易圣通分布式

易圣通

Redis-抢红包系统

卫昆

简介

- 在微服务、分布式系统架构的时代，Redis作为一款具有高性能存储的缓存中间件，具有很广泛的应用，特别是在以内容、服务为主打运营模式的互联网产品中，几乎都可以看到Redis的踪影。
- 我们在这里将重点介绍Redis的典型应用场景，即“抢红包系统”的设计与开发实战，巩固Redis的各种特性和典型数据结构在实际生产环境中的应用。
- “抢红包”起初是在微信中兴起并得到普及的。其“发红包”与“抢红包”的业务着实给企业带来了巨大的用户流量。在本章中我们将自主设计一款能“扛住”用户高并发请求的抢红包系统。

简介

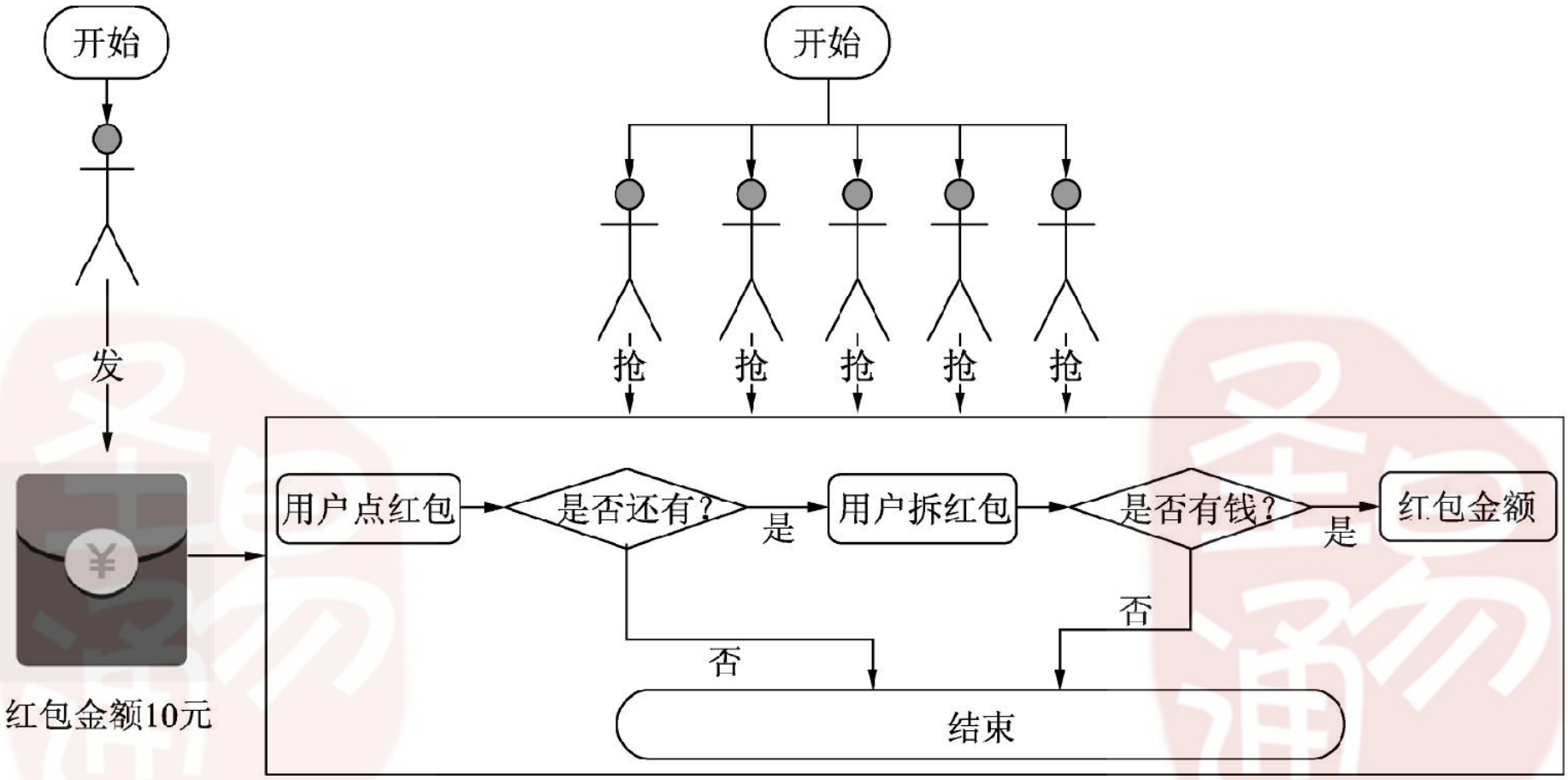
- 我们通过该系统，你将得到：抢红包系统整体业务流程介绍、业务流程分析和业务模块划分。
- 系统整体开发环境的搭建、相关数据库表的设计及整体开发流程介绍。
- “红包金额”随机数算法之二倍均值法的介绍、分析、代码实战和自测。
- “发红包”“抢红包”业务模块的分析、代码实战和自测，采用Jmeter工具压力测试高并发下抢红包出现的问题。
- 对高并发下产生的问题进行分析，并采用分布式锁对整体业务逻辑进行优化。

简介

- 概括性地讲，一个红包系统主要由以下三大部分组成：
- 信息流：包括用户操作背后的请求通信和红包信息在不同用户与用户群中的流转等。
- 业务流：主要包括发红包、点红包和抢红包等业务逻辑。
- 资金流：主要包括红包背后的资金转账和入账等流程。
- 由于课时有限，我们将重点分析其中的业务流，并借鉴微信红包中的一种模式进行实例演示。
- 此模式的场景为：用户发出一个固定金额的红包，让若干个人抢。
- 如下图。整体业务流程图

