欢聚时代

账号平台技术历程

议题

- 账号平台简介
- 架构演进之路
  - 服务化
  - 高可用改造
  - 数据分析

账号平台的使命

2012年-2013年

质量

YY登陆服务质量优化

安全

统一登陆（入口/存储）

数据分级安全体系

服务

安全中心/申述中心

快速登陆

账号互联

2014年

质量

- YY登陆服务质量优化

安全

- 统一登陆（入口/存储）
- 数据分级安全体系
- 统一策略验证
- 账号运营

服务

- 安全中心/申述中心
- 快速登陆
- 账号互联
- 手机安全中心
- 自助接入平台

技术历程

质量优化（服务化）

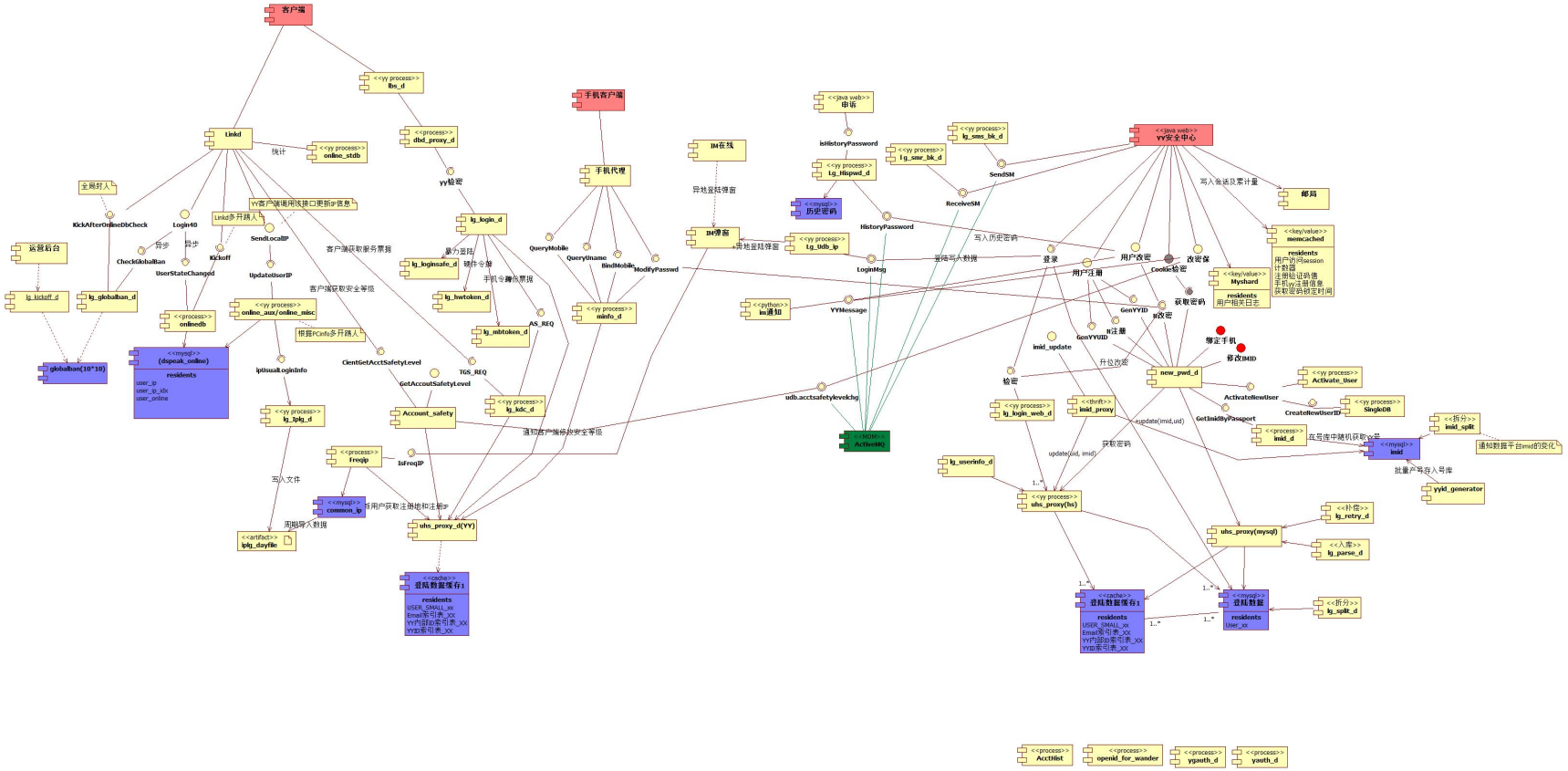
质量优化（高可用改造）

基于大数据分析的策略运营

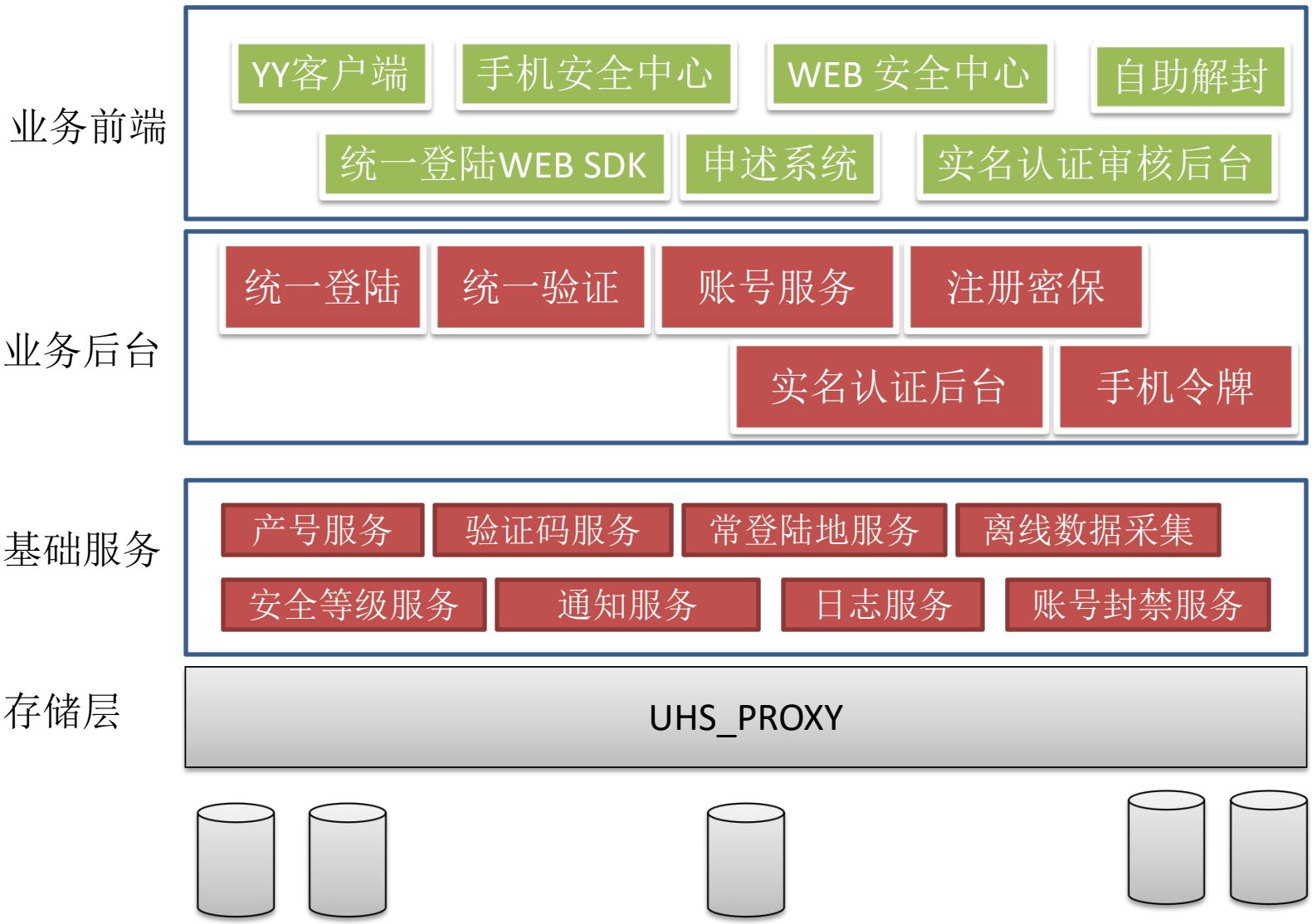
服务化

- 曾经的蜘蛛网
- 层次化的账号平台架构
- 合理的服务划分

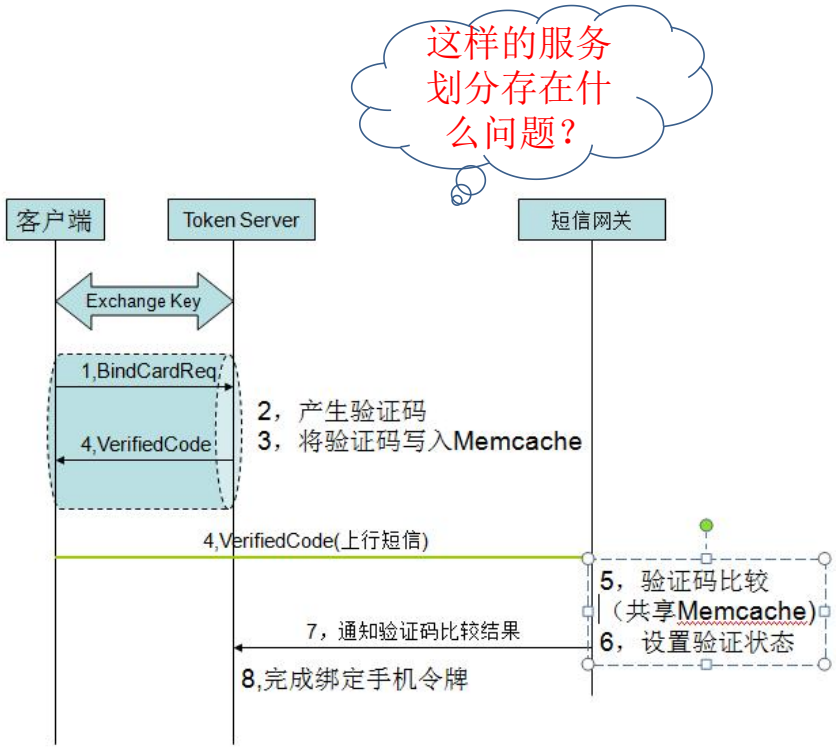
曾经的蜘蛛网

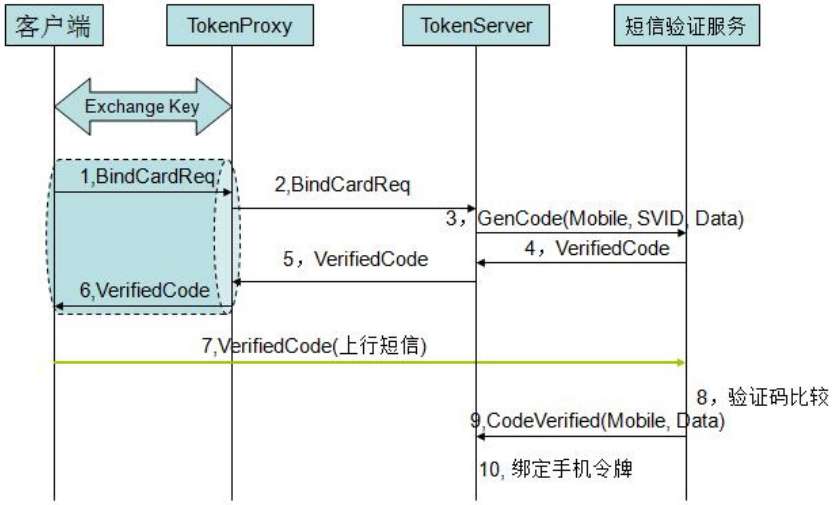

客户端
<<yy process>>
lbs_d
Linkd
统计
<<yy process>>
online_stdb
<<process>>
dbd_proxy_d
yy验密
手机客户端
手机代理
IM在线
异地登陆弹窗
IM弹窗
<<java web>>
申诉
isHistoryPassword
<<yy process>>
lg_Hspwd_d
<<mysql>>
历史密码
HistoryPassword
<<yy process>>
lg_sms_bk_d
SendSM
ReceiveSM
<<java web>>
YY安全中心
写入会话及累计量
邮局
全局封人
KickAfterOnlineDbCheck
Login40
Linkd多开踢人
SendLocalIP
Kickoff
CheckGlobalBan
UserStateChanged
UpdateUserIP
