SpringCloud之五

熔断器hystrix

在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用（RPC），在Spring Cloud可以用RestTemplate+Ribbon和Feign来调用。为了保证其高可用，单个服务通常会集群部署。由于网络原因或者自身的原因，服务并不能保证100%可用，如果单个服务出现问题，调用这个服务就会出现线程阻塞，此时若有大量的请求涌入，Servlet容器的线程资源会被消耗完毕，导致服务瘫痪。服务与服务之间的依赖性，故障会传播，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。
为了解决这个问题，业界提出了断路器模型。

一、Hystrix介绍

Hystrix是一个延迟和容错库，旨在隔离对远程系统，服务和第三方库的访问点，停止级联故障，并在复杂的分布式系统中实现弹性，在这些系统中，故障是不可避免的。

Hystrix的历史

Hystrix从Netflix API团队于2011年开始的弹性工程工作演变而来。2012年，Hystrix继续发展和成熟，Netflix的许多团队都采用了它。今天，每天在Netflix通过Hystrix执行数百亿个线程隔离和数千亿信号量隔离的调用。这导致了正常运行时间和弹性的显着改善。

以下链接提供了有关Hystrix的更多背景信息以及它试图解决的挑战：

- “使Netflix API更具弹性”
- “高容量，分布式系统中的容错”
- “Netflix API的性能和容错能力”
- “面向服务架构中的应用程序弹性”
- [“Netflix的应用弹性工程与运营”]（https://speakerdeck.com/benjchristensen/application-resilience-engineering-and-operations-at-netflix）

什么是Hystrix？

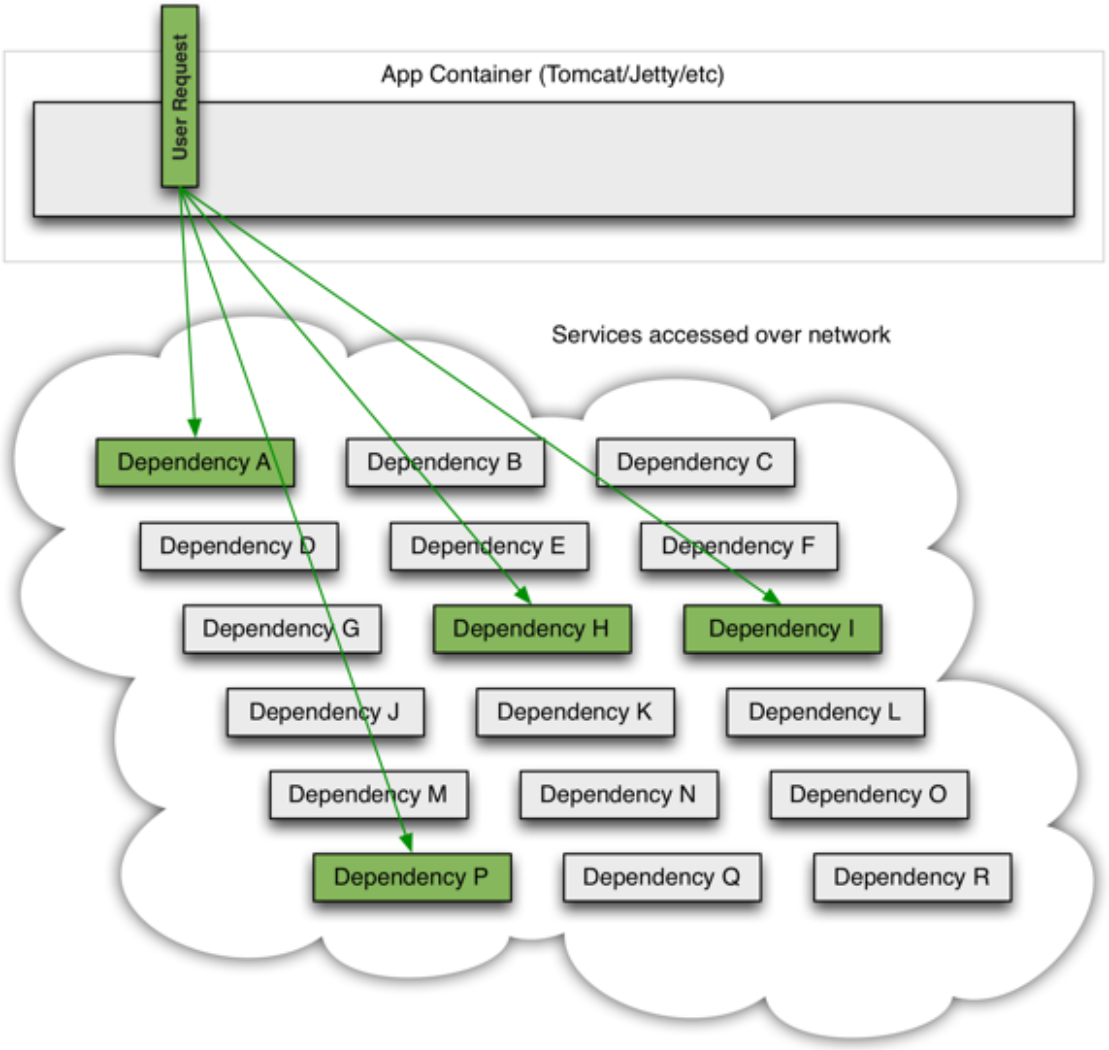
Hystrix旨在执行以下操作：

- 通过第三方客户端库访问（通常通过网络）依赖关系，以防止和控制延迟和故障。
- 在复杂的分布式系统中停止级联故障。
- 快速失败并迅速恢复。
- 在可能的情况下，后退并优雅地降级。

