Bipin Kumar

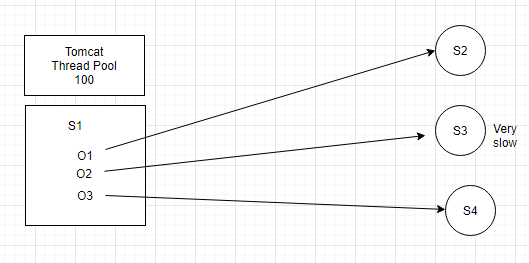
Shael.dhn88@gmail.com

Hystrix

Hystrix setup

Bulk Head pattern:

Let take an example we have following server S1, S2, S3, S4 etc. And S1 have operation O1, O2 and O3. And S3 service is very slow. We may deploy those service to tomcat or other server and it has some thread pool. Suppose S1 is deployed to tomcat server and its thread pool size is 100. So, what will happen if O2 calls S3 service, due to slowness it might be waiting for response. Similarly, there might be chance that all 100 threads are calling to S3 and due to slowness is waiting for response and our application consume all the thread pool and no thread is available for other operation.

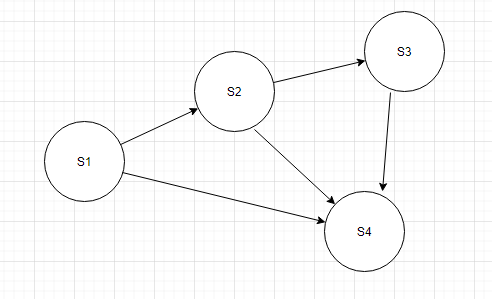


Now take example of bank having toll free no. with 100 limits. Suppose it has many departments like saving account, loan, credit card etc. When caller call the toll, free no. based on it’s choice call connected to different department. Suppose Loan department taking time to describe loan feature and if we don’t have any mechanism then all the 100 limits get used. So, what we can do we can assign some limit to each department like SA can have 30 call max, loan can have 40 calls max and CC can have 30 calls max. So, what will happen in this case if more than 40 callers try to connect to loan department either they will wait or call get disconnected after sometime automatically. By this here we can utilize limits. Means 60 call will be available for another department.

Similarly, in above case of microservice we can limit pool size specific to each service. This pattern is call Bulk head pattern. Means issue in one service could not make entire system down. This concept came from ship companies. We don’t need to implement bulk head pattern Netflix provide one more library called Hystrix which implement bulk head pattern. It also used in circuit breaker pattern.

Circuit breaker pattern.

Look below diagram, S1 calling S2, S2 calling S3 and S3 calling S4, S2 calling S4 and S1 calling S4. By the looks of this it looks like a circuit.



Here suppose due to some reason Service S4 fails, what will happen then after sometime S3 fails then S2 and then complete services.

One thing we can do we can write some fallback logic which will execute once S4 is down. Now suppose S4 is down and S3 is calling S4 1,2… 20 times it got failure and it execute fallback logic. Suppose if we have mechanism which will already identify that S4 is down without wasting time calling S4 it should execute fallback logic. In circuit breaker there are lots of logic which decide whether a circuit is open or close based. E.g. success count, failure count, time out, bucket with data all SC, FC, TC of 100sec etc.

Hystrix:

1. Add below dependency in booking service.

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>

</dependency>

1. Go to main class and add @EnableHystrix
2. We have created one interface and its implementation and added below annotation on implemented method to enable hystrix. CargoRoutingServiceImpl, CargoRoutingService

@HystrixCommand(fallbackMethod = "findOptimalRouteFallback",

commandProperties = {@HystrixProperty(name="execution.timeout.enabled", value="false")})

1. The above service is called from ExternalCargoRoutingService, where we autowired the service class of CargoRoutingService. Also change the logic to from where externalcoargoRoutingservice called.
2. Now run eureka and booking service then create a booking using swagger-ui.html. and use generated id to route the cargo. When you try to route the cargo, you will get server error because our fall back logic called. You can see in the logs.

Hystrix internal logic:

Hystrix internally using AOP. Here we have annotated one of method of CargoRoutingServiceImpl.java with hystrix annotation. What it will do? It will automatically generate its proxy class and inject this proxy class to ExternalCargoRoutingService. When a call come to ECRS class it will call proxy class which create a hystrix dynamic command object and it call hc.execute(). This method calls circuit breaker and in this it calls method isOpen. Suppose if isOpen return false in that case hystrix command will call the actual class rest api call. Suppose it return true then in that case it directly executes fallback logic.

<https://github.com/Netflix/Hystrix/wiki/Configuration>

Open above URL you will find out different properties we can configured in hystrix.

You can configure hystrix property like we did in above example

commandProperties = {@HystrixProperty(name="execution.timeout.enabled", value="false"),

@HystrixProperty(name="circuitBreaker.errorThresholdPercentage", value="40")}

In above case we need to write this annotation every where we want to use. But suppose if we want to change in future it will be pain full task. So instead of defining it to method level we can define it to configuration file i.e. yml or property file.

If all service use same value then hystrix.command.default.circuitBreaker.errorThresholdPercentage

If all service use different value then we can configure using command key like below

@HystrixCommand(fallbackMethod = "findOptimalRouteFallback",commandKey = "s2ckey",

And in configuration file we configure like below:

hystrix.command.s2ckey.circuitBreaker.errorThresholdPercentage

By default, its execution is thread which is bulk head pattern i.e. thread pool. but suppose you don’t want to use thread as execution pattern you can change it by using below properties.

execution.isolation.strategy

**Monitoring multiple circuit using hystrix dashboard.:**

I want to monitor all the metric, like thread pool count, sleep time, threshold etc. We can implement hystrix dashboard. We all know using actuator we can get all the matrix for the application.

1. Add below dependency in pom.xml

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>

</dependency>

1. Add below annotation in main application class

@EnableHystrixDashboard

1. Application which needs to be monitored. Add actuator dependency.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

1. Add below property

management:

endpoints:

web:

exposure:

include: hystrix.stream

1. Add below property in dashboard application.

spring:

application:

name: hystrix-dashboard

<http://localhost:5555/hystrix>

<http://localhost:8080/actuator/hystrix.stream>

and click on Monitor.

If we get too many time isOpen by default hystrix command will execute fallback logic without making call to actual api call.

Turbine it is eureka client which is used to gather all the matrix of application.

Add below dependency in dashboard application.

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-turbine</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

Enable below annotation:

@EnableTurbine

@EnableDiscoveryClient

turbine:

app-config: first-service,second-service

cluster-name-expression: new String("default")

eureka:

client:

service-url:

defaultZone: http://localhost:5001/eureka

Use below url in hystrix dashboard.

<http://localhost:8080/turbine.stream>