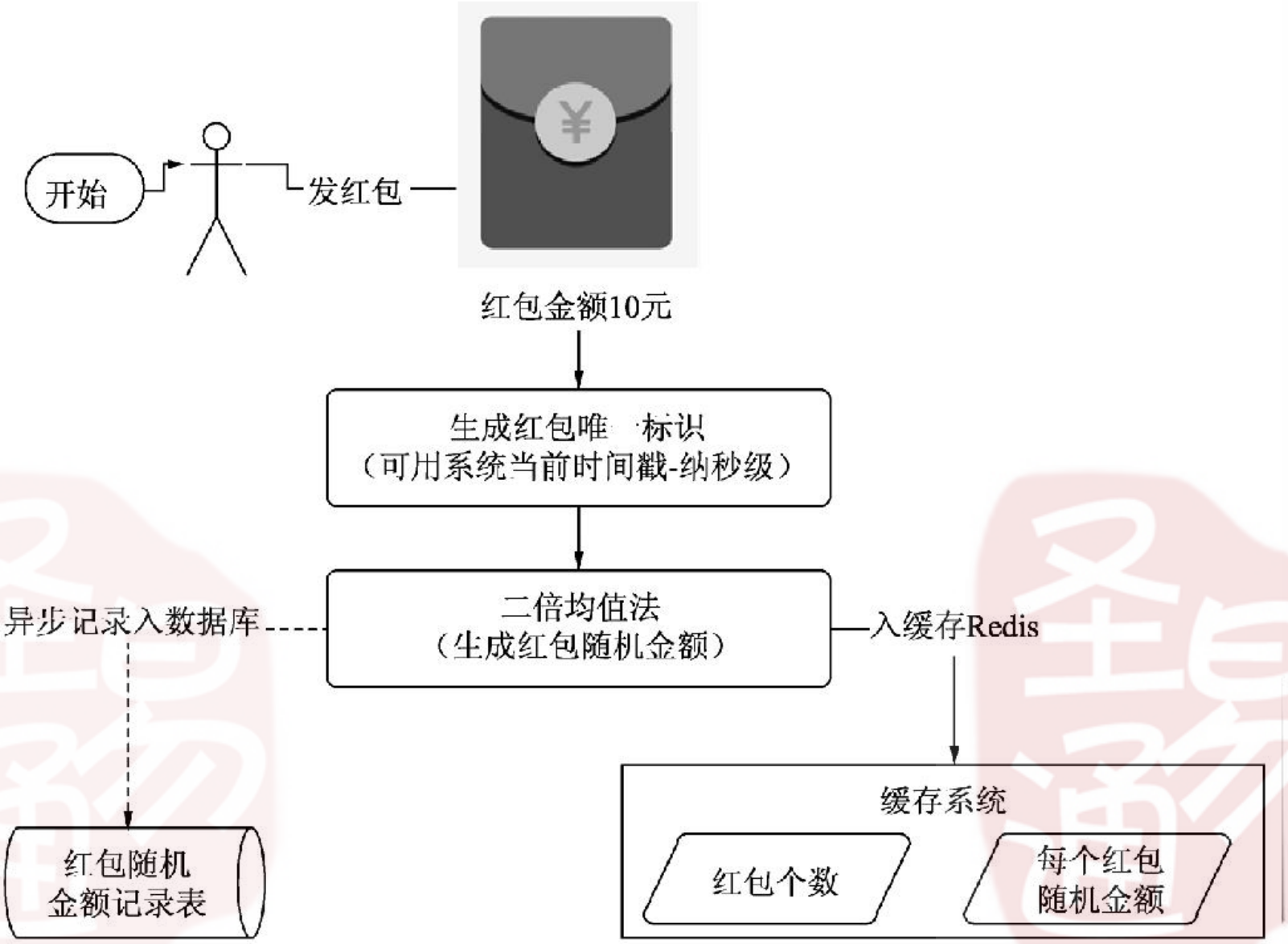
抢红包业务流程图

- 其中，左边（小人图标）为发红包者，右边为用户群里其他的抢红包者（当然可以包括发红包者本人）。



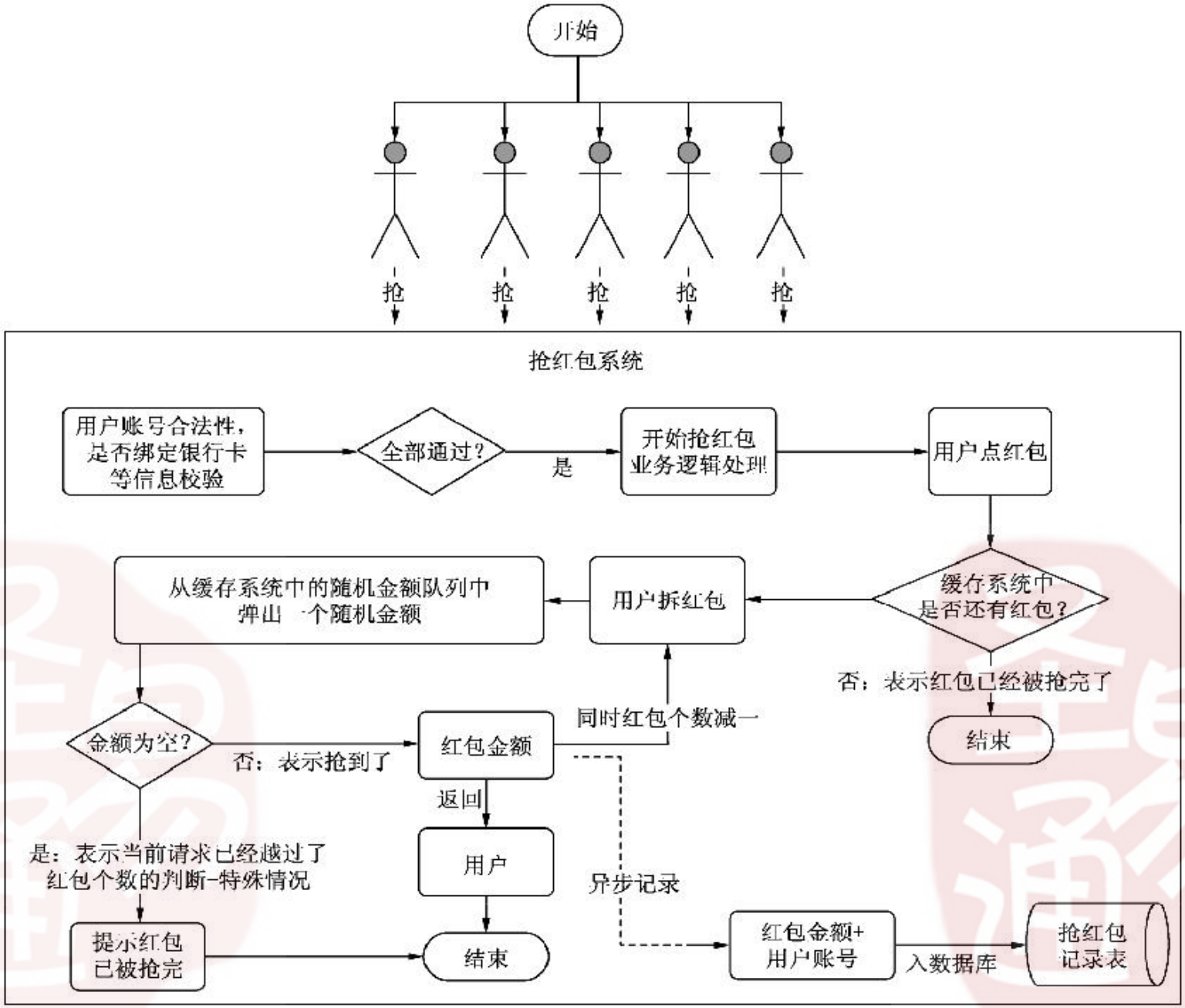
业务流程分析

- 由上图可以看出，系统整体业务流程主要由两大业务组成：发红包和抢红包。其中，抢红包又可以拆分为两个小业务，即用户点红包和用户拆红包，相信玩过微信红包或者QQ红包的读者对此并不陌生。下面介绍一下这两大业务模块具体的业务流程。
- 首先是用户发红包流程。用户单击进入某个群（如微信群、QQ群），然后单击发红包按钮，输入总金额与红包个数，最后确认，输入支付密码后，将在群界面生成一个红包的图样，之后群里的所有成员就可以开始抢红包了，整体业务流程如下图所示。


开始
发红包
红包金额10元
生成红包唯一标识
（可用系统当前时间戳-纳秒级）
二倍均值法
（生成红包随机金额）
异步记录入数据库
入缓存Redis
红包随机
金额记录表
缓存系统
红包个数
每个红包
随机金额

业务流程分析

- 用户抢红包流程：当群里的成员看到红包图样的时候，正常情况下都会单击红包图样，由此便开始了系统的抢红包流程。
- 系统后端接口在接收到前端用户抢红包的请求时：
- 1.校验用户账号的合法性：（比如当前账号是否已被禁用抢红包等）、是否绑定银行卡等。当全部信息校验通过之后，将真正开启抢红包业务逻辑的处理。整体业务流程如下图所示。

