服务熔断

某一台机器不太好，服务不可用，如何不用人工干预实行服务降级

极客时间

扫码试看/订阅

《玩转 Spring 全家桶》

使用 Hystrix 实现服务熔断

注意，Hystrix和Eureka一样，已经官方不再维护了！！！！

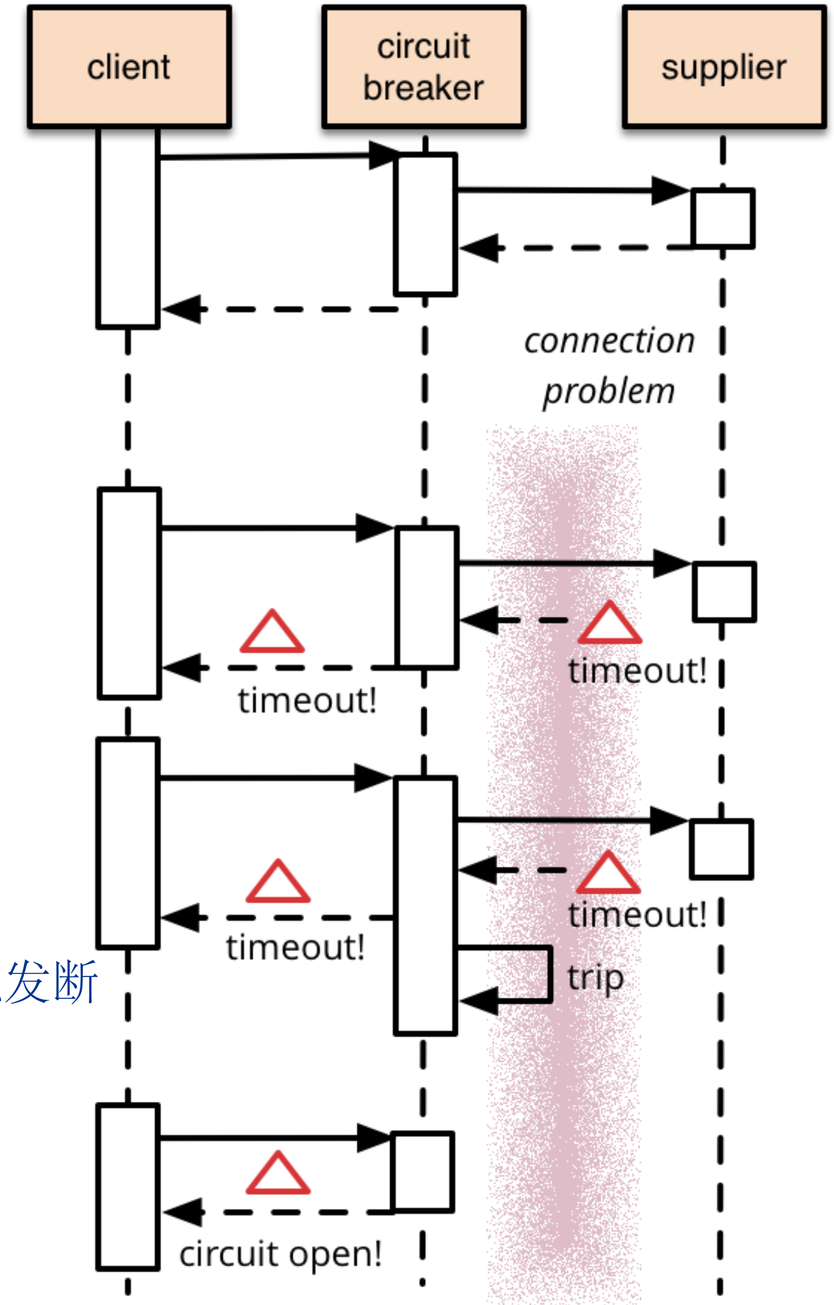
断路器模式

断路器

- Circuit Breaker pattern - Release It, Michael Nygard
- CircuitBreaker, Martin Fowler
  - https://martinfowler.com/bliki/CircuitBreaker.html

核心思想

- 在断路器对象中封装受保护的方法调用
- 该对象监控调用和断路情况 监控成功还是失败的，发生什么问题。触发短路保护后还会监控，埋点
- 调用失败触发阈值后，后续调用直接由断路器返回错误，不再执行实际调用

