

曲翔宇工作汇报

目录

- 产品功能实现
 - 元数据新型组合形态
 - 轻颜相机专题体系
 - 剪映海外版Hashtag分类
 - 其他关联需求
 - 商业化变现能力
 - Capcut会员订阅
 - 影像生态基础建设
 - 社区生态
 - 创作者体系
 - 运营能力
- 技术建设和服务架构
 - 海外API读接口通用容灾方案
 - V1.0整体架构
 - V2.0整体架构
 - 其他技术建设
- 团队建设和分享
- 长尾问题修复

产品功能实现

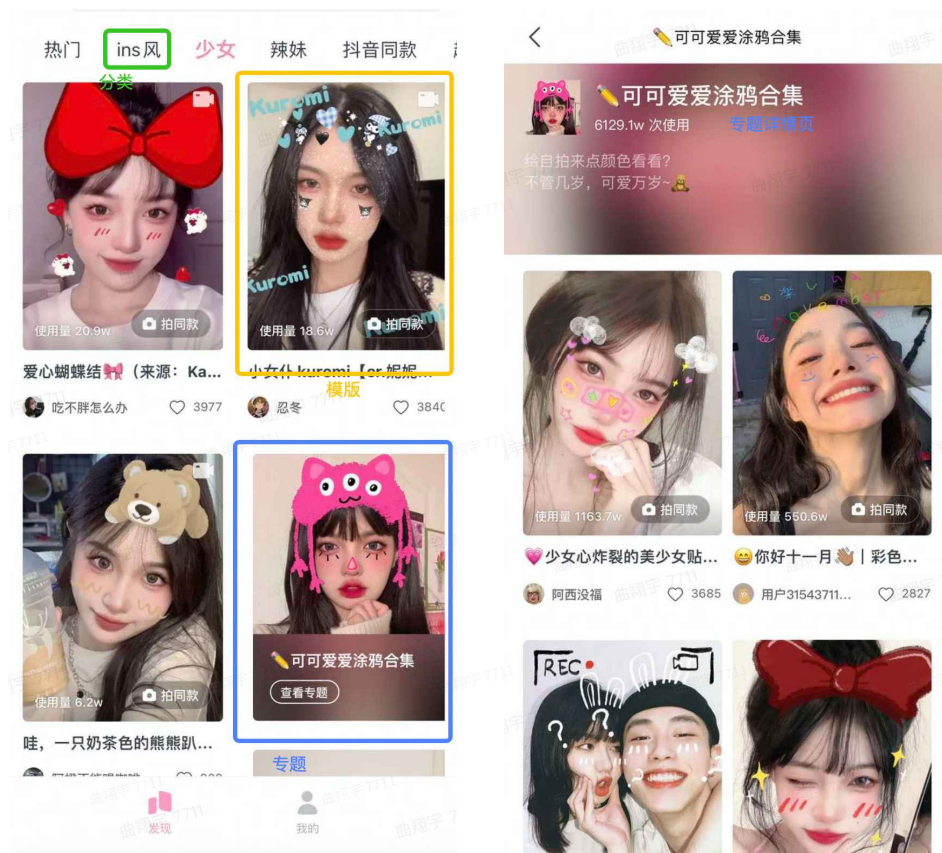
元数据新型组合形态



对影像业务而言,「模版」是最基本的元数据,绝大部分的产品能力都是基于模版进行衍生与变换,通过对元数据进行组合能够形成新的数据形态,基于新的数据形态以繁衍出更多的用户消费场景,丰富以模版为核心的整个影像生态.

