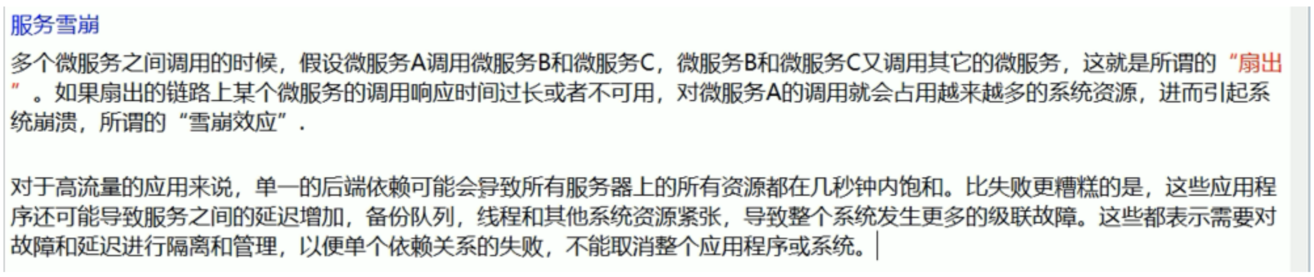
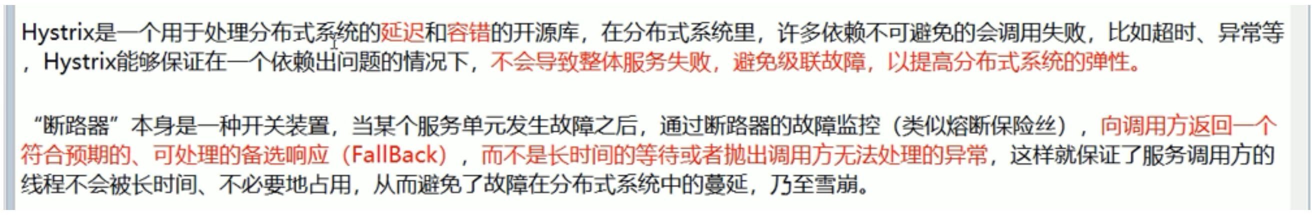
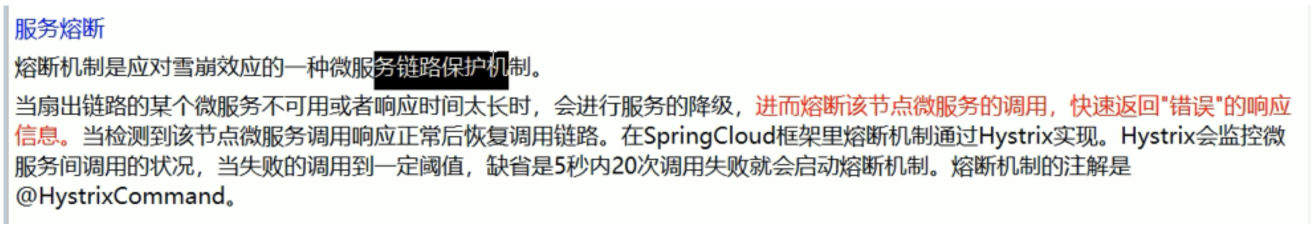
一、hystrix简介


服务雪崩
多个微服务之间调用的时候，假设微服务A调用微服务B和微服务C，微服务B和微服务C又调用其它的微服务，这就是所谓的“扇出”。如果扇出的链路上某个微服务的调用响应时间过长或者不可用，对微服务A的调用就会占用越来越多的系统资源，进而引起系统崩溃，所谓的“雪崩效应”.

对于高流量的应用来说，单一的后端依赖可能会导致所有服务器上的所有资源都在几秒钟内饱和。比失败更糟糕的是，这些应用程序还可能导致服务之间的延迟增加，备份队列，线程和其他系统资源紧张，导致整个系统发生更多的级联故障。这些都表示需要对故障和延迟进行隔离和管理，以便单个依赖关系的失败，不能取消整个应用程序或系统。


Hystrix是一个用于处理分布式系统的延迟和容错的开源库，在分布式系统里，许多依赖不可避免的会调用失败，比如超时、异常等，Hystrix能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故障，以提高分布式系统的弹性。

“断路器”本身是一种开关装置，当某个服务单元发生故障之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个符合预期的、可处理的备选响应（FallBack），而不是长时间的等待或者抛出调用方无法处理的异常，这样就保证了服务调用方的线程不会被长时间、不必要地占用，从而避免了故障在分布式系统中的蔓延，乃至雪崩。


服务熔断
熔断机制是应对雪崩效应的一种微服务链路保护机制。
当扇出链路的某个微服务不可用或者响应时间太长时，会进行服务的降级，进而熔断该节点微服务的调用，快速返回"错误"的响应信息。当检测到该节点微服务调用响应正常后恢复调用链路。在SpringCloud框架里熔断机制通过Hystrix实现。Hystrix会监控微服务间调用的状况，当失败的调用到一定阈值，缺省是5秒内20次调用失败就会启动熔断机制。熔断机制的注解是@HystrixCommand。

二、服务熔断测试

1. 项目搭建

1. 新建springcloud-provider-dept-hystrix-8001

仿照springcloud-provider-dept-8001搭建springcloud-provider-dept-hystrix-8001项目

POM依赖

```
<dependencies>
```

```xml
        <!-- 引入自己定义的api通用包，可以使用Dept部门Entity -->
        <dependency>
            <groupId>com.haoge.cloud</groupId>
            <artifactId>springcloud-api</artifactId>
            <version>${project.version}</version>
        </dependency>
        <!-- actuator主管监控信息完善 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
        <!-- 将微服务provider侧注册进eureka eureka后面没有server表示是eureka的客户端-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-eureka</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-config</artifactId>
        </dependency>


        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-core</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jetty</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>
        <!-- 修改后立即生效，热部署 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>springloaded</artifactId>
        </dependency>
        <dependency>
```

```xml
<groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
        </dependency>
        <!-- hystrix -->
        <dependency>
<groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-hystrix</artifactId>
        </dependency>
    </dependencies>
```

