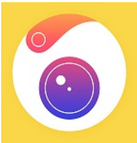
基于Swoole的品果微服务框架创新实践

Camera360社区服务器端团队 徐典阳

为什么要研发新的PHP框架?



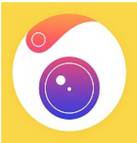
关于我

- 徐典阳（phpboy，http://www.phpboy.net/）
- 社区服务器端团队负责人（Camera360 PHP架构师）
- 原今日头条特卖频道PHP架构师
- 开源项目：firephp splclassloader yaf.auto.complete

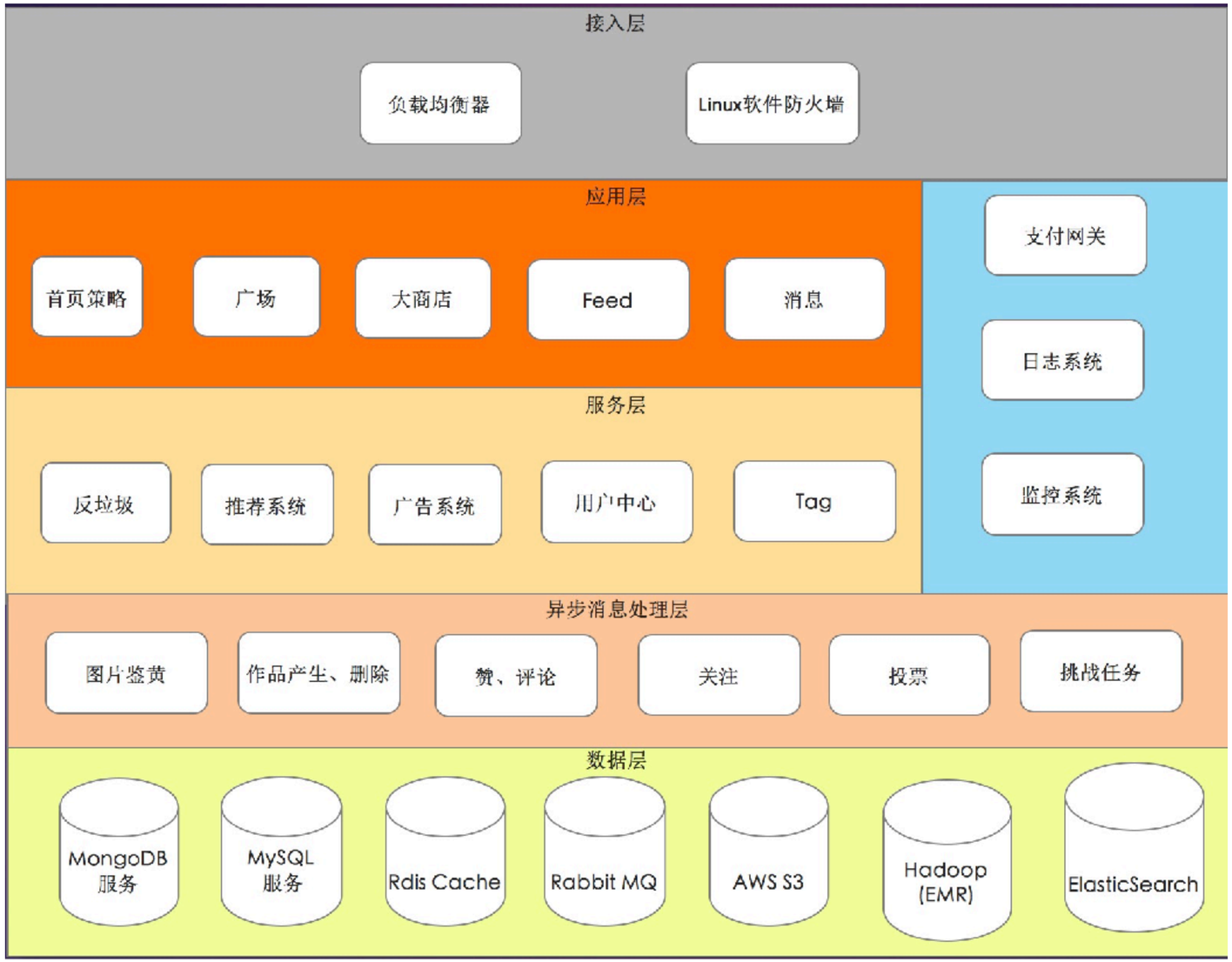
大纲

1. 自主研发微服务框架背景与目标
2. 基于Swoole的高性能组件
3. 业务重构后性能提升

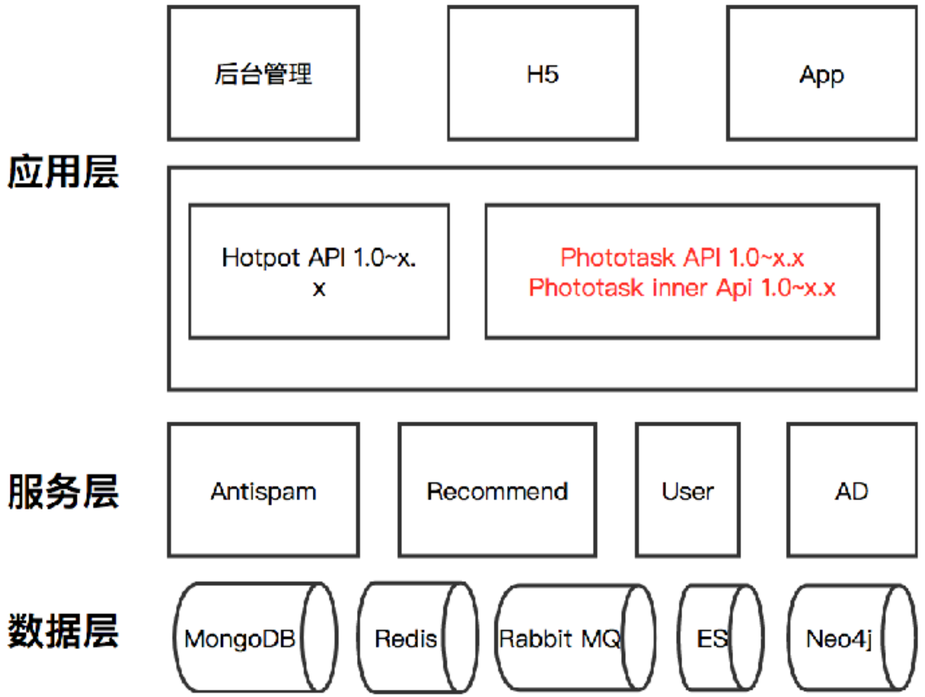
1.自主研发框架背景与目标



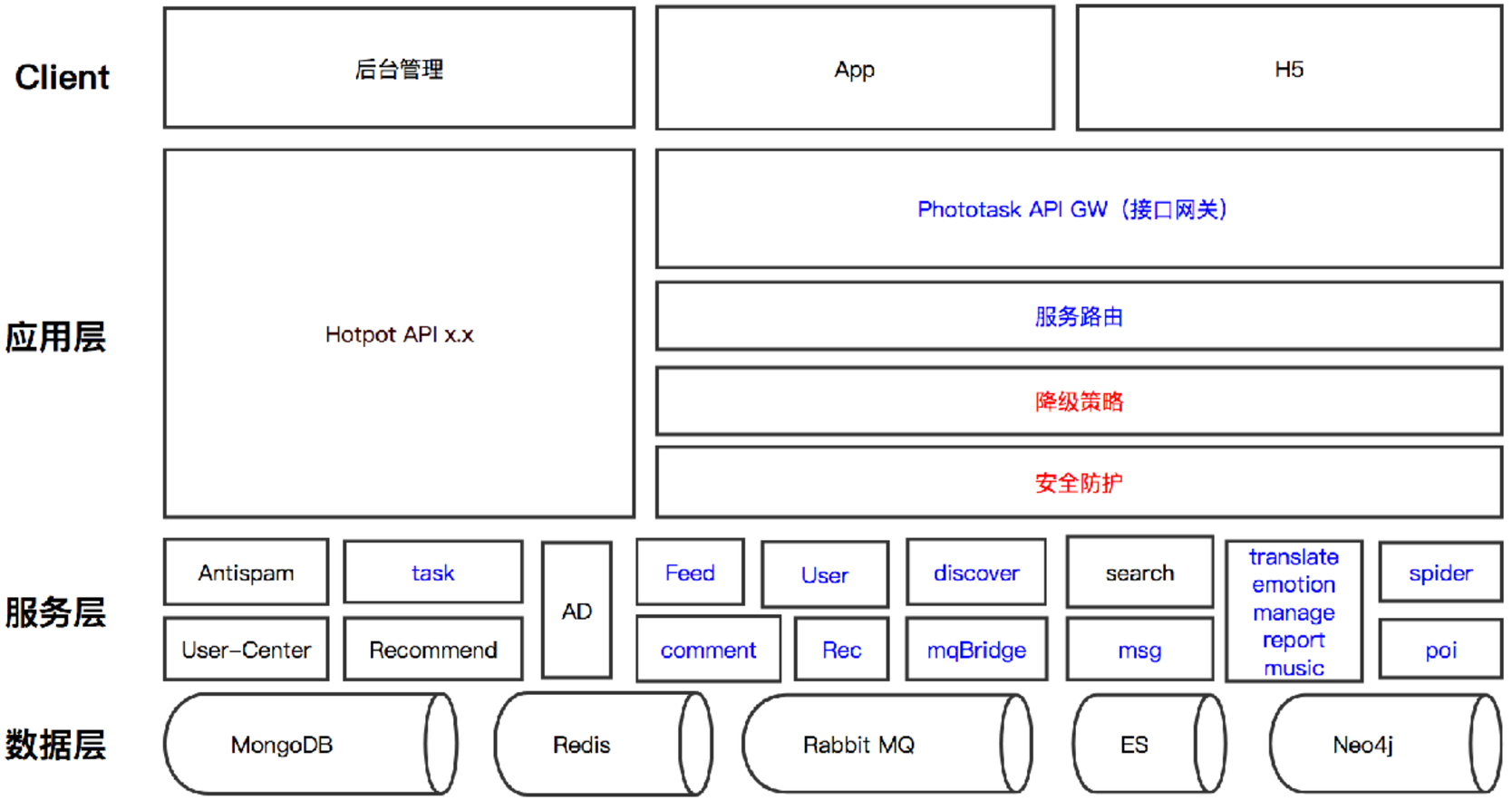
社区总体业务架构


接入层
负载均衡器
Linux软件防火墙
应用层
首页策略
广场
大商店
Feed
消息
支付网关
日志系统
监控系统
服务层
反垃圾
推荐系统
广告系统
用户中心
Tag
异步消息处理层
图片鉴黄
作品产生、删除
赞、评论
关注
投票
挑战任务
数据层
MongoDB 服务
MySQL 服务
Rdis Cache
Rabbit MQ
AWS S3
Hadoop (EMR)
ElasticSearch