轻颜相机专题体系

👁️ 专题是指许多相似模版组成的合集(国外通常叫做hashtag, 一般以符号#开头)



技术文档: [目 轻颜专题 Wiki](#)

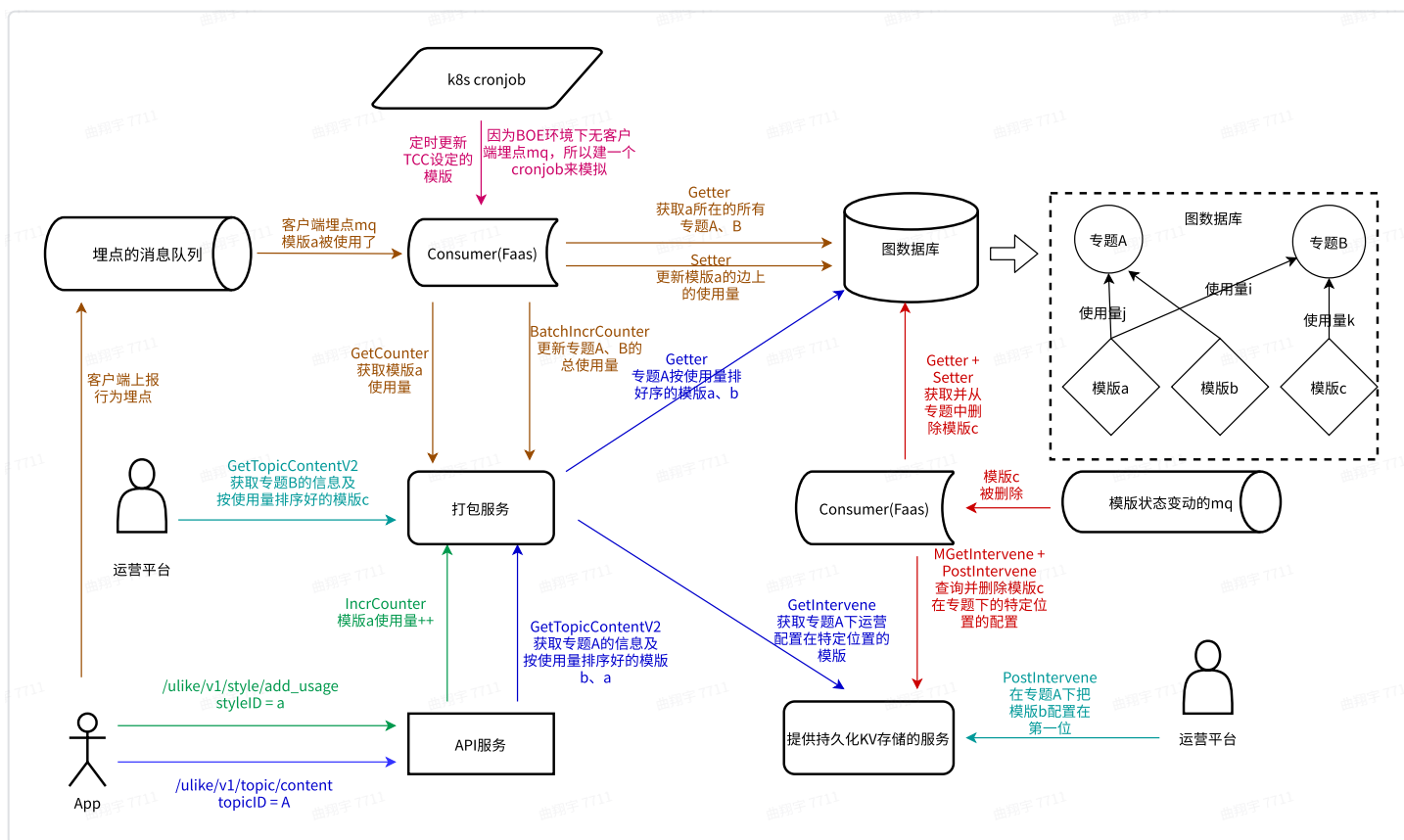
架构设计:

褐色和红色分别是基于Kafka的专题模版数据(使用量,etc)和状态的离线更新的数据流

蓝色是专题及专题下模版在线下发的数据流

粉色是基于k8s cronjob的用于测试的数据流

青蓝色是运营人工配置操作



技术思考:

1. 区别于国内剪映, 这里选型图数据库作为底层存储, 因为需要正查和反查的双向操作; 国内剪映采用 item.list 链的形式存储 专题->list<模版>, 在模版的item结构中新增专题字段存储 模版->list<专题>, 使的同一个关系存储在两个地方, 容易出现数据不一致的情况, 而使用图数据库解决了这一问题.
2. 关于异步更新和同步更新: 绝大部分数据都可以异步更新, 采用消息队列的方式异步更新能够减少资源占用、逻辑解耦、方便限流等

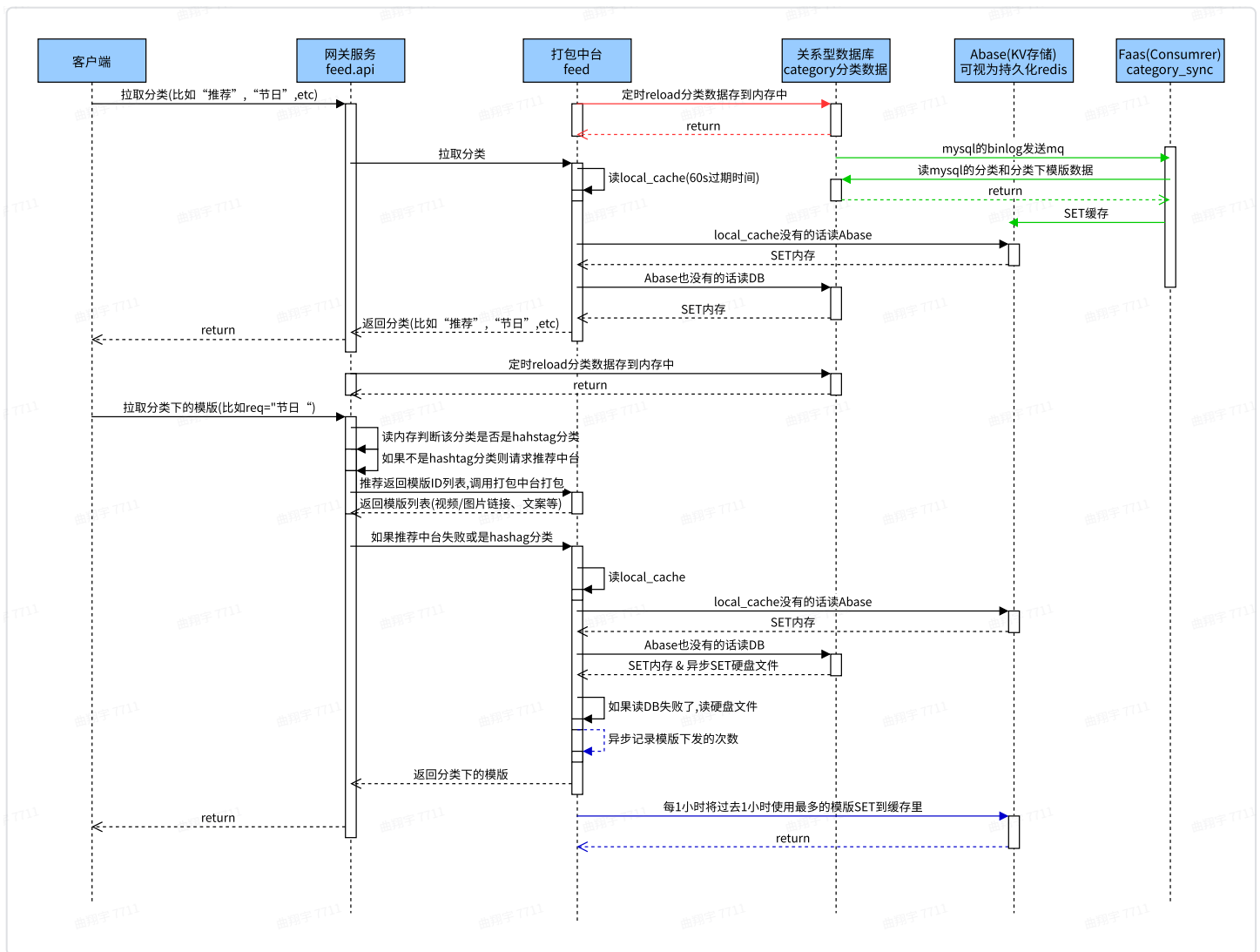
剪映海外版Hashtag分类

👁️ 用户投稿模版时会带上#标签(海外称为hashtag), 比如#happy, 带有相同标签的模版属于同一hashtag; 本次需求将实现由运营配置一系列hashtag到同一个类目中, 服务端将多个hashtag下的模版取出并按照使用量排序下发到这个分类中