2. yaml配置文件

```yaml
server:
  port: 8001    # 项目的端口号

mybatis:
  config-location: classpath:mybatis/mybatis.cfg.xml        # mybatis配置文件所在路径
  type-aliases-package: com.atguigu.springcloud.entities    # 所有Entity别名类所在包
  mapper-locations:
```

```yaml
  - classpath:mybatis/mapper/**/*.xml          # mapper映射文件

spring:
   profiles:
    active:
    - dev
   application:
    name: springcloud-dept  # 当前微服务向外暴露的微服务名称
   datasource:
    type: com.alibaba.druid.pool.DruidDataSource          # 当前数据源操作类型
    driver-class-name: org.gjt.mm.mysql.Driver          # mysql驱动包
    url: jdbc:mysql://localhost:3306/cloudDB01          # 数据库名称
    username: root
    password: 123456
    dbcp2:
      min-idle: 5          # 数据库连接池的最小维持连接数
      initial-size: 5          # 初始化连接数
      max-total: 5          # 最大连接数
      max-wait-millis: 200          # 等待连接获取的最大超时时间
```

```yaml
eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
#      defaultZone: http://localhost:7001/eureka  #单机版
      defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
  instance:
    instance-id: deptService8001-hystrix     #对当前服务起的别名
    prefer-ip-address: true     #我们在eureka服务端查看服务名称的时候：访问路径可以显示IP地址
info:
  app.name: Deptservicecloud
  company.name: www.haoge.com
  build.artifactId: $project.artifactId$
  build.version: $project.version$

```

3. 修改controller

```java
@RestController
public class DeptController {
    @Autowired
    private DeptService service;

    /**
     * 根据Id查询部门
     * @param id
     * @return
     */
    //一旦调用服务方法失败并抛出了错误信息后，会自动调用@HystrixCommand标注好的fallbackMethod调用类中的指定方法
    @HystrixCommand(fallbackMethod="processHystrix_Get")
    @RequestMapping(value="/dept/get/{id}",method=RequestMethod.GET)
    public Dept get(@PathVariable Long id) {
        Dept dept= service.get(id);
        if (dept==null) {
            throw new RuntimeException("该ID：" + id + "没有没有对应的信息");
        }
        return dept;
    }
    public Dept processHystrix_Get(@PathVariable("id") Long id) {
        Dept dept = new Dept();
        dept.setDname("该ID：" + id + "没有没有对应的信息,null--@HystrixCommand");
        dept.setDeptno(id);
        dept.setDb_source("no this database in MySQL");
        return dept;
    }

}
```

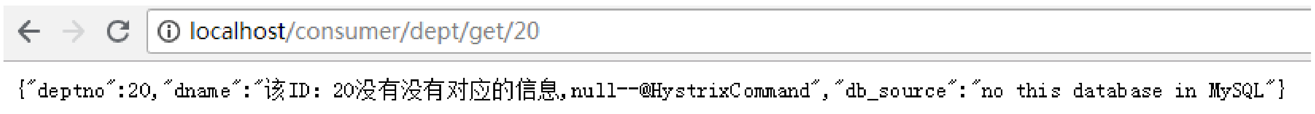
4. 主启动类@EnableCircuitBreaker

```
@SpringBootApplication
@EnableEurekaClient//这个注解的意思是本服务启动后会自动注册进eureka服务端中
@EnableDiscoveryClient//允许服务发现的注解
@EnableCircuitBreaker//对hystrix熔断机制的支持
public class DeptProvider8001_hystrix {
    public static void main(String[] args) {
        SpringApplication.run(DeptProvider8001_hystrix.class, args);
    }
}
```

5. 测试

启动7001,7002,7003项目，启动springcloud-provider-dept-hystrix-8001项目，启动springcloud-consumer-dept-80

效果：


localhost/consumer/dept/get/20

```
{"deptno":20,"dname":"该ID: 20没有没有对应的信息,null--@HystrixCommand","db_source":"no this database in MySQL"}
```

6. 工作过程

当我们访问不存在的数据的时候，控制器throw new RuntimeException

，这个时候在Hystrix的作用下走我们指定的方法processHystrix_Get，返回一个预期的结果。

三、服务降级测试

服务降级：整体资源不够了，我们先关掉一部分，待度过难关之后再开启回来。

服务降级是在客户端完成的，和服务端没有关系。

1. 修改springcloud-api

新建DeptClientServiceFallBackFactory，实现FallbackFactory<eptClientService>。代码如下

```java
@Component
public class DeptClientServiceFallBackFactory implements FallbackFactory<DeptClientService> {

    @Override
    public DeptClientService create(Throwable arg0) {
        return new DeptClientService() {

            @Override
            public Dept get(long id) {
                Dept dept = new Dept();
                dept.setDname("该ID：" + id + "没有没有对应的信息,Consumer客户端提供的降级信息,此刻服务Provider已经关闭");
                dept.setDeptno(id);
                dept.setDb_source("no this database in MySQL");
                return dept;
            }

            @Override
            public List<Dept> list() {
                // TODO Auto-generated method stub
                return null;
            }

            @Override
            public boolean add(Dept dept) {
                // TODO Auto-generated method stub
                return false;
                }
            };
    }
}
```