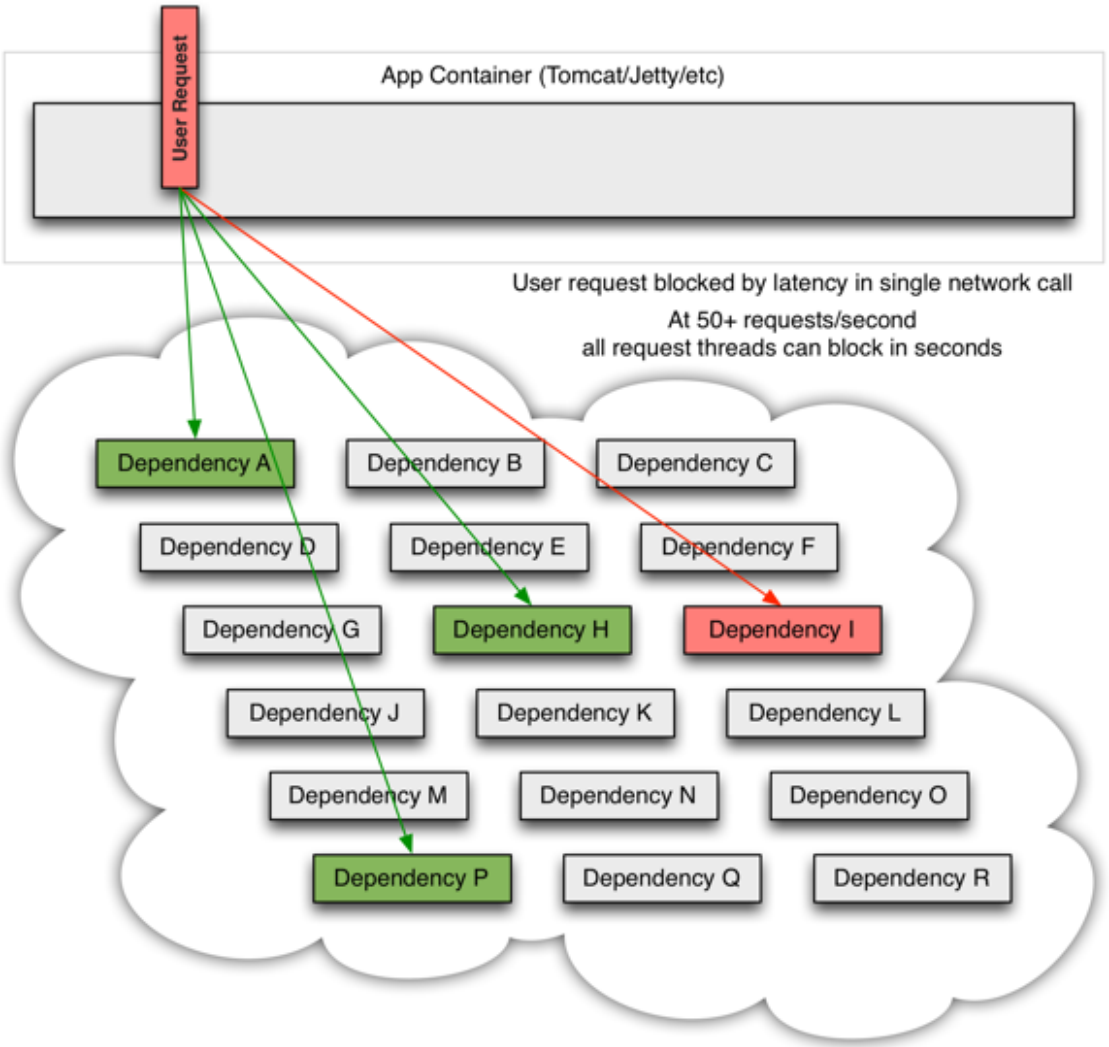
- 实现近实时监控，警报和操作控制。

Netflix开源了Hystrix组件，实现了断路器模式，SpringCloud对这一组件进行了整合。 在微服务架构中，一个请求需要调用多个服务是非常常见的，如下图：

当一切都很健康时，请求流可能如下所示：

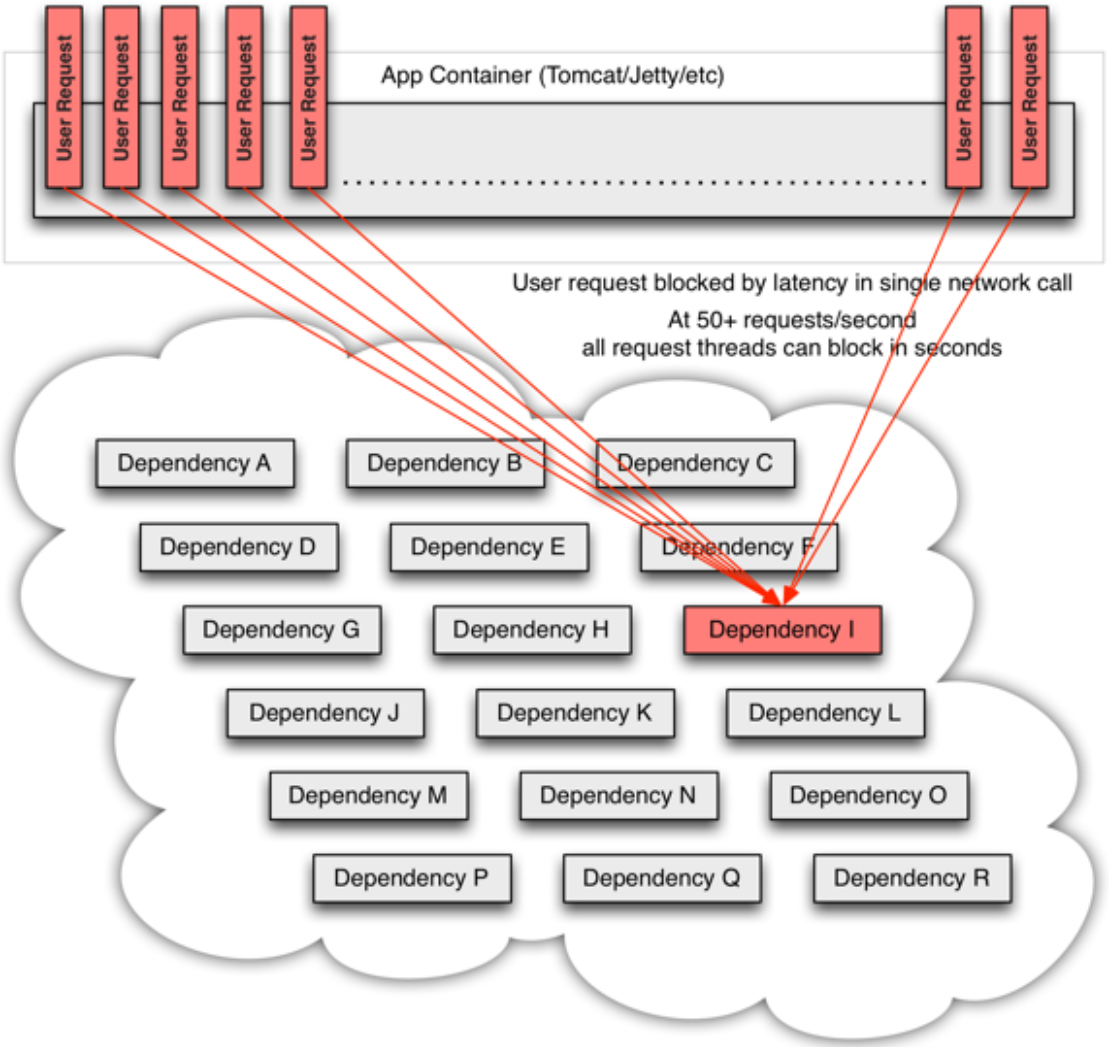


当许多后端系统中的一个变得失效时，它可以阻止整个用户请求：



对于高流量处理，单个后端依赖性变为潜在可能导致所有服务器上的所有资源在几秒钟内变得饱和。

应用程序中通过网络或可能导致网络请求进入客户端库的每个点都是潜在故障的来源。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，从而备份队列，线程和其他系统资源，从而导致整个系统出现更多级联故障。



当通过第三方客户端执行网络访问时，这些问题会加剧 - 一个“黑匣子”，其中隐藏实施细节并且可以随时更改，并且每个客户端库的网络或资源配置不同，并且通常难以监控更改。更糟糕的是传递依赖性，这些依赖性执行潜在的昂贵或容易出错的网络调用，而不会被应用程序显式调用。网络连接失败或降级。服务和服务器失败或变慢。新库或服务部署会更改行为或性能特征。客户端库有bug。所有这些都表示需要隔离和管理的故障和延迟，**以便单个故障依赖性不会占用整个应用程序或系统。**

Hystrix的工作原理是：

- 防止任何单个依赖项用尽所有容器（例如Tomcat）用户线程。
- 脱落负载并快速失败而不是排队。
- 在可行的地方提供回退以保护用户免于失败。

- 使用隔离技术（例如隔板，泳道和断路器模式）来限制任何一个依赖项的影响。
- 通过近实时指标，监控和警报优化发现时间
- 通过配置更改的低延迟传播优化恢复时间，并支持Hystrix大多数方面的动态属性更改，从而允许您使用低延迟反馈循环进行实时操作修改。
- 防止整个依赖关系客户端执行中的故障，而不仅仅是网络流量。