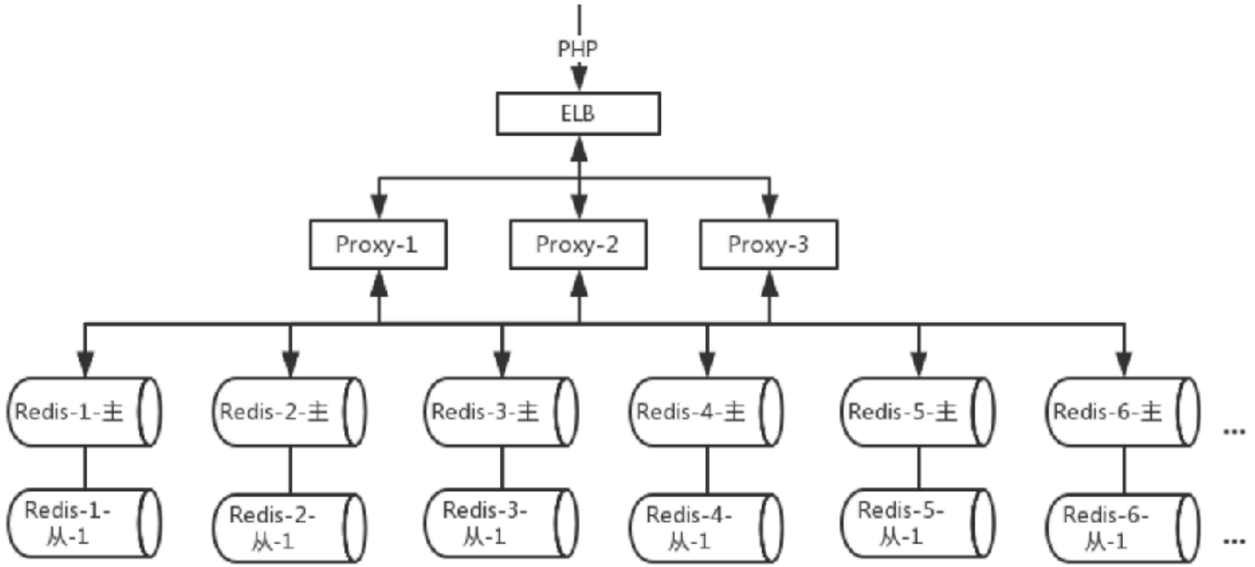
社区架构v1


后台管理
H5
App
应用层
Hotpot API 1.0~x.x
Phototask API 1.0~x.x
Phototask inner Api 1.0~x.x
服务层
Antispam
Recommend
User
AD
数据层
MongoDB
Redis
Rabbit MQ
ES
Neo4j

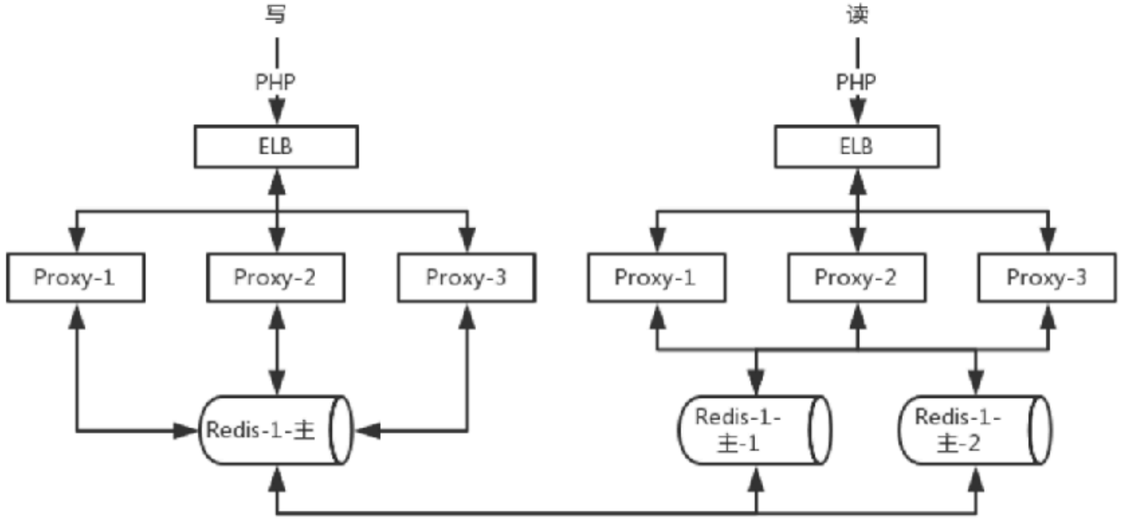
社区架构v2


Client
后台管理
App
H5
应用层
Hotpot API x.x
Phototask API GW（接口网关）
服务路由
降级策略
安全防护
服务层
Antispam
task
AD
Feed
User
discover
search
translate
emotion
manage
report
music
spider
User-Center
Recommend
comment
Rec
mqBridge
msg
poi
数据层
MongoDB
Redis
Rabbit MQ
ES
Neo4j

1.新加坡分布式缓存

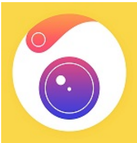


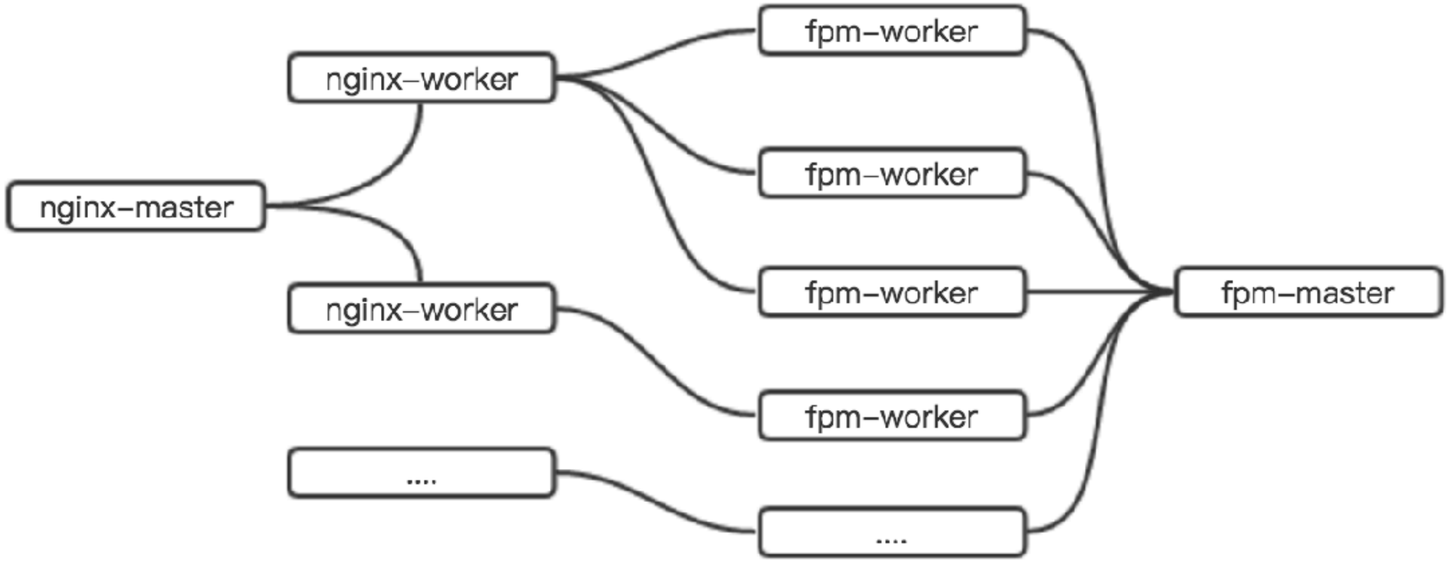
2.新加坡热数据缓存



php-fpm

- Fastcgi进程管理器，实现fastcgi协议
- 同步阻塞IO进程模型
- 请求结束后释放所有资源和内存
- 并发受限于进程数
- PHP框架初始化占用大量的计算资源





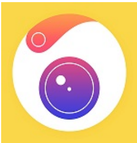
nginx+php-fpm工作模式

1. nginx基于epoll事件模型，一个worker同时可处理多个请求
2. fpm-worker在同一时刻可处理一个请求
3. master进程只负责处理worker进程的监控、日志等
4. 用户端请求由elb解析，再经过nginx解析
5. fpm-worker每次处理请求前需要重新初始化mvc框架，然后再释放资源
6. 高并发请求时，fpm-worker不够用，nginx直接响应502
7. fpm-worker进程间切换消耗大（4核8G 内存服务器实质可利用16个进程）