2. 修改DeptClientService

```
//@FeignClient(value = "springcloud-dept")
@FeignClient(value="springcloud-dept",fallbackFactory=DeptClientServiceFallBackFactory.class)
public interface DeptClientService {

    @RequestMapping(value = "/dept/get/{id}", method = RequestMethod.GET)
    public Dept get(@PathVariable("id") long id);

    @RequestMapping(value = "/dept/list", method = RequestMethod.GET)
    public List<Dept> list();

    @RequestMapping(value = "/dept/add", method = RequestMethod.POST)
    public boolean add(Dept dept);

}
```

3. 修改springcloud-consumer-dept-feign yaml文件

```yaml
server:
  port: 80

eureka:
  client:
    register-with-eureka: false #自己不能注册
    service-url:
      defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
#增加的内容
feign:
  hystrix:
    enabled: true
```

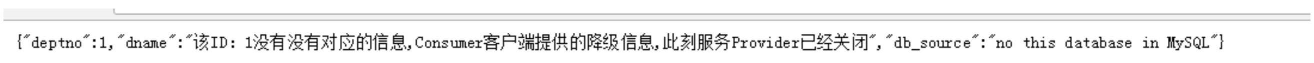
4. 测试

启动7001,7002,7003三个eureka Server,然后启动springcloud-provider-dept-8001，再启动springcloud-consumer-dept-feign。

正常访问http://localhost/consumer/dept/get/1

然后故意关闭springcloud-provider-dept-8001再次访问http://localhost/consumer/dept/get/1

5. 测试结果


```
{"deptno":1,"dname":"该ID：1没有没有对应的信息,Consumer客户端提供的降级信息,此刻服务Provider已经关闭","db_source":"no this database in MySQL"}
```

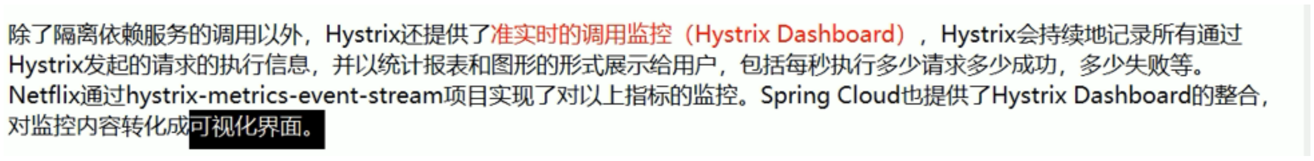
6. 工作原理

将所有关于DeptClientService中方法的异常处理统一用接口处理，如上若

DeptClientService中有服务dwon掉了，我们就找DeptClientServiceFallBackFactory中对应的方法进行服务降级处理

四、hystrix Dashboard

1. 简介


除了隔离依赖服务的调用以外，Hystrix还提供了准实时的调用监控（Hystrix Dashboard），Hystrix会持续地记录所有通过Hystrix发起的请求的执行信息，并以统计报表和图形的形式展示给用户，包括每秒执行多少请求多少成功，多少失败等。Netflix通过hystrix-metrics-event-stream项目实现了对以上指标的监控。Spring Cloud也提供了Hystrix Dashboard的整合，对监控内容转化成可视化界面。

2. 项目搭建

1. 新建maven子项目

新建maven子项目 springcloud-consumer-hystrix-dashboard

2. POM依赖

```
<dependencies>
```

```xml
        <!-- 自己定义的api -->
        <dependency>
            <groupId>com.haoge.springCloud</groupId>
            <artifactId>firstMainCloud-api</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- 修改后立即生效，热部署 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>springloaded</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
        </dependency>
        <!-- Ribbon相关 -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-eureka</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-ribbon</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-config</artifactId>
        </dependency>
        <!-- feign相关 -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-feign</artifactId>
        </dependency>
        <!-- hystrix和 hystrix-dashboard相关 -->
        <dependency>
```

```xml
<groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-hystrix</artifactId>
        </dependency>
        <dependency>
<groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>
        </dependency>
    </dependencies>
```

3. YAML文件

```yaml
server:
  port: 9001
```

4. 主启动类

```java
@SpringBootApplication
@EnableHystrixDashboard
public class DeptConsumer_DashBoard_App {

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

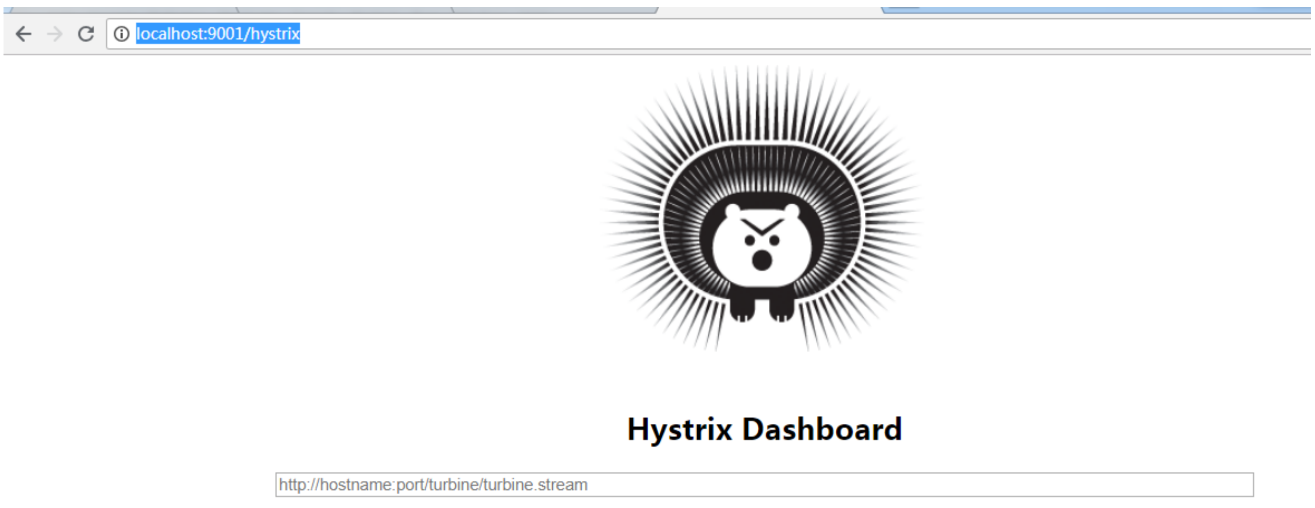
所有provider项目（8001,8002,8003）都需要有监控组件。