异步
登步
YY客户端调用该接口更新ip信息
客户端获取服务票据
客户端获取安全等级
运营后台
lg_kickoff_d
lg_globalban_d
<<process>>
onlinedb
<<yy process>>
online_aux/online_misc
根据PCinfo多开踢人
<<mysql>>
(dspeak_online)
residents
user_ip
user_ip_idx
user_online
globalban(10*10)
ipUsualLoginInfo
<<yy process>>
lg_lplg_d
CientGetAcctSafetyLevel
GetAccountSafetyLevel
Account_safety
TGS_REQ
<<yy process>>
lg_kdc_d
通知客户端修改安全等级
<<process>>
Freqip
IsFreqIP
<<mysql>>
common_ip
用户获取注册地和注册IP
写入文件
周期导入数据
<<artifact>>
lplg_dayfile
uhs_proxy_d(YY)
<<cache>>
登陆数据缓存1
residents
USER_SMALL_xx
Email索引表_xx
YY内部ip索引表_xx
YYID索引表_xx
<<yy process>>
lg_login_d
暴力登陆
硬件令牌
手机令牌动态票据
lg_loginsafe_d
lg_hwtoken_d
lg_mbtoken_d
AS_REQ
QueryMobile
QueryUname
BindMobile
ModifyPasswd
<<yy process>>
minfo_d
<<yy process>>
Lg_Udb_ip
LoginMsg
异地写入数据
写入历史密码
<<python>>
im通知
YYMessage
udb.acctsafetylevelchg
<<mysql>>
ActiveMQ
登录
用户注册
用户改密
改密保
Cookie验密
<<key/value>>
Myshard
residents
用户相关日志
<<key/value>>
