

Spring Cloud Hystrix: 服务容错保护

Spring Cloud Hystrix 是Spring Cloud Netflix 子项目的核心组件之一，具有服务容错及线程隔离等一系列服务保护功能，本文将对其用法进行详细介绍。

Hystrix 简介

在微服务架构中，服务与服务之间通过远程调用的方式进行通信，一旦某个被调用的服务发生了故障，其依赖服务也会发生故障，此时就会发生故障的蔓延，最终导致系统瘫痪。Hystrix实现了断路器模式，当某个服务发生故障时，通过断路器的监控，给调用方返回一个错误响应，而不是长时间的等待，这样就不会使得调用方由于长时间得不到响应而占用线程，从而防止故障的蔓延。Hystrix具备服务降级、服务熔断、线程隔离、请求缓存、请求合并及服务监控等强大功能。

创建一个hystrix-service模块

这里我们创建一个hystrix-service模块来演示hystrix的常用功能。

在pom.xml中添加相关依赖

```
1  <dependency>
2      <groupId>org.springframework.cloud</groupId>
3      <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
4  </dependency>
5  <dependency>
6      <groupId>org.springframework.cloud</groupId>
7      <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
8  </dependency>
9  <dependency>
10     <groupId>org.springframework.boot</groupId>
11     <artifactId>spring-boot-starter-web</artifactId>
12 </dependency>Copy to clipboardErrorCopied
```

在application.yml进行配置

主要是配置了端口、注册中心地址及user-service的调用路径。

```

1  server:
2    port: 8401
3  spring:
4    application:
5      name: hystrix-service
6  eureka:
7    client:
8      register-with-eureka: true
9      fetch-registry: true
10   service-url:
11     defaultZone: http://localhost:8001/eureka/
12  service-url:
13    user-service: http://user-serviceCopy to clipboardErrorCopied

```

在启动类上添加@EnableCircuitBreaker来开启Hystrix的断路器功能

```

1  @EnableCircuitBreaker
2  @EnableDiscoveryClient
3  @SpringBootApplication
4  public class HystrixServiceApplication {
5
6      public static void main(String[] args) {
7          SpringApplication.run(HystrixServiceApplication.class, args);
8      }Copy to clipboardErrorCopied

```

创建UserHystrixController接口用于调用user-service服务

服务降级演示

- 在UserHystrixController中添加用于测试服务降级的接口：

```

1  @GetMapping("/testFallback/{id}")
2  public CommonResult testFallback(@PathVariable Long id) {
3      return userService.getUser(id);
4  }Copy to clipboardErrorCopied

```

- 在UserService中添加调用方法与服务降级方法，方法上需要添加@HystrixCommand注解：

```

1  @HystrixCommand(fallbackMethod = "getDefaultUser")
2  public CommonResult getUser(Long id) {
3      return restTemplate.getForObject(userServiceUrl + "/user/{1}",
4      CommonResult.class, id);
5  }
6
7  public CommonResult getDefaultUser(@PathVariable Long id) {
8      User defaultUser = new User(-1L, "defaultUser", "123456");
9      return new CommonResult<>(defaultUser);
10 }Copy to clipboardErrorCopied

```

- 启动eureka-server、user-service、hystrix-service服务；

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
HYSTRIX-SERVICE	n/a (1)	(1)	UP (1) - 192.168.56.1:hystrix-service:8401
USER-SERVICE	n/a (1)	(1)	UP (1) - 192.168.56.1:user-service:8201

- 调用接口进行测试: <http://localhost:8401/user/testFallback/1>

← → ↻ ⓘ localhost:8401/user/testFallback/1

```
{
  "data": {
    "id": 1,
    "username": "macro",
    "password": "123456"
  },
  "message": "操作成功",
  "code": 200
}
```

- 关闭user-service服务重新测试该接口, 发现已经发生了服务降级:

← → ↻ ⓘ localhost:8401/user/testFallback/1

```
{
  "data": {
    "id": -1,
    "username": "defaultUser",
    "password": "123456"
  },
  "message": "操作成功",
  "code": 200
}
```

