go语言并发编程实践

以360消息推送系统为例

周洋

部门： 360手机助手

Weibo: @johntech-o

Date: 2015.04.25

目录

在高并发，通信交互复杂，重业务逻辑的分布式系统中，

Go语言优势体现在：开发体验好 、一定量级下服务稳定 、性能满足需要

以360消息推送系统为例

一定量级下服务稳定：

50+内部产品，万款开发平台app
实时长连接数亿量级，日独数十亿量级
1分钟内亿量级广播,日下发峰值百亿量级
400台物理机，9个独立集群，国内外近10个IDC
运维管理的go语言编写的常驻service服务实例接近3000个。

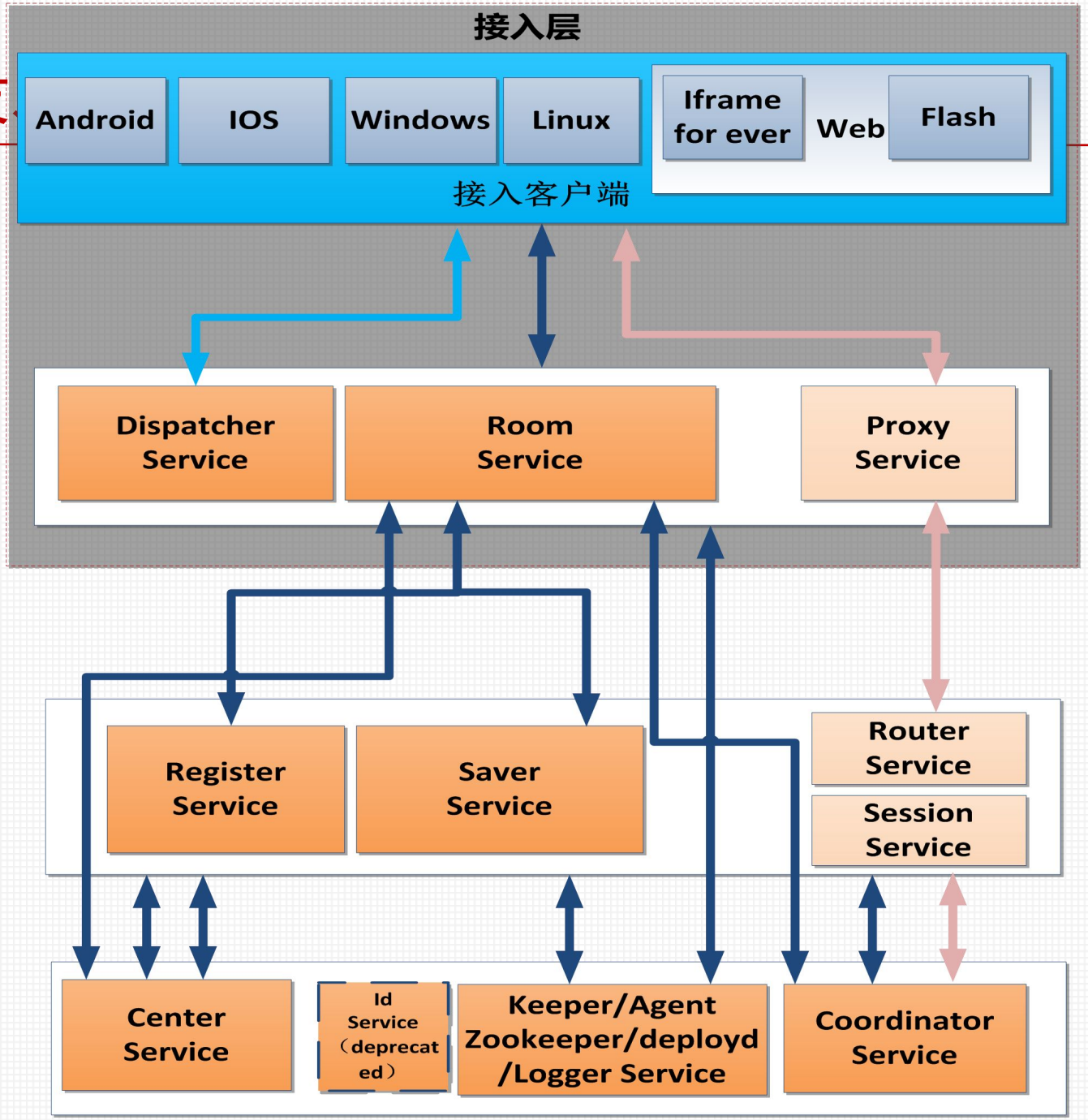
业务场景多样：

支持聊天场景业务，稳定支持多款聊天业务app
单通道多app复用
上行通道，回调支持
对智能硬件产品，提供定制化消息推送与转发服务

性能满足需要：

线上单机最高160w长连接（24核 E5-2630 @ 2.30GHz 64G内存）
qps在2~5w（取决于协议版本，业务逻辑，接入端网络状况）
测试环境，可以通过300w长连接压测（网络,连接稳定,无带宽限制,实际可以更高,决定于广播时候业务内存开销的cpu消耗带来的心跳或者业务延时能否接受）