memcached
residents
用户访问session
计数器
注册验证码值
手机yy注册信息
获取登录锁定时间
获取密码
N注册
GenYYUID
GenYYID
改密
绑定手机
修改IMID
imid_update
升位改密
验密
new_pwd_d
<<yy process>>
lg_login_web_d
<<thrift>>
imid_proxy
update(imid,uid)
update(uid, imid)
获取密码
<<yy process>>
Activate_User
ActivateNewUser
CreateNewUserID
<<yy process>>
SingleDB
GetImidByPassport
<<process>>
imid_d
在号库中随机获取yy号
<<mysql>>
imid
<<拆分>>
imid_split
通知数据平台imid的变化
批量产号容入号库
yyid_generator
lg_useriinfo_d
<<yy process>>
uhs_proxy(hs)
uhs_proxy(mysql)
<<补偿>>
lg_retry_d
<<入库>>
lg_parse_d
<<cache>>
登陆数据缓存1
residents
USER_SMALL_xx
Email索引表_xx
YY内部ip索引表_xx
YYID索引表_xx
<<mysql>>
登陆数据
residents
User_xx
<<拆分>>
lg_split_d
<<process>>
AcctHist
<<process>>
openid_for_wander
<<process>>
ygauth_d
<<process>>
yauth_d

层次化的账号平台架构


业务前端
YY客户端
手机安全中心
WEB 安全中心
自助解封
统一登陆WEB SDK
申述系统
实名认证审核后台
业务后台
统一登陆
统一验证
账号服务
注册密保
实名认证后台
手机令牌
基础服务
产号服务
验证码服务
常登陆地服务
离线数据采集
安全等级服务
通知服务
日志服务
账号封禁服务
存储层
UHS_PROXY

