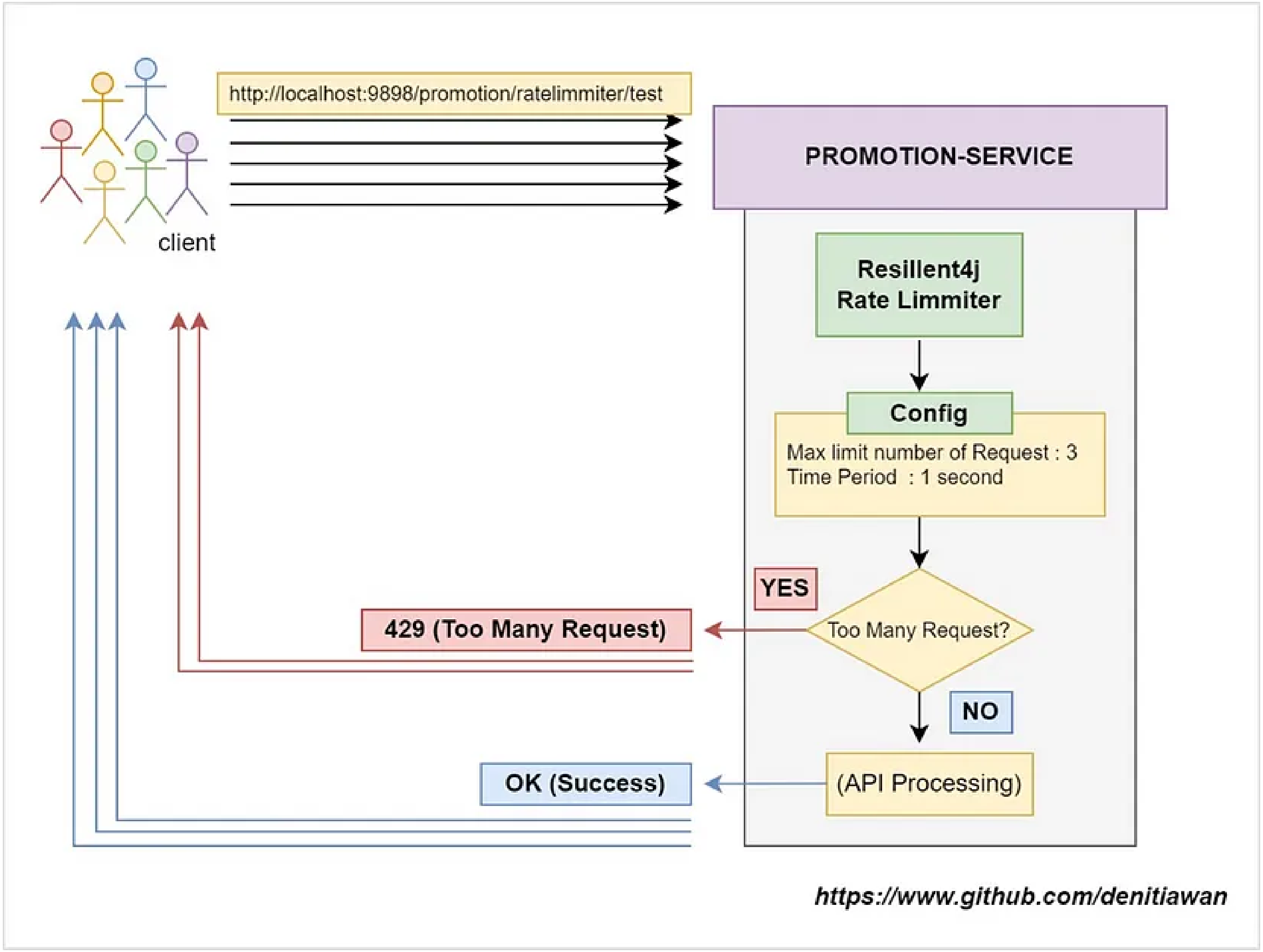
Basic Implement Rate Limmiter on Springboot 2.0.6

Search

Write


http://localhost:9898/promotion/ratelimmiter/test
client
PROMOTION-SERVICE
Resillent4j Rate Limmiter
Config
Max limit number of Request : 3
Time Period : 1 second
YES
Too Many Request?
429 (Too Many Request)
NO
(API Processing)
OK (Success)
https://www.github.com/denitiawan

Introduction

On this article, i want to share about ***how to implement Rate Limmiter on Springboot 2.0.6-RELEASE.*** that version was release since 2018.