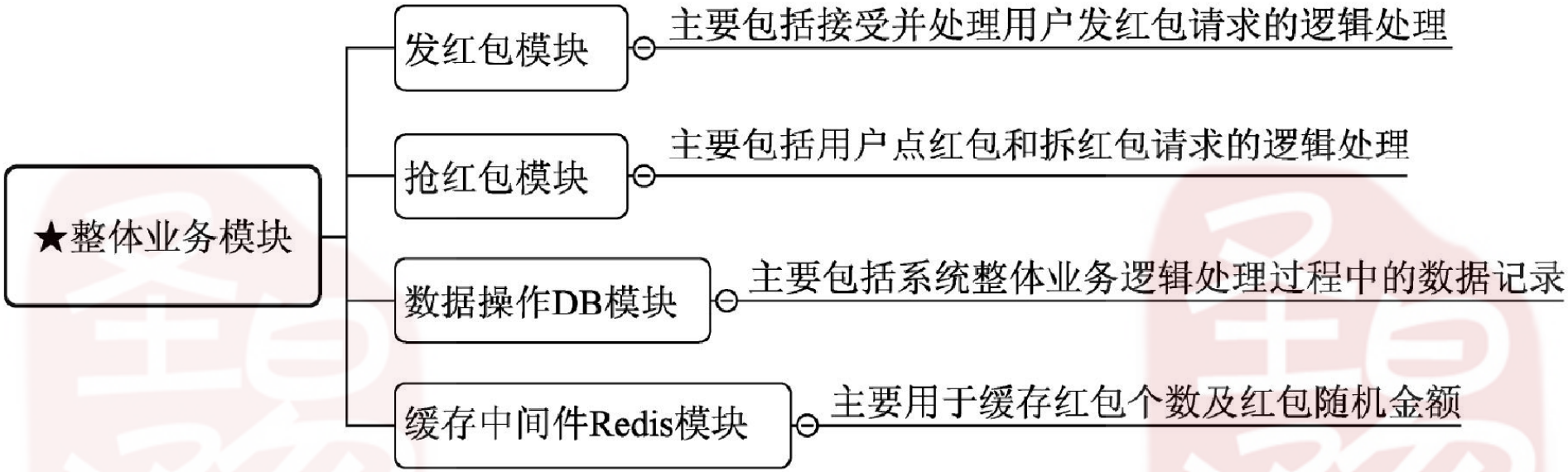


业务流程分析

- 2.“用户点红包”：主要用于判断缓存系统中红包个数是否大于0。如果小于等于0。则意味着红包已经被抢光了；如果红包个数大于0，则表示缓存系统中还有红包，仍然可以抢。
- 3.“用户拆红包”：主要是从缓存系统的红包随机金额队列中弹出一个随机金额，如果金额不为空，则表示该用户抢到红包了，缓存系统中红包个数减1，同时异步记录用户抢红包的记录并结束流程；如果金额为空，则意味着用户来晚一步，红包已被抢光（比如用户点开红包后，却迟迟不能拆的情况）。

业务模块划分

- 系统整体业务模块的划分依据主要来源于对系统整体业务流程的分析和拆分，抢红包系统整体业务流程主要包括发和抢，基于此，对系统进行整体业务模块的划分。如下图：

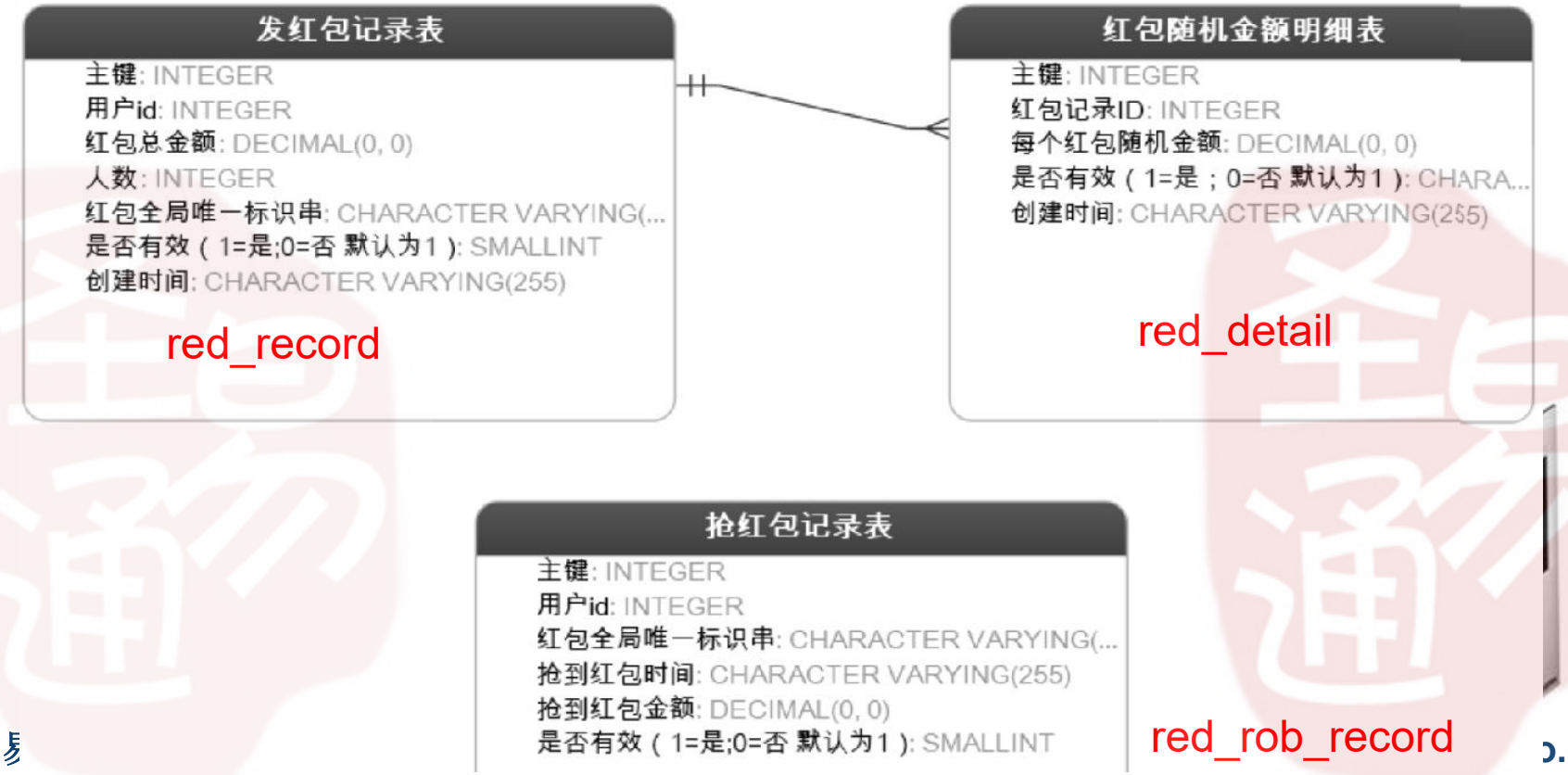


业务模块划分

- 而在系统整体业务操作过程中，将产生各种各样的核心业务数据，这部分数据将由数据操作DB模块存储至数据库中。
- 最后是缓存中间件Redis操作模块，主要用于发红包时缓存红包个数和由随机数算法产生的红包随机金额列表，同时将借助Redis单线程特性与操作的原子性实现抢红包时的锁操作。
- 我们将看到Redis的引入将给系统带来诸多的贡献。一方面它将大大减少高并发情况下频繁查询数据库的操作，从而减少数据库的压力；另一方面则是提高系统的整体响应性能和保证数据的一致性。