合理的服务划分


这样的服务划分存在什么问题？
客户端
Token Server
短信网关
Exchange Key
1,BindCardReq
2，产生验证码
3，将验证码写入Memcache
4,VerifiedCode
4,VerifiedCode(上行短信)
5，验证码比较
（共享Memcache）
6，设置验证状态
7，通知验证码比较结果
8,完成绑定手机令牌


客户端
TokenProxy
TokenServer
短信验证服务
Exchange Key
1,BindCardReq
2,BindCardReq
3，GenCode(Mobile, SVID, Data)
4，VerifiedCode
5，VerifiedCode
6,VerifiedCode
7,VerifiedCode(上行短信)
8，验证码比较
9,CodeVerified(Mobile, Data)
10, 绑定手机令牌

跟随服务化的质量优化

质量优化

- 层层容灾
- 柔性可用
- 业务隔离
- 去中心/去单点
- 无状态
- 服务间调用避免跨机房访问
- 异步化
- 自动运维（离线监控）

层层容灾

客户端容灾

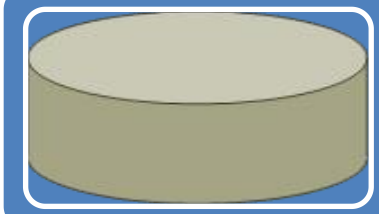
- DNS解析失败
- 接入点连接失败
- 自动测速，就近接入

接入点

- 业务后端容灾（网络故障，进程故障，机房故障）
- 负载均衡

业务后端

- 柔性可用，降级服务
- 自动发现，自动隔离业务流量

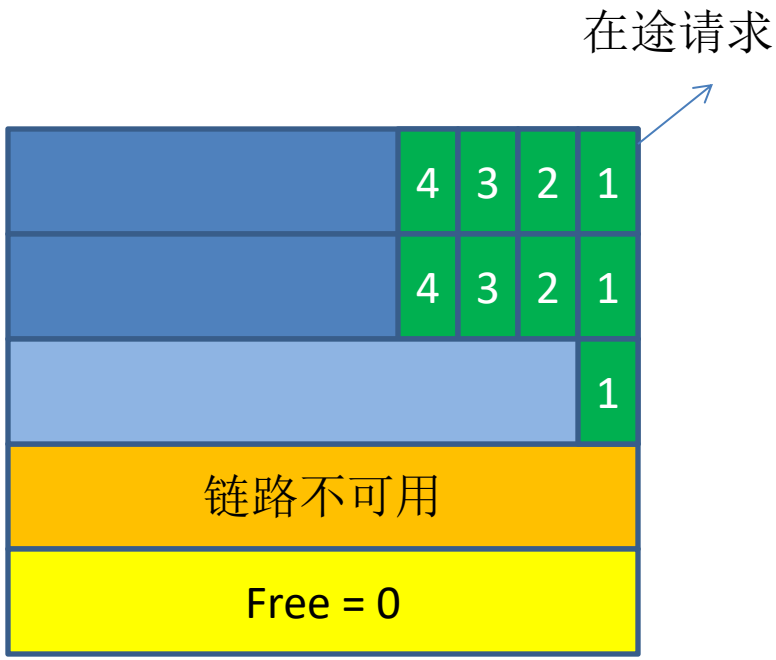


存储层

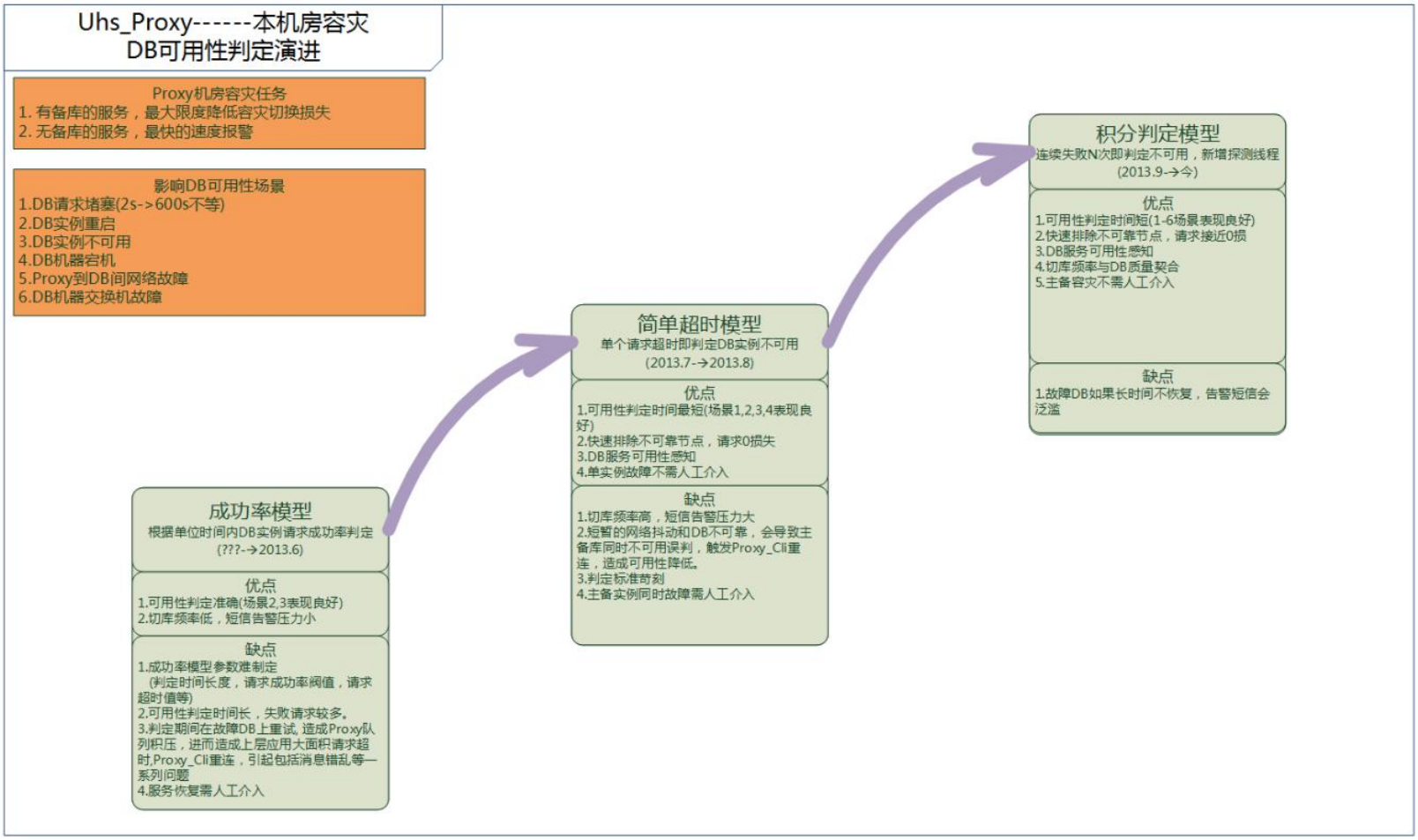
- 自动发现DB故障
- 读操作自动切换，写操作流量隔离

跨机房网络容灾中间件（SERVICE AGVENT）

- 基于服务接口的请求队列
- 动态路由策略
