知乎Feed流架构演进

知乎

主办方 Geekbang. 极客邦科技 InfoQ

AiCon

全球人工智能与机器学习技术大会

助力人工智能落地

2018.1.13 - 1.14　北京国际会议中心

扫描关注大会官网

极客时间

重拾极客精神·提升技术认知

下载极客时间App

获取有声IT新闻、技术产品专栏，每日更新

扫一扫下载极客时间App

姚钢强

2013 年加入知乎，知乎 Feed 流技术负责人，负责期间 Server 端 P95 响应时间从 1.6S 降低到 700Ms，稳定性由 99.9% 提升到 99.995%

知乎

提纲

A. Feed 流的需求和特点
B. 老 Feed 流的构架遇到的问题
C. 新构架 Redis module 技术方案
D. Redis module 方案遇到的问题
E. 新的问题与挑战

知乎

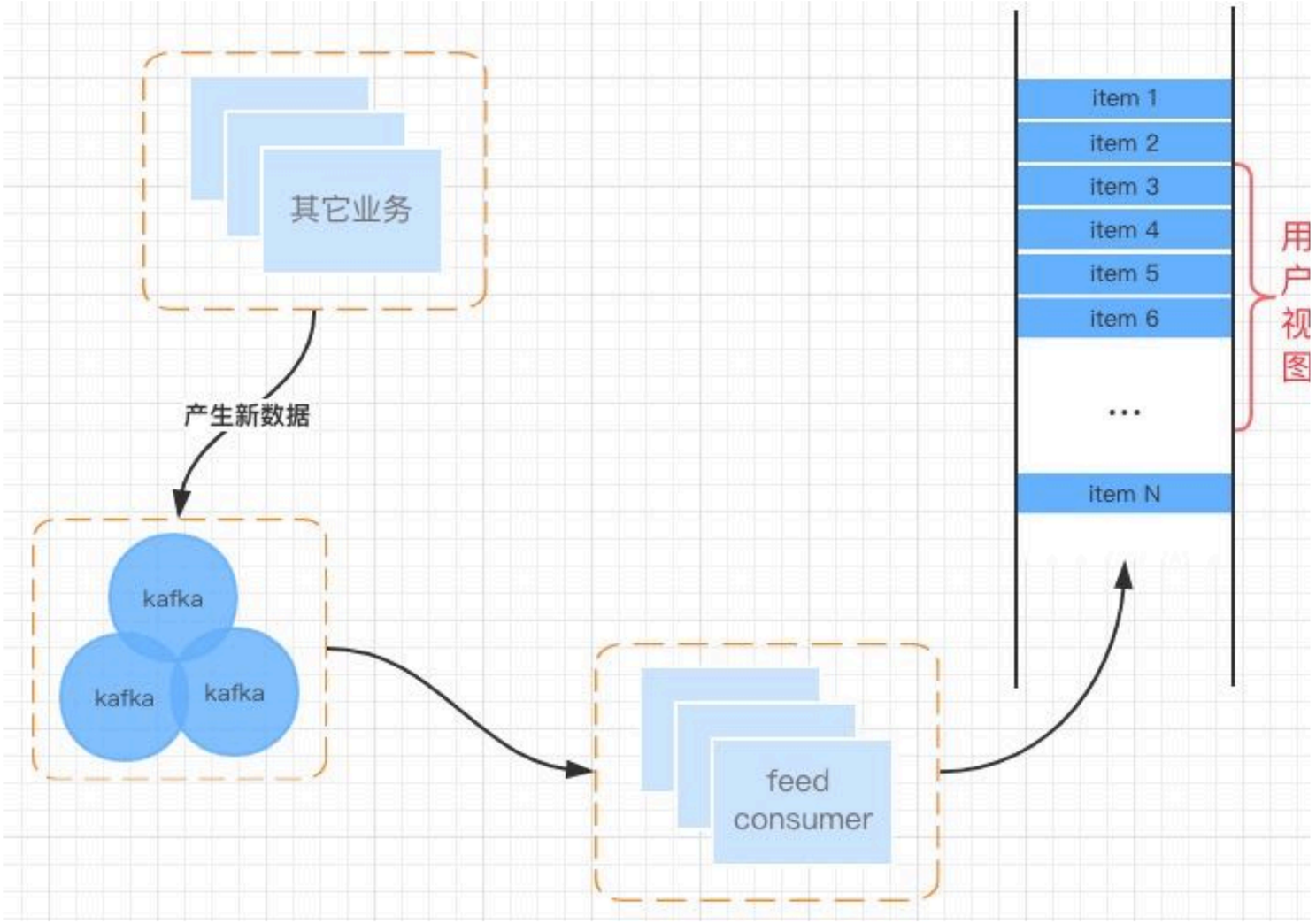
知乎 Feed 流的需求和特点

A. DAU 2600万（2017.9）

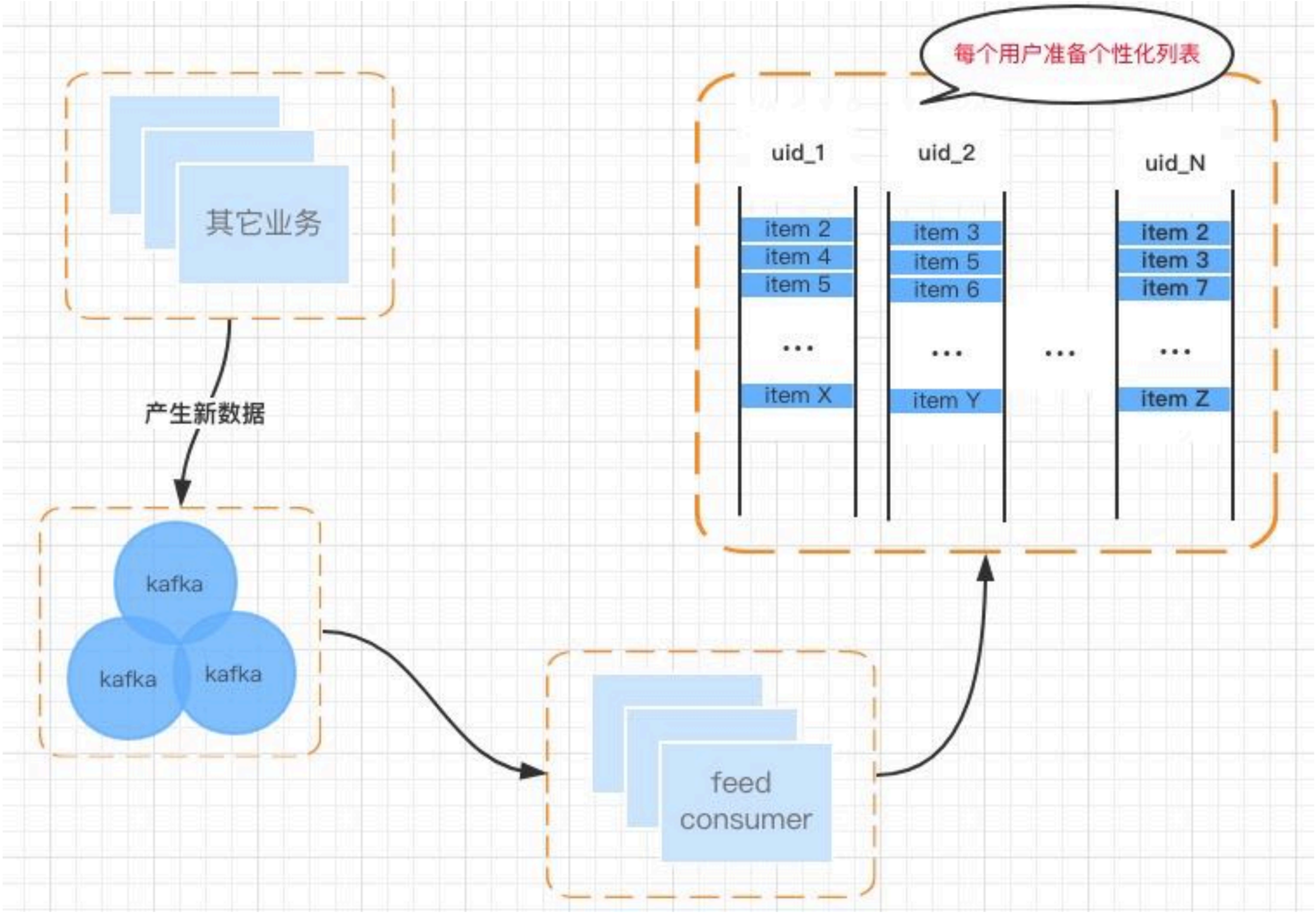
B. 个性化推荐，每次请求返回内容不同（与搜索不同， cache 难做）

C. 用户动态准实时分发

无个性化（热门榜单）


其它业务
产生新数据
kafka
kafka
kafka
feed
consumer
item 1
item 2
item 3
item 4
item 5
item 6
...
item N
用户视图

