

# 腾讯移动分析Crash系统实时化建设与实践

李国栋

腾讯大数据高级工程师



数据平台部

# 目录

01

行业的  
现状与挑战

02

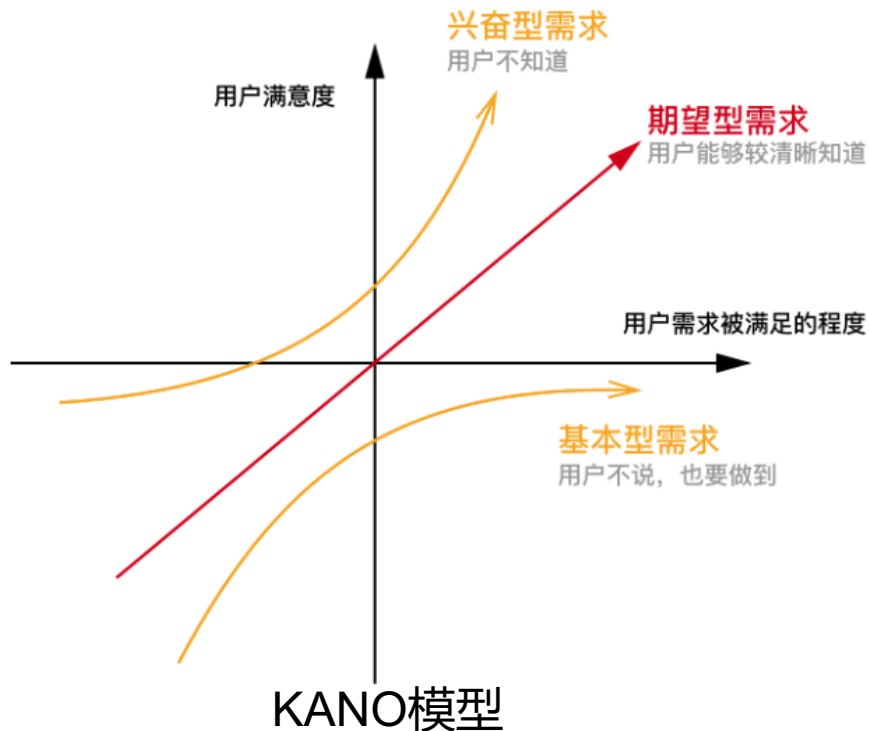
腾讯移动分  
析解决方案

03

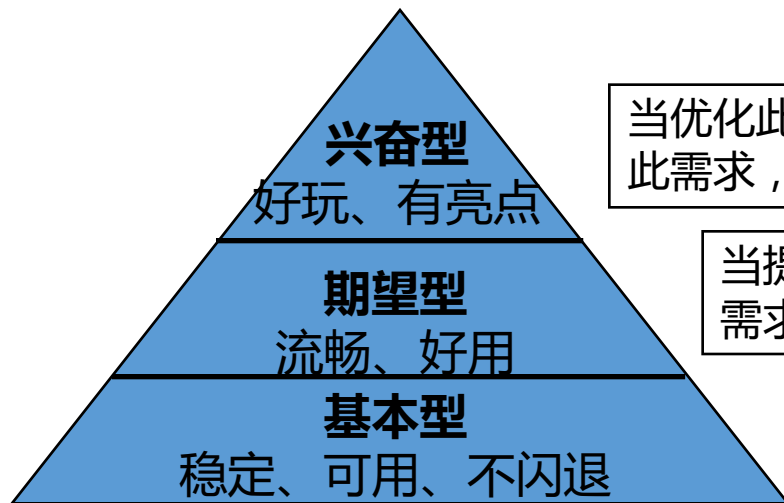
终端  
全平台建设

04

实时与运营  
系统建设



体现产品性能和用户满意之间的**非线性**关系。



当优化此需求，用户满意度不会提升，当不提供此需求，用户满意度会大幅降低。

当提供此需求，用户满意度会提升，当不提供此需求，用户满意度会降低。

当优化此需求，用户满意度不会提升，当不提供此需求，用户满意度会大幅降低。

## 现在的移动App质量情况

60+%



用户遇到过Crash的比例

20+%



首次启动Crash，选择立即卸载的比例

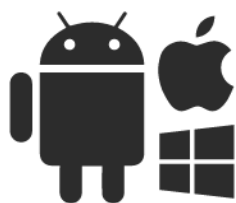
70+%



使用过程中，Crash后用户给应用打差评比例

行业平均Crash率在**3%**以上！

# 面临的挑战



多平台覆盖

iOS  
Android  
游戏引擎



海量实时处理

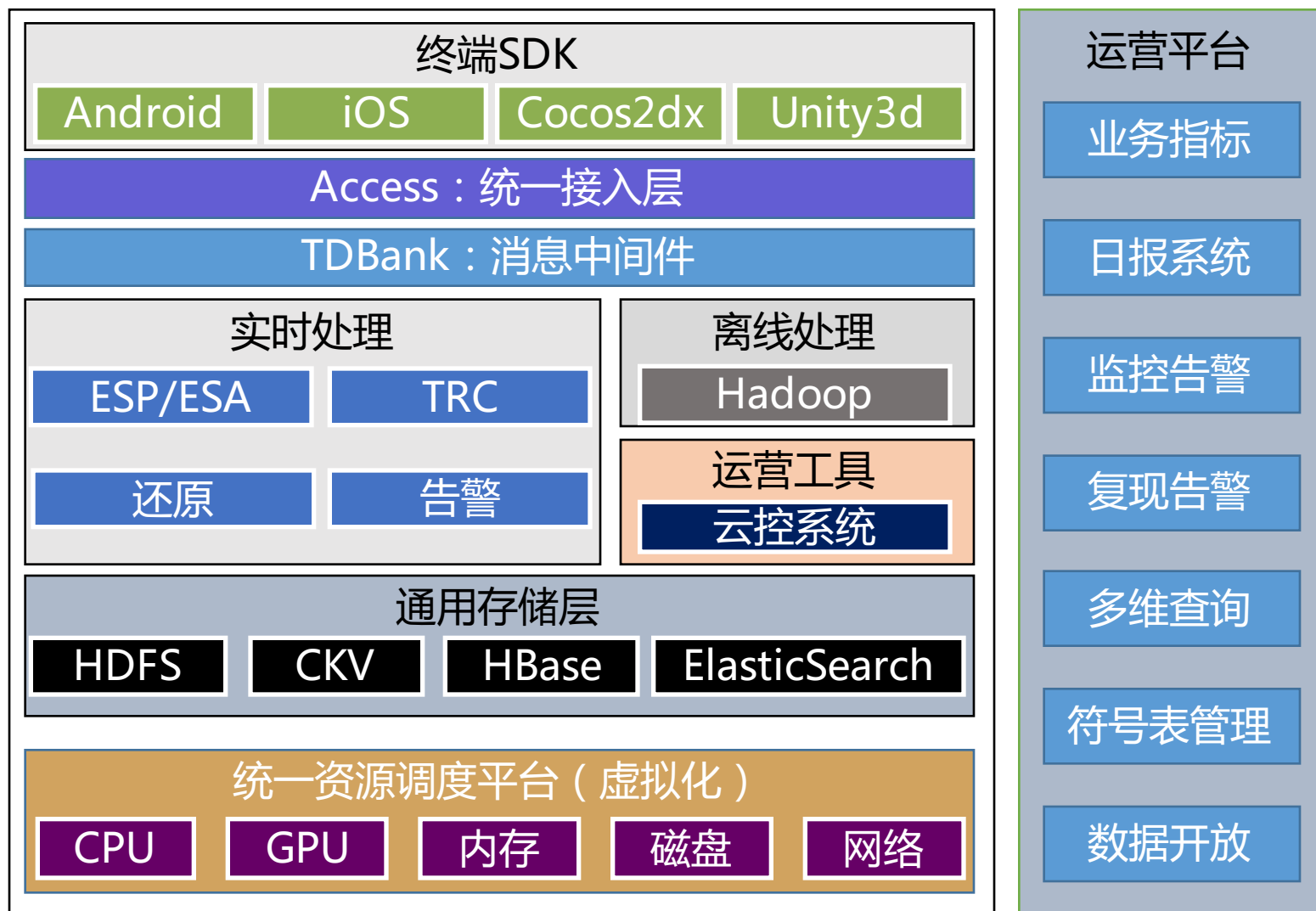
亿级日流量  
实时还原  
实时计算  
实时告警



智能合并检索

堆栈提取  
智能合并  
多维检索

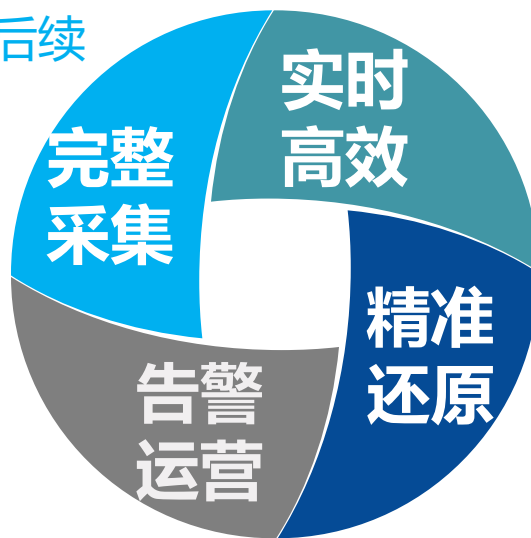
# 腾讯移动分析Crash系统解决方案



# 解决的问题

不同平台和CPU架构环境下的异常数据、堆栈、环境属性、运行参数等数据的完整获取，是后续定位的基础。

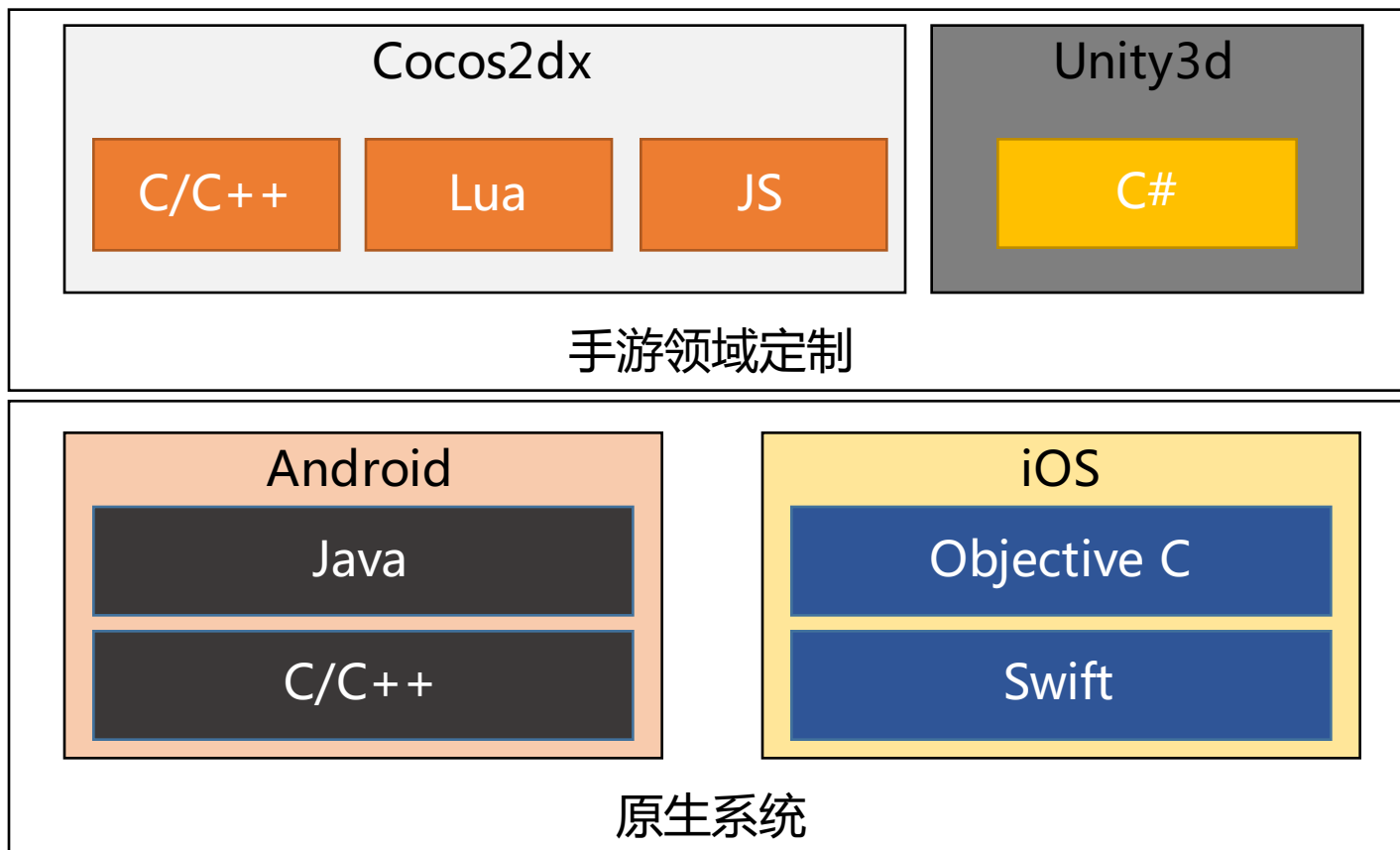
将异常数据实时化处理，第一时间展示给用户，了解产品质量。



完备的监报告警机制能及时监控App质量的波动，把握质量情况；云控助力远程解决问题。

准确的异常追踪过程、精确到行号的堆栈还原，是还原Crash现场的最有力数据。

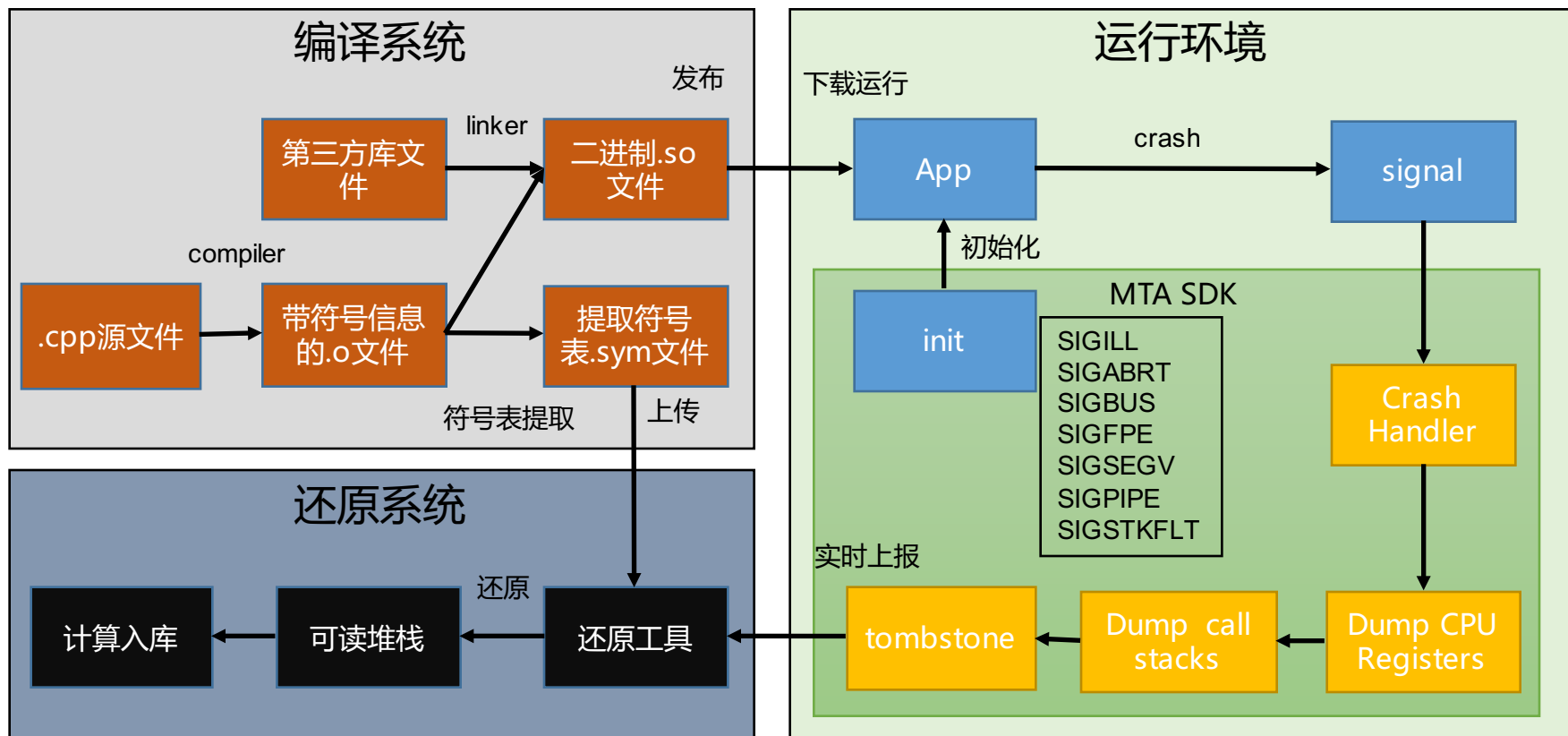
# 终端Crash采集平台建设概览



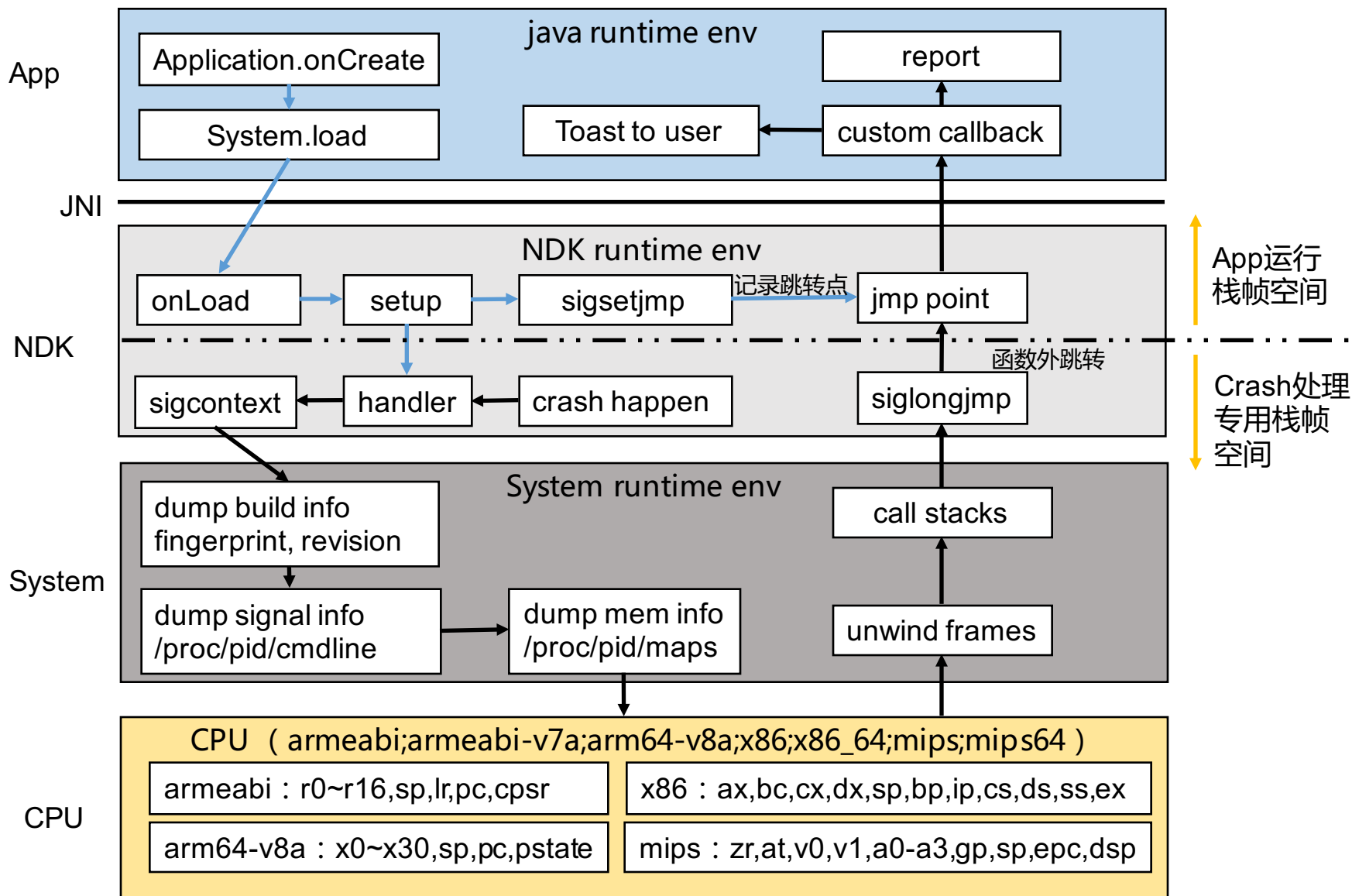
