**RateLimitter in SpringBoot with Resilience4J-2024**

A rate limiter is a mechanism used to control the frequency of operations in a system. It sets a cap on how often a specific action can be performed within a given timeframe. This helps prevent server overload, reduce strain on resources, and mitigate certain types of malicious activities like brute force attacks or denial-of-service (DoS) attacks.

**Rate Limiting**: Rate limiting defines how often an operation is allowed in a specified duration. This avoids overloading the server and helps in preventing denial of service attacks.

**Bulkhead**: Bulkhead limits the number of **concurrent calls** at any given time. This helps in preventing system failures.

**Pom.xml Relevant portion**

<dependency>

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

<artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>

</dependency>

<dependency>

<groupId>io.github.resilience4j</groupId>

<artifactId>resilience4j-bulkhead</artifactId>

<version>2.2.0</version>

</dependency>

Main Class for SpringBoot application does not require any special annotation.

Application.properties

resilience4j.ratelimiter.instances.infoCall1.limit-for-period=5

resilience4j.ratelimiter.instances.infoCall1.limit-refresh-period=60s

resilience4j.ratelimiter.instances.infoCall1.timeout-duration=0s

# for BukHead

resilience4j.bulkhead.instances.courseBulkheadApi.max-concurrent-calls=3

resilience4j.bulkhead.instances.courseBulkheadApi.max-wait-duration=1

**Bried Explanation**

**limit-for-period**: This option specifies the maximum number of requests allowed within a specific time period. For example, if you set it to 10, it means that only 10 requests are allowed within that period.

**limit-refresh-period**: This option defines the duration of the time period mentioned above. It determines how frequently the rate limit will be reset. For instance, if you set it to 1 second, the rate limit will be reset every second, allowing the specified number of requests again.

**timeout-duration**: This option sets the timeout duration for requests. If a request exceeds this duration, it will be considered as taking too long and might trigger fallback behavior or an exception. Setting it to 3s means request will fail after three seconds if it did’t get a chance to execute.

**Sample Controller Class**

@RestController

**public** **class** TempController {

@RateLimiter(name = "infoCall1", fallbackMethod = "rateLimitingFallback")

@GetMapping(path = "/v1/info/{id}")

**public** ResponseEntity<String> getInfo(@PathVariable String id) {

System.***out***.println("Input ID: " + id);

**return** **new** ResponseEntity<>("Some Value", HttpStatus.***OK***);

}

**public** ResponseEntity<String> rateLimitingFallback(String id, RequestNotPermitted ex) {

HttpHeaders responseHeaders = **new** HttpHeaders();

responseHeaders.set("Retry-After", "60s"); // retry after one second

**return** ResponseEntity

.*status*(**HttpStatus.*TOO\_MANY\_REQUESTS***)

.headers(responseHeaders) // send retry header

.body("Too Many Requests - Retry After 1 Minute");

}

// For BulkHead

@GetMapping(value = "/course/{id}")

@Bulkhead(name = "courseBulkheadApi", fallbackMethod = "bulkheadFallback")

**public** ResponseEntity<String> getCourse(@PathVariable String id) {

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**return** **new** ResponseEntity<>("Some course", HttpStatus.***OK***);

}

**public** ResponseEntity bulkheadFallback(String id, BulkheadFullException ex) {

**return** ResponseEntity.*status*(HttpStatus.***BANDWIDTH\_LIMIT\_EXCEEDED***)

.body("Too many request - No further calls are accepted");

}

}

Example: If you hit the below URL 5 times, you will get **Too Many Requests - Retry After 1 Minute**

<http://localhost:8080/v1/info/123>