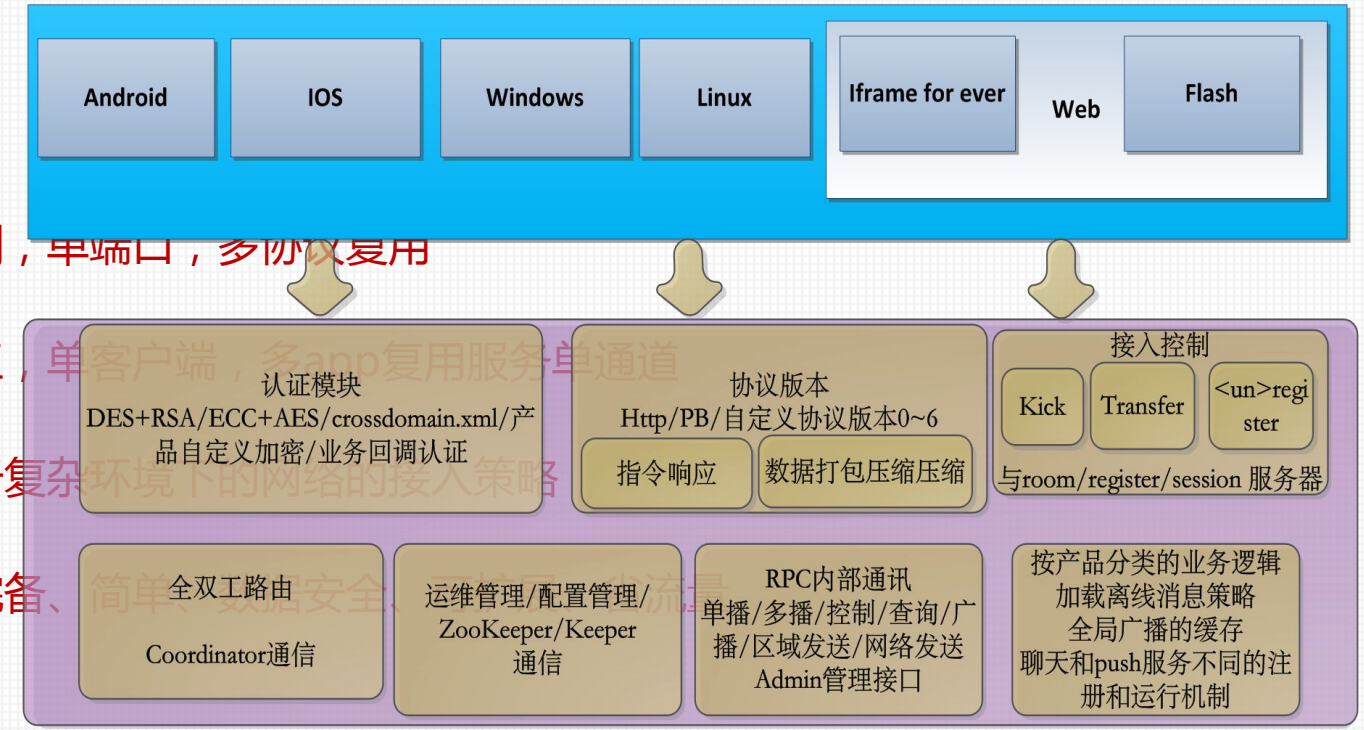
高并发


接入层
Android
IOS
Windows
Linux
Iframe for ever
Web
Flash
接入客户端
Dispatcher Service
Room Service
Proxy Service
Register Service
Saver Service
Router Service
Session Service
Center Service
Id Service（deprecated）
Keeper/Agent Zookeeper/deployd /Logger Service
Coordinator Service