  - 调用延时
  - 负载
  - 成功率
- 业务自动隔离机制
  - 设置free=0
  - 服务端反向控制客户端的流量切换



存储层


Uhs_Proxy------本机房容灾
DB可用性判定演进
Proxy机房容灾任务
1. 有备库的服务，最大限度降低容灾切换损失
2. 无备库的服务，最快的速度报警
影响DB可用性场景
1.DB请求堵塞(2s->600s不等)
2.DB实例重启
3.DB实例不可用
4.DB机器宕机
5.Proxy到DB间网络故障
6.DB机器交换机故障
成功率模型
根据单位时间内DB实例请求成功率判定
(???->2013.6)
优点
1.可用性判定准确(场景2,3表现良好)
2.切库频率低，短信告警压力小
缺点
1.成功率模型参数难制定
(判定时间长度，请求成功率阀值，请求超时值等)
2.可用性判定时间长，失败请求较多。
3.判定期间在故障DB上重试,造成Proxy队列积压，进而造成上层应用大面积请求超时,Proxy_Cli重连，引起包括消息错乱等一系列问题
4.服务恢复需人工介入
简单超时模型
单个请求超时即判定DB实例不可用
(2013.7->2013.8)
优点
1.可用性判定时间最短(场景1,2,3,4表现良好)
2.快速排除不可靠节点，请求0损失
3.DB服务可用性感知
4.单实例故障不需人工介入
缺点
1.切库频率高，短信告警压力大
2.短暂的网络抖动和DB不可靠，会导致主备库同时不可用误判，触发Proxy_Cli重连，造成可用性降低。
3.判定标准苛刻
4.主备实例同时故障需人工介入
积分判定模型
连续失败N次即判定不可用，新增探测线程
(2013.9->今)
优点
1.可用性判定时间短(1-6场景表现良好)
2.快速排除不可靠节点，请求接近0损
3.DB服务可用性感知
4.切库频率与DB质量契合
5.主备容灾不需人工介入
缺点
1.故障DB如果长时间不恢复，告警短信会泛滥

