180Wtps超高并发、大流量生产案例：字节钱包 架构与落地方案

1. 背景&挑战&目标

1.1 业务背景

（1）**支持八端**：2022 年字节系产品春节活动需要支持八端 APP 产品（包含抖音/抖音火山/抖音极速版/西瓜/头条/头条极速版/番茄小说/番茄畅听）的奖励互通。用户在上述任意一端都可以参与活动，得到的奖励在其他端都可以提现与使用。

（2）**玩法多变**：主要有集卡、朋友页红包雨、红包雨、集卡开奖与烟火大会等。

（3）**多种奖励**：奖励类型包含现金红包、补贴视频红包、商业化广告券、电商券、支付券、消费金融券、保险券、信用卡优惠券、喜茶券、电影票券、dou+券、抖音文创券、头像挂件等。

1.2 核心挑战

（1）设计&实现八端奖励入账与展示互通的大流量的方案，最高预估有 360w QPS 发奖。

（2）多种发奖励的场景，玩法多变；奖励类型多，共 10 余种奖励。对接多个下游系统。

（3）从奖励系统稳定性、用户体验、资金安全与运营基础能力全方位保障，确保活动顺利进行 。

1.3 最终目标

（1）**奖励入账**：设计与实现八端奖励互通的奖励入账系统，对接多个奖励下游系统，抹平不同奖励下游的差异，对上游屏蔽底层奖励入账细节，设计统一的接口协议提供给业务上游。提供统一的错误处理机制，入账幂等能力和奖励预算控制。

（2）**奖励展示/使用**：设计与实现活动钱包页，支持在八端展示用户所获得的奖励，支持用户查看、提现（现金），使用卡券/挂件等能力。

（3）**基础能力**：

- 【基础 sdk】提供查询红包余额、累计收入、用户在春节活动是否获得过奖励等基础 sdk，供业务方查询使用。
- 【预算控制】与上游奖励发放端算法策略打通，实现大流量卡券入账的库存控制能力，防止超发。
- 【提现控制】在除夕当天多轮奖励发放后，提供用户提现的灰度放量能力、提现时尚未入账的处理能力。
- 【运营干预】活动页面灵活的运营配置能力，支持快速发布公告，及时触达用户。为应对黑天鹅事件，支持批量卡券和红包补发能力。

（4）**稳定性保障**：在大流量的入账场景下，保证钱包核心路径稳定性与完善，通过常用稳定性保障手段如资源扩容、限流、熔断、降级、兜底、资源隔离等方式保证用户奖励方向的核心体验。

（5）**资金安全**：在大流量的入账场景下，通过幂等、对账、监控与报警等机制，保证资金安全，保证用户资产应发尽发，不少发。

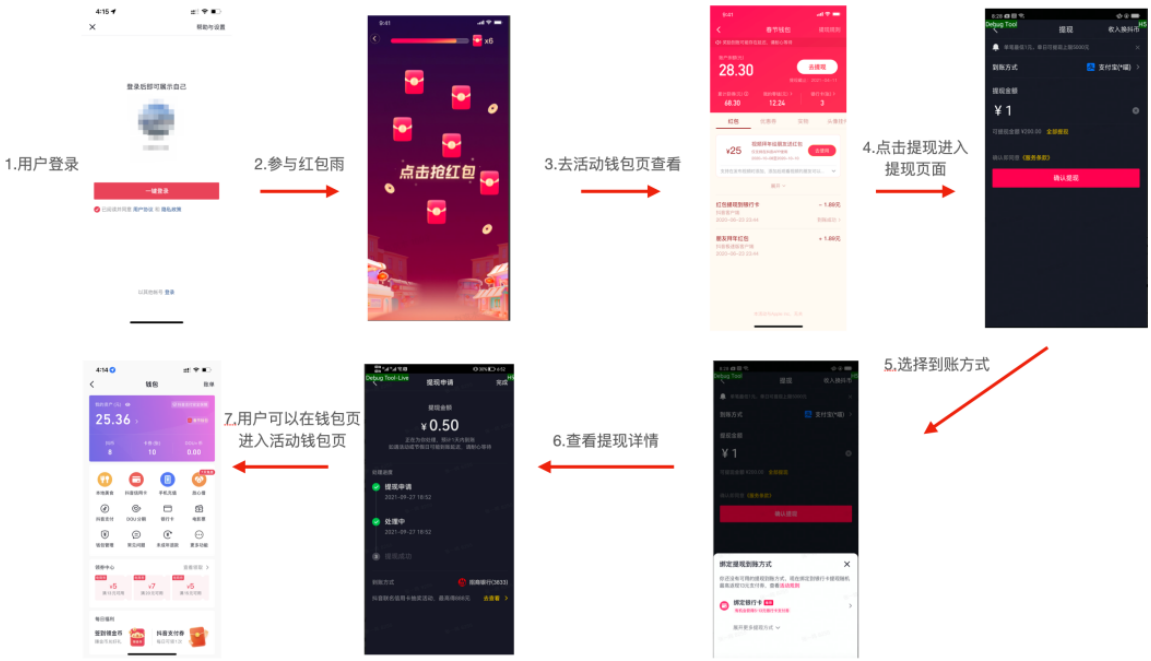
（6）**活动隔离**：实现内部测试活动、灰度放量活动和正式春节活动三个阶段的奖励入账与展示的数据隔离，不互相影响。

2. 产品需求介绍

用户可以在任意一端参与字节的春节活动获取奖励，以抖音红包雨现金红包入账场景为例，具体的业务流程如下：

登录抖音 → 参与活动 → 活动钱包页 → 点击提现按钮 → 进入提现页面 → 进行提现 → 提现结果页，

另外从钱包页也可以进入活动钱包页。



奖励发放核心场景：

1. **集卡**：集卡抽卡时发放各类卡券，集卡锦鲤还会发放大额现金红包，集卡开奖时发放瓜分奖金和优惠券；
2. **红包雨**：发红包、卡券以及视频补贴红包，其中红包和卡券最高分别 180w QPS；
3. **烟火大会**：发红包、卡券以及头像挂件。