@HystrixCommand详解

@HystrixCommand中的常用参数

- fallbackMethod: 指定服务降级处理方法;
- ignoreExceptions: 忽略某些异常, 不发生服务降级;
- commandKey: 命令名称, 用于区分不同的命令;
- groupKey: 分组名称, Hystrix会根据不同的分组来统计命令的告警及仪表盘信息;
- threadPoolKey: 线程池名称, 用于划分线程池。

设置命令、分组及线程池名称

- 在UserHystrixController中添加测试接口：

```
1 @GetMapping("/testCommand/{id}")
2 public CommonResult testCommand(@PathVariable Long id) {
3     return userService.getUserCommand(id);
4 }Copy to clipboardErrorCopied
```

- 在UserService中添加方式实现功能：

```
1 @HystrixCommand(fallbackMethod = "getDefaultUser",
2     commandKey = "getUserCommand",
3     groupKey = "getUserGroup",
4     threadPoolKey = "getUserThreadPool")
5 public CommonResult getUserCommand(@PathVariable Long id) {
6     return restTemplate.getForObject(userServiceUrl + "/user/{1}",
7         CommonResult.class, id);
8 }Copy to clipboardErrorCopied
```

使用ignoreExceptions忽略某些异常降级

- 在UserHystrixController中添加测试接口：

```
1 @GetMapping("/testException/{id}")
2 public CommonResult testException(@PathVariable Long id) {
3     return userService.getUserException(id);
4 }Copy to clipboardErrorCopied
```

- 在UserService中添加实现方法，这里忽略了NullPointerException，当id为1时抛出IndexOutOfBoundsException，id为2时抛出NullPointerException：

```
1 @HystrixCommand(fallbackMethod = "getDefaultUser2", ignoreExceptions =
2     {NullPointerException.class})
3 public CommonResult getUserException(Long id) {
4     if (id == 1) {
5         throw new IndexOutOfBoundsException();
6     } else if (id == 2) {
7         throw new NullPointerException();
8     }
9     return restTemplate.getForObject(userServiceUrl + "/user/{1}",
10         CommonResult.class, id);
11 }
12
13 public CommonResult getDefaultUser2(@PathVariable Long id, Throwable e) {
14     LOGGER.error("getDefaultUser2 id:{},throwable class:{}, id, e.getClass());
15     User defaultUser = new User(-2L, "defaultUser2", "123456");
16     return new CommonResult<>(defaultUser);
17 }Copy to clipboardErrorCopied
```

- 调用接口进行测试：<http://localhost:8401/user/testException/1>

```
{
  "data": {
    "id": -2,
    "username": "defaultUser2",
    "password": "123456"
  },
  "message": "操作成功",
  "code": 200
}
```

异常被处理导致服务降级

- 调用接口进行测试: <http://localhost:8401/user/testException/1>

Whitelabel Error Page

异常被忽略，服务未降级

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Sep 15 16:35:48 CST 2019

There was an unexpected error (type=Internal Server Error, status=500).

No message available

Hystrix的请求缓存

当系统并发量越来越大时，我们需要使用缓存来优化系统，达到减轻并发请求线程数，提供响应速度的效果。

相关注解

- @CacheResult: 开启缓存，默认所有参数作为缓存的key，cacheKeyMethod可以通过返回String类型的方法指定key;
- @CacheKey: 指定缓存的key，可以指定参数或指定参数中的属性值为缓存key，cacheKeyMethod还可以通过返回String类型的方法指定;
- @CacheRemove: 移除缓存，需要指定commandKey。

测试使用缓存

- 在UserHystrixController中添加使用缓存的测试接口，直接调用三次getUserCache方法:

```
1 @GetMapping("/testCache/{id}")
2 public CommonResult testCache(@PathVariable Long id) {
3     userService.getUserCache(id);
4     userService.getUserCache(id);
5     userService.getUserCache(id);
6     return new CommonResult("操作成功", 200);
7 }Copy to clipboardErrorCopied
```

