Hystrix Circuit Breaker

Overview

This document is a quick guide to implement Hystrix Circuit breaker for Spring Boot Micro services.

Circuit Breaker is a well-recognized design pattern used to detect failures and encapsulates the logic of preventing a failure from constantly recurring, during maintenance, temporary external system failure or unexpected system difficulties. The basic idea behind the circuit breaker is very simple. You wrap a protected function call in a circuit breaker object, which monitors for failures. Once the failures reach a certain threshold, the circuit breaker trips, and all further calls to the circuit breaker return with an error, without the protected call being made at all.

“What defines us is how well we rise after falling”

Implementation

Step 1: **Download Dependency jar – POM.xml**

There are various versions of Hystrix plugins available, the latest and Spring supported

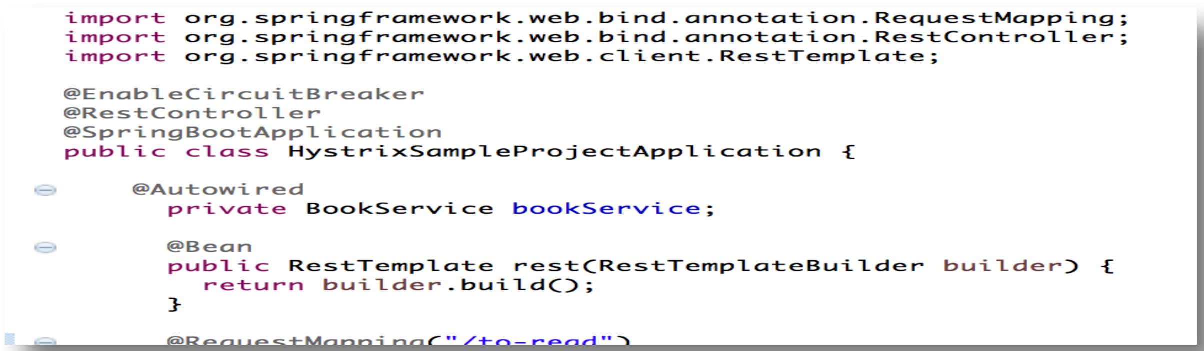
Maven Plugin:

```
<dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId> spring-cloud-starter-netflix-hystrix</artifactId>
      <version>1.4.2.RELEASE</version>
</dependency>
```

The below jar file is needed only in the case if there is an error in compilation as "*build path is incomplete. cannot find class file rx/Observable*"

```
<!-- https://mvnrepository.com/artifact/io.reactivex.rxjava2/rxjava -->
<dependency>
    <groupId>io.reactivex.rxjava2</groupId>
    <artifactId>rxjava</artifactId>
    <version>2.1.8</version>
</dependency>
```

Step 2: **Marker Annotation @EnableCiruitBreaker**



```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@EnableCircuitBreaker
@RestController
@SpringBootApplication
public class HystrixSampleProjectApplication {

    @Autowired
      private BookService bookService;

      @Bean
      public RestTemplate rest(RestTemplateBuilder builder) {
        return builder.build();
      }

      @RequestMapping("/to-read")
```

To work with a Circuit Breaker in your application must include the **@EnableCircuitBreaker** annotation on a configuration class.

***Step 3:* CB client implementation**

To apply a circuit breaker to a method, annotate the method with @HystrixCommand, giving the annotation the name of a fallbackMethod.

```
@HystrixCommand(fallbackMethod = "reliableFallbackMethod")
public String retireveListFromBookService(){
   return restTemplate.getForObject("http://company/available", String.class);
}
```

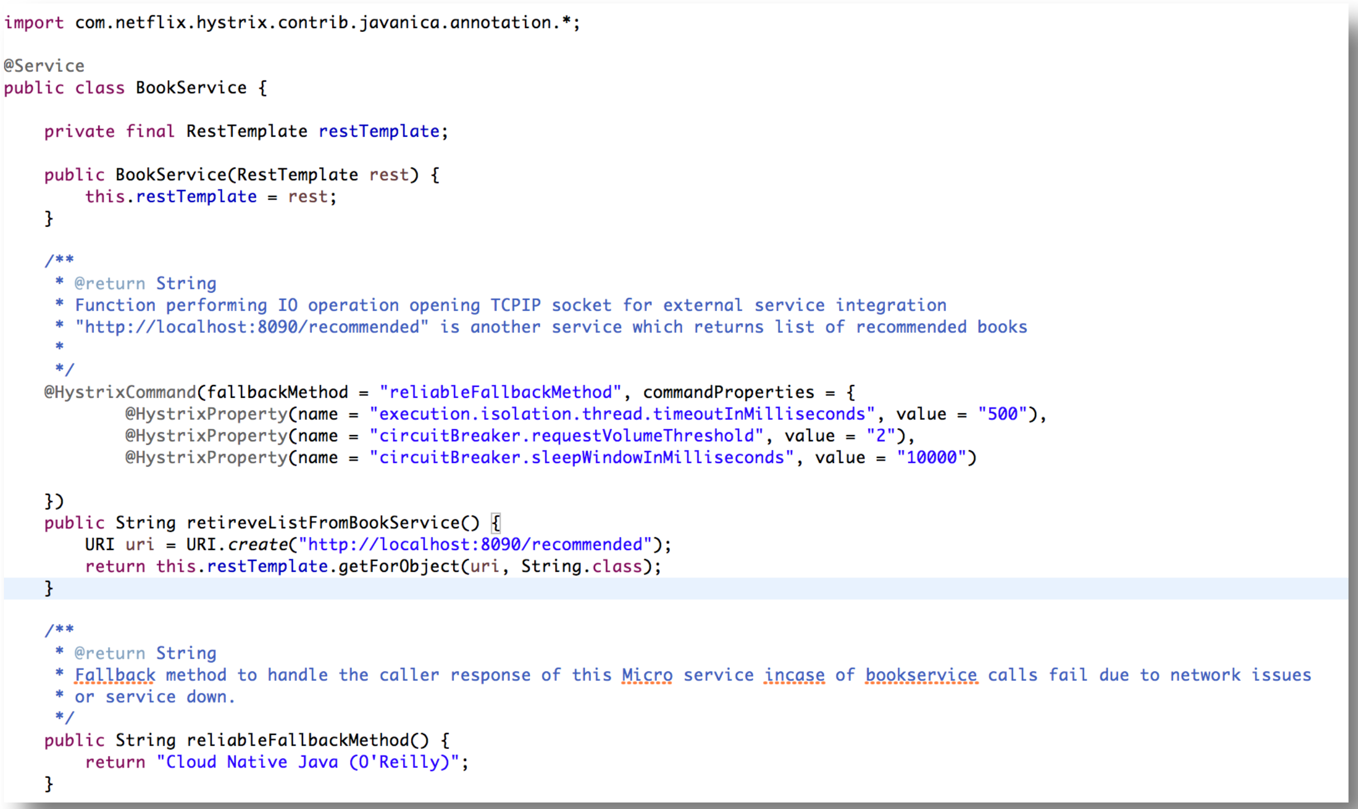
The **retireveListFromBookService ()** method uses a Rest Template to obtains data from another application called Company, which is registered with a Service Registry instance. The method thus relies on the Company application to return a response, and if the Company application fails to do so, calls to **retireveListFromBookService()** will fail. When the failures exceed the threshold, the breaker on **retireveListFromBookService()** will open the circuit.