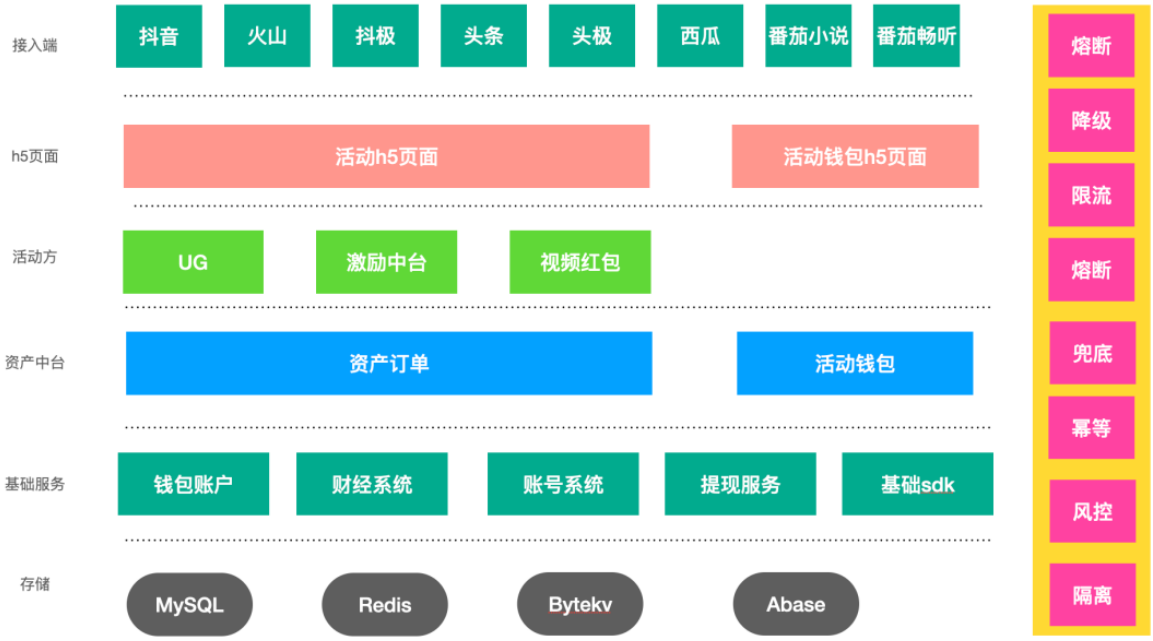
3. 钱包资产中台设计与实现

在 2022 年春节活动中，UG 主要负责活动的玩法实现，包含集卡、红包雨以及烟火大会等具体的活动相关业务逻辑和稳定性保障。

而钱包方向定位是大流量场景下实现奖励入账、奖励展示、奖励使用与资金安全保障的相关任务。

其中资产中台负责**奖励发放与奖励展示**部分。

3.1 春节资产资产中台总体架构图如下：



钱包资产中台核心系统划分如下：

1. **资产订单层**：收敛八端奖励入账链路，提供统一的接口协议对接上游活动业务方如 UG、激励中台、视频红包等的奖励发放功能，同时对上游屏蔽对接奖励业务下游的逻辑处理，支持预算控制、补偿、订单号幂等。
2. **活动钱包 api 层**：收敛八端奖励展示链路，同时支持大流量场景

3.2 资产订单中心设计

核心发放模型：

说明：

1. 活动 ID 唯一区分一个活动，本次春节分配了一个单独的母活动 ID
2. 场景 ID 和具体的一种奖励类型一一对应，定义该场景下发奖励的唯一配置，场景 ID 可以配置的能力有：发奖励账单文案；是否需要补偿；限流配置；是否进行库存控制；是否要进行对账。提供可插拔的能力，供业务可选接入。

实现效果：

1. 实现不同活动之间的配置隔离
2. 每个活动的配置呈树状结构，实现一个活动发多种奖励，一种奖励发多种奖励 ID
3. 一种奖励 ID 可以有多种分发场景，支持不同场景的个性化配置

订单号设计：

资产订单层支持订单号维度的发奖幂等，订单号设计逻辑为

```
${actID}_${scene_id}_${rain_id}_${award_type}_${statge}`
```

从单号设计层面保证不超发，每个场景的奖励用户最多只领一次。

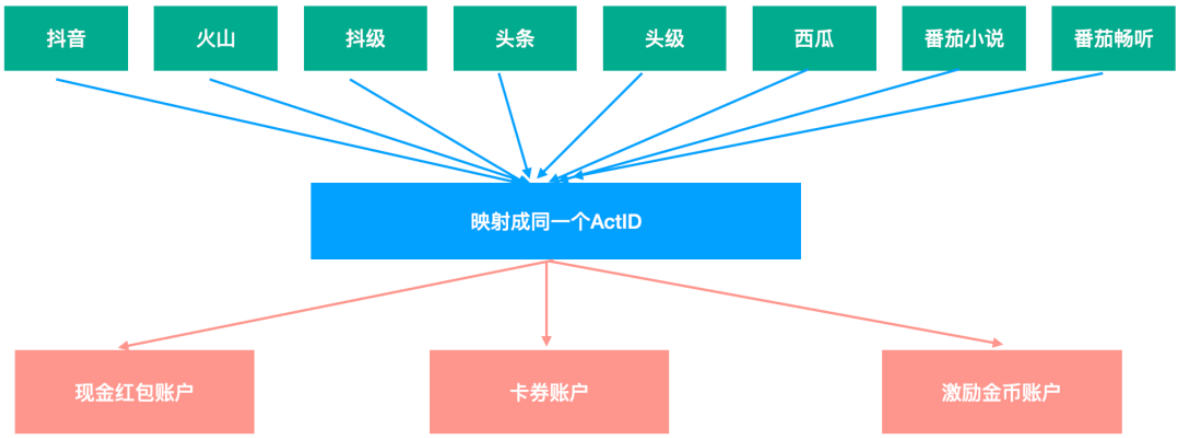
4. 核心难点问题解决

4.1 难点一：支持八端奖励数据互通

前文背景已经介绍过了，参与 2022 年春节活动一共有八个产品端，其中抖音系和头条系 APP 是不同的账号体系，所以不能通过用户 ID 打通奖励互通。具体解决方案是字节账号中台打通了八端的账号体系给每个用户生成唯一的 actID（手机号优先级最高，如果不同端登录的手机号一样，在不同端的 actID 是一致的）。

钱包侧基于字节账号中台提供的唯一 actID 基础上，设计实现了支持八端奖励入账、查看与使用的通用方案，即每个用户的奖励数据是绑定在 actID 上的，入账和查询是通过 actID 维度实现的，即可实现八端奖励互通。

示意图如下：



4.2 难点二：高场景下的奖励入账实现

每年的春节活动，发现金红包都是最关键的一环，今年也不例外。有几个原因如下：

1. 预估发现金红包最大流量有 180w TPS。
2. 现金红包本身价值高，需要保证资金安全。
3. 用户对现金的敏感度很高，在保证用户体验与功能完整性同时也要考虑成本问题。

终上所述，**发现金红包面临比较大的技术挑战。**

发红包其实是一种交易行为，资金流走向是从公司成本出然后进入个人账户。

（1）从技术方案上是要支持订单号维度的幂等，同一订单号多次请求只入账一次。订单号生成逻辑为

```
`${actID}_${scene_id}_${rain_id}_${award_type}_${statge}`
```