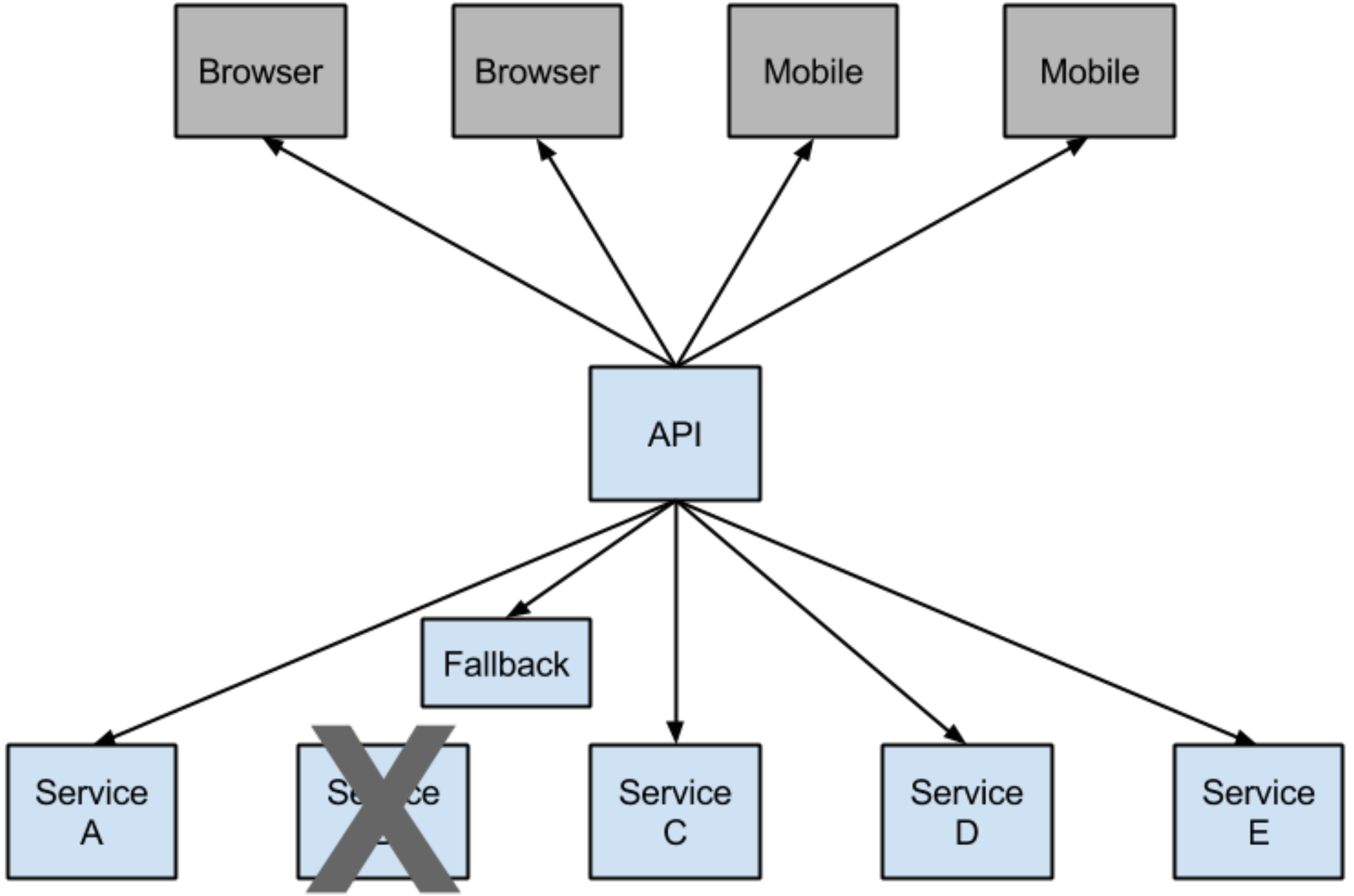


到达阈值后不再超时处理，直接触发断路保护

“Talk is cheap, show me the code.”

Chapter 13 / circuit-break-demo

Netflix Hystrix



Netflix Hystrix

- 实现了断路器模式
- @HystrixCommand
  - fallbackMethod / commandProperties
    发生问题时候转到指定方法　　指定属性
    - @HystrixProperty(name="execution.isolation.strategy", value="SEMAPHORE")　　默认到另一个线程执行
  - https://github.com/Netflix/Hystrix/wiki/Configuration

Netflix Hystrix

Spring Cloud 支持

```
@FeignClient(name = "waiter-service", contextId = "coffee",
        qualifier = "coffeeService", path="/coffee",
        fallback = FallbackCoffeeService.class)
// 如果用了Fallback，不要在接口上加@RequestMapping，path可以用在这里
```

- spring-cloud-starter-netflix-hystrix
- @EnableCircuitBreaker

Feign 支持

- feign.hystrix.enabled=true
- @FeignClient
  - fallback / fallbackFactory

获取异常，在factory中创建fallback对象

fallback到指定类

“Talk is cheap, show me the code.”

