Hystrix

一 Hystrix介绍

1.1 Hystrix是什么

在分布式环境中，许多服务依赖项中的一些必然会失败,或者我们的服务被突发性的高并发访问导致出现问题。Hystrix是一个库，通过添加延迟容忍和容错逻辑，帮助你控制这些分布式服务之间的交互。Hystrix通过隔离服务之间的访问点、停止级联失败和提供回退选项来实现这一点，所有这些都可以提高系统的整体弹性。

https://github.com/Netflix/Hystrix/wiki

1.2 Hystrix为了什么

Hystrix被设计的目标是：

1. 对通过第三方客户端库访问的依赖项（通常是通过网络）的延迟和故障进行保护和控制。
2. 在复杂的分布式系统中阻止级联故障。
3. 快速失败，快速恢复。
4. 回退，尽可能优雅地降级。
5. 启用近实时监控、警报和操作控制。

1.3 Hystrix解决了什么问题

复杂分布式体系结构中的应用程序有许多依赖项，每个依赖项在某些时候都不可避免地会失败。如果主机应用程序没有与这些外部故障隔离，那么它有可能被他们拖垮。

例如，对于一个依赖于30个服务的应用程序，每个服务都有99.99%的正常运行时间，你可以期望如下：

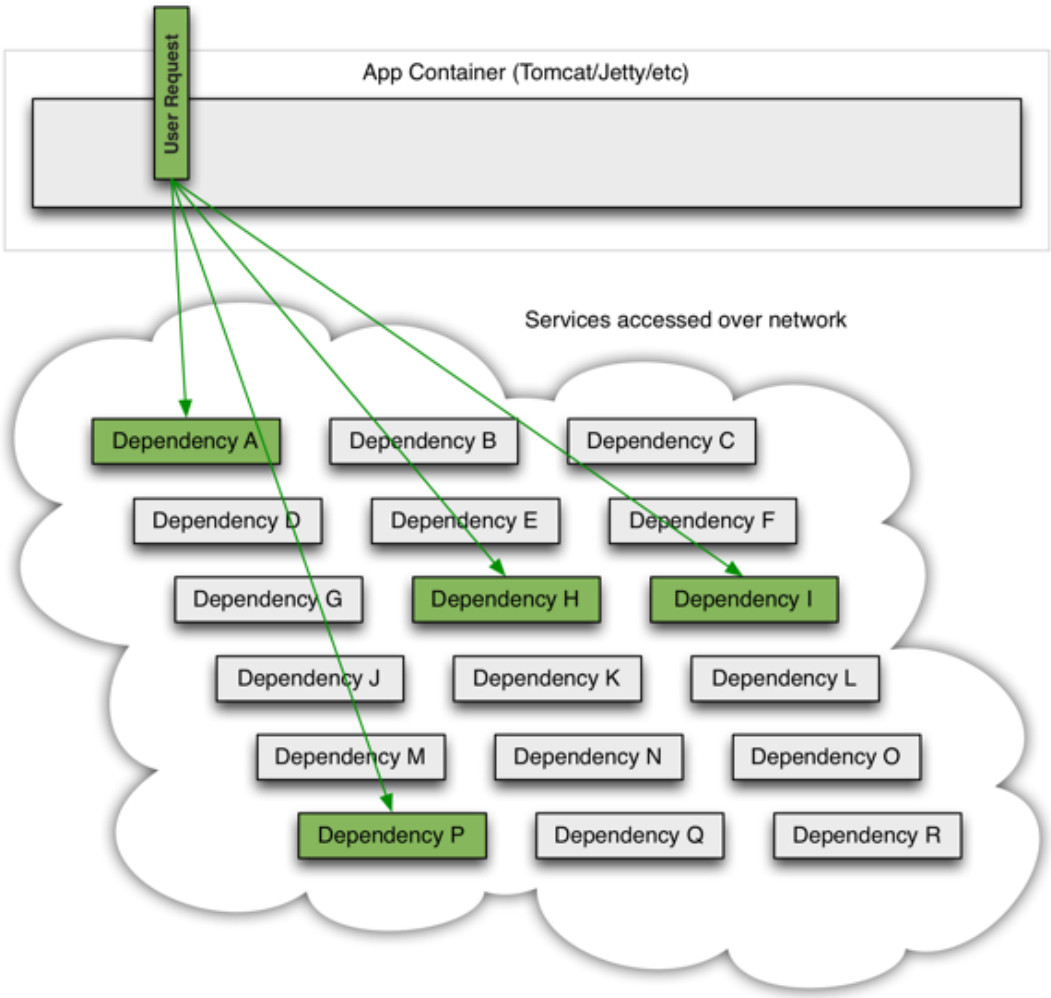
99.99*30 = 99.7% 可用