从单号设计层面保证不超发。

（2）支持高并发，有以下 2 个传统方案：

具体方案类型	实现思路	优点	缺点
同步入账	申请和预估流量相同的计算和存储资源	1.开发简单； 2.不容易出错；	**浪费存储成本。** 拿账户数据库举例，经实际压测结果：支持 30w 发红包需要 152 个数据库实例，如果支持 180w 发红包，至少需要 1152 个数据库实例，还没有算上 tce 和 redis 等其他计算和存储资源。
异步入账	申请部分计算和存储资源资源，实际入账能力与预估有一定差值	1.开发简单； 2.不容易出错； 3.不浪费资源；	**用户体验受到很大影响。** 入账延迟较大，以今年活动举例会有十几分钟延迟。用户参与玩法得到奖励后在活动钱包页看不到奖励，也无法进行提现，会有大量客诉，影响抖音活动的效果。

以上两种传统意义上的技术方案都有明显的缺点，那么进行思考，既能相对节约资源又能保证用户体验的方案是什么?
最终采用的是红包雨 token 方案，具体方案是使用异步入账加较少量分布式存储和较复杂方案来实现，下面具体介绍一下。

4.2.1 红包雨 token 方案:

本次春节活动在红包雨/集卡开奖/烟火大会的活动下有超大流量发红包的场景，

前文介绍过发奖 QPS 最高预估有 180w QPS，按照现有的账户入账设计，需要大量存储和计算资源支撑，

根据预估发放红包数/产品最大可接受发放时间，计算得到钱包实际入账最低要支持的 TPS 为 30w，所以实际发放中有压单的过程。

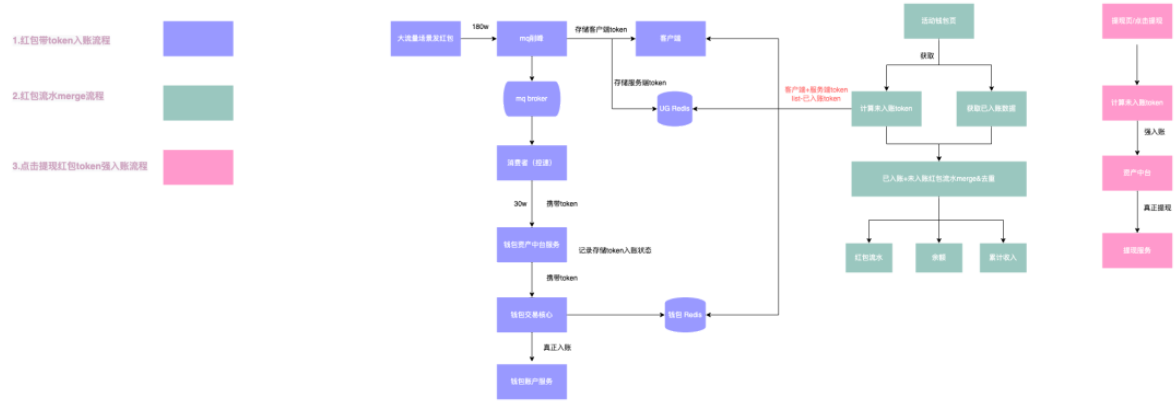
设计目标:

在活动预估给用户发放（180w）与实际入账（30w）有很大 gap 的情况下，保证用户的核心体验。用户在前端页面查看与使用过当中不能感知压单的过程，即查看与使用体验不能受到影响，相关展示的数据包含余额，累计收入与红包流水，使用包含提现等。

具体设计方案:

我们在大流量场景下每次给用户发红包会生成一个加密 token（**使用非对称加密**，包含发红包的元信息：红包金额，actID，与发放时间等），分别存储在客户端和服务端（**容灾互备**），每个用户有个 token 列表。每次发红包的时候会在 Redis 里记录该 token 的入账状态，然后用户在活动钱包页看到的现金红包流水、余额等数据，是合并已入账红包列表+token 列表-已入账/入账中 token 列表的结果。同时为保证用户提现体验不感知红包压单流程，在进入提现页或者点击提现时将未入账的 token 列表进行强制入账，保证用户提现时账户的余额为应入账总金额，不 block 用户提现流程。

示意图如下:



token 数据结构：

token 使用的是 pb 格式，经单测验证存储消耗实际比使用 json 少了一倍，节约请求网络的带宽和存储成本；同时序列化与反序列化消耗 cpu 也有降低。

```
// 红包雨token结构
type RedPacketToken struct {
    AppID      int64  `protobuf: varint,1,opt  json: AppID,omitempty ` // 端ID
    ActID     int64  `protobuf: varint,2,opt  json: UserID,omitempty ` // ActID
    ActivityID string `protobuf: bytes,3,opt  json: ActivityID,omitempty ` // 活动ID
    SceneID    string `protobuf: bytes,4,opt  json: SceneID,omitempty ` // 场景ID
    Amount     int64  `protobuf: varint,5,opt  json: Amount,omitempty ` // 红包金额
    OutTradeNo string `protobuf: bytes,6,opt  json: OutTradeNo,omitempty ` // 订单号
    OpenTime   int64  `protobuf: varint,7,opt  json: OpenTime,omitempty ` // 开奖时间
    RainID     int32  `protobuf: varint,8,opt,name=rainID  json: rainID,omitempty ` // 红包雨ID
    Status     int64  `protobuf: varint,9,opt,name=status  json: status,omitempty ` //入账状态
}
```

token 状态机流转：

在调用账户真正入账之前会置为处理中（2）状态，调用账户成功为成功（8）状态，发红包没有失败的情况，后续都是可以重试成功的。

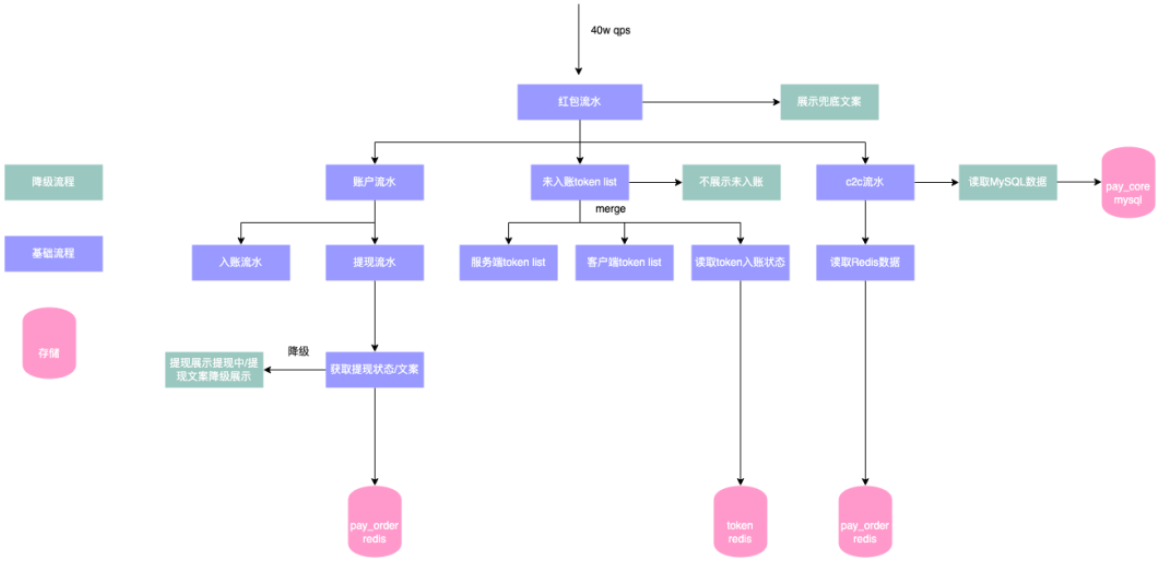
token 安全性保障：

采用非对称加密算法来保障存储在的客户端尽可能不被破解，其中加密算法为秘密仓库，限制其他人访问。同时考虑极端情况下如果 token 加密算法被黑产破译，可监控报警发现，可降级。

