十年架构 成长之路

SACC 第十届中国系统架构师大会

SYSTEM ARCHITECT CONFERENCE CHINA 2018

2018年10月17-10月21日 北京海淀永泰福朋喜来登酒店

IT168.com ChinaUnix ITPUB

微信后台架构与基础设施简介

许家滔 / 微信技术架构部

sunnyxu@tencent.com

回顾微信发展历程

2011.1.21
即时消息
照片分享

2011.10
摇一摇
漂流瓶

2012.7
视频聊天

2014.10
小视频

连接人
连接设备
连接服务

1.0 2.0 3.0 4.0 4.2 5.0 6.0 Now

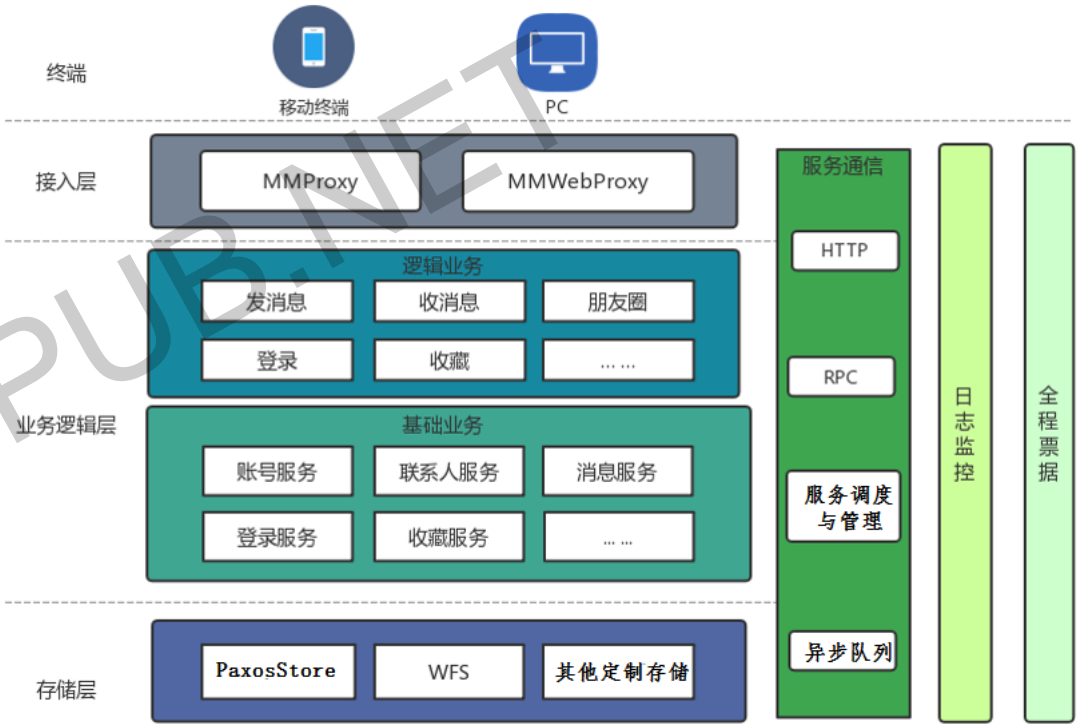
2011.5
语音对讲
查看附近的人

2012.4
朋友圈

2013.8
游戏中心
表情商店
微信支付

微信后台系统架构

- 接入层
  - mmproxy
  - mmwebproxy
- 业务逻辑层
  - 逻辑业务（Logicsvr）
  - 基础业务
- 存储层
  - PaxosStore
    (table,value,set,queue,etc.)
  - WFS
  - Media