Chapter 13 / hystrix-customer-service

如何观察服务熔断

如何了解熔断的情况

打日志

- 在发生熔断时打印特定该日志

看监控

- 主动向监控系统埋点，上报熔断情况
- 提供与熔断相关的 Endpoint，让第三方系统来拉取信息

让第三方譬如普罗米修斯来拉取信息

单机 Hystrix Dashboard

Spring Cloud 为我们提供了

- Hystrix Metrics Stream
  - spring-boot-starter-actuator
    - /actuator/hystrix.stream
- Hystrix Dashboard
  - spring-cloud-starter-netflix-hystrix-dashboard
    - `@EnableHystrixDashboard` 配置类加上
    - /hystrix Localhost:9090/hystrix 然后把hystrix.stream的url配置到搜索框里

红色的就是Failure信息

极客时间

“Talk is cheap, show me the code.”

Chapter 13 / hystrix-stream-customer-service hystrix-dashboard-demo

聚合集群熔断信息

Spring Cloud 为我们提供了

把获取的聚合信息贴到hystrix dashboard的monitor里面

- Netflix Turbine
  - spring-cloud-starter-netflix-turbines
    - `@EnableTurbine`
    - /turbine.stream?cluster=集群名

去服务信息上获取所有服务的节点

聚合信息：
turbine.aggregator.cluster-config=customer-service
turbine.app-config=customer-service

这个依赖会依赖eureka,这里用consul

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-turbine</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

熔断信息输出到监控系统中，统一预警

“Talk is cheap, show me the code.”

Chapter 13 / turbine-demo

官方推荐

使用 Resilience4j 实现服务熔断

Hystrix 以外的选择

Hystrix

- Netflix 停止维护，给了官方推荐

Resilience4j

- https://github.com/resilience4j/resilience4j
- 一款受 Hystrix 启发的轻量级且易于使用的容错库　只有一个库，没有依赖别的库
- 针对 Java 8 与函数式编程设计

核心组件

组件名称	功能
resilience4j-circuitbreaker	Circuit breaking 熔断
resilience4j-ratelimiter	频率控制
resilience4j-bulkhead	依赖隔离&负载保护
resilience4j-retry	自动重试
resilience4j-cache	应答缓存
resilience4j-timelimiter	超时控制

附加组件

组件名称	功能
resilience4j-reactor	Spring Reactor 支持
resilience4j-micrometer	Micrometer Metrics 输出
resilience4j-prometheus	Prometheus Metrics 输出
resilience4j-spring-boot2	Spring Boot 2 Starter
resilience4j-feign	Feign 适配器

断路器

实现

- 基于 ConcurrentHashMap 的内存断路器
- CircuitBreakerRegistry
- CircuitBreakerConfig

依赖

- resilience4j-spring-boot2　自动引入如下依赖
  - resilience4j-circuitbreaker
  - resilience4j-micrometer
  - ……

断路器

注解方式

- @CircuitBreaker(name = "名称")

