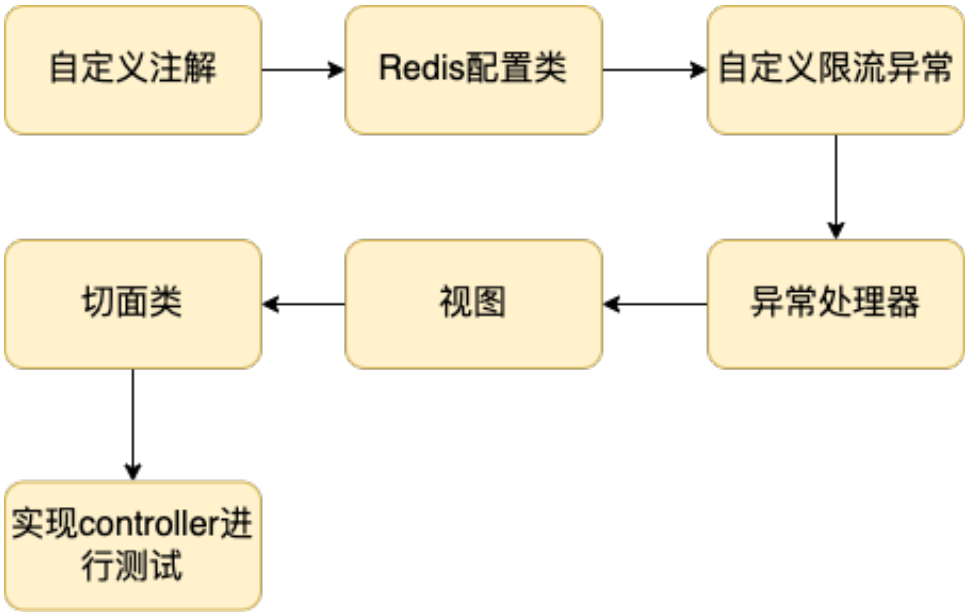
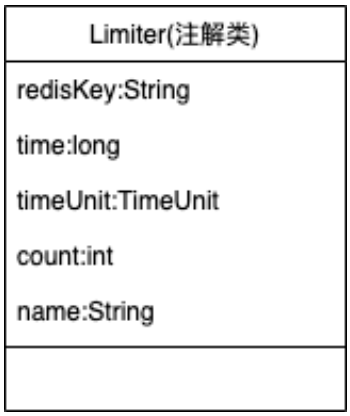
设计文档

编码流程图



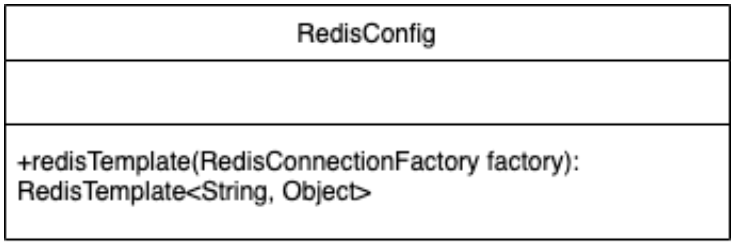
- 自定义注解

自定义注解Limiter，注解中包含的参数有：redisKey，存入Redis中的key值，默认为空；限流的时间time；时间单位timeUnit，默认为秒；限流的访问次数count；接口的名称name，用于描述接口功能。