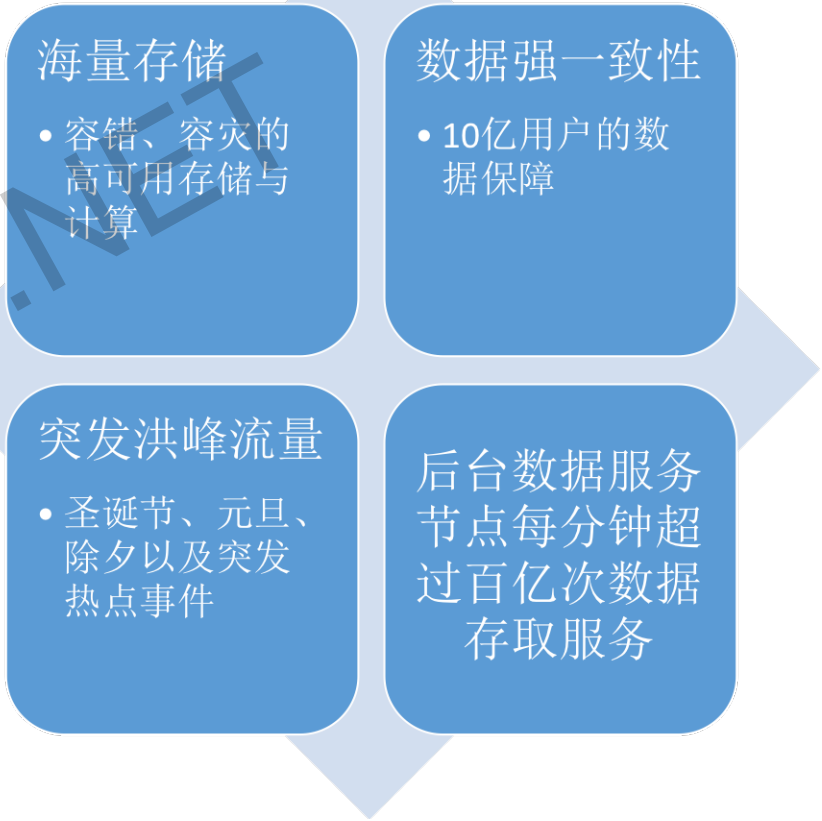
高可用与敏捷开发

- 高可用的关键技术
  - 数据强一致
  - 微服务框架
- 敏捷开发的烦恼
  - 内存泄漏
  - Coredump
  - …

数据存储与微服务框架

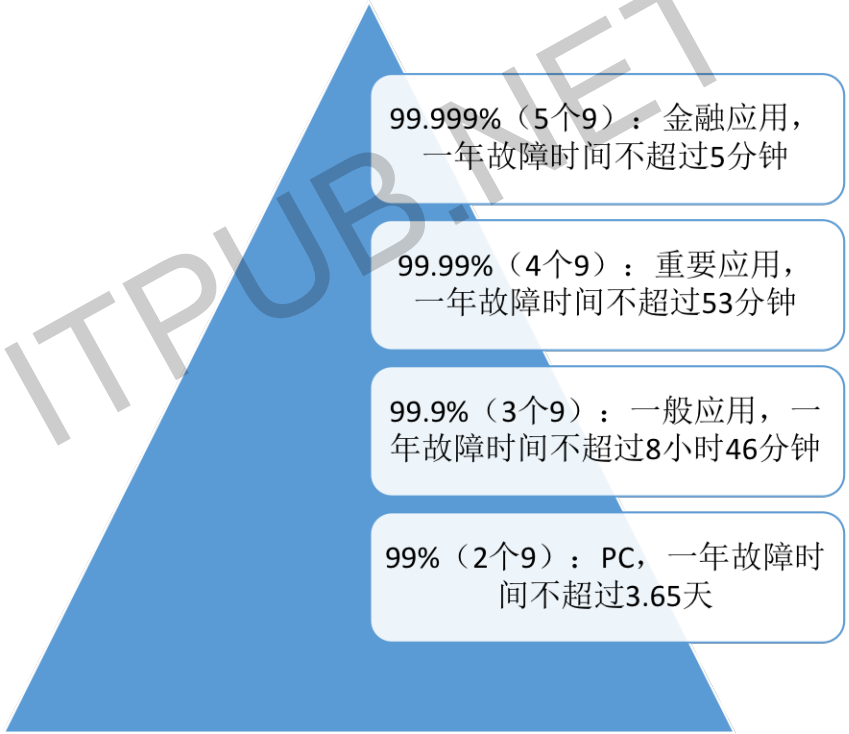
数据业务背景与挑战

- 十亿级用户、百亿千亿级服务
- 微信用户数：10亿
- 微信消息数：1000+亿/日
- 朋友圈：10+亿发表和点赞/日，100+亿浏览/日
- 开放平台，微信支付等业务活跃度持续增长



目标

- 可用性（正常工作时间/（正常工作时间+故障时间））



挑战

•机器故障是常态，微信希望提供连续无间断的服务能力

–业界数据可用性通常通过RAFT，Paxos租约等来实现数据复制

–机器故障的时候系统会进入等待租约过期并重新选主的状态，也即会产生30秒级别的服务中断，我们希望能够规避。

•相对于传统的基于故障转移的系统设计，我们需要构建一个多主同时服务的系统

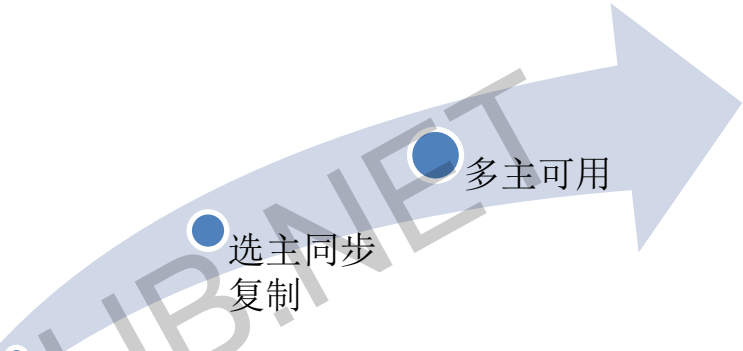
–系统始终在多个数据中心中运行

–数据中心之间自适应地移动负载

–透明地处理不同规模的中断（主机，机房，城市园区等）

关键设计

- **基于故障切换的系统**
  - **–Raft**
  - –基于租约Paxos Log
  - –“Paxos make live”
  - –其他例如bigtable等等
- **基于多主的系统**
  - –基于非租约Paxos Log
  - –Google MegaStore
  - –微信PaxosStore



- **多主系统相关难点**
  - –Paxos Log管理，对存储引擎的设计要求
  - –代码假设在一个cas环境中运行
  - –服务随时可用，对cache，热点的处理
  - –追流水/恢复流程的时效性要求

多主系统的收益

- 7 X 24
  - –应对计划内与计划外的停机维护
  - –不再有尘封已久的切换流程
    - •由于多主可用，所以类似快照，数据对齐等行为已经在在线核心逻辑中充分体现
  - –变更发布