配置

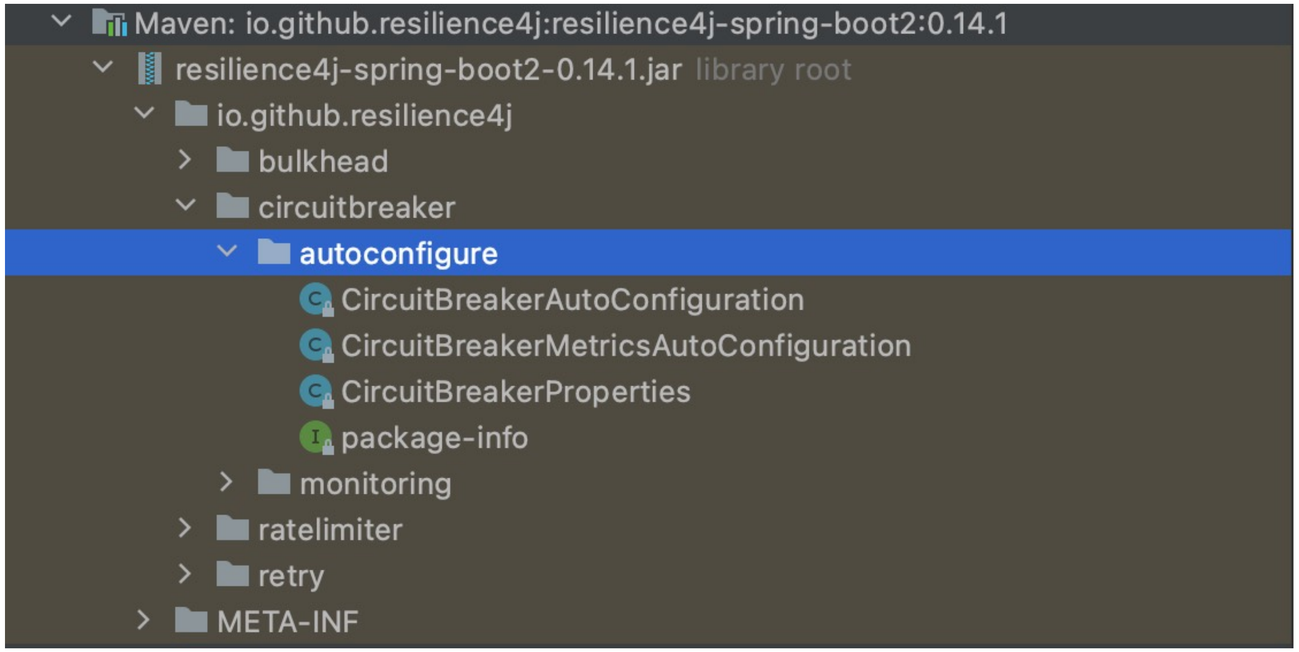
- CircuitBreakerProperties 配置文件

配置文件里加入：

  - resilience4j.circuitbreaker.backends.名称
    - failure-rate-threshold 失败率阈值
    - wait-duration-in-open-state 打开等多少时间

“Talk is cheap, show me the code.”

Chapter 13 / resilience4j-circuitbreaker-demo


Maven: io.github.resilience4j:resilience4j-spring-boot2:0.14.1
resilience4j-spring-boot2-0.14.1.jar library root
io.github.resilience4j
bulkhead
circuitbreaker
autoconfigure
CircuitBreakerAutoConfiguration
CircuitBreakerMetricsAutoConfiguration
CircuitBreakerProperties
package-info
monitoring
ratelimiter
retry
META-INF

使用 Resilience4j 实现服务限流

1.对下游调用并发度上控制，多余的请求进行排队，排队到一定时间后直接返回-bulkhead

2.一段时间段内只能接受多少请求，超过数目的只能等到下一个时间窗口，要么失败。

Bulkhead 这个控制并发的限流加在customer-service上

目的

- 防止下游依赖被并发请求冲击
- 防止发生连环故障

用法

- `BulkheadRegistry / BulkheadConfig` config类配置
- `@Bulkhead(name = "名称")` 注解方式

使用Registry方式：

```
public CustomerController(CircuitBreakerRegistry circuitBreakerRegistry,
                          BulkheadRegistry bulkheadRegistry) {
    circuitBreaker = circuitBreakerRegistry.circuitBreaker( name: "menu");
    bulkhead = bulkheadRegistry.bulkhead( name: "menu");
}

@GetMapping("/menu")
public List<Coffee> readMenu() {
    return Try.ofSupplier(
            Bulkhead.decorateSupplier(bulkhead,
                    CircuitBreaker.decorateSupplier(circuitBreaker,
                            () -> coffeeService.getAll())))
            .recover(CircuitBreakerOpenException.class, Collections.emptyList())
            .recover(BulkheadFullException.class, Collections.emptyList())
            .get();
}
```

Bulkhead

配置

- BulkheadProperties properties文件
  - resilience4j.bulkhead.backends.名称
    - max-concurrent-call 最大请求数量
    - max-wait-time 最大等待时间

```
resilience4j.circuitbreaker.backends.order.failure-rate-threshold=50
resilience4j.circuitbreaker.backends.order.wait-duration-in-open-state=5000
resilience4j.circuitbreaker.backends.order.ring-buffer-size-in-closed-state=5
resilience4j.circuitbreaker.backends.order.ring-buffer-size-in-half-open-state=3
resilience4j.circuitbreaker.backends.order.event-consumer-buffer-size=10
```