也就是说一亿个请求的0.03% = 3000000 会失败

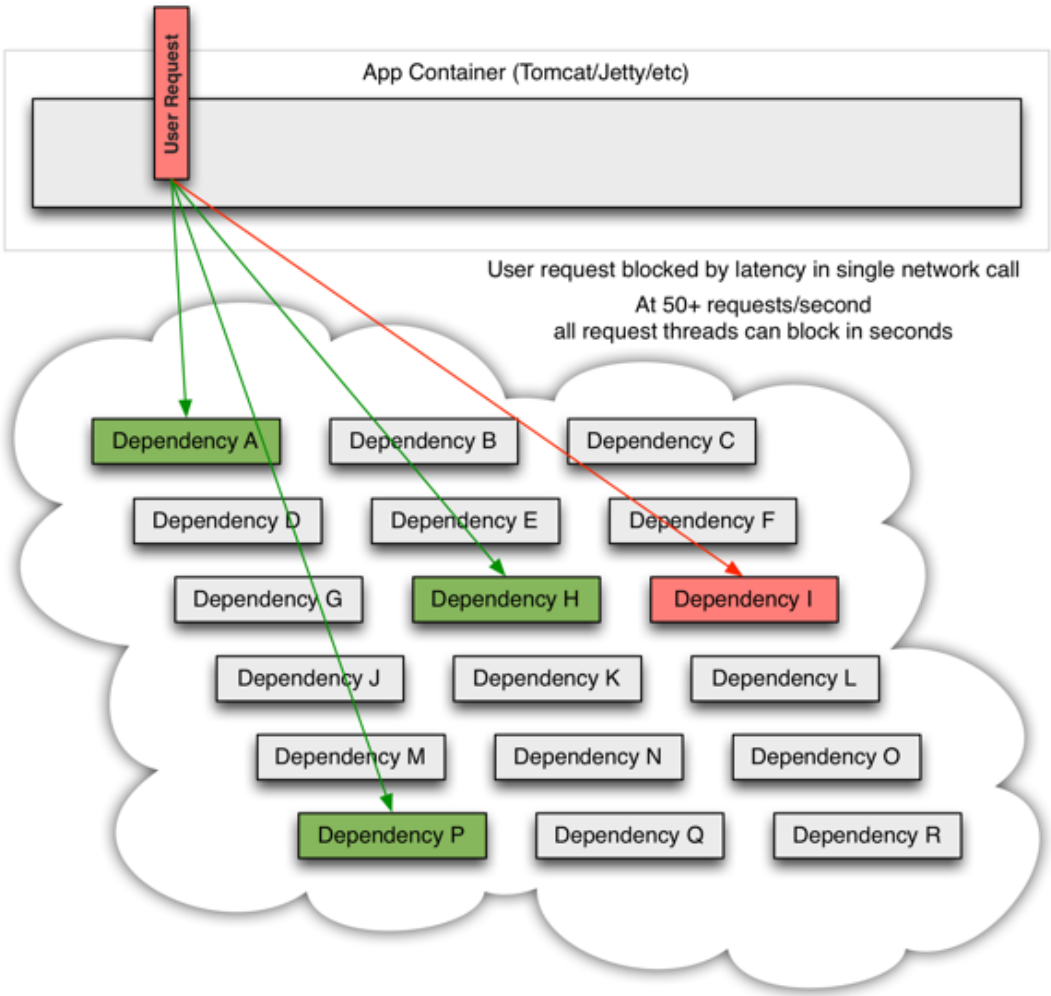
如果一切正常，那么每个月有2个小时服务是不可用的

现实通常是更糟糕

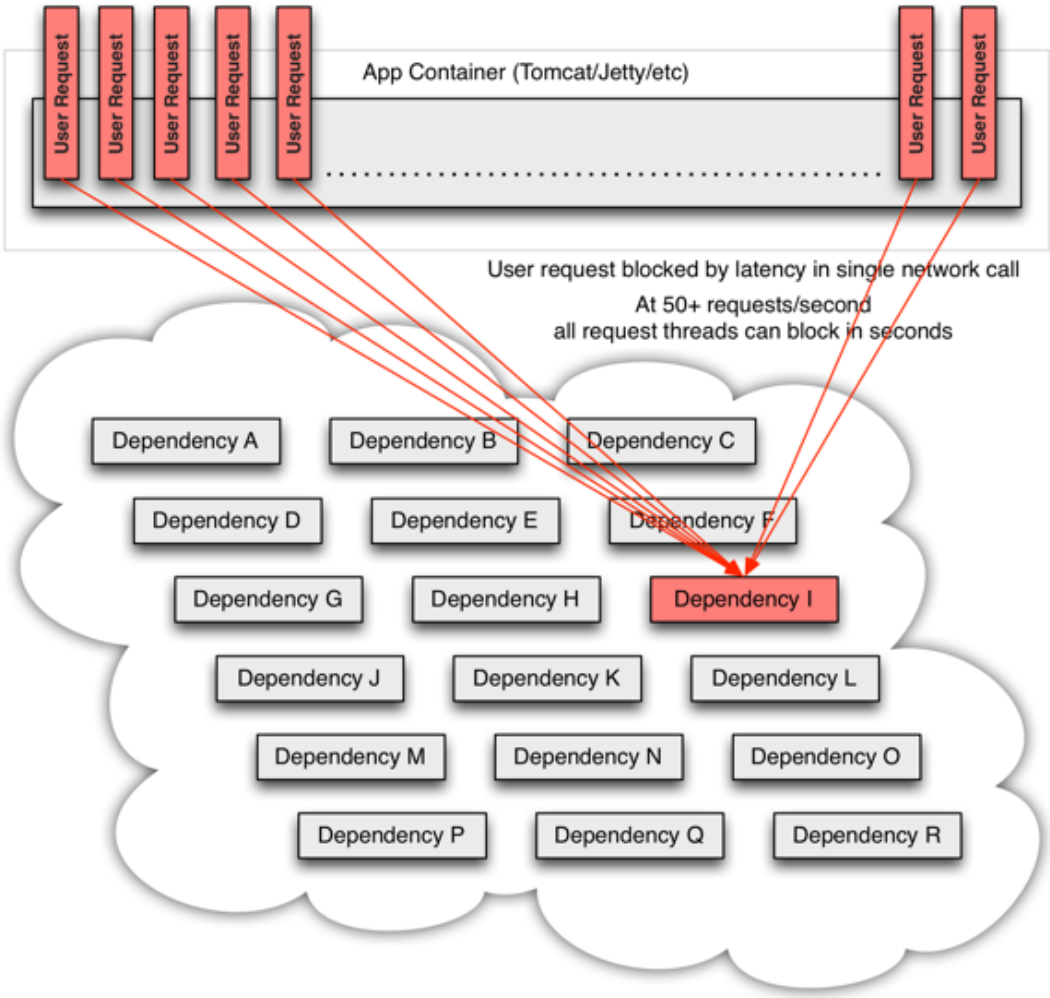
当一切正常时，请求看起来是这样的:



当其中有一个系统有延迟时，它可能阻塞整个用户请求:



在高流量的情况下，一个后端依赖项的延迟可能导致所有服务器上的所有资源在数秒内饱和（PS：意味着后续再有请求将无法立即提供服务）



1.4 Hystrix设计原则是什么

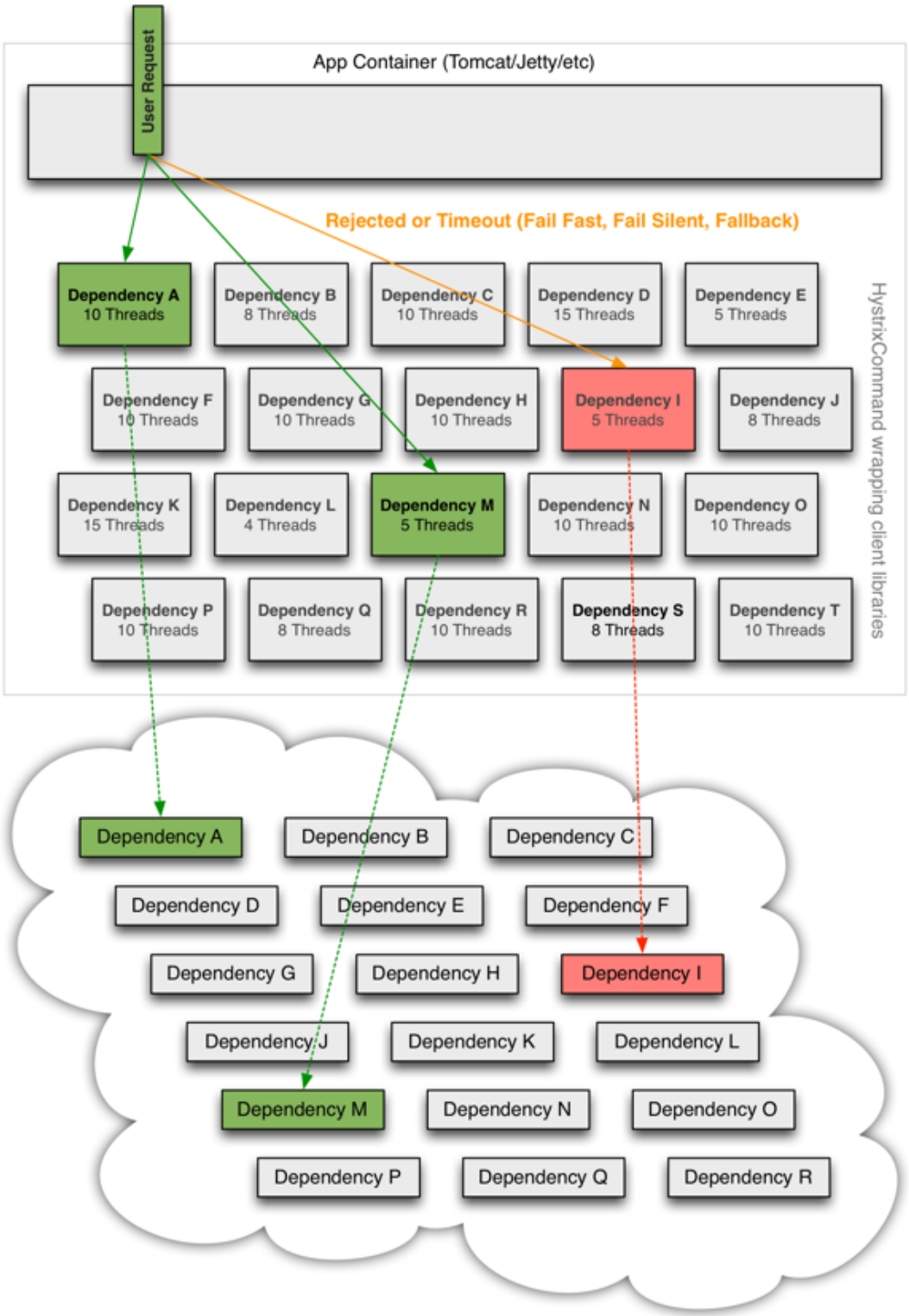
- 防止任何单个依赖项耗尽所有容器（如Tomcat）用户线程。
- 甩掉包袱，快速失败而不是排队。
- 在任何可行的地方提供回退，以保护用户不受失败的影响。
- 使用隔离技术（如隔离板、泳道和断路器模式）来限制任何一个依赖项的影响。
- 通过近实时的度量、监视和警报来优化发现时间。
- 通过配置的低延迟传播来优化恢复时间。
- 支持对Hystrix的大多数方面的动态属性更改，允许使用低延迟反馈循环进行实时操作修改。
- 避免在整个依赖客户端执行中出现故障，而不仅仅是在网络流量中。