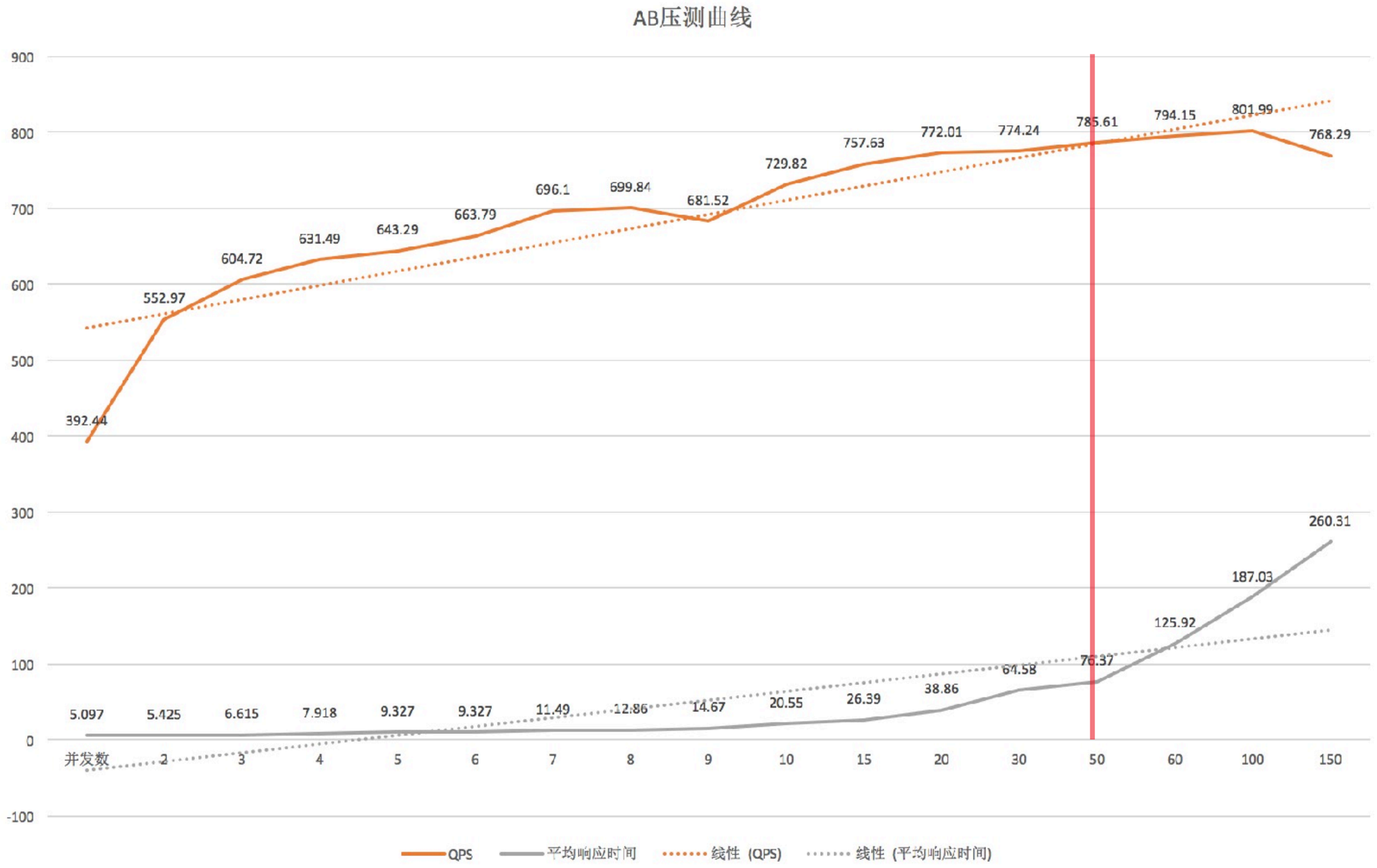
微服务化的问题

- 单机处理能力(QPS)低
- 服务架构运行成本高
- 运维及部署成本高，服务不易治理
- 微服务间通信成本高

核心问题：高并发高性能

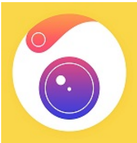


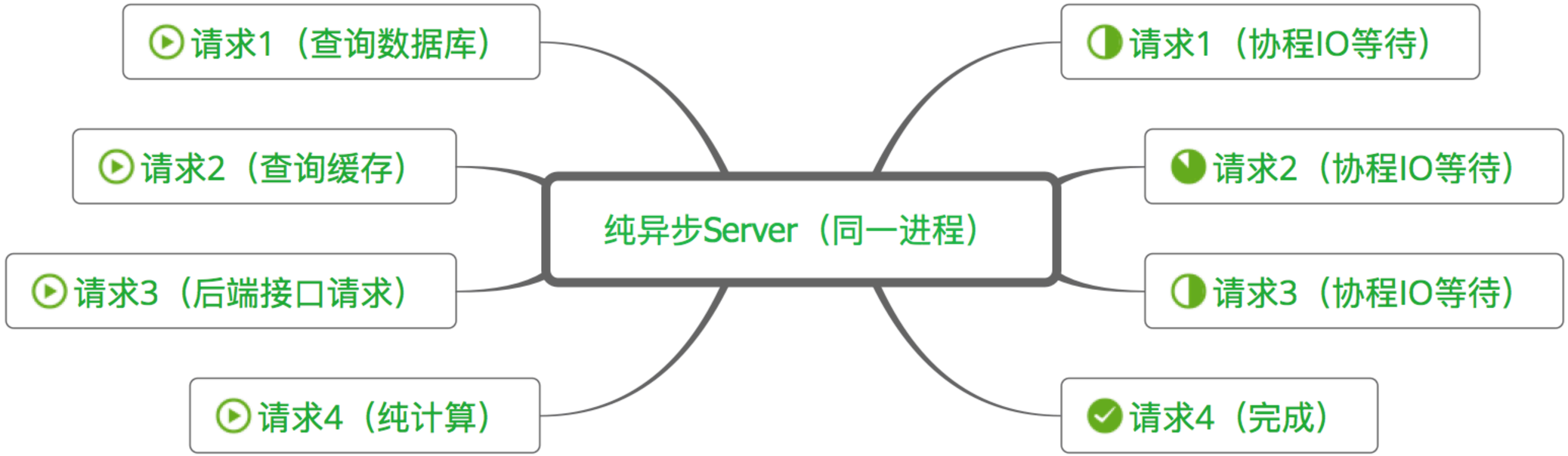
$$QPS\uparrow = \frac{C\uparrow}{T}$$


AB压测曲线
900
800
700
600
500
400
300
200
100
0
-100
392.44
552.97
604.72
631.49
643.29
663.79
696.1
699.84
681.52
729.82
757.63
772.01
774.24
785.61
794.15
801.99
768.29
5.097
5.425
6.615
7.918
9.327
9.327
11.49
12.86
14.67
20.55
26.39
38.86
64.58
76.37
125.92
187.03
260.31
并发数
2
3
4
5
6
7
8
9
10
15
20
30
50
60
100
150
QPS
平均响应时间
线性 (QPS)
线性 (平均响应时间)

新框架开发的目标

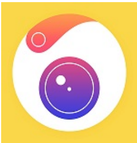
- 精简版的MVC框架
- IO密集性业务的单机处理能力提升5-10倍
- 代码可长驻内存，支持对象池、连接池等
- 支持异步、并行、协程等非阻塞编程模型
- 原生支持纯异步Tcp/Http Server
- RPC Server/Client




请求1（查询数据库）
请求2（查询缓存）
请求3（后端接口请求）
请求4（纯计算）
纯异步Server（同一进程）
请求1（协程IO等待）
请求2（协程IO等待）
请求3（协程IO等待）
请求4（完成）

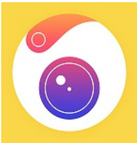
框架底层选型一

- WorkerMan
  - 优势
    - 一个高性能的PHP Socket服务器框架
    - 支持TCP/UDP/HTTP
    - 纯PHP开发
    - 组件丰富
  - 劣势
    - PHP的内存管理粒度粗
    - PHP无法直接调用操作系统API



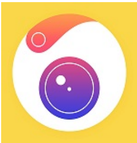
框架底层选型二

- Swoole
  - 优势
    - PHP异步、并行、事件驱动高性能网络通信的 C 扩展
    - TCP/UDP/HTTP/WEBSOCKET服务器
    - 原生支持异步Redis/MySQL/HttpClient
    - 基于epoll的reactor模型
    - 稳定
    - 社区相当活跃
  - 劣势
    - 底层问题解决难度相对较大



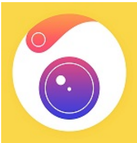
框架底层选型三

- 自主开发
  - 优势
    - 功能定制（select/poll/epoll/Reactor模型/TCP/IP协议等等）
    - 灵活度高
  - 劣势
    - 重复造轮子
    - 成本高，时间周期长
    - 知识储备不够
    - 缺乏人才



框架底层选型四

- Go lang
  - 优势
    - 语言级的高性能高并发
    - 协程
    - 适合底层后端系统的开发
  - 劣势
    - 静态语言，开发效率和PHP相差大
    - 不符合现有团队的基团



技术选型的优序矩阵

		MVC	5-10倍	代码长驻	协程	Http/Tcp Server	RPC
2	MVC		5-10倍	代码长驻	协程	MVC	MVC
5	5-10倍			5-10倍	5-10倍	5-10倍	5-10倍
1	代码长驻				协程	Http/Tcp Server	Http/Tcp Server
4	协程					协程	协程
3	Http/Tcp Server						Http/Tcp Server
0	RPC						