消息系统规模架构：重业务逻辑

消息系统简要架构

接入层



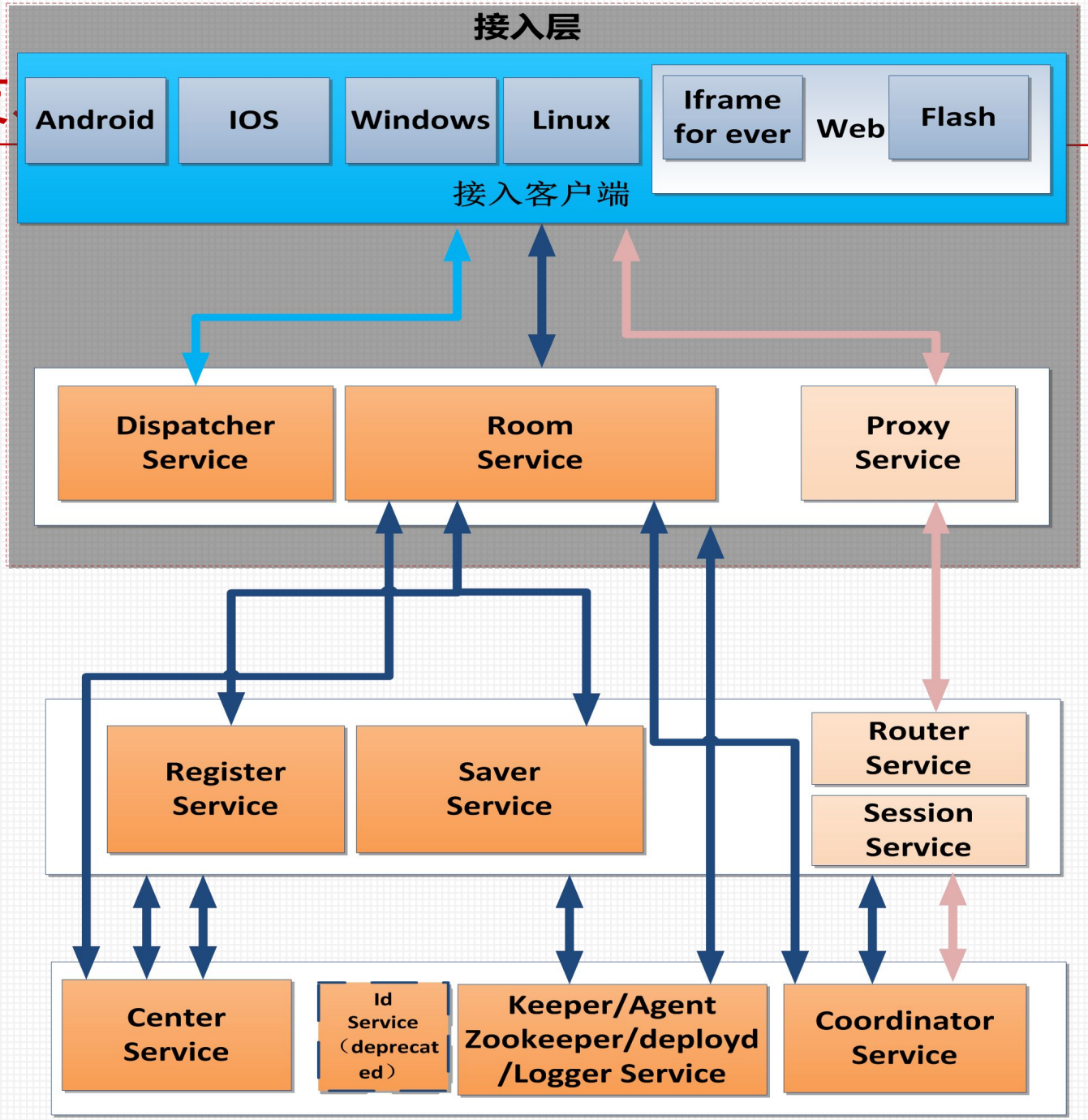
单实例，单端口，多协议复用

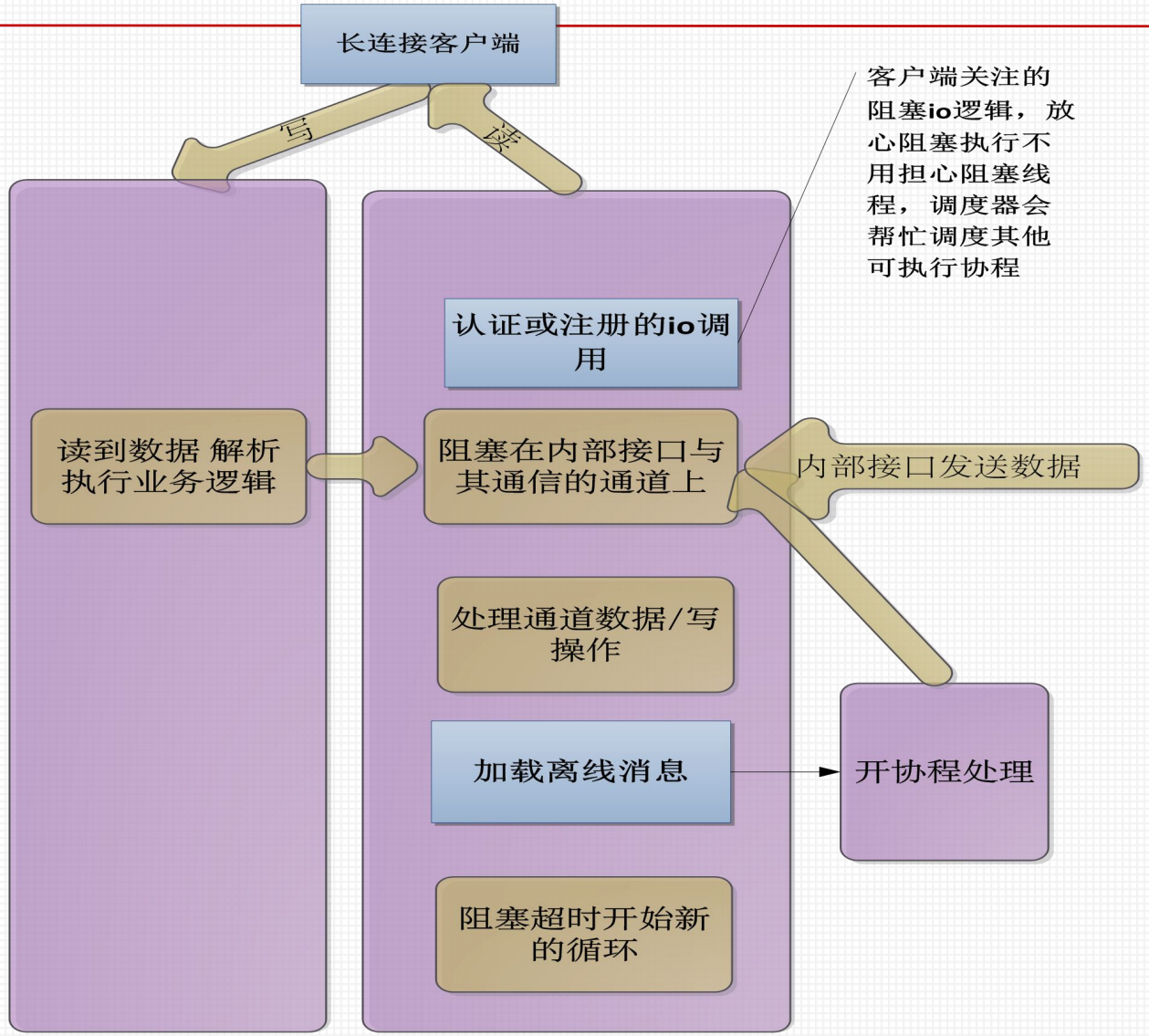
全双工，单客户端，多app复用服务单通道

适用于复杂环境下的网络的接入策略

协议完备、简单、安全、支持同步和流量

高并发


接入层
Android
IOS
Windows
Linux
Iframe for ever
Web
Flash
接入客户端
Dispatcher Service
Room Service
Proxy Service
Register Service
Saver Service
Router Service
Session Service
Center Service
Id Service（deprecated）
Keeper/Agent Zookeeper/deployd /Logger Service
Coordinator Service


长连接客户端
写
读
客户端关注的阻塞io逻辑，放心阻塞执行不用担心阻塞线程，调度器会帮忙调度其他可执行协程
认证或注册的io调用
读到数据 解析执行业务逻辑
阻塞在内部接口与其通信的通道上
内部接口发送数据
处理通道数据/写操作
加载离线消息
开协程处理
阻塞超时开始新的循环

开发体会的对比

Golang开发	C语言开发
按用户来思考问题，按用户数量开协程，对一个用户至少有两个协程为其服务	Oneloop per thread原则，人为控制线程数量，使用epoll+timefd+eventfd来做用户io控制，超时控制，对用户的通知
io是阻塞执行的，直接设置deadline，调度器会对阻塞的协成进行调度，deadline到了，阻塞解除，超时出错	对于所有io操作建立的fd映射到指定loop，同时记录上下文关系进行回调设置，超时控制使用timefd
通过channel与其他用户通信 推荐使用	使用eventfd事件通知的方式，根据epoll获取的fd绑定的回调函数和参数进行回调操作
对外的通信采取阻io或者也可以go出去，不阻塞主循环	对外通信信全局消息list，在映射到的eventloop上开连接池进行消耗

目录

go语言在基础服务开发领域的优势？

我遭遇了哪些挑战？

如何应对的？

具有go特色的运维

问题与瓶颈

问题与瓶颈

10.108.83.24	42.35G	36.27G	Aug31
10.108.83.25	43.55G	39.67G	Aug31
10.108.83.26	42.01G	36.57G	Aug31
10.108.83.27	42.58G	39.62G	Aug31
10.108.83.28	42.15G	38.72G	Aug31
10.108.83.29	42.44G	35.80G	Aug31
10.108.83.30	47.50G	38.14G	Aug31
10.131.81.89	48.16G	42.64G	Aug31

10.102.70.175	29.46G	15.86G	Aug31
10.102.70.176	55.62G	51.63G	Aug31
10.102.70.177	50.59G	33.09G	Aug31
10.102.70.178	51.33G	35.53G	Aug31

