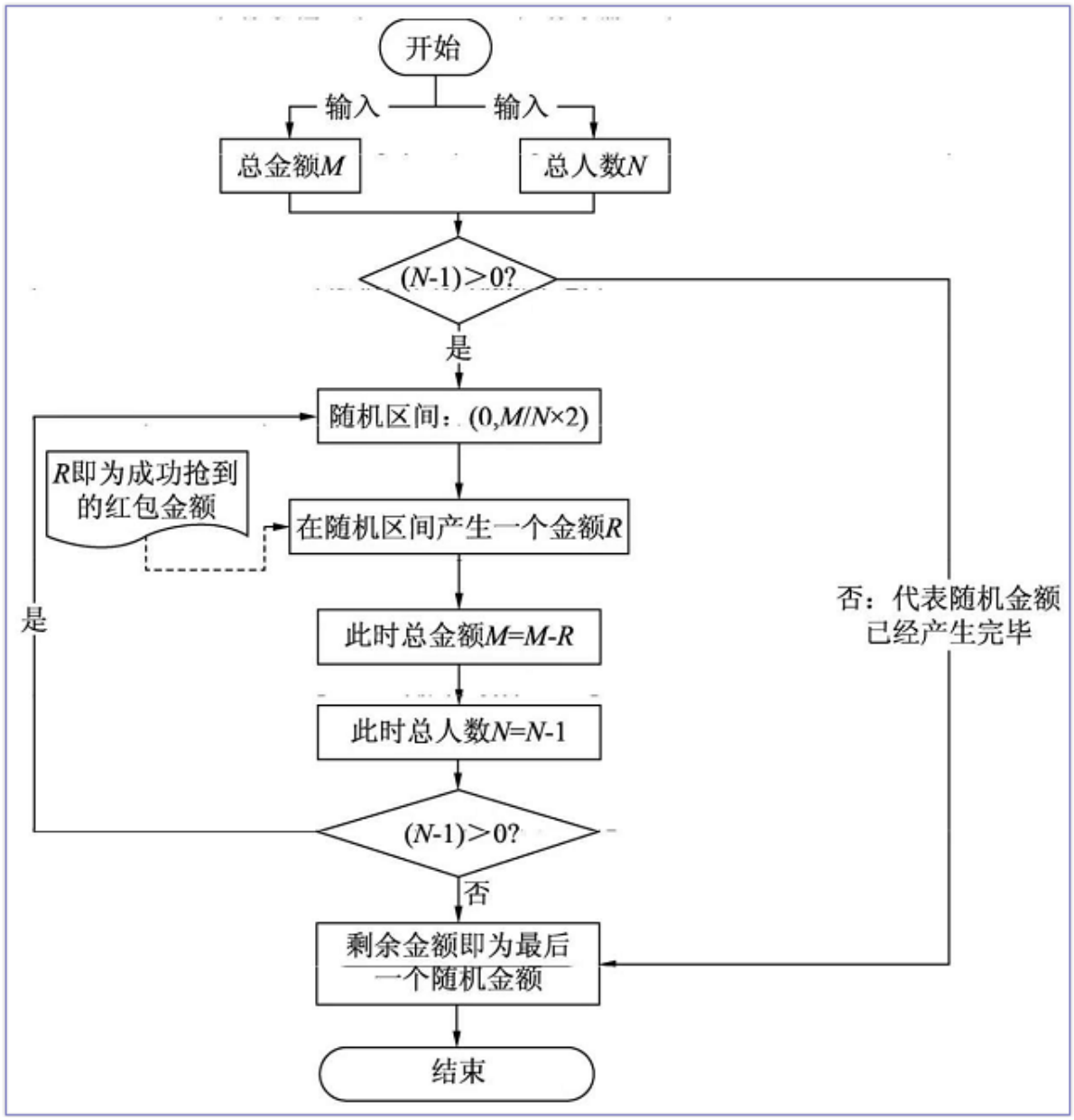
4.3 “红包金额”随机生成算法实战

4.3.5 红包随机金额生成算法自测

1. 二倍均值法生成红包随机金额的流程



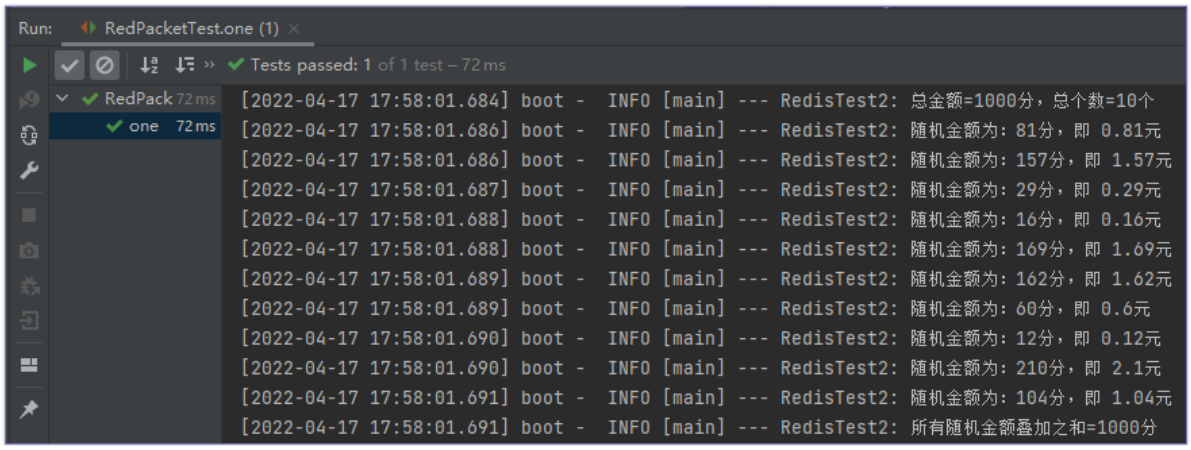
2. 源代码

```
/**
* 发红包算法，金额参数以分为单位
*
* @param totalAmount 红包总金额（单位为分）
* @param totalPeopleNum 总人数
* @return
*/
public static List<Integer> divideRedPackage(Integer totalAmount, Integer
totalPeopleNum) {
```

```
    // 用于存储每次产生的小红包随机金额列表，金额单位为分
    List<Integer> amountList = new ArrayList<>();
    // 判断总金额和总人数参数的合法性
    if (totalAmount > 0 && totalPeopleNum > 0) {
        // 记录剩余的总金额，初始化时即为红包的总金额
        Integer restAmount = totalAmount;
        // 记录剩余的总人数，初始化时即为红包的总人数
        Integer restPeopleNum = totalPeopleNum;
        // 定义产生随机数的实例对象
        Random random = new Random();
        // 不断循环遍历、迭代更新地产生随机金额，直到 N-1 ≤ 0
        for (int i = 0; i < totalPeopleNum - 1; i++) {
            // 随机范围：[1，剩余人均金额的两倍)，左闭右开，amount 即为产生的随机金额 R，单
位为分
            int amount = random.nextInt(restAmount / restPeopleNum * 2 - 1) + 1;
            // 更新剩余的总金额 M = M - R
            restAmount -= amount;
            // 更新剩余的总人数 N = N - 1
            restPeopleNum--;
            // 将产生的随机金额添加进列表中
            amountList.add(amount);
        }
        // 循环完毕，剩余的金额即为最后一个随机金额，也需要将其添加进列表中
        amountList.add(restAmount);
    }
    // 将最终产生的随机金额列表返回
    return amountList;
}
```