```xml
<!-- actuator主管监控信息完善 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
```

5. 测试

启动项目springcloud-consumer-hystrix-dashboard

访问 http://localhost:9001/hystrix

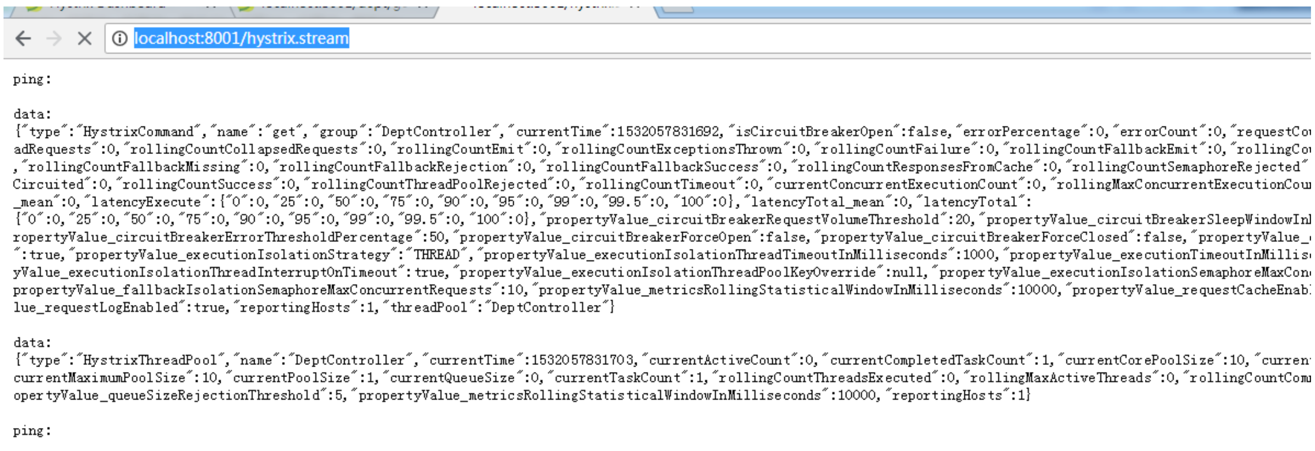


分别启动7001,7002,7003 eureka server三个项目，再启动springcloud-provider-dept-hystrix-8001

访问：http://localhost:8001/dept/get/2

访问：http://localhost:8001/hystrix.stream

效果：



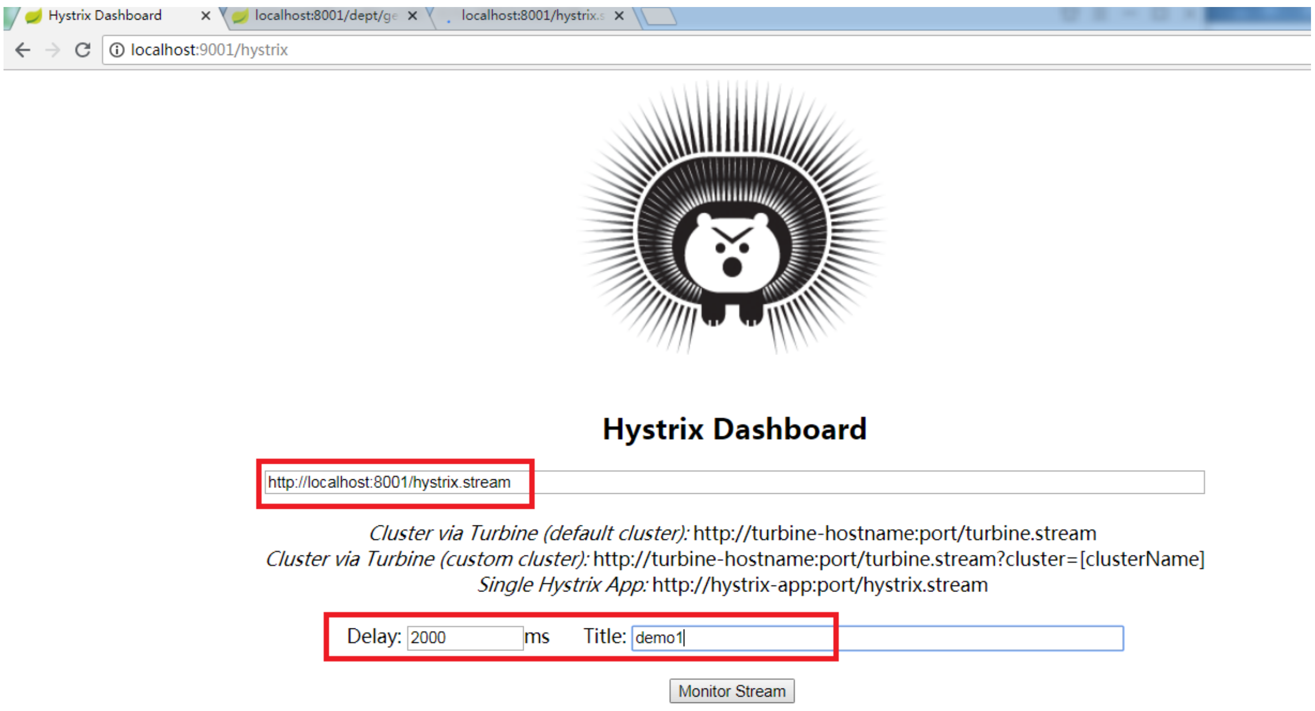
```
ping:

data:
{"type":"HystrixCommand","name":"get","group":"DeptController","currentTime":1532057831692,"isCircuitBreakerOpen":false,"errorPercentage":0,"errorCount":0,"requestCou
adRequests":0,"rollingCountCollapsedRequests":0,"rollingCountEmit":0,"rollingCountExceptionsThrown":0,"rollingCountFailure":0,"rollingCountFallbackEmit":0,"rollingCou
,"rollingCountFallbackMissing":0,"rollingCountFallbackRejection":0,"rollingCountFallbackSuccess":0,"rollingCountResponsesFromCache":0,"rollingCountSemaphoreRejected"
Circuited":0,"rollingCountSuccess":0,"rollingCountThreadPoolRejected":0,"rollingCountTimeout":0,"currentConcurrentExecutionCount":0,"rollingMaxConcurrentExecutionCou
_mean":0,"latencyExecute":{"0":0,"25":0,"50":0,"75":0,"90":0,"95":0,"99":0,"99.5":0,"100":0},"latencyTotal_mean":0,"latencyTotal":
{"0":0,"25":0,"50":0,"75":0,"90":0,"95":0,"99":0,"99.5":0,"100":0},"propertyValue_circuitBreakerRequestVolumeThreshold":20,"propertyValue_circuitBreakerSleepWindowIn
ropertyValue_circuitBreakerErrorThresholdPercentage":50,"propertyValue_circuitBreakerForceOpen":false,"propertyValue_circuitBreakerForceClosed":false,"propertyValue_
":true,"propertyValue_executionIsolationStrategy":"THREAD","propertyValue_executionIsolationThreadTimeoutInMilliseconds":1000,"propertyValue_executionTimeoutInMillis
yValue_executionIsolationThreadInterruptOnTimeout":true,"propertyValue_executionIsolationThreadPoolKeyOverride":null,"propertyValue_executionIsolationSemaphoreMaxCon
propertyValue_fallbackIsolationSemaphoreMaxConcurrentRequests":10,"propertyValue_metricsRollingStatisticalWindowInMilliseconds":10000,"propertyValue_requestCacheEnab
lue_requestLogEnabled":true,"reportingHosts":1,"threadPool":"DeptController"}

data:
{"type":"HystrixThreadPool","name":"DeptController","currentTime":1532057831703,"currentActiveCount":0,"currentCompletedTaskCount":1,"currentCorePoolSize":10,"curren
currentMaximumPoolSize":10,"currentPoolSize":1,"currentQueueSize":0,"currentTaskCount":1,"rollingCountThreadsExecuted":0,"rollingMaxActiveThreads":0,"rollingCountCom
opertyValue_queueSizeRejectionThreshold":5,"propertyValue_metricsRollingStatisticalWindowInMilliseconds":10000,"reportingHosts":1}