- 覆盖主流平台和语言
- 从系统底层到应用层
- 从通用领域到专业领域定制



# Android C/C++ Crash处理全流程



# C/C++ Crash捕获流程



# 全面数据采集—辅助快速定位异常问题

## When : 时间维度

- 异常时间
- 上报时间
- 运行时长
- 使用时长

单一堆栈到多  
维灵活数据，  
全面还原现场



## What : 状态维度

- 内存使用
- 网络状态
- 上下文数据
- 埋点日志
- 页面执行路径
- 线程状态

## Who : 身份维度

- 厂商机型
- 用户账号
- 设备ID

## Why : 归因维度

- 异常堆栈
- 错误描述
- 信号量
- 关联子线程信息

## Where : 位置维度

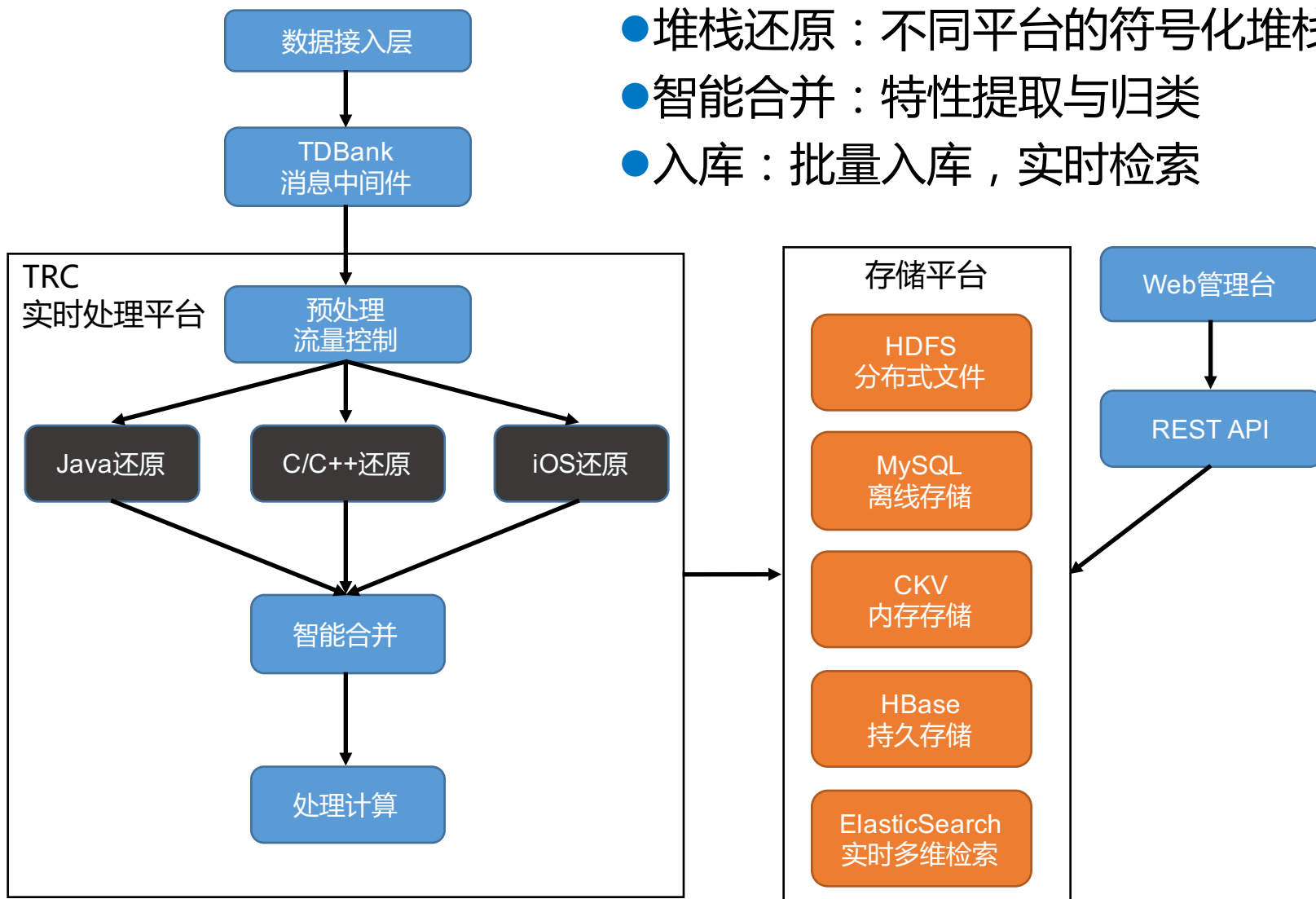
- 组件/模块/页面
- 类/方法/行号

# 实时系统建设主题



# 实时还原系统建设

- 预处理：流量控制、数据过滤
- 堆栈还原：不同平台的符号化堆栈内容
- 智能合并：特性提取与归类
- 入库：批量入库，实时检索



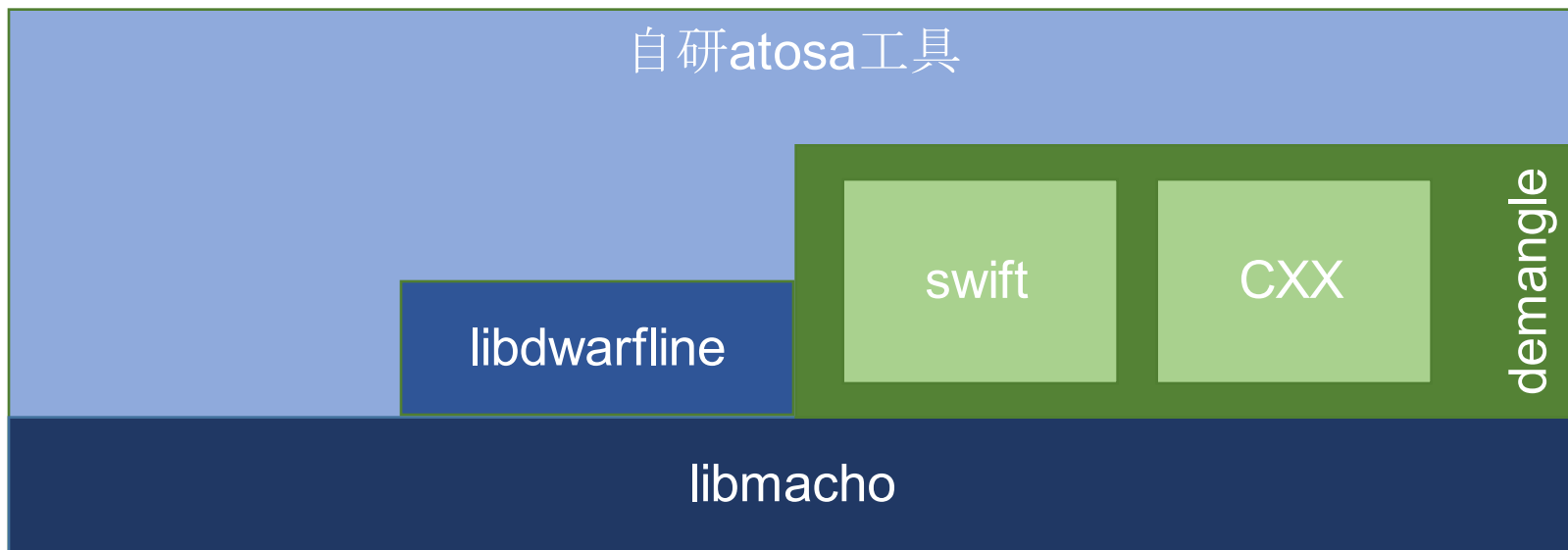
# iOS符号还原

组成：

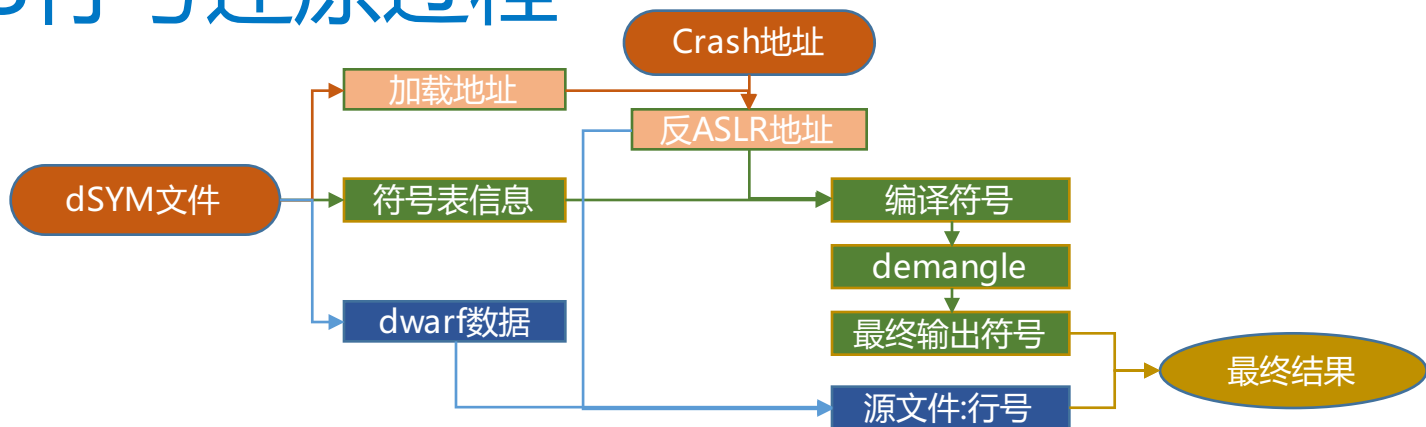
- atosa: 符号还原工具
- libmacho: 解析Mach-O二进制文件
- libdwarfline: 解析dwarf数据
- demangle: 还原swift、C++符号

特性：

- 不依赖于Mac OS系统
- 支持framework符号还原
- 支持Swift语言
- 支持Demangle
- 精准到行号



# iOS行号还原过程



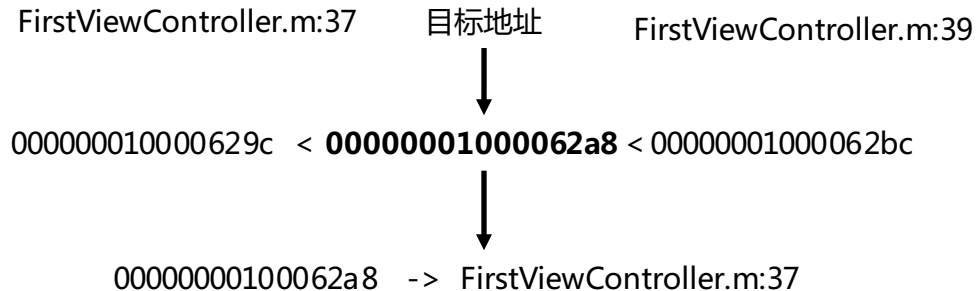
## 1. dSYM文件内容

```
include_directories[ 1] = '/Users/tyzual/code/gitcode/tencent/MTA/MTA_V1/public/MTA-Demo/sdk'
include_directories[ 2] = '/Users/tyzual/code/gitcode/tencent/MTA/MTA_V1/public/MTA-Demo/MTA-Demo'