WorkerMan		Swoole		自主研发		Go Lang	
优	劣	优	劣	优	劣	优	劣
5-10倍	MVC	5-10倍	RPC		MVC	5-10倍	MVC
代码长驻	协程	代码长驻			5-10倍	代码长驻	RPC
Http/Tcp Server	RPC	协程			代码长驻	协程	
		Http/Tcp Server			协程	Http/Tcp Server	
		MVC			Http/Tcp Server		
					RPC		

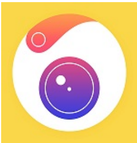
技术选型其他因素

- 适合团队/公司的技术栈（PHP）
- 将擅长的技术做到专业领域更深
- 团队是否具备PHP内核及扩展开发的经验

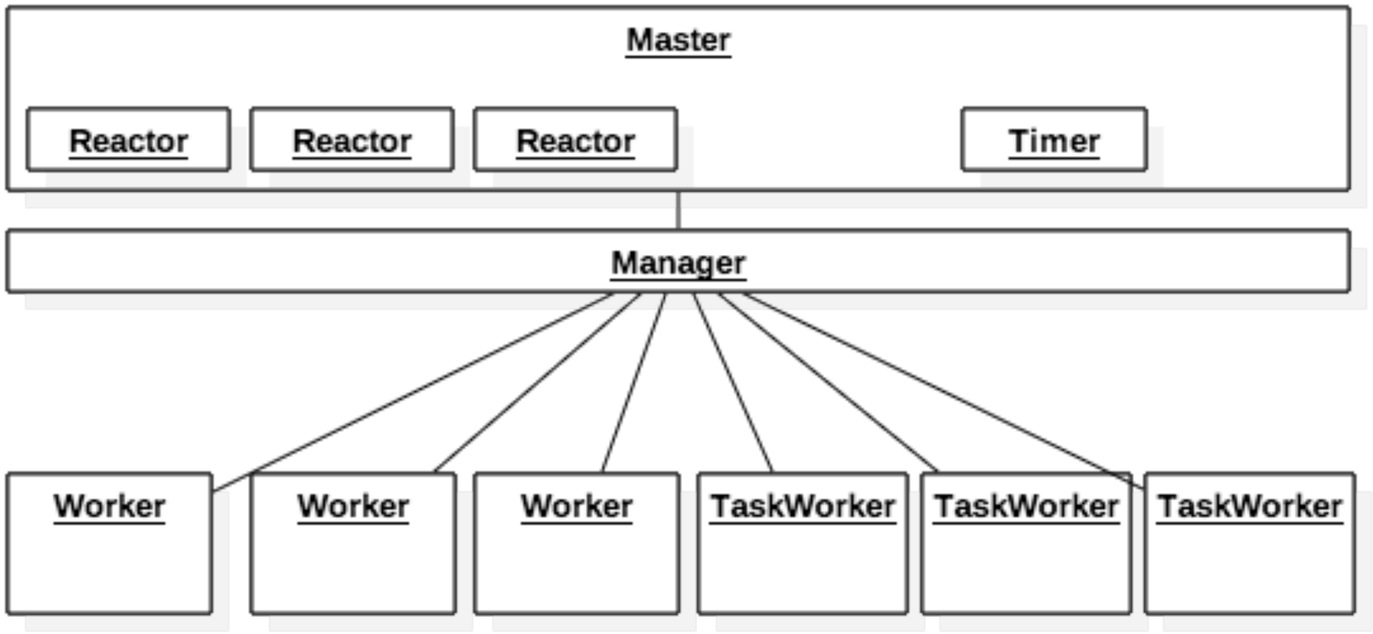
技术选型决策

C扩展+PHP MVC框架

- 支持PHP7以上版本
- C扩展Swoole提供低层Tcp/Http通信
- PHP精简版MVC
- 基于PHP Yield协程调度器
- PHP实现对象池、连接池等特性
- PHP实现RPC Server/Client



Swoole进程模型



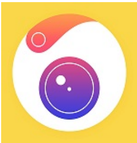
- 多进程模式的框架，启动时创建2 + n + m个进程；
- 其中n为Worker进程数，m为TaskWorker进程数，Master进程和Manager进程；
- Reactor线程实际运行epoll实例，用于accept客户端连接以及接收客户端数据；
- Manager进程为管理进程，该进程的作用是创建、管理所有的Worker进程和TaskWorker进程；

类似开源项目

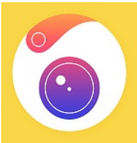
- 腾讯：tsf
- 有赞：zanphp
- 白猫：SwooleDistributed

仅供参考，需要自行大量的封装

- 日志（调用链）
- 协程调度器的性能
- 请求上下文（切面编程的支持）
- 对象池
- 连接池
- RPC/RESTFul
- 公共库（通用性）
- 命令行模式...

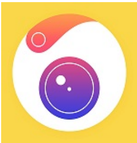


2.基于Swoole的高性能组件



PHP实现协程

- 英文：coroutine
- 特性：协程和线程一样共享堆，不共享栈，协程由程序员在协程的代码里显示调度；
- 优点：协程避免了无意义的调度，由此可以提高性能；
- 劣势：无标准线程使用多CPU的能力；
- PHP使用生成器和Yield关键字实现协程，生成器是一种具有中断点的函数，而yield可构成中断点，进而把控制权交回给协程调度器；
- 调度器<—数据（Yield）—>任务
- 利用 IO并行、异步事件的机制，配合yield关键字实现协程，很好的解决了异步IO回调的写法，让程序看上去是同步执行的。



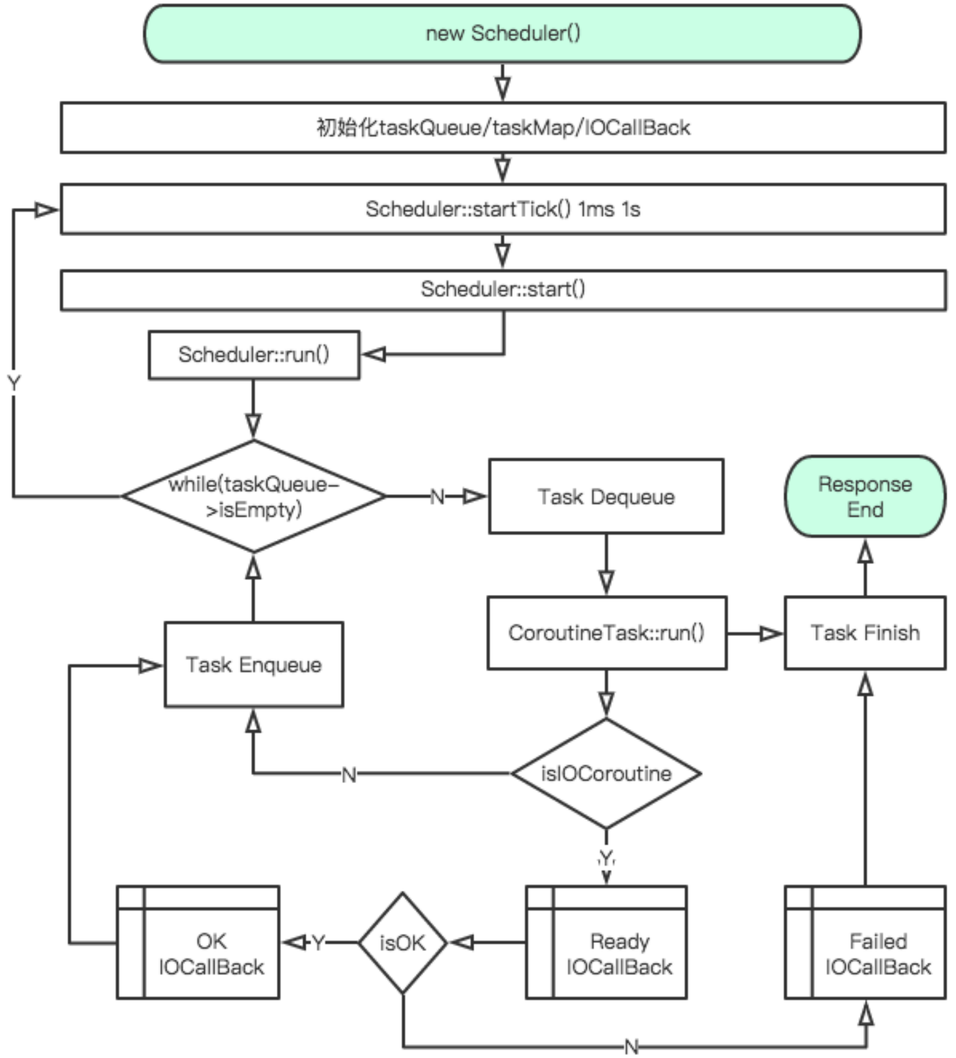
```php
/**
 * 异步回调的方式实现(A && B) || C
 */
public function httpCallBackMode()
{
    $client = new \swoole_redis;
    $client->connect('127.0.0.1', 6379, function (\swoole_redis $client, $result) {
        $client->get('apiCacheForABCallBack', function (\swoole_redis $client, $result) {
            if (!$result) {
                swoole_async_dns_lookup("www.baidu.com", function($host, $ip) use ($client) {
                    $cli = new \swoole_http_client($ip, 443, true);
                    $cli->setHeaders([
                        'Host' => $host,
                    ]);
                    $apiA = "";
                    $cli->get('/', function ($cli) use ($client, $apiA) {
                        $apiA = $cli->body;
                        swoole_async_dns_lookup("www.qiniu.com", function($host, $ip) use ($client, $apiA) {
                            $this->outputJson( data: 'ok');return;
                            $cli = new \swoole_http_client($ip, 443, true);
                            $cli->setHeaders([
                                'Host' => $host,
                            ]);
                            $apiB = "";
                            $cli->get('/', function ($cli) use ($client, $apiA, $apiB) {
                                $apiB = $cli->body;
                                if ($apiA && $apiB) {
                                    $client->set('apiCacheForABCallBack', $apiA . $apiB, function (\swoole_redis $client, $result) {});
                                    $this->outputJson( data: $apiA . $apiB);
                                } else {
                                    $this->outputJson( data: '', message: 'error');
                                }
                            });
                        });
                    });
                });
            } else {
                $this->outputJson($result);
            }
        });
    });
}
```

```php
/**
 * 协程的方式实现(A && B) || C
 */
public function httpCoroutineMode()
{
    // 从Redis获取get k1
    $get      = $this->getRedisPool( poolName: 'tw')->get('k1');
    $response = yield $get;
    if (!$response) {
        // 并行请求百度和七牛首页
        $requests = [
            // 注册的服务接口名
            'baidu' => [
            ],
            // 注册的服务接口名
            'qiniu' => [
            ],
        ];
        $results  = yield ConcurrentClient::request($requests, $this);
        $response = $results['baidu']['body'] . $results['qiniu']['body'];

        // 写入redis,并不获取服务器返回结果（可大辐提升性能）
        $this->getRedisPool( poolName: 'tw')->set( key: 'k1', $response)->break();
    }

    // 响应结果
    $this->outputJson($response);
}
```