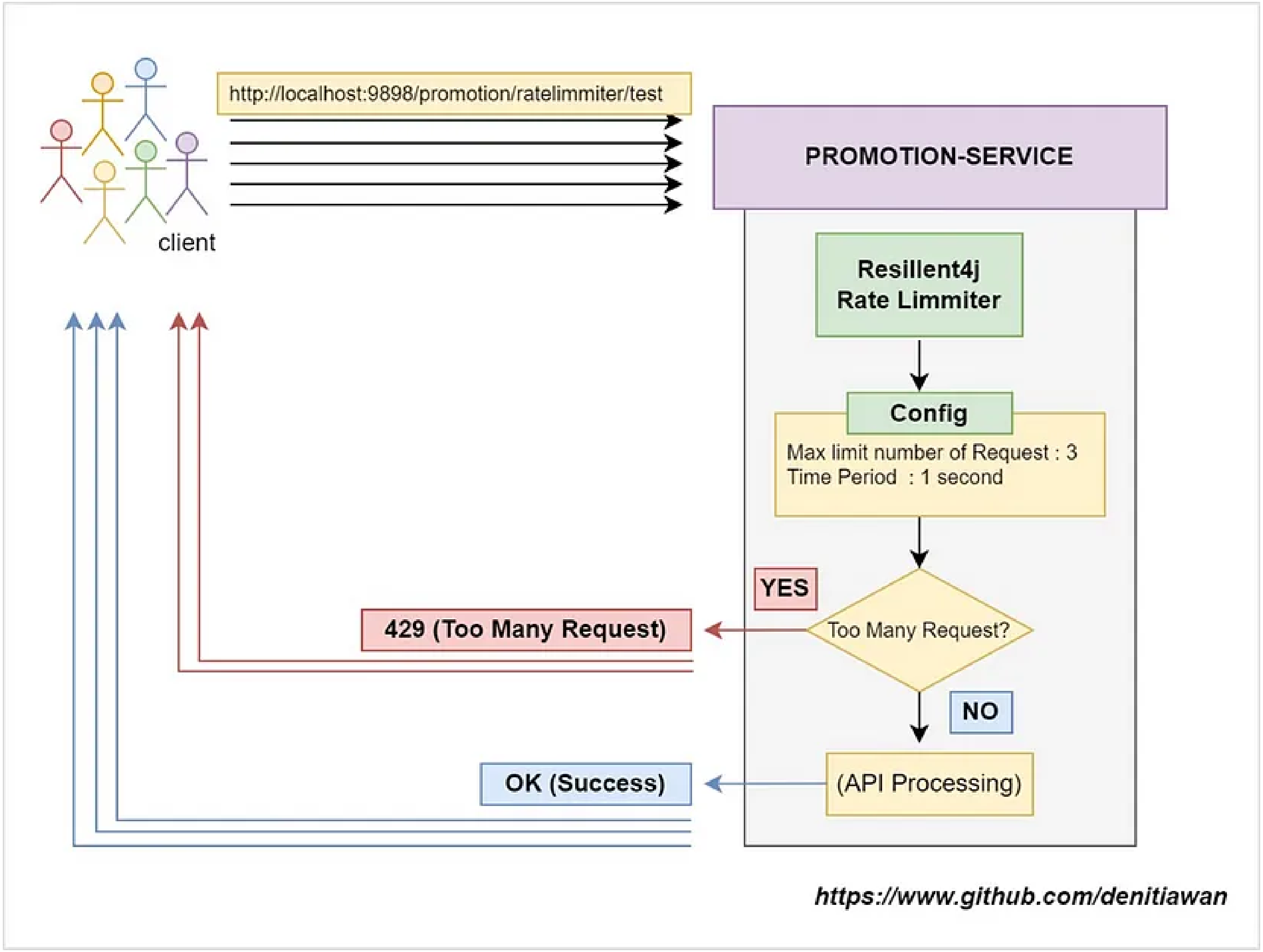
and for the java version is 1.8

so let’s to implement the **Rate Limmiter** on springboot project.

