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？

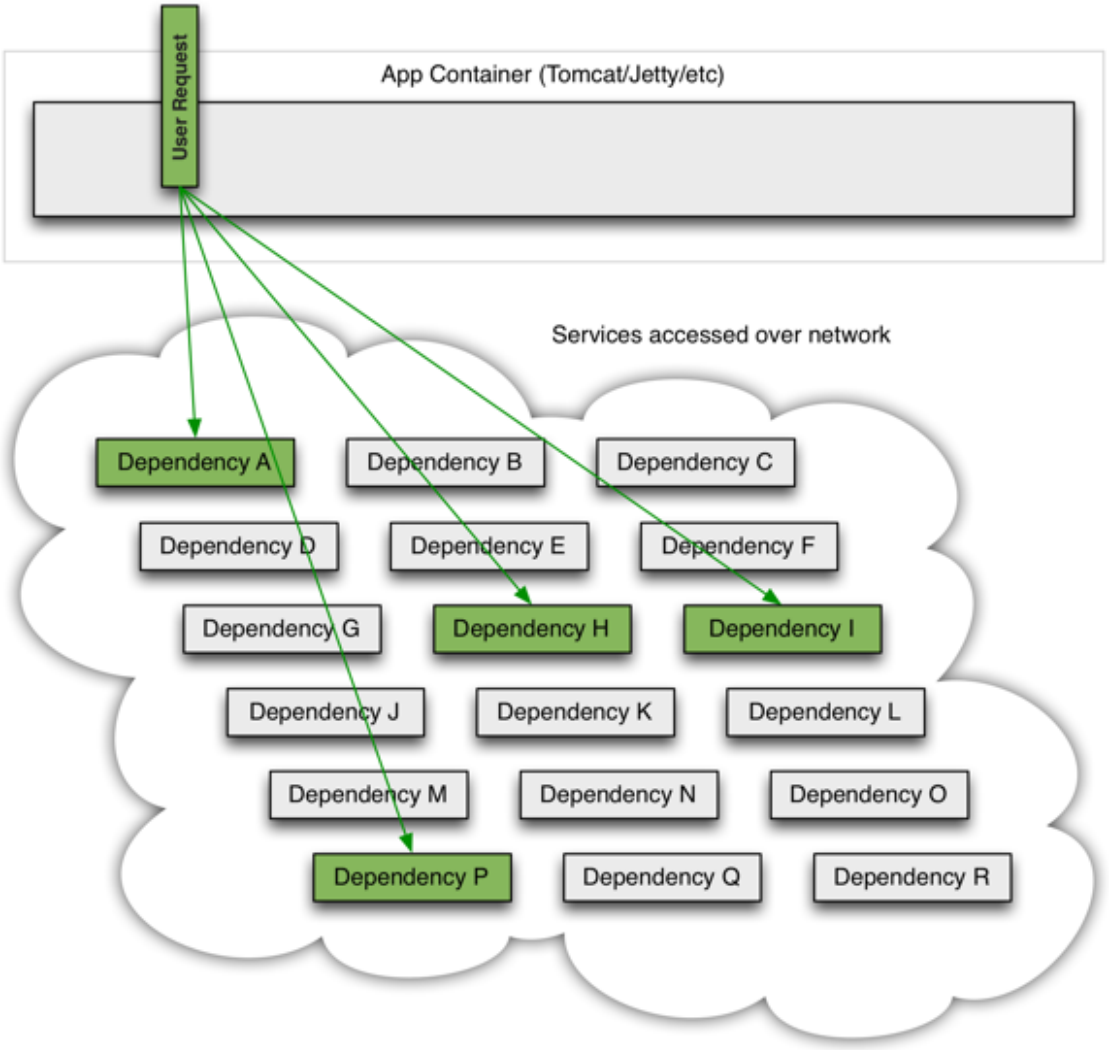
Hystrix旨在执行以下操作：

- 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。

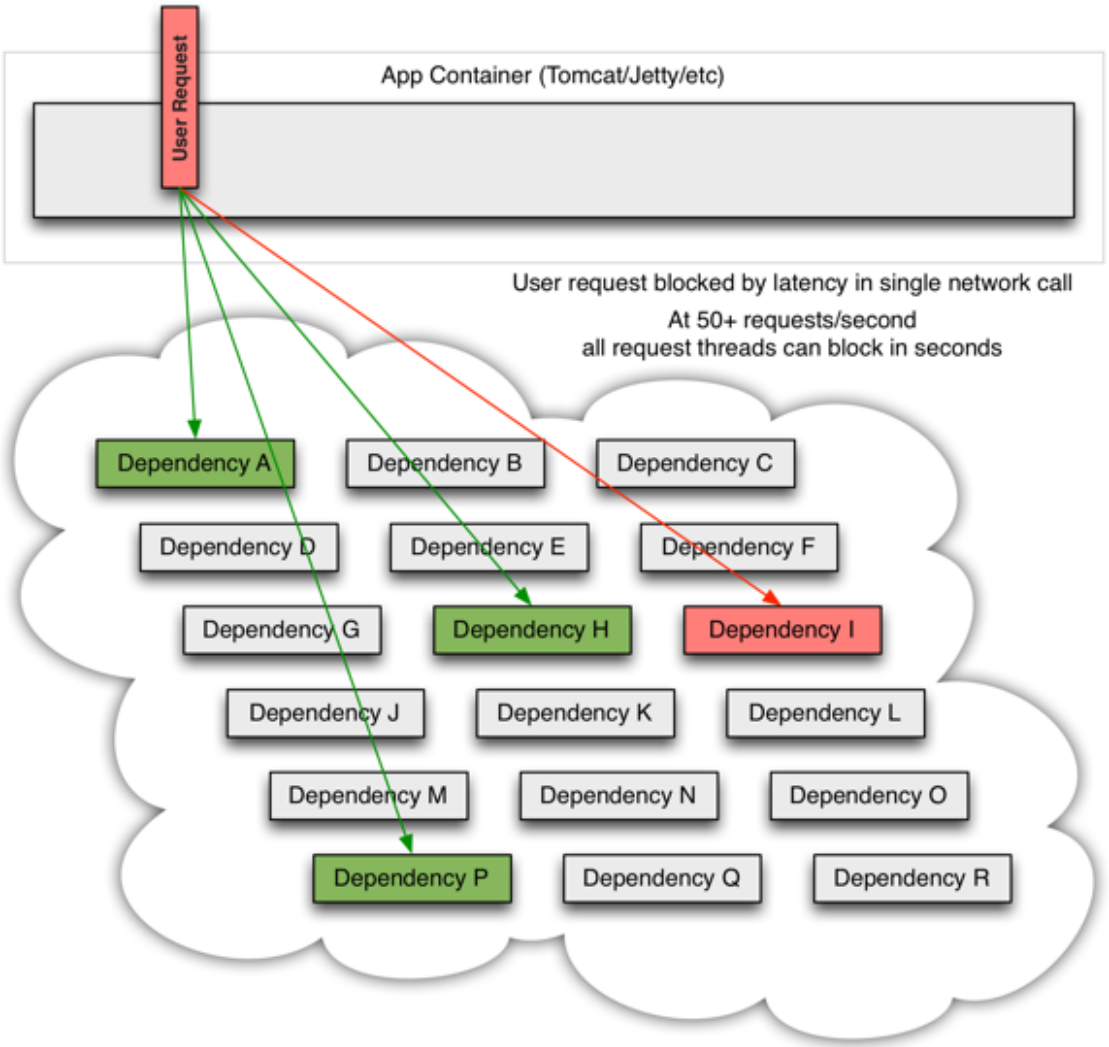