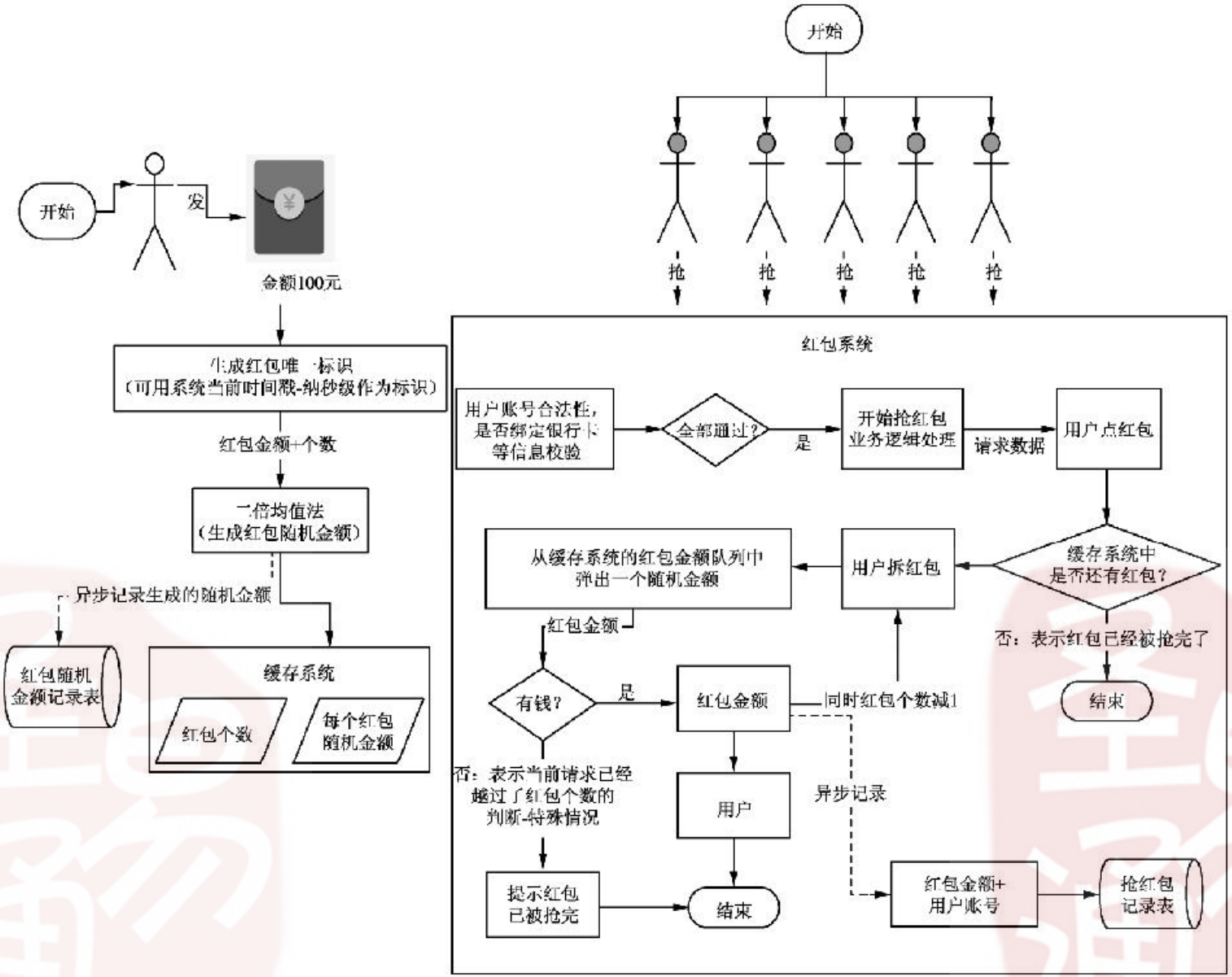
数据库表设计与环境搭建

- 如下图：主要包括3张数据表，即发红包时记录红包相关信息表、发红包时生成的对应随机金额信息表，以及抢红包时用户抢到的红包金额记录表。请参见项目的red_record，red_detail，red_rob_record三张表。



开发流程介绍

- 需要梳理一下系统的整体开发流程，主要目的是为了跟踪用户在前端发起的操作请求，以及由此产生的数据流向。如下图所示为抢红包系统用户请求处理的整体过程及由此产生的数据流向。从图中可以看出，用户发起的请求主要包括“发红包”和“抢红包”请求。



抢红包系统处理用户请求的过程和数据流向

开发流程介绍

- 发红包：在本系统中的后端主要是根据用户输入的金额和个数预生成相应的红包随机金额列表，并将红包的总个数及对应的随机金额列表缓存至Redis中，同时将红包的总金额、随机金额列表和红包全局唯一标识串等信息异步记录到相应的数据库表中。
- 抢红包：后端接口首先接收前端用户账号及红包全局唯一标识串等请求信息，并假设用户账号合法性等信息全部校验通过，然后开始处理用户“点红包”的逻辑，主要是从缓存系统中获取当前剩余红包的个数，根据红包个数是否大于0判断是否仍然还有红包。假设剩余红包个数大于0，则开始处理用户“拆红包”的逻辑，这个逻辑主要是从缓存系统的随机金额队列中弹出一个红包金额，并根据金额是否为Null判断是否成功抢到红包。

开发流程介绍

- 值得一提的是，为了保证系统整体开发流程的规范性、可扩展性和接口的健壮性，我们约定了处理用户请求信息后将返回统一的响应格式。
- 这种格式主要是借鉴了HTTP协议的响应模型，即响应信息应当包含状态码、状态码的描述和响应数据。为此在项目的api模块引入两个类，分别是BaseResponse类和StatusCode类。
- 参见实例：RedPackets项目api模块

红包金额随机生成算法实战

- 系统架构的核心部分主要在于“抢红包”逻辑的处理，而是否能抢到红包，主要的决定因素在于红包个数和红包随机金额列表，特别是后者将起到决定性的作用。
- 在本系统中，红包随机金额列表主要是采用“预生成”的方式产生的（这种方式跟微信红包的实时生成方式不一样），即通过给定红包的总金额M和人员总数N，采用某种随机数算法生成红包随机金额列表，并将其存至缓存中，用于“拆红包”逻辑的处理。

红包算法-随机数算法[蒙托卡罗]

- 随机数算法是数学算法中的一种，主要是在算法中引入一个或多个随机因素，通过随机数选择算法的下一步操作，从而产生某种约定范围内的随机数值。
- 蒙托卡罗：这种算法主要是通过建立一个模型，并对模型中的随机变量建立抽样方法，在计算机中反复多次进行模拟测试，最终得到一个或多个估计值，即随机数列表。

红包随机金额生成算法要求

- 当用户发出固定总金额为M、红包个数为N的红包后，由于系统后端采用的是预生成方式，因而将在后端生成N个随机金额的小红包并存储至缓存中，等待着被抢。
- 小红包随机金额的产生对用户而言是透明的，用户也无须知晓红包随机金额的产生原理与规则，因而对于抢红包系统来说，只需要保证系统后端每次对于总金额M和红包个数N生成的小红包金额是随机且平等的，即间接性地保证每个用户抢到的红包金额是随机产生且概率是平等的即可。
- 除了保证预生成红包金额的随机性和概率平等性之外，抢红包系统后端还需保证如图所示的3点要求。

抢红包系统设计要求
（场景：发出一个固定金额的红包，由若干个人来抢）

- 所有人抢到的金额之和等于红包金额，既不能超，也不能少
- 每个人至少抢到1分钱
- 要保证所有人抢到金额的几率相等（由生成红包随机金额的算法决定）

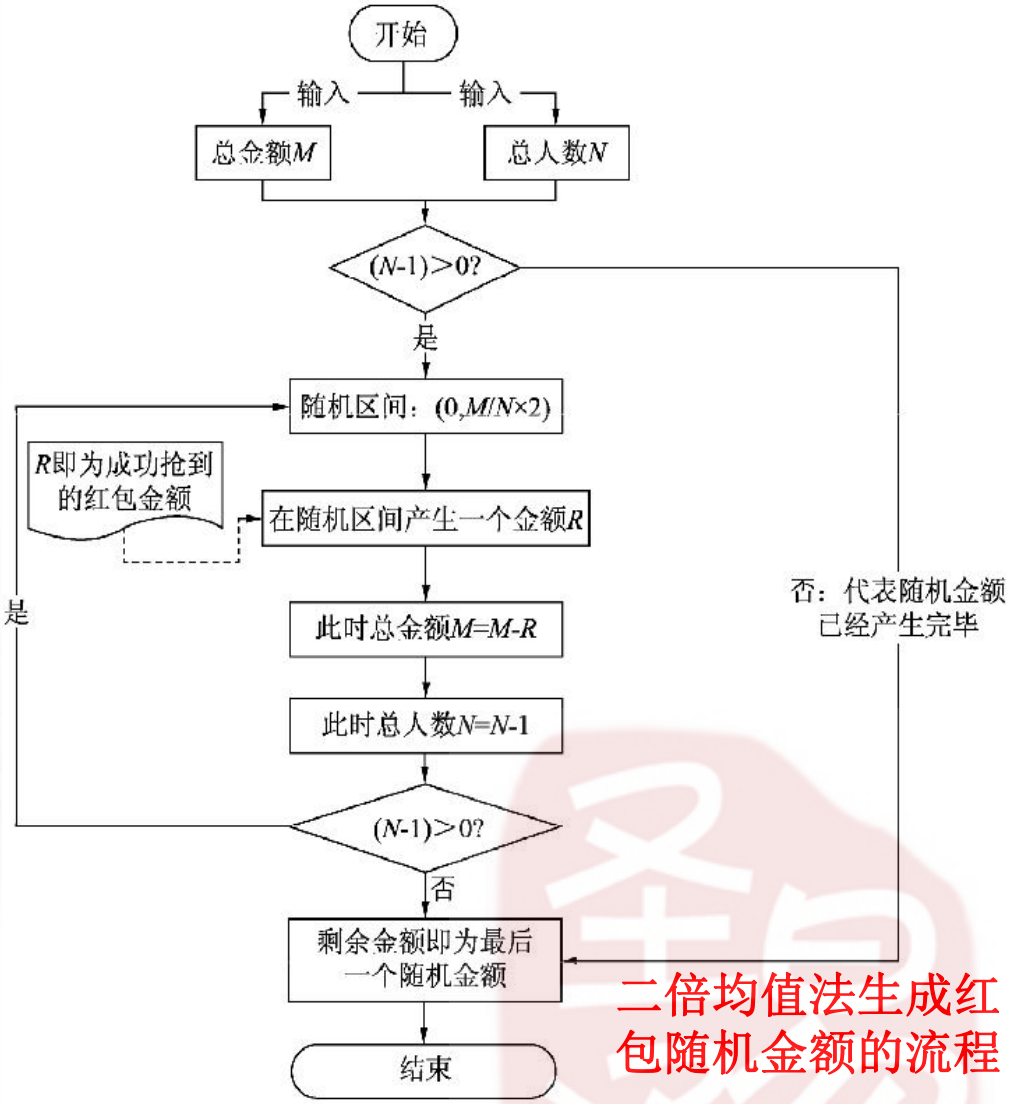
- 应保证所有人抢到的红包金额之和等于总金额M，这一点是毋庸置疑的。
- 每个参与抢红包的用户，当抢到红包（即红包金额不为Null）时，金额应至少是1分钱，即0.01元。
- 应当保证参与抢红包的用户抢到红包金额时，几率是相等的，这一点由于是采用预生成的方式，因而可以交给随机数生成算法进行控制。