Overviews

1. Diagram
2. pom.xml
3. Application.yml
4. RateLimmiterConfig.java
5. RateLimmiter Annotation
6. Unitest

Diagram


http://localhost:9898/promotion/ratelimmiter/test
client
PROMOTION-SERVICE
Resillent4j Rate Limmiter
Config
Max limit number of Request : 3
Time Period : 1 second
YES
Too Many Request?
429 (Too Many Request)
NO
(API Processing)
OK (Success)
https://www.github.com/denitiawan

Description

- Client do 5 requests at the same times to one api
- backend will automatically check the **Rate Limmiter Config** is requested got the **Too Many Request status**

- Backend is only accept & process the requested who not got the **Too Many Requests status**

Pom.xml

We need add defendencies on **“pom.xml”** class, and then refresh the pom, for download the new librarries

```
<!--
 Release 2.x will be using Java 11
 Release 1.x is still using Java 8
 -->
  <dependency>
      <groupId>io.github.resilience4j</groupId>
      <artifactId>resilience4j-spring-boot2</artifactId>
      <version>1.3.1</version>
  </dependency>
```

Application.yml

Wee need add some config script for enable the **”Rate Limmiter”** .

- application.yml

```yaml
# ratelimmiter config
management:
  endpoints:
    web:
      exposure:
        include: "*"

  health:
    ratelimiters:
      enabled: true
```

RateLimmiterConfig.java

RateLimmiterConfig.java is an class for handling the rate limiter configuration. On this class we can setting

- **Limmit for periode**

This is for define 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.

- **Limmit refresh period**

This is for 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.

- **Time out duration**

This is for define 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.

- **Rate limmiter registry name**

This is for define the **"rate limiter name"** for mapping to rest controller class, the registry name can be global (one name for many endpoints) or (one name for one endpoint)

```
package com.deni.promotion.ratelimmiter;

import io.github.resilience4j.ratelimiter.RateLimiter;
import io.github.resilience4j.ratelimiter.RateLimiterConfig;
import io.github.resilience4j.ratelimiter.RateLimiterRegistry;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.time.Duration;
import java.time.temporal.ChronoUnit;


@Configuration
public class RateLimmiterConfig {
    @Autowired
    private RateLimiterRegistry rateLimiterRegistry;

    @Bean
    public RateLimiter globalRateLimiter() {
        RateLimiterConfig customConfig = RateLimiterConfig.custom()
                .limitForPeriod(10)
                .limitRefreshPeriod(Duration.of(1, ChronoUnit.SECONDS))
                .timeoutDuration(Duration.of(1, ChronoUnit.SECONDS))

                .build();

        return rateLimiterRegistry.rateLimiter("globalRateLimiter", customConfig
    }

}
```

Rate Limmiter Annotation

- name

Parameter **“name”** is for define Initial name for spesifiec api who want implement the **“rate limmiter”**

The **”globalRateLimmiter”** in this case, already created & define by “RateLimmiterConfig.java”

```
    @RateLimiter(name = "globalRateLimiter")
```

- **fallback**

Parameter **“fallback”** is for define an callback function, callback function will trigger when the rest api got httpstatus is (429 : Too Many Request), the (429) is default httpstatus from httpservlet and ratelimmiter exception handling

```
  @RateLimiter(fallbackMethod = "fallbackTest")
```