- 实现近实时监控，警报和操作控制。

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：

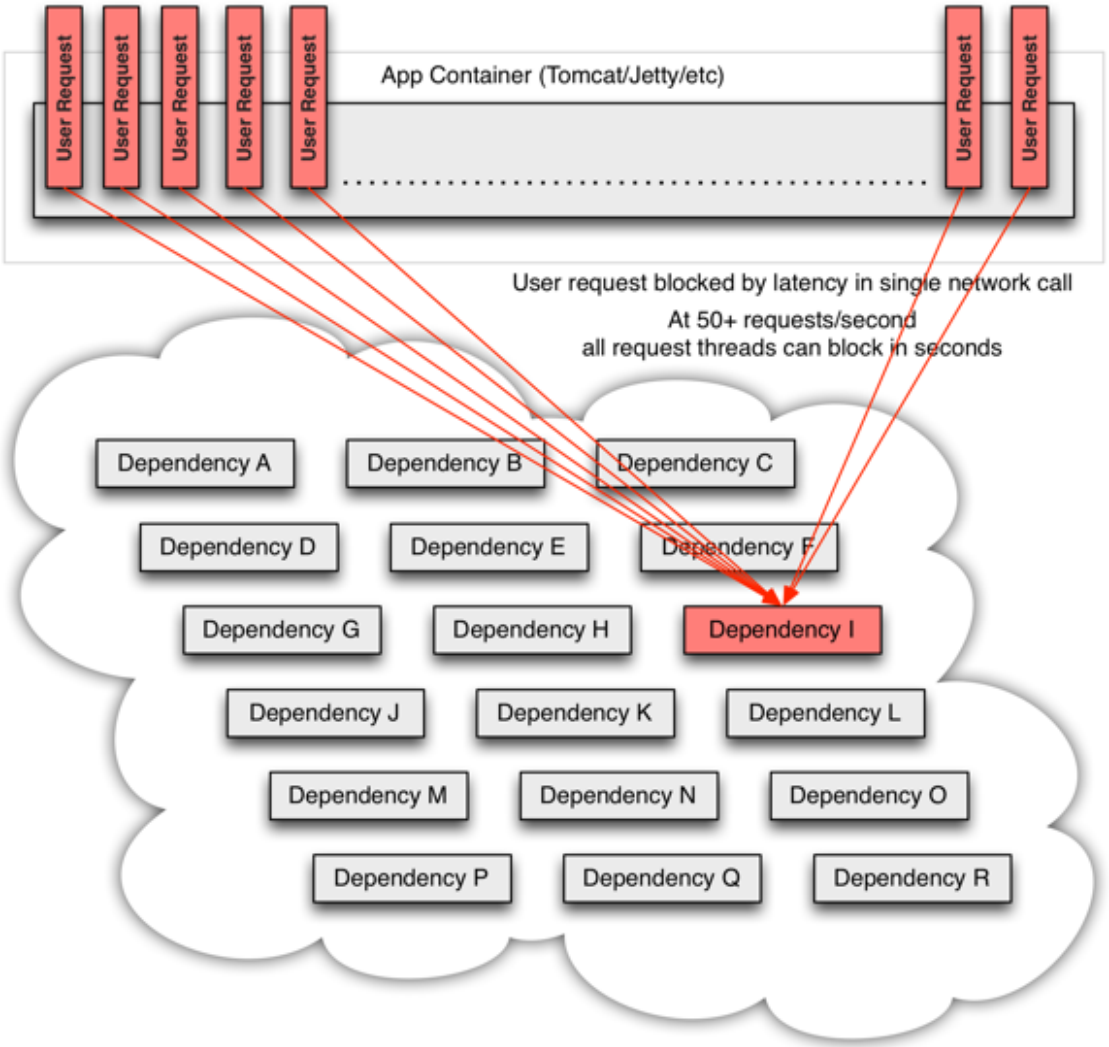


当许多后端系统中的一个变得失效时，它可以阻止整个用户请求：



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。