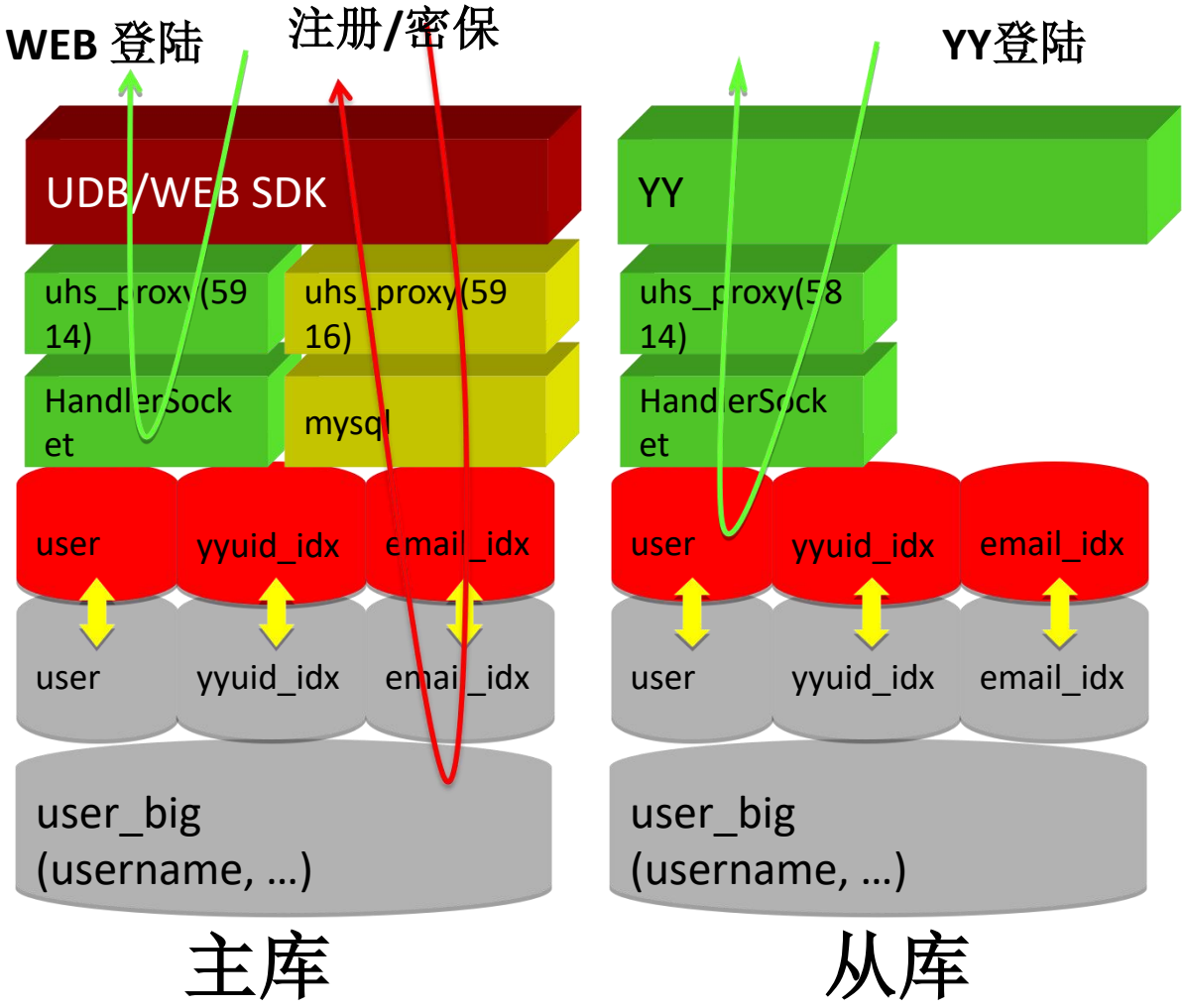
柔性可用

- 2012年3-27日故障分析报告

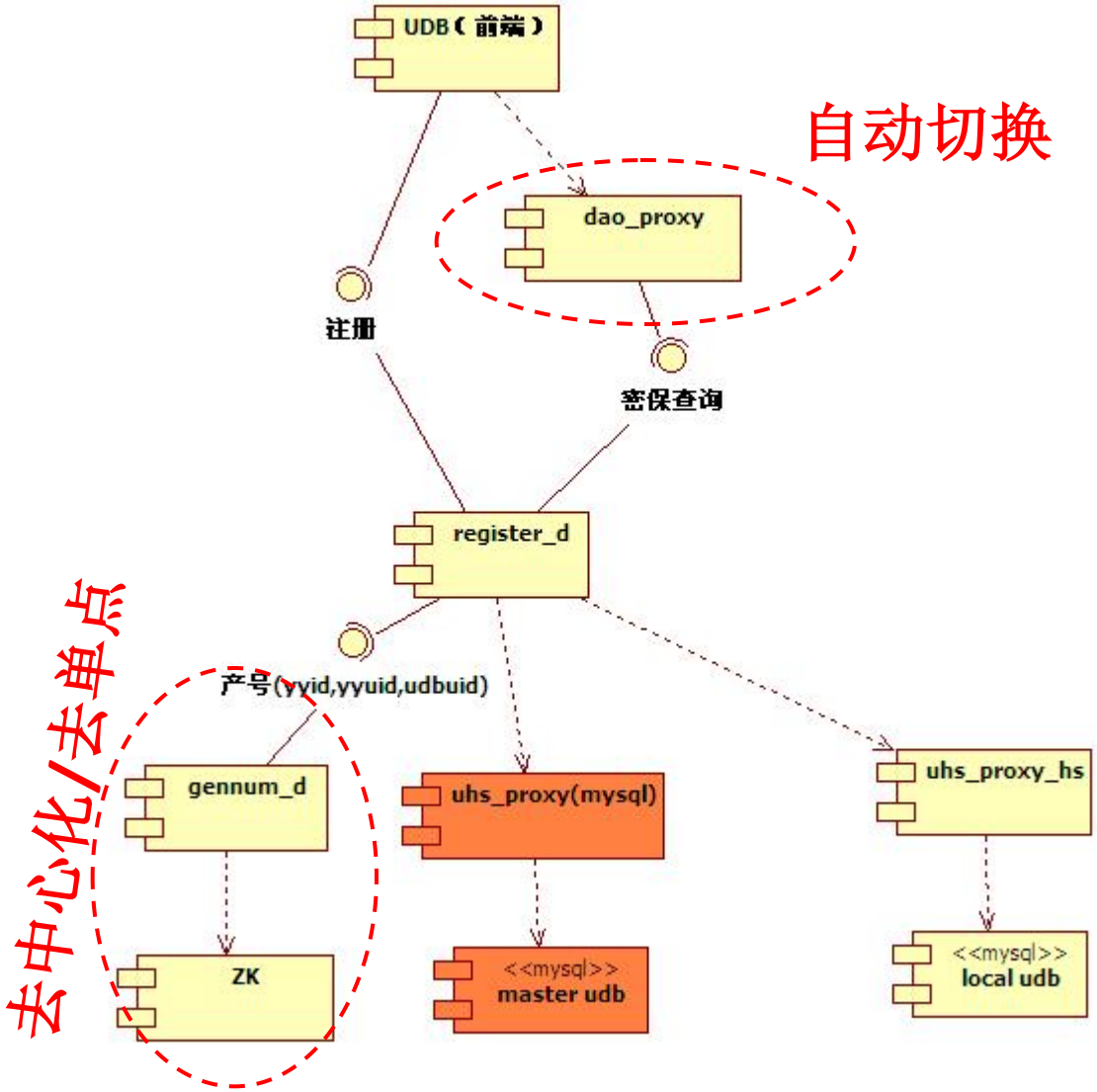
Microsoft Office
Word 文档

- 登陆服务的关键步骤是验密，非关键步骤
  - 账号锁服务
  - 策略服务
  - 图片验证码服务
- 非关键服务不可用时，提供降级服务

业务分离


WEB 登陆
注册/密保
YY登陆
UDB/WEB SDK
YY
uhs_proxy(5914)
uhs_proxy(5916)
uhs_proxy(5814)
HandlerSocket
mysql
HandlerSocket
user
yyuid_idx
email_idx