PHP协程调度器

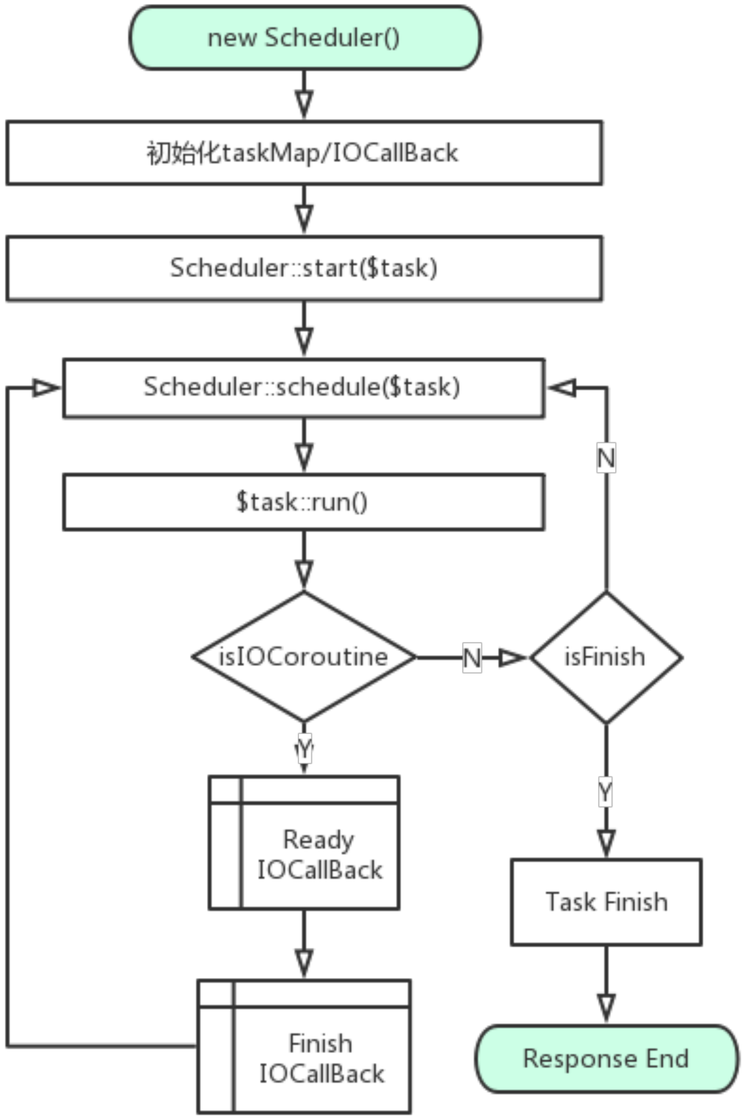
- 简单来说，是可以在多个任务之间相互协调，及任务之间相互切换的一种进程资源的分配器。
- 基于PHP Yield调度器的实现方式有多种，大致分为四类：
- 一是，队列；二是，定时器；三是，定时器+队列；四是，基于IO事件触发。

协程调度流程图v1



损失50%

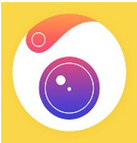
协程调度流程图v2



损失20%

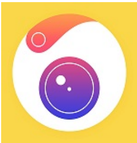
请求上下文

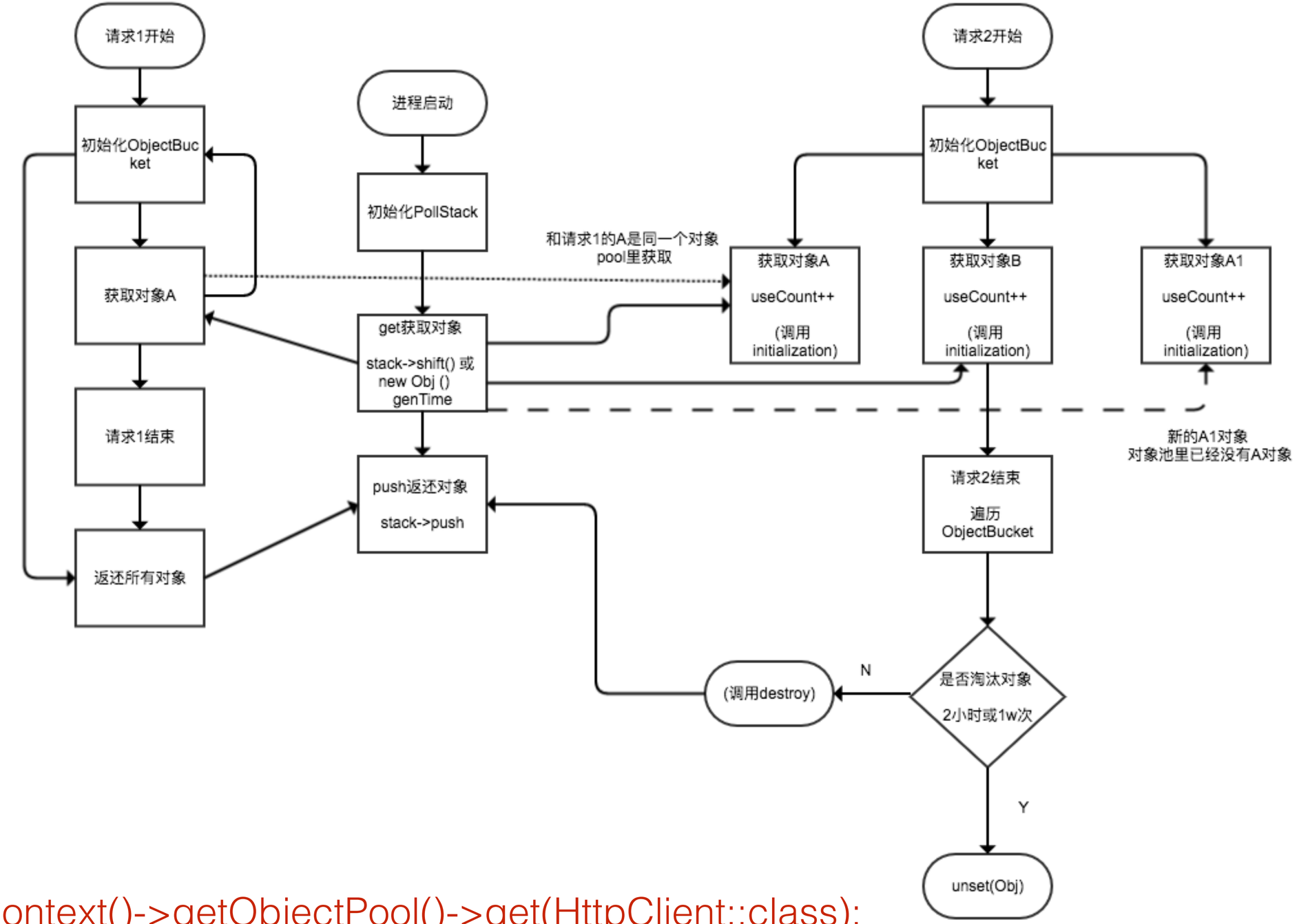
- 应该暂时忘记
  - $_GET/$_POST
  - $_GLOBAL
  - $_REQUEST
  - 类的静态属性
- 在任何地方都应该是：$this->getContext()
  - getInput()
  - getLog()
  - getOutput()
  - getObjectPool()
  - getUserDefined()
  - …