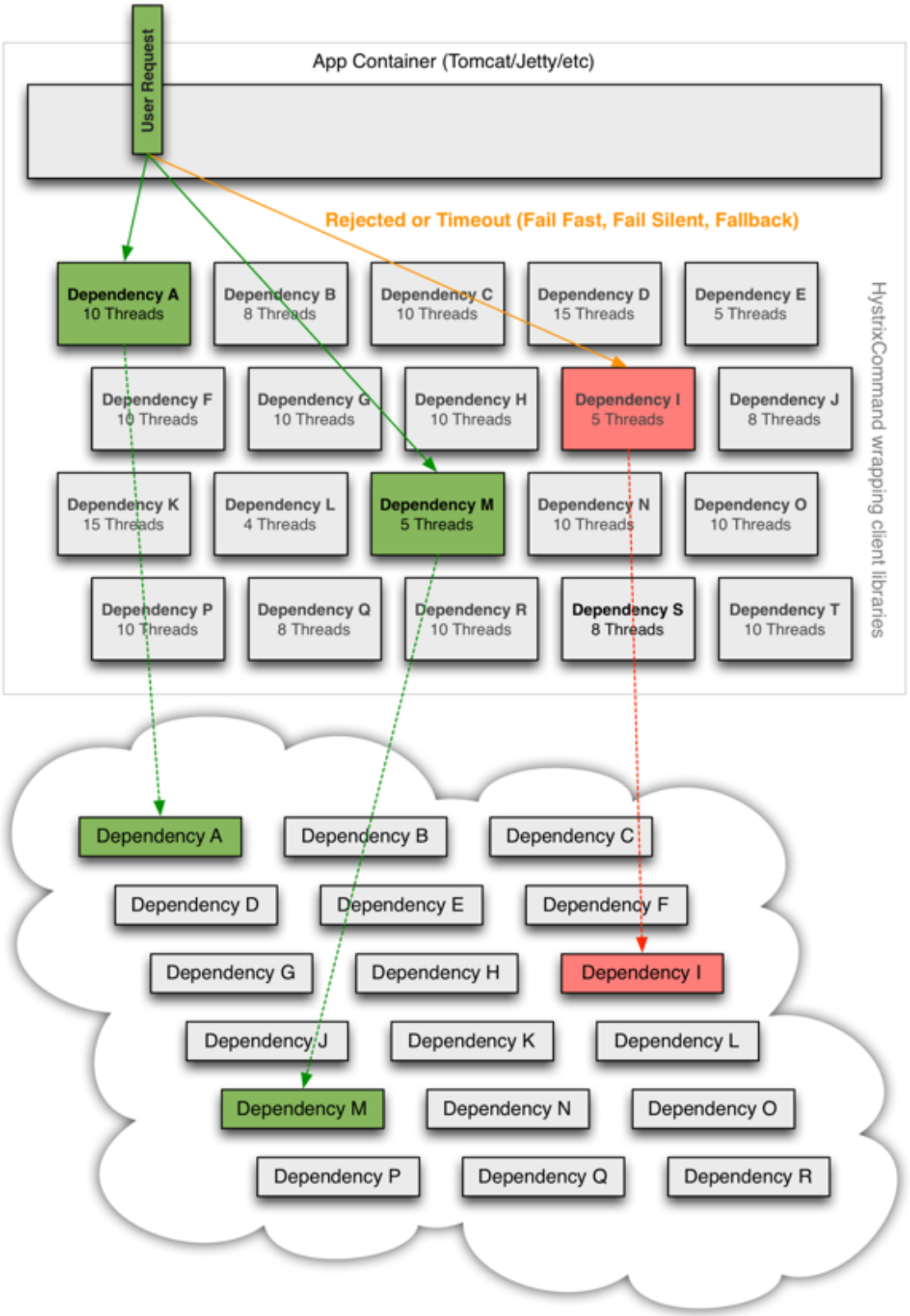
Hystrix通过以下方式实现：

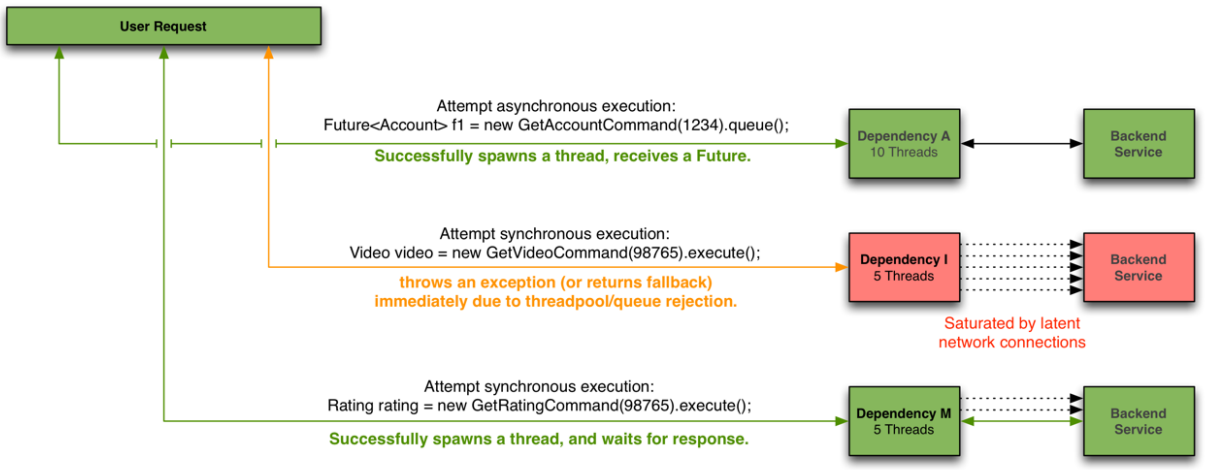
- 将对外部系统（或“依赖项”）的所有调用包含在通常在单独线程内执行的对象HystrixCommand或HystrixObservableCommand对象中（这是命令模式的示例）。