4.2.2 活动钱包页展示红包流水

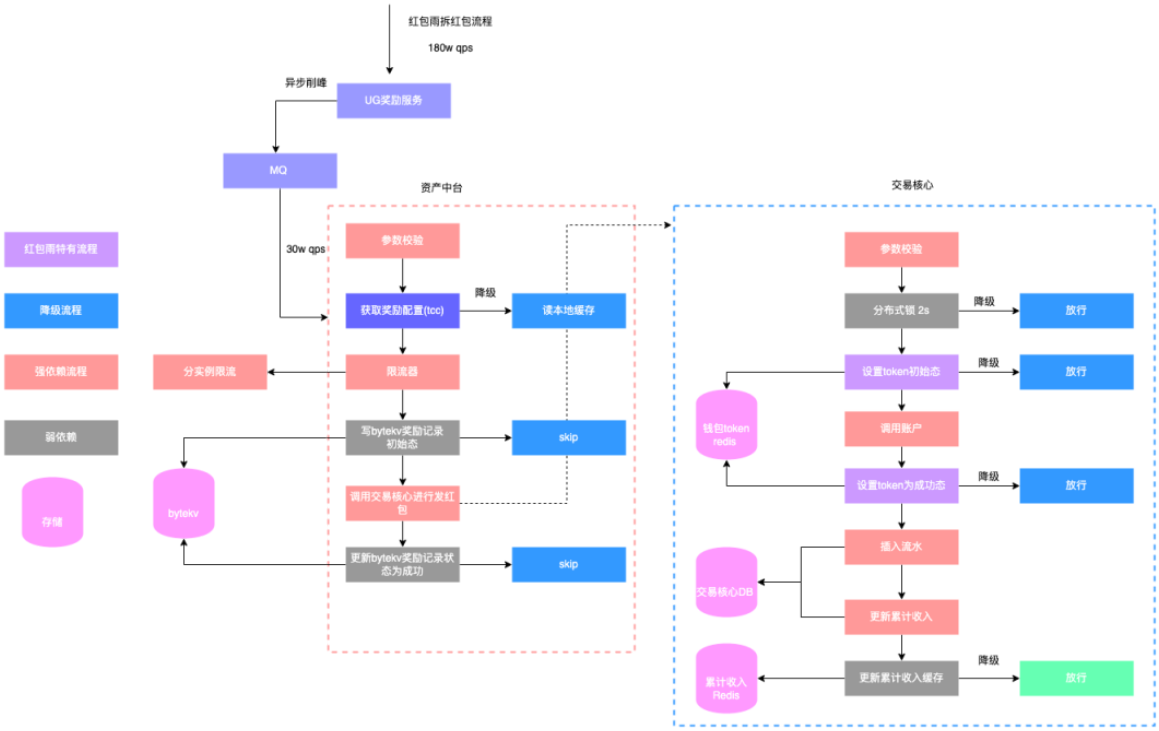
需求背景：

活动钱包页展示的红包流水是现金红包入账流水、提现流水、c2c 红包流水三个数据源的合并，按照创建时间倒叙排列，需要支持分页，可降级，保证用户体验不感知发现金红包压单过程。



4.3 难点三：发奖励链路依赖多的稳定性保障

发红包流程降级示意图如下：



根据历史经验，实现的功能越复杂，依赖会变多，对应的稳定性风险就越高，那么如何保证高依赖的系统稳定性呢？

解决方案：

现金红包入账最基础要保障的功能是将用户得到的红包进行入账，同时支持幂等与预算控制（避免超发），红包账户的幂等设计强依赖数据库保持事务一致性。但是如果极端情况发生，中间的链路可能会出现问题，如果是弱依赖需要支持降级掉，不影响发放主流程。钱包方向发红包**最短路径**为依赖服务实例计算资源和 MySQL 存储资源实现现金红包入账。

发红包强弱依赖梳理图示：

psm	依赖服务	是否强依赖	降级方案	降级后影响
资产中台	tcc	**是**	降级读本地缓存	无
	bytkekv	否	主动降级开关，跳过 bytekv，依赖下游做幂等	无
资金交易层	分布式锁 Redis	否	被动降级，调用失败，直接跳过	基本无
	token Redis	否	主动降级开关，不调用 Redis	用户能感知到入账有延迟，会有很多客诉
	MySQL	**是**	主有问题，联系 dba 切主	故障期间发红包不可用