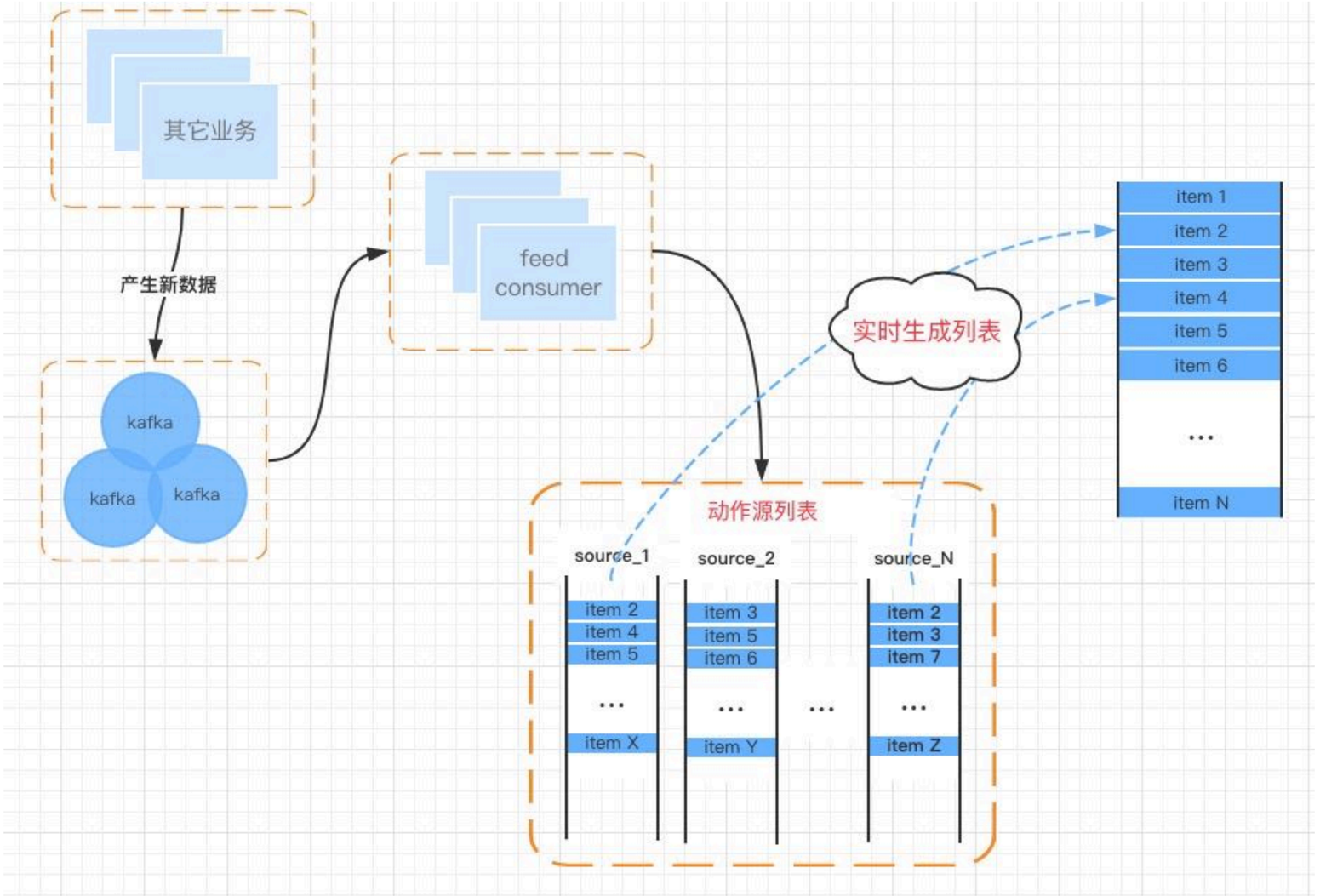
用户个性化 push


每个用户准备个性化列表
uid_1
uid_2
uid_N
item 2
item 4
item 5
item X
item 3
item 5
item 6
item Y
item 2
item 3
item 7
item Z
其它业务
产生新数据
kafka
kafka
kafka
feed
consumer

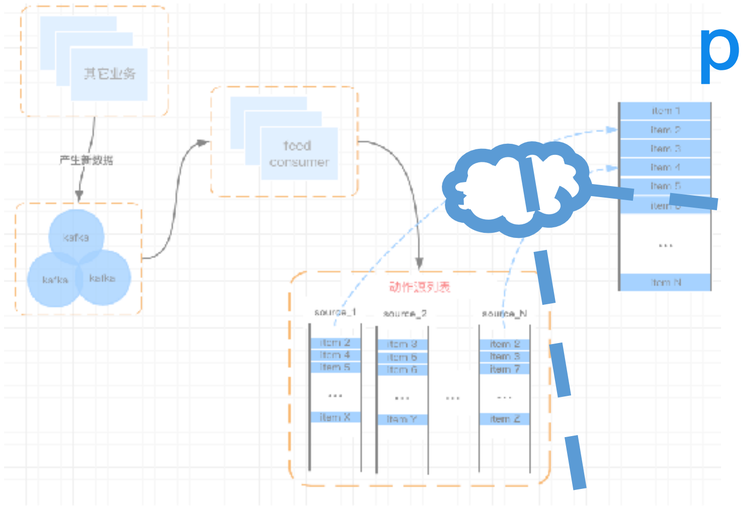
push 存在的问题

A. 资源消耗严重，计算量大，存储量大
B. 智能推荐，排序难以实时调整
C. 过滤比较难做（关注 or 被删除）
D. 动态准实时分发难以达到 （高粉丝用户）

实时 pull


其它业务
产生新数据
kafka
kafka
kafka
feed
consumer
实时生成列表
item 1
item 2
item 3
item 4
item 5
item 6
...
item N
动作源列表
source_1
source_2
source_N
item 2
item 4
item 5
...
item X
item 3
item 5
item 6
...
item Y
item 2
item 3
item 7
...
item Z