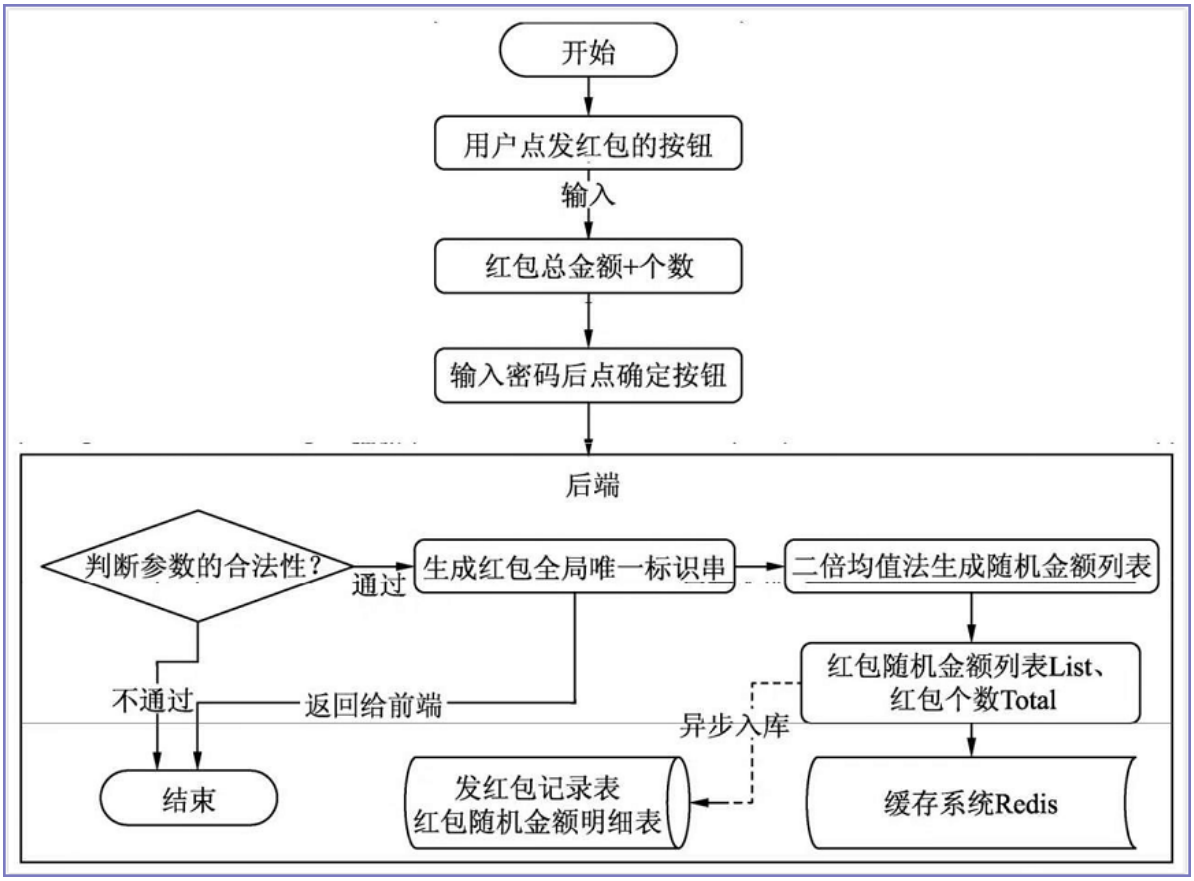
```
public void one() throws Exception {
    // 总金额单位为分，在这里假设总金额为 1000 分，即 10 元
    Integer amout = 1000;
    // 总人数，即红包总个数，在这里假设为10个
    Integer total = 10;
    // 得到随机金额列表
    List<Integer> list = RedPacketUtil.divideRedPackage(amout, total);
    log.info("总金额={}分，总个数={}个", amout, total);

    // 用于统计生成的随机金额之和是否等于总金额
    Integer sum = 0;
    // 遍历输出每个随机金额
    for (Integer i : list) {
        log.info("随机金额为：{}分，即 {}元", i, new
BigDecimal(i.toString()).divide(new BigDecimal(100)));
        sum += i;
    }
    log.info("所有随机金额叠加之和={}分", sum);
}
```

3. 运行结果



4.4 “发红包”模块实战

4.4.1 业务模块分析



4.4.2 整体流程实战

1. 源代码

   RedPacketController.java

```
/**
* 发红包请求，请求方法为 POST，请求参数采用 JSON 格式进行提交
*/
```

```
@RequestMapping(value = prefix + "/hand/out", method = RequestMethod.POST,
consumes = MediaType.APPLICATION_JSON_UTF8_VALUE)
public BaseResponse handOut(@Validated @RequestBody RedPacketDto dto,
BindingResult result) {
    // 参数校验
    if (result.hasErrors()) {
        return new BaseResponse(StatusCode.InvalidParams);
    }
    BaseResponse response = new BaseResponse(StatusCode.Success);
    try {
        // 核心业务逻辑处理服务，最终返回红包全局唯一标识串
        String redId = redPacketService.handOut(dto);
        // 将红包全局唯一标识串返回前端
        response.setData(redId);
    } catch (Exception e) {
        // 如果报异常，则输出日志并返回相应的错误信息
        log.error("发红包发生异常：dto={} ", dto, e.fillInStackTrace());
        response = new BaseResponse(StatusCode.Fail.getCode(), e.getMessage());
    }
    return response;
}
```

RedPacketService.java

```
public String handOut(RedPacketDto dto) throws Exception {
    // 判断参数的合法性
    if (dto.getTotal() > 0 && dto.getAmount() > 0) {
        // 采用二倍均值法生成随机金额列表
        List<Integer> list = RedPacketUtil.divideRedPackage(dto.getAmount(),
dto.getTotal());
        // 生成红包全局唯一标识，并将随机金额、个数存入缓存
        String timestamp = String.valueOf(System.nanoTime());
        // 根据缓存 key 的前缀与其他信息拼接成一个新的用于存储随机金额列表的 key
        String redId = new
StringBuffer(keyPrefix).append(dto.getUserId()).append(":").append(timestamp).to
String();
        // 将随机金额列表存入缓存列表中
        redisTemplate.opsForList().leftPushAll(redId, list);
        // 根据缓存 key 的前缀与其他信息拼接成一个新的用于存储红包总数的 Key
        String redTotalKey = redId + ":total";
        // 将红包总数存入缓存中
        redisTemplate.opsForValue().set(redTotalKey, dto.getTotal());
        // 异步记录红包的全局唯一标识串、红包个数与随机金额列表存入数据库
        redService.recordRedPacket(dto, redId, list);
        // 将红包的全局唯一标识串返回给前端
        return redId;
    } else {
        throw new Exception("系统异常-分发红包-参数不合法!");
    }
}
```

```
/**
 * 发红包记录
 *
```

```
     * @param dto    红包总金额+个数
     * @param redId 红包全局唯一标识串
     * @param list   红包随机金额列表
     * @throws Exception
     */
    @Override
    @Async
    @Transactional(rollbackFor = Exception.class)
    public void recordRedPacket(RedPacketDto dto, String redId, List<Integer> list)
    throws Exception {
        // 定义实体类对象
        RedRecord redRecord = new RedRecord();
        // 设置字段的取值信息
        redRecord.setUserId(dto.getUserId());
        redRecord.setRedPacket(redId);
        redRecord.setTotal(dto.getTotal());
        redRecord.setAmount(BigDecimal.valueOf(dto.getAmount()));
        redRecord.setCreateTime(new Date());
        // 将对象信息插入数据库
        redRecordMapper.insertSelective(redRecord);
        // 定义红包随机金额明细实体类对象
        RedDetail detail;
        // 遍历随机金额列表，将金额等信息设置到相应的字段中
        for (Integer i : list) {
            detail = new RedDetail();
            detail.setRecordId(redRecord.getId());
            detail.setAmount(BigDecimal.valueOf(i));
            detail.setCreateTime(new Date());
            // 将对象信息插入数据库
            redDetailMapper.insertSelective(detail);
        }
    }
```