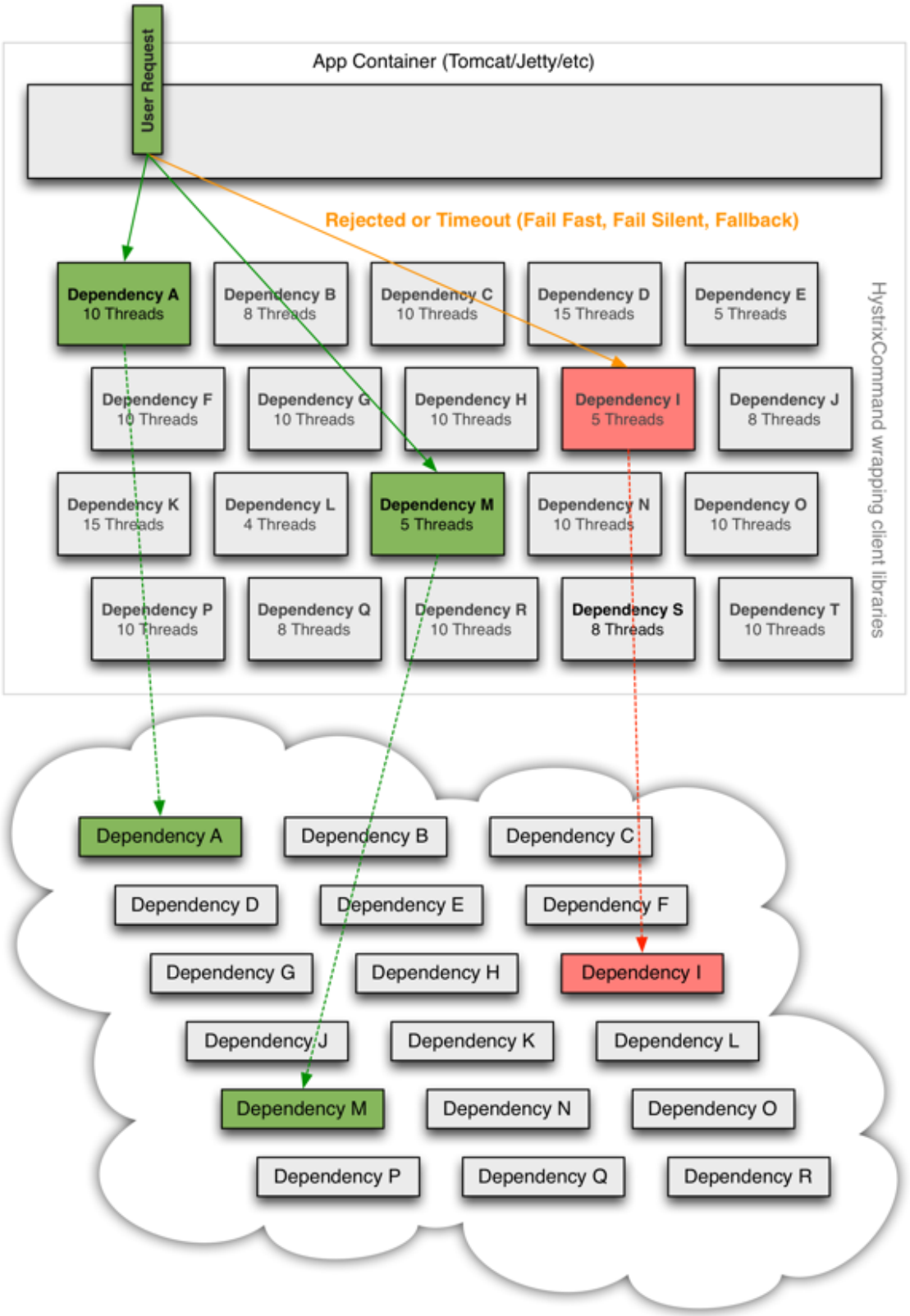
Hystrix通过以下方式实现：

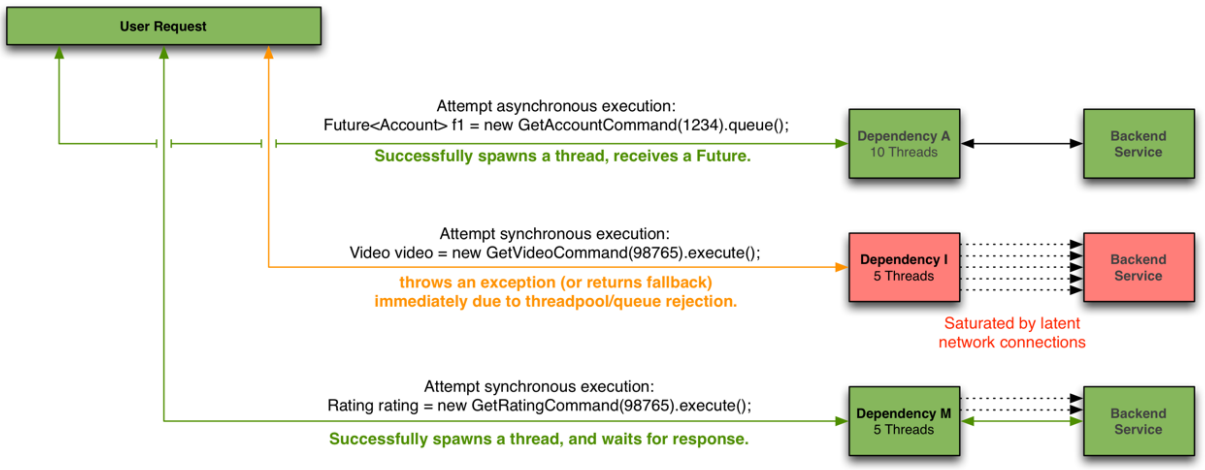
- 将对外部系统（或“依赖项”）的所有调用包含在通常在单独线程内执行的对象HystrixCommand或HystrixObservableCommand对象中（这是命令模式的示例）。
- 定时调用的时间超过您定义的阈值。有一个默认的，而是由“属性”的方式对大多数依赖你自定义设置这些超时，使它们比测量的99.5略高日每个依存性百分性能。

99.99^{30} = 99.7% uptime
0.3% of 1 billion requests = 3,000,000 failures
2+ hours downtime/month even if all dependencies have excellent uptime.

- 为每个依赖项维护一个小的线程池（或信号量）；如果它变满，则会立即拒绝发往该依赖项的请求，而不是排队。
- 衡量成功，失败（客户端引发的异常），超时和线程拒绝。
- 如果服务的错误百分比超过阈值，则手动或自动地使断路器跳闸以停止对特定服务的所有请求一段时间。
- 当请求失败时执行回退逻辑，被拒绝，超时或短路。
- 近乎实时地监控指标和配置更改。

当您使用Hystrix来包装每个底层依赖项时，上图中显示的体系结构将更改为类似于下图。每个依赖项彼此隔离，在发生延迟时可以饱和的资源受到限制，并且在回退逻辑中涵盖，该逻辑决定在依赖项中发生任何类型的故障时要做出的响应：


User Request
App Container (Tomcat/Jetty/etc)
Rejected or Timeout (Fail Fast, Fail Silent, Fallback)
Dependency A
10 Threads
Dependency B
8 Threads
Dependency C
10 Threads
Dependency D
15 Threads
Dependency E
5 Threads
Dependency F
10 Threads
Dependency G
10 Threads
Dependency H
10 Threads
Dependency I
5 Threads
Dependency J
8 Threads
Dependency K
15 Threads
Dependency L
4 Threads
Dependency M
5 Threads
Dependency N
10 Threads
Dependency O
10 Threads
Dependency P
10 Threads
Dependency Q
8 Threads
Dependency R
10 Threads
Dependency S
8 Threads
Dependency T
10 Threads
HystrixCommand wrapping client libraries
Dependency A
Dependency B
Dependency C
Dependency D
Dependency E
Dependency F
Dependency G
Dependency H
Dependency I
Dependency J
Dependency K
Dependency L
Dependency M
Dependency N
Dependency O
Dependency P
Dependency Q
Dependency R



废话说的太多了，让人昏昏欲睡，api文档看的也让我老眼昏花，还是实际操作一下！！！！

二、Hystrix的使用

2.1、在ribbon项目中使用Hystrix

2.1.1、准备

启动eureka-server项目，端口号是8761。

启动login-service项目，端口号是8762。

2.1.2、改造ribbon-server项目

首先在pox.xml文件中加入spring-cloud-starter-netflix-hystrix的起步依赖：

pom.xml文件添加依赖如下：

```
<dependency>
 <groupId>org.springframework.cloud</groupId>
 <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
```

然后在程序的启动类ServiceRibbonApplication 加@EnableHystrix注解开启Hystrix：

```
package xyz.jiangnanke.ribbonservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
```

```
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.hystrix.EnableHystrix;
import org.springframework.context.annotation.Bean;
 import org.springframework.web.client.RestTemplate;

@EnableEurekaClient
@SpringBootApplication
@EnableDiscoveryClient
@EnableHystrix
public class RibbonServiceApplication {

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

 @Bean
 @LoadBalanced
 RestTemplate restTemplate() {
 return new RestTemplate();
 }

}