红包算法-二倍均值法

- 目前市面上关于红包随机金额的生成算法有许多种，二倍均值法属于其中比较典型的一种。
- 二倍均值算法的核心思想是根据每次剩余的总金额M和剩余人数N，执行M/N再乘以2的操作得到一个边界值E，然后制定一个从0到E的随机区间，在这个随机区间内将产生一个随机金额R，此时总金额M将更新为M-R，剩余人数N更新为N-1。再继续重复上述执行流程，以此类推，直至最终剩余人数N-1为0，即代表随机数已经产生完毕。
- 这一算法的执行流程如下图所示。

二倍均值法流程图

- 该算法的执行流程其实是
- 一个for循环产生数据的
- 过程，循环终止的条件是
- N-1<=0，即代表随机数已
- 经生成完毕。而对于随机
- 数的生成，主要是在约束
- 的随机区间(0,M/N×2)中
- 产生，这样着实可以保证
- 每次数值R产生的随机性
- 和平等性；除此之外，由
- 于每次总金额M的更新采用的是递减的方式，即M=M-R，因而可以保证最终产生的所有随机金额之和等于M。



二倍均值法生成红包随机金额的流程

红包随机金额生成算法实战

- 算法流程图在某种程度上是实战代码的直接体现。从二倍均值法的算法执行流程图可以看出，其核心执行逻辑在于不断地更新总金额M和剩余人数N，并根据M和N组成一个随机区间，最终在这个区间内产生一个随机金额，如此不断地进行循环迭代，直至N-1为0，此时剩余的金额即为最后一个随机金额。该算法的执行流程对应的源代码如下：
- 参见实例：RedPackets项目service模块的RedPacketUtil类
- 值得一提的是，因为随机区间的边界具有“除以2”的操作，因而为了程序处理的方便，上述实战代码中总金额M采用“分”作为单位。在调用该方法时需要注意这一点，即如果发红包者输入的红包金额为“10元”，则后端接口接收到该参数后，调用该方法时总金额参数需赋值为“1000分”。

红包随机金额生成算法单元测试

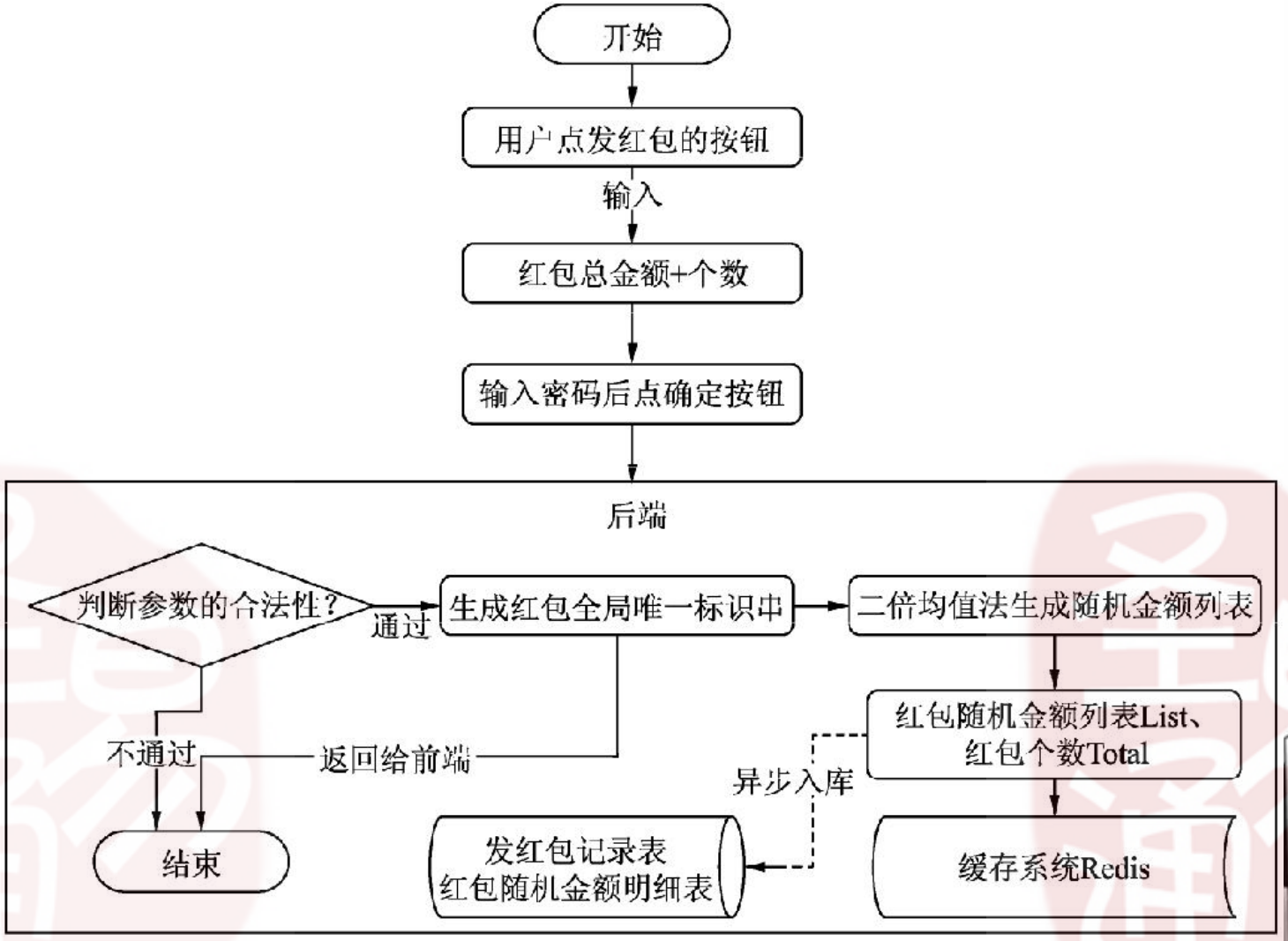
- 对于一名程序员而言，良好的编程习惯列表中除了写代码前的构思和写代码时的规范之外，还应当有代码完成后的自测。“二倍均值法”的代码实战，之前已经实现了，接下来通过Java单元测试方法进行测试。
- 参见实例：RedPackets项目service模块的test目录下的ServiceApplicationTests类
- 从多次单元测试方法运行的结果
- 来看，可以得出小红包金额的生
- 成满足随机性、概率平等性，以
- 及所有小红包金额之和等于总金
- 额等特性。

```
: 总金额=1000分，总个数=10个
: 随机金额为：61分，即0.61元
: 随机金额为：18分，即0.18元
: 随机金额为：119分，即1.19元
: 随机金额为：122分，即1.22元
: 随机金额为：173分，即1.73元
: 随机金额为：145分，即1.45元
: 随机金额为：153分，即1.53元
: 随机金额为：123分，即1.23元
: 随机金额为：47分，即0.47元
: 随机金额为：39分，即0.39元
: 所有随机金额叠加之和=1000分
```