4.4.3 业务模块自测

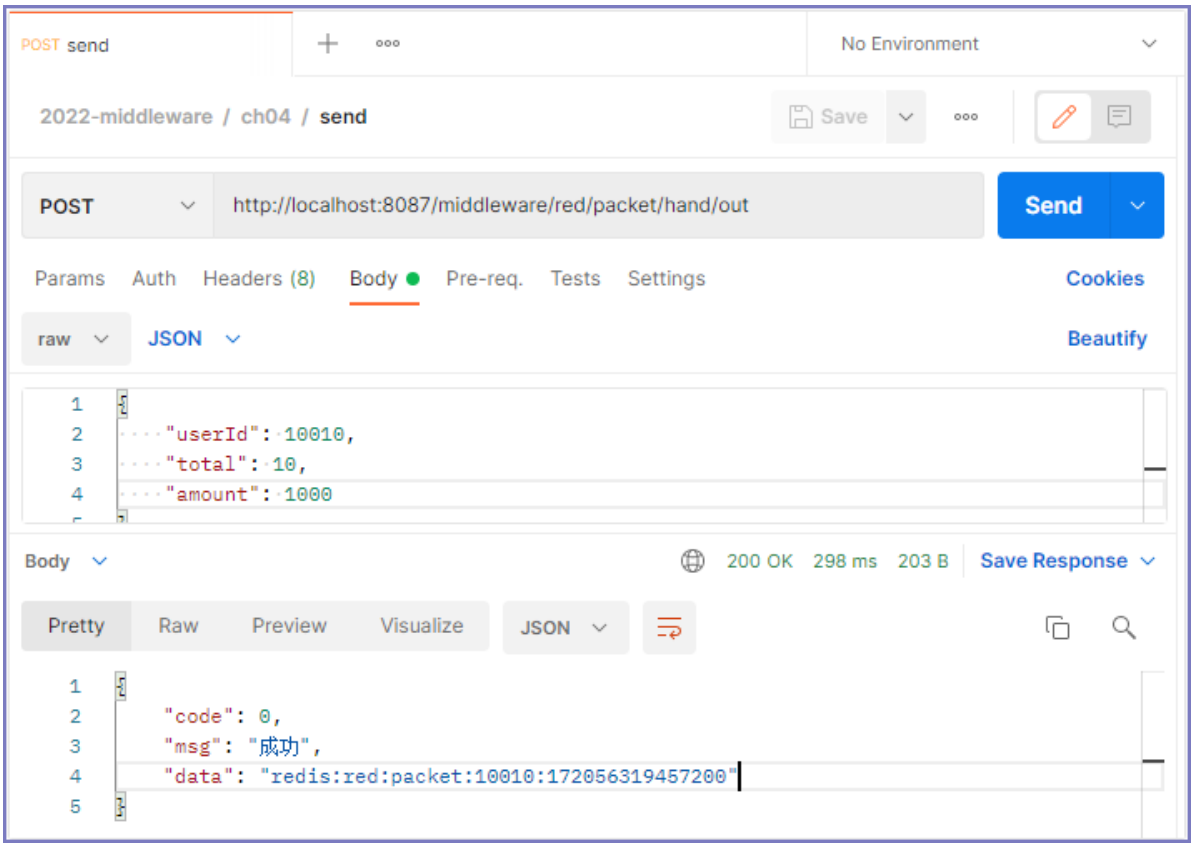
1. request

```
{
    "userId": 10010,
    "total": 10,
    "amount": 1000
}
```

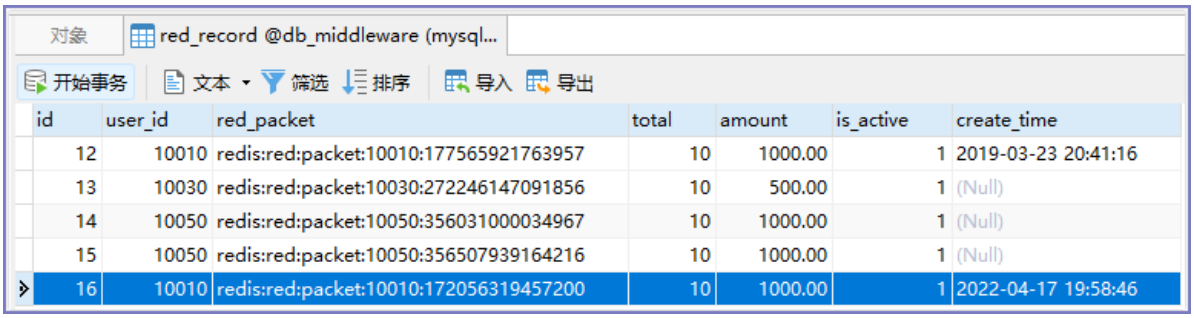
2. response

```
{
    "code": 0,
    "msg": "成功",
    "data": "redis:red:packet:10010:172056319457200"
}
```