- •资源调度
  - –流量控制和逻辑层类似简单

- •成本
  - –数据中心之间动态共享工作以平衡负载

设计与实践

同步复制

- 在数据分区内部实现完整ACID语义，维护细粒度的海量数据分区
- 每一次数据读写都基于非租约的Paxos交互实现，每分钟超过百亿次

存储引擎

- 针对微信丰富的业务场景，设计多种高性能的存储模型
- 支持单表过亿行的二维表格/FIFO队列/key-value等等数据结构

负载均衡

- 数据节点动态计算负载能力尽力服务，超过服务能力部分自动转移到其他复制节点

实际成果

•万级服务器

•核心数据存储服务可用性达到6个9
- 99.9%（3个9）：一般应用，一年故障时间不超过8小时46分钟
- 99.99%（4个9）：重要应用，一年故障时间不超过53分钟
- 99.999%（5个9）：金融应用，一年故障时间不超过5分钟

•经验输出
- 数据库会议VLDB 2017

“**PaxosStore**: High-availability Storage Made Practical in WeChat”
http://www.vldb.org/pvldb/vol10/p1730-lin.pdf

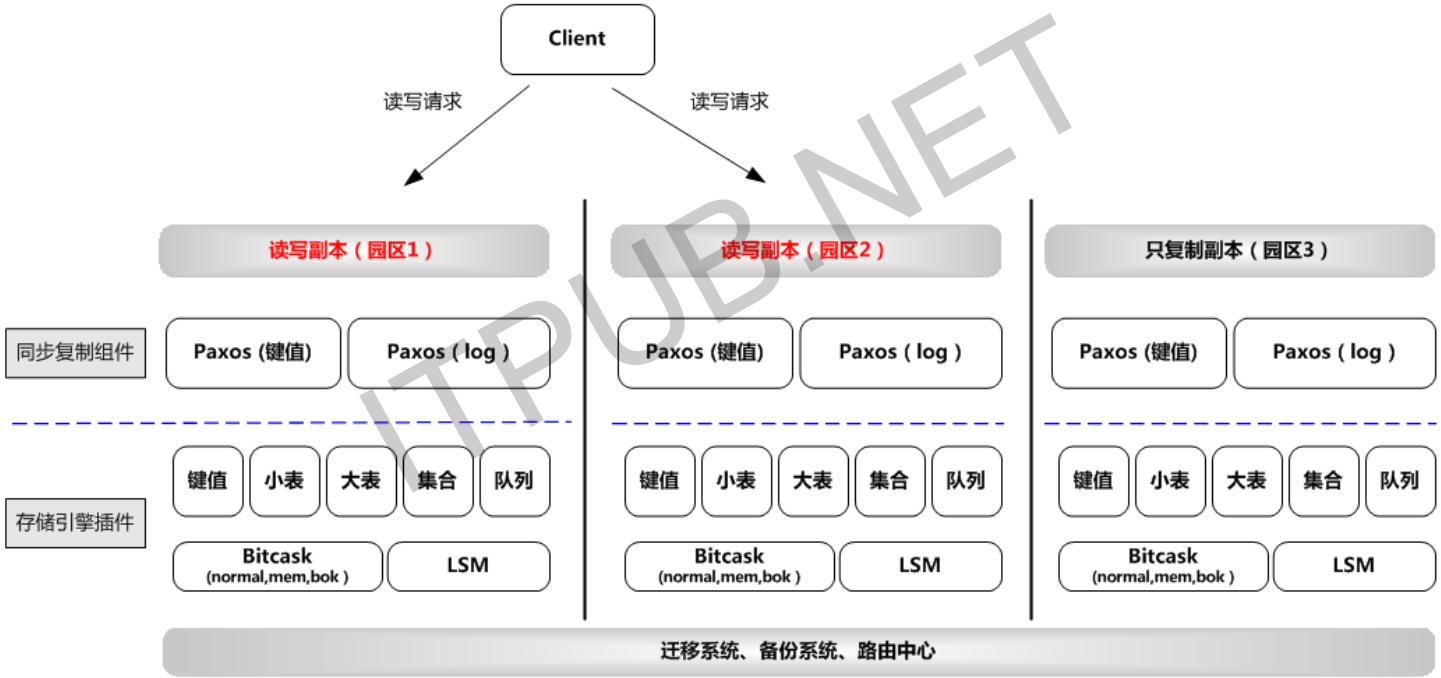
- 论文相关示例代码开源

github.com/tencent/paxosstore

PaxosStore广泛支持微信在线应用

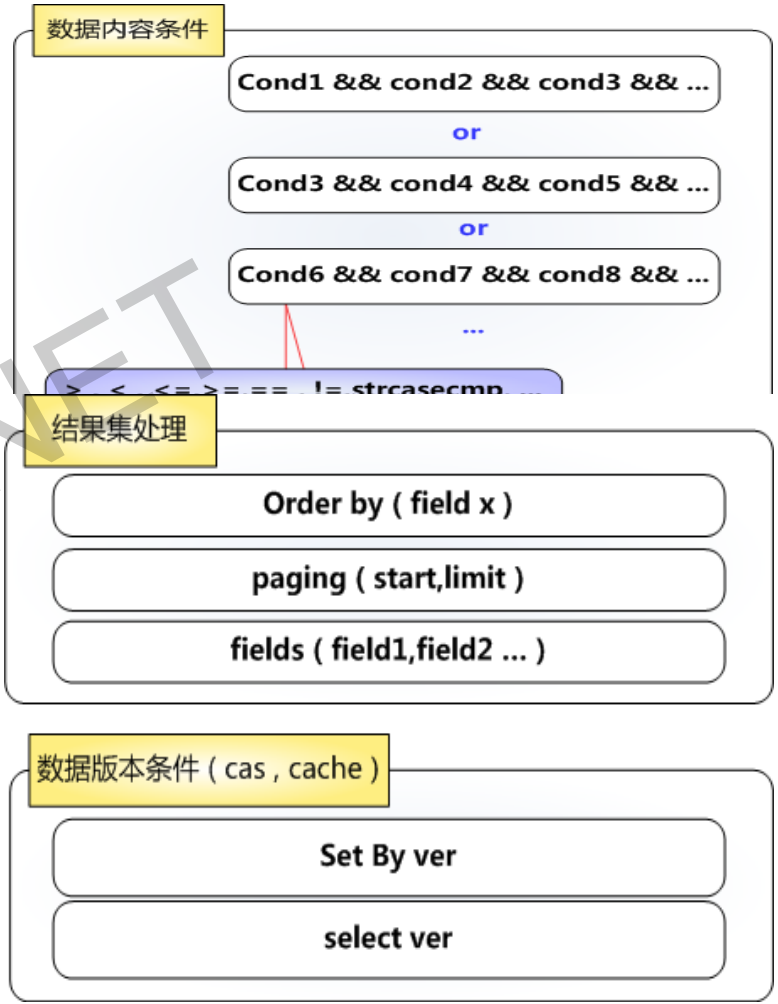
PaxosStore架构简介

PaxosStore整体架构

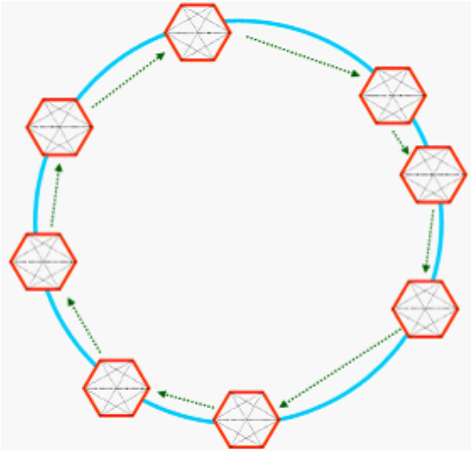


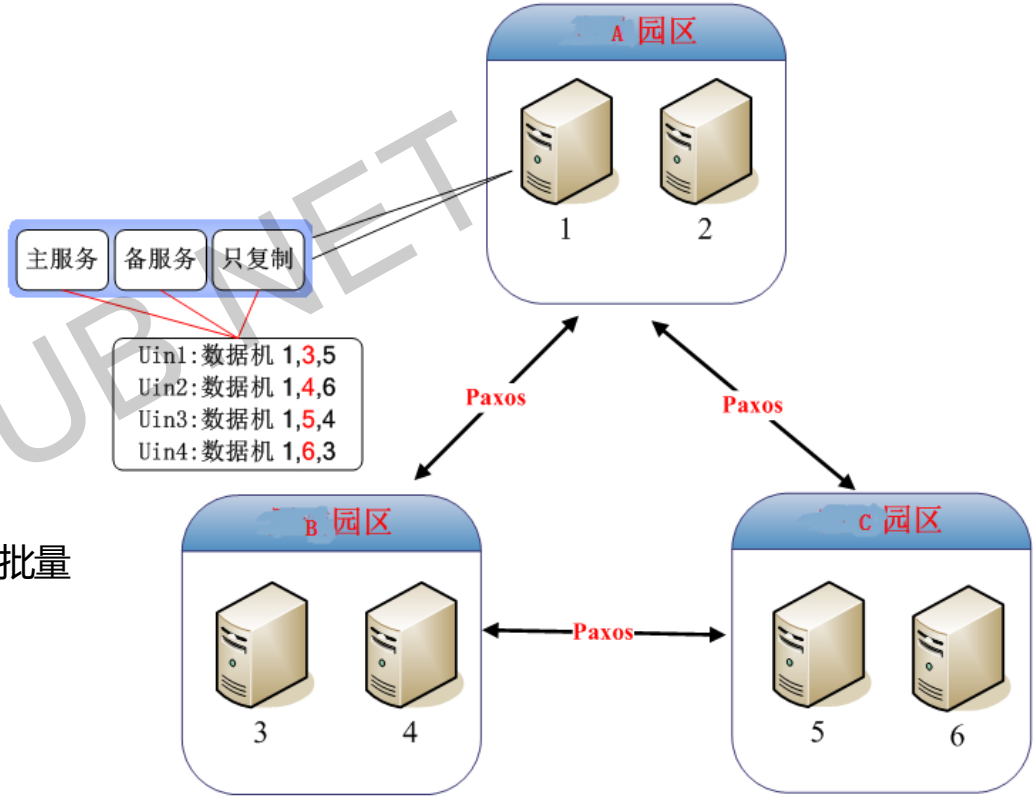
PaxosStore：数据模型

- 丰富数据类型，为业务快速开发提供保障
  - 键值
  - 二维小表
  - 大表
  - 集合
  - 队列
- 二维小表：每个小表包含N行，用户自定义列属性，
  对外提供类SQL操作，单表20M限制。
- 大表： 具备二维小表所有功能
  单表支持上亿行，二级索引。



PaxosStore：数据分布与复制





- 按Set划分，Set间一致性Hash均匀分布
- 上下层间调用按Set对齐隔离单Set故障和提升批量RPC效果
- Set内三园区互为主备， 负载均衡
- 单机/园区分级容灾
- 单机 /单园区故障，服务不受影响
- 就近访问