- 定时调用的时间超过您定义的阈值。有一个默认的，而是由“属性”的方式对大多数依赖你自定义设置这些超时，使它们比测量的99.5略高日每个依存性百分性能。

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();
 }

}

```

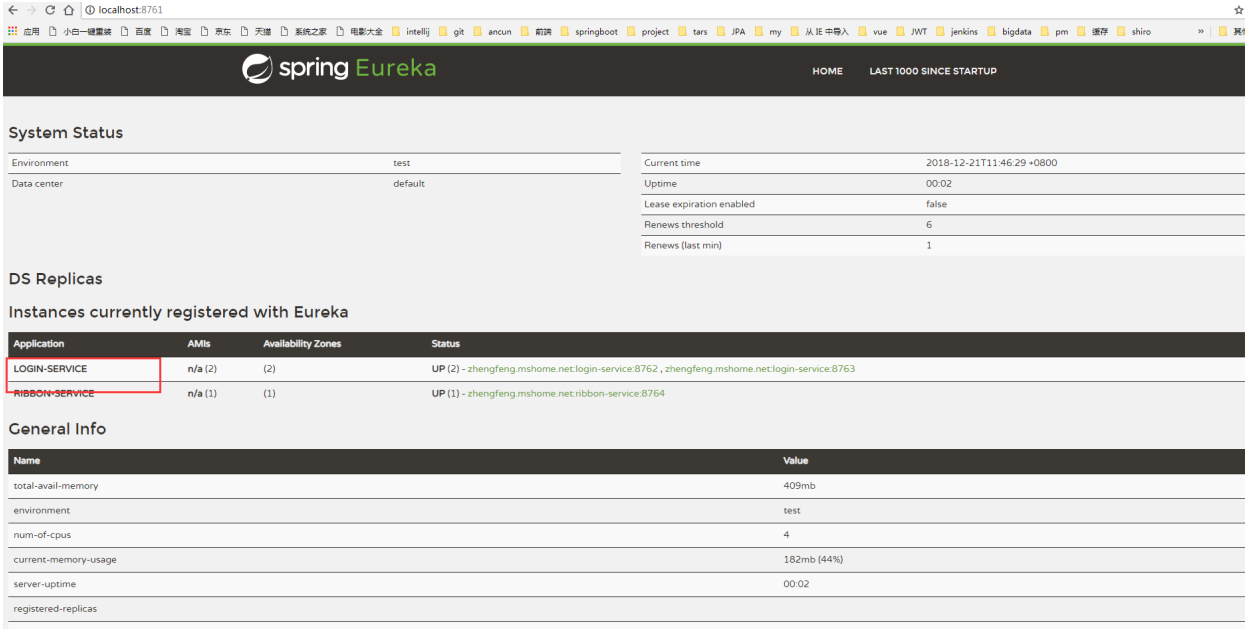
接下来就是修改熔断逻辑，即LoginService.java类了，在login(String name)方法上加上@HystrixCommand注解。该注解对该方法创建了熔断器的功能，并指定了fallbackMethod熔断方法loginError，熔断方法直接返回了一个字符串，字符串为： "hi " + name + ",I am very Sorry, you login is Error!"，代码如下：