include_directories[ 3] = '/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS10.3.sdk/usr/include/objc'
-----
Dir Mod Time File Len File Name
-----
file_names[ 1] 1 0x00000000 0x00000000 MTAConfig.h
file_names[ 2] 2 0x00000000 0x00000000 AppDelegate.h
file_names[ 3] 2 0x00000000 0x00000000 AppDelegate.m
file_names[ 4] 3 0x00000000 0x00000000 objc.h
0x00003860: DW_LNE_set_address( 0x00009f5e )
0x00003867: DW_LNS_set_file( 3 )
0x00003869: DW_LNS_advance_line( 23 )
0x0000386b: DW_LNS_copy
0x0000000000009f5e 3 24 0 is_stmt
```

## 2. 解析后地址与文件行号映射关系

地址	文件 : 行号
000000010000626c	FirstViewController.m:29
0000000100006270	FirstViewController.m:36
0000000100006298	FirstViewController.m:37
000000010000629c	FirstViewController.m:37
00000001000062bc	FirstViewController.m:39

## 3. 计算目标地址行号



# 特征堆栈提取

```

NSRangeException
*** -[__NSArray0 objectAtIndex:]: index 23 beyond bounds for empty NSArray
0  CoreFoundation      __exceptionPreprocess
1  libobjc.A.dylib     _objc_exception_throw