“发红包”模块实战

- 在整个业务流程中，系统后端接口需要根据红包个数N和总金额M采用二倍均值法拆分成多个随机金额，并生成红包的全局唯一标识串返回给前端，前端用户发起抢红包请求时将带上这个标识串参数，从而实现后续的“点红包”“拆红包”流程。
- 标识串参数目的是为了给发出的红包打标记，并根据这一标记去缓存中查询红包个数和随机金额列表等数据。

发红包模块流程


开始
用户点发红包的按钮
输入
红包总金额+个数
输入密码后点确定按钮
后端
判断参数的合法性?
通过
生成红包全局唯一标识串
二倍均值法生成随机金额列表
不通过
返回给前端
红包随机金额列表List、
红包个数Total
异步入库
结束
发红包记录表
红包随机金额明细表
缓存系统Redis

发红包模块流程

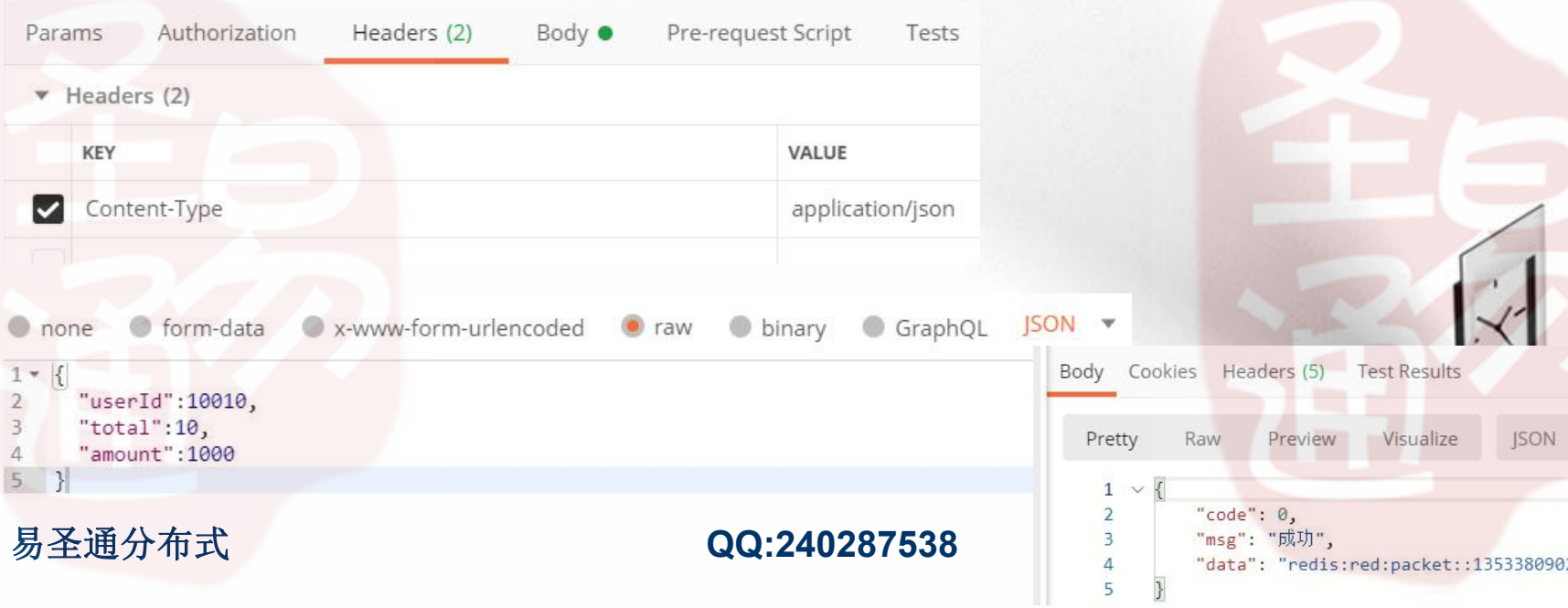
- 在处理“发红包”的请求时，后端接口进需要接收红包总金额和总个数等参数，因而将其封装成实体对象RedPacketDto。
- 参见实例：RedPackets项目api模块的dto包下的RedPacketDto类
- 首先是处理发红包请求的RedPacketController，主要用于接收前端用户请求的参数并执行响应的判断处理逻辑。
- 参见RedPacketController类
- “发红包”请求对应的处理方法需要接受红包总金额、个数和发红包者账号id参数信息，并要求请求方式为Post，数据采用JSON的格式进行提交。

发红包模块流程

- IRedPacketService为红包业务逻辑处理接口，主要包括对“发红包”与“抢红包”逻辑的处理。对应的实现类RedPacketService主要用于处理真正的业务逻辑。
- 参见RedPacketService类
- “红包业务逻辑处理过程数据记录接口”服务IRedService，主要用于将发红包时红包的相关信息与抢红包时用户抢到的红包金额等信息记入数据库中。
- 参见RedService类

发红包模块流程

- 发红包业务模块的完整代码实战已经做完了。下面采用Postman工具测试抢红包系统发红包业务的整体流程。
- 单击运行应用程序的启动入口类MainApplication，观察控制台的输出信息，如果没有报错，则表示项目启动成功。打开Postman，在地址栏中输入“发红包”请求对应的链接，即http://127.0.0.1:8087/red/packet/hand/out，选择请求方式为Post，并在请求体中输入请求参数，如下：

