基于AOP的熔断降级

1. 快速入门

1. 引入依赖

```
<dependency>
    <groupId>com.hdu</groupId>
    <artifactId>degrete-starter</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

2. 需要保护的方法上面添加注解

```
import com.hdu.degrete.degrate.Degrate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import java.util.concurrent.TimeUnit;

import static
com.hdu.degrete.degrate.CircuitBreaker.DegrateType.CUSTOM_RESULT;


@RestController
public class TestController {


    @GetMapping("test/{num}")
    @Degrate(
        degrateTime = 5,
```

```
        timeUnit = TimeUnit.SECONDS,
        degrateType = CUSTOM_RESULT,
        failRate = 0.5,
        customResultHandler = MyCustomResultHandler.class,
        methodName = "returnCustomResult"
    )
    public String test(@PathVariable int num) {
        if (num == 0) {
            throw new RuntimeException();
        } else {
            return "success";
        }
    }
}
```

3. 自定义降级结果

```
public class MyCustomResultHandler {

    public String returnCustomResult() {
        return "测试自定义返回结果";
    }
}
```

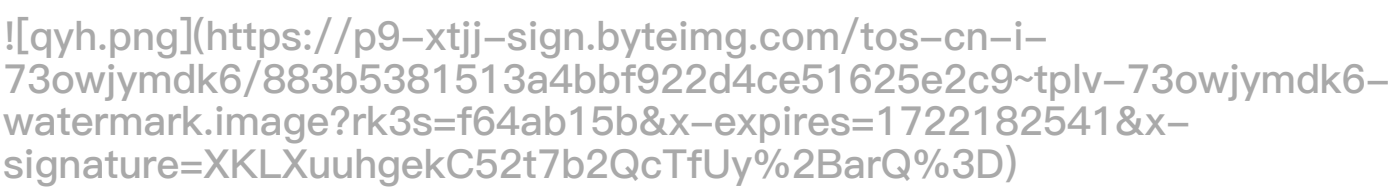
解释：

1. failRate，调用失败概率，如果**失败率 >= 50%**, 那么就会进入熔断状态。
2. degrateTime + timeUnit，熔断状态持续的时间。也就是说 如果该接口的**失败率 >= 50%** ，那么会进入熔断状态，**熔断状态持续的时间为5s**。
3. degrateType，标识熔断方式 **CUSTOM_RESULT** 表示返回自定义降级结果。
4. customResultHandler + methodName，指定 自定义降级结果的返回方法。也就是当该接口处于熔断状态的时候，返回的降级定义返回 **"测试自定义返回结果"**

2. 实现原理

熔断器设计

熔断器的状态是一个有限状态机。



CLOSE：CLOSE状态下请求可以正常执行。

OPEN：OPEN状态下请求会被熔断。

HALF_OPEN：HALF_OPEN是一个中间状态，会尝试发起一次请求，如果成功，切换为 CLOSE状态，如果失败 继续维持 OPEN状态。

3. 核心代码

熔断器

```
package com.hdu.degrete.degrate;

import lombok.Data;
import org.aspectj.lang.ProceedingJoinPoint;

import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicLong;
import java.util.concurrent.atomic.AtomicReference;
```

```
@Data
public class CircuitBreaker {


    public enum State {
        OPEN,
        HALF_OPEN,
        CLOSE
    }

    public enum DegrateType {
        /**
         * 自定义返回类型
         */
        CUSTOM_RESULT,
        /**
         * 抛异常
         */
        THROW_EXCEPTION
    }

    private final AtomicLong SUCCESS_COUNT = new AtomicLong(0);
    private final AtomicLong FAIL_COUNT = new AtomicLong(0);
    private final AtomicReference<State> STATE = new
AtomicReference<>(State.CLOSE);
    private final long DEGRATE_TIME_INTERVAL;
    private long waitCloseTime;


    public CircuitBreaker(long degrateTimeInterval, TimeUnit
degrateTimeIntervalUnit) {
        this.DEGRATE_TIME_INTERVAL =
degrateTimeIntervalUnit.toMillis(degrateTimeInterval);
    }


    public boolean tryPass(ProceedingJoinPoint proceedingJoinPoint) {

        if (STATE.get() == State.CLOSE) {
            return true;
        } else if (STATE.get() == State.OPEN) {
            return retryTimeArrived() &&
tryOneTimeAndTryOpen2Close(proceedingJoinPoint);
        }