- example

```
package com.deni.promotion.ratelimmiter;

import com.deni.common.model.response.Response;
import io.github.resilience4j.ratelimiter.RequestNotPermitted;
import io.github.resilience4j.ratelimiter.annotation.RateLimiter;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import static com.deni.common.model.response.rest.ResponseHelper.error;

/**
 * @author https://www.github.com/denitiawan
 */
@Slf4j
@RestController
@RequestMapping("/ratelimmiter")
public class RateLimmiterTestController {

    final String url = "http://localhost:9999/promotion/ratelilmmiter/test";

    @RequestMapping(value = "/test", method = {RequestMethod.GET})
    @RateLimiter(name = "globalRateLimiter", fallbackMethod = "fallbackTest")
    public ResponseEntity<Response> test() {
        Response response = error(String.valueOf(HttpStatus.OK.value()), HttpSta
        return ResponseEntity.status(HttpStatus.OK).body(response);
    }


    public ResponseEntity<Response> fallbackTest(RequestNotPermitted exception)
        Response response = error(String.valueOf(HttpStatus.TOO_MANY_REQUESTS.va
        return ResponseEntity.status(HttpStatus.TOO_MANY_REQUESTS).body(response
        }

    }
```

Unitest

We can test the **rate limmiter, when the request got the maximum request,** likes what we already define on “RateLimmiterConfig.java” class.

See the example code below, for testing the rate limiter exception handling.

