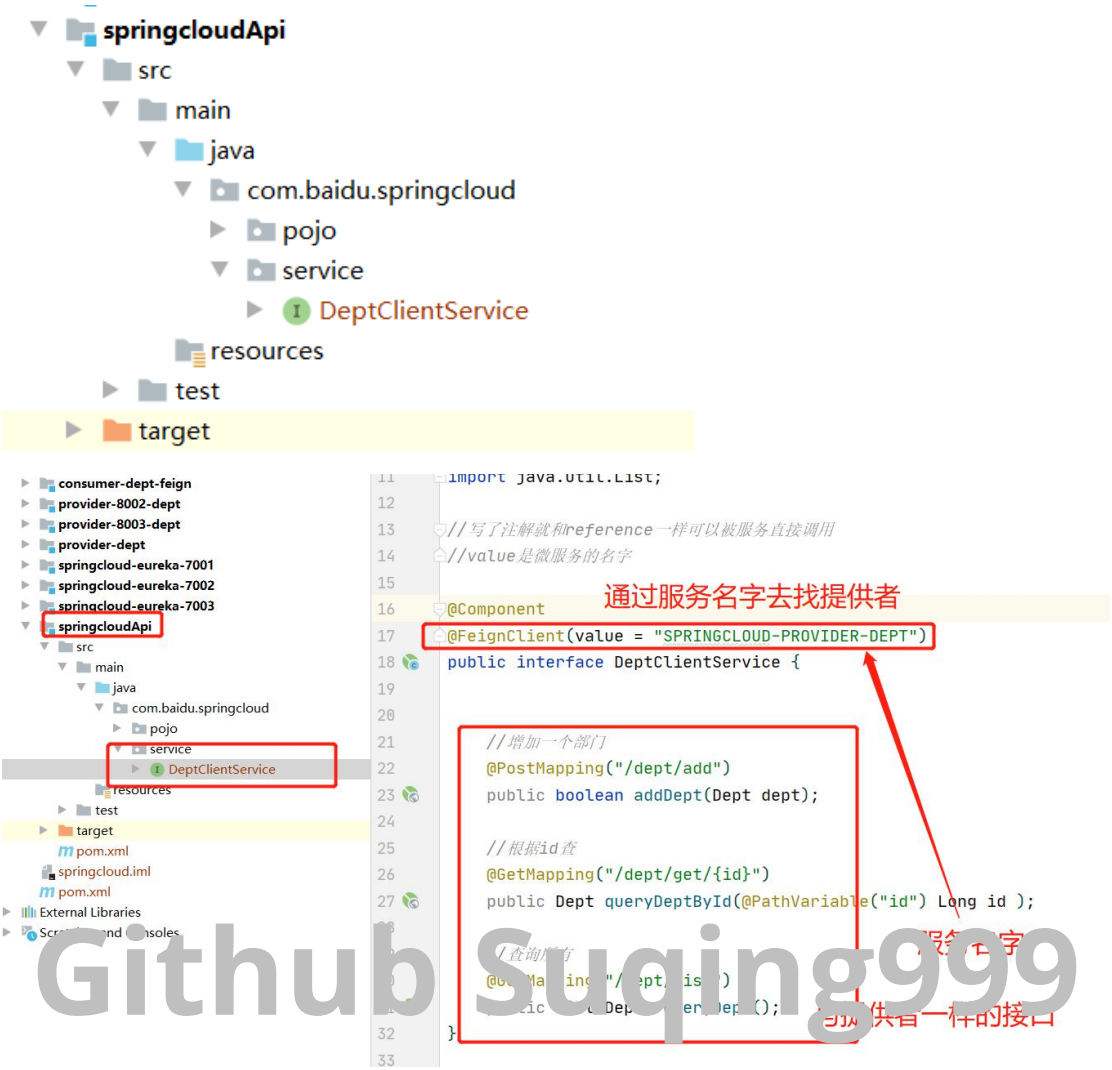
注册中心的集群配置



Feign 实现

扫描包





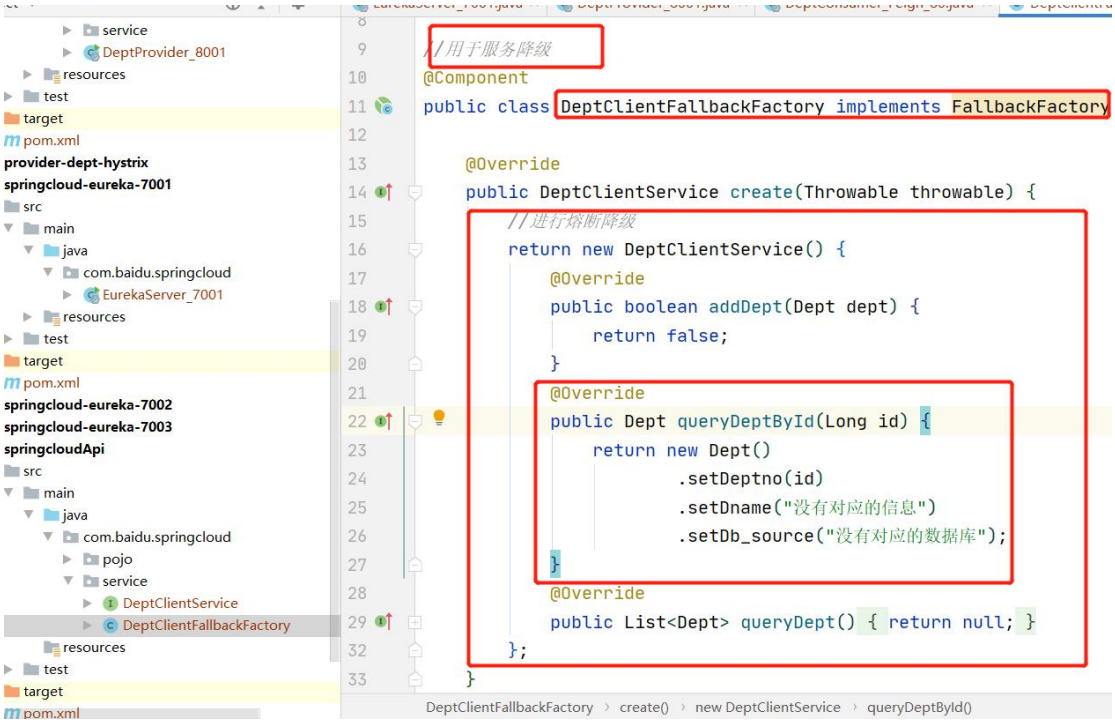
Hysitrix 服务熔断

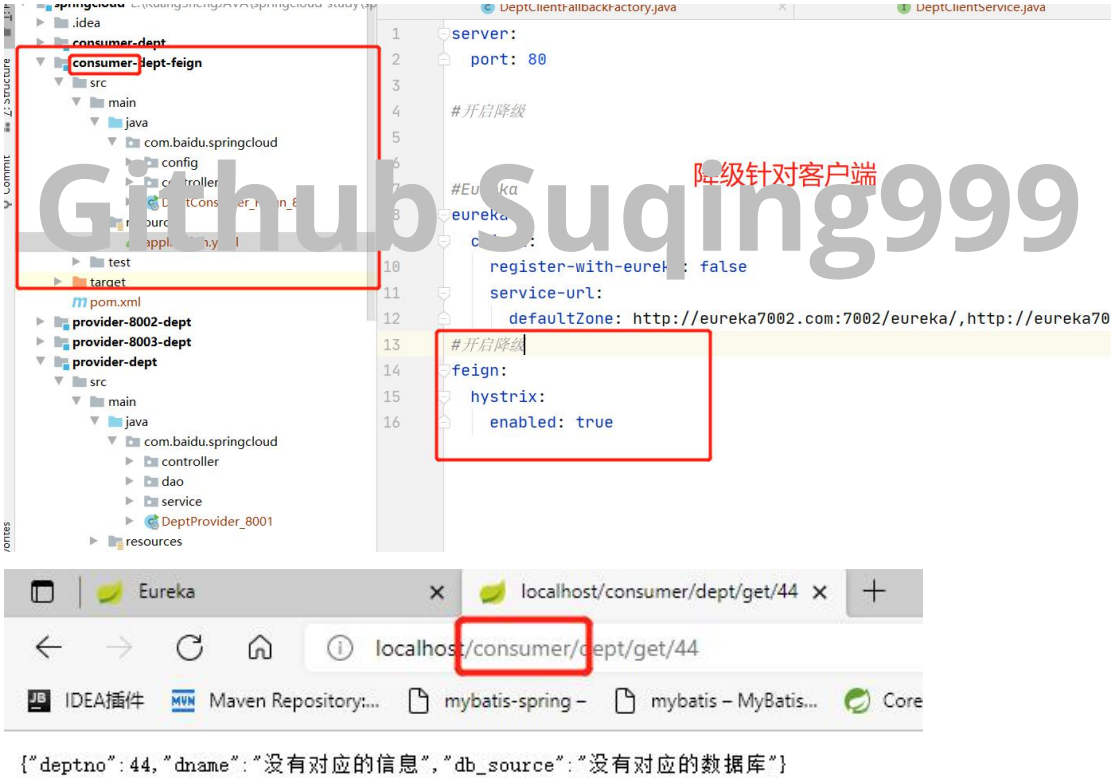
```
//启动类
@SpringBootApplication
@EnableEurekaClient
@EnableCircuitBreaker //添加对容器的支持（打开断路器）
public class DeptProvider_8001_hystrix {
    public static void main(String[] args) { SpringApplication.run(DeptProvider_8001_
}
```

```java
@HystrixCommand(fallbackMethod = "queryByid_Hystrix") //失败了调用备选方法（方法名字）
@GetMapping("/dept/get/{id}")
public Dept queryByid(@PathVariable("id") Long id){
    Dept dept = deptService.queryDeptById(id);
    if(dept==null){
        throw new RuntimeException("id=>"+id+"不存在该用户，或者信息无法找到");//id不存在抛出异常
    }
    return dept;
}
//备选方案
public Dept queryByid_Hystrix(@PathVariable("id") Long id){
    return new Dept()
            .setDeptno(id)
            .setDname("hystrix 不存在该id")
            .setDb_source("没有这个数据库"); //链式编程 需要在pojo加@Accessors(chain = true) 注解
}
```