```

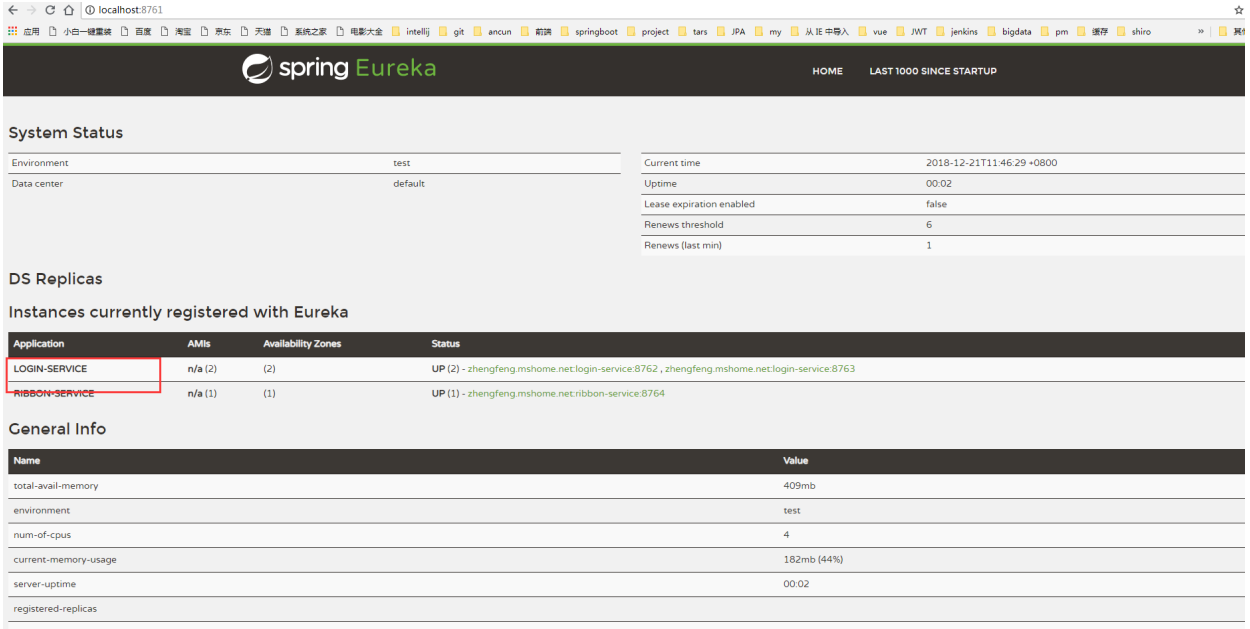
接下来就是修改熔断逻辑，即LoginService.java类了，在login(String name)方法上加上@HystrixCommand注解。该注解对该方法创建了熔断器的功能，并指定了fallbackMethod熔断方法loginError，熔断方法直接返回了一个字符串，字符串为： "hi " + name + ",I am very Sorry, you login is Error!"，代码如下：

```
package xyz.jiangnanke.ribbonservice.service;

import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

/**
 * @Auther: zhengfeng
 * @Date: 2018\12\20 0020 10:18
 * @Description:
 */
@Service
```

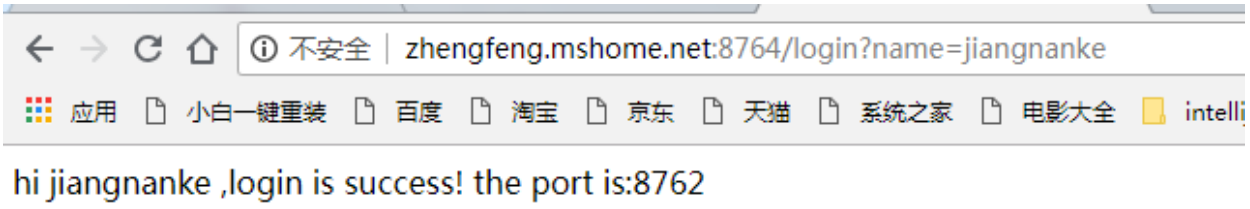
```java
public class LoginService {

 @Autowired
 RestTemplate restTemplate;

 @HystrixCommand(fallbackMethod = "loginError")
 public String login(String name) {
 return restTemplate.getForObject("http://LOGIN-SERVICE/login?name="+name,String.class);
 }

 public String loginError(String name){
 return "hi " + name + ",I am very Sorry, you login is Error!"
 }
}
```

2.1.3、运行项目

OK！ 大功告成，可以启动项目了，然后测试一下！查看到浏览器返回的情况如下图：

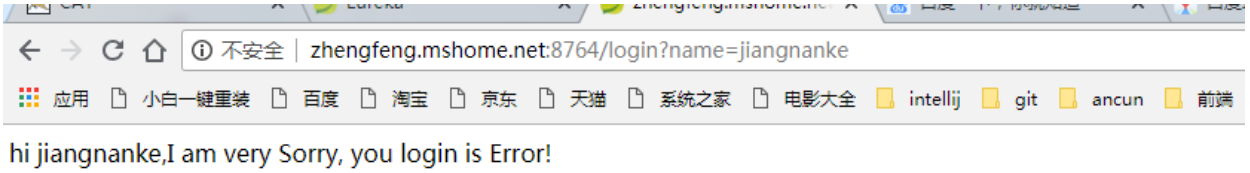


正常情况：



hi jiangnanke ,login is success! the port is:8762

异常情况（把login-service给停掉再访问）



这就说明当 login-service 工程不可用的时候，ribbon-service 调用 login-service 的API接口时，会执行快速失败，直接返回一组字符串，而不是等待响应超时，这很好的控制了容器的线程阻塞。

2.2、在Feign项目中使用Hystrix

2.2.1、准备

启动eureka-server项目，端口号是8761。

启动login-service项目，端口号是8762。

2.2.2、改造ribbon-server项目

Feign是自带断路器的，在D版本(*Dalston.RC1*)的Spring Cloud之后，它没有默认打开。需要在配置文件中配置打开它，在配置文件加以下代码：

```
server:
 port: 8765

spring:
 application:
 name: feign-service

eureka:
 client:
  serviceUrl:
  defaultZone: http://localhost:8761/eureka/

feign:
 hystrix:
 enabled: true
```

基于feign-service 工程进行改造，只需要在FeignClient的SchedualLoginService接口的注解中加上fallback的指定类就行了：

```
package xyz.jiangnanke.feignservice.service;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

/**
 * @Auther: zhengfeng
 * @Date: 2018\12\20 0020 15:49
 * @Description: 指定调用login-service服务
 */
@FeignClient(value = "login-service",
 fallback = SchedualLoginServiceHystrix.class)
public interface SchedualLoginService {
 /**
 * 调用login-service服务的login接口请求
 * @param name
 * @return
 */
 @RequestMapping(value = "/login",method = RequestMethod.GET)
 String loginOne(@RequestParam(value = "name") String name);
}
```

当然还得添加SchedualLoginServiceHystrix类，SchedualLoginServiceHystrix需要实现SchedualLoginService 接口，并注入到Ioc容器中，代码如下：

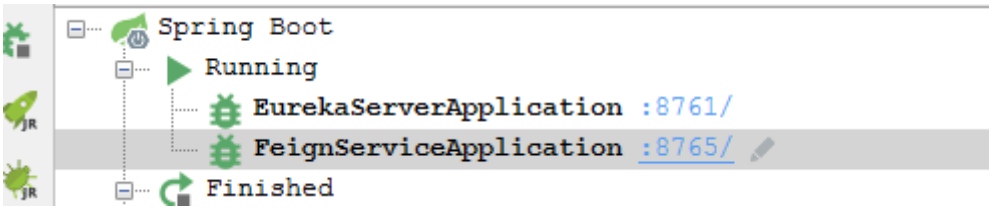
```
package xyz.jiangnanke.feignservice.service;

import org.springframework.stereotype.Component;

/**
 * @Auther: zhengfeng
 * @Date: 2018\12\21 0021 13:50
 * @Description:
 */
@Component
public class SchedualLoginServiceHystrix implements
SchedualLoginService{
```