pull 计算流程（慢）



client

member
推荐的相关源

从指定的源拉取条目

过滤条目

条目
够用

N

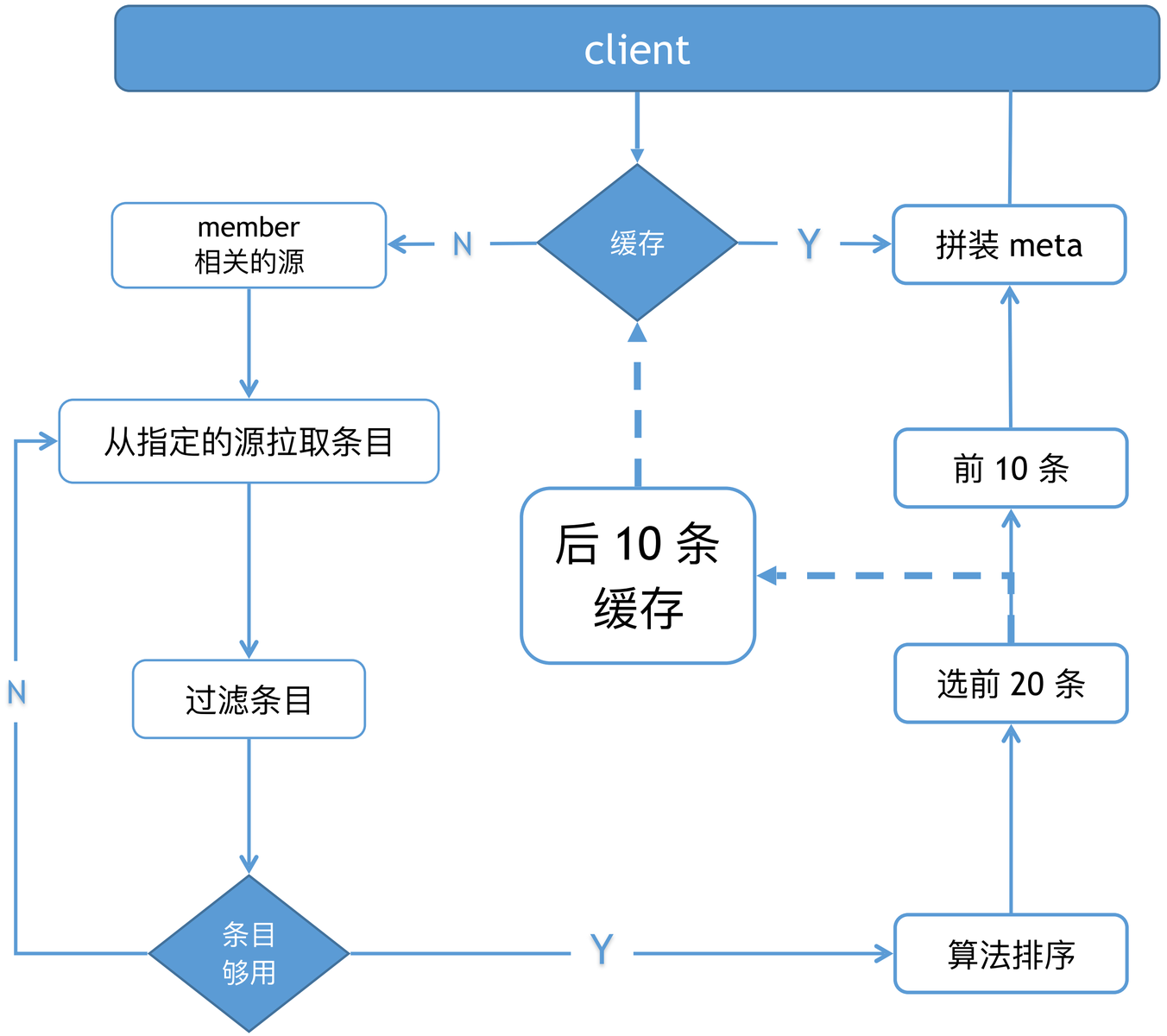
Y

算法排序

选前 10 条

拼装 meta

提前计算，做缓存


client
缓存
N
Y
member
相关的源
拼装 meta
从指定的源拉取条目
前 10 条
后 10 条
缓存
过滤条目
选前 20 条
N
条目
够用
Y
算法排序