ping:
```

6. 使用图形化监控界面



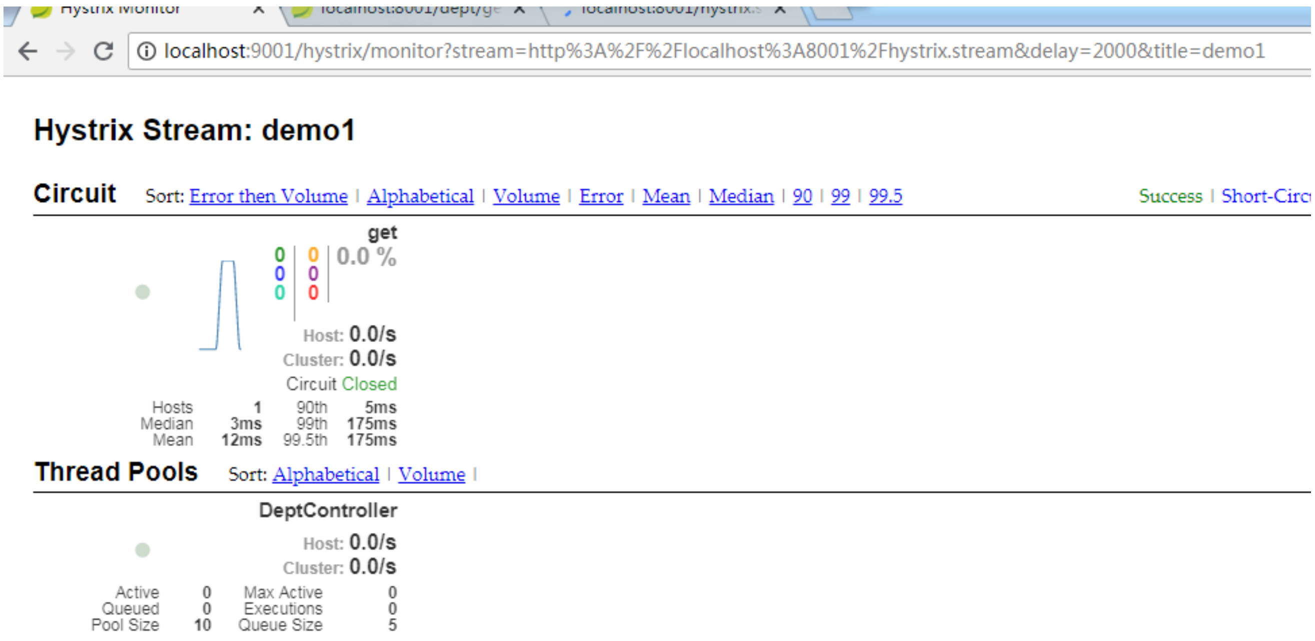
Hystrix Dashboard

http://localhost:8001/hystrix.stream

Cluster via Turbine (default cluster): http://turbine-hostname:port/turbine.stream
Cluster via Turbine (custom cluster): http://turbine-hostname:port/turbine.stream?cluster=[clusterName]
Single Hystrix App: http://hystrix-app:port/hystrix.stream

Delay: 2000 ms Title: demo1

Monitor Stream



localhost:9001/hystrix/monitor?stream=http%3A%2F%2Flocalhost%3A8001%2Fhystrix.stream&delay=2000&title=demo1

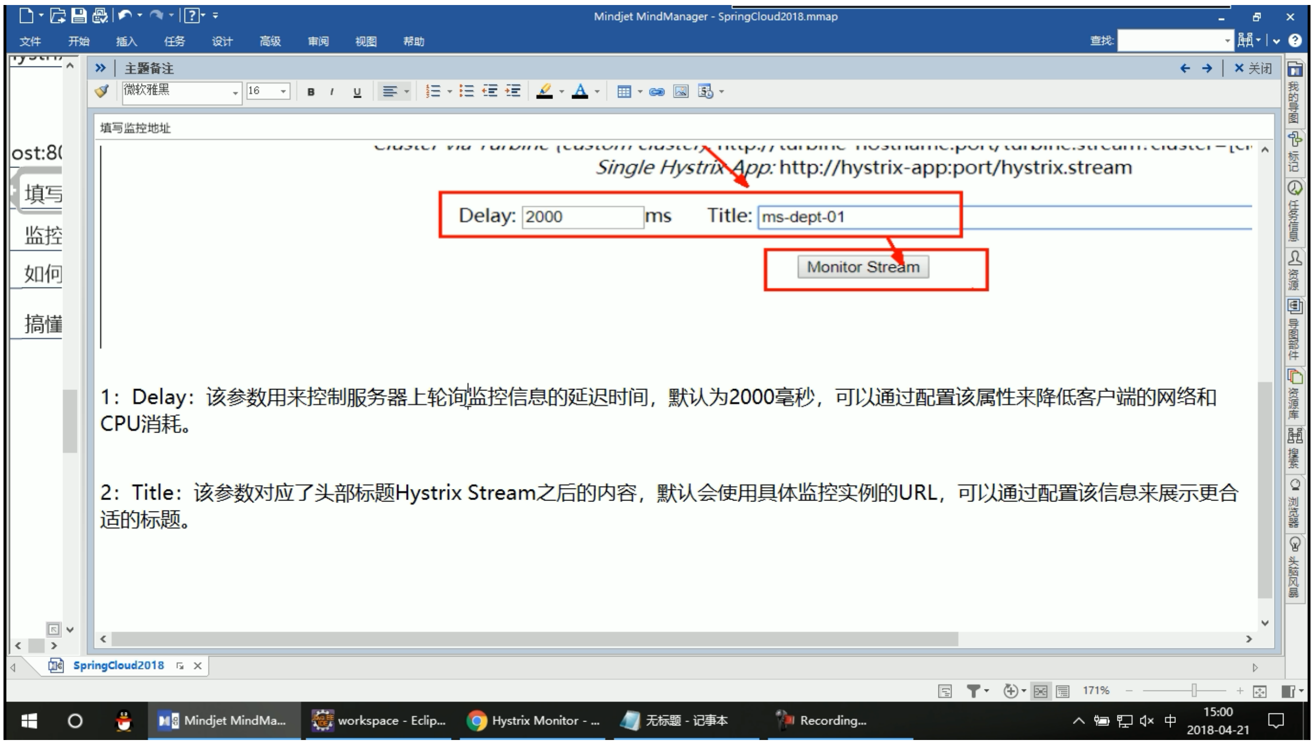
Hystrix Stream: demo1

Circuit Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5 Success | Short-Circ

```
get
0 | 0 | 0.0 %
0 | 0
0 | 0
Host: 0.0/s
Cluster: 0.0/s
Circuit Closed
Hosts      1    90th     5ms
Median   3ms    99th   175ms
Mean    12ms  99.5th   175ms
```

Thread Pools Sort: Alphabetical | Volume |