2  CoreFoundation     __CFArrayGetCallbacks
3  MTA-Demo            -[MTAEvent toJsonString]
4  MTA-Demo            -[MTACustomEvent encode:]
5  MTA-Demo            -[FirstViewController clickNormaltButton:] (FirstViewController.m:37)
6  UIKit               -[UIApplication sendAction:to:from:forEvent:]
7  UIKit               -[UIControl touchesEnded:withEvent:]
8  UIKit               -[UIWindow sendEvent:]
9  UIKit               -[UIApplication sendEvent:]
10 UIKit               __dispatchPreprocessedEventFromEventQueue
11 CoreFoundation     _CFRunLoopRunSpecific
12 GraphicsServices   _GSEventRunModal
13 UIKit               _UIApplicationMain
14 MTA-Demo            _main(main.m:20)
15 libdyld.dylib      _start
    
```

直接相关性

```

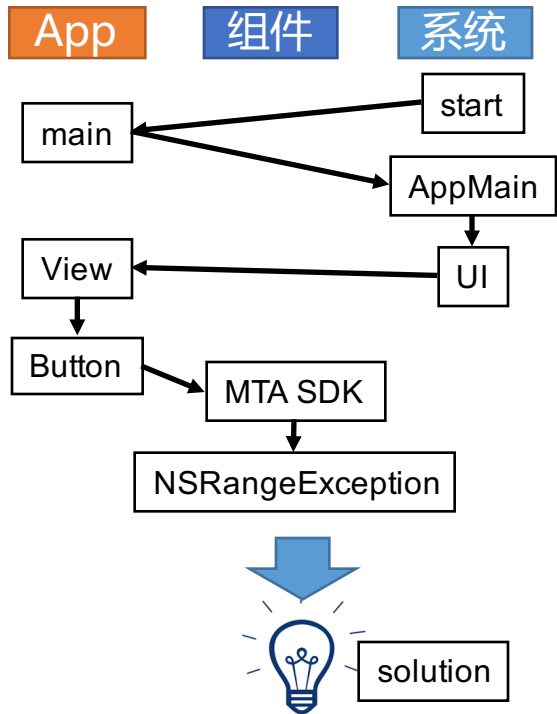
NSRangeException
*** -[__NSArray0 objectAtIndex:]: index 23 beyond bounds for empty NSArray
-[MTAEvent toJsonString]
-[MTACustomEvent encode:]
-[FirstViewController clickNormaltButton:] (FirstViewController.m:37)
_main(main.m:20)
    
```