提前计算的问题

- 存在冗余计算，占用资源多
- 冷启动 P95 响应时间长
- 用户行为分发延迟，体验差
- 离线计算策略复杂，难以维护
- 推荐算法难以实时调整

如何优化，难点儿在哪儿？

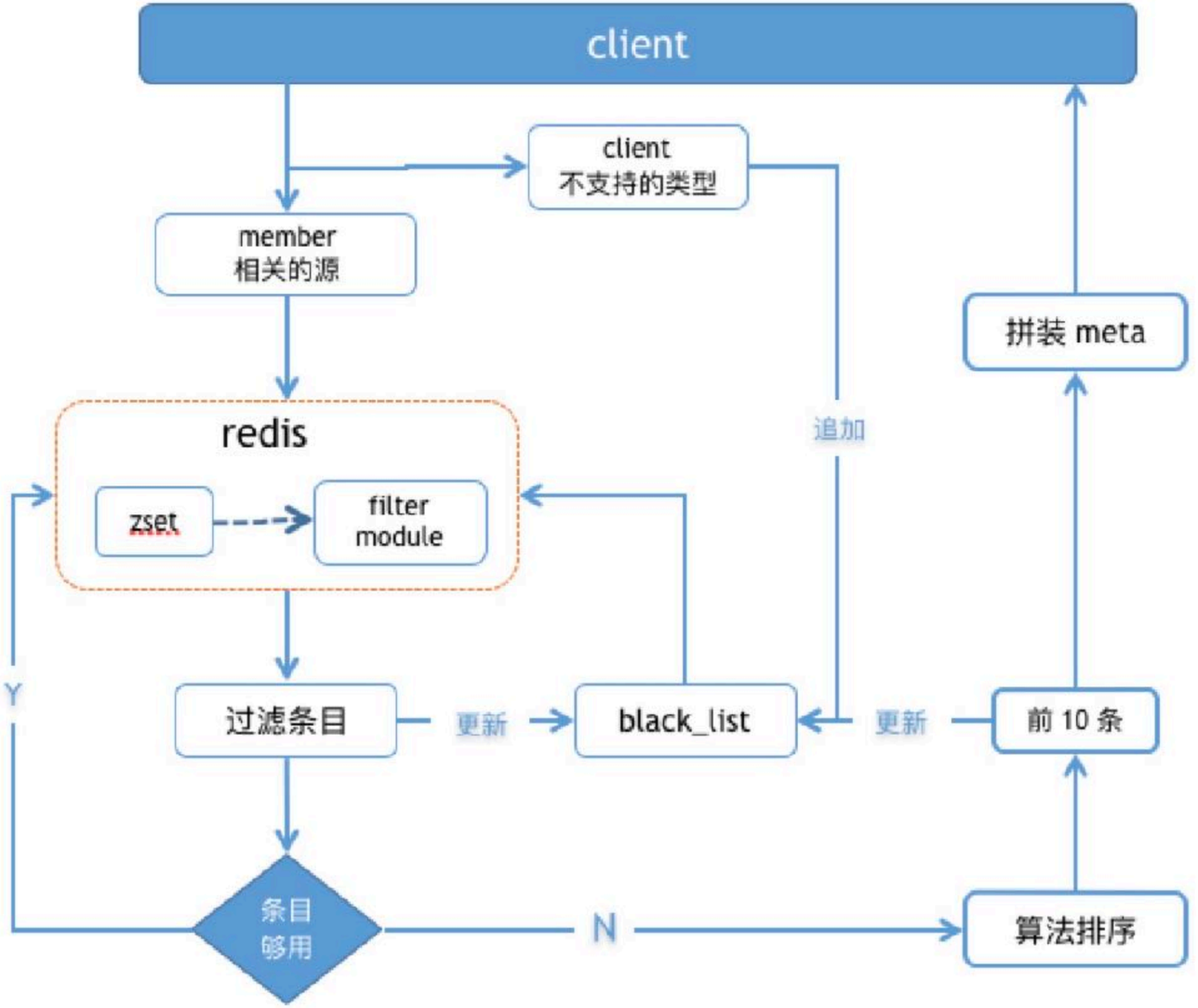
- 依赖服务响应慢
  - redis cache + local cache
  - gevent 并发
  - 超时做降级
- Python 计算太慢
  - Cython 模块替换
- 由于条目不够，反复访问底层源的存储 feesource

可能的解决方案

- 拉取出更多的条目，防止被过滤掉?
  - 拉取更多的条目也会浪费时间，过滤压力大
- 根据算法拉取出更精准的条目?
  - 算法期望召回池越大越好

计算下推，接近存储

新 feed 计算逻辑


client
client
不支持的类型
member
相关的源
redis
zset
filter
module
过滤条目
更新
black_list
更新
前 10 条
拼装 meta
追加
Y
条目
够用
N
算法排序