```
DeptController
Host: 0.0/s
Cluster: 0.0/s
Active      0   Max Active   0
Queued      0   Executions   0
Pool Size  10   Queue Size   5
```



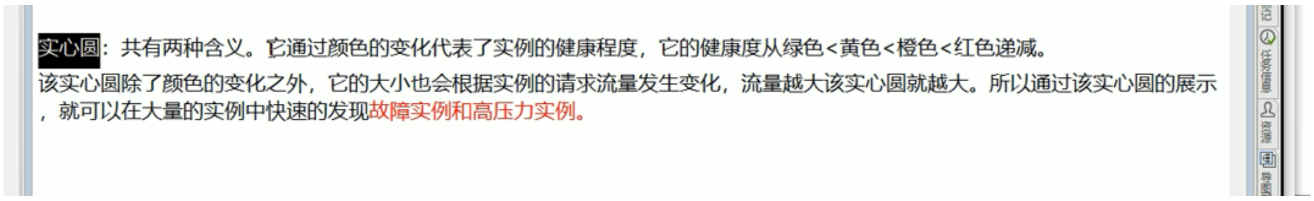
Single Hystrix App: http://hystrix-app:port/hystrix.stream

Delay: 2000 ms Title: ms-dept-01

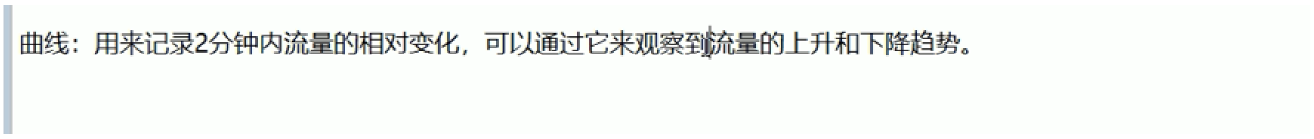
Monitor Stream

1：Delay：该参数用来控制服务器上轮询监控信息的延迟时间，默认为2000毫秒，可以通过配置该属性来降低客户端的网络和CPU消耗。

2：Title：该参数对应了头部标题Hystrix Stream之后的内容，默认会使用具体监控实例的URL，可以通过配置该信息来展示更合适的标题。



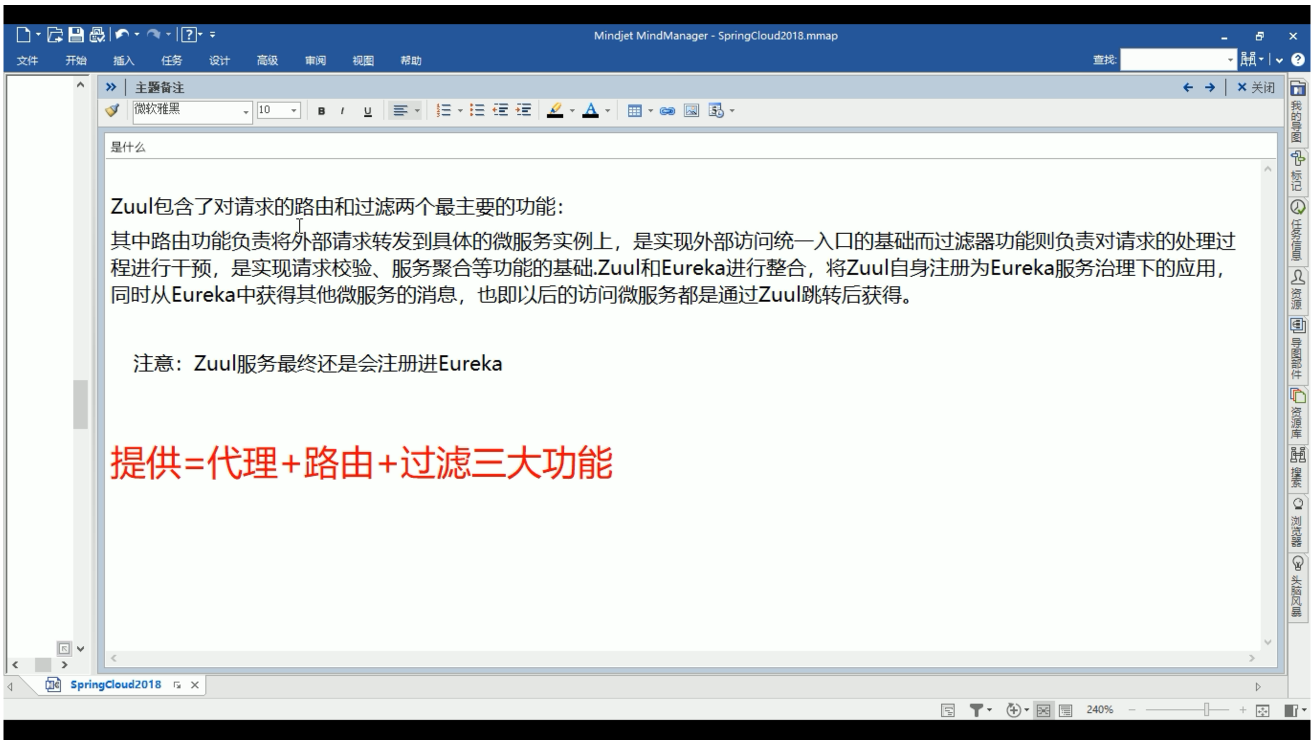
实心圆：共有两种含义。它通过颜色的变化代表了实例的健康程度，它的健康度从绿色<黄色<橙色<红色递减。
该实心圆除了颜色的变化之外，它的大小也会根据实例的请求流量发生变化，流量越大该实心圆就越大。所以通过该实心圆的展示，就可以在大量的实例中快速的发现故障实例和高压力实例。



曲线：用来记录2分钟内流量的相对变化，可以通过它来观察到流量的上升和下降趋势。

五、Zuul

1. Zuul简介



是什么

Zuul包含了对请求的路由和过滤两个最主要的功能：

其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础而过滤器功能则负责对请求的处理过程进行干预，是实现请求校验、服务聚合等功能的基础.Zuul和Eureka进行整合，将Zuul自身注册为Eureka服务治理下的应用，同时从Eureka中获得其他微服务的消息，也即以后的访问微服务都是通过Zuul跳转后获得。

注意：Zuul服务最终还是会注册进Eureka

提供=代理+路由+过滤三大功能

2. 项目搭建

新建maven子模块springcloud-zuul-gateway-9527

1. POM依赖

```xml
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://maven.apache.org/POM/
4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.haoge.cloud</groupId>
    <artifactId>springcloud</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>springcloud-zuul-gateway-9527</artifactId>