The test case of Rate limiter exception handling

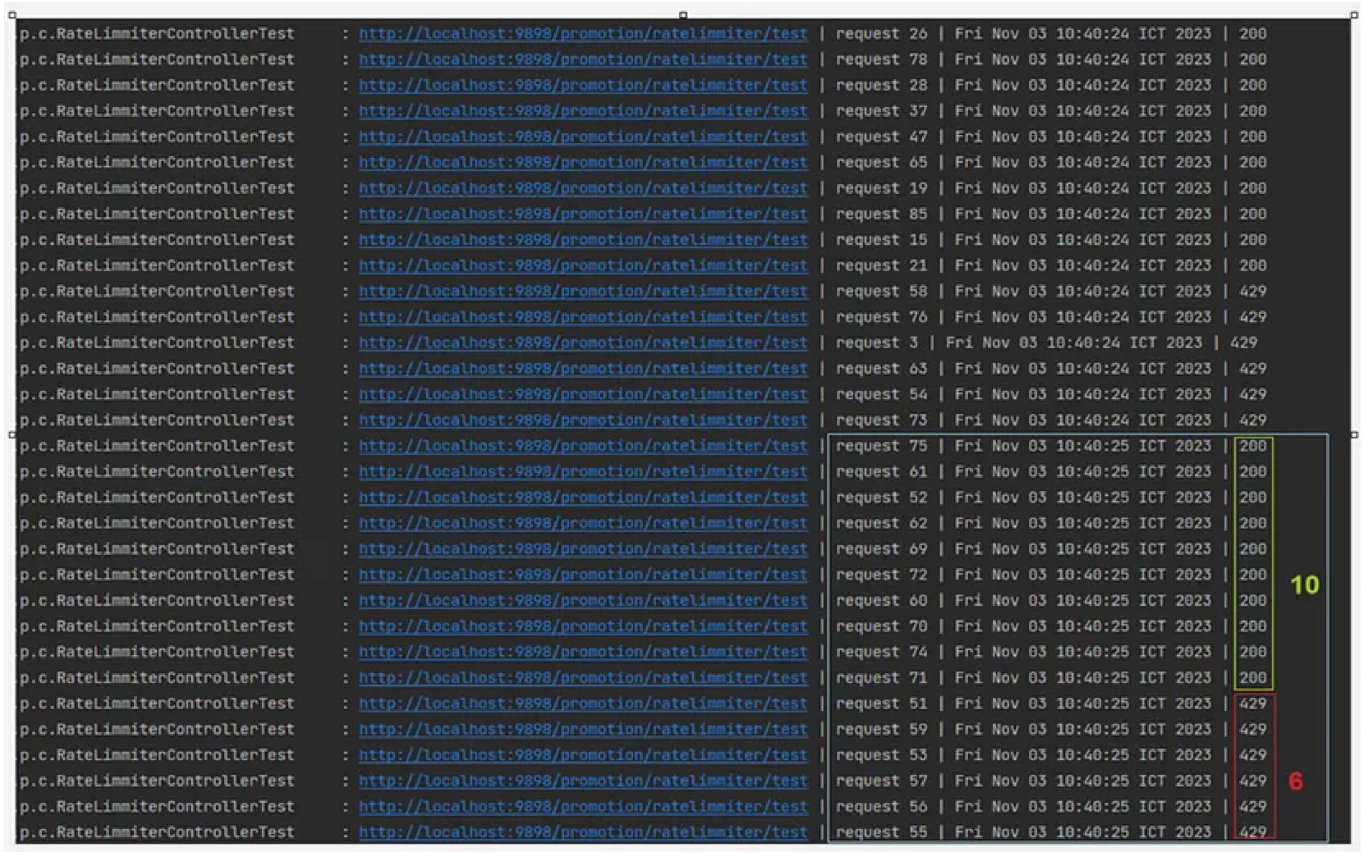
- Max Number of request = 10
- Time periode in second = 1
- Given number of request = 100
- Expected result = Too Many Request

```
    package com.deni.promotion.controller;
```

```java
import com.deni.promotion.gt.GTTestApp;
import com.deni.promotion.config.DataSourceTestConfig;
import lombok.extern.slf4j.Slf4j;
import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabas
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.RequestEntity;
import org.springframework.http.ResponseEntity;
import org.springframework.test.annotation.DirtiesContext;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.client.RestTemplate;

import java.net.URI;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.IntStream;


/**
 * @author https://www.github.com/denitiawan
 */
@Transactional
@RunWith(SpringRunner.class)
@SpringBootTest(classes = {
        GTTestApp.class,
        DataSourceTestConfig.class
}, webEnvironment = SpringBootTest.WebEnvironment.DEFINED_PORT)
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)
@DirtiesContext(classMode = DirtiesContext.ClassMode.AFTER_EACH_TEST_METHOD)
@Slf4j
```

```
public class RateLimmiterControllerTest {

    private RestTemplate restTemplate = new TestRestTemplate().getRestTemplate()


    /**
     * max number of request = 1
     * times periode in second = 1
     * given number of request = 1
     * expected = ok
     */
    @Test
    public void TEST_REQUEST_OK() {
        String uri = "http://localhost:9898/promotion/ratelimmiter/test";
        RequestEntity<Void> request = RequestEntity
                .get(URI.create(uri))
                .accept(MediaType.APPLICATION_JSON).build();

        ResponseEntity<String> response = restTemplate.exchange(request, String.
        Assert.assertEquals(HttpStatus.OK, response.getStatusCode());
    }


    /**
     * max number of request = 10
     * times periode in second = 1
     * given number of request = 100
     * expected = too many request
     */
    @Test
    public void TEST_TOO_MANY_REQUEST() {

        Map<Integer, Integer> responseCount = new HashMap<>();


        /**
         .rangeClosed(startInclusive, endInclusive)
         */
        IntStream.rangeClosed(1, 100)
                .parallel()
                .forEach(i -> {
                    String uri = "http://localhost:9898/promotion/ratelimmiter/t
                    RequestEntity<Void> request = RequestEntity
                            .get(URI.create(uri))
                            .accept(MediaType.APPLICATION_JSON).build();

                    ResponseEntity<String> response = restTemplate.exchange(requ

                    // masukan : http status, counter
                    responseCount.put(response.getStatusCodeValue(), i + 1);
                    log.info(uri + " | request " + i + " | " + new Date().toStri
                });

        // chek response had too many request (got rate limmiter)
        boolean expected = responseCount.containsKey(HttpStatus.TOO_MANY_REQUEST
        Assert.assertTrue(expected);

    }


}
```