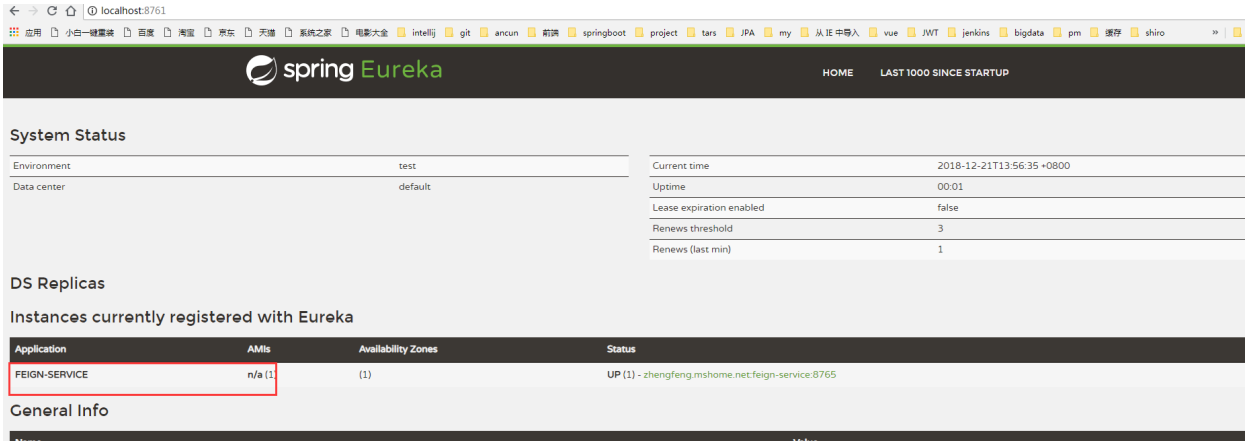
```
12
13  @Override
14  public String loginOne(String name) {
15  return "sorry " + name + ", you login is failed!";
16  }
17 }
```

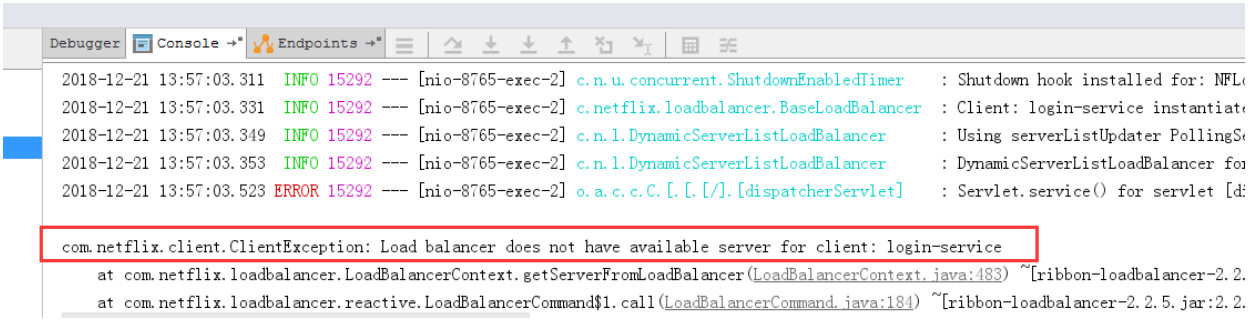
2.2.3、运行项目

OK，继续下去，就是运行项目，然后测试一下：

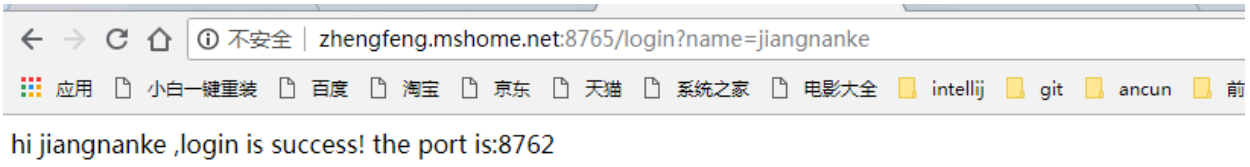
首先只运行了eureka-server项目和feign-service项目，没有运行login-service项目，得到的情况如图：



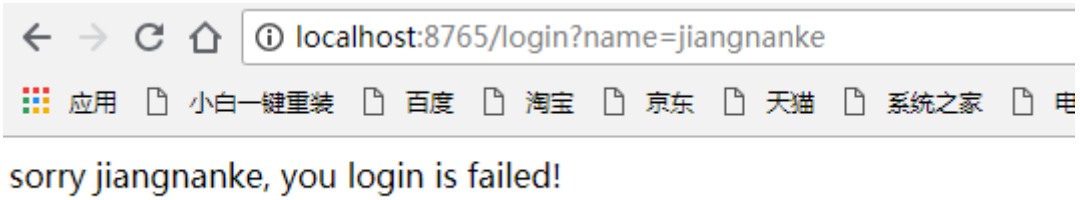




发现启动报错，还是使用原来的吧，都启动起来得到情况如图：



然后把login-service给停掉，得到情况如图：



这证明断路器起到作用了。

3、其他

在Hystrix中我们一般是用的默认配置，有些时候需要调整一些参数来获取更好的处理性能

配置官方文档：https://github.com/Netflix/Hystrix/wiki/Configuration

3.1、Execution相关的属性的配置：

- hystrix.command.default.execution.isolation.strategy 隔离策略，默认是Thread, 可选Thread | Semaphore
- thread 通过线程数量来限制并发请求数，可以提供额外的保护，但有一定的延迟。一般用于网络调用

semaphore 通过semaphore count来限制并发请求数，适用于无网络的高并发请求

- hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds 命令执行超时时间，默认1000ms
- hystrix.command.default.execution.timeout.enabled 执行是否启用超时，默认启用true
- hystrix.command.default.execution.isolation.thread.interruptOnTimeout 发生超时是是否中断，默认true
- hystrix.command.default.execution.isolation.semaphore.maxConcurrentRequests 最大并发请求数，默认10，该参数当使用ExecutionIsolationStrategy.SEMAPHORE策略时才有效。如果达到最大并发请求

数，请求会被拒绝。理论上选择semaphore size的原则和选择thread size一致，但选用semaphore时每次执行的单元要比较小且执行速度快（ms级别），否则的话应该用thread。

semaphore应该占整个容器（tomcat）的线程池的一小部分。

3.2、Fallback相关的属性

这些参数可以应用于Hystrix的THREAD和SEMAPHORE策略

- hystrix.command.default.fallback.isolation.semaphore.maxConcurrentRequests 如果并发数达到该设置值，请求会被拒绝和抛出异常并且fallback不会被调用。默认10
- hystrix.command.default.fallback.enabled 当执行失败或者请求被拒绝，是否会尝试调用hystrixCommand.getFallback() 。默认true

3.3、Circuit Breaker相关的属性

- hystrix.command.default.circuitBreaker.enabled 用来跟踪circuit的健康性，如果未达标则让request短路。默认true
- hystrix.command.default.circuitBreaker.requestVolumeThreshold 一个rolling window内最小的请求数。如果设为20，那么当一个rolling window的时间内（比如说1个rolling window是10秒）收到19个请求，即使19个请求都失败，也不会触发circuit break。默认20
- hystrix.command.default.circuitBreaker.sleepWindowInMilliseconds 触发短路的时间值，当该值设为5000时，则当触发circuit break后的5000毫秒内都会拒绝request，也就是5000毫秒后才会关闭circuit。默认5000
- hystrix.command.default.circuitBreaker.errorThresholdPercentage错误比率阀值，如果错误率>=该值，circuit会被打开，并短路所有请求触发fallback。默认50
- hystrix.command.default.circuitBreaker.forceOpen 强制打开熔断器，如果打开这个开关，那么拒绝所有request，默认false
- hystrix.command.default.circuitBreaker.forceClosed 强制关闭熔断器 如果这个开关打开，circuit将一直关闭且忽略circuitBreaker.errorThresholdPercentage