1.5 Hystrix是如何实现它的目标的

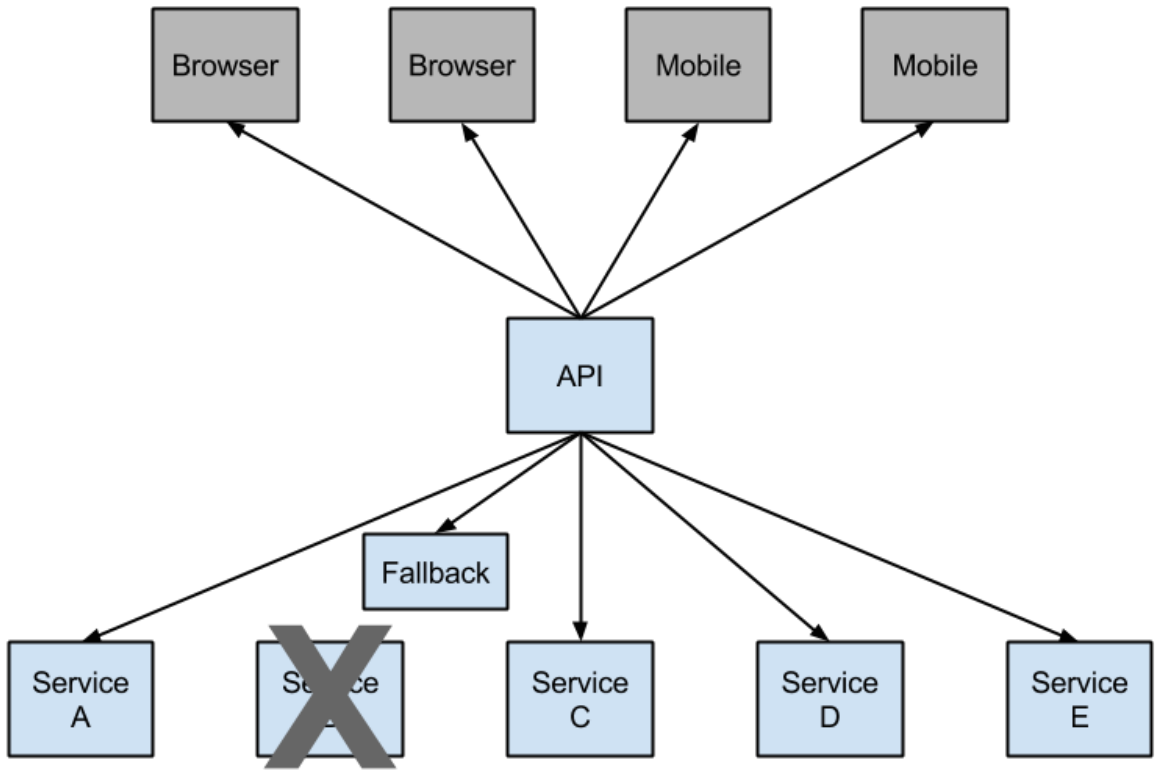
1. 用一个HystrixCommand 或者 HystrixObservableCommand （这是命令模式的一个例子）包装所有的对外部系统（或者依赖）的调用，典型地它们在一个单独的线程中执行
2. 调用超时时间比你自己定义的阈值要长。有一个默认值，对于大多数的依赖项你是可以自定义超时时间的。
3. 为每个依赖项维护一个小的线程池(或信号量)；如果线程池满了，那么该依赖性将会立即拒绝请求，而不是排队。
4. 调用的结果有这么几种：成功、失败（客户端抛出异常）、超时、拒绝。
5. 在一段时间内，如果服务的错误百分比超过了一个阈值，就会触发一个断路器来停止对特定服务的所有请求，无论是手动的还是自动的。
6. 当请求失败、被拒绝、超时或短路时，执行回退逻辑(返回一些托底数据)。
7. 近实时监控指标和配置变化。

当你使用Hystrix来包装每个依赖项时，上图中所示的架构会发生变化，如下图所示：

> 每个依赖项相互隔离，当延迟发生时，它会被限制在资源中，并包含回退逻辑，该逻辑决定在依赖项中发生任何类型的故障时应作出何种响应：



如下图,当我们访问的服务B出现问题的时候,不能影响其他的访问



二 Hystrix使用

在SpringCloud中使用Hystrix是非常简单的事情,SpringCloud对Hystrix进行了包装,让我们可以方便的使用,只需要通过注解来指定当我们服务出现问题时候需要执行的返回托底数据的方法即可

```
参考12consumer-eureka-hystrix
```

2.1 POM 文件

```
                    <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
            </dependency>
<!--从注册中心中获取服务列表-->
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
            </dependency>
<!--hystrix 依赖-->
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
            </dependency>
```

2.2 application.yml

```yaml
server:
  port: 9000
eureka: #注册中心的地址
  client:
    service-url:
      defaultZone: http://zhangsan:abc@localhost:10000/eureka #curl风格
spring:
  application:
    name: 12consumer-eureka-hystrix
```

2.3 controller

```
import com.netflix.appinfo.InstanceInfo;
import com.netflix.discovery.EurekaClient;
import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
import com.netflix.hystrix.contrib.javanica.annotation.HystrixProperty;
import com.qianfeng.microservice.consumer.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;

/**
 * Created by jackiechan on 19-1-8/下午12:00
 *
 * @Author jackiechan
 */
@RestController
@RequestMapping("/userconsumer")
public class UserController {
    @Autowired
    private RestTemplate template;

    @Autowired
    private EurekaClient eurekaClient; //这个对象spring会自动创建不需要我们额外创建

    @GetMapping("/info/{id}")
  //开启熔断降级
    @HystrixCommand(fallbackMethod = "abc") //给当前方法设置失败后的操作,失败后到底做什么 fallbackMethod指定一个方法名,当出现问题的时候会去执行这个方法,要求这个方法的参数和返回值必须和当前方法一致
    public User getUserById(@PathVariable Long id) throws Exception {
        System.out.println("出错的线程" + Thread.currentThread().getName());
        InstanceInfo instanceInfo = eurekaClient.getNextServerFromEureka("04PROVIDER-EUREKA", false);//cong euerka上获取指定名字的服务的信息
        String url = instanceInfo.getHomePageUrl();//获取服务的地址 也就是 类似于 http://localhost:8000
        System.out.println("服务提供者的地址:=======>"+url);
        User user = template.getForObject(url+"/user/info/" + id, User.class);//请求指定地址并将返回的json字符串转换为User对象
        return user;
    }

    @PostMapping("/save")
    public String addUser(@RequestBody User user) {
```

```
        String result = template.postForObject("http://localhost:8000/user/save/",
user, String.class);//post方式请求这个地址,并将user作为参数传递过去(json格式),并将返回结果转换为String类型
        return result;

    }


    public User abc(Long id){
        User user = new User();
        user.setId(-100);
        user.setUsername("刚才失败传递的id是:==>" + id);
        return user;
    }
}
```

2.4 主程序

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

/**
 * Created by jackiechan on 19-1-8/下午12:04
 *
 * @Author jackiechan
 */
@SpringBootApplication
@EnableEurekaClient
@EnableCircuitBreaker//开启熔断机制
public class ConsumerStarter {


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

    @Bean
    public RestTemplate restTemplate() {
        RestTemplate restTemplate = new RestTemplate();
        return restTemplate;
    }
}
```

2.5 测试

> 启动测试,当我们04PROVIDER-EUREKA启动的时候能返回正常数据,当04PROVIDER-EUREKA关闭后再次访问会返回我们指定当服务

三 修改 Hystrix 配置

> Hystrix 内部是通过线程池或者信号量的方式来进行包装的,当我们选择使用线程池的时候,下游服务的调用会被放到 hystrix 内部的线程池进行调用,当使用信号量的时候,下游服务的调用会和当前请求(如用户的请求线程)在一起
>
> https://github.com/Netflix/Hystrix/wiki/Configuration
>
> 属性参见 目录四

3.1 修改配置

> 我们只需要在注解中添加一个属性即可指定调用方式为线程池或者信号量