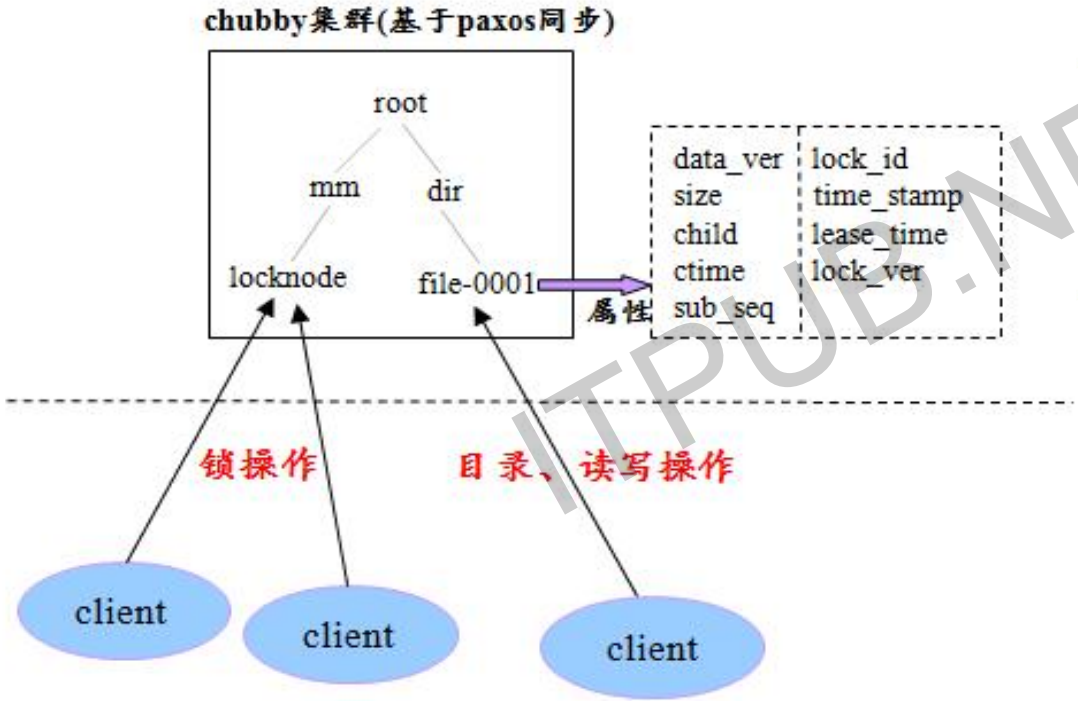
基于PaxosStore的在线基础组件

- 分布式元数据系统/文件系统/表格系统
  - 业界：Chubby/zookeeper, gfs/hdfs , bigtable/hbase

- 远距离跨城常量存储
  - 远距离 上海 – 深圳 – 天津
    考虑故障的实际影响范围以及专线的物理情况
    例如深圳与汕头，上海与杭州，杭州和杭州旁边的城市，这些都只是 短距离容灾
  - 跨城写，本地读