技术文档: [Capcut新增hashtag分类服务端技术方案](#)

整体流程:

红色绿色蓝色表示异步更新缓存的流程



技术思考:

一套完整的业务缓存+容灾设计(不含故障转移): 内存缓存 -> redis集群 -> RDS数据库 -> 磁盘文件 -> API缓存

其中RDS数据库左边是指缓存, 主要目的是提升访问性能; RDS数据库右边是指容灾, 主要目的兜底返回保证用户体验

算法在实际工程中的使用:

? 在这个需求中有一个分类比如“节日”, 运营在这个分类下配置了“国庆节”、“情人节”、“圣诞节”、“A” ... “G” 这个10个hashtag, 每个hashtag下又有很多模版, 当客户端用户点开这个分类时, 服务端需要将这10个hashtag下模版做综合排序按使用量从高到低返回20个模版

使用最小堆 container/heap 结构解决TopK问题:

```
1 type itemWithUsage struct {
```

```

2   ID          int64 // 模版ID
3   UsageCount int64 // 模版使用量
4 }
5
6 type minHeap []*itemWithUsage
7 func (h minHeap) Len() int {
8     return len(h)
9 }
10 func (h minHeap) Less(i, j int) bool {
11     return h[i] != nil && h[j] != nil && h[i].UsageCount < h[j].UsageCount
12 }
13 func (h minHeap) Swap(i, j int) {
14     h[i], h[j] = h[j], h[i]
15 }
16 func (h *minHeap) Push(x interface{}) {
17     *h = append(*h, x.(*itemWithUsage))
18 }
19 func (h *minHeap) Pop() interface{} {
20     old := *h
21     n := len(old)
22     x := old[n-1]
23     *h = old[0 : n-1]
24     return x
25 }

