Spring Cloud Hystrix

In real-world scenario, there are multiple microservices service a single business application, these micro services are requited to work and remain up round the clock to server the business purpose.

Though, how many instances we scale up, how much strong infrastructural we set up, there are still chances of failures that are beyond our control.

In such scenario, Spring cloudHystrix, acts as a saviour, for the microservices. Its a fault-tolerance library

If a micro service is enabled with the Hystrix Circuit breaker, it will look for faults and failures in the downstream systems, and external dependencies of the microservices during real-time.

On identifying the failures, it will fall back to the second option that we have specified as backup. we have an option to further specify the fall back for the secondary backup as well.

Here, a fall back is a method serving from the Cache, another service or system to gracefully handle the failure scenario.

For Example:

A microservice, interacts with a downstream system or rest API to fecth the data from the Mongo DB. The same data is stored in the main frame as a backup. we can enable a hystrix circuit breaker for this API.

Incase there is a failue in the rest API that is getting the data from the Mongo DB, the hystrix will detect the failure and the transfer the control to the fall back i.e. the mainframe system to fetch the data. This way the microservices behaves as expected without giving any failure indication to the user. It continous operating when a method call fails and prevent the failure from cascading and giving faulty service(the primary one) time to recover.

In worst scenario, when the secondary fall back also failed, then there is the **fail-fast** method, gracefully handles or throws the business expection to the user. Eeven a cache can be put in place to return the data that is stored from the previous responses. Static data , empty response or stubbed response can also be returned. Depending on the business requirement this can be customized and designed.

Narsi

Command Pattern

Hystrix is built on top of the command pattern. This is annototated @HystrixCommand when applied ton the method having a third party or downstream systme call, wraps that method, and looks for the faults in those external calls and sends the control to the secondary method. Commands are indentitified, by keys and are organized in groups.

Hystrix Circuit

There is a concept of the **open circuit and closed circuit.** ***Closed-Circuits indicates everything is working as expected***. incase of failues, firstly , the primary method is checked, if it fails, then controll transfers to the secondary method.

During the failure, this happens for every request untill the request volumes threshold(defaults 20 requests) and rolling window(defaults 10 seconds) as specified by us in the hystrix properties or default values not reached.

Once these properties threshold is reached, the hystrix opens the circuit. During the time circuit is open, all the requests are directed to the secondary method, skipping the primary method to avoid latency in response that could occur due to the checking the primary method. This will happens until the sleep window time is up, again requests first directed to the primary method, if the primary method worked, hystrix marks the circuit closed, otherwise , control directs to the second method and the circuit remains open.

Narsi

Hystix Properties

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

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

so for all 20 requests, it will first go to the prmary method and then falls back to the second method during failues. Default value-20

sleepWindowInMilliseconds:

Amount of time, after tripping circuit, to reject requests to the primary method before allowing attempts again to determine if the circuit should again be closed. default value- 5000 ms.

timeoutInMilliseconds:

After a specified time passed and did not recieve the response, mark the execution timeout and perform the fallback login. Default value is -1000ms.

This means, if the response didnt return from the downstream system in 1 seconds, the hystrix will timeout the request and controll directs to the second method. These are the multiple properties that could be used as per the requirements and given at the link: https://github.com/Netflix/Hystrix/wiki/Configuration

Hystix Thread Pool:

Hystrix has a thread pool per downstream dependency and manages the thread pool. There are two isolation strategies- Semaphore and Thread.

To isolate each call on a seperate thread ad keeps away from impacting others, Thread strategy is uused. There is some minimal cost incurred of extra execution overhead with a hystrix thread pool. The thread pool properties like core pool size, and maximum queue size allowing the thread pool configuration.

coreSize: sets the core thread-pool size. Default Value is 10.

maxQuesize: sets the maximum queue size, of the Blocking QUEUE implementation. if we set this to -1, then synchronous queue will be used, otherwise a positive value will be used with LinkedBlockingQueue.

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

To change between synchronous queue and LinkedBlockingQueue requires a restart.

default Value: -1.

Narsi

Spring **Cloud**

Narsi