```
import com.netflix.appinfo.InstanceInfo;
import com.netflix.discovery.EurekaClient;
import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
import com.netflix.hystrix.contrib.javanica.annotation.HystrixProperty;
import com.qianfeng.microservice.consumer.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;

/**
 * Created by jackiechan on 19-1-8/下午12:00
 *
 * @Author jackiechan
 */
@RestController
@RequestMapping("/userconsumer")
public class UserController {
    @Autowired
    private RestTemplate template;

    @Autowired
    private EurekaClient eurekaClient; //这个对象spring会自动创建不需要我们额外创建

    @GetMapping("/info/{id}")
    //execution.isolation.strategy 上下文的传播策略 有多线程和信号量两个机制,多线程指的是我们调用04PROVIDER-EUREKA的时候会和当前方法所在的线程不一样,信号量指的是当前方法的调用线程和04PROVIDER-EUREKA服务的调用在一个线程
    @HystrixCommand(fallbackMethod = "abc",commandProperties = {@HystrixProperty(name = "execution.isolation.strategy",value = "SEMAPHORE")}) //给当前方法设置失败后的操作,失败后到底做什么fallbackMethod指定一个方法名,当出现问题的时候会去执行这个方法,要求这个方法的参数和返回值必须和当前方法一致
    public User getUserById(@PathVariable Long id) throws Exception {
        System.out.println("出错的线程" + Thread.currentThread().getName());
        InstanceInfo instanceInfo = eurekaClient.getNextServerFromEureka("04PROVIDER-EUREKA", false);//cong euerka上获取指定名字的服务的信息
        String url = instanceInfo.getHomePageUrl();//获取服务的地址 也就是 类似于 http://localhost:8000
        System.out.println("服务提供者的地址:=======>"+url);
        User user = template.getForObject(url+"/user/info/" + id, User.class);//请求指定地址并将返回的json字符串转换为User对象
```

```
        return user;
    }
    @PostMapping("/save")
    public String addUser(@RequestBody User user) {
        String result = template.postForObject("http://localhost:8000/user/save/",
user, String.class);//post方式请求这个地址,并将user作为参数传递过去(json格式),并将返回结果转换为String类型
        return result;

    }

    public User abc(Long id){
        System.out.println("降级的线程" + Thread.currentThread().getName());
        User user = new User();
        user.setId(-100);
        user.setUsername("刚才失败传递的id是:==>" + id);
        return user;
    }
}
```

四 Feign 中使用 Hystrix

> Feign 中也可以使用 Hystrix,但是默认情况下是禁用的,需要我们手动开启
>
> 参考13consumer-eureka-feign-hystrix

4.1 POM

```
             <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>

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

        </dependency>
```

4.2 application.yml

```yaml
server:
  port: 9001
eureka: #注册中心的地址
  client:
    service-url:
      defaultZone: http://zhangsan:abc@localhost:10000/eureka #curl风格
spring:
  application:
    name: 13consumer-eureka-feign-hystrix
04provider-eureka: #给这个服务配置负载均衡规则
  ribbon:
    NFLoadBalancerRuleClassName: com.netflix.loadbalancer.RandomRule
feign: #默认情况下 feign屏蔽了hystrix ,需要打开才可以
  hystrix:
    enabled: true
```

4.3 创建托底类

> 类似于我们的 fallbackfactory,Feign 因为是一个接口,所以 hystrix 要求我们指定一个接口的实现类并重写内部的方法来作为降级的方法,调用 feign 中的哪个方法失败就会调用实现类的哪个方法

```java


import com.qianfeng.microservice.consumer.pojo.User;
import org.springframework.stereotype.Component;

/**
 * Created by jackiechan on 19-1-9/下午2:43
 *
 * @Author jackiechan
 */
@Component //这个注解是必须得加的
public class MyFeign01Fallback implements FeignClient01 {
    @Override
    public User getUserById(Long id) {
        User user = new User();
        user.setId(-200);
        user.setUsername("小可爱是李登进");
        return user;
    }

    @Override
    public String save(User user) {
        return null;
    }
}
```

4.4 修改 Feign 接口

```
import com.qianfeng.microservice.consumer.pojo.User;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

/**
 * Created by jackiechan on 19-1-9/上午10:27
 *
 * @Author jackiechan
 */
//当配置fallback的时候,如果接口中的方法出现问题,就会去执行指定类里面的方法,因为类里面的方法可以随便写,所以这个类有特殊要求,必须实现当前feign接口
@FeignClient(value = "04provider-eureka",fallback = MyFeign01Fallback.class)//声明当前接口是一个feignclient,参数为想访问哪个服务,写的是在eureka上面的服务的名字
public interface FeignClient01 {

    @GetMapping("/user/info/{id}")
    User getUserById(@PathVariable("id") Long id);//注意在controller里面PathVariable的路径参数和方法的形参名字一样的时候可以忽略里面的名字.但是在feign里面必须写

    @PostMapping("/user/save")//这个方法上面只要有参数 就会被发出去,如果传递的是复杂对象,不管这里是不是GET都会以post方式发出去,出错还是不出错,取决于提供者那边的限定
    String save(User user);
}
```

4.4 主程序

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.openfeign.EnableFeignClients;

/**
 * Created by jackiechan on 19-1-8/下午12:04
 *
 * @Author jackiechan
 */
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients//开启对feign的支持
@EnableCircuitBreaker//开启熔断
public class ConsumerStarter {


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


}
```

4.5 fallbackfatory 模式

> 在Feign中使用hystrix 除了上面的方式外还有一种方式来使用,本质上这两种方式没区别,都是通过返回一个 feign 的默认实现类来完成

4.5.1 fallbackfactory

```
import feign.hystrix.FallbackFactory;
import org.springframework.stereotype.Component;

/**
 * Created by jackiechan on 19-1-9/下午2:56
 *
 * @Author jackiechan
 */
//这个类是 feign 接口的实现类的工厂对象,通过泛型来指定为哪个 feign 接口
@Component
public class MyFacbackFactory  implements FallbackFactory<FeignClient01> {
  //此处需要返回一个实现类对象
    @Override
    public FeignClient01 create(Throwable throwable) {
      //返回一个 feign 接口的实现类对象,此处是使用匿名对象
//        return new FeignClient01() {
//            @Override
//            public User getUserById(Long id) {
//                return null;
//            }
//
//            @Override
//            public String save(User user) {
//                return null;
//            }
//        };
        return  new MyFeign01Fallback();//此处返回的是上面我们写的实现类的对象
    }
}
```

4.5.2 修改 Feign 接口

```
import com.qianfeng.microservice.consumer.pojo.User;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

/**
 * Created by jackiechan on 19-1-9/上午10:27
 *
 * @Author jackiechan
 */
//当配置了fallbackFactory,出现问题的时候会执行里面的cretae的方法获取到当前接口的一个实现类对象,然后调用里面和失败方法同名的方法
@FeignClient(value = "04provider-eureka",fallbackFactory = MyFacbackFactory.class)//声明当前接口是一个feignclient,参数为想访问哪个服务,写的是在eureka上面的服务的名字
public interface FeignClient01 {
```

```
    @GetMapping("/user/info/{id}")
    User getUserById(@PathVariable("id") Long id);//注意在controller里面PathVariable的路径参数和方法的形参名字一样的时候可以忽略里面的名字.但是在feign里面必须写

    @PostMapping("/user/save")//这个方法上面只要有参数 就会被发出去,如果传递的是复杂对象,不管这里是不是GET都会以post方式发出去,出错还是不出错,取决于提供者那边的限定
    String save(User user);
}
```

4.5.3 其他

> 其他和之前的一样

五 Command Properties(属性)

The following Properties control `HystrixCommand` behavior:

Execution

The following Properties control how `HystrixCommand.run()` executes.

execution.isolation.strategy

This property indicates which isolation strategy `HystrixCommand.run()` executes with, one of the following two choices:

- `THREAD` — it executes on a separate thread and concurrent requests are limited by the number of threads in the thread-pool
- `SEMAPHORE` — it executes on the calling thread and concurrent requests are limited by the semaphore count