- Redis配置类

Redis配置类RedisConfig，主要是Redis的序列化配置。

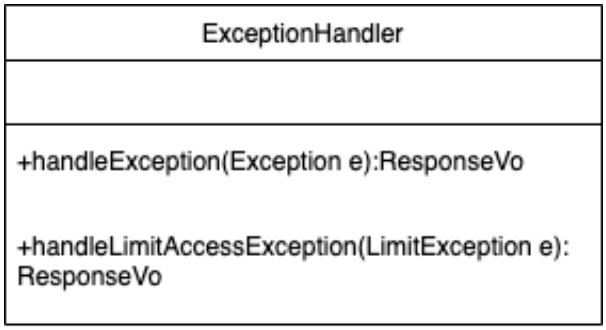


- 自定义限流异常

LimitException类继承RuntimeException，显示自定义的错误消息。

- 全局异常处理器

异常处理器ExceptionHandler，捕获访问次数超限的限流异常和其他异常，分别进行提示，返回提示视图。

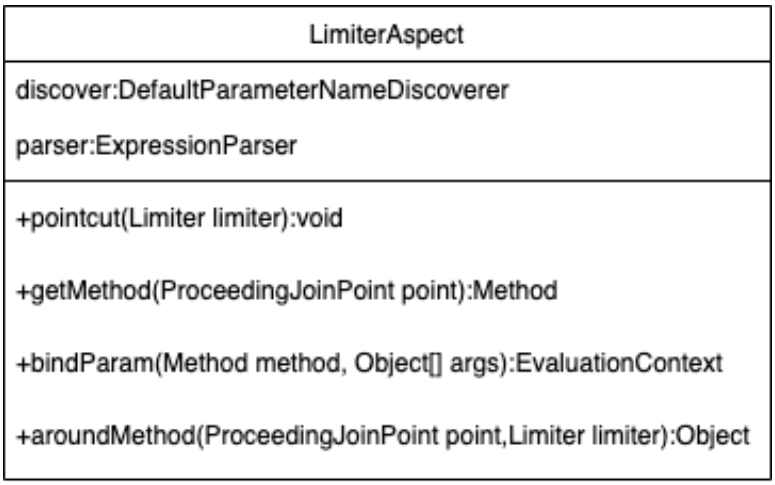


- 视图

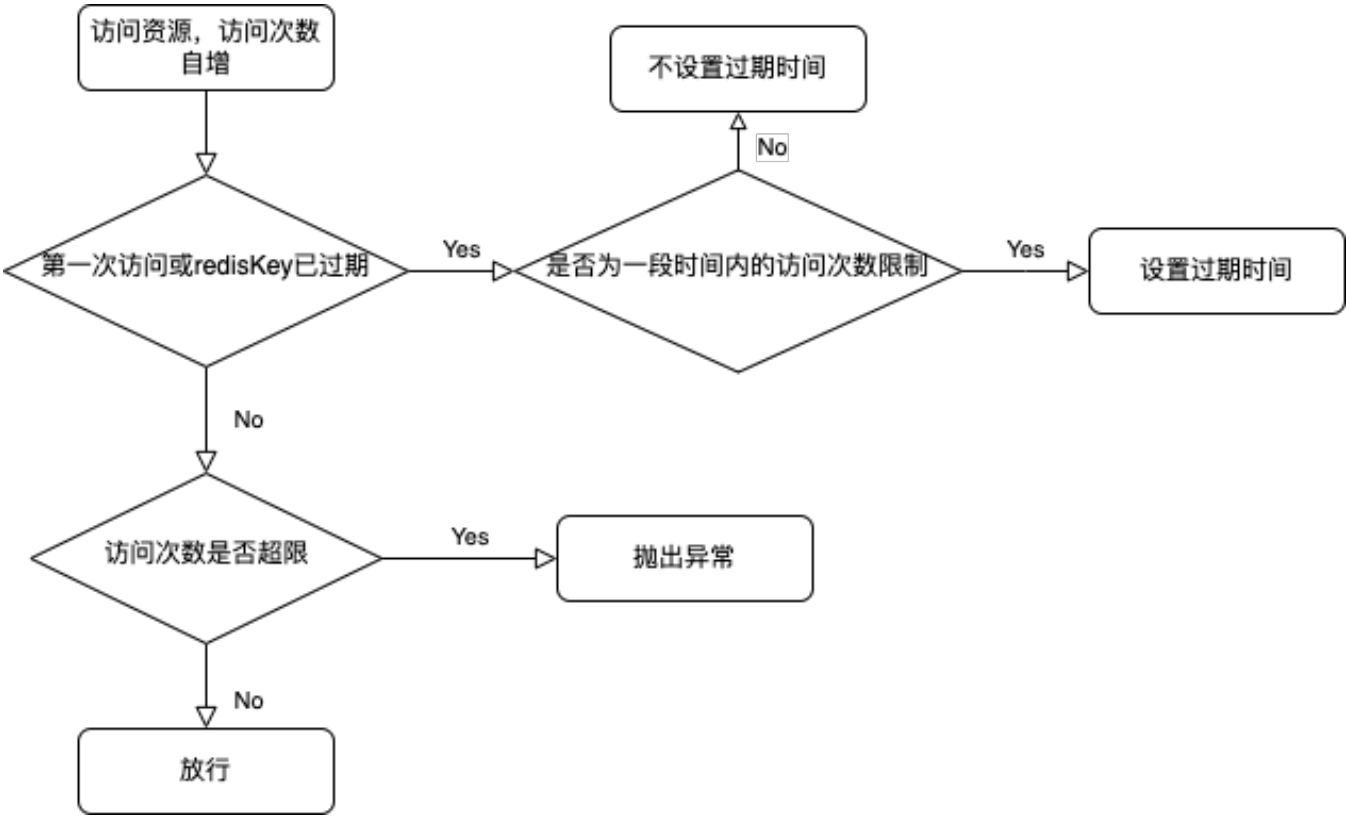
视图ResponseVo，继承HashMap<String,Object>，当用户访问资源时返回的简单视图，主要用于显示提示信息和自测。

- 切面类

切面类LImiterAspect，定义切点为有自定义注解Limiter的方法。通过Method方法，可以获取当前连接点的方法。bindParam方法将方法的参数名和参数值绑定。在通知中使用SpEL表达式解析器parser解析SpEl表达式，从而获得所需对应的参数值，拼接成redisKey。根据redisKey，使用Redis实现限流的功能。



限流逻辑



- 实现controller进行自测

TestController中包含四种类型的测试，分别为公共资源一段时间内访问次数限制；公共资源访问总次数的限制；资源根据用户名进行限流；资源根据用户Id进行限流。

TestController
+test1():ResponseVo +test2(@RequestParam("username") String username):ResponseVo +test3(@RequestParam("userId") int userId):ResponseVo +test4():ResponseVo

演示

版本变更说明(工作流程)

一.

周二上午搭建好环境，周二下午开始分析考核任务，然后将近晚上开始编码。周三下午完成了第一版代码并上传到了gitlab。第一个版本中只考虑了http的场景，考虑的并不完善；实现了一个工具类来获取用户的IP，如果是对单个用户限流的场景就使用用户IP来拼接redisKey；此外，编写了一个lua脚本，redis通过加载该脚本实现限流。

二.

周三晚上将第一版代码交给祝老师查看后，根据存在的问题改变思路和更改方案，并开始撰写需求文档。周四上午完成了第二版的编码工作。第二个版本运用了SpEL，使用SpEL解析器去获取表达式中所需的参数，例如username和userId等，然后拼接成redisKey；使用redis的工具类，通过自增和设置过期时间等方法替代了lua脚本。

三.

周四下午完成需求文档和设计展示。周五上午对并发存在的问题进行了一定的优化。然后尝试做另外的限流算法，但是存在一定问题，因为时间原因没有完成。下午添加了对传参为实体类的测试。