4.4 难点四：大流量发卡券预算控制

需求背景：

春节活动除夕晚上 7 点半会开始烟火大会，是大流量集中发券的一个场景，钱包侧与算法策略配合进行卡券发放库存控制，防止超发。

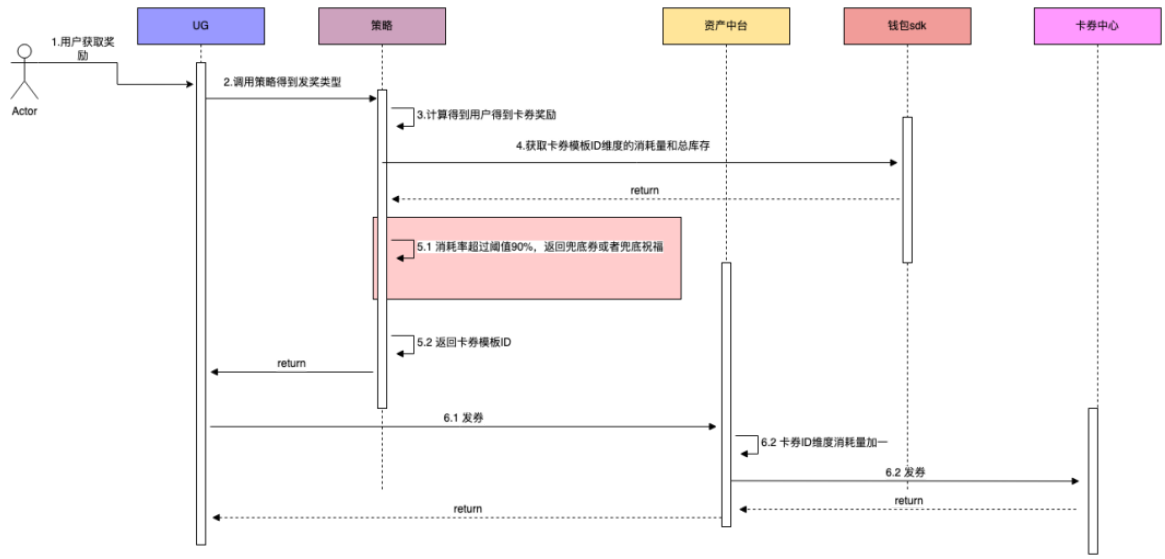
具体实现：

（1）钱包资产中台维护每个卡券模板 ID 的消耗发放量。

（2）每次卡券发放前算法策略会读取钱包 sdk 获取该卡券模板 ID 的消耗量以及总库存数。同时会设置一个阈值，如果卡券剩余量小于 10%后不发这个券（使用兜底券或者祝福语进行兜底）。

（3） 同时钱包资产中台方向在发券流程累计每个券模板 ID 的消耗量（使用 Redis incr 命令原子累加消耗量），然后与总活动库存进行比对，如果消耗量大于总库存数则拒绝掉，防止超发，也是一个兜底流程。

具体流程图：



优化方向：

（1）大流量下使用 Redis 计数，单 key 会存在热 key 问题，需要拆分 key 来解决。

（2）大流量场景下操作 Redis 会存在超时问题，返回上游处理中，上游继续重试发券会多消耗库存少发，本次春节活动实际活动库存在预估库存基础上加了 5%的量级来缓解超时带来的少发问题。

4.5 难点五：高 QPS 场景下的热 key 的读取和写入稳定性保障

需求背景：

在除夕晚上 7 点半开始会开始烟火大会活动，展示所有红包雨与烟火大会红包的实时累计发放总额，最大流量预估读取有 180wQPS，写入 30wQPS。

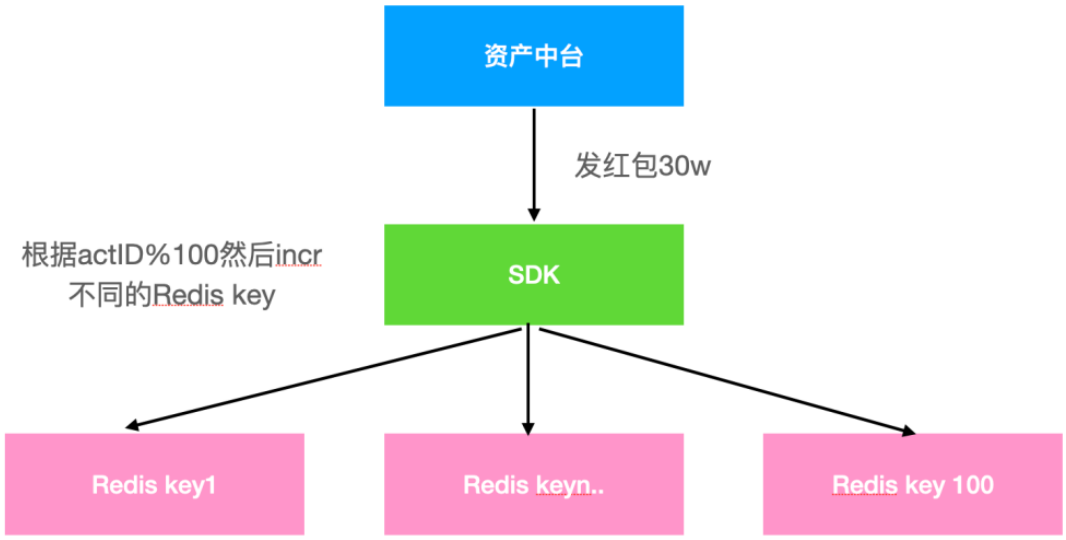
这是典型的超大流量，热点 key、更新延迟不敏感，非数据强一致性场景（数字是一直累加），同时要做好**容灾降级处理**，最后实际活动展示的金额与产品预计发放数值误差小于 1%。

4.5.1 方案一

提供 sdk 接入方式，复用了主会场机器实例的资源。高 QPS 下的读取和写入单 key，比较容易想到的是使用 Redis 分布式缓存来进行实现，但是单 key 读取和写入的会打到一个实例上，压测过单实例的瓶颈为 3w QPS。所以做的一个优化是拆分多个 key，然后用本地缓存兜底。

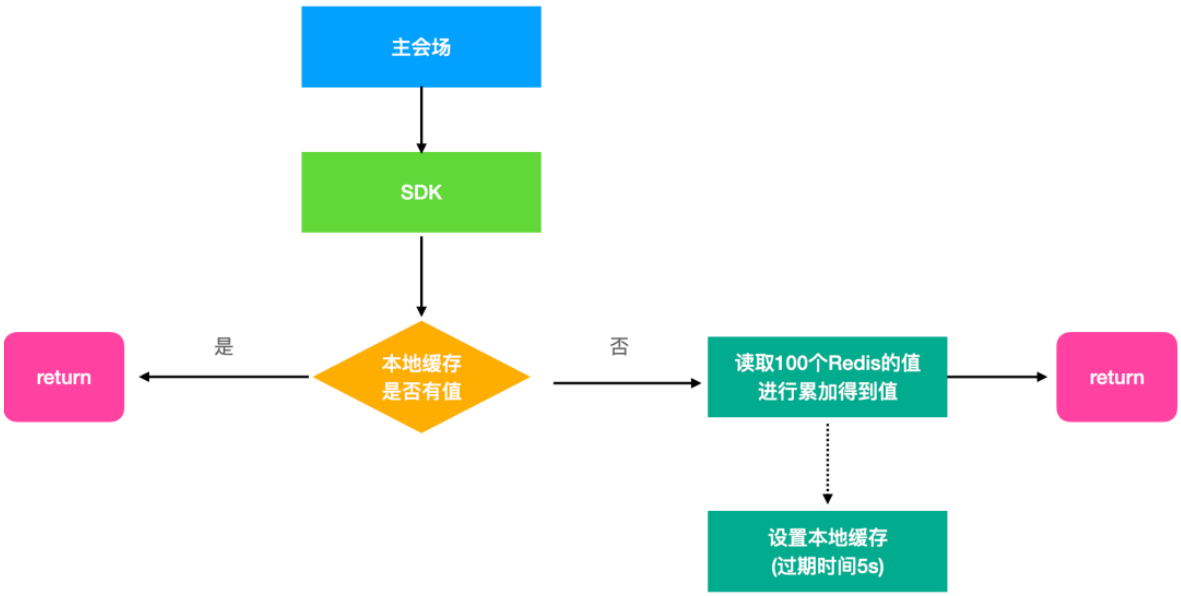
具体写入流程：

设计拆分 100 个 key，每次发红包根据请求的 actID%100 使用 incr 命令累加该数字，因为不能保证幂等性，所以超时不重试。



读取流程：

与写入流程类似，优先读取本地缓存，如果本地缓存值为为 0，那么去读取各个 Redis 的 key 值累加到一起，进行返回。



问题：

（1）拆分 100 个 key 会出现读扩散的问题，需要申请较多 Redis 资源，存储成本比较高。而且可能存在读取超时问题，不能保证一次读取所有 key 都读取成功，故返回的结果可能会较上一次有减少。