user
yyuid_idx
email_idx
user
yyuid_idx
email_idx
user
yyuid_idx
email_idx
user_big
(username, ...)
user_big
(username, ...)
主库
从库

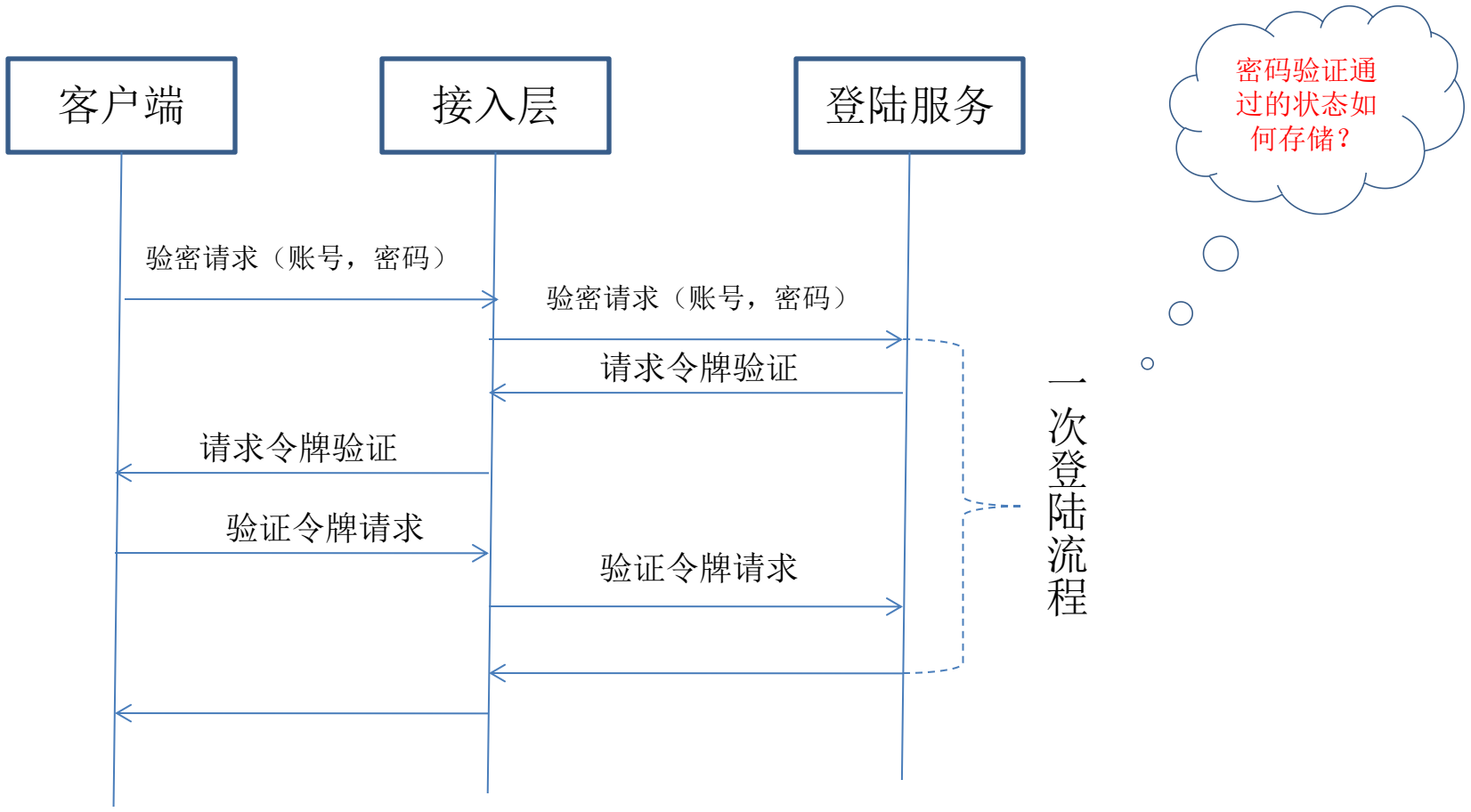
去中心/去单点-注册服务


UDB（前端）
自动切换
dao_proxy
注册
密保查询
register_d
产号(yyid,yyuid,udbuid)
去中心化/去单点
gennum_d
uhs_proxy(mysql)
uhs_proxy_hs
ZK
<<mysql>>
master udb
<<mysql>>
local udb