3.4、Metrics相关参数

- hystrix.command.default.metrics.rollingStats.timeInMilliseconds 设置统计的时间窗口值的，毫秒值，circuit break 的打开会根据1个rolling window的统计来计算。若rolling window被设为10000毫秒，则rolling window会被分成n个

buckets，每个bucket包含success，failure，timeout，rejection的次数的统计信息。默认10000

- hystrix.command.default.metrics.rollingStats.numBuckets 设置一个rolling window被划分的数量，若numBuckets = 10，rolling window = 10000，那么一个bucket的时间即1秒。必须符合rolling window % numberBuckets == 0。默认10
- hystrix.command.default.metrics.rollingPercentile.enabled 执行时是否enable指标的计算和跟踪，默认true
- hystrix.command.default.metrics.rollingPercentile.timeInMilliseconds 设置rolling percentile window的时间，默认60000
- hystrix.command.default.metrics.rollingPercentile.numBuckets 设置rolling percentile window的numberBuckets。逻辑同上。默认6
- hystrix.command.default.metrics.rollingPercentile.bucketSize 如果bucket size = 100，window = 10s，若这10s里有500次执行，只有最后100次执行会被统计到bucket里去。增加该值会增加内存开销以及排序的开销。默认100
- hystrix.command.default.metrics.healthSnapshot.intervalInMilliseconds 记录health 快照（用来统计成功和错误绿）的间隔，默认500ms

3.5、Request Context 相关参数

- hystrix.command.default.requestCache.enabled 默认true，需要重载getCacheKey()，返回null时不缓存
- hystrix.command.default.requestLog.enabled 记录日志到HystrixRequestLog，默认true

3.6、Collapser Properties 相关参数

- hystrix.collapser.default.maxRequestsInBatch 单次批处理的最大请求数，达到该数量触发批处理，默认Integer.MAX_VALUE
- hystrix.collapser.default.timerDelayInMilliseconds 触发批处理的延迟，也可以为创建批处理的时间 + 该值，默认10
- hystrix.collapser.default.requestCache.enabled 是否对HystrixCollapser.execute() and HystrixCollapser.queue()的cache，默认true

3.7、ThreadPool 相关参数

线程数默认值10适用于大部分情况（有时可以设置得更小），如果需要设置得更大，那有个基本得公式可以follow：

requests per second at peak when healthy × 99th percentile latency in seconds + some breathing room

每秒最大支撑的请求数 (99%平均响应时间 + 缓存值)

比如：每秒能处理1000个请求，99%的请求响应时间是60ms，那么公式是：
1000 *（0.060+0.012）
基本得原则时保持线程池尽可能小，他主要是为了释放压力，防止资源被阻塞。
当一切都是正常的时候，线程池一般仅会有1到2个线程激活来提供服务

- hystrix.threadpool.default.coreSize 并发执行的最大线程数，默认10
- hystrix.threadpool.default.maxQueueSize BlockingQueue的最大队列数，当设为 - 1，会使用SynchronousQueue，值为正时使用LinkedBlcokingQueue。该设置只会在初始化时有效，之后不能修改threadpool的queue size，除非reinitialising thread executor。默认 - 1。
- hystrix.threadpool.default.queueSizeRejectionThreshold 即使maxQueueSize没有达到，达到queueSizeRejectionThreshold该值后，请求也会被拒绝。因为maxQueueSize不能被动态修改，这个参数将允许我们动态设置该值。if maxQueueSize == -1，该字段将不起作用
- hystrix.threadpool.default.keepAliveTimeMinutes 如果corePoolSize和maxPoolSize设成一样（默认实现）该设置无效。如果通过plugin（https://github.com/Netflix/Hystrix/wiki/Plugins）使用自定义实现，该设置才有用，默认1.
- hystrix.threadpool.default.metrics.rollingStats.timeInMilliseconds 线程池统计指标的时间，默认10000
- hystrix.threadpool.default.metrics.rollingStats.numBuckets 将rolling window划分为n个buckets，默认10

四、使用Hystrix Dashboard

4.1、Hystrix Dashboard 介绍

Hystrix Dashboard是作为断路器状态的一个组件，提供了数据监控和友好的图形化界面。它主要用来实时监控Hystrix的各项指标信息。通过Hystrix Dashboard反馈的实时信息，可以帮助我们快速发现系统中存在的问题。下面通过一个例子来学习。

4.2、更改项目

4.2.1、添加依赖

pom.xml文件添加依赖如下：

```
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
 <groupId>org.springframework.cloud</groupId>
 <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
</dependency>
```

4.2.2、添加启动注解

在application启动类添加 @EnableHystrixDashboard 注解，达到启用Hystrix Dashboard 的功能。在启动类上添加@EnableCircuitBreaker 开启断路器功能，并且添加上对应的熔断访问属性。

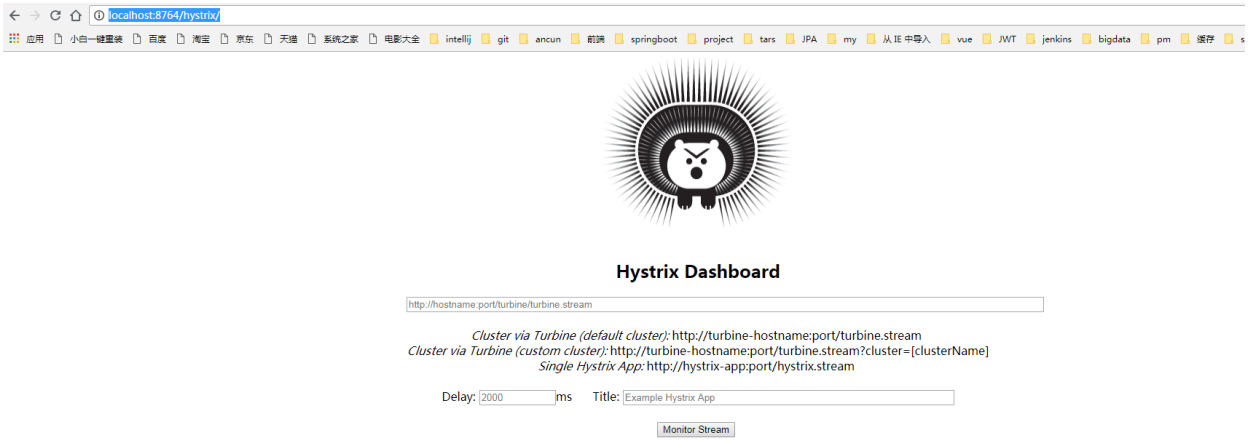
application.yml 添加内容如下：

```
management:
 endpoints:
 web:
 exposure:
 include: hystrix.stream
```

management.endpoints.web.exposure.include这个是用来暴露 endpoints 的。由于 endpoints 中会包含很多敏感信息，除了 health 和 info 两个支持 web 访问外，其他的默认不支持 web 访问。

4.2.3、启动运行

启动之后，访问：http://localhost:8764/hystrix/ ，得到效果如下：



通过 Hystrix Dashboard 主页面的文字介绍，我们可以知道，Hystrix Dashboard 共支持三种不同的监控方式：

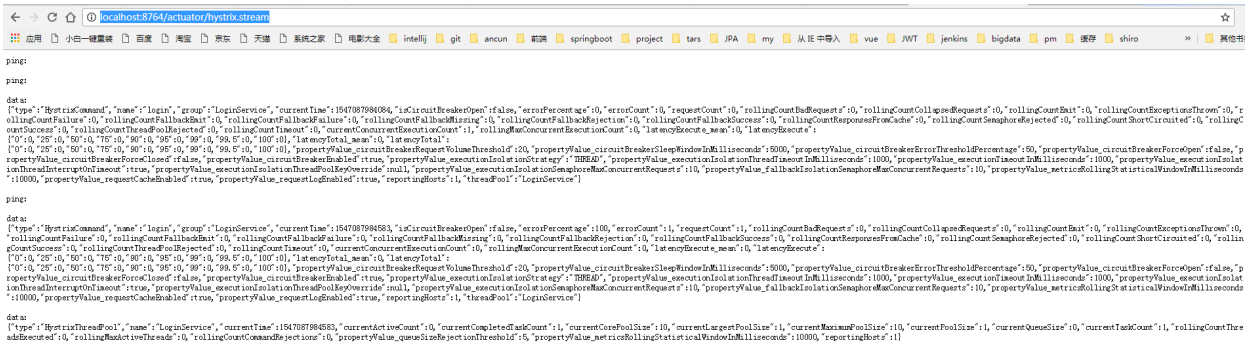
1. **默认的集群监控**：通过 URL：http://turbine-hostname:port/turbine.stream 开启，实现对默认集群的监控。
2. **指定的集群监控：**通过 URL：http://turbine-hostname:port/turbine.stream?cluster=[clusterName] 开启，实现对 clusterName 集群的监控。
3. **单体应用的监控：** 通过 URL：http://hystrix-app:port/hystrix.stream 开启 ，实现对具体某个服务实例的监控。（现在这里的 URL 应该为 http://hystrix-app:port/actuator/hystrix.stream，Actuator 2.x 以后endpoints 全部在/actuator下，可以通过management.endpoints.web.base-path修改）。