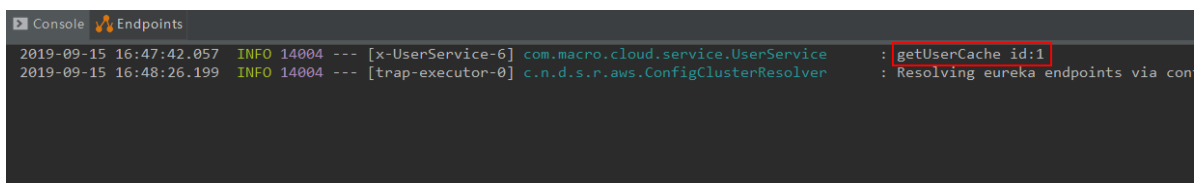
- 在UserService中添加具有缓存功能的getUserCache方法:

```

1  @CacheResult(cacheKeyMethod = "getCacheKey")
2  @HystrixCommand(fallbackMethod = "getDefaultUser", commandKey = "getUserCache")
3  public CommonResult getUserCache(Long id) {
4      LOGGER.info("getUserCache id:{0}", id);
5      return restTemplate.getForObject(userServiceUrl + "/user/{1}",
        CommonResult.class, id);
6  }
7
8  /**
9   * 为缓存生成key的方法
10  */
11  public String getCacheKey(Long id) {
12      return String.valueOf(id);
13  }Copy to clipboardErrorCopied

```

- 调用接口测试 <http://localhost:8401/user/testCache/1>,这个接口中调用了三次getUserCache方法,但是只打印了一次日志,说明有两次走的是缓存:



```

2019-09-15 16:47:42.057 INFO 14004 --- [x-UserService-6] com.macro.cloud.service.UserService : getUserCache id:1
2019-09-15 16:48:26.199 INFO 14004 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via con

```

测试移除缓存

- 在UserHystrixController中添加移除缓存的测试接口,调用一次removeCache方法:

```

1  @GetMapping("/testRemoveCache/{id}")
2  public CommonResult testRemoveCache(@PathVariable Long id) {
3      userService.getUserCache(id);
4      userService.removeCache(id);
5      userService.getUserCache(id);
6      return new CommonResult("操作成功", 200);
7  }Copy to clipboardErrorCopied

```

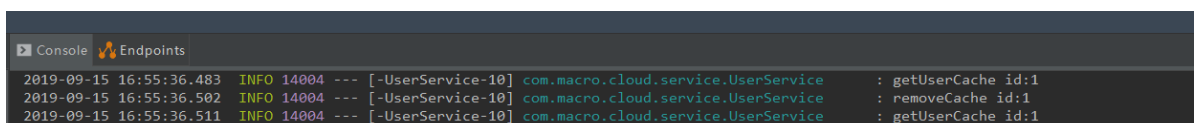
- 在UserService中添加具有移除缓存功能的removeCache方法:

```

1  @CacheRemove(commandKey = "getUserCache", cacheKeyMethod = "getCacheKey")
2  @HystrixCommand
3  public CommonResult removeCache(Long id) {
4      LOGGER.info("removeCache id:{0}", id);
5      return restTemplate.postForObject(userServiceUrl + "/user/delete/{1}", null,
        CommonResult.class, id);
6  }Copy to clipboardErrorCopied

```

- 调用接口测试 <http://localhost:8401/user/testRemoveCache/1>,可以发现两次查询都走的是接口:



```

2019-09-15 16:55:36.483 INFO 14004 --- [-UserService-10] com.macro.cloud.service.UserService : getUserCache id:1
2019-09-15 16:55:36.502 INFO 14004 --- [-UserService-10] com.macro.cloud.service.UserService : removeCache id:1
2019-09-15 16:55:36.511 INFO 14004 --- [-UserService-10] com.macro.cloud.service.UserService : getUserCache id:1