bjsc

单机内存占用

高达69G

GC 3~6s

问题与瓶颈

问题与瓶颈

问题与瓶颈

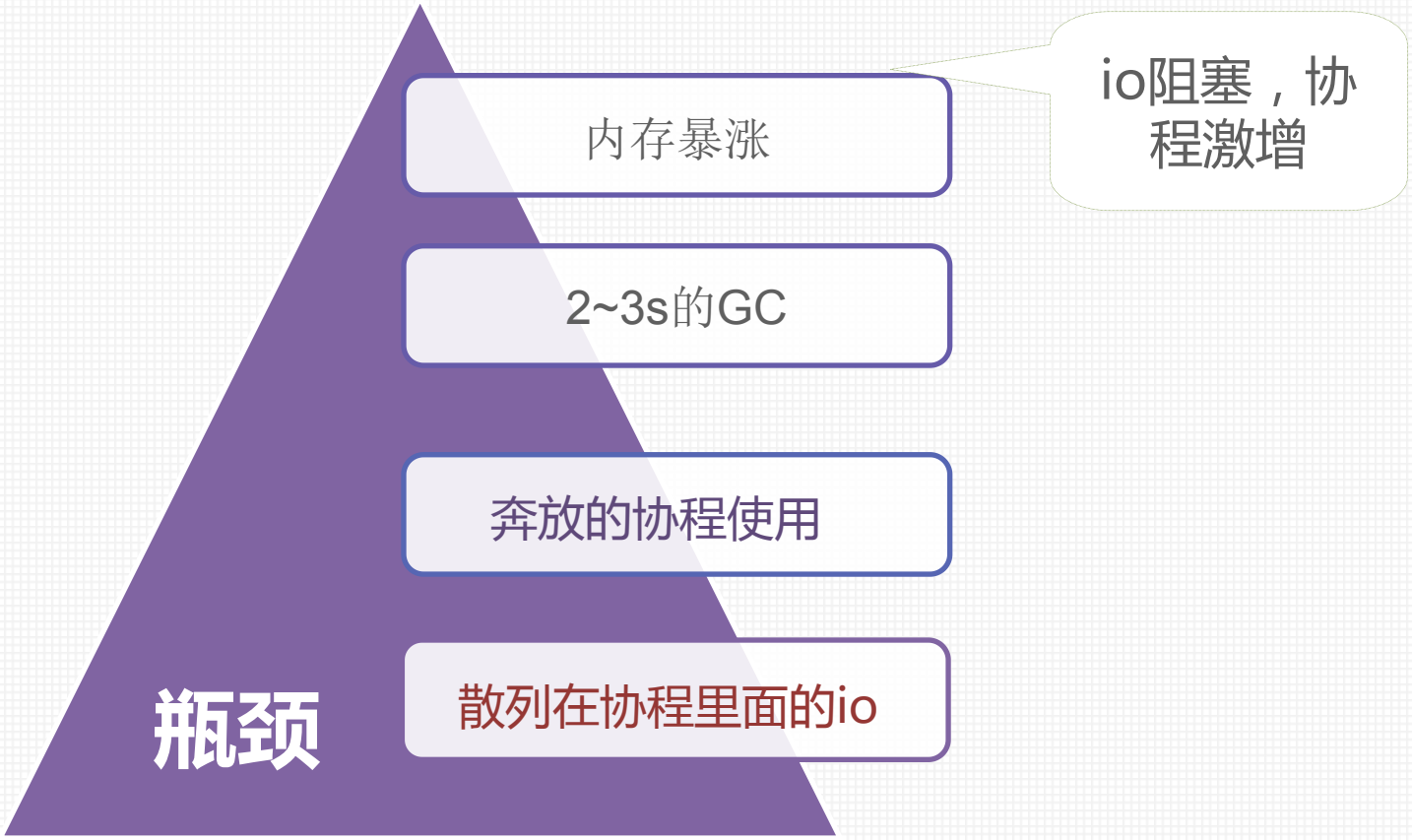
问题与瓶颈

问题与瓶颈

问题与瓶颈

问题与瓶颈

问题与瓶颈



问题与瓶颈

问题与瓶颈

目录

经验一

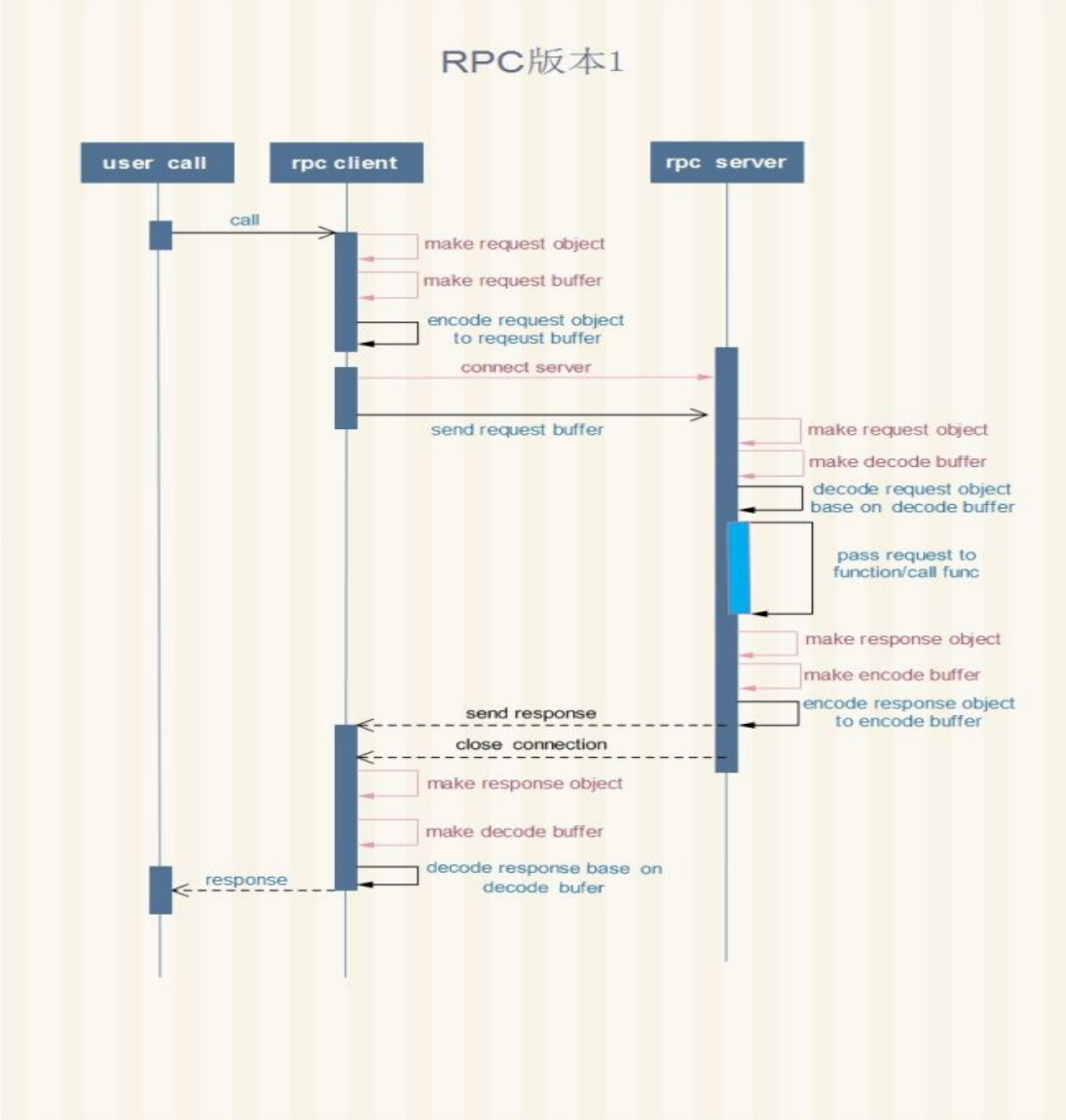
go语言程序开发需要找到一种**平衡**，既利用协程带来的便利性又**做适当集中化处理**

