



服务容错 | Hystrix

通过服务注册发现与服务调用，整个微服务应用的基本功能已经可以实现了。但在实际项目中，除了功能性，我们还需要考虑服务在面对请求压力时的可靠性，请求用户的感官体验等。

- **服务熔断**：当某个服务出现问题时，切断该服务与系统的联系，以防止波及其它服务。
- **服务降级**：服务不可用时(如熔断后)，提供一个简单的服务来代替原服务响应，以优化用户的体验。
- **服务隔离**：使服务之间相互隔离，防止出现雪崩效应（服务多级调用时，一个服务拥堵，从而导致多级服务都拥堵）。
- **服务限流**：限制服务的请求频率。

服务容错是由调用者来考虑，而不是由服务自身来考虑。

Hystrix

服务熔断

Hystrix 内部已经实现了熔断，并提供了默认的熔断配置。我们只需要修改配置即可：

```
hystrix:
  command:
    default:
      execution:
        isolation:
          thread:
            timeoutInMilliseconds: 2000 # 超时设置：设置为2s没获取到数据熔断，默认为1s
      circuitBreaker:
        requestVolumeThreshold: 20 # 触发熔断的最小请求次数，默认20次/10s
        sleepWindowInMilliseconds: 10000 # 熔断几ms后请求尝试，默认5s
        errorThresholdPercentage: 50 # 触发熔断的失败请求最小占比，默认50%
```

YAML

hystrix熔断器，即被@HystrixCommand注解的方法，它有3种状态：

- OPEN：打开：所有请求都进入降级方法。
- CLOSED：关闭：可访问全部请求，统计失败次数，超过给定时间时半开，超过指定时间内失败次数上限时打开。
- HALF_OPEN：半开：打开5s后，进入半开（尝试释放一个请求到服务，访问成功时关闭，否则保持打开）。

熔断器工作原理

熔断器默认关闭，但统计对服务请求失败的次数，并设置一个阈值（假设10s内不能失败100次），如果达到这个阈值，熔断器就会进入打开状态，将所有对服务的请求都进行降级操作。但服务不能一直被降级，熔断器每隔5s会尝试释放一个请求到服务，此时为半开状态，如果这个请求访问成功，则关闭断路器。

服务降级

Hystrix可以对失败、拒绝、超时的请求进行统一降级处理。

调用者引入Hystrix依赖：

```
<!--引入hystrix依赖-->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
<!-- 含@HystrixCommand-->
<dependency>
  <groupId>com.netflix.hystrix</groupId>
  <artifactId>hystrix-javanica</artifactId>
  <version>1.5.12</version>
</dependency>
```

XML

启动类启动 @EnableCircuitBreaker 注解激活

Java

```
@SpringBootApplication
@EnableFeignClients //激活Fegin (这里通过feign服务调用)
@EnableCircuitBreaker//激活Hystrix
public class MainApplication {

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

}
```

降级方法：

- 单接口降级：使用降级方法保护一个接口，优先级高， @HystrixCommand 注解

Java

```
@Autowired
private ImgFeignClient ifc;

// 熔断保护方法，fallbackMethod:指定熔断后的降级方法
@HystrixCommand(fallbackMethod = "himg")
@RequestMapping("/img4/{id}")
public Img fignimg(@PathVariable long id){
    Img img=ifc.findid(id);
    return img;
}

// 熔断降级方法，和需要受到保护的方法参数、返回值一致
public Img himg(long id){
    Img img=new Img();
    img.setName("触发降级方法");
    return img;
}
```

- 统一降级：使用降级方法保护整个类的全部接口， @DefaultProperties 注解

Java

```
@RestController
@RequestMapping("/main")
@DefaultProperties(defaultFallback = "tyimg")//指定公共的熔断降级方法
public class MainController {

    @Autowired
    private ImgFeignClient ifc;

    // 指定统一降级方法,不允许有参数，会替代所有熔断的接口
    public Img tyimg(){
        Img img=new Img();
        img.setName("触发统一降级方法");
        return img;
    }

    @HystrixCommand // 统一指定时使用
    @RequestMapping("/img/{id}")
    public Img fignimg(@PathVariable long id){
        Img img=ifc.findid(id);
        return img;
    }

}
```

- 实现类降级（仅针对 openFeign）： openFeign 通过接口来绑定对应服务并调用，那么可以构建实现类来实现接口，并使实现方法作为降级方法。

Feign 已经集成了 Hystrix，因此可以直接进行配置

YAML

```
feign:
  hystrix: #开启对hystrix的支持
    enabled: true
```

调用接口：

Java

```
// fallback: 指定作为降级的实现类
@FeignClient(name="img-service",fallback= ImgFeignClientCallBack.class)
public interface ImgFeignClient {

    // 配置需要调用的微服务接口
    @RequestMapping(value = "/img/findimg/{id}",method = RequestMethod.GET)
    public Img findid( @PathVariable long id);

}
```