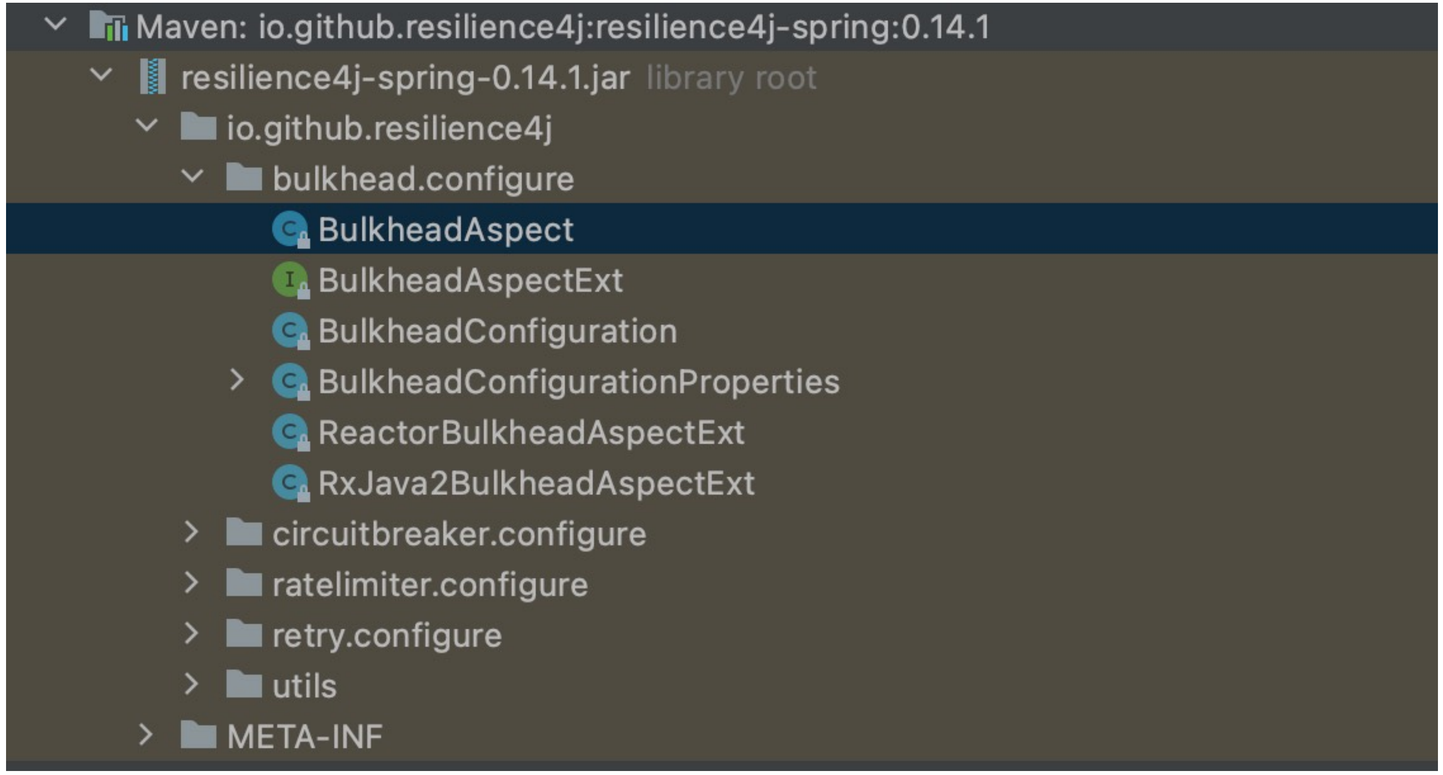
设置并发：

ab –c 5 –n 10 http://localhost:8090/…　5个并发请求十次

ab –从5 –n 20 –p /tmp/empty.txt http://localhsot:8090/…　带请求头文件的post请求

“Talk is cheap, show me the code.”