        return false;
    }
```

```
    public void tryFromCloseToOpen(double failRate) {
        if (getFailRate() >= failRate) {
            STATE.compareAndSet(State.CLOSE, State.OPEN);
            refreshNextRetryTime();
        }
    }

    private boolean tryOneTimeAndTryOpen2Close(ProceedingJoinPoint
proceedingJoinPoint) {
        try {
            if (STATE.compareAndSet(State.OPEN, State.HALF_OPEN)) {
                proceedingJoinPoint.proceed();
                STATE.compareAndSet(State.HALF_OPEN, State.CLOSE);
                return true;
            } else {
                return false;
            }
        } catch (Throwable e) {
            STATE.set(State.OPEN);
            return false;
        }
    }


    public void refreshNextRetryTime() {
        waitCloseTime = System.currentTimeMillis() +
DEGRATE_TIME_INTERVAL;
    }

    private boolean retryTimeArrived() {
        return System.currentTimeMillis() >= waitCloseTime;
    }

    public void success() {
        SUCCESS_COUNT.incrementAndGet();
    }

    public void fail() {
        FAIL_COUNT.incrementAndGet();
    }


    public double getFailRate() {
        return FAIL_COUNT.get() / (double) (FAIL_COUNT.get() +
SUCCESS_COUNT.get());
    }
```

}

```

熔断切面
----

```
@Aspect
public class DegrateAspect {

    private final Map<String, CircuitBreaker> CIRCUIT_BREAKER_MAP =
new ConcurrentHashMap<>();
    private final Map<String, BeanAndMethodHolder>
BEAN_AND_METHOD_HOLDER_MAP = new ConcurrentHashMap<>();

    @Pointcut("@annotation(com.hdu.degrete.degrate.Degrate)")
    public void degrateMethodPt() {

    }

    @Around("degrateMethodPt()")
    public Object handleDegrateMethod(ProceedingJoinPoint pjp) {
        MethodSignature methodSignature = (MethodSignature)
pjp.getSignature();
        Method method = methodSignature.getMethod();
        Degrate degrate = method.getAnnotation(Degrate.class);
        String identity = getIdentity(pjp);
        CIRCUIT_BREAKER_MAP.putIfAbsent(
            identity,
            new CircuitBreaker(
                degrate.degrateTime(),
                degrate.timeUnit()
)
        );
        CircuitBreaker circuitBreaker =
CIRCUIT_BREAKER_MAP.get(identity);
        if (!circuitBreaker.tryPass(pjp)) {
            CircuitBreaker.DegrateType degrateType =
degrate.degrateType();
            switch (degrateType) {
                case CUSTOM_RESULT:
```
```

```
                    BEAN_AND_METHOD_HOLDER_MAP.putIfAbsent(
                        identity,
                        createBeanAndMethodHolder(degrate)
                    );
                    return
BEAN_AND_METHOD_HOLDER_MAP.get(identity).invoke();
                case THROW_EXCEPTION:
                    throw new DegrateException(identity + ”is degrate”);
                default:
                    throw new UnsupportedOperationException(”unsupported
degrate type”);
            }
        }
        try {
            Object result = pjp.proceed();
            circuitBreaker.success();
            return result;
        } catch (Throwable e) {
            circuitBreaker.fail();
            circuitBreaker.tryFromCloseToOpen(degrate.failRate());
            throw new RuntimeException(e);
        }
    }


    private BeanAndMethodHolder createBeanAndMethodHolder(Degrate
degrate) {
        try {
            Object customerResultHandler =
degrate.customResultHandler().newInstance();
            return new BeanAndMethodHolder(
                customerResultHandler,
                customerResultHandler.getClass().getDeclaredMethod(
                    degrate.methodName()
                )
            );
        } catch (NoSuchMethodException | InstantiationException |
IllegalAccessException e) {
            throw new RuntimeException(e);
        }
    }

    private String getIdentity(ProceedingJoinPoint proceedingJoinPoint) {
        return proceedingJoinPoint.getSignature().toString();
    }
}
```

```

## 4. 源码

[degrete: 基于aop的熔断降级组件 (gitee.com)](http://cxyroad.com/”https://gitee.com/K0n9DiKuA/degrete”)

原文链接: https://juejin.cn/post/7393539934391599104
```