（2）容灾方案方面，如果申请备份 Redis，也需要较多的存储资源，需要的额外存储成本。

4.5.2 方案二

设计思路：

在方案一实现的基础上进行优化，并且要考虑数字不断累加、节约成本与实现容灾方案。在写场景，通过本地缓存进行合并写请求进行原子性累加，读场景返回本地缓存的值，减少额外的存储资源占用。使用 Redis 实现中心化存储，最终大家读到的值都是一样的。

具体设计方案：

每个 docker 实例启动时都会执行定时任务，分为读 Redis 任务和写 Redis 任务。

读取流程：

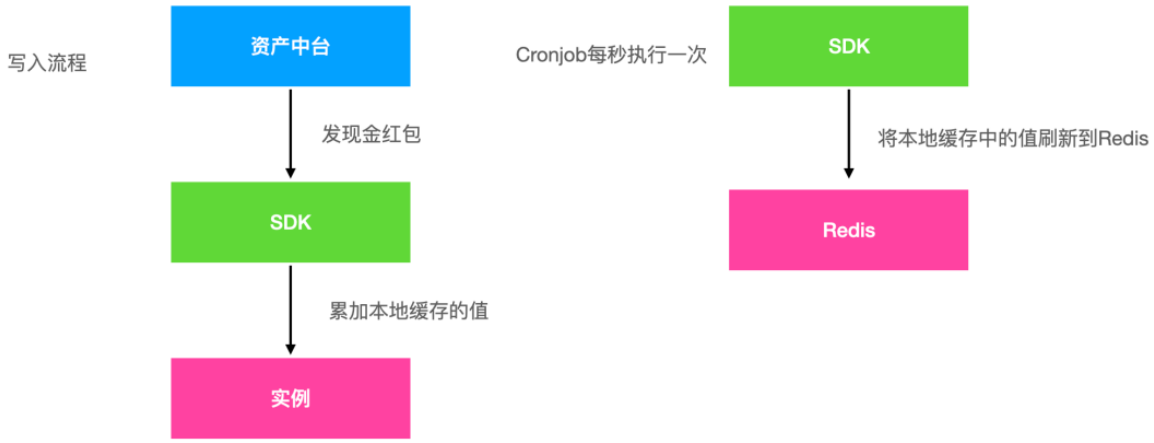
1. 本地的定时任务每秒执行一次，读取 Redis 单 key 的值，如果获取到的值大于本地缓存那么更新本地缓存的值。
2. 对外暴露的 sdk 直接返回本地缓存的值即可。
3. 有个问题需要注意下，每次实例启动第一秒内是没有数据的，所以会阻塞读，等有数据再返回。

写入流程：

1. 因为读取都是读取本地缓存（本地缓存不过期），所以处理好并发情况下的写即可。
2. 本地缓存写变量使用 go 的 atomic.AddInt64 支持原子性累加本地写缓存的值。
3. 每次执行更新 Redis 的定时任务，先将本地写缓存复制到 amount 变量，然后再将本地写缓存原子性减去 amount 的值，最后将 amount 的值 incr 到 Redis 单 key 上，实现 Redis 的单 key 的值一直累加。
4. 容灾方案是使用备份 Redis 集群，写入时进行双写，一旦主机群挂掉，设计了一个配置开关支持读取备份 Redis。两个 Redis 集群的数据一致性，通过定时任务兜底实现。

本方案调用 Redis 的流量是跟实例数成正比，经调研读取侧的服务为主会场实例数 2 万个，写入侧服务为资产中台实例数 8 千个，所以实际 Redis 要支持的 QPS 为 2.8 万/定时任务执行间隔（单位为 s），经压测验证 Redis 单实例可以支持单 key2 万 get，8k incr 的操作，所以设置定时任务的**执行时间间隔是 1s**，如果实例数更多可以考虑延长执行时间间隔。

具体写入流程图如下：



4.5.3 方案对比

	优点	缺点
方案一	1. 实现成本简单	1. 浪费存储资源； 2. 难以做容灾； 3. 不能做到一直累加；
方案二	1. 节约资源； 2. 容灾方案比较简单，同时也节约资源成本；	1. 实现稍复杂，需要考虑好并发原子性累加问题