二次优化

```

NSRangeException
-[MTAEvent toJsonString]
-[MTACustomEvent encode:]
    
```

- 结构化数据
- 分级优化
- 智能路径分析

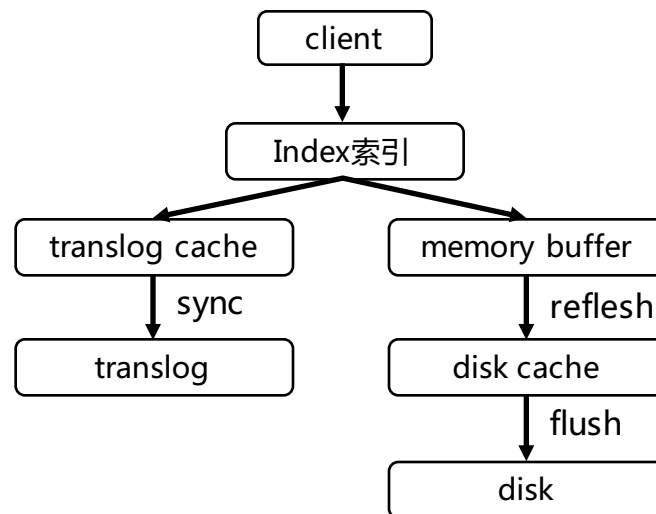
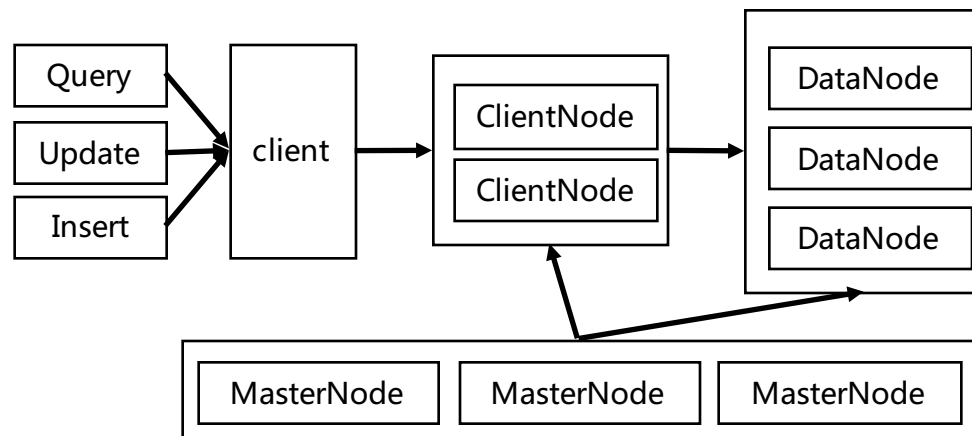




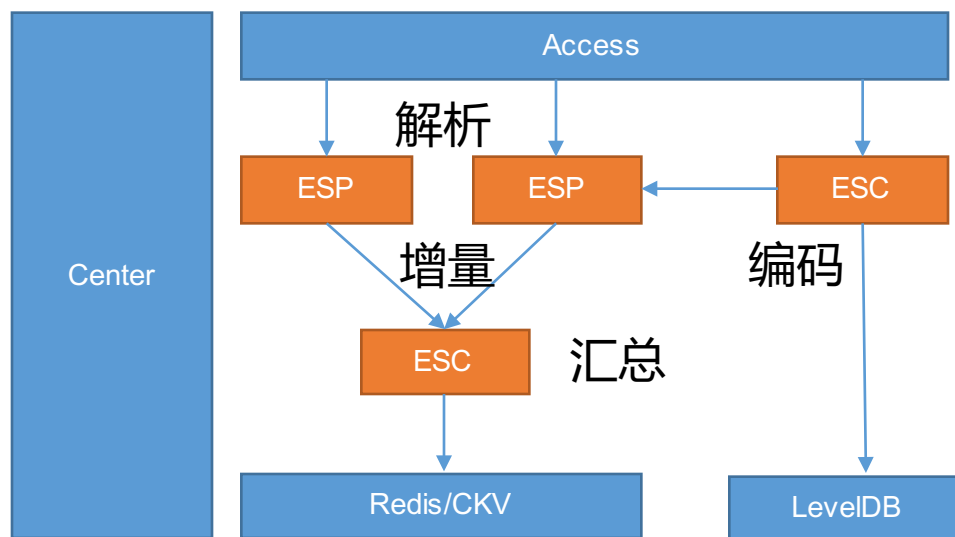
# 实时多维检索--ElasticSearch

- 批量插入/更新
  - cache
- 按时间维度冷热数据分离
  - 3个月之前数据迁移到磁盘
- 按数据维度切割
  - 结构化数据
    - 版本
    - 机型
  - 全文搜索
    - 堆栈
    - 特性堆栈
- 按前台访问LRU分级
  - 天访问 > 周访问 > 月访问
- 按应用规模分库

产品规模	数量(万)	库表数量	刷新时间(秒)
小型	小于10	固定10个	3
中型	10-100	单个App专用	10
大型	超过100	多个, 别名访问	30



# 秒级实时计算系统



特点：

**Access**：协议的解析，数据的清洗及格式化  
**ESP**：Event Streaming Processor，数据按照app、UID重新组织并计算PV、UV等app分析指标  
**ESA**：Event Streaming Aggregator，ESP实时计算后的数据汇总，并将汇总结果写入到存储节点中  
**ESC**：Event Streaming Coder，将系统不可枚举的数据类型编码成为整型，并将对应关系持久化  
**Center**：系统的中心节点，系统配置、数据路由的管理，并承担容灾切换功能

- 增量计算模型：单层单节点内计算小段时间（秒级）的增量，定期发送到下一层节点进行汇总
- 全整数运算：减少字符串处理开销
- 全内存处理：文件持久化支持，保证端到端的消息可靠传输
- 智能容灾策略：本地文件备份、双机热备、一致性hash

# 全内存实时Event处理系统

## ● 数据组织

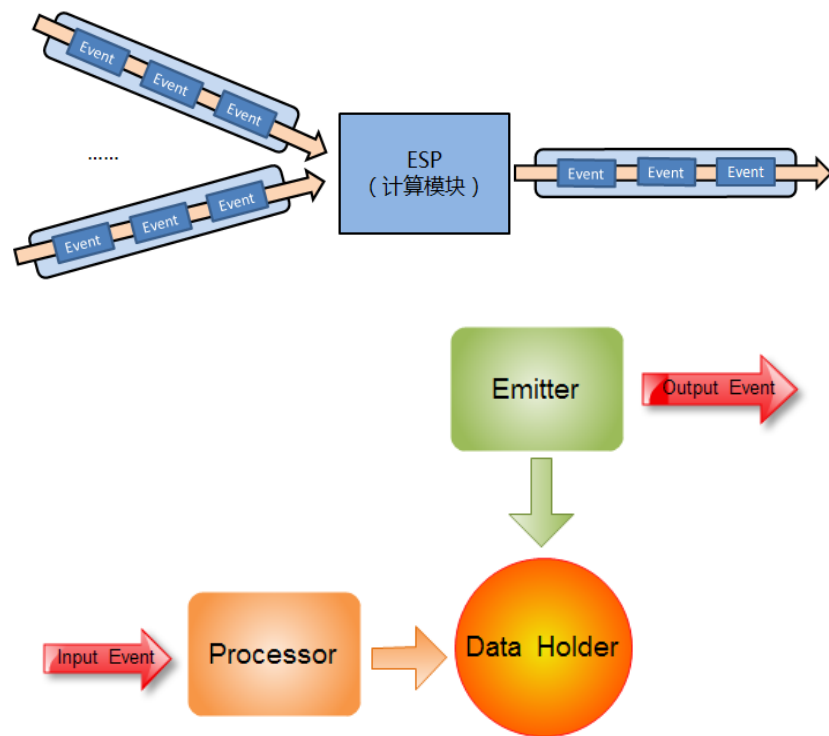
- 万物皆“整型”
- 所有非整型的数据类型通过算法映射为整型

## ● 可扩展的Event结构

- 支持半自动化的序列化/反序列化机制
- 紧凑的二进制编码

## ● 增量计算模型

- Processor：负责具体业务逻辑的计算处理
- Data Holder：负责保存增量结果数据，以及计算依赖的中间状态数据
- Emitter：负责定期输出清空增量计算结果