Chapter 13 / bulkhead-customer-service


Maven: io.github.resilience4j:resilience4j-spring:0.14.1
resilience4j-spring-0.14.1.jar library root
io.github.resilience4j
bulkhead.configure
BulkheadAspect
BulkheadAspectExt
BulkheadConfiguration
BulkheadConfigurationProperties
ReactorBulkheadAspectExt
RxJava2BulkheadAspectExt
circuitbreaker.configure
ratelimiter.configure
retry.configure
utils
META-INF

RateLimiter 这个限流保护加在wait-service上

目的

- 限制特定时间段内的执行次数

用法

- RateLimiterRegistry / RateLimiterConfig
- @RateLimiter(name = "名称")

RateLimiter

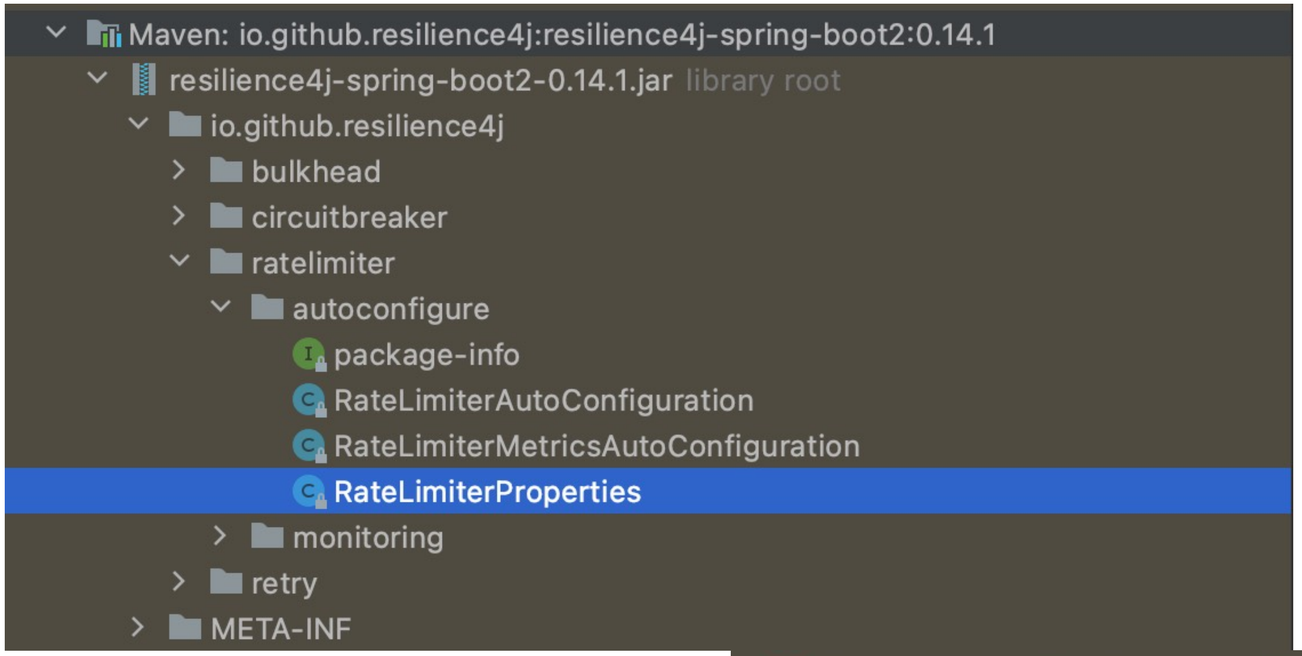
配置

- RateLimiterProperties
  - resilience4j.ratelimiter.limiters.名称
    - limit-for-period
    - limit-refresh-period-in-millis
    - timeout-in-millis

“Talk is cheap, show me the code.”

Chapter 13 / ratelimiter-waiter-service

配置项信息：



RateLimiterConfigurationProperties.class

```
public Map<String, RateLimiterConfigurationProperties.LimiterProperties> getLimiters() { return this.limiters; }

public static class LimiterProperties {
    private Integer limitForPeriod;
    private Integer limitRefreshPeriodInMillis;
    private Integer timeoutInMillis;
    private Boolean subscribeForEvents = false;
    private Boolean registerHealthIndicator = false;
    private Integer eventConsumerBufferSize = 100;

    public LimiterProperties() {
    }
```

SpringBucks 实战项目进度小结

本章小结

几种模式

- 断路器 / 隔舱 / 速率限制器

两个工具

- Netflix Hystrix / Resilience4j
- 建议学习，Google Guava

观察与监控

- Hystrix Dashboard / Micrometer

SpringBucks 进度小结

waiter-service

- 使用 Resilience4j 的 `RateLimiter` 进行防护

customer-service

- 使用 Hystrix 进行熔断
- 使用 Resilience4j 进行熔断和并发控制

极客时间

扫码试看/订阅

《玩转 Spring 全家桶》