Here the result after run the unitest



```
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 26 | Fri Nov 03 10:40:24 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 78 | Fri Nov 03 10:40:24 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 28 | Fri Nov 03 10:40:24 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 37 | Fri Nov 03 10:40:24 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 47 | Fri Nov 03 10:40:24 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 65 | Fri Nov 03 10:40:24 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 19 | Fri Nov 03 10:40:24 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 85 | Fri Nov 03 10:40:24 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 15 | Fri Nov 03 10:40:24 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 21 | Fri Nov 03 10:40:24 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 58 | Fri Nov 03 10:40:24 ICT 2023 | 429
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 76 | Fri Nov 03 10:40:24 ICT 2023 | 429
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 3 | Fri Nov 03 10:40:24 ICT 2023 | 429
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 63 | Fri Nov 03 10:40:24 ICT 2023 | 429
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 54 | Fri Nov 03 10:40:24 ICT 2023 | 429
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 73 | Fri Nov 03 10:40:24 ICT 2023 | 429
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 75 | Fri Nov 03 10:40:25 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 61 | Fri Nov 03 10:40:25 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 52 | Fri Nov 03 10:40:25 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 62 | Fri Nov 03 10:40:25 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 69 | Fri Nov 03 10:40:25 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 72 | Fri Nov 03 10:40:25 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 60 | Fri Nov 03 10:40:25 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 70 | Fri Nov 03 10:40:25 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 74 | Fri Nov 03 10:40:25 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 71 | Fri Nov 03 10:40:25 ICT 2023 | 200
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 51 | Fri Nov 03 10:40:25 ICT 2023 | 429
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 59 | Fri Nov 03 10:40:25 ICT 2023 | 429
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 53 | Fri Nov 03 10:40:25 ICT 2023 | 429
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 57 | Fri Nov 03 10:40:25 ICT 2023 | 429
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 56 | Fri Nov 03 10:40:25 ICT 2023 | 429
p.c.RateLimmiterControllerTest     : http://localhost:9898/promotion/ratelimmiter/test | request 55 | Fri Nov 03 10:40:25 ICT 2023 | 429
```

Description

- The 100 request hit with parallel mechanism
- See the blue box we focused on what happened on every 1 second on server
- On Fri Nov 03 10:40:25, got 16 request per 1 second
- But just 10 request is OK (200)

- And 6 request is Too Many Request (429)

So we can see the Rate limiter handling its work on api http://localhost:9999/promotion/ratelimmiter/test" .

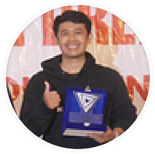
Conclusion

This article just **Basic how to implement RateLimmiter on sprinboot project.**

Thank you.

Happy Coding & Keep Learning 🚀

Spring Boot 2 Resilience4j Rate Limiter Rate Limiting



Written by Deni Setiawan

Follow

49 Followers

Backend & System Analyst 🚀

More from Deni Setiawan

Deni Setiawan

Create rest api for export data to excel and pdf using springboot

Springboot + Excel & PDF

Deni Setiawan

Create Rest API using Springboot for Searching data to Elastic...

Learn how to create Rest API using springboot for searching data to Elastic...

7 min read · Feb 14

13

8 min read · Jan 10

7

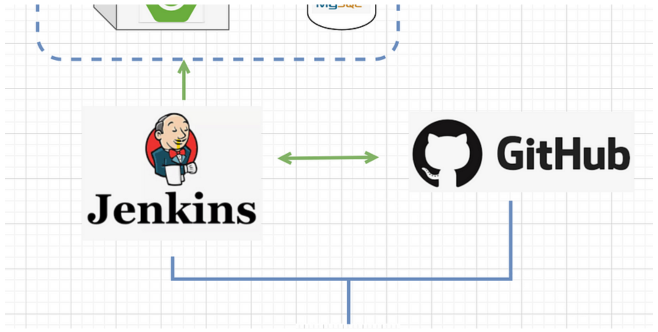
Deni Setiawan

Learn how to database migration using liquibase with MySql...

Learn how to database migration using liquibase with MySql Database in Springboo...