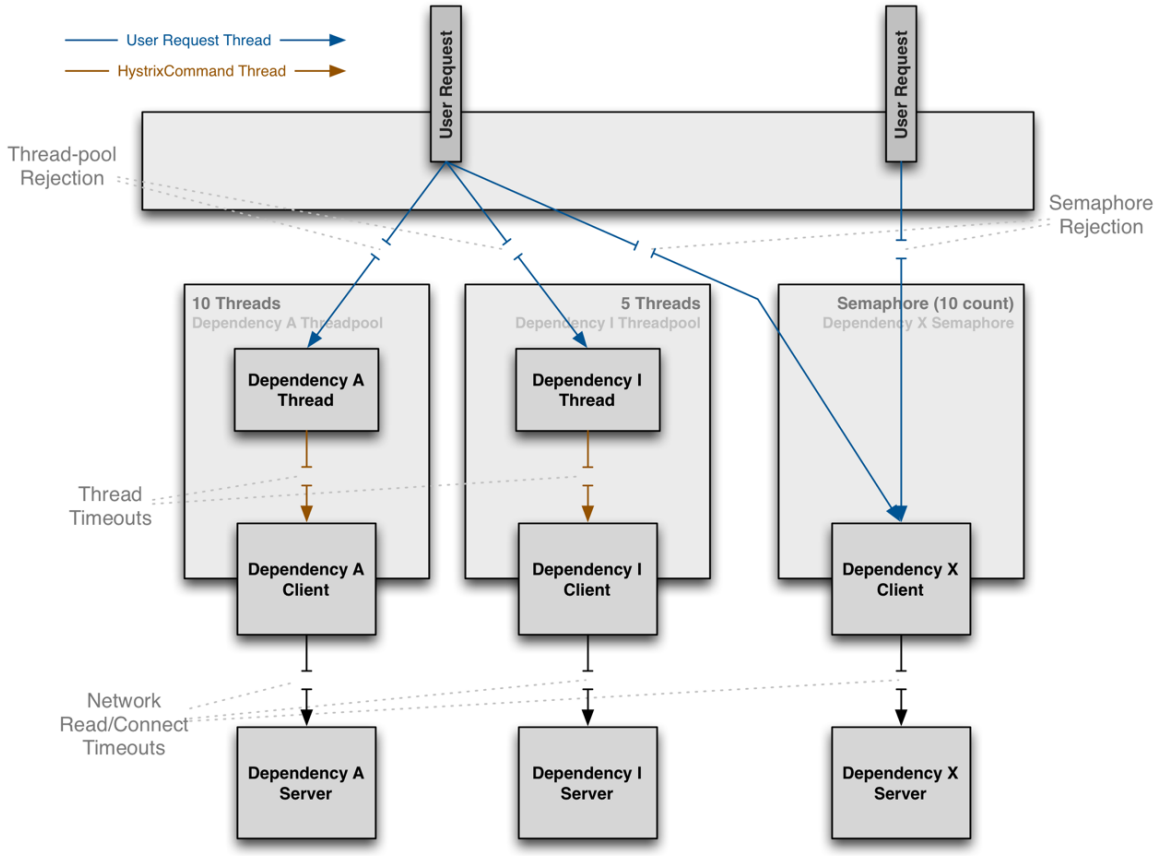
Thread or Semaphore

The default, and the recommended setting, is to run `HystrixCommand` s using thread isolation (`THREAD`) and `HystrixObservableCommand` s using semaphore isolation (`SEMAPHORE`).

Commands executed in threads have an extra layer of protection against latencies beyond what network timeouts can offer.

Generally the only time you should use semaphore isolation for `HystrixCommand` s is when the call is so high volume (hundreds per second, per instance) that the overhead of separate threads is too high; this typically only applies to non-network calls.

> Netflix API has 100+ commands running in 40+ thread pools and only a handful of those commands are not running in a thread - those that fetch metadata from an in-memory cache or that are façades to thread-isolated commands (see **“Primary + Secondary with Fallback” pattern** for more information on this).



See **how isolation works** for more information about this decision.

Default Value	`THREAD` (see `ExecutionIsolationStrategy.THREAD`)
Possible Values	`THREAD`, `SEMAPHORE`
Default Property	`hystrix.command.default.execution.isolation.strategy`
Instance Property	`hystrix.command.*HystrixCommandKey*.execution.isolation.strategy`
How to Set Instance Default:	`// to use thread isolation HystrixCommandProperties.Setter() .withExecutionIsolationStrategy(ExecutionIsolationStrategy.THREAD) // to use semaphore isolation HystrixCommandProperties.Setter() .withExecutionIsolationStrategy(ExecutionIsolationStrategy.SEMAPHORE)`

execution.isolation.thread.timeoutInMilliseconds

This property sets the time in milliseconds after which the caller will observe a timeout and walk away from the command execution. Hystrix marks the `HystrixCommand` as a TIMEOUT, and performs fallback logic. Note that there is configuration for turning off timeouts per-command, if that is desired (see command.timeout.enabled).

Note: Timeouts will fire on `HystrixCommand.queue()`, even if the caller never calls `get()` on the resulting Future. Before Hystrix 1.4.0, only calls to `get()` triggered the timeout mechanism to take effect in such a case.

Default Value	`1000`
Default Property	`hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds`
Instance Property	`hystrix.command.*HystrixCommandKey*.execution.isolation.thread.timeoutInMilliseconds`
How to Set Instance Default	`HystrixCommandProperties.Setter() .withExecutionTimeoutInMilliseconds(int value)`

execution.timeout.enabled

This property indicates whether the `HystrixCommand.run()` execution should have a timeout.

Default Value	`true`
Default Property	`hystrix.command.default.execution.timeout.enabled`
Instance Property	`hystrix.command.*HystrixCommandKey*.execution.timeout.enabled`
How to Set Instance Default	`HystrixCommandProperties.Setter() .withExecutionTimeoutEnabled(boolean value)`

execution.isolation.thread.interruptOnTimeout

This property indicates whether the `HystrixCommand.run()` execution should be interrupted when a timeout occurs.

Default Value	`true`
Default Property	`hystrix.command.default.execution.isolation.thread.interruptOnTimeout`
Instance Property	`hystrix.command.*HystrixCommandKey*.execution.isolation.thread.interruptOnTimeout`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withExecutionIsolationThreadInterruptOnTimeout(boolean value)`

execution.isolation.thread.interruptOnCancel

This property indicates whether the `HystrixCommand.run()` execution should be interrupted when a cancellation occurs.

Default Value	`false`
Default Property	`hystrix.command.default.execution.isolation.thread.interruptOnCancel`
Instance Property	`hystrix.command.*HystrixCommandKey*.execution.isolation.thread.interruptOnCancel`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withExecutionIsolationThreadInterruptOnCancel(boolean value)`

execution.isolation.semaphore.maxConcurrentRequests

This property sets the maximum number of requests allowed to a `HystrixCommand.run()` method when you are using `ExecutionIsolationStrategy.SEMAPHORE` .

If this maximum concurrent limit is hit then subsequent requests will be rejected.

The logic that you use when you size a semaphore is basically the same as when you choose how many threads to add to a thread-pool, but the overhead for a semaphore is far smaller and typically the executions are far faster (sub-millisecond), otherwise you would be using threads.

> For example, 5000rps on a single instance for in-memory lookups with metrics being gathered has been seen to work with a semaphore of only 2.

The isolation principle is still the same so the semaphore should still be a small percentage of the overall container (i.e. Tomcat) thread pool, not all of or most of it, otherwise it provides no protection.

Default Value	`10`
Default Property	`hystrix.command.default.execution.isolation.semaphore.maxConcurrentRequests`
Instance Property	`hystrix.command.*HystrixCommandKey*.execution.isolation.semaphore.maxConcurrentRequests`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withExecutionIsolationSemaphoreMaxConcurrentRequests(int value)`

Fallback

The following properties control how `HystrixCommand.getFallback()` executes. These properties apply to both `ExecutionIsolationStrategy.THREAD` and `ExecutionIsolationStrategy.SEMAPHORE` .

fallback.isolation.semaphore.maxConcurrentRequests

This property sets the maximum number of requests a `HystrixCommand.getFallback()` method is allowed to make from the calling thread.