  <dependencies>
        <!-- zuul路由网关 -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-zuul</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-eureka</artifactId>
        </dependency>
        <!-- actuator监控 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
        <!-- hystrix容错 -->
        <dependency>
```

```xml
<groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-hystrix</artifactId>
        </dependency>
        <dependency>
<groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-config</artifactId>
        </dependency>
        <!-- 日常标配 -->
        <dependency>
<groupId>com.haoge.springCloud</groupId>
            <artifactId>firstMainCloud-api</artifactId>
            <version>${project.version}
</version>
        </dependency>
        <dependency>
<groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jetty</artifactId>
        </dependency>
        <dependency>
```

```xml
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>
        <!-- 热部署插件 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>springloaded</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
        </dependency>
    </dependencies>
</project>
```

2. yaml文件

```yaml
server:
  port: 9527

spring:
  application:
    name: springcloud-zuul-gateway

eureka:
  client:
    service-url:
      defaultZone: http://eureka7001.com:7001/eureka,http://eureka7002.com:7002/eureka,http://eureka7003.com:7003/eureka
  instance:
    instance-id: gateway-9527.com
    prefer-ip-address: true

zuul:
#  ignored-services: springcloud-dept
  prefix: /haoge
  ignored-services: "*"
  routes:
    mydept.serviceId: springcloud-dept
    mydept.path: /mydept/**

info:
  app.name: Deptservicecloud
  company.name: www.haoge.com
  build.artifactId: $project.artifactId$
  build.version: $project.version$

```

3. 主程序

```java
@SpringBootApplication
@EnableZuulProxy
public class Zuul_9527_StartSpringCloudApp {

    public static void main(String[] args) {
        SpringApplication.run(Zuul_9527_StartSpringCloudApp.class, args);
    }
}
```

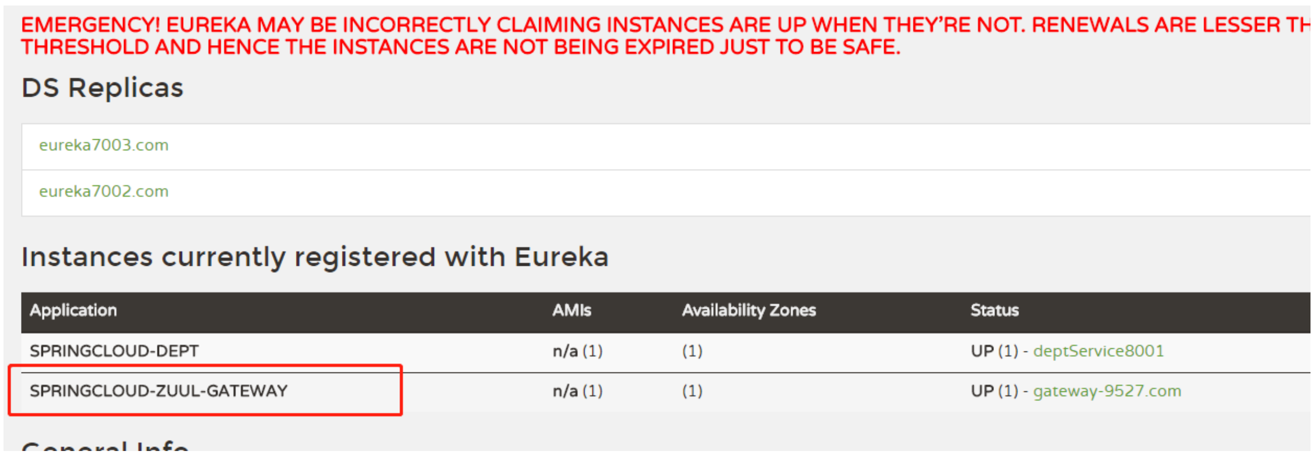
4. 配置host

```
127.0.0.1 myzuul.com
```

5. 测试

启动三个eureka server 7001,7002,7003项目。再启动springcloud-provider-dept-8001。再启动springcloud-zuul-gateway-9527

访问http://localhost:7001/



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

DS Replicas

eureka7003.com

eureka7002.com

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SPRINGCLOUD-DEPT	n/a (1)	(1)	UP (1) - deptService8001
SPRINGCLOUD-ZUUL-GATEWAY	n/a (1)	(1)	UP (1) - gateway-9527.com

即zuul也会把自己作为服务注册在eureka中。

访问数据：

不启用路由：http://localhost:8001/dept/get/2

启用路由：http://myzuul.com:9527/springcloud-dept/dept/get/2

3. zuul路由映射规则

1. 配置用mydept代替springcloud-dept

```
zuul:
  routes:
    mydept.serviceId: springcloud-dept
    mydept.path: /mydept/**
```

此时

http://myzuul.com:9527/springcloud-dept/dept/get/2 http://myzuul.com:9527/mydept/dept/get/2

都可以访问

2. 禁用服务

```
#禁用springcloud-dept服务（禁用单个服务）
ignored-services: springcloud-dept
#禁用所有的服务名称
ignored-services: "*"
```

示例如下：

```
zuul:
  ignored-services: springcloud-dept
#  ignored-services: "*"
  routes:
    mydept.serviceId: springcloud-dept
    mydept.path: /mydept/**
```

此时http://myzuul.com:9527/springcloud-dept/dept/get/2不可以访问

http://myzuul.com:9527/mydept/dept/get/2可以访问

3. 增加前缀

访问服务之前增加haoge前缀。

```yaml
zuul:
  ignored-services: springcloud-dept
  prefix: /haoge
  routes:
    mydept.serviceId: springcloud-dept
    mydept.path: /mydept/**
```

此时访问路径改为：

http://myzuul.com:9527/haoge/mydept/dept/get/2