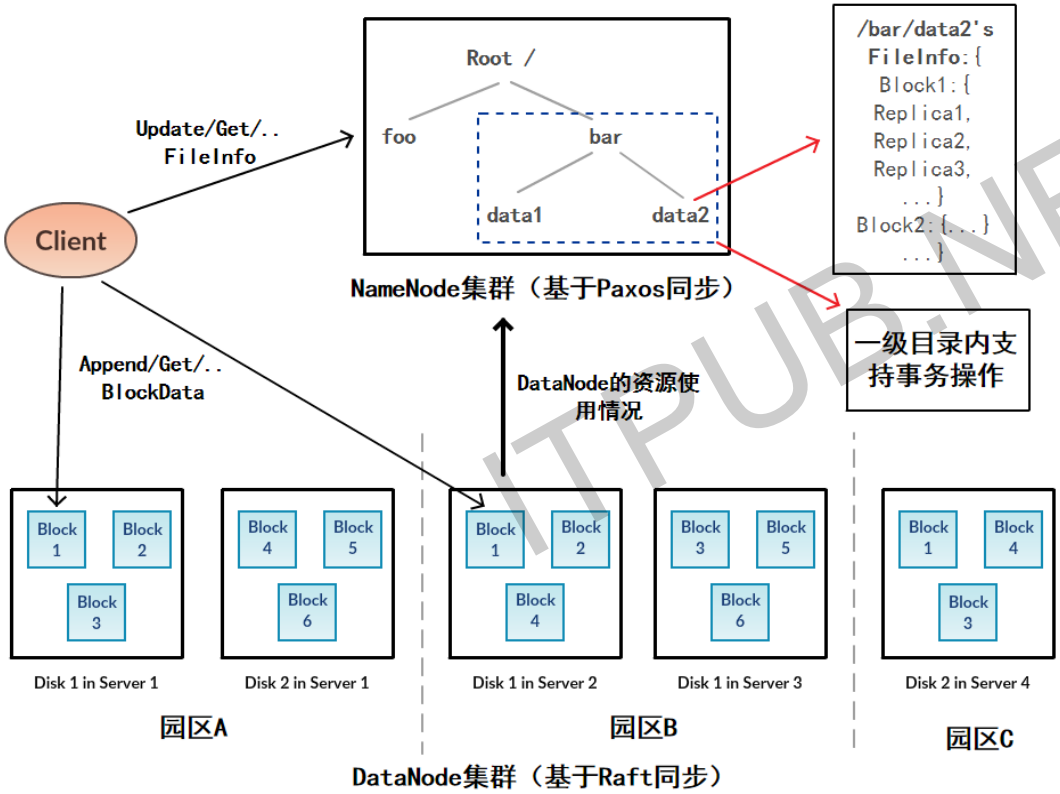
- 针对微信支付复杂业务定制的事务容器
- 针对搜索推荐业务的高性能特征存储

微信chubby



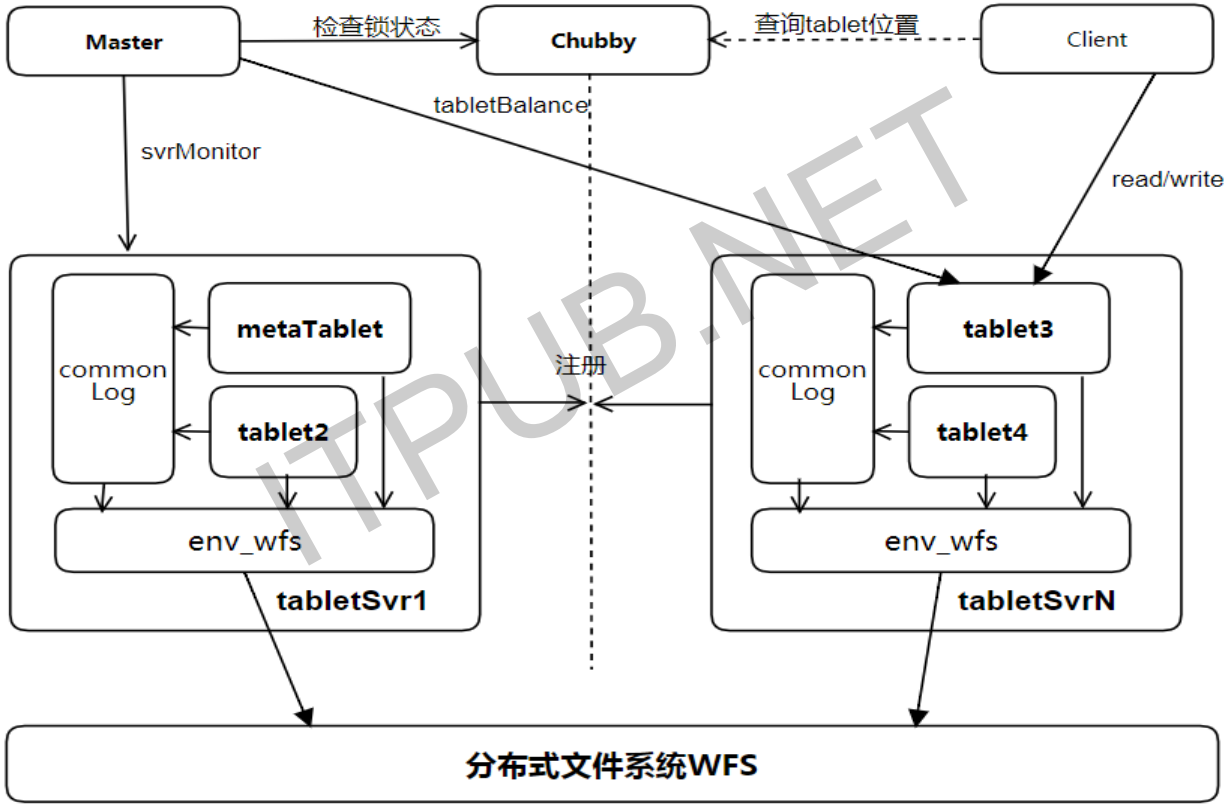
- 元数据存储：小文件存储和目录管理
  GetStat/GetContent/SetContent
  MultiSetContent ：原子操作、CAS
- 支持分布式锁
  AcquireLock/ReleaseLock,
  ChekHeldLock/锁失效回调

