参考文档：

<http://www.mamicode.com/info-detail-2463285.html>

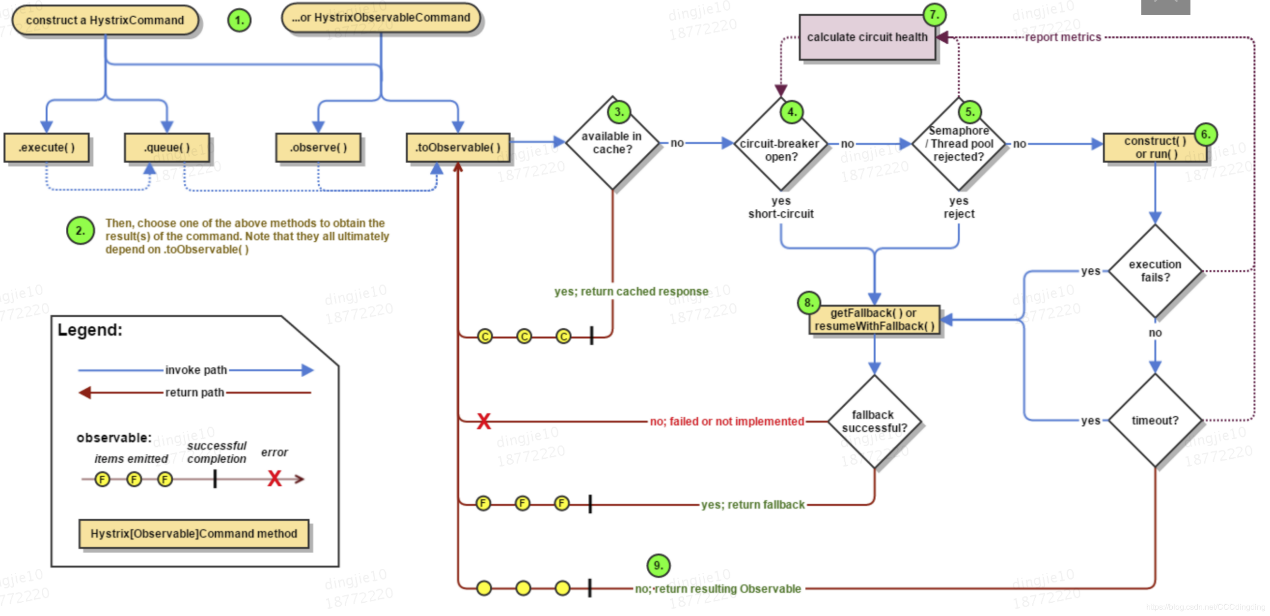
<https://cloud.tencent.com/developer/article/1384049>

<https://segmentfault.com/q/1010000016370993?utm_source=tag-newest>

<https://blog.csdn.net/liyingblog/article/details/84785796> 路由+熔断

<https://www.e-learn.cn/content/qita/2357712>

分布式系统中经常会出现某个基础服务不可用造成整个系统不可用的情况, 这种现象被称为服务雪崩效应



四种情况会调用Fallback

（1）构建一个command，command 有两种模式：一种是正常的线程模式，另外一种是观察者模式。

（2）执行command 命令，执行command有四种方式：execute（以同步阻塞方式执行command的

run方法）、queue（以异步方式执行command的run方法）、observe（事件注册前执行run/construct）、

toObservable（事件注册前执行run/construct）

（3）缓存命中验证：验证当前请求是否命中缓存，如果命中缓存直接返回命中结果。

（4）熔断开关判定：打开直接进去command的回退逻辑

（5）信号量或者线程池容量判定：判断当前的command对应的请求数量是否超过最大信号量或者线程池

阈值，超过则进入异常处理逻辑fallback 。否则则执行command逻辑，并重新计算熔断值。

（6）执行command逻辑：HystrixObservableCommand.construct() 、HystrixCommand.run()

（7）计算熔断的健康状态

（8）异常回退处理

（9）返回command执行结果

zuul路由熔断

实现FallbackProvider接口，然后注册到实例到springcontext就好了

|  |
| --- |
| @Component public class GatewayFallbackProvider implements FallbackProvider {   private final Logger log = LoggerFactory.*getLogger*(GatewayFallbackProvider.class);   @Override  public String getRoute() {  return "\*";  }   @Override  public ClientHttpResponse fallbackResponse(String route, Throwable cause) {  return new ClientHttpResponse() {  @Override  public HttpStatus getStatusCode() throws IOException {  return HttpStatus.*GATEWAY\_TIMEOUT*;  }   @Override  public int getRawStatusCode() throws IOException {  return 0;  }   @Override  public String getStatusText() throws IOException {  return "接口响应时间大于熔断时间，触发熔断";  }   @Override  public void close() {  RequestContext requestContext = RequestContext.*getCurrentContext*();  String servletPath = requestContext.getRequest().getServletPath();  log.info("servletPath {}", servletPath);  Object ribbonResponse = requestContext.get("ribbonResponse");  if (ribbonResponse instanceof RibbonApacheHttpResponse) {  RibbonApacheHttpResponse response = (RibbonApacheHttpResponse)ribbonResponse;  String uri = response.getRequestedURI().toString();  log.info("服务 {} 被熔断", uri);  }  }   @Override  public InputStream getBody() throws IOException {  RequestContext requestContext = RequestContext.*getCurrentContext*();  String servletPath = requestContext.getRequest().getServletPath();  JsonObject jsonObject = new JsonObject();  jsonObject.addProperty("timestamp", LocalDateTime.*now*().toString());  jsonObject.addProperty("status", getStatusCode().value());  jsonObject.addProperty("error", getStatusText());  jsonObject.addProperty("message", "service[" + servletPath + "] response time beyond the gateway time");  return new ByteArrayInputStream(jsonObject.toString().getBytes());  }   @Override  public HttpHeaders getHeaders() {  HttpHeaders headers = new HttpHeaders();  headers.setContentType(MediaType.*APPLICATION\_JSON*);  return headers;  }  };  } } |

可以看到zuul的熔断器主要目的和意义在于针对某个微服务的异常状态进行控制，并不能具体的针对某个具体的请求方法进行控制，譬如我的请求需要关联商品、用户、订单三个微服务，需要三个服务的返回值组合成一个结果返回给用户，那么这个熔断器就不能做出合适的处理了。当然了，需要关联多个微服务的请求，我们是采用feign来完成。feign里也自带的有hystrix，可以更精细地控制某个微服务挂掉后通过熔断的回调赋予默认值，然后用默认值来组合结果，来保证即便挂掉一个服务，其他的服务还能正常工作时的用户请求不会无响应。

#zuul集成hystrix默认1s.这里如果不设置超时时间就会报错

ribbon:

ReadTimeout: 1000

ConnectTimeout: 500

#熔断机制

hystrix:

command:

default:

execution:

isolation:

thread:

timeoutInMilliseconds: 25000