降级实现类：需要注册为 bean

```
@Component
public class ImgFeignClientCallBack implements ImgFeignClient {

    @Override // 熔断降级的方法
    public Img findid(long id) {
        Img img=new Img();
        img.setName("触发降级方法");
        return img;
    }

}
```

Java

服务隔离

服务隔离有两种常见的方式：

- 信号量隔离：使用一个原子计数器记录当前运行的线程数，如果超过指定数量，丢弃请求。此方式严格控制线程且立即返回，无法应对突发流量。

```
hystrix:
  command:
    default:
      execution:
        isolation:
          strategy:
            ExecutionIsolationStrategy: SEMAPHORE # 信号量隔离，线程池隔离为THREAD
            maxConcurrentRequests: 20 # 最大信号量上限
```

YAML

- 线程池隔离：Tomcat以线程池方式处理请求，当某一服务压力过大而堵塞时，可能会造成整个系统的崩溃。

使用一个线程池存储并处理当前请求，设置任务返回超时时间，堆积的请求堆积入线程池队列。此方式要为所有需要的服务请求线程池（资源消耗略高），可以应对突发流量。

引入相关依赖：

```
<dependency>
  <groupId>com.netflix.hystrix</groupId>
  <artifactId>hystrix-metrics-event-stream</artifactId>
  <version>1.5.12</version>
</dependency>
<dependency>
  <groupId>com.netflix.hystrix</groupId>
  <artifactId>hystrix-javanica</artifactId>
  <version>1.5.12</version>
</dependency>
```

XML

实现 HystrixCommand 接口，并进行线程池配置：

```
public class OrderCommand extends HystrixCommand<String> {

    private RestTemplate restTemplate;
    private Long id;

    public OrderCommand(RestTemplate restTemplate, Long id) {
        super(setter());
        this.restTemplate = restTemplate;
        this.id = id;
    }

    @Override // 隔离的服务
    protected String run() throws Exception {
```

Java

```

    return restTemplate.getForObject("http://localhost/product/"+id, String.class);
}

@Override // 降级方法
protected String getFallback(){
    return null;
}

// 服务配置
private static Setter setter() {
    // 服务分组、// 服务标识 、// 线程池名称
    HystrixCommandGroupKey gk = HystrixCommandGroupKey.Factory.asKey("order_product");
    HystrixCommandKey ck = HystrixCommandKey.Factory.asKey("product");
    HystrixThreadPoolKey tpk = HystrixThreadPoolKey.Factory.asKey("order_product_pool");
    HystrixThreadPoolProperties.Setter tpp = HystrixThreadPoolProperties.Setter()
        .withCoreSize(50) // 线程池大小
        .withKeepAliveTimeMinutes(15) // 线程存活时间15s
        .withQueueSizeRejectionThreshold(100); // 队列等待的阈值为100,超过100执行拒绝 策略

    // 命令属性配置Hystrix 开启超时
    HystrixCommandProperties.Setter cp = HystrixCommandProperties.Setter()
        // 服务隔离方式：线程池隔离
        .withExecutionIsolationStrategy(HystrixCommandProperties.ExecutionIsolationStrategy.THREAD)
        .withExecutionTimeoutEnabled(false); // 禁止

    return HystrixCommand.Setter.withGroupKey(gk).andCommandKey(ck).andThreadPoolKey(tpk)
        .andThreadPoolPropertiesDefaults(tpp).andCommandPropertiesDefaults(cp);
}
}

```

控制层进行服务调用

```

@Autowired
private RestTemplate restTemplate;

@GetMapping("/buy/{id}")
public String order(@PathVariable Long id) throws Exception {
    return new OrderCommand(restTemplate,id).execute();
}

```

Java

实时监控平台

HystrixCommand与HystrixObservableCommand在执行时，会生成执行结果和运行指标，比如每秒请求数等。这些状态会暴露在Actuator提供的/health的端点中。

引入监控平台依赖，并开启hystrix：

```

<!-- 引入actuator依赖 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

<!-- 引入hystrix依赖 -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
<!-- dashboard -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
</dependency>

```

XML

暴露所有 actuator 监控的端点：

```

management:
  endpoints:
    web:
      exposure:
        include: '*'