微信分布式文件系统



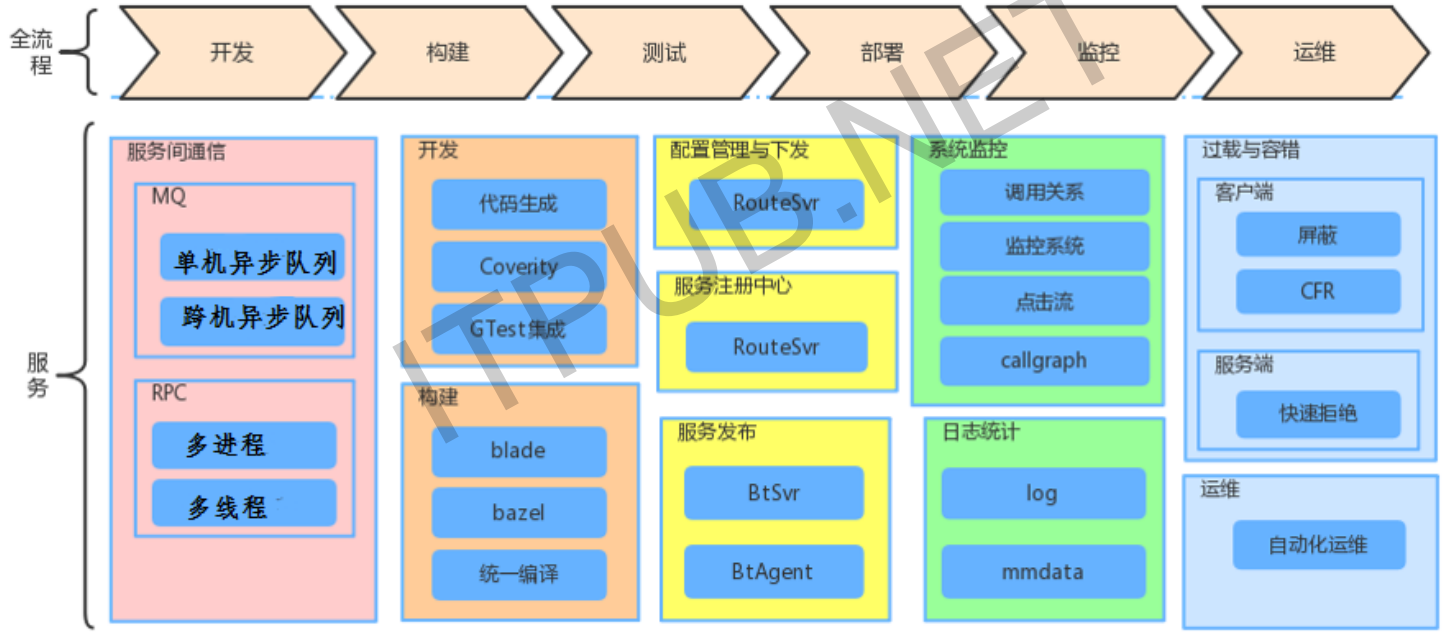
- Namenode 基于PaxosStore
- DataNode基于Raft协议
- 文件AppendWrite和随机读，常用目录操作
- 支持回收站等功能

分布式表格：架构图



微服务框架

- 封装了包含服务定义、服务发现、错误重试、监控容灾、灰度发布等一系列面向服务的高级特性的统一框架



业务逻辑Worker模型选择

	优点	缺点
多进程	进程间相互独立，奔溃互不影响	进程间通信开销大
	Maxloop特性，容忍偶发core或内存泄漏，支持业务快速迭代开发	资源无法共享
多线程	资源进程内共享	一个线程的奔溃影响整个程序
	程序逻辑和控制方式简单	
	性能好	
协程	进程/线程内调度，性能好，并发能力高	cpu密集型服务会导致其它协程卡顿

主要介绍一下协程的应用

Libco背景

- 同步模型
  - 支持业务快速迭代开发
- 问题
  - 一个请求占用一个业务进程（线程）同步处理
  - 调用链路长，网络延迟抖动引发整个调用链路失败
  - 系统并发处理能力与业务请求量不匹配导致故障频繁
- 解决办法
  - 切换到异步模型（工程量巨大）
  - 探索协程解决方案，少量修改代码达到同步编码，异步执行效果

举个例子

- 把Leveldb的本地文件切换为远程文件系统
  - 异步代码如何实现
  - 协程如何实现

异步服务与协程服务的对比

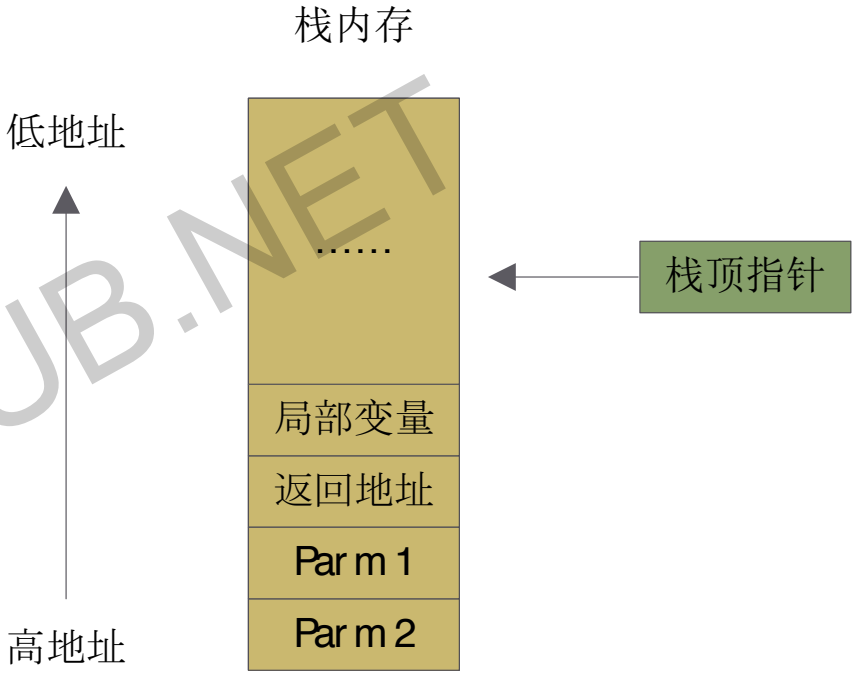
- 传统基于事件驱动的异步网络服务
  - 优势，高性能，cpu利用率高
  - 编写困难，需要业务层维护状态机，业务逻辑被异步编码拆分得支离破碎
  - Future/promise等技术，趋近于同步编程，但仍然繁琐
  - 并发任务难以编写与维护
- 协程
  - 同步编程，异步执行
  - 由于不需要自己设计各种状态保存数据结构，临时状态/变量在一片连续的栈中分配，性能比手写异步通常要高
  - 非常方便地编写并发任务