3. Postman

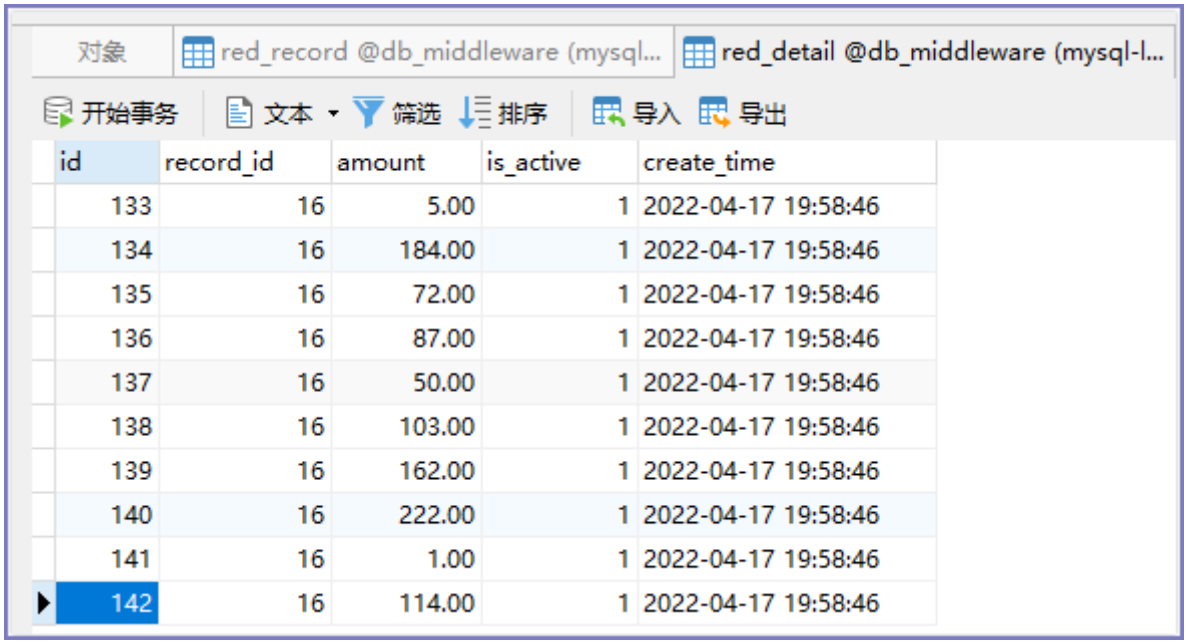


4. red_record



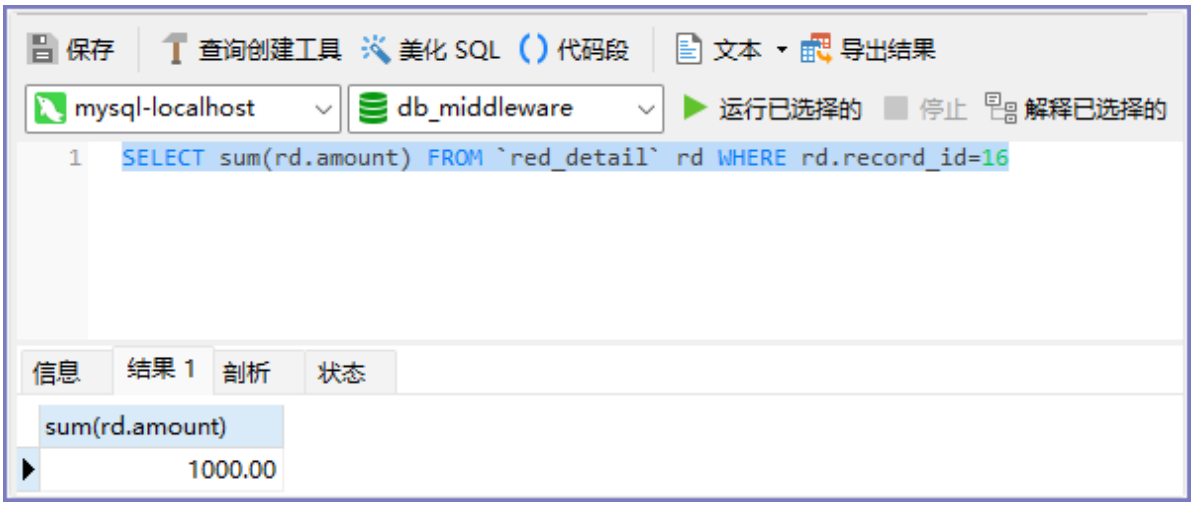
id	user_id	red_packet	total	amount	is_active	create_time
12	10010	redis:red:packet:10010:177565921763957	10	1000.00	1	2019-03-23 20:41:16
13	10030	redis:red:packet:10030:272246147091856	10	500.00	1	(Null)
14	10050	redis:red:packet:10050:356031000034967	10	1000.00	1	(Null)
15	10050	redis:red:packet:10050:356507939164216	10	1000.00	1	(Null)
16	10010	redis:red:packet:10010:172056319457200	10	1000.00	1	2022-04-17 19:58:46

5. red_detail



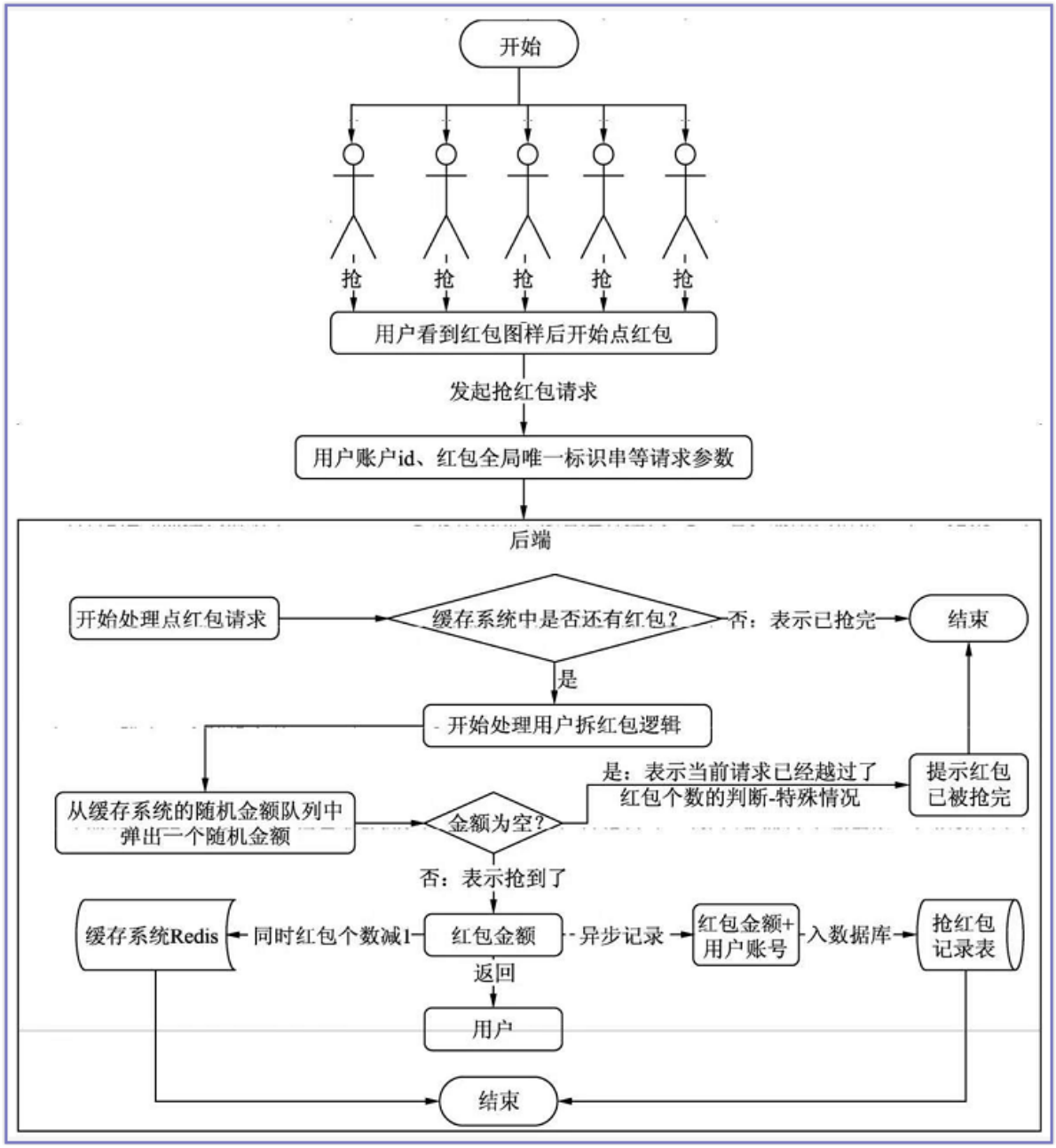
id	record_id	amount	is_active	create_time
133	16	5.00	1	2022-04-17 19:58:46
134	16	184.00	1	2022-04-17 19:58:46
135	16	72.00	1	2022-04-17 19:58:46
136	16	87.00	1	2022-04-17 19:58:46
137	16	50.00	1	2022-04-17 19:58:46
138	16	103.00	1	2022-04-17 19:58:46
139	16	162.00	1	2022-04-17 19:58:46
140	16	222.00	1	2022-04-17 19:58:46
141	16	1.00	1	2022-04-17 19:58:46
142	16	114.00	1	2022-04-17 19:58:46

6. amount



4.5 “抢红包”模块实战

4.5.1 业务模块分析



4.5.2 整体流程实战

1. 源代码

- RedPacketController.java

```
/**
 * 处理抢红包请求：接收当前用户 ID 和红包全局唯一标识串参数
 */
@RequestMapping(value = prefix + "/rob", method = RequestMethod.GET)
public BaseResponse rob(@RequestParam Integer userId, @RequestParam String
redId) {
    // 定义相应对象
    BaseResponse response = new BaseResponse(StatusCode.Success);
    try {
        // 调用红包业务逻辑处理接口中的抢红包方法，最终返回抢到的红包的金额
        // 单位为元（不为Null时表示抢到了，否则代表已经被抢完了
        BigDecimal result = redPacketService.rob(userId, redId);
        if (result != null) {
            // 将抢到的红包金额返回到前端
```

```
            response.setData(result);
        } else {
            // 没有抢到红包，即已经被抢完了
            response = new BaseResponse(StatusCode.Fail.getCode(), "红包已被抢
完!");
        }
    } catch (Exception e) {
        // 处理过程如果发生异常，则输出异常信息并返回给前端
        log.error("抢红包发生异常：userId={} redId={}", userId, redId,
e.fillInStackTrace());
        response = new BaseResponse(StatusCode.Fail.getCode(), e.getMessage());
    }
    // 返回处理结果给前端
    return response;
}
```