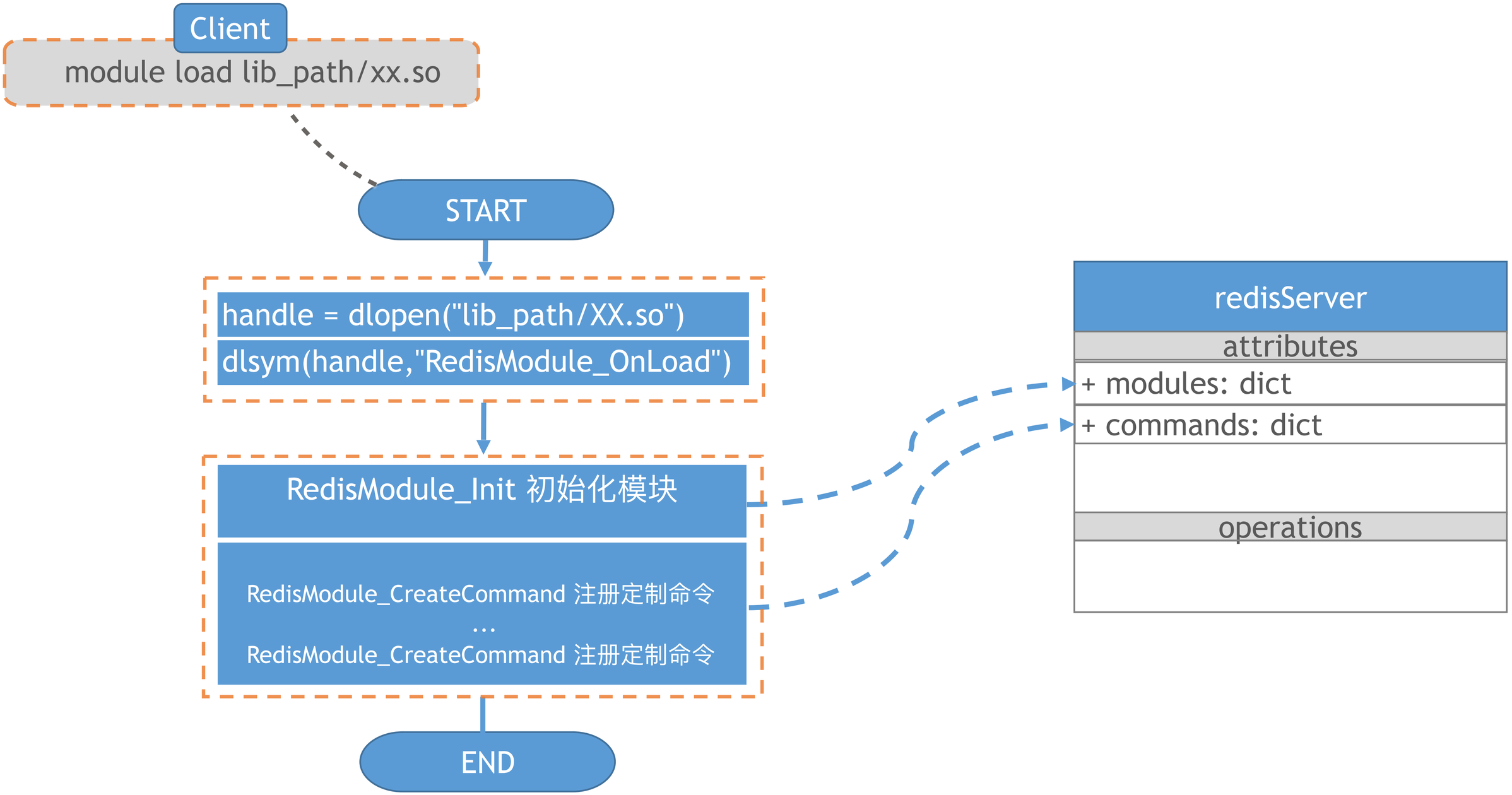
Redis Module

Redis modules make possible to extend Redis functionality using external modules, implementing new Redis commands at a speed and with features similar to what can be done inside the core itself.