结论：

从实现效果，资源成本和容灾等方面考虑，最终选择了方案二上线。

4.6 难点六：进行母活动与子活动的平滑切换

需求背景：

为了保证本次春节活动的最终上线效果和交付质量，实际上分了三个阶段进行的。

（1）第一阶段是内部人员测试阶段。

（2）第二个阶段是外部演练阶段，圈定部分外部用户进行春节活动功能的验证（灰度放量），也是发现暴露问题以及验证对应解决机制最有效的手段，影响面可控。

（3）第三个阶段是正式春节活动。

而产品的需求是这三个阶段是分别独立的阶段，包含用户获得奖励、展示与使用奖励都是隔离的。

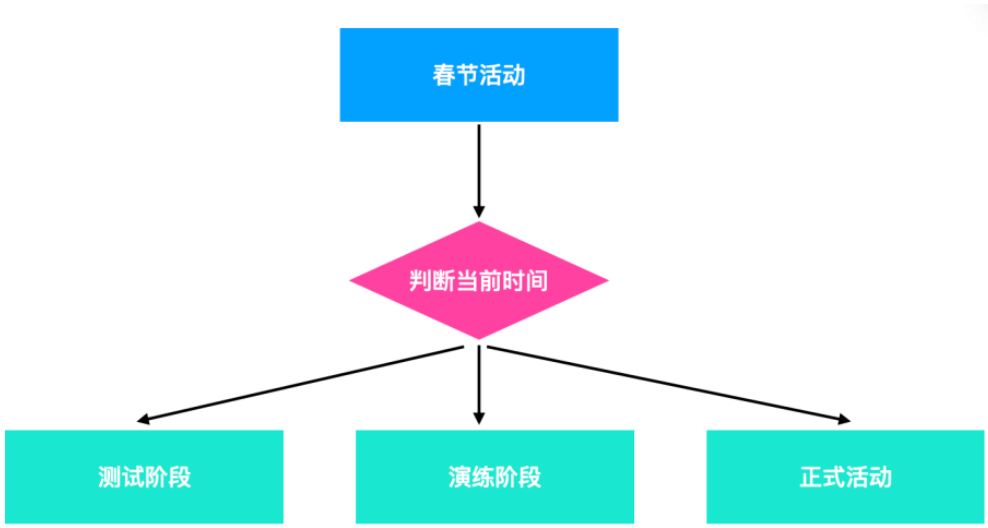
技术挑战：

有多个上游调用钱包发奖励，同时钱包有多个奖励业务下游，所以大家一起改本身沟通成本较高，配置出错的概率就比较大，而且不能同步改，会有较大的技术安全隐患。

设计思路：

作为奖励入账的唯一入口，钱包资产中台收敛了整个活动配置切换的实现。设计出母活动和子活动的分层配置，上游请求参数统一传母活动 ID 代表春节活动，钱包资产中台根据请求时间决定采用哪个子活动配置进行发奖，以此来实现不同时间段不同活动的产品需求。降低了沟通成本，减少了配置出错的概率，并且可以同步切换，较大地提升了研发与测试人效。

示意图：



4.7 难点七：大流量场景下资金安全保障

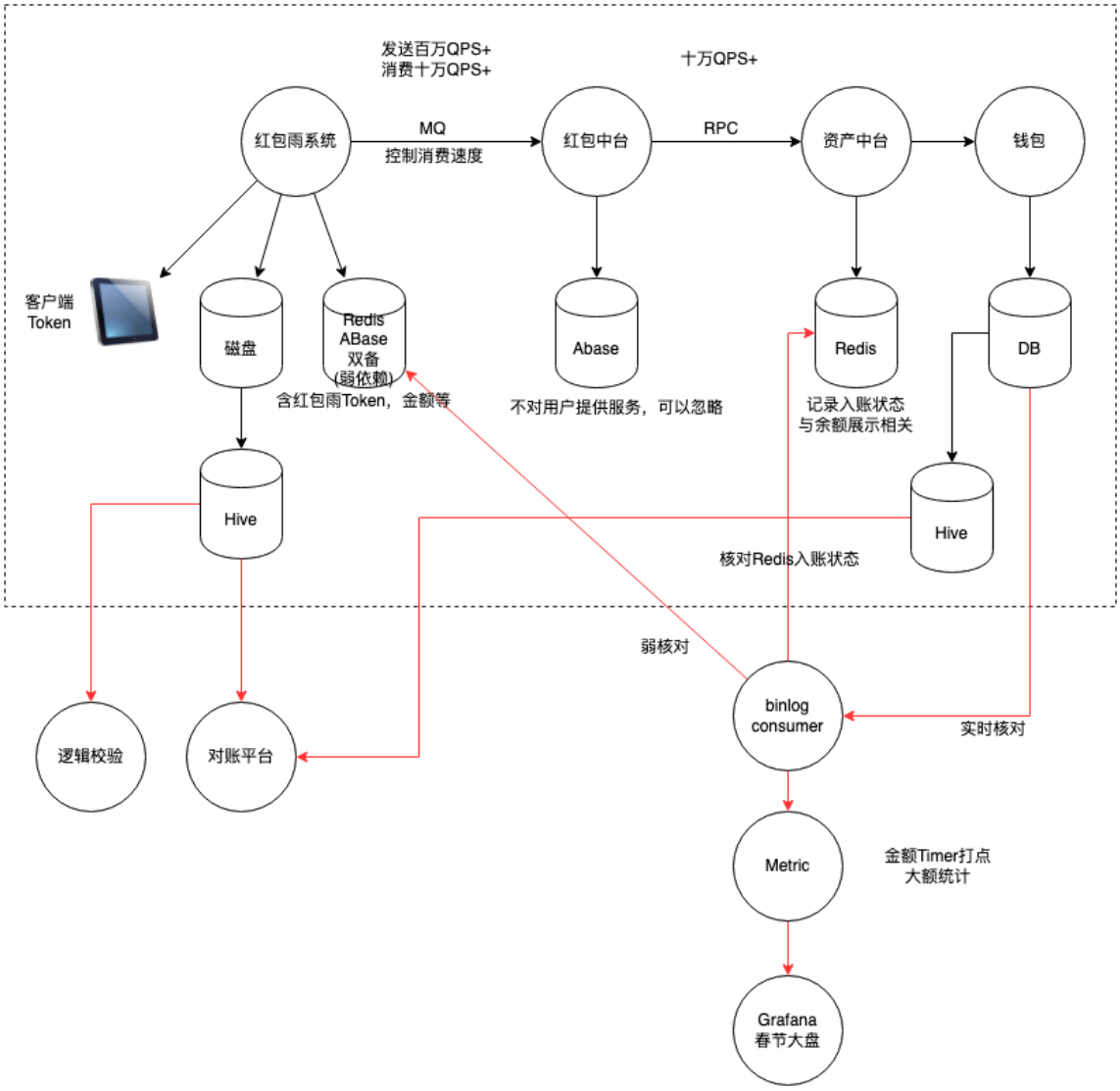
钱包方向在本次春节活动期间做了三件事情来保障大流量大预算的现金红包发放的资金安全：

1. 现金红包发放整体预算控制的拦截
2. 单笔现金红包发放金额上限的拦截
3. 大流量发红包场景的资金对账

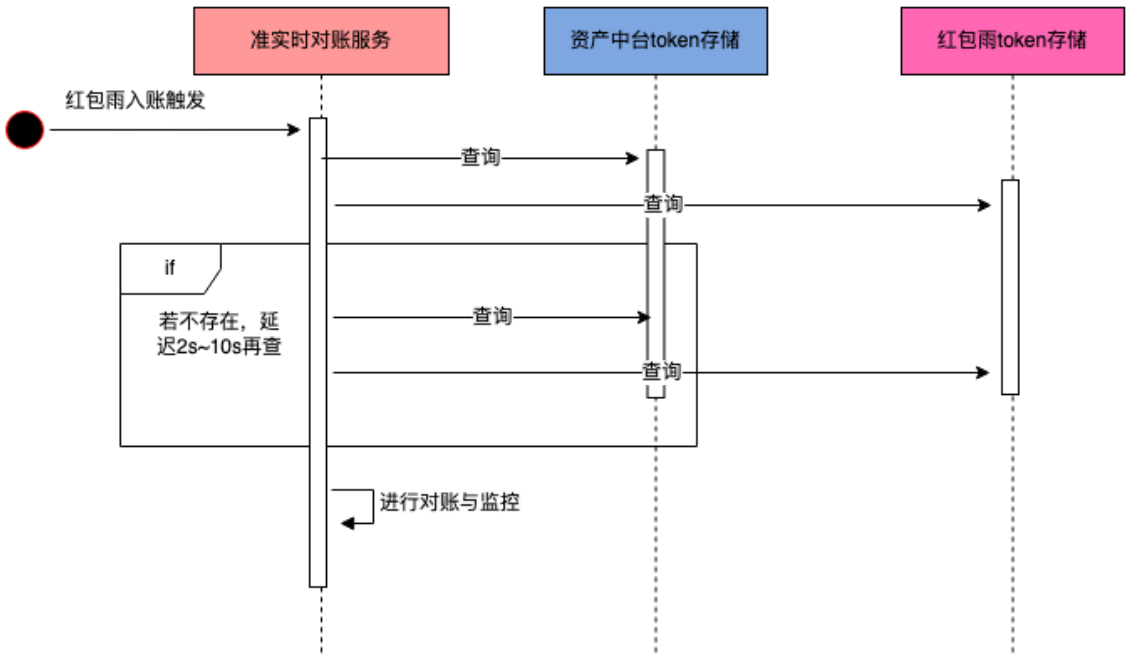
- 小时级别对账：支持红包雨/集卡/烟火红包发放 h+1 小时级对账，并针对部分场景设置兜底 h+2 核对。