- RedPacketService.java

```
/**
 * 不加分布式锁的情况
 * 抢红包-分“点”与“抢”处理逻辑
 *
 * @param userId 当前用户 ID
 * @param redId  红包全局唯一标识串
 * @return 返回抢到的红包金额或者抢不到红包金额的 Null
 * @throws Exception
 */
@Override
public BigDecimal rob(Integer userId, String redId) throws Exception {
    // 定义 Redis 操作组件的值操作方法
    ValueOperations valueOperations = redisTemplate.opsForValue();
    // 在处理用户抢红包之前，需要先判断一下当前用户是否已经抢过该红包了
    // 如果已经抢过了，则直接返回红包金额，并在前端显示出来
    Object obj = valueOperations.get(redId + userId + ":rob");
    if (obj != null) {
        return new BigDecimal(obj.toString());
    }
    // “点红包”业务逻辑，主要用于判断缓存系统中是否仍然有红包，即红包剩余个数是否大于 0
    Boolean res = click(redId);
    // 表示 res 为 true，则可以进入“拆红包”业务逻辑
    if (res) {
        // "抢红包"-且红包有钱
        // 首先从小红包随机金额列表中弹出一个随机金额
        Object value = redisTemplate.opsForList().rightPop(redId);
        // value 不为 null，则表示当前弹出的红包金额不为 Null，即红包金额不为0，进而表示当
前用户抢到了一个红包了，则可以进入后续的更新缓存
        // 与记录信息入数据库了
        if (value != null) {
            // 构造红包记录Key
            String redTotalKey = redId + ":total";
            // 首先更新缓存系统中剩余红包个数，即红包个数减 1
            Integer currTotal = valueOperations.get(redTotalKey) != null ?
(Integer) valueOperations.get(redTotalKey) : 0;
            valueOperations.set(redTotalKey, currTotal - 1);
            // 将红包金额返回给用户的同时，将抢红包记录入数据库与缓存
            BigDecimal result = new BigDecimal(value.toString()).divide(new
BigDecimal(100));
```

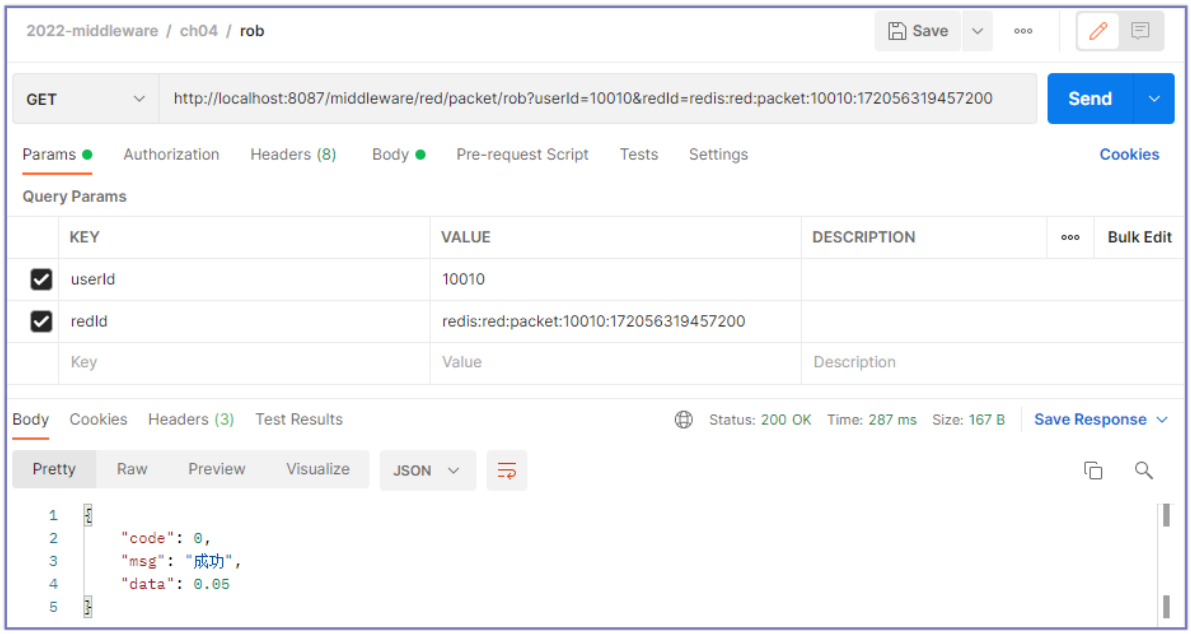
```
            // 记录抢到红包时用户的账号信息以及抢到的金额等信息入数据库
            redService.recordRobRedPacket(userId, redId, new
BigDecimal(value.toString()));
            // 将当前抢到红包的用户信息放置进缓存系统中，用于表示当前用户已经抢过红包了
            valueOperations.set(redId + userId + ":rob", result, 24L,
TimeUnit.HOURS);
            // 输出当前用户抢到红包的记录信息
            log.info("当前用户抢到红包了：userId={} key={} 金额={} ", userId, redId,
result);
            // 将结果返回
            return result;
        }
    }
    // null 则表示当前用户没有抢到红包
    return null;
}

/**
 * 点红包-返回true，则代表红包还有，个数>0
 *
 * @param redId 红包全局唯一标识串
 * @return 是否还有红包
 * @throws Exception
 */
private Boolean click(String redId) throws Exception {
    // 定义 Redis 的 Bean 操作组件（值操作组件）
    ValueOperations valueOperations = redisTemplate.opsForValue();
    // 定义用于查询缓存系统中红包剩余个数的 Key
    String redTotalKey = redId + ":total";
    // 获取缓存系统 Redis 中红包剩余个数
    Object total = valueOperations.get(redTotalKey);
    // 判断红包剩余个数 total 是否大于 0 ，如果大于 0， 则返回 true，表示还有红包；返回
false，则表示已经没有红包可抢了
    return total != null && Integer.valueOf(total.toString()) > 0;
}
```