While the circuit is open, the breaker redirects calls to the annotated method, and they instead call the designated fallbackMethod. The fallback method must be in the same class and have the same method signature (i.e., have the same return type and accept the same parameters) as the annotated method. In the sample application, the **retireveListFromBookService**() method in the class falls back to **reliableFallbackMethod**().

```
public String reliableFallbackMethod() {
            return "Cloud Native Java (O'Reilly)";
      }
```

Important properties to be customized for every implementation based on their needs are below

- **@HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "2000"),**
- **@HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "3"),**
- **@HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value = "10000")**


```java
import com.netflix.hystrix.contrib.javanica.annotation.*;

@Service
public class BookService {

    private final RestTemplate restTemplate;

    public BookService(RestTemplate rest) {
        this.restTemplate = rest;
    }

    /**
     * @return String
     * Function performing IO operation opening TCPIP socket for external service integration
     * "http://localhost:8090/recommended" is another service which returns list of recommended books
     *
     */
    @HystrixCommand(fallbackMethod = "reliableFallbackMethod", commandProperties = {
            @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "500"),
            @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "2"),
            @HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value = "10000")

    })
    public String retireveListFromBookService() {
        URI uri = URI.create("http://localhost:8090/recommended");
        return this.restTemplate.getForObject(uri, String.class);
    }

    /**
     * @return String
     * Fallback method to handle the caller response of this Micro service incase of bookservice calls fail due to network issues
     * or service down.
     */
    public String reliableFallbackMethod() {
        return "Cloud Native Java (O'Reilly)";
    }
```

For more Command Properties refer below list:

1. Command Properties
   i. Execution
      a. `execution.isolation.strategy`
      b. `execution.isolation.thread.timeoutInMilliseconds`
      c. `execution.timeout.enabled`
      d. `execution.isolation.thread.interruptOnTimeout`
      e. `execution.isolation.thread.interruptOnCancel`
      f. `execution.isolation.semaphore.maxConcurrentRequests`
   ii. Fallback
      a. `fallback.isolation.semaphore.maxConcurrentRequests`
      b. `fallback.enabled`
   iii. Circuit Breaker
      a. `circuitBreaker.enabled`
      b. `circuitBreaker.requestVolumeThreshold`
      c. `circuitBreaker.sleepWindowInMilliseconds`

   d. `circuitBreaker.errorThresholdPercentage`
   e. `circuitBreaker.forceOpen`
   f. `circuitBreaker.forceClosed`

iv. Metrics
   a. `metrics.rollingStats.timeInMilliseconds`
   b. `metrics.rollingStats.numBuckets`
   c. `metrics.rollingPercentile.enabled`
   d. `metrics.rollingPercentile.timeInMilliseconds`
   e. `metrics.rollingPercentile.numBuckets`
   f. `metrics.rollingPercentile.bucketSize`
   g. `metrics.healthSnapshot.intervalInMilliseconds`

v. Request Context
   a. `requestCache.enabled`
   b. `requestLog.enabled`

2. Collapser Properties
   i. `maxRequestsInBatch`
   ii. `timerDelayInMilliseconds`
   iii. `requestCache.enabled`
3. Thread Pool Properties
   i. `coreSize`
   ii. `maximumSize`
   iii. `maxQueueSize`
   iv. `queueSizeRejectionThreshold`
   v. `keepAliveTimeMinutes`
   vi. `allowMaximumSizeToDivergeFromCoreSize`
   vii. `metrics.rollingStats.timeInMilliseconds`
   viii. `metrics.rollingStats.numBuckets`

If you wish, you may also annotate fallback methods themselves with @HystrixCommand to create a fallback chain. A detailed feature of sync / Async, multiple fall back ,default fall back etc. are explained in the below reference link. Please browse.

Reference:

- *https://github.com/Netflix/Hystrix/tree/master/hystrix-contrib/hystrix-javanica#configuration*
- *https://spring.io/guides/gs/circuit-breaker/*
- *https://martinfowler.com/bliki/CircuitBreaker.html*