- 准实时对账：红包雨已入账的红包数据反查钱包资产中台和活动侧做准实时对账

多维度核对示意图：



准实时对账流程图：



说明：

准实时对账监控和报警可以及时发现是否异常入账情况，如果报警发现会有紧急预案处理。

5. 通用模式抽象

在经历过春节超大流量活动后的设计与实现后，有一些总结和经验与大家一起分享一下。

5.1 容灾降级层面

大流量场景，为了保证活动最终上线效果，容灾是一定要做好的。参考业界通用实现方案，如降级、限流、熔断、资源隔离，根据预估活动参与人数和效果进行使用存储预估等。

5.1.1 限流层面

（1）限流方面应用了 api 层 nginx 入流量限流，分布式入流量限流，分布式出流量限流。这几个限流器都是字节跳动公司层面公共的中间件，经过大流量的验证。

（2）首先进行了实际单实例压测，根据单实例扛住的流量与本次春节活动预估流量打到该服务的流量进行扩容，并结合下游能抗住的情况，在 tlb 入流量、入流量限流以及出流量限流分别做好了详细完整的配置并同。

限流目标：

保证自身服务稳定性，防止外部预期外流量把本身服务打垮，防止造成雪崩效应，保证核心业务和用户核心体验。

简单集群限流是实例维度的限流，每个实例限流的 QPS=总配置限流 QPS/实例数，对于多机器低 QPS 可能会有不准的情况，要**经过实际压测并且及时调整配置值。**

对于分布式入流量和出流量限流，两种使用方式如下，每种方式都支持高低 QPS，区别只是 SDK 使用方式和功能不同

。一般低 QPS 精度要求高，采用 redis 计数方式，使用方提供自己的 redis 集群。高 QPS 精度要求低，退化为总 QPS/tce 实例数的单实例限流。

5.1.2 降级层面

对于高流量场景，每个核心功能都要有对应的**降级方案**来保证突发情况核心链路的稳定性。

（1）本次春节奖励入账与活动活动钱包页方向做好了充分的操作预案，一共有 26 个降级开关，关键时刻弃车保帅，防止有单点问题影响核心链路。

（2）以发现金红包链路举例，钱包方向最后完全降级的方案是只依赖 docker 和 MySQL，其他依赖都是可以降级掉的，MySQL 主有问题可以紧急联系切主，虽说最后一个都没用上，但是前提要设计好保证活动的万无一失。

5.1.3 资源隔离层面

（1）提升开发效率**不重复造轮子。**因为钱包资产中台也日常支持抖音资产发放的需求，本次春节活动也复用了现有的接口和代码流程支持发奖。

（2）同时针对本次春节活动，服务层面做了**集群隔离**，创建专用活动集群，底层存储资源隔离，活动流量和常规流量互不影响。

5.1.4 存储预估

（1）不但要考虑和验证了 Redis 或者 MySQL 存储能抗住对应的流量，同时也要按照实际的获取参与和发放数据等预估存储资源是否足够。

（2）对于字节跳动公司的 Redis 组件来讲，可以进行**垂直扩容**（每个实例增加存储，最大 10G），也可以进行**水平扩容**（单机房上限是 500 个实例），因为 Redis 是三机房同步的，所以计算存储时只考虑一个机房的存储上限即可。要留足 buffer，因为水平扩容是很慢的一个过程，突发情况遇到存储资源不足只能通过配置开关提前下掉依赖存储，需要提前设计好。

5.1.5 压测层面

本次春节活动，钱包奖励入账和活动钱包页做了充分的全链路压测验证，下面是一些经验总结。

1. 在压测前要建立好压测整条链路的监控大盘，在压测过程当中及时和方便的发现问题。
2. 对于 MySQL 数据库，在红包雨等大流量正式活动开始前，进行小流量压测预热数据库，峰值流量前提前建链，减少正式活动时的大量建链耗时，保证发红包链路数据库层面的稳定性。
3. 压测过程当中一定要传压测标，支持全链路识别压测流量做特殊逻辑处理，与线上正常业务互不干扰。
4. 针对压测流量不做特殊处理，压测流量处理流程保持和线上流量一致。
5. 压测中要验证计算资源与存储资源是否能抗住预估流量

- **梳理好压测计划**，基于历史经验，设置合理初始流量，渐进提升压测流量，实时观察各项压测指标。
- **存储资源压测数据要与线上数据隔离**，对于 MySQL 和 Bytekv 这种来讲是建压测表，对于 Redis 和 Abase 这种来讲是压测 key 在线上 key 基础加一下压测前缀标识 。
- **压测数据要及时清理**，Redis 和 Abase 这种加短时间的过期时间，过期机制处理比较方便，如果忘记设置过期时间，可以根据写脚本识别压测标前缀去删除。

1. 压测后也要关注存储资源各项指标是否符合预期。

5.2 微服务思考

在日常技术设计中，大家都会遵守微服务设计原则和规范，根据系统职责和核心数据模型拆分不同模块，提升开发迭代效率并不互相影响。

但是微服务也有它的弊端，对于超大流量的场景功能也比较复杂，会经过多个链路，这样是极其消耗计算资源的。

本次春节活动**资产中台提供了 sdk 包代替 rpc 进行微服务链路聚合对外提供基础能力**，如查询余额、判断用户是否获取过奖励，强制入账等功能。访问流量最高上千万，与使用微服务架构对比节约了上万核 CPU 的计算资源。

6. 系统的未来演进方向

（1）梳理上下游需求和痛点，优化资产中台设计实现，完善基础能力，优化服务架构，提供一站式服务，让接入活动方可以更专注进行活动业务逻辑的研发工作。

（2）加强实时和离线数据看板能力建设，让奖励发放数据展示的更清晰更准确。

（3）加强配置化和文档建设，对内减少对接活动的对接成本，对外提升活动业务方接入效率。