4 min read · Mar 1

1



Deni Setiawan

Deploy Springboot App using Jenkins

5 min read · May 3

3 1

See all from Deni Setiawan

Recommended from Medium

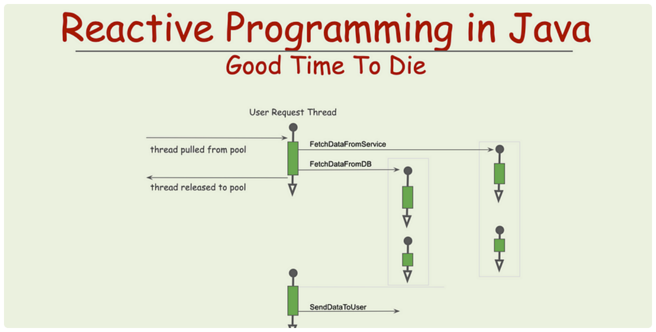
Raphael De Lio

Chaining Kafka and Database Transactions with Spring Boot: A...

In this article, we explore the complexities of handling transactions across Kafka and...

7 min read · Oct 28

54



Viraj Shetty

Reactive Programming in Java — Good Time to Die

This article explains the reason for Reactive Programming, why it is not popular with...

9 min read · Oct 12

228 9

Lists

Staff Picks

497 stories · 430 saves

Stories to Help You Level-Up at Work

19 stories · 293 saves

Self-Improvement 101

20 stories · 859 saves

Productivity 101

20 stories · 783 saves

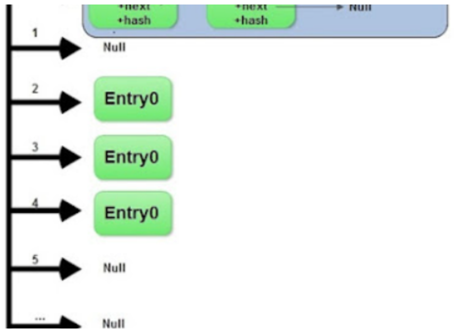
Hiten Pratap Singh in Javarevisited

Unlocking Precision Metrics in Spring Boot with Micrometer: A...

In the dynamic world of software development, measuring application...

6 min read · 6 days ago

81

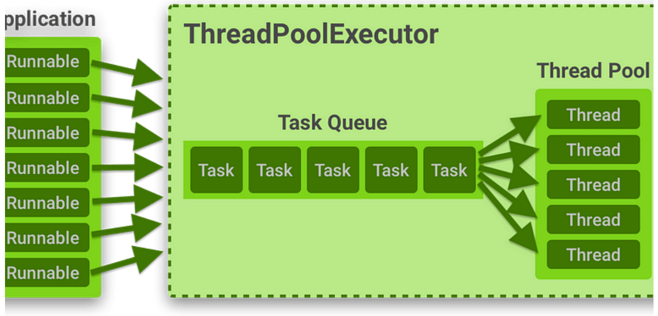


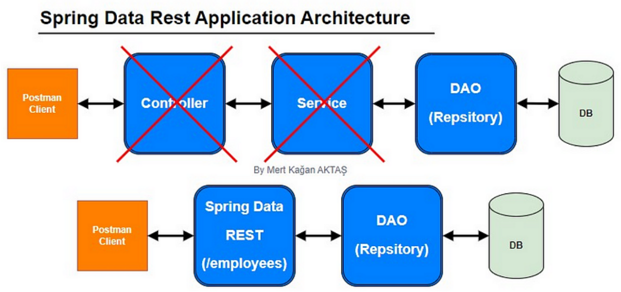
Vivek Kumar Gupta

Java Interview Questions Series-16

7 min read · Jul 4

2





The Java Trail

How to Determine Java Thread Pool Size: A Comprehensive Guide

Thread creation in Java incurs a noticeable cost. Creating threads consumes time, adds...

8 min read · Sep 18

119 1

Mert Kağan Aktaş

Spring Data REST: Say Goodbye to Controller and Service.

In the rapidly evolving landscape of software development, managing database access...

6 min read · Oct 31

501 13

See more recommendations

Help Status About Careers Blog Privacy Terms Text to speech Teams