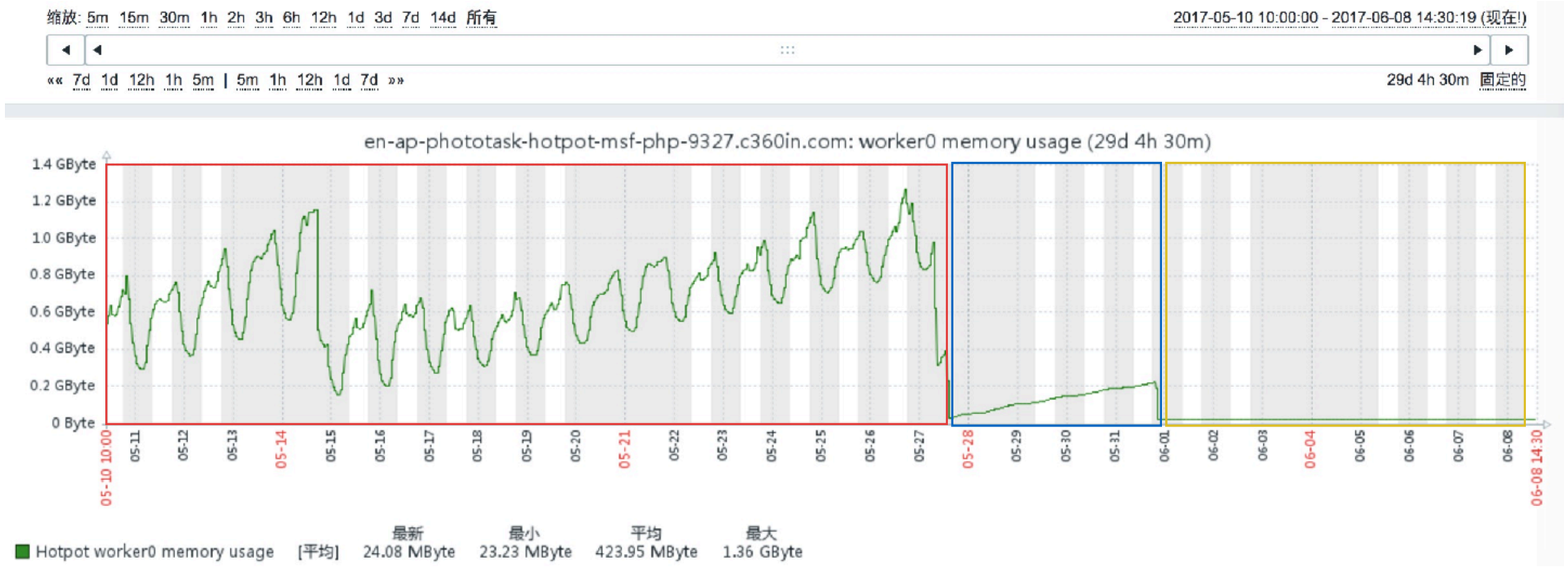
对象池

- 目标
  - 减少从头创建每个对象的系统开销；
- 特性
  - 创建并发所需数量的对象；
  - 需要时从池中提取，不需要时归还池中；
  - 自动归还对象；
  - 根据有效期和使用次数淘汰对象；

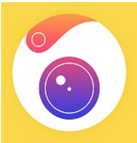




$this->getContext()->getObjectPool()->get(HttpClient::class);

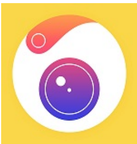


- 近30天社区首页业务的其中一台机器的一个worker进程内存占用的监控数据,从图中有几点:
- 5-10~5-27,近15天内存占用波峰达1.25G,波谷的值在持续的攀升;
- 5-27~5-31,近5天内存占用从一个较低的点持续攀升;
- 6-01之后,内存占用持续稳定在25M上下小辐波动;
- 在1阶段,框架大量类的对象是直接使用new关键字创建,完全依赖的PHP GC进行内存资源的回收;
- 在2阶段,框架采用对象池的方案完全重构大量的逻辑,效果很明显,但仍然有部分内存泄露;
- 在3阶段,优化了业务逻辑,完全按照“资源释放”的策略调整业务代码,内存占用已稳定。



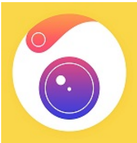
资源释放

- className::destroy()
  - 每一个类,定义destroy()方法,用于资源的手工释放,但是不需要显示调用,在请求结束时由框架自动调用。
  - 对于和请求相关的数据,理论上我们是需要在请求结束后释放相关资源,如下:
    - 1. 响应请求后调用Controller::destroy()
    - 2. 依次调用当前请求所使用的对象的destroy()方法
    - 3. 将所有public的类属性的值设置为初始值
- 通常情况下,destroy方法用于处理private,protected的类属性,public由框架自动清理。



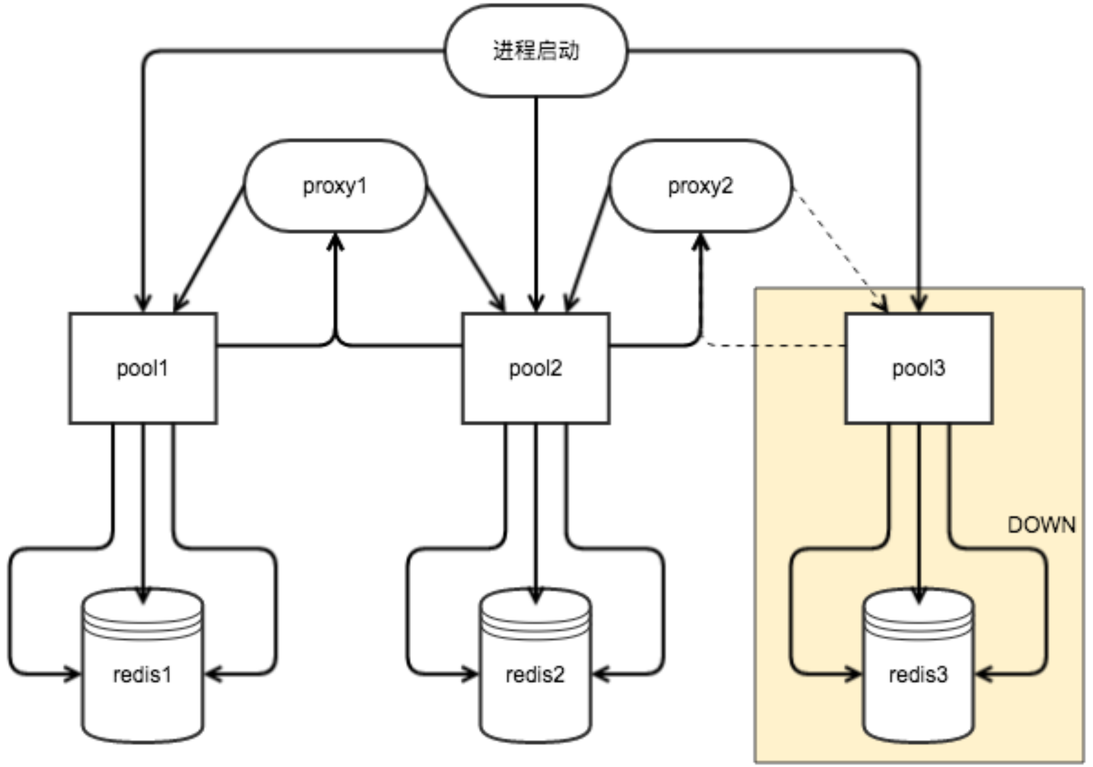
Redis连接池与代理

- Redis连接池主要特性
  - 支持异步+协程
  - 支持断线重连
  - 支持自动提取和归还连接
- Redis代理主要特性
  - 支持Redis分布式自动分片
  - 支持Redis master-slave读写分离
  - 支持故障自动failover



Redis连接池与代理关系

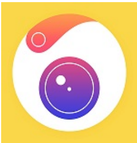
```
Config Demo

redis:
        pool1
        pool2
        pool3

proxy:
        proxy1:
                pool1
                pool2
        proxy2:
                pool2
                pool3
```




```
$this->getRedisPool('tw')->get('k1');

$this->getRedisProxy('cluster')->set('k1', 'v1');
```

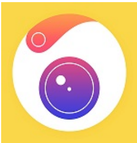
Worker-Tasker机制

- Worker异步非阻塞、Task同步阻塞
- 异步(同步)任务投递并托管
- Unix Socket管道通信、全内存、无IO消耗
- 进程忙闲检查、空闲投递、任务排队
- 以协程方式调用



Worker-Tasker应用场景

- 任何无法直接异步的逻辑均可封装为Task
- 批量处理、耗时逻辑



RPC

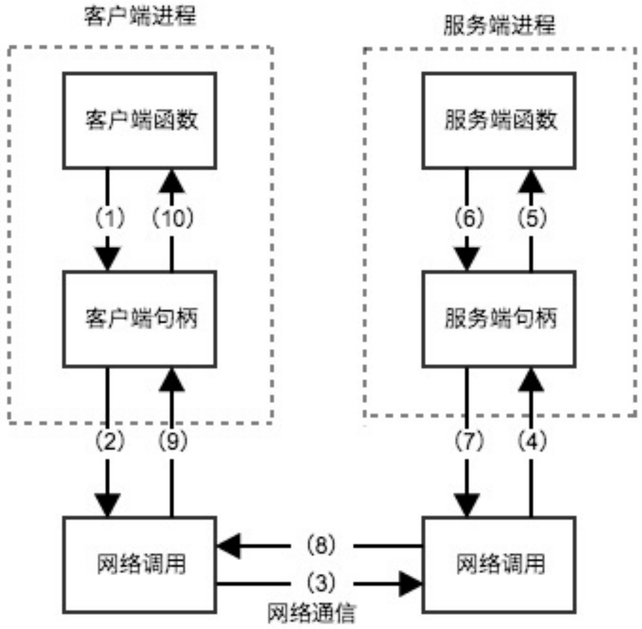
- 定义

远程过程调用,是指网络计算机之间相互调用过程函数,而不需要关心底层的网络通信。
MSF的RPC 目前是基于Http协议,当然使用者不需要关心,因为后续有可能会有调整。