- RedService.java

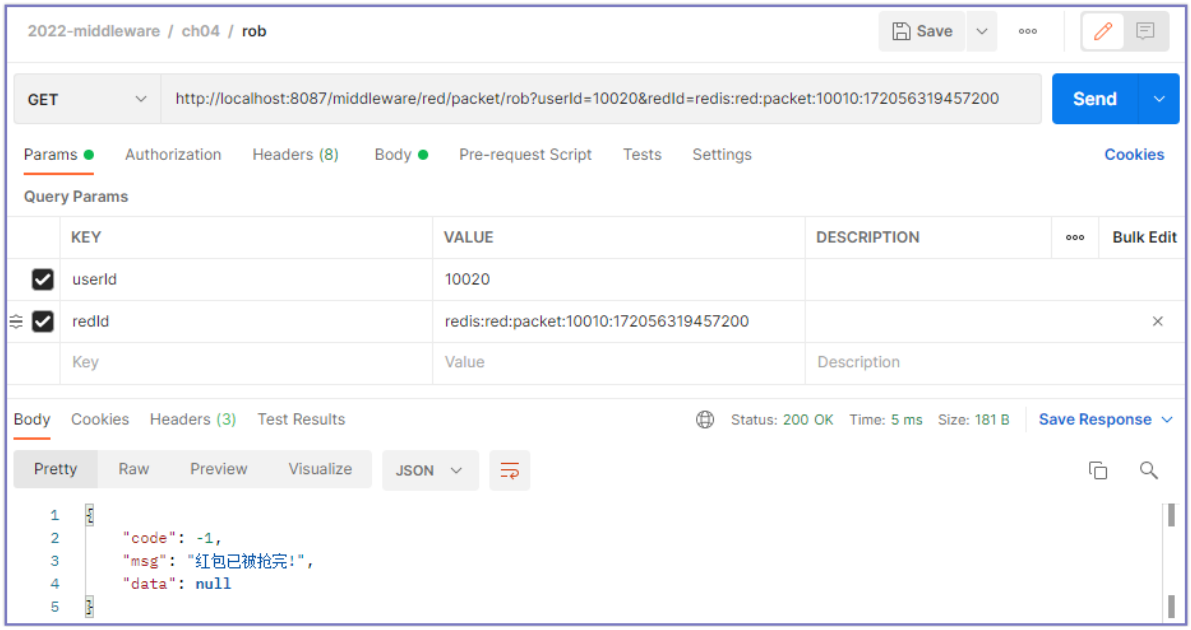
```
/**
 * 抢红包记录
 * 成功抢到红包时将当前用户账号信息以及对应的红包金额等信息记录进数据库中
 *
 * @param userId 用户 ID
 * @param redId  红包全局唯一标识串
 * @param amount 抢到的红包金额
 * @throws Exception
 */
@Override
@Async
public void recordRobRedPacket(Integer userId, String redId, BigDecimal amount)
throws Exception {
    // 定义记录抢到红包时录入相关信息的实体对象，并设置相应字段的取值
    RedRobRecord redRobRecord = new RedRobRecord();
    redRobRecord.setUserId(userId); // 设置用户 ID
    redRobRecord.setRedPacket(redId); // 设置红包全局唯一标识串
    redRobRecord.setAmount(amount); // 设置抢到的红包金额
    redRobRecord.setRobTime(new Date());// 设置抢到的时间
```

```
        // 将实体对象信息插入数据库中
        redRobRecordMapper.insertSelective(redRobRecord);
    }
```

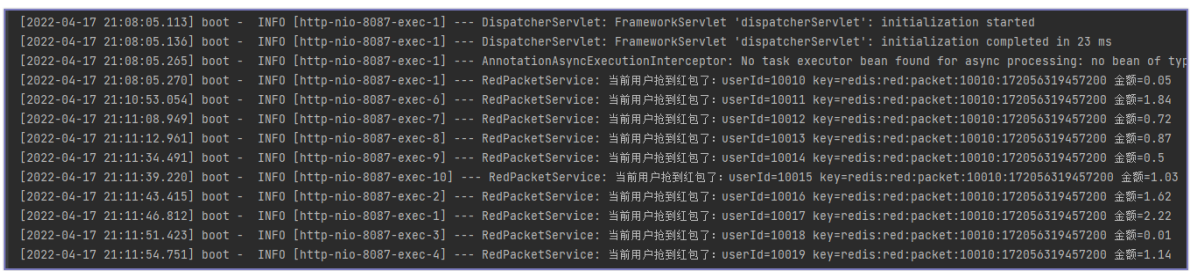
- 执行第1次



- 执行第11次



- 控制台信息



- red_detail 红包生成记录

对象 red_rob_record @db_middleware (... red_detail @db_middleware (mysql-l...

开始事务 文本 筛选 排序 导入 导出

id	record_id	amount	is_active	create_time
133	16	5.00	1	2022-04-17 19:58:46
134	16	184.00	1	2022-04-17 19:58:46
135	16	72.00	1	2022-04-17 19:58:46
136	16	87.00	1	2022-04-17 19:58:46
137	16	50.00	1	2022-04-17 19:58:46
138	16	103.00	1	2022-04-17 19:58:46
139	16	162.00	1	2022-04-17 19:58:46
140	16	222.00	1	2022-04-17 19:58:46
141	16	1.00	1	2022-04-17 19:58:46
142	16	114.00	1	2022-04-17 19:58:46

- red_rob_record 抢红包记录

对象 red_rob_record @db_middleware (...

开始事务 文本 筛选 排序 导入 导出

id	user_id	red_packet	amount	rob_time	is_active
118	10010	redis:red:packet:10010:17	5.00	2022-04-17 21:08:05	1
119	10011	redis:red:packet:10010:17	184.00	2022-04-17 21:10:53	1
120	10012	redis:red:packet:10010:17	72.00	2022-04-17 21:11:09	1
121	10013	redis:red:packet:10010:17	87.00	2022-04-17 21:11:13	1
122	10014	redis:red:packet:10010:17	50.00	2022-04-17 21:11:34	1
123	10015	redis:red:packet:10010:17	103.00	2022-04-17 21:11:39	1
124	10016	redis:red:packet:10010:17	162.00	2022-04-17 21:11:43	1
125	10017	redis:red:packet:10010:17	222.00	2022-04-17 21:11:47	1
126	10018	redis:red:packet:10010:17	1.00	2022-04-17 21:11:51	1
127	10019	redis:red:packet:10010:17	114.00	2022-04-17 21:11:55	1

4.6 JMeter压测高并发抢红包

2. 运行结果

3.3.3 集合

1. 源代码

```
public void three() throws Exception {
    //构造一组用户姓名列表
    List<String> userList = new ArrayList<>();
    userList.add("debug");
    userList.add("jack");
    userList.add("修罗");
    userList.add("大圣");
    userList.add("debug");
    userList.add("jack");
    userList.add("steadyheart");
    userList.add("修罗");
    userList.add("大圣");

    log.info("待处理的用户姓名列表：{} ", userList);

    //遍历访问，剔除相同姓名的用户并塞入集合中，最终存入缓存中
    final String key = "redis:test:3";
    SetOperations setOperations = redisTemplate.opsForSet();
    for (String str : userList) {
        setOperations.add(key, str);
    }

    //从缓存中获取已剔除的用户集合
    Object res = setOperations.pop(key);
    while (res != null) {
        log.info("从缓存中获取的用户集合-当前用户：{} ", res);
        res = setOperations.pop(key);
    }
}
```

2. 运行结果

3.3.4 有序集合

1. 源代码

```
public void four() throws Exception {
    // 构造一组无序的用户手机充值对象列表
    List<PhoneUser> list = new ArrayList<>();
    list.add(new PhoneUser("103", 130.0));
    list.add(new PhoneUser("101", 120.0));
    list.add(new PhoneUser("102", 80.0));
    list.add(new PhoneUser("105", 70.0));
    list.add(new PhoneUser("106", 50.0));
    list.add(new PhoneUser("104", 150.0));
    log.info("构造一组无序的用户手机充值对象列表:{}", list);
```

```
    // 遍历访问充值对象列表，将信息塞入Redis的有序集合中
    final String key = "redis:test:4";
    // 因为zSet在add元素进入缓存后，下次就不能进行更新了，故而为了测试方便，
    // 进行操作之前先清空该缓存(当然实际生产环境中不建议这么使用)
    redisTemplate.delete(key);

    ZSetOperations zSetOperations = redisTemplate.opsForZSet();
    for (PhoneUser u : list) {
        zSetOperations.add(key, u, u.getFare());
    }

    // 前端获取访问充值排名靠前的用户列表
    Long size = zSetOperations.size(key);
    // 从小到大排序
    Set<PhoneUser> resSet = zSetOperations.range(key, 0L, size);
    // 从大到小排序
    //Set<PhoneUser> resSet=zSetOperations.reverseRange(key,0L,size);
    for (PhoneUser u : resSet) {
        log.info("从缓存中读取手机充值记录排序列表，当前记录：{} ", u);
    }
}
```