```

缓存使用过程中的问题

- 在缓存使用过程中，我们需要在每次使用缓存的请求前后对HystrixRequestContext进行初始化和关闭，否则会出现如下异常：

```
1 java.lang.IllegalStateException: Request caching is not available. Maybe you need
  to initialize the HystrixRequestContext?
2     at com.netflix.hystrix.HystrixRequestCache.get(HystrixRequestCache.java:104) ~
[hystrix-core-1.5.18.jar:1.5.18]
3     at com.netflix.hystrix.AbstractCommand$7.call(AbstractCommand.java:478) ~
[hystrix-core-1.5.18.jar:1.5.18]
4     at com.netflix.hystrix.AbstractCommand$7.call(AbstractCommand.java:454) ~
[hystrix-core-1.5.18.jar:1.5.18]Copy to clipboardErrorCopied
```

- 这里我们通过使用过滤器，在每个请求前后初始化和关闭HystrixRequestContext来解决该问题：

```
1  /**
2   * Created by macro on 2019/9/4.
3   */
4  @Component
5  @WebFilter(urlPatterns = "/*", asyncSupported = true)
6  public class HystrixRequestContextFilter implements Filter {
7      @Override
8      public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
9          HystrixRequestContext context =
HystrixRequestContext.initializeContext();
10         try {
11             filterChain.doFilter(servletRequest, servletResponse);
12         } finally {
13             context.close();
14         }
15     }
16 }Copy to clipboardErrorCopied
```

请求合并

微服务系统中的服务间通信，需要通过远程调用来实现，随着调用次数越来越多，占用线程资源也会越来越多。Hystrix中提供了@HystrixCollapser用于合并请求，从而达到减少通信消耗及线程数量的效果。

@HystrixCollapser的常用属性

- batchMethod：用于设置请求合并的方法；
- collapserProperties：请求合并属性，用于控制实例属性，有很多；
- timerDelayInMilliseconds：collapserProperties中的属性，用于控制每隔多少时间合并一次请求；

功能演示

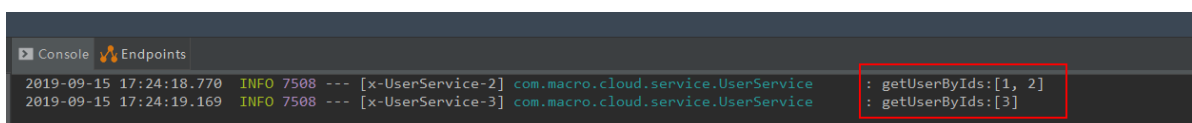
- 在UserHystrixController中添加testCollapser方法，这里我们先进行两次服务调用，再间隔200ms以后进行第三次服务调用：

```
1  @GetMapping("/testCollapser")
2  public CommonResult testCollapser() throws ExecutionException,
    InterruptedException {
3      Future<User> future1 = userService.getUserFuture(1L);
4      Future<User> future2 = userService.getUserFuture(2L);
5      future1.get();
6      future2.get();
7      ThreadUtil.safeSleep(200);
8      Future<User> future3 = userService.getUserFuture(3L);
9      future3.get();
10     return new CommonResult("操作成功", 200);
11 }Copy to clipboardErrorCopied
```

- 使用@HystrixCollapser实现请求合并，所有对getUserFuture的的多次调用都会转化为对getUserByIds的单次调用：

```
1  @HystrixCollapser(batchMethod = "getUserByIds",collapserProperties = {
2      @HystrixProperty(name = "timerDelayInMilliseconds", value = "100")
3  })
4  public Future<User> getUserFuture(Long id) {
5      return new AsyncResult<User>(){
6          @Override
7          public User invoke() {
8              CommonResult commonResult = restTemplate.getForObject(userServiceUrl +
"/user/{1}", CommonResult.class, id);
9              Map data = (Map) commonResult.getData();
10             User user = BeanUtil.mapToBean(data, User.class, true);
11             LOGGER.info("getUserById username:{0}", user.getUsername());
12             return user;
13         }
14     };
15 }
16
17 @HystrixCommand
18 public List<User> getUserByIds(List<Long> ids) {
19     LOGGER.info("getUserByIds:{0}", ids);
20     CommonResult commonResult = restTemplate.getForObject(userServiceUrl +
"/user/getUserByIds?ids={1}", CommonResult.class, CollUtil.join(ids, ","));
21     return (List<User>) commonResult.getData();
22 }Copy to clipboardErrorCopied
```