```


然后起n个协程并发请求每个hashtag下的模版,并将结果通过channel发送到处理最小堆排序的协程

其他关联需求

1. 国轻颜相机专题使用IMC绑定搜索词技术方案
2. 国专题支持用户投稿模版服务端技术方案

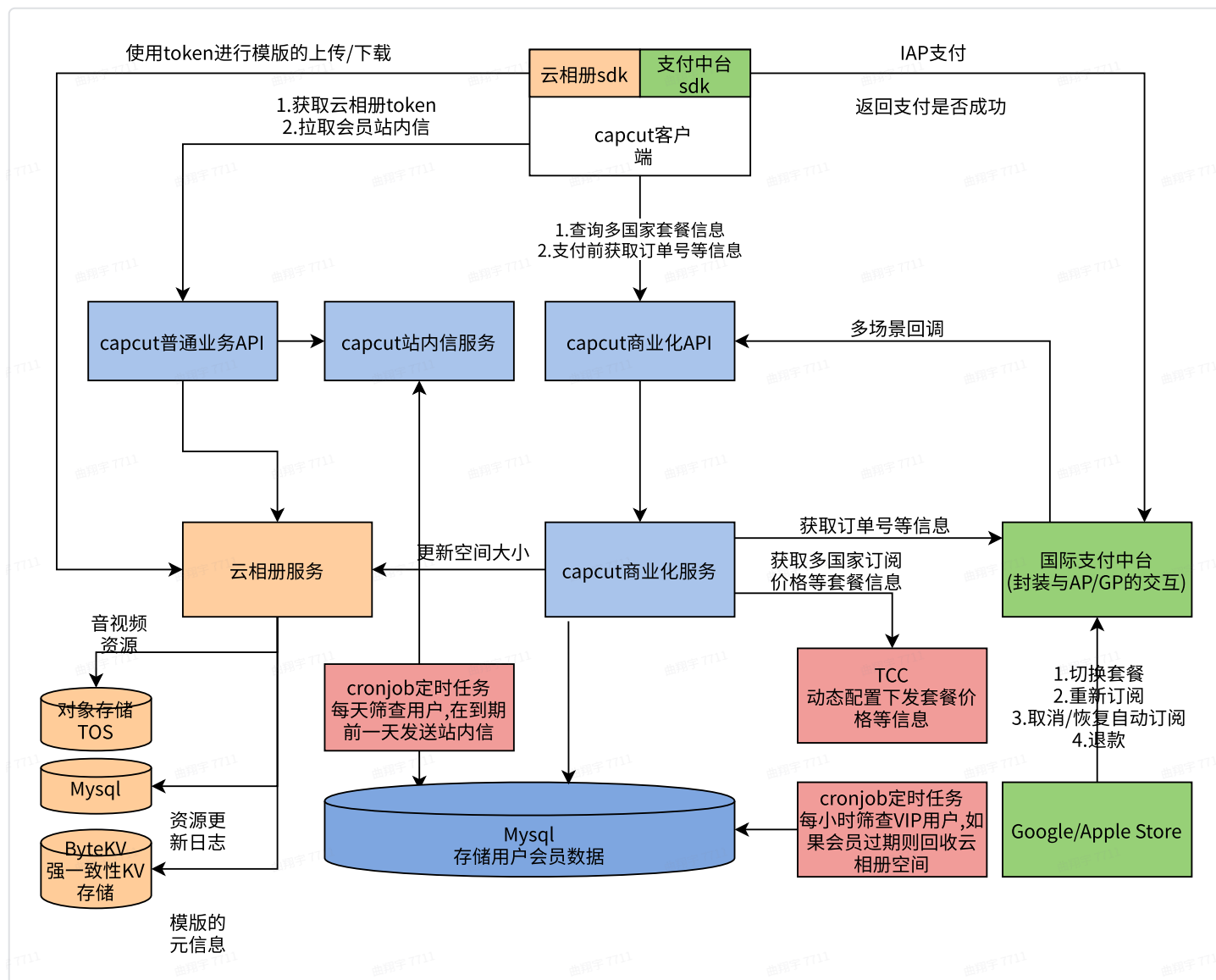
商业化变现能力

Capcut会员订阅

 会员订阅和虚拟商品付费是工具类APP两大技术变现手段之一, capcut通过给予VIP用户更多权益(诸如优质模版解锁、云空间升级等)使得在满足普通用户基本功能的前提下, 促使更多用

技术文档: [CC会员订阅](#) | [服务端技术方案&接口文档](#)

整体架构:



技术思考:

1. 东南亚五国和巴西都需要给会员发送即将到期的站内信, 但数据库存储的截止时间是时间戳, 所以需要根据时区转换成当地时间:

```

1 func GetLocalTimeStrByUnixTimeStamps(ctx context.Context, timeStamps int64, re
2     conf, _ := client_tcc.TCCGetRegionZoneConf(ctx)
3     // 比如 conf = {"ID", -1}
4     offset, _ := conf.RegionZone[region]
5     var zoneName string
6     if offset > 0 {
7         zoneName = fmt.Sprintf("UTC+%d", offset)

```

```

8      } else {
9          zoneName = fmt.Sprintf("UTC-%d", -offset)
10     }
11
12     localTime := time.Unix(timeStamps, 0) // 返回UTC时间
13     zoneLocal := time.FixedZone(zoneName, offset*60*60) // 返回所在时区
14     timeInZone := localTime.In(zoneLocal) // 转换成所在时区时间
15     return timeInZone.Format("2006-01-02 15:04:05"), nil
16 }

```

2. 商业化DB保证缓存一致性的策略:

- a. 读场景先读cache,如果cache miss,则读db并写cache
- b. 写场景直接写db,通过mysql的binlog触发consumer删除缓存(使用的是rocketmq保证低延迟)

影像生态基础建设



影像的各类产品(APP)基本都可分为工具+社区两大部分,工具满足用户基本的修图、剪视频等诉求,而社区为用户提供分享和交流的空间,二者共同构成影像生态,其中服务端更加focus社区部分,通过引入推荐、搜索、评论、音乐等功能丰富整个社区生态

社区生态

1. 站内信: [📄 Capcut美国&巴西上线官方站内信功能技术文档](#)
2. 音乐: [📄 剪映海外音乐搜索技术文档](#)
3. 推荐: [📄 醒图工具页模版接入推荐技术文档](#)
4. 评论: [📄 轻颜相机上线评论功能技术文档](#)
5. Sug: [📄 轻颜相机上线sug联想词技术文档](#)
6. 搜索: [📄 【CC】剪同款搜索结果筛选功能](#)
7. 搜推: [📄 轻颜相机主拍页模版接入推荐&搜索技术文档](#)
8. 搜索: [📄 CC搜索使用领航者入库分享和点赞数据](#)

创作者体系

1. [📄 Capcut创作者批量流量激励服务端实现方案](#)

运营能力

1. [🇺🇸 搜索Debug平台上线Hashtag搜索技术文档](#)

技术建设和服务架构

海外API读接口通用容灾方案

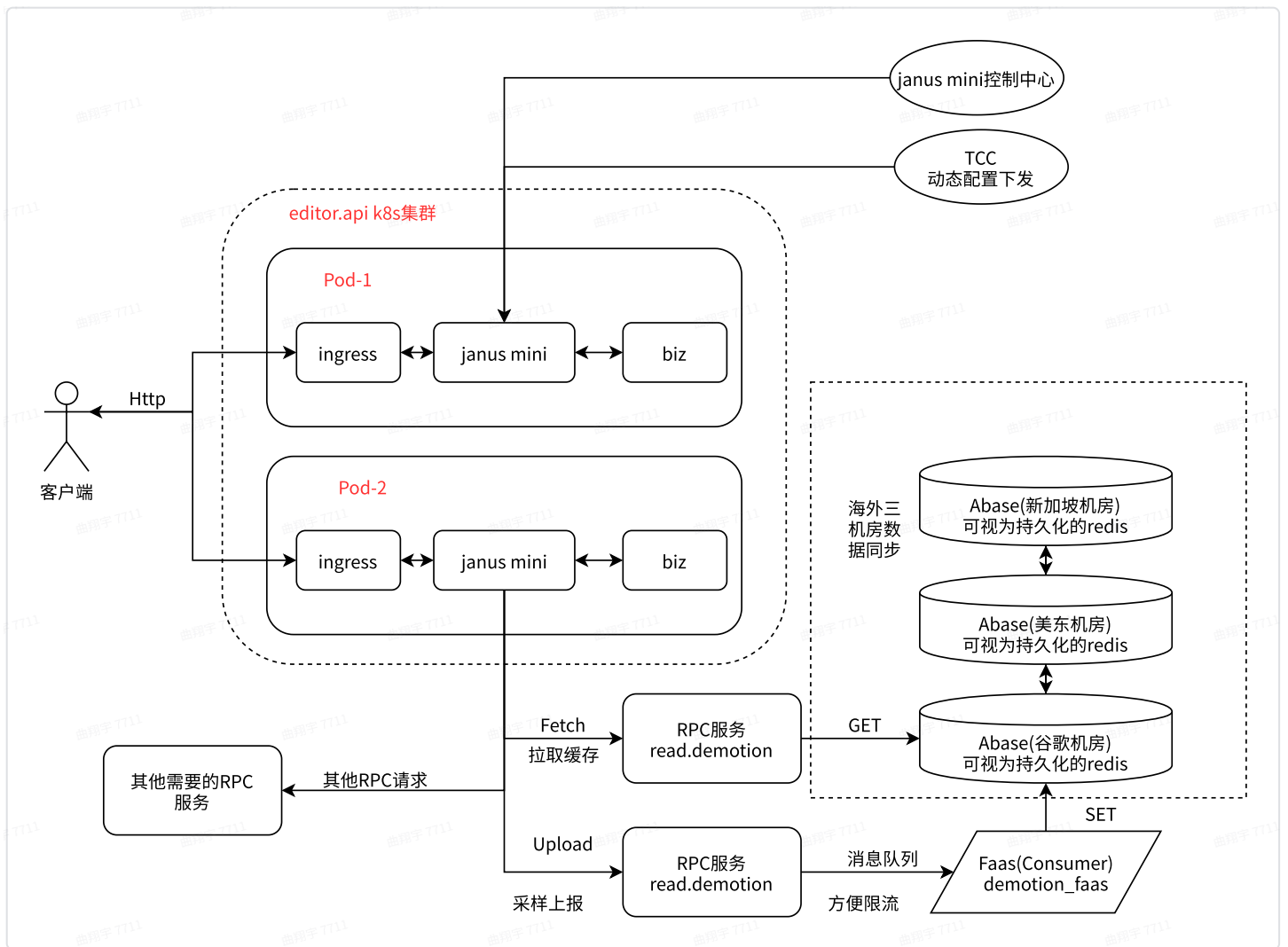
✓ **容灾能力**即在服务出问题时能够保证核心功能可用，或尽可能降低影响范围保证用户体验，一般基于**数据复制**和**故障转移**机制实现，故障转移通常使用切流的手段，该方案属于数据复制，是基于Janus Mini(一种sidecar网关)的缓存读写方案，采用可配置、可扩展的思想，方便新API接口无代码接入