2. 运行结果

3.3.5 哈希存储

1. 源代码

```
public void five() throws Exception {
    // 构造学生对象列表，水果对象列表
    List<Student> students = new ArrayList<>();
    List<Fruit> fruits = new ArrayList<>();

    students.add(new Student("10010", "debug", "大圣"));
    students.add(new Student("10011", "jack", "修罗"));
    students.add(new Student("10012", "sam", "上古"));

    fruits.add(new Fruit("apple", "红色"));
    fruits.add(new Fruit("orange", "橙色"));
    fruits.add(new Fruit("banana", "黄色"));

    // 分别遍历不同对象队列，并采用Hash哈希存储至缓存中
    final String sKey = "redis:test:5";
    final String fKey = "redis:test:6";

    HashOperations hashOperations = redisTemplate.opsForHash();
    for (Student s : students) {
        hashOperations.put(sKey, s.getId(), s);
```

```
    }
    for (Fruit f : fruits) {
        hashOperations.put(fKey, f.getName(), f);
    }

    // 获取学生对象列表与水果对象列表
    Map<String, Student> sMap = hashOperations.entries(sKey);
    log.info("获取学生对象列表：{} ", sMap);

    Map<String, Fruit> fMap = hashOperations.entries(fKey);
    log.info("获取水果对象列表：{} ", fMap);

    // 获取指定的学生对象、水果对象
    String sField = "10012";
    Student s = (Student) hashOperations.get(sKey, sField);
    log.info("获取指定的学生对象：{} -> {} ", sField, s);

    String fField = "orange";
    Fruit f = (Fruit) hashOperations.get(fKey, fField);
    log.info("获取指定的水果对象：{} -> {} ", fField, f);
}
```

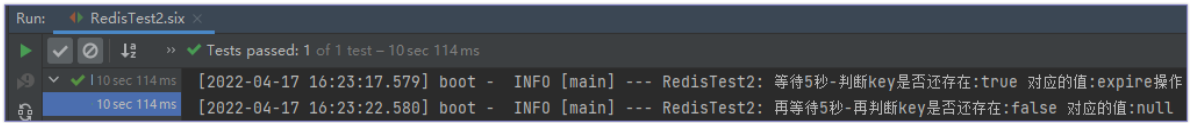
2. 运行结果

3.3.6 Key失效与判断是否存在

1. 源代码

```
public void six() throws Exception {
    // 构造key与redis操作组件
    final String key1 = "redis:test:6";
    ValueOperations valueOperations = redisTemplate.opsForValue();

    // 第一种方法：在往缓存中set数据时,提供一个ttl,表示ttl时间一到,缓存中的key将自动失效,即被清理
    // 在这里TTL是10秒
    valueOperations.set(key1, "expire操作", 10L, TimeUnit.SECONDS);

    // 等待5秒-判断key是否还存在
    Thread.sleep(5000);
    Boolean existKey1 = redisTemplate.hasKey(key1);
    Object value = valueOperations.get(key1);
    log.info("等待5秒-判断key是否还存在:{} 对应的值:{}", existKey1, value);

    // 再等待5秒-再判断key是否还存在
    Thread.sleep(5000);
    existKey1 = redisTemplate.hasKey(key1);
    value = valueOperations.get(key1);
    log.info("再等待5秒-再判断key是否还存在:{} 对应的值:{}", existKey1, value);
}
```

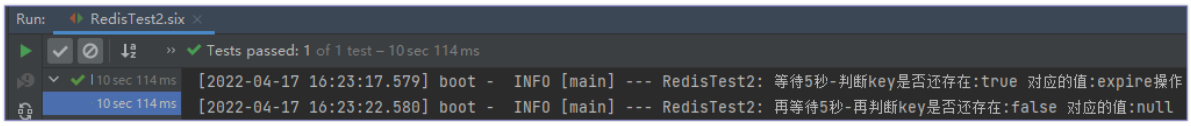
2. 运行结果



3. 源代码

```
public void seven() throws Exception {
    // 构造key与redis操作组件
    final String key2 = "redis:test:7";
    ValueOperations valueOperations = redisTemplate.opsForValue();

    // 第二种方法：在往缓存中set数据后,采用redisTemplate的expire方法失效该key
    valueOperations.set(key2, "expire操作-2");
    redisTemplate.expire(key2, 10L, TimeUnit.SECONDS);

    // 等待5秒-判断key是否还存在
    Thread.sleep(5000);
    Boolean existKey = redisTemplate.hasKey(key2);
    Object value = valueOperations.get(key2);
    log.info("等待5秒-判断key是否还存在:{} 对应的值:{}", existKey, value);

    // 再等待5秒-再判断key是否还存在
    Thread.sleep(5000);
    existKey = redisTemplate.hasKey(key2);
    value = valueOperations.get(key2);
    log.info("再等待5秒-再判断key是否还存在:{} 对应的值:{}", existKey, value);
}
```