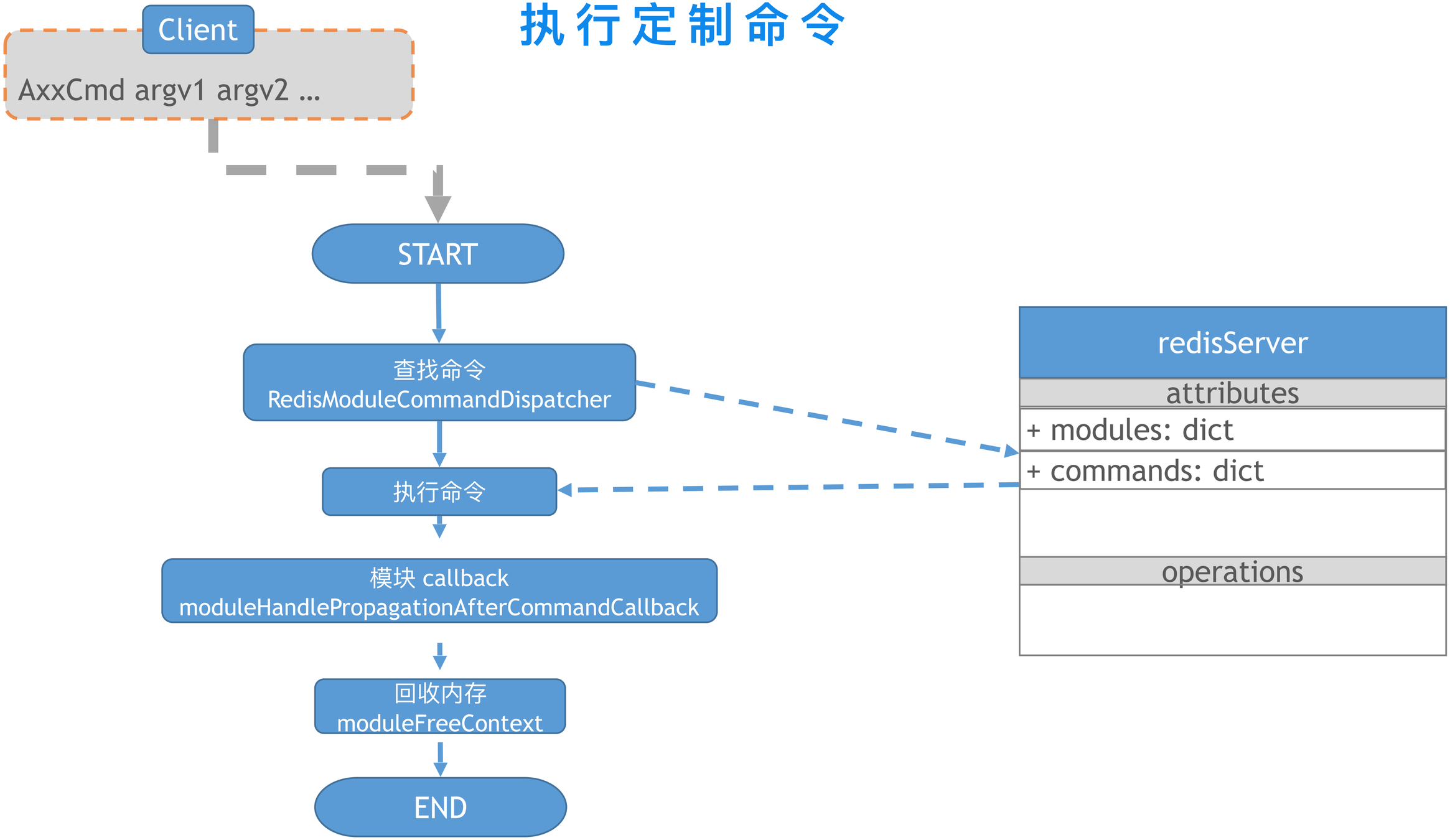
Redis Module

- 加载定制命令（MODULE LOAD module load lib_path/xx.so）
- 执行定制命令
- 卸载定制命令（MODULE UNLOAD mymodule）