```

YAML

访问监控平台：<http://localhost:8080/actuator/hystrix.stream>，但获取到的内容不直观，可以使用官方提供的仪表盘，开启仪表盘方式如下：


Java

```
@SpringBootApplication
@EnableFeignClients    // 激活Fegin
@EnableCircuitBreaker  // 激活Hystrix
@EnableHystrixDashboard // 激活仪表板
public class MainApplication {

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

}
```

访问仪表盘：<http://localhost:8080/hystrix>



Hystrix Dashboard

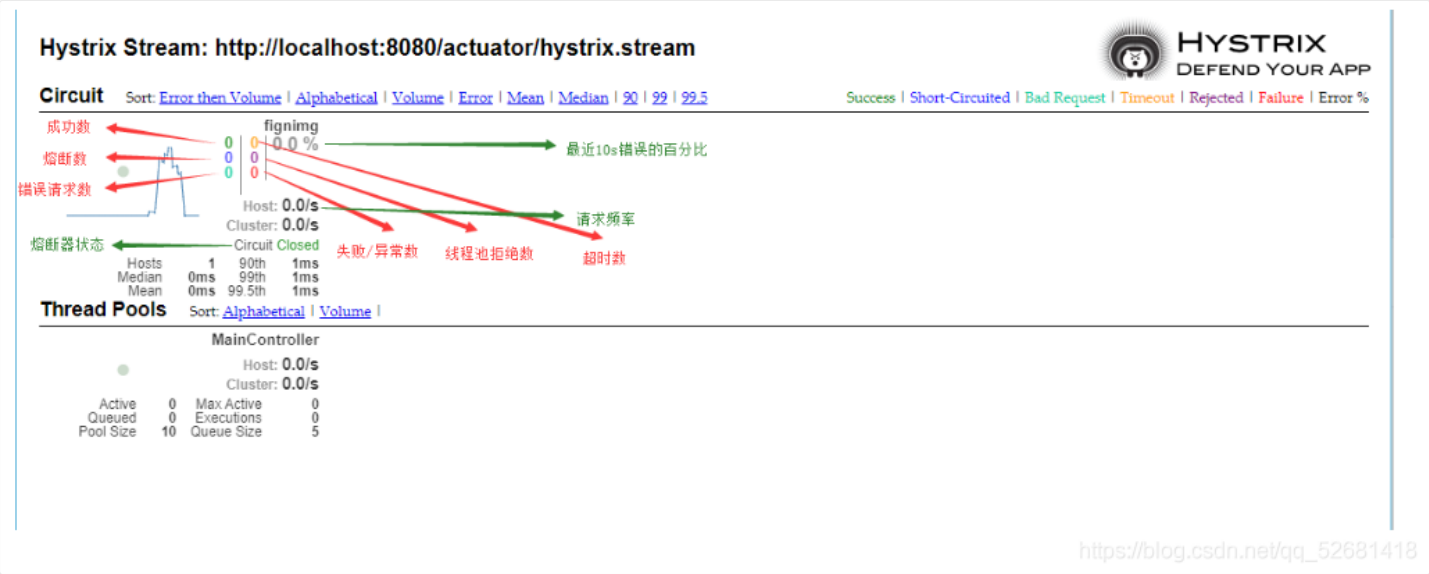
Cluster via Turbine (default cluster): <https://turbine-hostname:port/turbine.stream>
Cluster via Turbine (custom cluster): [https://turbine-hostname:port/turbine.stream?cluster=\[clusterName\]](https://turbine-hostname:port/turbine.stream?cluster=[clusterName])
Single Hystrix App: <https://hystrix-app:port/actuator/hystrix.stream>

Delay: ms Title:

Monitor Stream

https://blog.csdn.net/qq_52681418

在仪表盘输入监控的服务地址【<http://localhost:8080/actuator/hystrix.stream>】点击按钮进入监控页面：



断路器聚合监控：上图中只对一个服务进行了监控，断路器聚合监控实现对服务进行统一监控。

引入断路器聚合监控 Turbine 依赖：

XML

```
<!--引入turbine依赖-->
<dependency>
```

```

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

<!--引入hystrix依赖-->
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
<!-- dashboard-->
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
</dependency>

```

配置 Turbine:

```

turbine:
  appConfig: img-service  #监控多个时，分隔
  clusterNameExpression: "'default'"

```

YAML

开启 Turbine:

```

@SpringBootApplication
@EnableFeignClients //激活Fegin
@EnableCircuitBreaker//激活Hystrix
@EnableHystrixDashboard//激活仪表盘
@EnableTurbine//激活turbine
public class MainApplication {

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

```

Java

在图表工具页面输入: <http://localhost:8185/turbine.stream>