熔断效果：

Hystrix 服务降级



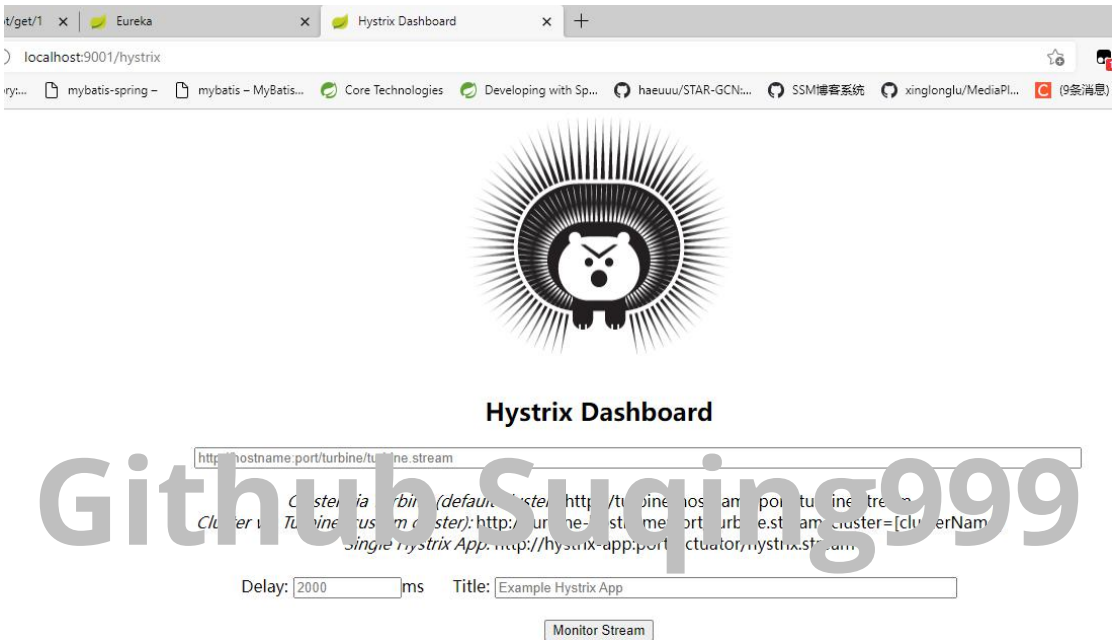


{"deptno":44,"dname":"没有对应的信息","db_source":"没有对应的数据库"}

Hystrix 监控工具 dashboard

新建一个 9001 客户端，导入 hy 和 dash 两个包，写个主启动类即可

```java
@SpringBootApplication
@EnableHystrixDashboard //开启注解dashboard
public class DeptConsumerDashboard {
    //端口9001
    //确保服务端有监控信息的依赖 actuator
    /*
     <!--监控信息-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>*/
    public static void main(String[] args) { SpringApplication.run(DeptConsumerDashboard.class, args);
}
```