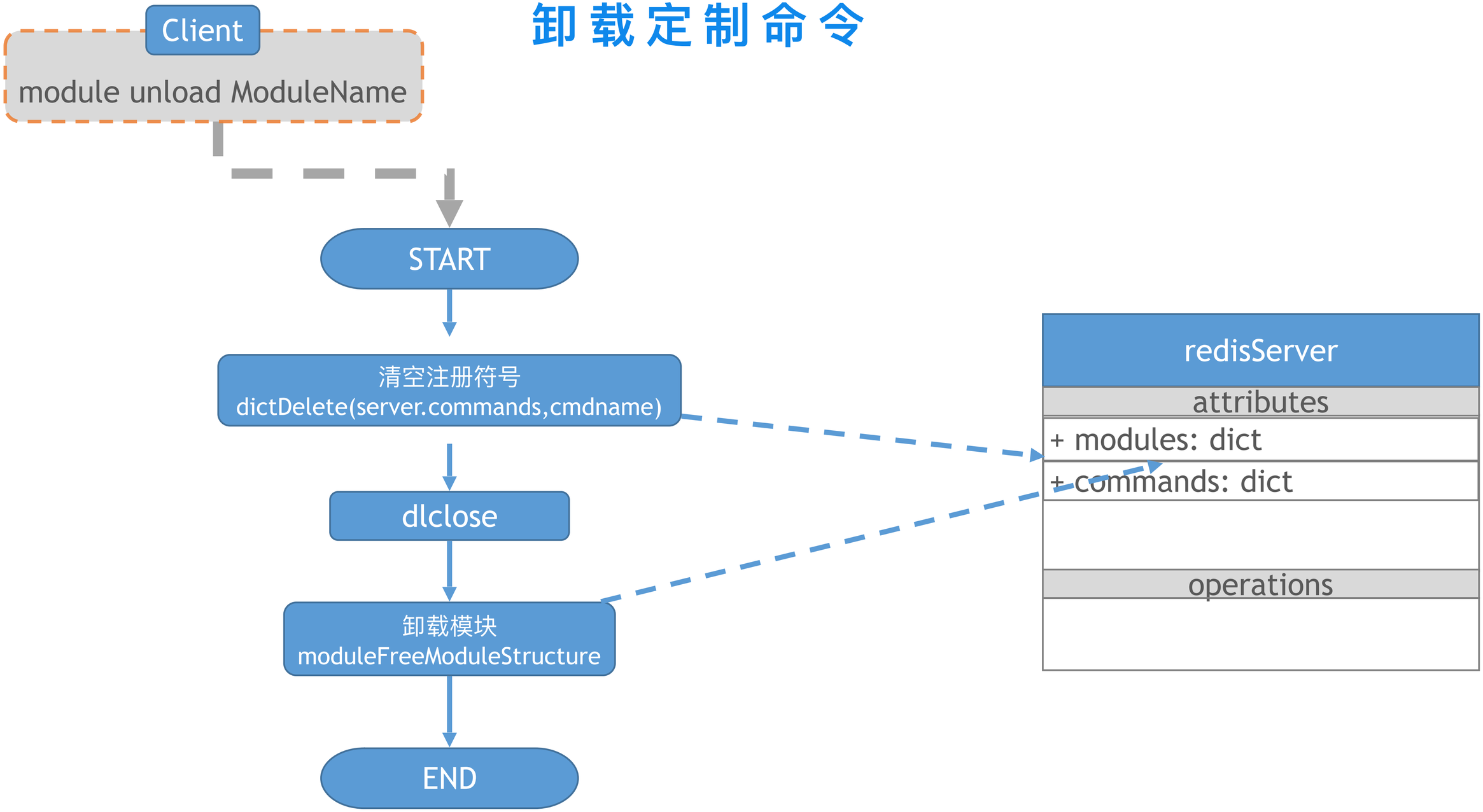
加载定制命令


Client
module load lib_path/xx.so
START
handle = dlopen("lib_path/XX.so")
dlsym(handle,"RedisModule_OnLoad")
RedisModule_Init 初始化模块
RedisModule_CreateCommand 注册定制命令
...
RedisModule_CreateCommand 注册定制命令
END
redisServer
attributes
+ modules: dict
+ commands: dict
operations

执行定制命令


Client
AxxCmd argv1 argv2 ...
START
查找命令
RedisModuleCommandDispatcher
执行命令
模块 callback
moduleHandlePropagationAfterCommandCallback
回收内存
moduleFreeContext
END
redisServer
attributes
+ modules: dict
+ commands: dict
operations

卸载定制命令


Client
module unload ModuleName
START
清空注册符号
dictDelete(server.commands,cmdname)
dlclose
卸载模块
moduleFreeModuleStructure
END
redisServer
attributes
+ modules: dict
+ commands: dict
operations

定制的命令

- open_core, close_core
  - Redis module 更新时打开 core dump，如果 crash 方便分析
- ts_query
  - Redis 内部的过滤，归并流程

ts_query 接口设计

- Request
  - [source_type, source_id]: 需要拉取 source
  - black_items: 需要过滤的黑名单条目
  - merge_strategy: 合并策略
  - required_number: 需要返回的条目数量
- Response
  - [item_type, item_id, item_action, item_score]: 返回需要数目的 feed 条目

带来的收益

- 响应时间 P95 降低 300ms
- 去掉了离线计算，节省 60% 的计算资源
- 内容候选池更大，为算法提供了更大的空间
- 用户动态实时分发，算法实时调整

遇到的问题

- Module 更新不生效，继续调用老 Module
- Redis 单线程 CPU 瓶颈，高可用
- Redis 的内存浪费

Module 更新不生效

- 加载新的 so 后，发现调用的还是老逻辑
  - GDB 发现存在符号被标记 DF_1_NODELETE
  - dlcose 仅仅声明 so 不在被系统使用，so 内存占用依旧存在
  - gcc 编译使用 STB_GNU_UNIQUE ，防止符号被标记为 DF_1_NODELETE
  - 按照依赖路径加载 so，不直接加载定制的 so

Redis 单线程 CPU 瓶颈，高可用

- 采用 Redis Sentinel 部署集群，保证高可用
- 一致性哈希 Redis shard，每个 shard 采用 master slave 的方式部署

Redis 内存浪费

- 采用 protobuf 和 Redis ziplist 数据压缩，减少 shard
- 有多个 slave，还是浪费内存，没有根本解决问题
  - Redis 仅仅作为任务队列，任务分派给其他进程处理?
  - 同一台机器伴随 Redis 部署计算节点?

Feed 架构的历程

- Feed 都一样
- Feed 个性化，推模型
- Feed 个性化，拉模型 + 离线计算
- Feed 个性化，拉模型，采用 Redis Module，计算接近存储

参考资料

- Serving Facebook Multifeed: Efficiency, performance gains through redesign
- dlclose - close a symbol table handle
- dlclose doesn't really unload shared object, no matter how many times it is called
- Redis Modules: an introduction to the API
- Redis Loadable Modules System

THANKS