套路：任务池集中数据合并请求、连接池+pipeline 利用全双工特性

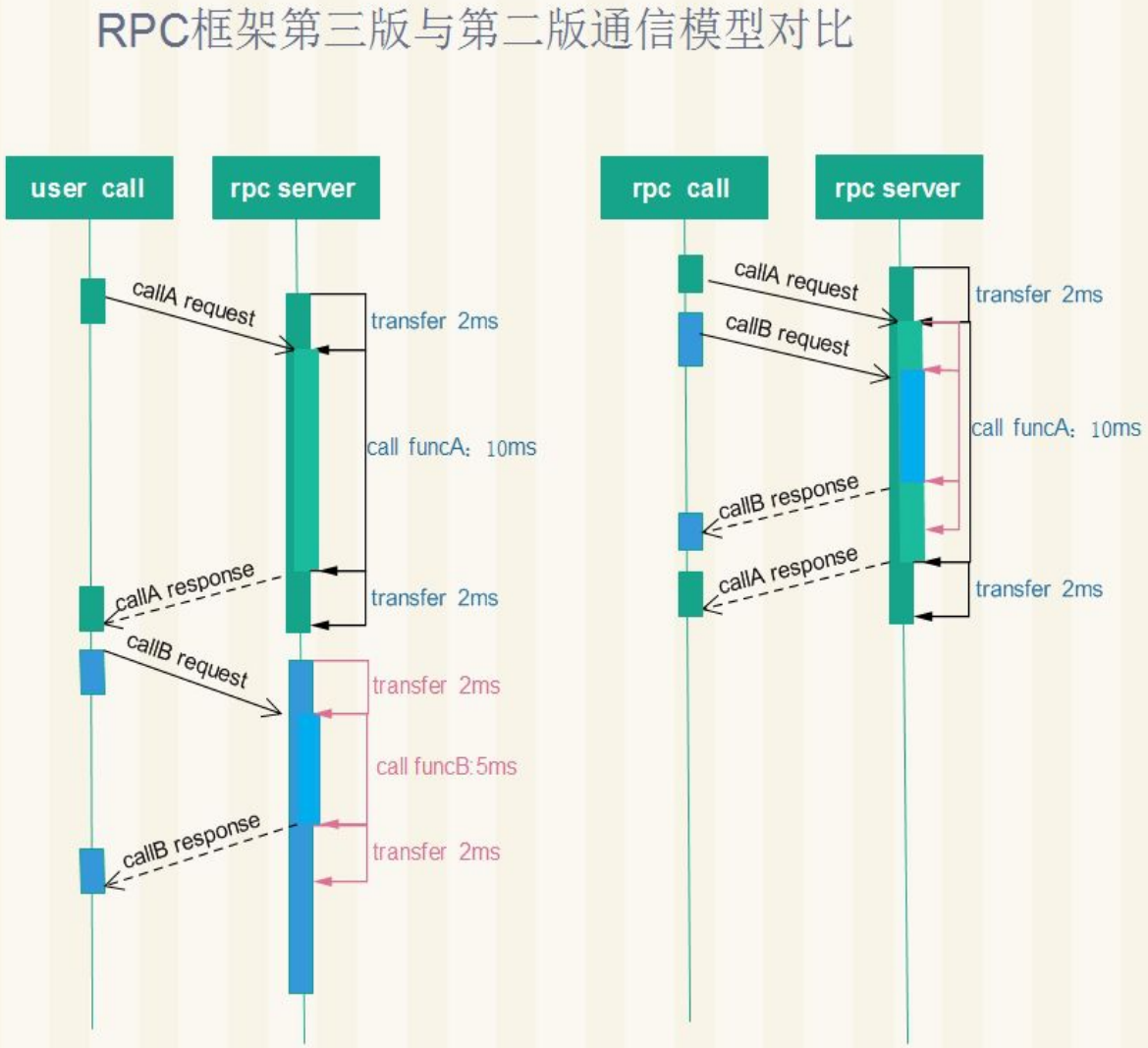
性能优化：io集中处理

性能优化

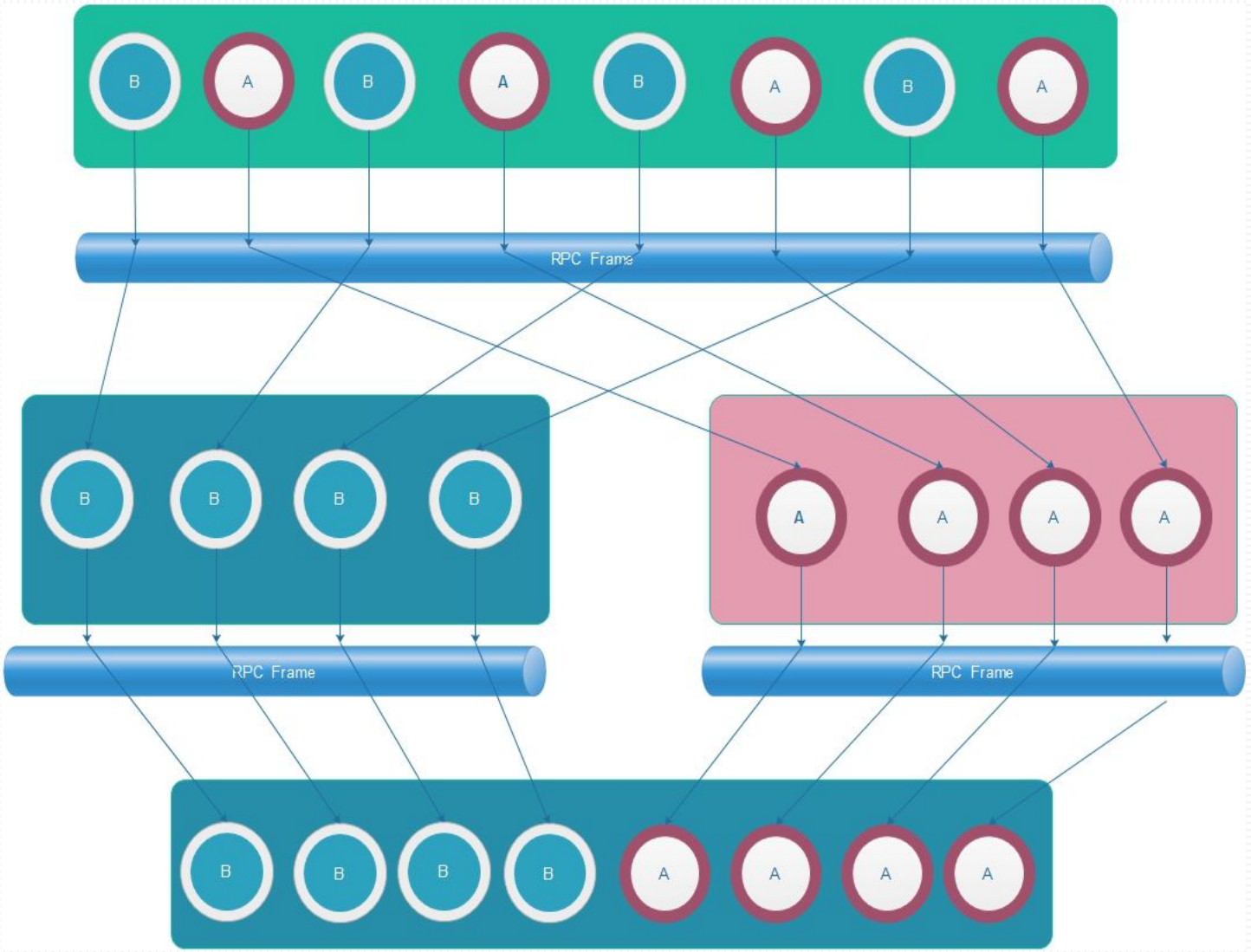
通信库



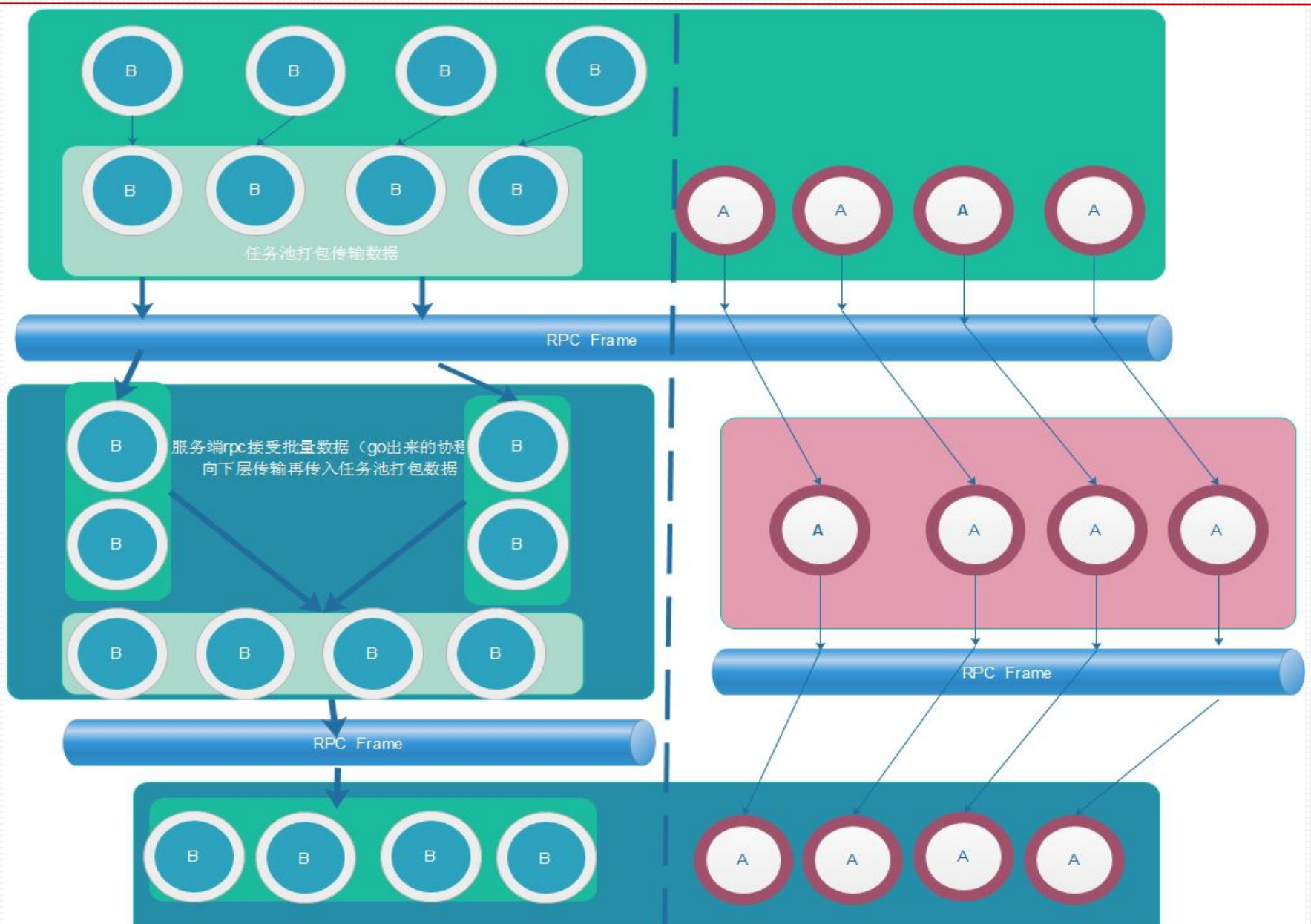
性能优化：io集中处理


RPC框架第三版与第二版通信模型对比
user call
rpc server
callA request
transfer 2ms
call funcA：10ms
callA response
transfer 2ms
callB request
transfer 2ms
call funcB:5ms
callB response
transfer 2ms
rpc call
rpc server
callA request
callB request
transfer 2ms
call funcA：10ms
callB response
callA response
transfer 2ms

性能优化：数据集中处理



性能优化：数据集中处理


B
B
B
B
B
B
B
B
任务池打包传输数据
A
A
A
A
RPC Frame
B
服务端rpc接受批量数据（go出来的协程
向下层传输再传入任务池打包数据
B
B
B
B
B
B
B
B
A
A
A
A
RPC Frame
RPC Frame
B
B
B
B
A
A
A
A

经验二

go语言开发追求开销优化的极限，**谨慎**引入其他语言领域高性能服务的**通用方案**

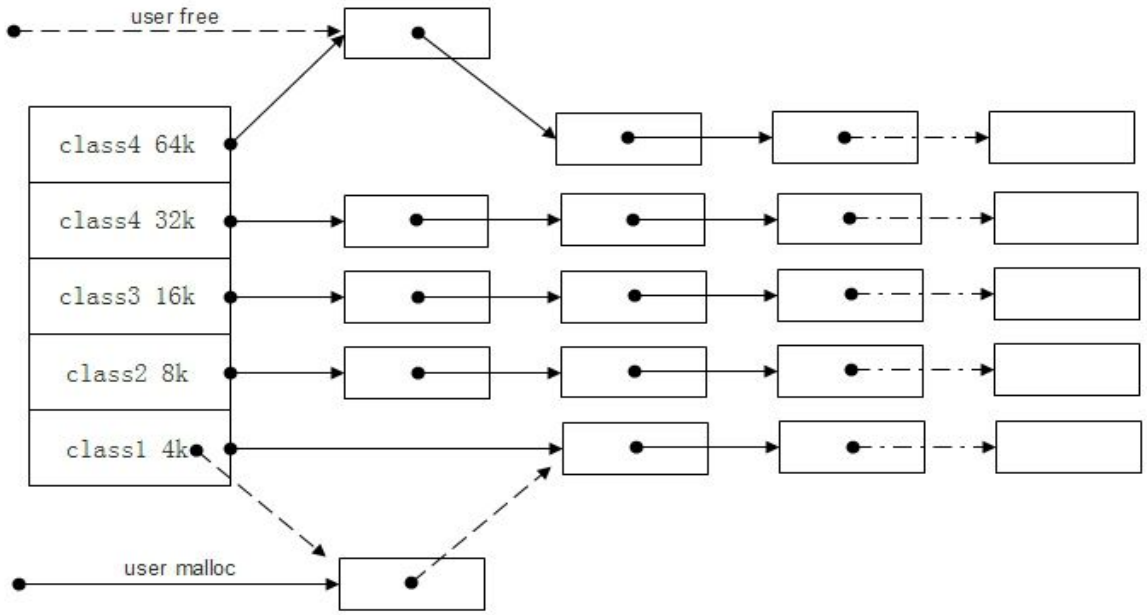
关注：内存池、对象池使用与代码可读性与整体效率的**权衡**

性能优化：通用方案

性能优化

内存池

```
atomic.CompareAndSwapPointer(bufList, buf.next, unsafe.Pointer(buf))
```




```
atomic.CompareAndSwapPointer(bufList, ptr, ((*ElasticBuf)(ptr)).next)
```

性能优化：通用方案

性能优化

对象池

```
// 每条 tcp 连接对应一个 MiopServer3Ext 实例
type MiopServer3Ext struct {
    *MiopServer
    messageChan     chan *UnConfirmedMsgType // 消息chan,发送一条消息,发送一
    quitReadSignal chan bool
    ackChan         chan bool // 读写串行控制chan
}