- 访问接口测试 <http://localhost:8401/user/testCollapser>，由于我们设置了100毫秒进行一次请求合并，前两次被合并，最后一次自己单独合并了。



Hystrix的常用配置

全局配置

```
1  hystrix:
2      command: #用于控制HystrixCommand的行为
3      default:
4          execution:
5              isolation:
6                  strategy: THREAD #控制HystrixCommand的隔离策略, THREAD->线程池隔离策略(默认), SEMAPHORE->信号量隔离策略
7                  thread:
8                      timeoutInMilliseconds: 1000 #配置HystrixCommand执行的超时时间, 执行超过该时间会进行服务降级处理
9                      interruptOnTimeout: true #配置HystrixCommand执行超时的时候是否要中断
10                     interruptOnCancel: true #配置HystrixCommand执行被取消的时候是否要中断
11                     timeout:
12                         enabled: true #配置HystrixCommand的执行是否启用超时时间
13                     semaphore:
14                         maxConcurrentRequests: 10 #当使用信号量隔离策略时, 用来控制并发量的大小, 超过该并发量的请求会被拒绝
15                     fallback:
16                         enabled: true #用于控制是否启用服务降级
17                     circuitBreaker: #用于控制HystrixCircuitBreaker的行为
18                         enabled: true #用于控制断路器是否跟踪健康状况以及熔断请求
19                         requestVolumeThreshold: 20 #超过该请求数的请求会被拒绝
20                         forceOpen: false #强制打开断路器, 拒绝所有请求
21                         forceClosed: false #强制关闭断路器, 接收所有请求
22                     requestCache:
23                         enabled: true #用于控制是否开启请求缓存
24             collapser: #用于控制HystrixCollapser的执行行为
25             default:
26                 maxRequestsInBatch: 100 #控制一次合并请求合并的最大请求数
27                 timerDelayInMilliseconds: 10 #控制多少毫秒内的请求会被合并成一个
28                 requestCache:
29                     enabled: true #控制合并请求是否开启缓存
30             threadpool: #用于控制HystrixCommand执行所在线程池的行为
31             default:
32                 coreSize: 10 #线程池的核心线程数
33                 maximumSize: 10 #线程池的最大线程数, 超过该线程数的请求会被拒绝
34                 maxQueueSize: -1 #用于设置线程池的最大队列大小, -1采用SynchronousQueue, 其他正数采用LinkedBlockingQueue
35                 queueSizeRejectionThreshold: 5 #用于设置线程池队列的拒绝阈值, 由于LinkedBlockingQueue不能动态改版大小, 使用时需要用该参数来控制线程数Copy to clipboardErrorCopied
```

实例配置

实例配置只需要将全局配置中的default换成与之对应的key即可。

```

1  hystrix:
2    command:
3      HystrixComandKey: #将default换成HystrixComrnandKey
4      execution:
5        isolation:
6          strategy: THREAD
7    collapser:
8      HystrixCollapserKey: #将default换成HystrixCollapserKey
9      maxRequestsInBatch: 100
10   threadpool:
11     HystrixThreadPoolKey: #将default换成HystrixThreadPoolKey
12     coreSize: 10Copy to clipboardErrorCopied

```

配置文件中相关key的说明

- HystrixComandKey对应@HystrixCommand中的commandKey属性;
- HystrixCollapserKey对应@HystrixCollapser注解中的collapserKey属性;
- HystrixThreadPoolKey对应@HystrixCommand中的threadPoolKey属性。

使用到的模块

```

1  springcloud-learning
2  |—— eureka-server -- eureka注册中心
3  |—— user-service  -- 提供User对象CRUD接口的服务
4  |—— hystrix-service -- hystrix服务调用测试服务

```