If the maximum concurrent limit is hit then subsequent requests will be rejected and an exception thrown since no fallback could be retrieved.

Default Value	`10`
Default Property	`hystrix.command.default.fallback.isolation.semaphore.maxConcurrentRequests`
Instance Property	`hystrix.command.*HystrixCommandKey*.fallback.isolation.semaphore.maxConcurrentRequests`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withFallbackIsolationSemaphoreMaxConcurrentRequests(int value)`

fallback.enabled

Since: 1.2

This property determines whether a call to `HystrixCommand.getFallback()` will be attempted when failure or rejection occurs.

Default Value	`true`
Default Property	`hystrix.command.default.fallback.enabled`
Instance Property	`hystrix.command.*HystrixCommandKey*.fallback.enabled`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withFallbackEnabled(boolean value)`

Circuit Breaker

The circuit breaker properties control behavior of the `HystrixCircuitBreaker` .

circuitBreaker.enabled

This property determines whether a circuit breaker will be used to track health and to short-circuit requests if it trips.

Default Value	`true`
Default Property	`hystrix.command.default.circuitBreaker.enabled`
Instance Property	`hystrix.command.*HystrixCommandKey*.circuitBreaker.enabled`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withCircuitBreakerEnabled(boolean value)`

circuitBreaker.requestVolumeThreshold

This property sets the minimum number of requests in a rolling window that will trip the circuit.

For example, if the value is 20, then if only 19 requests are received in the rolling window (say a window of 10 seconds) the circuit will not trip open even if all 19 failed.

Default Value	`20`
Default Property	`hystrix.command.default.circuitBreaker.requestVolumeThreshold`
Instance Property	`hystrix.command.*HystrixCommandKey*.circuitBreaker.requestVolumeThreshold`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withCircuitBreakerRequestVolumeThreshold(int value)`

circuitBreaker.sleepWindowInMilliseconds

This property sets the amount of time, after tripping the circuit, to reject requests before allowing attempts again to determine if the circuit should again be closed.

Default Value	`5000`
Default Property	`hystrix.command.default.circuitBreaker.sleepWindowInMilliseconds`
Instance Property	`hystrix.command.*HystrixCommandKey*.circuitBreaker.sleepWindowInMilliseconds`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withCircuitBreakerSleepWindowInMilliseconds(int value)`

circuitBreaker.errorThresholdPercentage

This property sets the error percentage at or above which the circuit should trip open and start short-circuiting requests to fallback logic.

Default Value	`50`
Default Property	`hystrix.command.default.circuitBreaker.errorThresholdPercentage`
Instance Property	`hystrix.command.*HystrixCommandKey*.circuitBreaker.errorThresholdPercentage`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withCircuitBreakerErrorThresholdPercentage(int value)`

circuitBreaker.forceOpen

This property, if `true`, forces the circuit breaker into an open (tripped) state in which it will reject all requests.

This property takes precedence over `circuitBreaker.forceClosed`.

Default Value	`false`
Default Property	`hystrix.command.default.circuitBreaker.forceOpen`
Instance Property	`hystrix.command.*HystrixCommandKey*.circuitBreaker.forceOpen`
How to Set Instance Default	`HystrixCommandProperties.Setter().withCircuitBreakerForceOpen(boolean value)`

circuitBreaker.forceClosed

This property, if `true`, forces the circuit breaker into a closed state in which it will allow requests regardless of the error percentage.

The `circuitBreaker.forceOpen` property takes precedence so if it is set to `true` this property does nothing.

Default Value	`false`
Default Property	`hystrix.command.default.circuitBreaker.forceClosed`
Instance Property	`hystrix.command.*HystrixCommandKey*.circuitBreaker.forceClosed`
How to Set Instance Default	`HystrixCommandProperties.Setter().withCircuitBreakerForceClosed(boolean value)`

Metrics

The following properties are related to capturing metrics from `HystrixCommand` and `HystrixObservableCommand` execution.

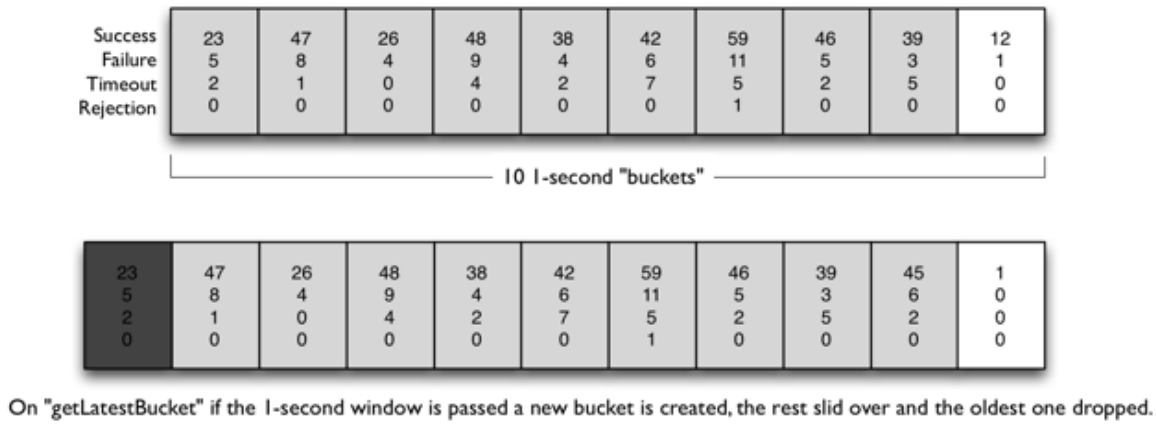
metrics.rollingStats.timeInMilliseconds

This property sets the duration of the statistical rolling window, in milliseconds. This is how long Hystrix keeps metrics for the circuit breaker to use and for publishing.

As of 1.4.12, this property affects the initial metrics creation only, and adjustments made to this property after startup will not take effect. This avoids metrics data loss, and allows optimizations to metrics gathering.

The window is divided into buckets and “rolls” by these increments.

For example, if this property is set to 10 seconds (`10000`) with ten 1-second buckets, the following diagram represents how it rolls new buckets on and old ones off:



On "getLatestBucket" if the 1-second window is passed a new bucket is created, the rest slid over and the oldest one dropped.

Default Value	`10000`
Default Property	`hystrix.command.default.metrics.rollingStats.timeInMilliseconds`
Instance Property	`hystrix.command.*HystrixCommandKey*.metrics.rollingStats.timeInMilliseconds`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withMetricsRollingStatisticalWindowInMilliseconds(int value)`

metrics.rollingStats.numBuckets

This property sets the number of buckets the rolling statistical window is divided into.

Note: The following must be true — “ `metrics.rollingStats.timeInMilliseconds % metrics.rollingStats.numBuckets == 0` ” — otherwise it will throw an exception.

In other words, 10000/10 is okay, so is 10000/20 but 10000/7 is not.

As of 1.4.12, this property affects the initial metrics creation only, and adjustments made to this property after startup will not take effect. This avoids metrics data loss, and allows optimizations to metrics gathering.

Default Value	`10`
Possible Values	Any value that `metrics.rollingStats.timeInMilliseconds` can be evenly divided by. The result however should be buckets measuring hundreds or thousands of milliseconds. Performance at high volume has not been tested with buckets <100ms.
Default Property	`hystrix.command.default.metrics.rollingStats.numBuckets`
Instance Property	`hystrix.command.*HystrixCommandKey*.metrics.rollingStats.numBuckets`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withMetricsRollingStatisticalWindowBuckets(int value)`

metrics.rollingPercentile.enabled

This property indicates whether execution latencies should be tracked and calculated as percentiles. If they are disabled, all summary statistics (mean, percentiles) are returned as -1.

Default Value	`true`
Default Property	`hystrix.command.default.metrics.rollingPercentile.enabled`
Instance Property	`hystrix.command.*HystrixCommandKey*.metrics.rollingPercentile.enabled`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withMetricsRollingPercentileEnabled(boolean value)`

metrics.rollingPercentile.timeInMilliseconds

This property sets the duration of the rolling window in which execution times are kept to allow for percentile calculations, in milliseconds.