# 增量计算算法--计算特征堆pv、uv

时间	用户id	特性堆栈
1	user A	RuntimeException XXX
1	user A	NullPointerException XXX
1	user B	NullPointerException XXX
2	user A	SecurityException XXX
2	user A	NullPointerException XXX
2	user B	RuntimeException XXX

编码

ESC状态 (编码)	
非int类型	int类型
user A	1
user B	2
RuntimeException XXX	3
SecurityException XXX	4
NullPointerException XXX	5

输出

时间	用户id	特性堆栈
1	1	3
1	1	5
1	2	5
2	1	4
2	1	5
2	2	3

增量计算

最终输出		
特性堆栈	pv	uv
RuntimeException XXX	2	2
NullPointerException XXX	3	2
SecurityException XXX	1	1

汇总输出

ESC计数状态 (计算汇总)			
时间	特性堆栈	pv	uv
时间=1	3	1	1
	5	2	2
时间=2	周期内保留计数		
	3	2	2
	5	3	2
	4	1	1

累加

ESP计数状态 (计算新增)			
时间	特性堆栈	pv	uv
时间=1	3	1	1
	5	2	2
时间=2	清空计数		
	3	1	1
	5	1	0
	4	1	1

备注：

pv：page view，次数

uv：user view，去重过的用户数

```

set compute_du = 10*60s
set total_pv=total_uv = 0
for sec in compute_du:
    receive err,pv,uv from ESP
    total_pv[err] += pv[err]
    total_uv[err] += uv[err]
end
output: err, total_pv,total_uv
    
```

```

set total_du = 60s, pv=uv=0
for sec in total_du:
    pv[err] += 1
    if userid not in useridSet:
        uv[err] += 1
end
send err,pv,uv to ESA
    
```

# 单机性能优化--海量并发连接

```

Active / Total Objects (% used) : 15078299 / 15185017 (99.3%)
Active / Total Slabs (% used)   : 1437021 / 1437093 (100.0%)
Active / Total Caches (% used)  : 90 / 215 (41.9%)
Active / Total Size (% used)    : 6638525.50K / 6651639.44K (99.8%)
Minimum / Average / Maximum Object : 0.02K / 0.44K / 4096.00K
    
```

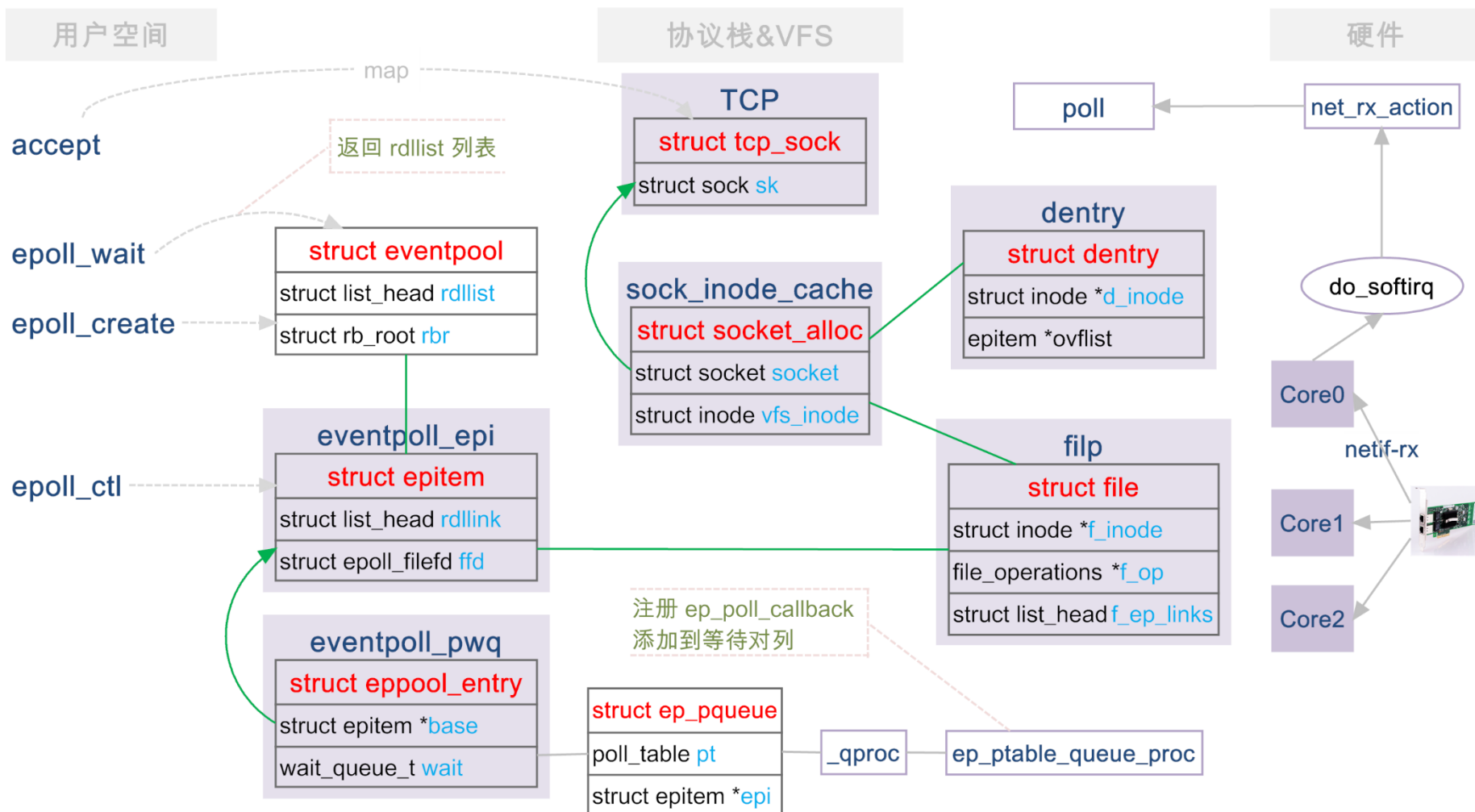
OBJS	ACTIVE	USE	OBJ	SIZE	SLABS	OBJ/SLAB	CACHE	SIZE	NAME
3492460	3491904	99%	0.19K	174623	20	698492K	dentry		
2001840	2001624	99%	0.19K	100092	20	400368K	filp		
2000273	1999995	99%	0.07K	37741	53	150964K	eventpoll_pwq		
2000100	1999995	99%	0.12K	66670	30	266680K	eventpoll_ep_i		
2000095	2000084	99%	0.69K	400019	5	1600076K	sock_inode_cache		
2000020	1999996	99%	1.44K	400004	5	3200032K	TCP		
1490376	1490039	99%	0.63K	248396	6	993584K	proc_inode_cache		

- C1机型：四核3.3Ghz/8GB/500GB
- 单个TCP长连接消耗 **3KB**
- 单机支持 **200W+** 并发连接

- TCP ( 1.6K )
- sock\_inode\_cache ( 0.8K )
- eventpoll\_ep\_i ( 0.13K )
- eventpoll\_pwq ( 0.07K )
- filp ( 0.2K )
- dentry ( 0.2K )

优化：硬件->驱动->系统->协议栈->架构

# 单机性能优化--关键环节



# 单机性能优化--硬件性能挖掘

```
int check_support_sse4_2() {
    int res=0;
    __asm__ __volatile__(
        "movl $1,%eax\n\t"
        "cpuid\n\t"
        "test $0x0100000,%ecx\n\t"
        "jz 1f\n\t"
        "movl $1,%0\n\t"
        "1:\n\t"
        : "=m" (res)
        : "eax", "ebx", "ecx", "edx");
    return res;
}
```

## SSE4.2检测

local_clock	122.000ms	308,660,000
ip_queue_xmit	121.000ms	111,320,000
tcp_transmit_skb	117.000ms	331,430,000
MD::BinUtil::crc32	116.000ms	212,520,000
igb_xmit_frame_ring	111.000ms	427,570,000
sched_clock_cpu	106.000ms	199,870,000
tcp_recvmsg	103.000ms	88,550,000

unsigned int \_mm\_crc32\_u8(...)  
 unsigned int \_mm\_crc32\_u16(...)  
 unsigned int \_mm\_crc32\_u32(...)  
 unsigned \_\_int64 \_mm\_crc32\_u64(...)