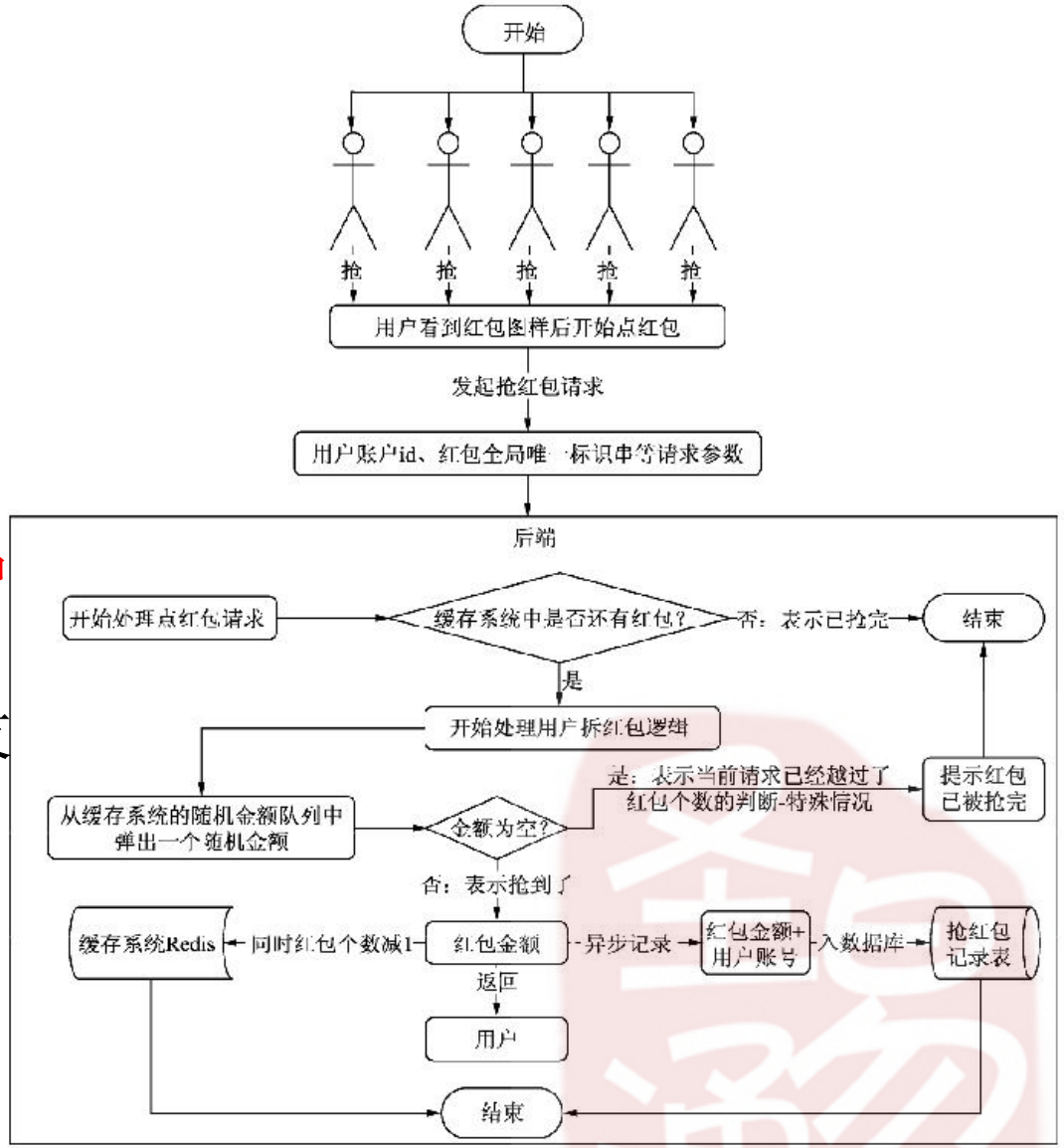


“抢红包”模块实战

- 当群里的成员看到有人发红包时，正常情况下都会点该红包图样，并由此开启后端处理抢红包请求逻辑的节奏。
- 首先后端接口将会接收红包全局唯一标识串和用户账号id，从缓存系统中取出红包的个数，判断个数是否大于0。如果大于0则表示缓存中仍然有红包。
- 然后从缓存系统的随机金额列表中弹出一个随机金额，判断是否为Null。如果不为Null，则表示当前用户抢到红包了，此时需要更新缓存系统的红包个数并异步记录相关信息到数据库中。

业务模块分析

“抢红包”业务模块的整体开发流程。从该流程图中可以看出，前端用户发起抢红包请求，需要带上红包全局唯一标识串resId和当前用户账户id到系统后端接口。其中，resId即为前端发起“发红包”请求后得到的响应结果中的data数值，后端接口将根据这一数值去缓存系统Redis中获取红包剩余个数与随机金额列表。



业务模块分析

- 1.从业务角度分析，抢红包业务模块需要实现两大业务逻辑，包括点红包业务逻辑和拆红包业务逻辑，简单地讲，包含两个“动作”。
- 2.从系统架构角度分析，“抢红包”业务模块对应的后端处理逻辑需要保证接口的稳定性、可扩展性和扛高并发性，对于相应接口的设计需要尽可能做到低耦合和服务的高内聚，我们采用的是面向接口和服务进行编程。
- 3.从技术角度分析，抢红包业务模块对应的后端接口需要频繁地访问缓存系统Redis，用于获取红包剩余个数和随机金额列表，进而判断用户点红包、拆红包是否成功。除此之外，在用户每次成功抢到红包之后，后端接口需要及时更新缓存系统中红包的剩余个数，将相应的信息记入数据库中等。

整体流程

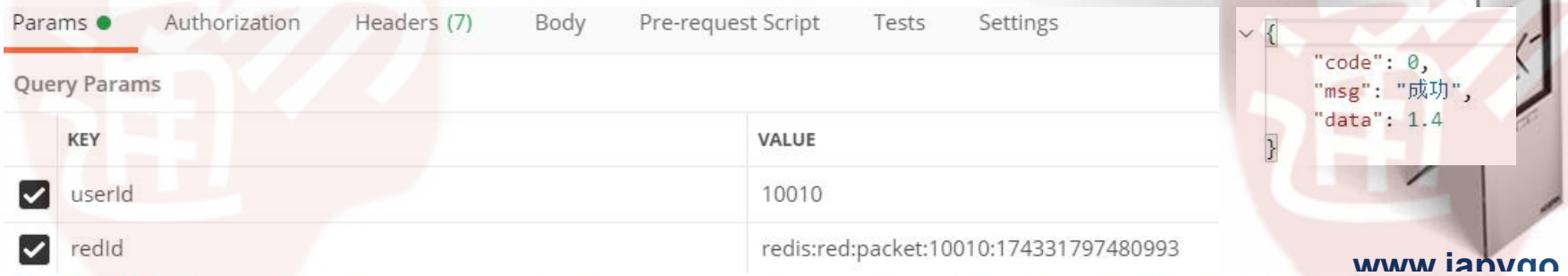
- 1.开发处理发红包请求的RedPacketController类，这个类文件已经在上一节中建立好了，在这里只需要添加一个处理抢红包请求的方法即可。
- 参见实例：RedPacket项目的RedPacketController类
- 注意，为了前端处理方便，后端接口在处理完成“抢红包”的请求之后，直接将抢到的红包金额单位处理成“元”，这对于前端开发而言，可以直接显示该返回结果，而不需要再经过二次处理。
- 2.对于红包业务逻辑处理接口中的抢红包方法，是定义在IRedPacketService接口的rob()方法中的，相应的实现类为RedPacketServiceImpl。
- 参见实例：RedPacket项目的RedPacketServiceImpl类

整体流程

- 3.当用户抢到红包后，需要将当前用户的账号信息及抢到的金额等信息记入数据库中。这一实现逻辑是通过调用redServiceImpl实现类的recordRobRedPacket()方法实现的。参见实例：RedPacket项目的RedServiceImpl类
- 4.至此，抢红包业务模块的整体实现过程就基本上完成了。可以看出，其核心处理逻辑主要在于RedPacketServiceImpl实现类的rob()方法，该方法主要用于实现“点红包”和“拆红包”业务逻辑。

业务模块自测

- 接下来我们对“抢红包”业务模块的整体流程进行自测。
- 由于抢红包，发红包的redId必须一致，因此我们需要以“发红包”业务模块自测中发出的红包作为测试数据，即redId的取值为redis:red:packet:10010:177565921763957
- 打开Postman，在地址栏中输入URL为http://127.0.0.1:8087/red/packet/rob?userId=10010&redId=redis:red:packet:10010:174331797480993，表示当前账号id为10010的用户发起了抢红包的请求，单击Send查看控制台输出及Postman的响应结果可以发现，当前用户抢到了1.4元。而同时，在数据库red_rob_record表中将会异步插入一条该用户抢到的红包金额记录等信息。



: 当前用户抢到红包了:userId=10010 key=redis:red:packet:10010:174331797480993 金额=1.4

总结

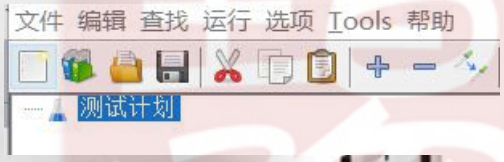
- 在整个自测过程中，缺陷在于Postman不断发起的请求几乎并非“同一时刻”，即这些请求几乎都是“串行”、有时间间隔的，而这并不能达到实际生产环境中“秒级高并发请求”的要求。所谓的“高并发”，其实指的就是在同一时刻突然有成千上万甚至上百万、上千万数量级的请求到达后端接口。
- 例如，春节期间微信群的抢红包，当某个成员发出红包后，正常情况下群里的所有成员均会点该红包图样，发起抢红包的请求。而这些请求中的很大一部分是在同一时刻同时到达后端接口的，在当前“抢红包”业务模块的代码实现过程中，并没有考虑高并发抢红包的情况。众所周知，高并发请求其实本质上是高并发多线程，多线程的高并发如

总结

- 果出现抢占共享资源而不加以控制的话，将会带来各种各样的并发安全问题，数据不一致便是其中典型的一种。
- 综述，我们将重点介绍如何借助高并发压力测试工具Jmeter模拟大数据量的并发请求，并观察由此产生的问题，以及针对出现的问题进行优化与代码问题。

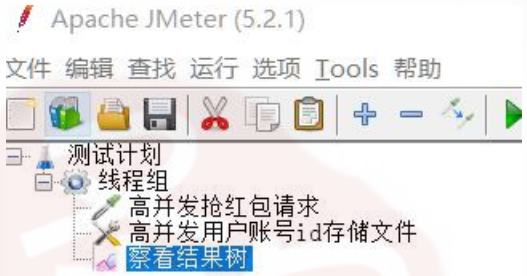
JMeter压力测试高并发抢红包

- Apache Jmeter是Apache组织开发的基于Java的压力测试工具。它可以通过产生来自不同类别的压力，模拟实际生产环境中高并发产生的巨大负载，从而对应用服务器、网络或对象整体性能进行测试，并对产生的测试结果进行分析和反馈。
- 下载完成之后将其解压到某个磁盘目录下，双击进入该文件夹，找到bin文件目录，然后双击jmeter.sh文件，会弹出DOS界面，此时是Jmeter在做一些初始化工作，稍微等待一定的时间，即可进入Jmeter的操作界面。Jmeter的主界面如右图。
- 匪夷所思的是你需要点击选项把该软件的
- 外观调成windows，否则右键不好使！艹。