func newMiopServer3Ext(tcpConnection *network.TcpConnection) *MiopServer3Ex
    return &MiopServer3Ext{
        MiopServer:     newMiopServer(tcpConnection),
        quitReadSignal: make(chan bool, 1),
        messageChan:    make(chan *UnConfirmedMsgType, 1),
        ackChan:        make(chan bool, 1),
    }
}

func (this *MiopServer3Ext) Reset(tcpConnection *network.TcpConnection) {
loop:
    for {
        select {
        case <-this.quitReadSignal:
        case <-this.offlineNotify:
        case <-this.messageChan:
        case <-this.ackChan:
        default:
            break loop
        }
    }
    this.MiopServer.Reset(tcpConnection)
}
```

```
case tcpConnection.IsMiopV5():
    var miopServer5 *MiopServer5
    if miop5Intf := Miop5ObjectPool.Get(); miop5Intf == nil {
        amount := atomic.AddUint64(&miop5ServerAmount, 1)
        if amount%10000 == 0 {
            Logger.Warn(fmt.Sprintf("new miop5 server amount = %v", amount))
        }
        miopServer5 = newMiopServer5(tcpConnection)
    } else {
        amount := atomic.AddUint64(&miop5ReuseAmount, 1)
        if amount%10000 == 0 {
            Logger.Warn(fmt.Sprintf("reuse miop5 server amount = %v", amount))
        }
        miopServer5 = miop5Intf.(*MiopServer5)
        miopServer5.Reset(tcpConnection)
    }
```