前两者都对集群的监控，需要整合 Turbine 才能实现。这一部分我们先实现对单体应用的监控，这里的单体应用就用使用 Hystrix 实现的服务提供者——microservicecloud-provider-dept-hystrix-8764。

页面上的另外两个参数：

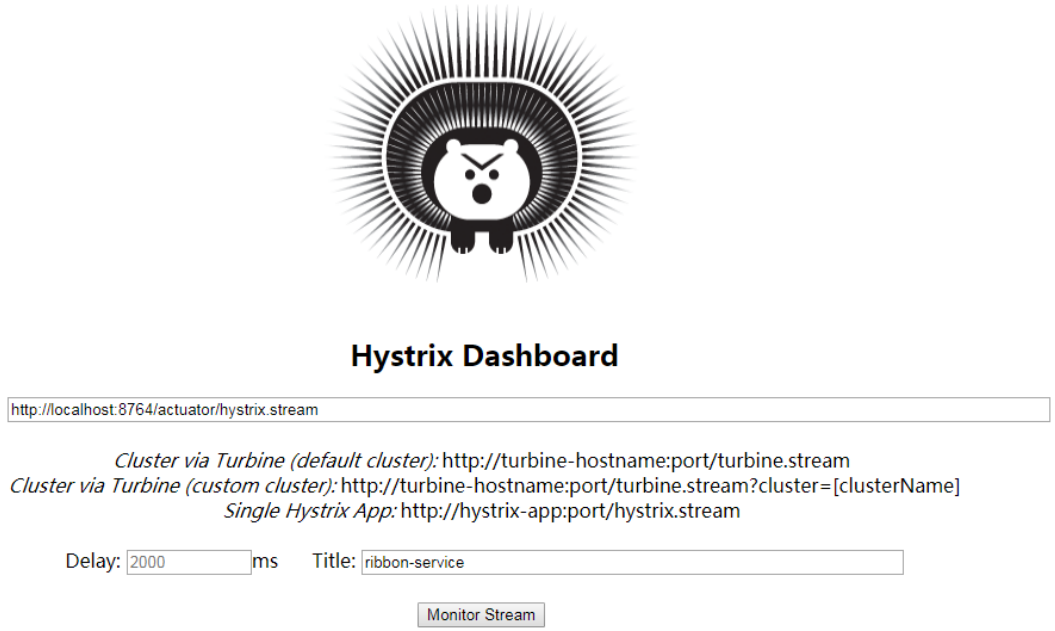
- **Delay：**控制服务器上轮询监控信息的延迟时间，默认为 2000 毫秒，可以通过配置该属性来降低客户端的网络和 CPU 消耗。
- **Title：**该参数可以展示合适的标题。

通过熔断机制访问：http://localhost:8764/actuator/hystrix.stream ，得到效果如下：



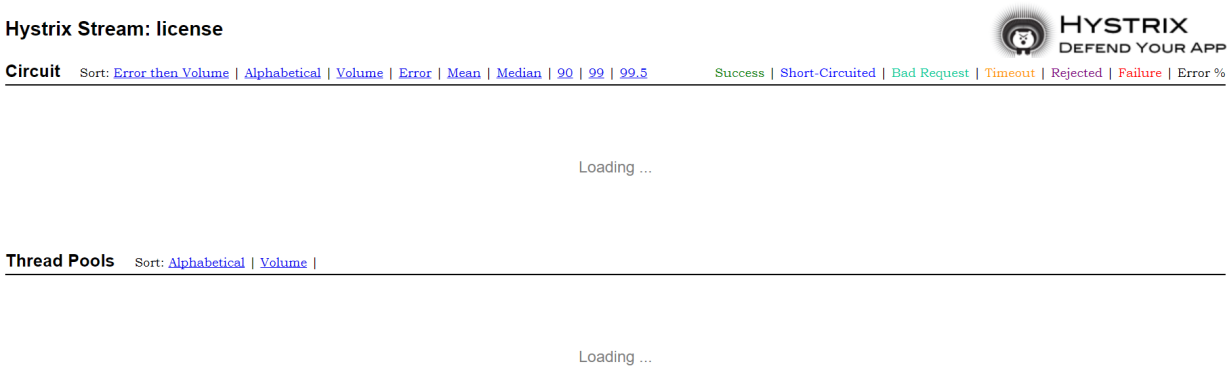
如果没有请求会一直显示 “Loading...” ，这时访问 http://localhost:8764/actuator/hystrix.stream 也是不断的显示 “ping” 。

这时候访问一下 http://localhost:8764/hystrix.stream ，

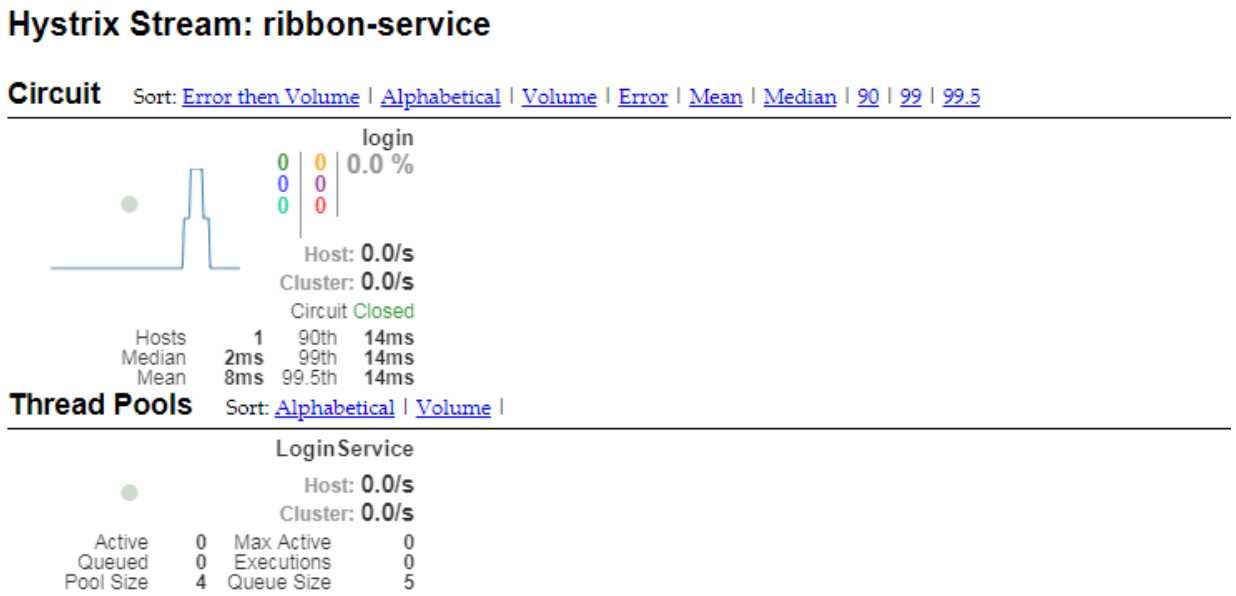


可以看到 Hystrix Dashboard 中出现了下面的效果

1、正常，但暂未执行过hystrix命令，会出现两个Loading...