协程定义

- 协程是什么
  - 微线程、用户态线程
  - 线程内有独立程序上下文的执行体
- 协程切换流程
  - 本协程上下文保存
  - 目标上下文恢复
  - 指令地址跳转

函数调用的基本原理

- 32位程序为例
- 函数调用
  - 压栈被调函数参数
  - 调用Call指令
    - 压栈当前指令地址
    - 跳转到目标函数地址
- 函数回溯
  - Ret指令
    - 从栈中恢复指令到eip



Libco协程切换核心代码

源代码	源码解释
`#函数定义` `#void coctx_swap( coctx_t* curr,coctx_t* next);`	函数定义：curr当前上下文保存地址，next目标上下文保存地址

`coctx_swap:`
`#if defined(__i386__)`
`movl 4(%esp), %eax`
`movl %esp, 28(%eax)`
`movl %ebp, 24(%eax)`
`movl %esi, 20(%eax)`
`movl %edi, 16(%eax)`
`movl %edx, 12(%eax)`
`movl %ecx, 8(%eax)`
`movl %ebx, 4(%eax)`
	保存当前上下文到curr指针指向的地址
	1、movl 4(%esp), %eax //获取curr指针
	2、保存寄存器的值到堆内存中

`movl 8(%esp), %eax`
`movl 4(%eax), %ebx`
`movl 8(%eax), %ecx`
`movl 12(%eax), %edx`
`movl 16(%eax), %edi`
`movl 20(%eax), %esi`
`movl 24(%eax), %ebp`
`movl 28(%eax), %esp`
	恢复next中的上下文到寄存器中
	1、movl 8(%esp), %eax //获取curr指针
	2、从堆内存中恢复数据

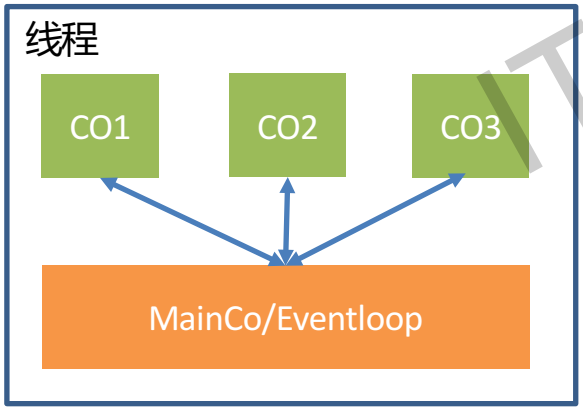
`ret`	函数回溯

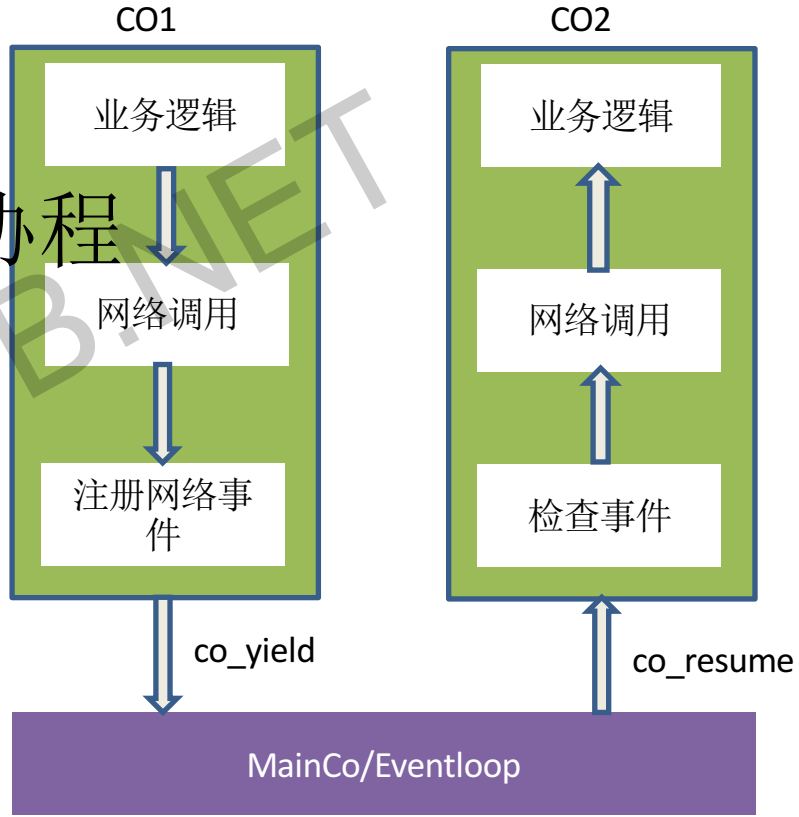
Libco框架

- 协程源语
  - co_create/co_resume/co_yield
  - 协程间同步cond/signal
- 轻量级网络事件库
  - 基于epoll的事件回调
  - 基于时间轮盘的定时器设计
- Socket族函数hook
  - 网络读写api hook
  - 支持mysqlclient/gethostbyname
  - 支持aio读磁盘
  - 支持cgi(getenv/setenv)

线程内协程切换图

- EventLoop主循环
- 基于网络事件切换协程





Libco下编程需要注意的点

- 栈大小默认设置为128k；
- 特殊场景可以开启共享栈，支持单机千万协程
- 需要注意的情况
  - 用poll(NULL,0, timeout)代替sleep
  - 对于计算较重的服务，需要分离计算线程与网络线程，避免互相影响

Q & A

sunnyxu@tencent.com

THANKS