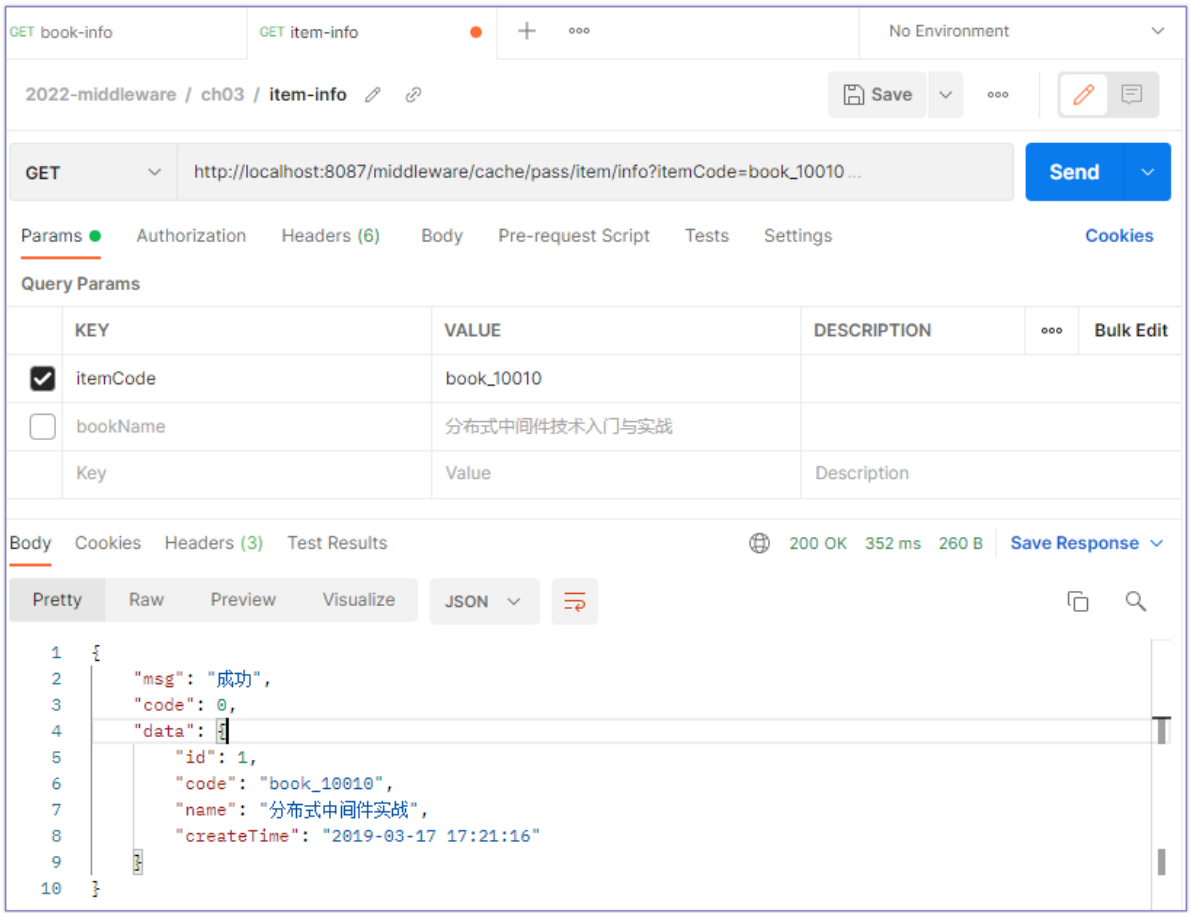
4. 运行结果



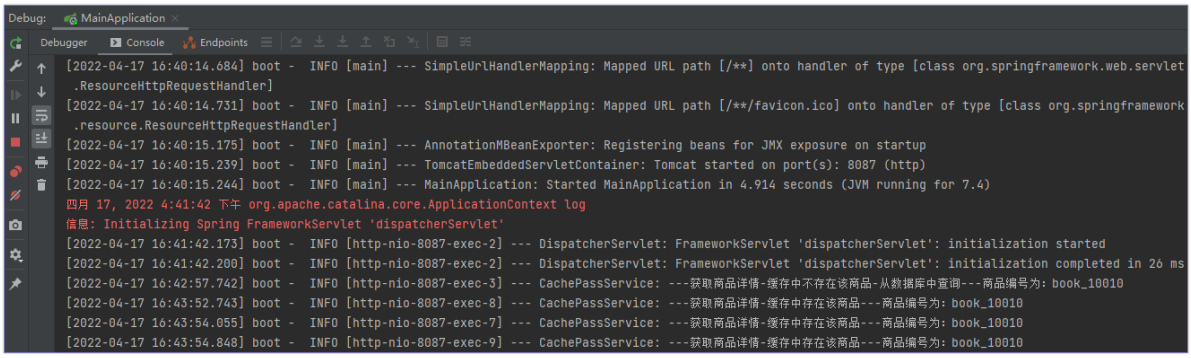
3.4 Redis实战场景之缓存穿透

3.4.3 未穿透

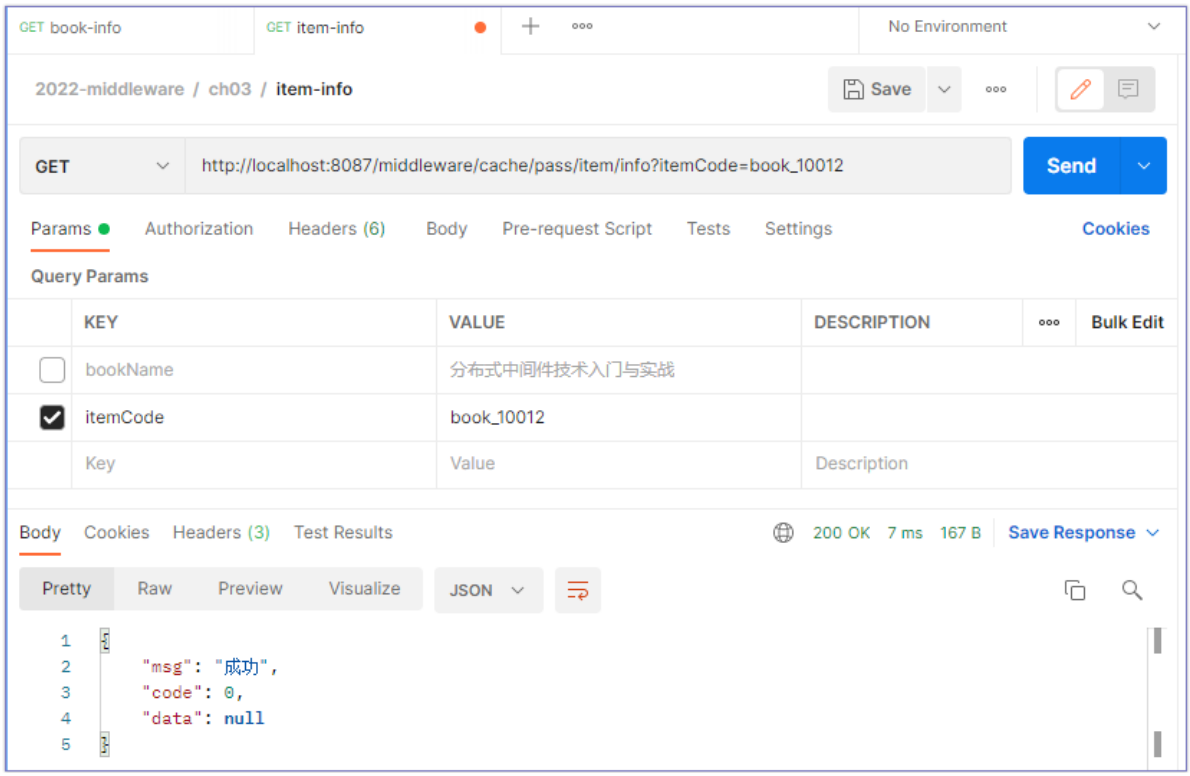
1. 源代码



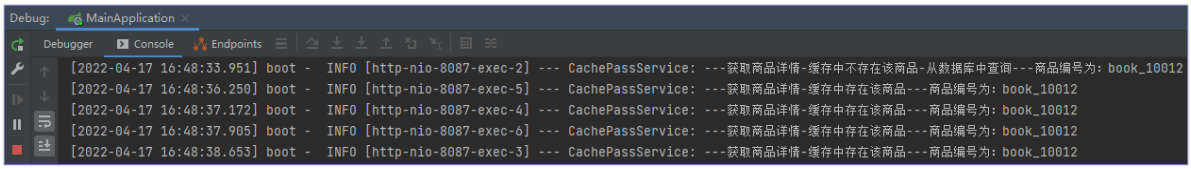
缓存穿透



3. 解决后的响应结果


GET book-info
GET item-info
No Environment
2022-middleware / ch03 / item-info
Save
GET
http://localhost:8087/middleware/cache/pass/item/info?itemCode=book_10012
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
bookName
分布式中间件技术入门与实战
itemCode
book_10012
Key
Value
Description
Body
Cookies
Headers (3)
Test Results
200 OK
7 ms
167 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"msg": "成功",
"code": 0,
"data": null
}

4. 日志


Debug:
MainApplication
Debugger
Console
Endpoints
[2022-04-17 16:48:33.951] boot - INFO [http-nio-8087-exec-2] --- CachePassService: ---获取商品详情-缓存中不存在该商品-从数据库中查询---商品编号为: book_10012
[2022-04-17 16:48:36.250] boot - INFO [http-nio-8087-exec-5] --- CachePassService: ---获取商品详情-缓存中存在该商品---商品编号为: book_10012
[2022-04-17 16:48:37.172] boot - INFO [http-nio-8087-exec-4] --- CachePassService: ---获取商品详情-缓存中存在该商品---商品编号为: book_10012
[2022-04-17 16:48:37.905] boot - INFO [http-nio-8087-exec-6] --- CachePassService: ---获取商品详情-缓存中存在该商品---商品编号为: book_10012
[2022-04-17 16:48:38.653] boot - INFO [http-nio-8087-exec-3] --- CachePassService: ---获取商品详情-缓存中存在该商品---商品编号为: book_10012

END