```
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
```

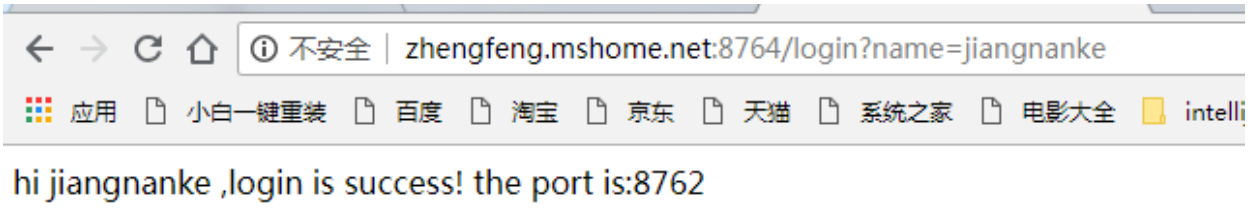
```java
public class LoginService {

 @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：

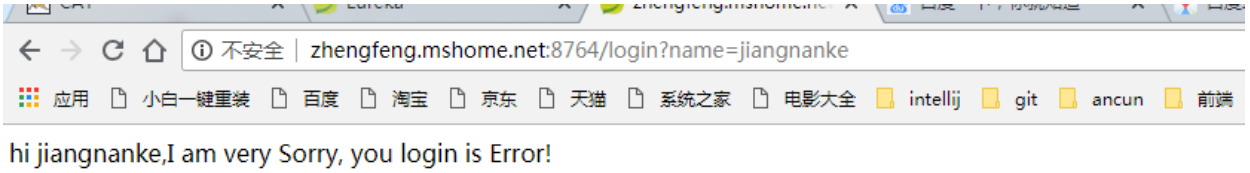


正常情况：



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：

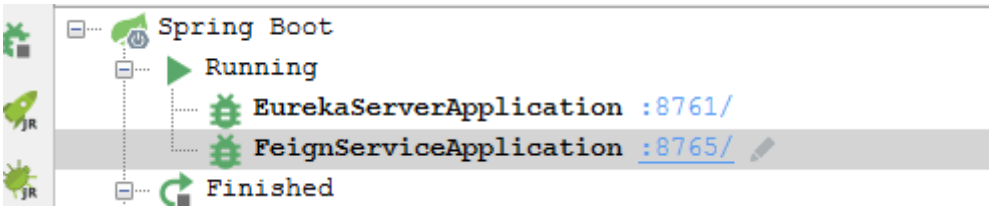
```
package xyz.jiangnanke.feignservice.service;

import org.springframework.stereotype.Component;

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

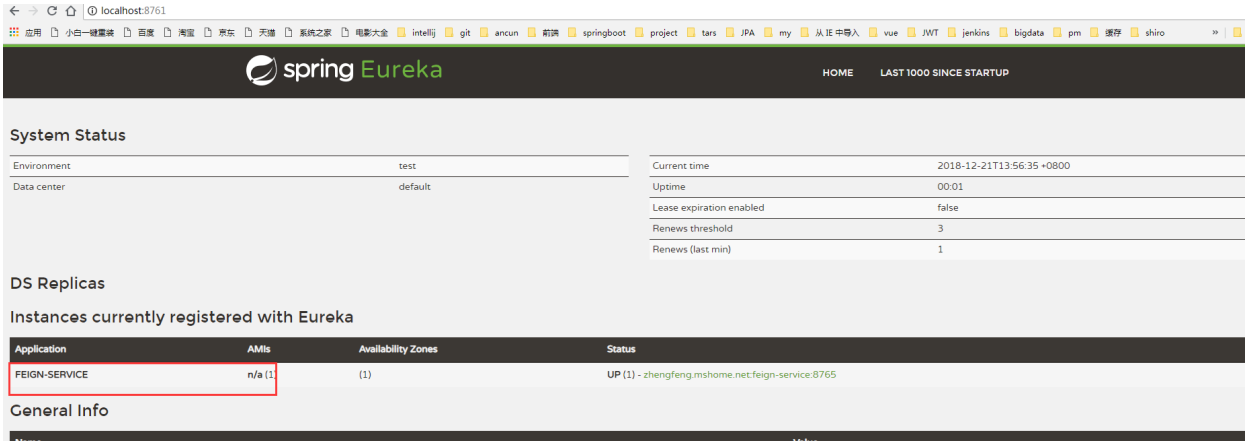