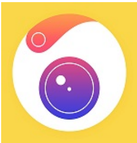
- 基本原则

1. 任何类可导出并提供服务
2. RPC Client与RPC Server的类名与方法名一致
3. 任何服务可以很方便的导出自己的类与类方法

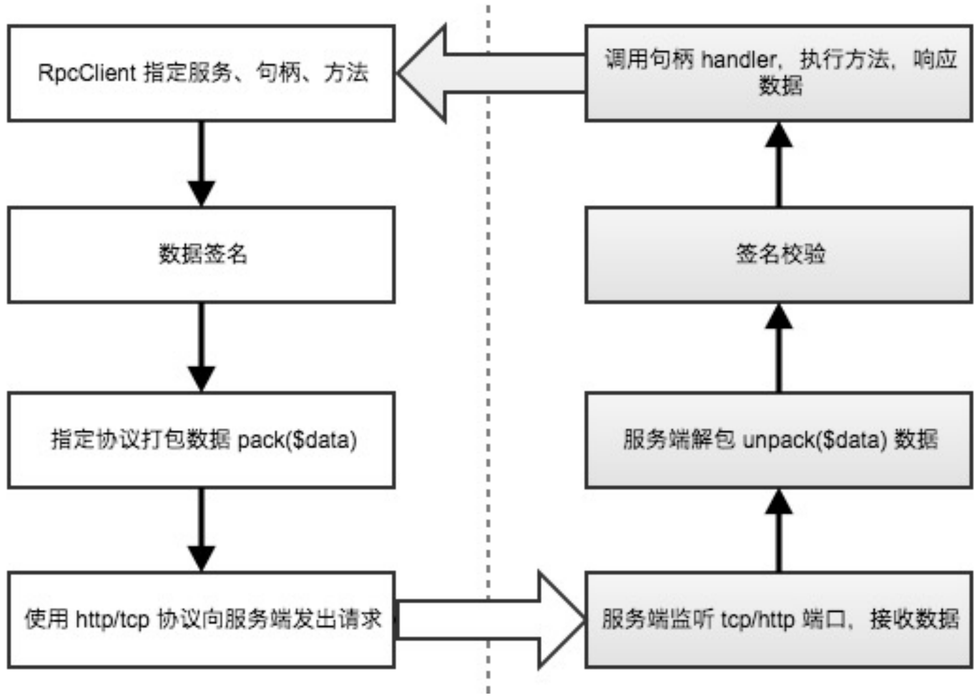
RPC调用过程



1. 调用客户端句柄，执行传送参数
2. 调用本地系统内核发送网络消息
3. 消息传送到远程主机
4. 服务器句柄得到消息并取得参数
5. 执行远程过程
6. 执行的过程将结果返回服务器句柄
7. 服务器句柄返回结果，调用远程系统内核
8. 消息传回本地主机
9. 客户句柄由内核接收消息
10. 客户接收句柄返回的数据



MSF RPC

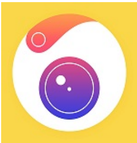


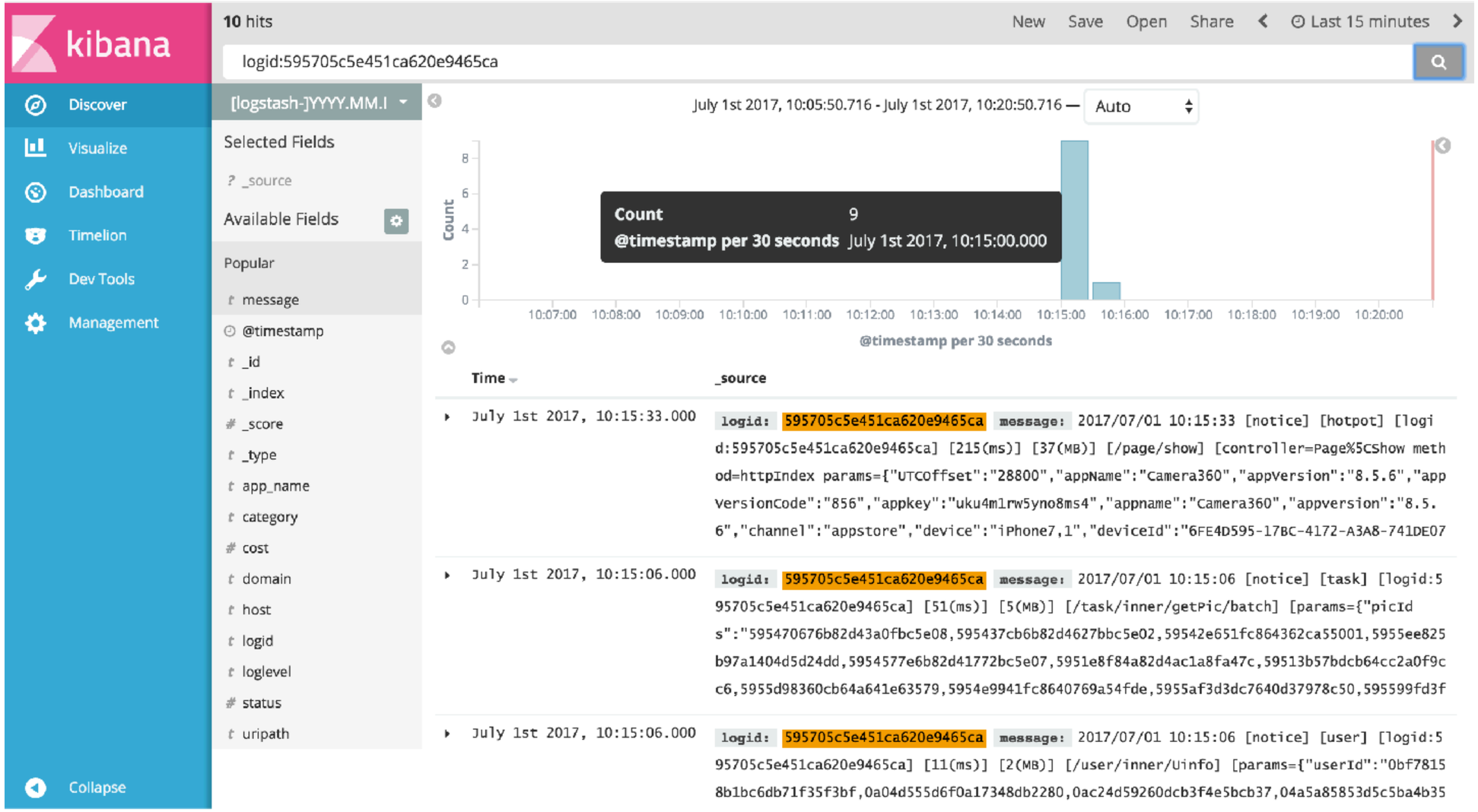

```
RpcClient::serv('user')->handler('UserInfo')->getByUid($uid);
```

- RpcClient 兼容不支持 Rpc 的服务

日志

- 异步写日志
  - swoole_async_writefile
- 性能分析
  - profile[init=0.2(ms)/1,verifySign=0.2(ms)/1,params=0.0(ms)/1,yac.get=0.1(ms)/1,ra=0.2(ms)/1,prepare=0.0(ms)/1,599#dns-http://xx1.com=0.0(ms)/1,273#dns-http://xx2.com=0.0(ms)/1,127#http://xx3.com=0.0(ms)/1,596#api-http://xx1.com/a/b/c=12.6(ms)/1,881#api-http://xx2.com/a/b/c=16.0(ms)/1,136#api-http://xx3.com/a/b/c=185.4(ms)/1,geo-ip:117.136.26.242=0.4(ms)/1]
- 服务调用链
  - $client->setHeader('X-Ngx-LogId', $this->context->getLogId());





Logstash+Elasticsearch+Kafka+Kibana

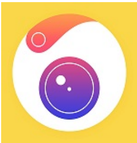
公共库 php-xxx

- 与框架“无关”，具有独立性
- 使用 composer 管理
- 具有版本特性
- 升级便捷，耦合度低
- 应用服务各取所需，单个库功能单一
- 协程的适配

公共库 php-xxx

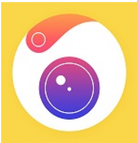
已封装的公共库列表（按功能划分）

- php-log 日志库
- php-exception 异常相关库
- php-filter 权限管理库（如 Acf）
- php-i18n 国际化翻译库
- php-inner-serv 品果内部服务库（如 MQ、Push、SMS、Geo、Poi）
- php-helper 工具助手库
- php-db 数据库lib
- php-third 第三方服务库（如 AWS、七牛）
- php-aop 切面编程支持
- php-context 请求上下文



其他特性

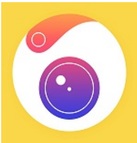
- call_user_func/call_user_func_array -> $func(…$args)
- destroy() -> $obj = null
- route cache -> http://a.b.com/api/pr/rec Api/Pr::rec()
- swoole_async_dns_lookup() -> dns cache
- large array -> SplFixedArray
- refcount()/get_object_handle()/dump()…
- msf command line mode/restful api
- enable inotify auto reload/server running info stat



遇到的问题与风险

上面的实现并非一蹴而就，过程遇到诸多

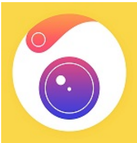
- swoole异步dns解析问题
- hiredis在启动jemalloc时进程崩溃
- php-xxx lib支持协程的问题
- 协程调度器优化
- 如何实现RPC、对象池
- 异步IO超时请求响应后，仍然回调的问题
- 如何实现Redis客户端分片
- swoole http client gzip内存溢出问题
- PHP7本身内存泄露的问题
- …



其他风险

新框架的运行、推广面临的主要有

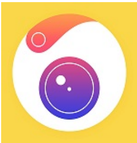
- PHP服务进程的稳定性
- yield影响开发效率
- 团队的学习和培训成本（编程思想、意识）
- 持续发展和维护
- 持续集成系统的改造