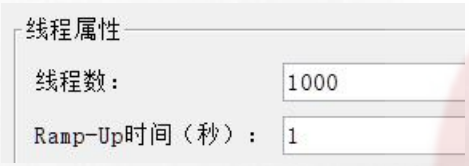


JMeter压力测试高并发抢红包

- 首先右击“文件”选项，新建一个测试计划，然后在该测试计划下新建线程组，最后在线程组下新建“HTTP请求”“CSV数据文件设置”“查看结果树”
- 新建的线程组内容如图4.32所示，在这里暂时指定1秒内并发的线程数，即请求数为10个。



线程组



高并发抢红包请求

JMeter压力测试高并发抢红包

- 然后是“CSV数据文件设置”的内容设置。由于抢红包的高并发主要来源于多个用户同时发起请求，因而需要设定几个固定的用户账户id列表，用于发起高并发请求时从此文件中随机读取用户账户id值，从而赋值给请求参数userId。在这里我们假设当前用户账号id的取值列表为10030~10035，共6个用户。

```
10030,
10031,
10032,
10033,
10034,
10035
```

- 最后是在线程组下新建“察看结果树”选项，目的是为了查看发起请求后系统的响应结果。

JMeter压力测试高并发抢红包

- 至此，我们已经完成了针对系统发起高并发抢红包请求的设置。在进行高并发压力测试之前，因为“抢红包”建立在“发红包”业务的基础上，因而需要通过Postman发起“发红包”请求，从而在缓存系统中生成红包剩余总个数和红包随机金额列表。
- 下面采用userId为10030、total为10、amount为500即5元的测试数据发起“发红包”请求。请求数据如下：单击Send按钮发起请求，即可得到响应结果。

```
{
  "userId":10030,
  "total":10,
  "amount":500
}
```

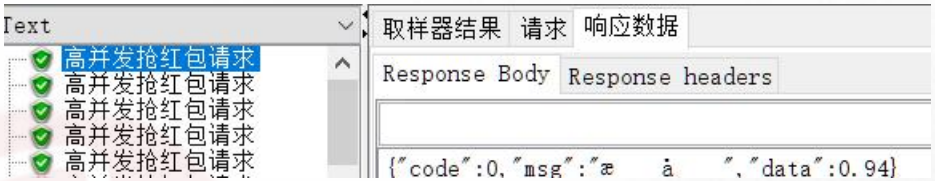
```
{
    "code": 0,
    "msg": "成功",
    "data": "redis:red:packet:10030:178829094505870"
}
```

- 将响应结果中的data取值复制出来，放到前面搭建好的Jmeter并发测试界面中“高并发抢红包请求”的请求参数redId取值中。

名称：	值
userId	${userId}
redId	redis:red:packet:10030:178829094505870

JMeter压力测试高并发抢红包

- 最后调整一下线程组中1秒并发的线程数为1000，表示在同一时刻群里将有1000个用户同时发起抢红包的请求。
- 单击“运行”按钮，即可发起高并发抢红包请求。单击Jmeter的察看结果树，即可看到1000个请求对应的响应结果。



- 看IDEA控制台

```
:userId=10034 key=redis:red:packet:10030:178829094505870 金额=0.69
:userId=10031 key=redis:red:packet:10030:178829094505870 金额=0.19
:userId=10033 key=redis:red:packet:10030:178829094505870 金额=0.17
:userId=10034 key=redis:red:packet:10030:178829094505870 金额=0.27
:userId=10031 key=redis:red:packet:10030:178829094505870 金额=0.74
```

JMeter压力测试高并发抢红包

- 看其输出结果会发现一个很明显的Bug，即同一个用户竟然抢到了多个不同随机金额的小红包，如userId=10031的用户账号，抢到了金额为0.19元、0.74元的小红包。这是很不可思议的大问题，违背了一个用户只能对若干个随机金额的小红包抢一次的规则。这个问题也可以通过查看数据库表red_rob_record的数据记录也可以得出。
- 从技术层面分析，这其实就是高并发多线程产生的并发安全导致的。

问题分析

- 造成这种问题的原因，其实就是我们并没有考虑“秒级同时并发多线程”的情况，当某一时刻的同一用户在“疯狂”地点红包图样时，如果前端不加以控制的话，同一时间的同一个用户将发起多个抢红包请求。当后端接收到这些请求时，将很有可能同时进行“缓存系统中是否有红包”的判断并成功通过，然后执行后面弹出红包随机金额的业务逻辑，导致同一个用户抢到多个红包的情况发生。
- 其实就是后端接口并没有考虑到高并发请求的情况。更深入地讲，其原因在于当前请求还没有处理完核心业务逻辑时，其他同样的请求已经到来，导致后端接口几乎来不及做重复判断的逻辑。

优化方案

- 在传统的单体Java应用中，为了解决多线程高并发的安全问题，最常见的做法是在核心的业务逻辑代码中加锁操作（同步控制操作），即加Synchronized关键字。
- 然而在微服务、分布式系统架构时代，这种做法是行不通的。因为Synchronized关键字是跟单一服务节点所在的JVM相关联，而分布式系统架构下的服务一般是部署在不同的节点（服务器）下，从而当出现高并发请求时，Synchronized同步操作将显得“力不从心”！
- 这种方案既要保证单一节点核心业务代码的同步控制，也要保证当扩展到多个节点部署时同样能实现核心业务逻辑代码的同步控制，这就是“分布式锁”出现的初衷。

优化方案

- 分布式锁的出现主要是为了解决分布式系统中高并发请求时并发访问共享资源导致并发安全的问题。
- 目前关于分布式锁的实现有许多种，典型的包括基于数据库级别的乐观锁和悲观锁，以及基于Redis的原子操作实现分布式锁和基于ZooKeeper实现分布式锁等。
- 我们将基于Redis的分布式锁解决抢红包系统中出现的高并发问题。
- Redis底层架构是采用单线程进行设计的，因而它提供的这些操作也是单线程的，即其操作具备原子性。
- 原子性：指的是同一时刻只能有一个线程处理核心业务逻辑，当有其他线程对应的请求过来时，如果前面的线程没有处理完毕，那么当前线程将进入等待状态（堵塞），直到前面的线程处理完毕。

Redis分布式锁实战

- “抢红包”的业务模块，其核心处理逻辑在于“拆红包”的操作。可以通过Redis的原子操作setIfAbsent()方法对该业务逻辑加分布式锁，表示如果当前的Key不存在于缓存中，则设置其对应的Value，该方法的操作结果返回True；如果当前的Key已经存在于缓存中，则设置其对应的Value失败，即该方法的操作结果将返回False。
- 由于该方法具备原子性（单线程）操作的特性，因而当多个并发的线程同一时刻调用setIfAbsent()时，Redis的底层是会将线程加入“队列”排队处理的。我们采用setIfAbsent()方法改造“抢红包”业务逻辑对应的代码。改造后的rob()方法如下：
- 参见实例：RedPacket项目的RedServiceImpl类的rob方法
- 首先还得通过postman产生红包，在使用JMeter进行压力测试。

```
userId=10033 key=redis:red:packet:10030:223441452079469 金额=0.04
userId=10030 key=redis:red:packet:10030:223441452079469 金额=0.98
userId=10035 key=redis:red:packet:10030:223441452079469 金额=0.73
userId=10032 key=redis:red:packet:10030:223441452079469 金额=0.74
userId=10034 key=redis:red:packet:10030:223441452079469 金额=0.73
userId=10031 key=redis:red:packet:10030:223441452079469 金额=0.6
```

- 不足之处：当发出的红包没有被抢完时，请问该如何处理剩下的红包？如发出一个固定总金额为10元的红包，设定个数为10个，但是最终却只有6个被抢到，那么剩下的4个红包汇总的金额该如何处理？对于目前的系统而言，是暂时没有实现这一功能的，算是其中的不足之处吧。对于这个问题的解决，读者可以尝试自己实现。（提示：最终剩下的金额当然是归还给发红包者。）

易圣通

感谢您的关注