目录

go语言在基础服务开发领域的优势？

我遭遇了哪些挑战？

如何应对的？

具有go特色的运维

go语言运维管理方面的独特魅力……

go语言原生提供的各组工具，构建分布式系统**配套设施**方面，提供了**便利**

配套设施= 测试 + 调优 + 监控 + 运维

便利 = 原生profiling工具 + 开协程模拟测试终端+协程协作模拟业务

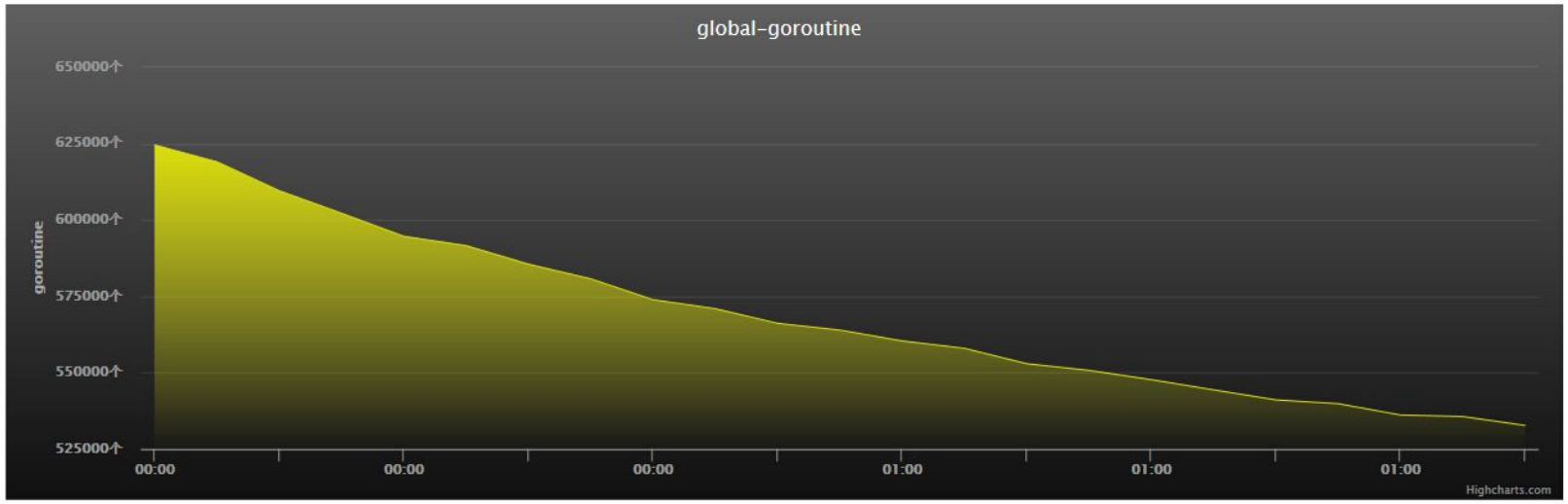
具有go语言特色的运维

global | 360tray | cluster1 | cluster2 | chat | ben | dispatcher | chatroom | ipcam | iot

选择时间［**查询时间跨度请小于8天**］

提交

◉ goroutine ○ pause ○ heapmem ○ heapnum ○ stack ○ alloc ○ malloc ○ mspan ○ mcache ○ buck



最近24小时性能图列表

时间	pprof	pprof	Func	Func	Goroutine	Thread	Block
03-17 00:00	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block
03-16 23:00	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block
03-16 22:00	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block
03-16 21:00	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block
03-16 20:00	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block
03-16 19:00	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block
03-16	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block

最近24小时性能图列表

时间	pprof	pprof	Func	Func	Goroutine	Thread	Block
03-16 12:00	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block
03-16 11:00	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block
03-16 10:00	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block
03-16 09:00	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block
03-16 08:00	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block
03-16 07:00	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block
03-16	Cpu	Heap	CpuFunc	HeapFunc	Goroutine	Thread	Block

具有go语言特色的运维：以项目为例

具有go语言特色的运维

- 可视化平台
  Profiling可视化

  将常规排查问题从手工经验化，变成流程化的过程

  比较不同时间维度，两次上线后，进程的各种状态

  对于优化上线的效果，可评估（上线新功能发现问题与后续确定KPI神器）

  通信库状态可视化

- 最接近业务场景的百万级别压测后台

架构迭代

所有实例组通信数据监控

saver zwt

机器编号	内网IP	端口	空闲	活跃	新建	新建失败	取超时	调用出错	空闲超时	qps	统计时间
888	10.108.102.227	6300	555	1	24599	0	13317	0	16045	3660	2015-04-16 18:54:00
889	10.108.102.228	6300	383	0	24117	15	5235	0	15285	3695	2015-04-16 18:54:00
890	10.108.102.229	6300	549	0	8822	12	13117	0	2315	3765	2015-04-16 18:54:00
891	10.108.102.230	6300	556	0	11771	0	13680	0	2599	3737	2015-04-16 18:54:00
892	10.108.102.231	6300	510	1	9672	0	13092	0	2178	3832	2015-04-16 18:54:00
893	10.108.102.232	6300	530	0	14662	0	13599	18	2397	3784	2015-04-16 18:54:00
1385	10.108.102.234	6300	366	0	10769	0	4915	0	2540	3742	2015-04-16 18:54:00

机器编号	内网IP	端口	idle	working	creating	read_amount	统计时间
1192	10.108.102.112	6020	214	0	0	28894941	2015-04-16 20:26:00
1186	10.108.102.91	6020	206	0	0	28688755	2015-04-16 20:26:00
1187	10.108.102.92	6020	207	0	0	28598266	2015-04-16 20:26:00
1188	10.108.102.93	6020	208	0	0	28810773	2015-04-16 20:26:00
1189	10.108.102.94	6020	207	0	0	28774618	2015-04-16 20:26:00
1190	10.108.102.95	6020	201	0	0	28537627	2015-04-16 20:26:00

未完成任务

任务ID	压测名	压测机器数量	添加任务时间	任务参数	执行状态	取消

所有任务

任务ID	压测名	压测机器数量	添加任务时间	任务参数	执行状态
11	ben_miop	9	2014-11-11 10:02:27	{"-g":"5000","-n":"100","-u":"z@zhuomian","-i":"10.108.87.71","sleep":"0","interval":1}	执行成功
10	ben_miop	9	2014-11-10 17:10:48	{"-g":"10000","-n":"10","-p":"6100","-t":"5","-u":"z@zhuomian","-i":"10.108.87.71","sleep":"1","interval":1}	执行成功
9	ben_miop	10	2014-08-20 10:16:52	{"-u":"qa@zhuomian","-g":"10000","-n":"10","-t":"100","-i":"10.108.76.57","sleep":"1","interval":2}	执行成功
8	ben_miop	10	2014-08-19 11:21:23	{"-v":"3","-g":"10000","-u":"qa@zhuomian","-i":"10.108.87.71"}	执行成功
7	ben_miop	2	2014-08-18 18:56:56	{"-u":"zzz@zhuomian","-g":"3000","-n":"5","-i":"10.108.87.71"}	执行成功

所有任务执行结果

任务ID	执行机器ip	任务开始时间	任务结束时间	状态码	状态描述
11	10.121.97.231	2014-11-11 10:03:02	2014-11-11 10:06:42	4	Cmd was killed
11	10.121.97.230	2014-11-11 10:03:02	2014-11-11 10:06:42	4	Cmd was killed
11	10.121.97.229	2014-11-11 10:03:02	2014-11-11 10:06:42	4	Cmd was killed
11	10.108.102.232	2014-11-11 10:03:02	2014-11-11 10:06:42	4	Cmd was killed
11	10.108.102.231	2014-11-11 10:03:02	2014-11-11 10:06:42	4	Cmd was killed
11	10.108.102.230	2014-11-11 10:03:02	2014-11-11 10:06:42	4	Cmd was killed
11	10.108.102.229	2014-11-11 10:03:02	2014-11-11 10:06:42	4	Cmd was killed
11	10.108.102.228	2014-11-11 10:03:02	2014-11-11 10:06:42	4	Cmd was killed
11	10.108.102.227	2014-11-11 10:03:02	2014-11-11 10:06:42	4	Cmd was killed
10	10.121.97.231	2014-11-10 17:11:11	2014-11-10 17:13:17	4	Cmd was killed
10	10.121.97.230	2014-11-10 17:11:10	2014-11-10 17:13:16	2	Cmd is finished successfully
10	10.121.97.229	2014-11-10 17:11:09	2014-11-10 17:13:15	2	Cmd is finished successfully
10	10.108.102.232	2014-11-10 17:11:08	2014-11-10 17:13:13	2	Cmd is finished successfully