```
12
13  @Override
14  public String loginOne(String name) {
15  return "sorry " + name + ", you login is failed!";
16  }
17 }
```

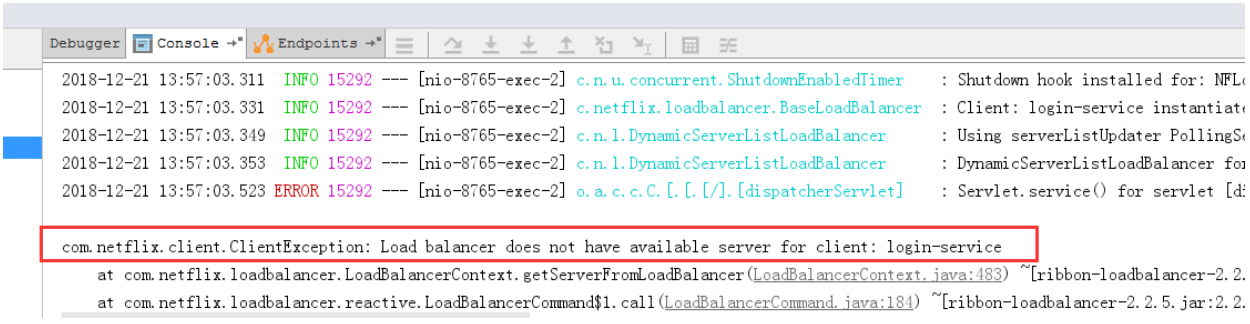
2.2.3、运行项目

OK，继续下去，就是运行项目，然后测试一下：

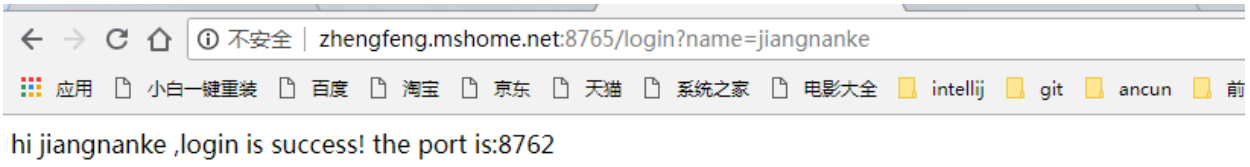
首先只运行了eureka-server项目和feign-service项目，没有运行login-service项目，得到的情况如图：



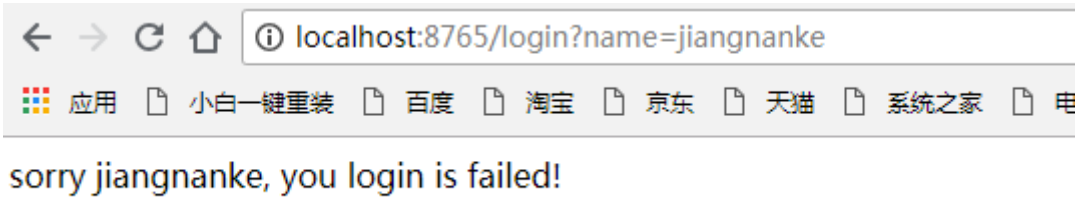




发现启动报错，还是使用原来的吧，都启动起来得到情况如图：



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。

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