技术文档: [🇺🇸 Capcut API读接口通用容灾方案](#)

接入手册: [🇺🇸 剪映海外API读接口通用容灾接入手册](#)

已接入的接口: [🇺🇸 剪映海外API读接口通用容灾资源汇总](#)

V1.0整体架构



V2.0整体架构

通用容灾V1.0的问题:

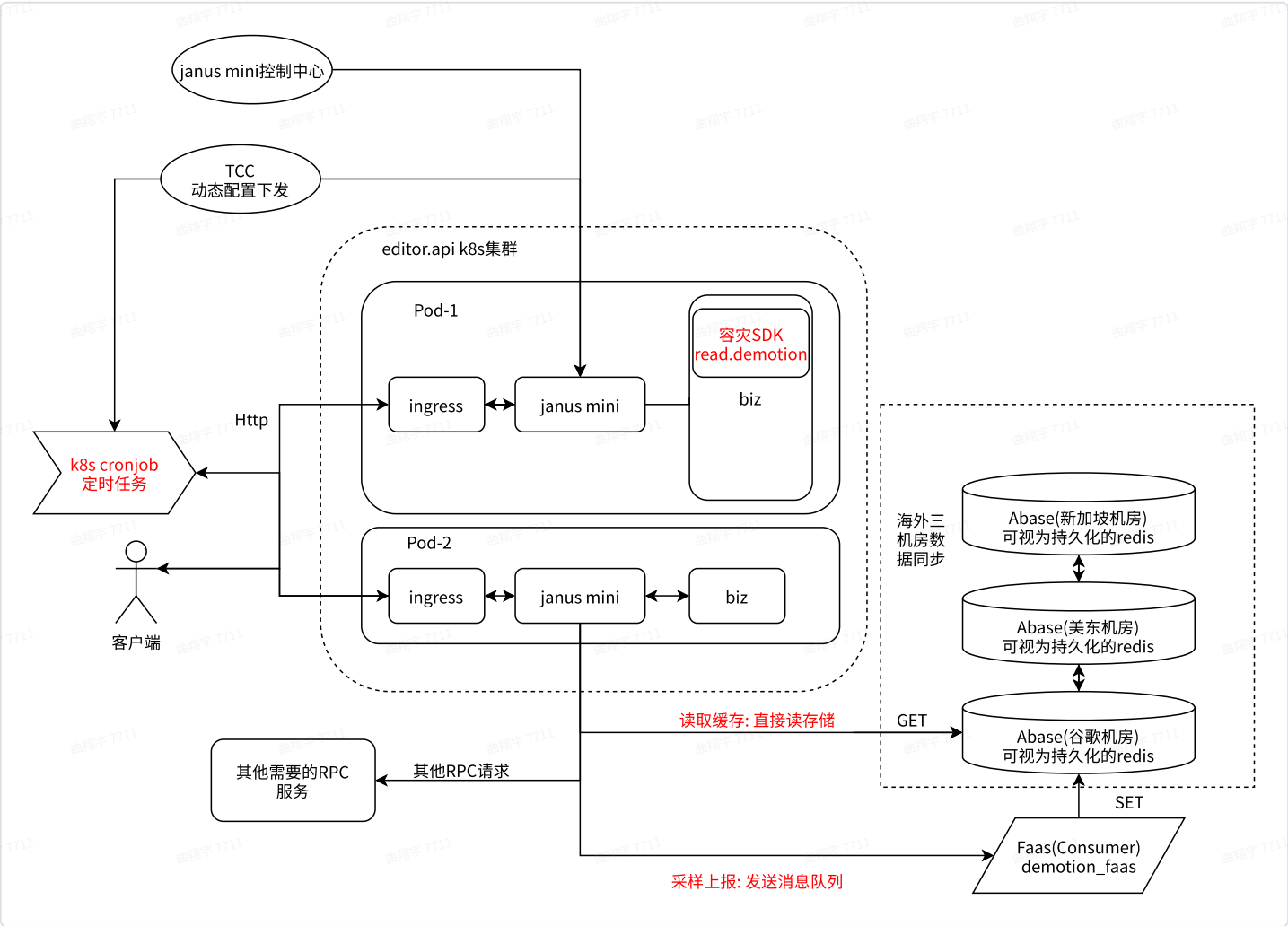
1. 独立的容灾RPC服务read.demotion无法撑住极端情况下的所有的读缓存流量, 因为海外缺乏机器资源(目前大流量场景的上报也采用了随机采样的方式).
2. “请求成功上报缓存”的上报模式更适用于“不可枚举”的情况, 比如搜索场景用户的query词是无穷无尽的, 而不太适用于“可枚举且访问较少”的场景, 比如音乐歌单场景只有几十个固定歌单, 而有些歌单可能比较久才会请求到, 此时有一个固定的兜底池更合适

通用容灾V2.0的优点:

1. 采用SDK的接入方式来取代原有的独立RPC服务的形式, 容灾逻辑依附于原有服务, 所以不存在资源不足而无法撑住极端情况的读缓存流量的问题.
2. 新增cronjob定时请求TCC配置的URL, 在不改变容灾服务原有设计的基础上, 能够确保“可枚举且访问较少”这种场景的缓存可用性

整体架构2.0

红色表示相较于架构1.0不同的地方



其他技术建设

- 1. [🇨🇳 使用WAF预警恶意流量攻击](#)
- 2. [🇨🇳 SG资源优化方案](#)
- 3. [🇨🇳 优化轻颜相机消费客户端埋点的Faas调中台RPC服务QPS过高问题](#)

团队建设和分享

- 1. [🇨🇳 TroubleShooting平台调研报告](#)

- 2. 国 Native Faas调研报告
- 3. 国 影像刷数据技术方案模版
- 4. 国 接口测试平台高峰期禁止访问线上接口
- 5. 国 CC搜索中台架构

长尾问题修复

- 1. 国 解决CC搜索入库缺失用户基础信息字段历史问题
- 2. 国 Capcut敏感词未命中历史问题处理
- 3. 国 影像finex框架不支持odin中间件问题处理
- 4. 国 Capcut模版Duration和实际时长不符问题排查
- 5. 国 轻颜重复订阅订单处理方案
- 6. 国 Hashtag分类拉空排查