The window is divided into buckets and “rolls” by those increments.

As of 1.4.12, this property affects the initial metrics creation only, and adjustments made to this property after startup will not take effect. This avoids metrics data loss, and allows optimizations to metrics gathering.

Default Value	`60000`
Default Property	`hystrix.command.default.metrics.rollingPercentile.timeInMilliseconds`
Instance Property	`hystrix.command.*HystrixCommandKey*.metrics.rollingPercentile.timeInMilliseconds`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withMetricsRollingPercentileWindowInMilliseconds(int value)`

metrics.rollingPercentile.numBuckets

This property sets the number of buckets the `rollingPercentile` window will be divided into.

Note: The following must be true — "`metrics.rollingPercentile.timeInMilliseconds % metrics.rollingPercentile.numBuckets == 0`" — otherwise it will throw an exception.

In other words, 60000/6 is okay, so is 60000/60 but 10000/7 is not.

As of 1.4.12, this property affects the initial metrics creation only, and adjustments made to this property after startup will not take effect. This avoids metrics data loss, and allows optimizations to metrics gathering.

Default Value	`6`
Possible Values	Any value that `metrics.rollingPercentile.timeInMilliseconds` can be evenly divided by. The result however should be buckets measuring thousands of milliseconds. Performance at high volume has not been tested with buckets <1000ms.
Default Property	`hystrix.command.default.metrics.rollingPercentile.numBuckets`
Instance Property	`hystrix.command.*HystrixCommandKey*.metrics.rollingPercentile.numBuckets`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withMetricsRollingPercentileWindowBuckets(int value)`

metrics.rollingPercentile.bucketSize

This property sets the maximum number of execution times that are kept per bucket. If more executions occur during the time they will wrap around and start over-writing at the beginning of the bucket.

For example, if bucket size is set to 100 and represents a bucket window of 10 seconds, but 500 executions occur during this time, only the last 100 executions will be kept in that 10 second bucket.

If you increase this size, this also increases the amount of memory needed to store values and increases the time needed for sorting the lists to do percentile calculations.

As of 1.4.12, this property affects the initial metrics creation only, and adjustments made to this property after startup will not take effect. This avoids metrics data loss, and allows optimizations to metrics gathering.

Default Value	`100`
Default Property	`hystrix.command.default.metrics.rollingPercentile.bucketSize`
Instance Property	`hystrix.command.*HystrixCommandKey*.metrics.rollingPercentile.bucketSize`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withMetricsRollingPercentileBucketSize(int value)`

metrics.healthSnapshot.intervalInMilliseconds

This property sets the time to wait, in milliseconds, between allowing health snapshots to be taken that calculate success and error percentages and affect circuit breaker status.

On high-volume circuits the continual calculation of error percentages can become CPU intensive thus this property allows you to control how often it is calculated.

Default Value	`500`
Default Property	`hystrix.command.default.metrics.healthSnapshot.intervalInMilliseconds`
Instance Property	`hystrix.command.*HystrixCommandKey*.metrics.healthSnapshot.intervalInMilliseconds`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withMetricsHealthSnapshotIntervalInMilliseconds(int value)`

Request Context

These properties concern `HystrixRequestContext` functionality used by `HystrixCommand`.

requestCache.enabled

This property indicates whether `HystrixCommand.getCacheKey()` should be used with `HystrixRequestCache` to provide de-duplication functionality via request-scoped caching.

Default Value	`true`
Default Property	`hystrix.command.default.requestCache.enabled`
Instance Property	`hystrix.command.*HystrixCommandKey*.requestCache.enabled`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withRequestCacheEnabled(boolean value)`

requestLog.enabled

This property indicates whether `HystrixCommand` execution and events should be logged to `HystrixRequestLog` .

Default Value	`true`
Default Property	`hystrix.command.default.requestLog.enabled`
Instance Property	`hystrix.command.*HystrixCommandKey*.requestLog.enabled`
How to Set Instance Default	`HystrixCommandProperties.Setter()` `.withRequestLogEnabled(boolean value)`

Collapser Properties

The following properties control `HystrixCollapser` behavior.

maxRequestsInBatch

This property sets the maximum number of requests allowed in a batch before this triggers a batch execution.

Default Value	`Integer.MAX_VALUE`
Default Property	`hystrix.collapser.default.maxRequestsInBatch`
Instance Property	`hystrix.collapser.*HystrixCollapserKey*.maxRequestsInBatch`
How to Set Instance Default	`HystrixCollapserProperties.Setter()` `.withMaxRequestsInBatch(int value)`

timerDelayInMilliseconds

This property sets the number of milliseconds after the creation of the batch that its execution is triggered.

Default Value	`10`
Default Property	`hystrix.collapser.default.timerDelayInMilliseconds`
Instance Property	`hystrix.collapser.*HystrixCollapserKey*.timerDelayInMilliseconds`
How to Set Instance Default	`HystrixCollapserProperties.Setter()` `.withTimerDelayInMilliseconds(int value)`

requestCache.enabled

This property indicates whether request caching is enabled for `HystrixCollapser.execute()` and `HystrixCollapser.queue()` invocations.

Default Value	`true`
Default Property	`hystrix.collapser.default.requestCache.enabled`
Instance Property	`hystrix.collapser.*HystrixCollapserKey*.requestCache.enabled`
How to Set Instance Default	`HystrixCollapserProperties.Setter()` `.withRequestCacheEnabled(boolean value)`

ThreadPool Properties

The following properties control the behavior of the thread-pools that Hystrix Commands execute on. Please note that these names match those in **the ThreadPoolExecutor Javadoc**

Most of the time the default value of 10 threads will be fine (often it could be made smaller).

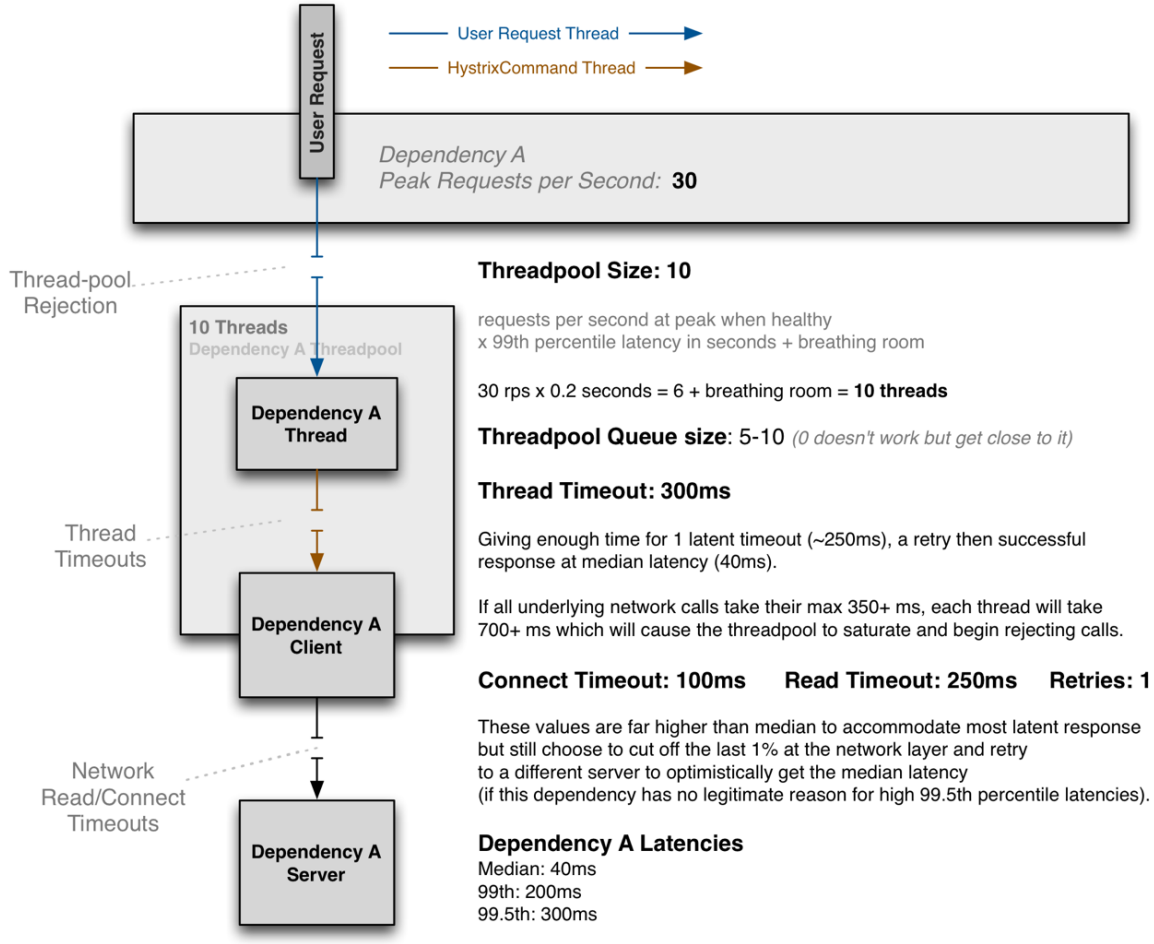
To determine if it needs to be larger, a basic formula for calculating the size is:

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

See the example below to see how this formula is put into practice.

The general principle is keep the pool as small as possible, as it is the primary tool to shed load and prevent resources from becoming blocked if latency occurs.

> Netflix API has 30+ of its threadpools set at 10, two at 20, and one at 25.



The above diagram shows an example configuration in which the dependency has no reason to hit the 99.5th percentile and therefore it cuts it short at the network timeout layer and immediately retries with the expectation that it will get median latency most of the time, and will be able to accomplish this all within the 300ms thread timeout.

If the dependency has legitimate reasons to sometimes hit the 99.5th percentile (such as cache miss with lazy generation) then the network timeout will be set higher than it, such as at 325ms with 0 or 1 retries and the thread timeout set higher (350ms+).

The thread-pool is sized at 10 to handle a burst of 99th percentile requests, but when everything is healthy this threadpool will typically only have 1 or 2 threads active at any given time to serve mostly 40ms median calls.

When you configure it correctly a timeout at the `HystrixCommand` layer should be rare, but the protection is there in case something other than network latency affects the time, or the combination of connect+read+retry+connect+read in a worst case scenario still exceeds the configured overall timeout.

The aggressiveness of configurations and tradeoffs in each direction are different for each dependency.

You can change configurations in real-time as needed as performance characteristics change or when problems are found, all without the risk of taking down the entire app if problems or misconfigurations occur.

coreSize

This property sets the core thread-pool size.

Default Value	10
Default Property	hystrix.threadpool.default.coreSize
Instance Property	hystrix.threadpool.*HystrixThreadPoolKey*.coreSize
How to Set Instance Default	HystrixThreadPoolProperties.Setter() .withCoreSize(int value)

maximumSize

Added in 1.5.9. This property sets the maximum thread-pool size. This is the maximum amount of concurrency that can be supported without starting to reject `HystrixCommand` s. Please note that this setting only takes effect if you also set `allowMaximumSizeToDivergeFromCoreSize` . Prior to 1.5.9, core and maximum sizes were always equal.

Default Value	10
Default Property	hystrix.threadpool.default.maximumSize
Instance Property	hystrix.threadpool.*HystrixThreadPoolKey*.maximumSize
How to Set Instance Default	HystrixThreadPoolProperties.Setter() .withMaximumSize(int value)

maxQueueSize

This property sets the maximum queue size of the `BlockingQueue` implementation.

If you set this to `-1` then `SynchronousQueue` will be used, otherwise a positive value will be used with `LinkedBlockingQueue` .

Note: This property only applies at initialization time since queue implementations cannot be resized or changed without re-initializing the thread executor which is not supported.

If you need to overcome this limitation and to allow dynamic changes in the queue, see the `queueSizeRejectionThreshold` property.

To change between `SynchronousQueue` and `LinkedBlockingQueue` requires a restart.

Default Value	-1
Default Property	hystrix.threadpool.default.maxQueueSize
Instance Property	hystrix.threadpool.*HystrixThreadPoolKey*.maxQueueSize
How to Set Instance Default	HystrixThreadPoolProperties.Setter() .withMaxQueueSize(int value)

queueSizeRejectionThreshold

This property sets the queue size rejection threshold — an artificial maximum queue size at which rejections will occur even if `maxQueueSize` has not been reached. This property exists because the `maxQueueSize` of a `BlockingQueue` cannot be dynamically changed and we want to allow you to dynamically change the queue size that affects rejections.

This is used by `HystrixCommand` when queuing a thread for execution.

Note: This property is not applicable if `maxQueueSize == -1` .

Default Value	`5`
Default Property	`hystrix.threadpool.default.queueSizeRejectionThreshold`
Instance Property	`hystrix.threadpool.*HystrixThreadPoolKey*.queueSizeRejectionThreshold`
How to Set Instance Default	`HystrixThreadPoolProperties.Setter()` `.withQueueSizeRejectionThreshold(int value)`

keepAliveTimeMinutes

This property sets the keep-alive time, in minutes.

Prior to 1.5.9, all thread pools were fixed-size, as `coreSize == maximumSize` . In 1.5.9 and after, setting `allowMaximumSizeToDivergeFromCoreSize` to `true` allows those 2 values to diverge, such that the pool may acquire/release threads. If `coreSize < maximumSize` , then this property controls how long a thread will go unused before being released.

Default Value	`1`
Default Property	`hystrix.threadpool.default.keepAliveTimeMinutes`
Instance Property	`hystrix.threadpool.*HystrixThreadPoolKey*.keepAliveTimeMinutes`
How to Set Instance Default	`HystrixThreadPoolProperties.Setter()` `.withKeepAliveTimeMinutes(int value)`

allowMaximumSizeToDivergeFromCoreSize

Added in 1.5.9. This property allows the configuration for `maximumSize` to take effect. That value can then be equal to, or higher, than `coreSize` . Setting `coreSize < maximumSize` creates a thread pool which can sustain `maximumSize` concurrency, but will return threads to the system during periods of relative inactivity. (subject to `keepAliveTimeInMinutes`)

Default Value	`false`
Default Property	`hystrix.threadpool.default.allowMaximumSizeToDivergeFromCoreSize`
Instance Property	`hystrix.threadpool.*HystrixThreadPoolKey*.allowMaximumSizeToDivergeFromCoreSize`
How to Set Instance Default	`HystrixThreadPoolProperties.Setter()` `.withAllowMaximumSizeToDivergeFromCoreSize(boolean value)`

metrics.rollingStats.timeInMilliseconds

This property sets the duration of the statistical rolling window, in milliseconds. This is how long metrics are kept for the thread pool.

The window is divided into buckets and “rolls” by those increments.

Default Value	`10000`
Default Property	`hystrix.threadpool.default.metrics.rollingStats.timeInMilliseconds`
Instance Property	`hystrix.threadpool.*HystrixThreadPoolKey*.metrics.rollingStats.timeInMilliseconds`
How to Set Instance Default	`HystrixThreadPoolProperties.Setter()` `.withMetricsRollingStatisticalWindowInMilliseconds(int value)`

metrics.rollingStats.numBuckets

This property sets the number of buckets the rolling statistical window is divided into.

Note: The following must be true — “ `metrics.rollingStats.timeInMilliseconds % metrics.rollingStats.numBuckets == 0` ” — otherwise it will throw an exception.

In other words, 10000/10 is okay, so is 10000/20 but 10000/7 is not.

Default Value	`10`
Possible Values	Any value that `metrics.rollingStats.timeInMilliseconds` can be evenly divided by. The result however should be buckets measuring hundreds or thousands of milliseconds. Performance at high volume has not been tested with buckets <100ms.
Default Property	`hystrix.threadpool.default.metrics.rollingStats.numBuckets`
Instance Property	`hystrix.threadpool.*HystrixThreadPoolProperties*.metrics.rollingStats.numBuckets`
How to Set Instance Default	`HystrixThreadPoolProperties.Setter()` `.withMetricsRollingStatisticalWindowBuckets(int value)`