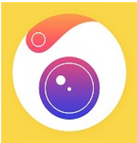
后续发展

基于MSF底成本的实现诸多微服务相关组件

- API Gateway
- 服务重试
- 服务限流
- 安全过滤
- 服务降级
- 服务注册与发现
- 配置系统
- 自动化
- …



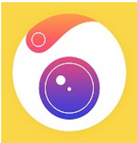
3.业务重构后性能提升



社区评论服务重构后性能

- aws c3.xlarge（4核8G）
- GetAll
  - 4次Redis Get
  - 1次 API
  - 20+次igbinary
  - 2次json_encode
- GetByIds
  - 1次Redis MGet（300+ID）
  - 300+次igbinary
  - 2次API
  - 1次json_encode
- isLike
  - 1次Redis MGet（100+ID）
  - 100+次igbinary
  - 1次json_encode

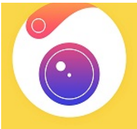
接口	重构前 QPS	重构后 QPS
GetAll	286（77.8%）	1020（83%）
GetByIds	60（90.7%）	210（87%）
isLike	391（86.2%）	2929（79%）



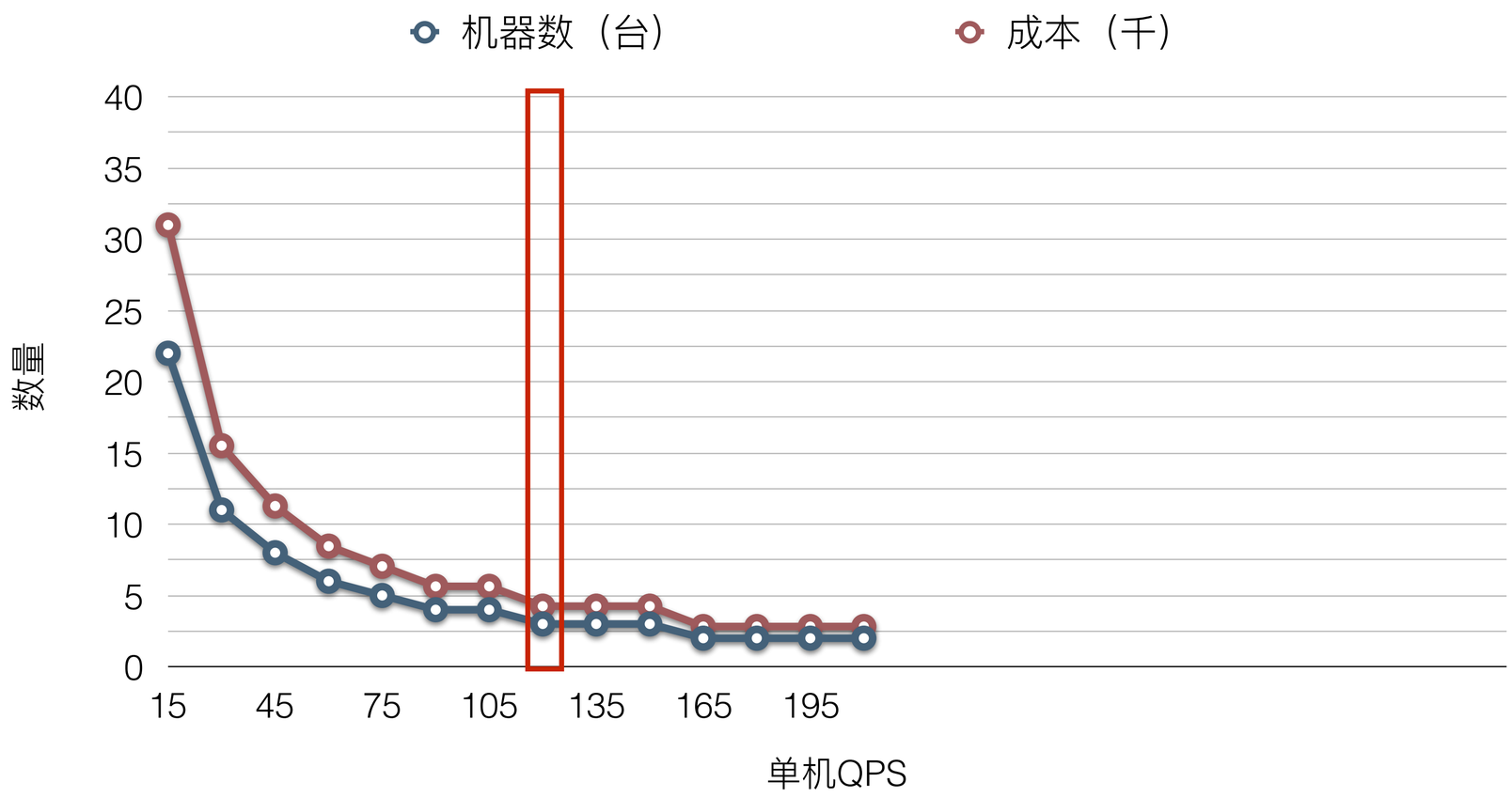
社区首页服务重构后性能

- aws c3.xlarge（4核8G）
- msf-2.0
- 社区首页接口包含
  - 1次Redis Get
  - 1次 MongoDB Query
  - 2个广告接口
  - 2个社区接口
  - 数据组装（100次以上json_encode）
- 瓶颈在于json_encode
- 在相同资源使用率来看，首页提升8倍QPS

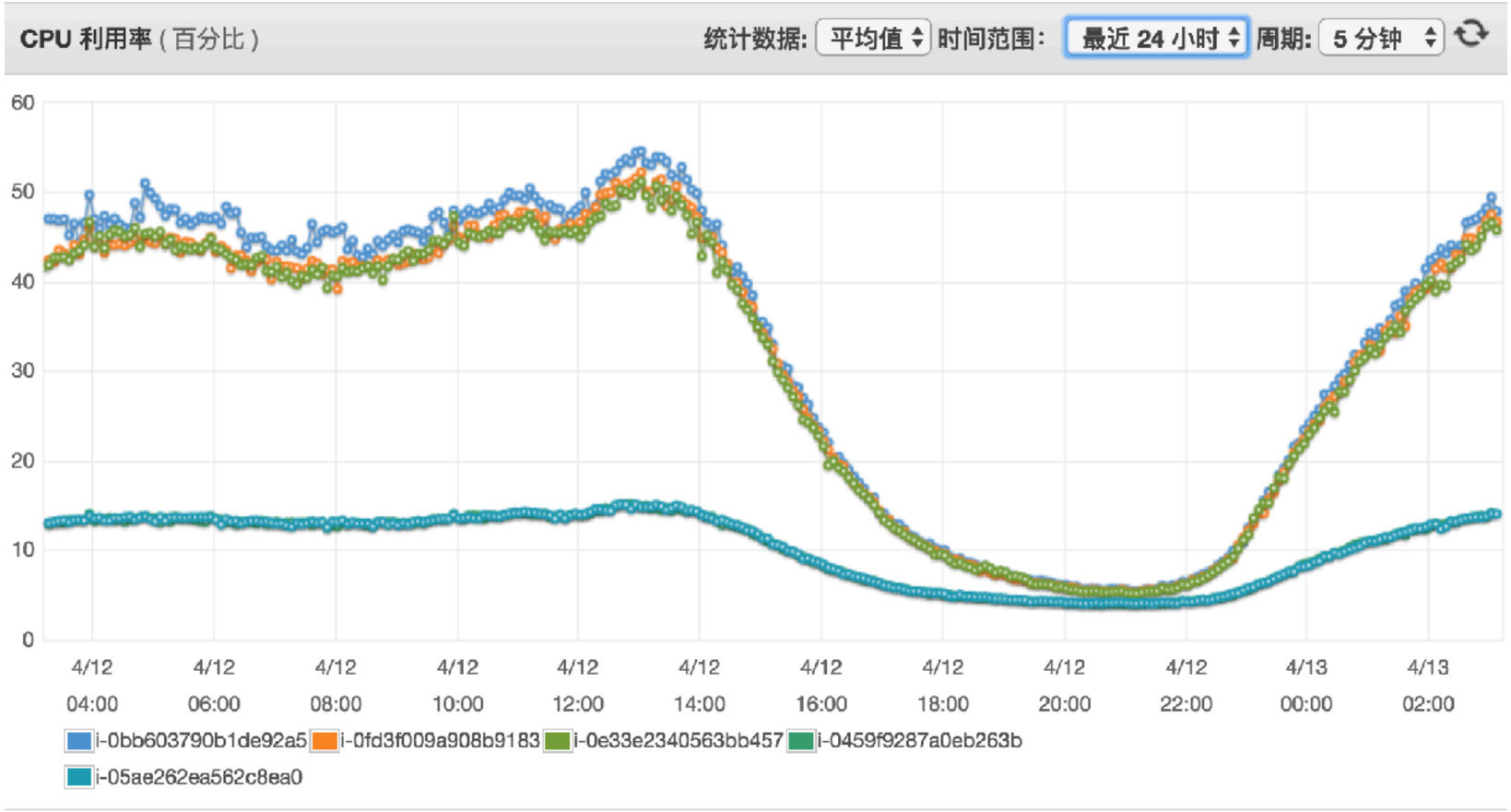
n	c	qps	平均响应时间(ms)	cpu
100	1	4.52	221.089	1.4%
1000	10	40.31	248.063	14%
5000	20	73.22	273.148	27%
5000	40	129.63	308.575	55%
5000	80	174.19	475.916	76%



社区首页成本



社区首页CPU使用


CPU 利用率 (百分比)
统计数据: 平均值
时间范围: 最近 24 小时
周期: 5 分钟
60
50
40
30
20
10
0
4/12 04:00
4/12 06:00
4/12 08:00
4/12 10:00
4/12 12:00
4/12 14:00
4/12 16:00
4/12 18:00
4/12 20:00
4/12 22:00
4/13 00:00
4/13 02:00
i-0bb603790b1de92a5
i-0fd3f009a908b9183
i-0e33e2340563bb457
i-0459f9287a0eb263b
i-05ae262ea562c8ea0

Q&A