2、正常，随便调用一个被hystrix管理的远程调用接口后，页面会刷新出类似如下的页面。



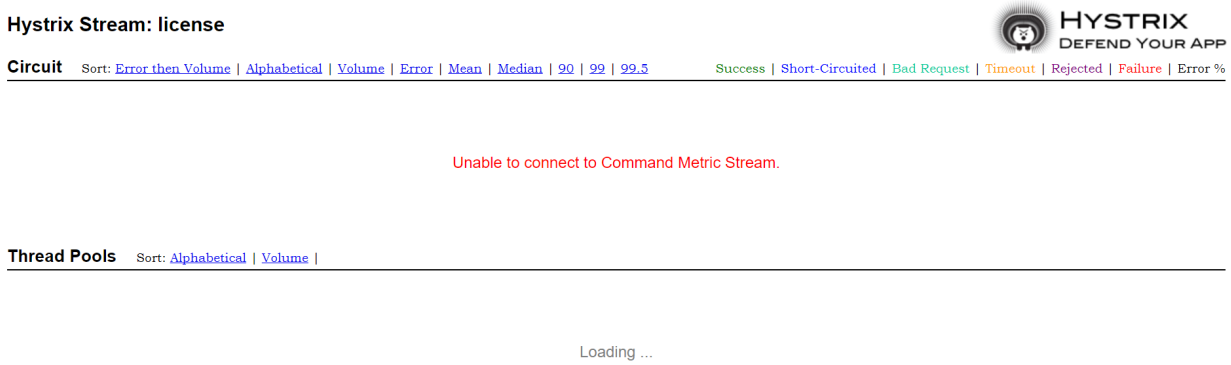
在监控的界面有两个重要的图形信息：一个实心圆和一条曲线。

·实心圆：1、通过颜色的变化代表了实例的健康程度，健康程度从绿色、黄色、橙色、红色递减。2、通过大小表示请求流量发生变化，流量越大该实心圆就越大。所以可以在大量的实例中快速发现故障实例和高压实例。

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

注意：当使用Hystrix Board来监控Spring Cloud Zuul构建的API网关时，Thread Pool信息会一直处于Loading状态。这是由于Zuul默认会使用信号量来实现隔离，只有通过Hystrix配置把隔离机制改成为线程池的方式才能够得以展示。

3、异常，目标服务没有引入spring-boot-starter-actuator启动依赖。



附：以上是在Ribbon里面使用的例子，在Feign负载均衡中使用Hystrix Dashboard 注意：一定需要配置打开断路器

```
feign.hystrix.enabled=true
```

4.3、使用Turbine

官网介绍：

与上面Dashload有区别的就是

1、在依赖上多添加一个turbine的依赖包，如下：

```
<dependency>
 <groupId>org.springframework.cloud</groupId>
 <artifactId>spring-cloud-starter-netflix-turbine</artifactId>
</dependency>
```

2、启动类上多加一个注解：@EnableTurbine，表示使用 Turbine

3、application.yml配置文件多添加turbine的 属性，如下：

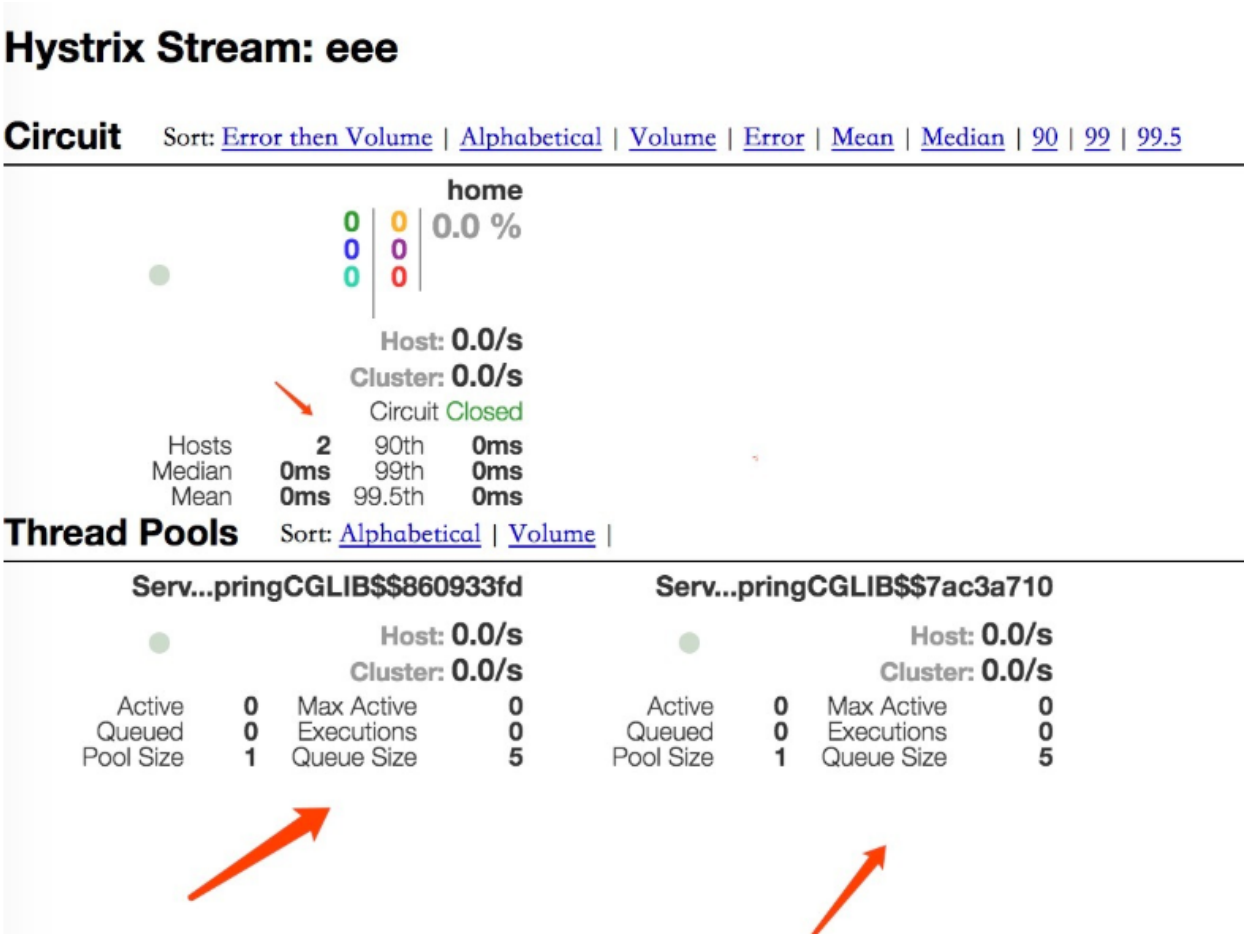
```
management:
 endpoints:
 web:
 exposure:
 include: "*"
 cors:
 allowed-origins: "*"
 allowed-methods: "*"

turbine:
 app-config: service-hi,service-lucy
 aggregator:
```

```
clusterConfig: default
clusterNameExpression: new String("default")
combine-host: true
instanceUrlSuffix:
default: actuator/hystrix.stream

```

4、访问的时候，用的是turbine.stream ，如：

http://localhost:8764/turbine.stream

打开http://localhost:8764/hystrix/ 可以访问得到多个服务，



参考资料

配置官方文档：https://github.com/Netflix/Hystrix/wiki/Configuration
https://github.com/Netflix/Hystrix/wiki
https://github.com/Netflix/Hystrix/wiki/How-it-Works
https://github.com/Netflix/Hystrix/wiki/How-To-Use
http://netflix.github.io/Hystrix/javadoc/
https://springcloud.cc/
https://github.com/Netflix/hystrix
http://cloud.spring.io/spring-cloud-static/Finchley.RELEASE/single/spring-cloud.html
http://cxytiandi.com/blog/detail/13331
https://blog.csdn.net/forezp/article/details/81041113
https://blog.csdn.net/u013739073/article/details/80627087