具有go语言特色的运维： 配置管理

push

拆分多实例

- 缓解GC压力（gc时间减少40%）

按业务类型聚类，广播（io密集），多播，点对点（内部通信密集），聊天室（cpu密集）

分层服务，按层次扩展改为分集群（Set/Cell思想），各自独立，又具备全被全部功能子集群

- 按业务拆分（助手，卫士，浏览器）
- 按功能拆分（push，聊天，嵌入式产品）
- 按IDC拆分 （zwt，bjsc，bjdt，bjcc，shgt，shjc，shhm，Amazon Singapore）

拆解后带来管理成本，引入(zookeeper + deployd) / (Keeper + Agent) 对各节点进行管理

- 监控集群
- 控制组件行为（用户重定向）
- 连接监控

具有go语言特色的运维

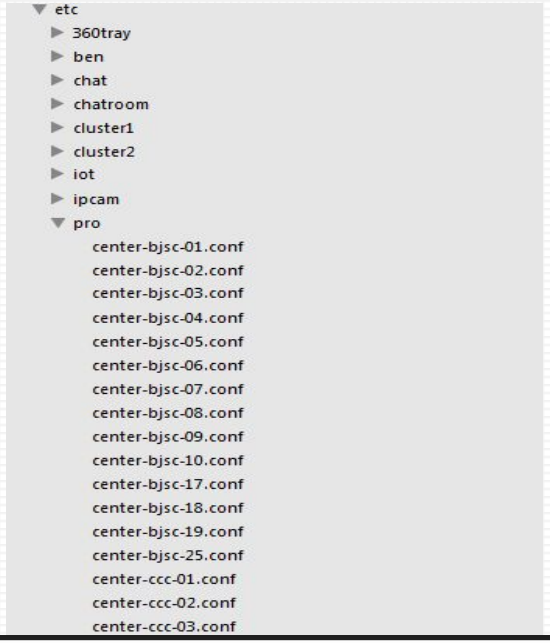
- 配置文件管理与监控

后台->生成配置文件->全部服务器->调用deployd接口，重启或者reload

zookeeper-> 动态修改配置文件 -> 各个服务器上实例通过sdk订阅

profiling 接口 —> 后台定期获取->调用go pprof生成文件->存储

通信库实时数据http接口-> 后台定期获取-> 进入数据库，展示



360tray | cluster1 | global | ben | cluster2 | chat | chatroom | ipcam | weishi

节点ID	第二节点	第三节点	当前版本号	节点内容	状态	操作 新增+
17	global.conf	products	6	zhuomian,360tray,autoTest,autoTest2,autoTest3,10,11,12,13,123,456,789,14,zhuomia...	上线	设置
18	global.conf	sourcecode	1	leshow:11,openapi:22...	上线	设置
19	global.conf	tabletnameproduct	3	wuxianbrowser:user_info_0,gameunion:user_info_1,zhuomian:user_info_2,autoTest:us...	上线	设置
77	global.conf	blacklistips	6	10.16.26.91...	上线	设置
115	global.conf	counterproducts	5	zhuomian,mobilesafe,30001...	上线	设置

具有go语言特色的运维

```
[Global]
debug                       = 0
#qucmsec是用户中心,m_gamehall游戏大厅
products                    = zhuomian,autoTest,autoTest2,autoTest3,123,456,789,10,11,12,13,14,openapi
area_broadcast_products     = zhuomian
```

局配置文件

- 客户端sdk->集成profiling功能-> keeper调用并存储

```
package data

import (
    "sync/atomic"
    "unsafe"
)

type Global struct {
    Key1     string
    Debug    int
    Products []string
}

var GlobalPtr unsafe.Pointer = unsafe.Pointer(&Global{})

func CurrentGlobal() *Global {
    return (*Global)(atomic.LoadPointer(&GlobalPtr))
}

func UpdateGlobal(v *Global) {
    atomic.StorePointer(&GlobalPtr, unsafe.Pointer(v))
}
```

库信息） -> keeper调用并存储

态信息，按时间和需求获取其他富信

通过agent调用命令行

架构迭代

caseid	用例名称	用例描述	测试结果	用例日志	comment	Cost
01	单播-离线补偿_0	360tray global cluster1 测试协议3 cluster2 测试协议5	PASS	log		143
02	广播离线补偿_0	360tray global cluster1 测试协议3 cluster2 测试协议5	PASS	log		164
03	发送二进制空消息_0	单播 1.用户建立长连接检查状态为在线 2.用单播接口发送在线消息（消息体为空字符串） 3.检查用户是否收到消息(预期收不到)	PASS	log		18
04	指定用户离线-发送离线消息_0	单播 1.用户建立长连接检查状态为在线 2.断开用户连接检查状态为离线 3.用单播接口发送离线消息 4.用户重新登录连接到之前room 5.检查用户是否收到消息(预期能收到)	PASS	log		33
05	单播-指定用户离线-发送离线消息(重连随机room)_0	单播 1.用户建立长连接检查状态为在线 2.断开用户连接检查状态为离线 3.用单播接口发送离线消息 4.用户重新登录随机选择room进行连接 5.检查用户是否收到消息(预期能收到)	PASS	log		33
06	单播-指定用户离线-发送在线消息_0	单播 1.用户建立长连接检查状态为在线 2.断开用户连接检查状态为离线 3.用单播接口发送在线消息 4.用户重新登录连接到之前room 5.检查用户是否收到消息(预期收不到)	PASS	log		47
07	单播-指定用户离线-发送在线消息(重连随机room)_0	单播 1.用户建立长连接检查状态为在线 2.断开用户连接检查状态为离线 3.用单播接口发送在线消息 4.用户重新登录随机选择room进行连接 5.检查用户是否收到消息(预期收不到)	PASS	log		45
08	单播-指定用户在线-发送在线消息_0	单播 1.用户建立长连接检查状态为在线 2.用单播接口发送在线消息 4.用3.检查用户是否收到消息(预期能收到)	PASS	log		53

总结回顾

go语言在基于**并发协作**的，**重业务逻辑**的基础服务方向非常**适用**

适用 = 开发体验好 + 服务稳定 + 性能满足需要

go语言程序开发需要找到一种平衡，既利用协程带来的便利性又做适当**集中化处理**

套路 = 按请求和业务逻辑并行+任务池集中数据合并请求 + 连接池集中收发

go语言开发追求开销优化的极限，谨慎引入其他语言领域高性能服务的**通用方案**

内存池+对象池使用 与 代码可读性与整体效率的权衡

go语言原生提供的各组工具，构建分布式系统**配套设施**方面，提供了**便利**

生态圈 = 测试 + 调优 + 监控 + 运维

便利 = 原生profiling工具 + 通信库集成监控+协程协作模拟业务压测

谢 谢！