## SSE4.2 CRC32支持

_errno_location	0ms	5,060,000
update_cfs_shares	20.000ms	15,180,000
retransmits_timed_out	20.000ms	32,890,000
MD::BinUtil::crc32csse	20.000ms	43,010,000
ipv4_dst_check	20.000ms	43,010,000
ip_output	20.000ms	50,600,000
inode_init_once	20.000ms	22,770,000

Intel SSE4.2 CRC32与常规CRC32性能对比  
 执行 35W 次计算结果 ( SSE 快接近 **6** 倍 )

# 单机性能优化--操作系统--配置部分

配置类型	配置项	值
系统打开文件资源限制	/proc/sys/fs/file-max	2500000
进程打开文件句柄限制	/proc/sys/fs/nr_open	2500000
Epoll监听fd个数限制	/proc/sys/fs/epoll/max_user_watches	2500000
内核分配给TCP内存	/proc/sys/net/ipv4/tcp_mem	754752 1006336 1509504
Listen fd backlog	/proc/sys/net/core/somaxconn	1024
fast recycling of TWsocketsv	/proc/sys/net/ipv4/tcp_tw_recycle	1
Reuse TW socket	/proc/sys/net/ipv4/tcp_tw_reuse	1
拥塞控制算法	/proc/sys/net/ipv4/tcp_westwood	0
Eth scatter-gather	Eth sg	Off
Eth generic-segmentation-offload	Eth gso	Off



# 单机性能优化--协议栈

- 通过 Hook ，同步 sock 异步化改造；
- 协程调度管理，进一步减少上下文切换，提升处理性能。

A：Kernel Socket 函数调用

```
int iSock = socket(PF_INET, SOCK_STREAM, 0);
```

B：mTCP 的函数函数调用

```
int iSock = mtcp_socket(g_pMctx, PF_INET, SOCK_STREAM, 0);
```

C：业务无感知的 mTCP 协议栈改造

```
typedef int (mTcpHookSock*)(int socket_family, int socket_type, int protocol);
int realSocketFunc(int socket_family, int socket_type, int protocol) {
    return mtcp_socket(g_pMctx, socket_family, socket_type, protocol);
}

typedef struct {
    mTcpHookSock    hookSockFunc;
    mTcpHookAccept  hookAcceptFunc;
} stMTCPHooFuncTable;

g_syncHookTable.hookSockFunc = dlsym(RTLD_NEXT, "socket");
```

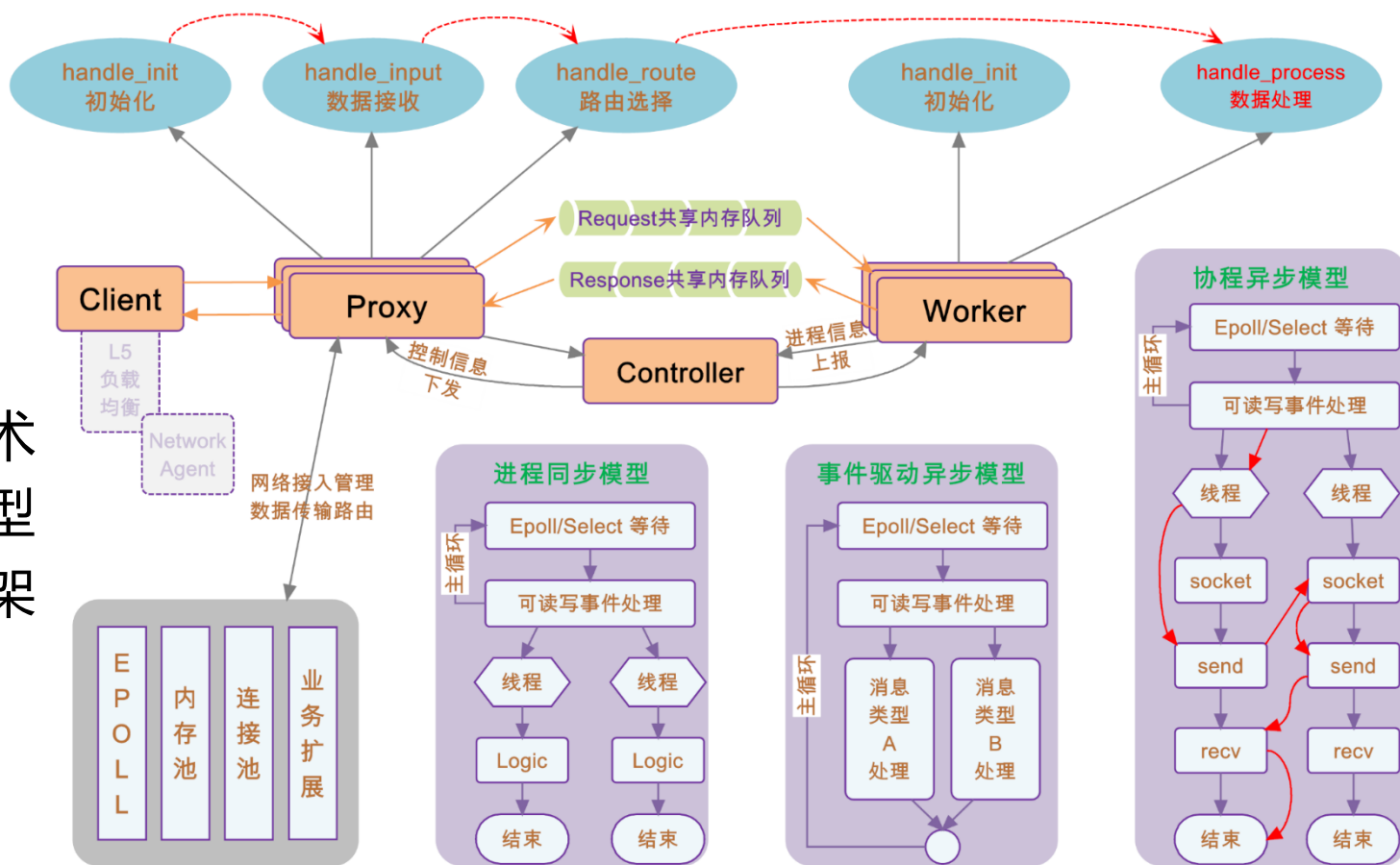
拦截系统调用  
指向 mtcp\_socket WRAP 函数

```
##
# @brief save_context
##
    .text
    .align 4
    .globl save_context
    .type save_context, @function
save_context:
    pop    %rsi
    xorl   %eax,%eax
    movq   %rbx, (%rdi)
    movq   %rsp, 8(%rdi)
    push  %rsi
```