监控某个地址　多少秒监控一次　自己写个 title

注意！服务端还需要写一个 bean，让该 9001 可以监控（必须是配置的 hystrix 的服务端）

```java
//启动类
@SpringBootApplication
@EnableEurekaClient
@EnableCircuitBreaker //添加对容器的支持（打开断路器！）

public class DeptProvider_8001_hystrix {
    public static void main(String[] args) { SpringApplication.run(DeptProvider_8001_hystrix.class, args);

    //增加一个dashboard的bean（死代码）
    @Bean
    public ServletRegistrationBean hystrixMetricsStreamServlet(){
        ServletRegistrationBean registrationBean = new ServletRegistrationBean(new HystrixMetricsStreamSer
        registrationBean.addUrlMappings("/actuator/hystrix.stream");//访问该页面就会被监控
        return registrationBean;
    }
```

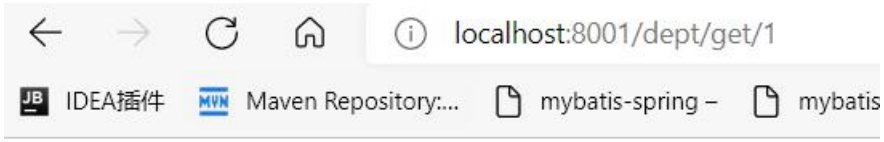
最好在监控方法加@HysitrixCommand

zuul 路由网关的作用

```java
@SpringBootApplication
@EnableZuulProxy
public class ZuulApplication_9527 {
    public static void main(String[] args) { SpringApplication.run(ZuulApplication_9527.class, args); }
}
```

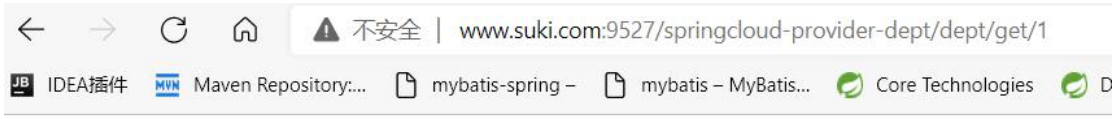
```yaml
server:
  port: 9527


spring:
  application:
    name: springcloud-zuul #服务中心注册的名字
eureka:
  client:
    service-url:
      defaultZone: http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/,http://eureka7001.com
```

启动即可，原来：



{"deptno":1,"dname":"开发部","db_source":"dbsuki"}

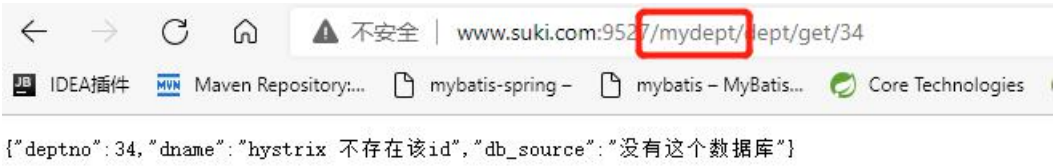
结果：避免暴露真实的微服务（但是还有微服务的名字在，不安全！）



{"deptno":1,"dname":"开发部","db_source":"dbsuki"}

避免暴露！！！！（加一个配置）

```yaml
zuul:
  routes:
    mydept.serviceId: springcloud-provider-dept  #这是原来的
    mydept.path: /mydept/**    #现在用mydept来进行
```



但是现在源路径 springcloud-provider-dept 还可以进行访问，不允许，需要忽略服务，继续配置（使用"*"通配符全部隐藏真实名字）

```yaml
zuul:
  routes:
    mydept.serviceId: springcloud-provider-dept  #这是原来的
    mydept.path: /mydept/**    #现在用mydept来进行
  ignored-services: springcloud-provider-dept #不可以再用这个访问
```

还可以再加一个公共的访问前缀，例：

```yaml
zuul:
  routes:
    mydept.serviceId: springcloud-provider-dept  #这是原来的
    mydept.path: /mydept/**    #现在用mydept来进行
  ignored-services: springcloud-provider-dept #不可以再用这个访问
  prefix: /suki
```

之后访问必须有 suki 开头