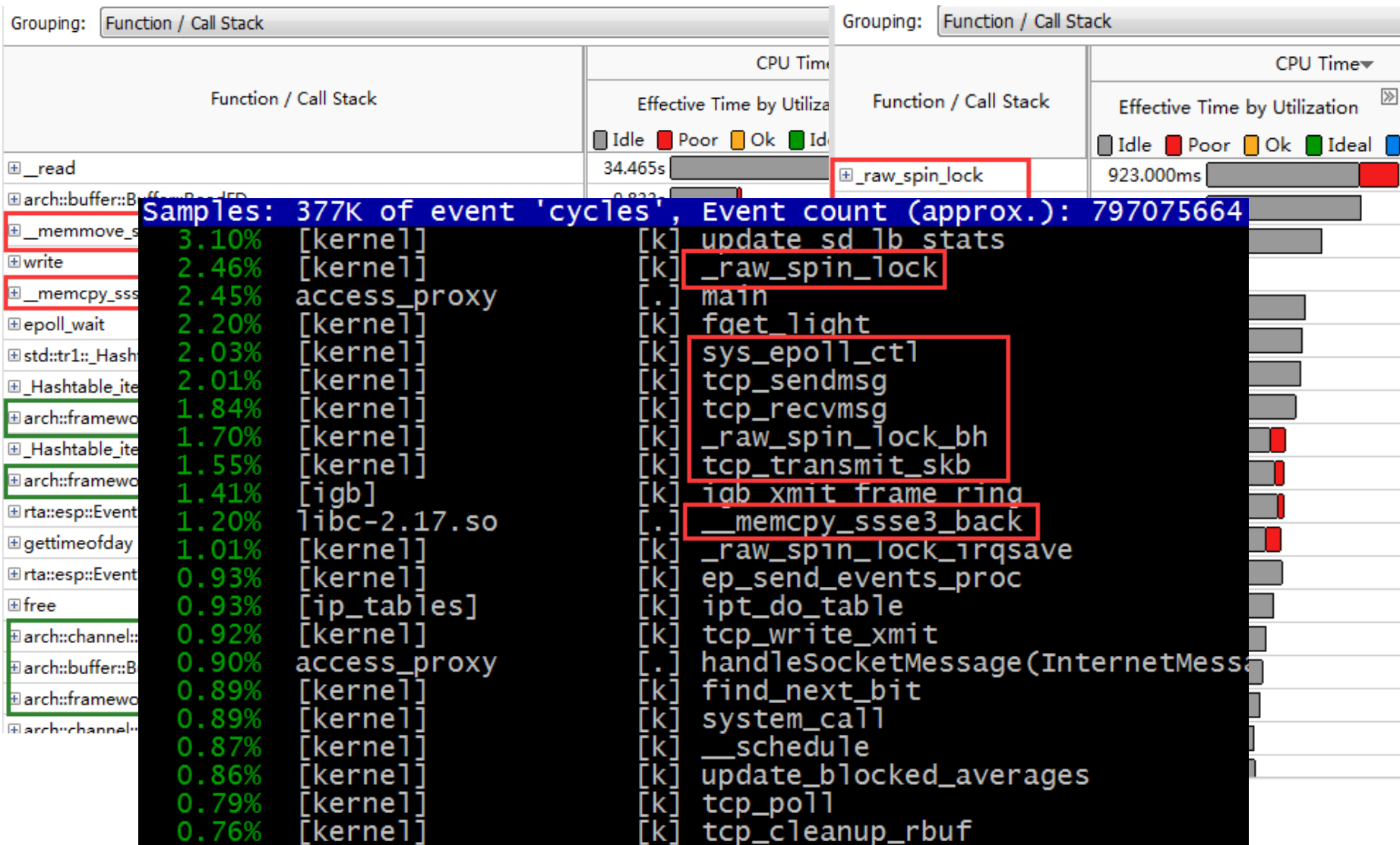
汇编优化

# 单机性能优化--高性能框架

- 池化技术
- 协程模型
- 无锁框架



# 单机性能优化--性能评测



# 实时监控告警

周期轮询

VS

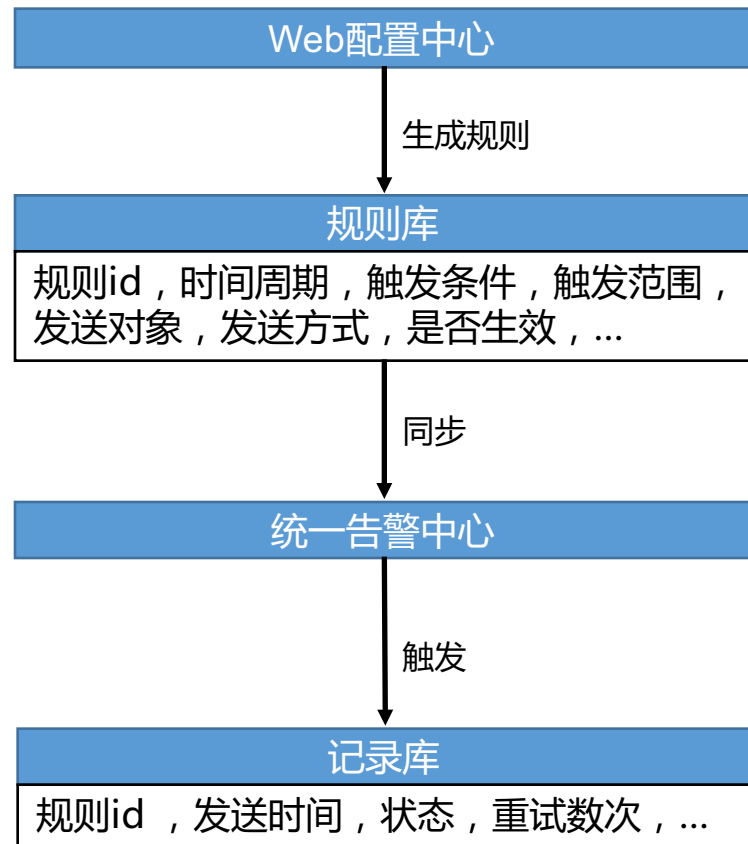
On-Call机制

周期性定期全量查询

- 资源消耗大
- 存在一定延迟
- 适合离线告警、日报

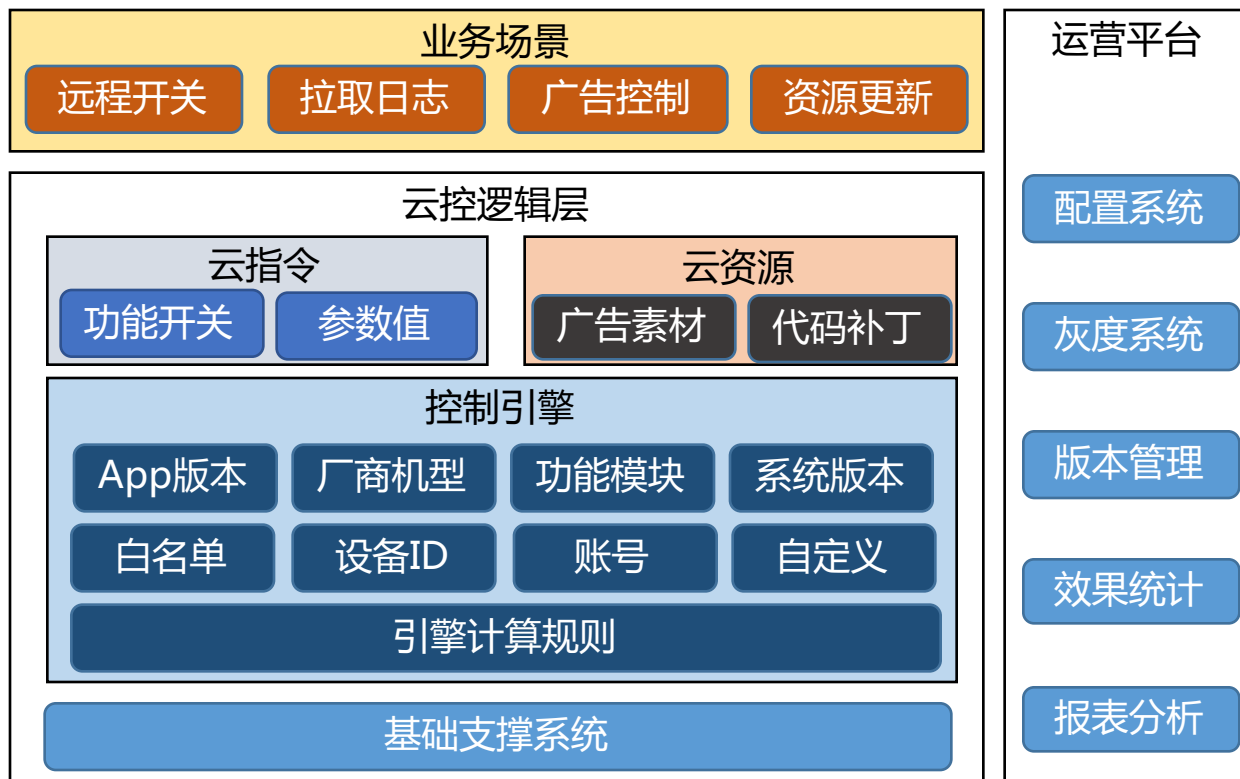
计算入库后检查

- 新结果入库的App
- 入库后检查
- 适合实时告警



# 云控系统--远程助力Crash修复

- 预埋逻辑
- 轻重分离
- 精准控制
- 灵活下发



# 一行代码，一个系统

Android : StatService.startStatService(); iOS : [MTA startWithAppkey:@"xxxx"];

上报详情	基本信息	出错堆栈	跟踪	上报详情	基本信息	出错堆栈	跟踪数据	跟踪日志	环境日志	符号表
------	------	------	----	------	------	------	------	------	------	-----

该应用版本未配置符号表文件，堆栈中的源代码类名、行号

## 基本信息 (最后一条上报记录)

#4 Thread java.lang.NullPointerException
0. java.lang.NullPointerException: Attempt to
1. com.tencent.cpp2.MainActivity\$1.onClick(S
2. android.view.View.performClick(View.java:5
3. android.widget.TextView.performClick(Text
4. android.view.View\$PerformClick.run(View.j;
5. android.os.Handler.handleCallback(Handler
6. android.os.Handler.dispatchMessage(Handl
7. android.os.Looper.loop(Looper.java:158)
8. android.app.ActivityThread.main(ActivityTh
9. java.lang.reflect.Method.invoke(Native Met
10. com.android.internal.os.ZygoteInit\$Metho
11. com.android.internal.os.ZygoteInit.main(Z

上报ID	#4	异常进程# 线程	com.tencent.cpp2#main
用户ID	810588817	发生时间	2017-05-24 18:06:23
包名	com.tencent.cpp2	上报时间	2017-05-24 18:06:23
应用版本	1.0	使用时长	1秒
帐号ID			

## 设备数据

设备机型	SM-N910P	系统版本	Android 6.0, API Level 23
CPU架构		网络APN	WIFI
可用内存大小	1452MB (1452/-1381)	可用存储空间	23972MB (23972/26172)
MTA SDK 版本号	3.0.0		

更多精彩，请访问MTA官网：<http://mta.qq.com>

#142 java.lang.NullPointerException	java.lang.NullPointerException android.view.View.mPrivateFla	APP未捕获异常	未处理	2017-05-31 06:59:5
-------------------------------------	--	----------	-----	--------------------

感谢聆听，更多请关注：腾讯移动分析



技术支持  
[dtsupport@tencent.com](mailto:dtsupport@tencent.com)

商务咨询  
[data@tencent.com](mailto:data@tencent.com)