去状态

- 业务请求间无进程内状态
  - 接入层请求分配简单
  - 单进程故障不影响业务流程
    - Crash
    - 升级
- 进程外状态存储
  - HTTP Cookie
  - 缓存
  - YY Cookie
  - 票据

去状态-令牌认证


客户端
接入层
登陆服务
密码验证通过的状态如何存储？
验密请求（账号，密码）
验密请求（账号，密码）
请求令牌验证
请求令牌验证
验证令牌请求
验证令牌请求
一次登陆流程

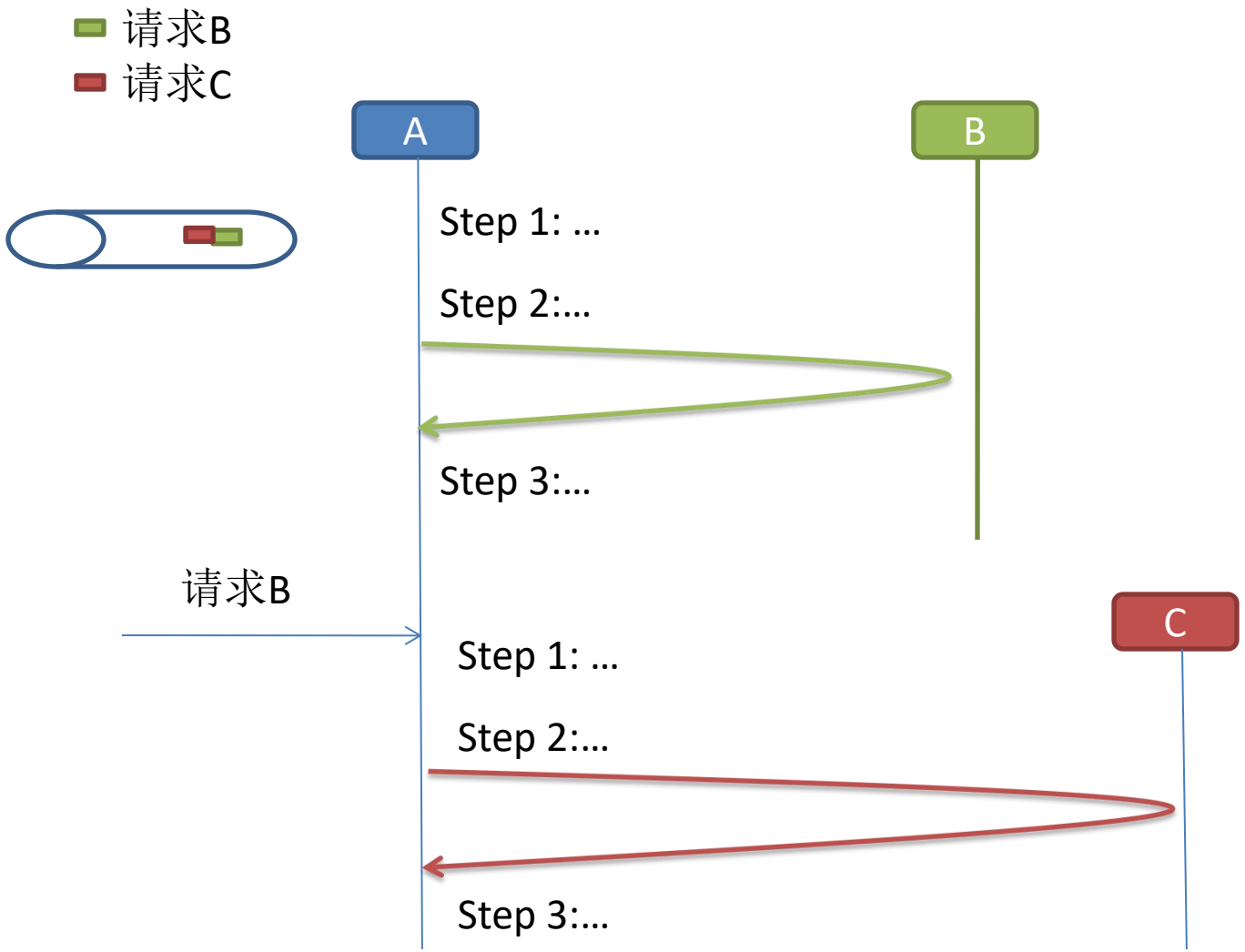
异步化

- 异步 vs 同步
- 异步状态机编程框架介绍
- 基于异步状态机设计的策略验证服务

同步vs异步

- 单线程同步顺序调用
- 多线程同步顺序调用
- 多线程异步顺序调用

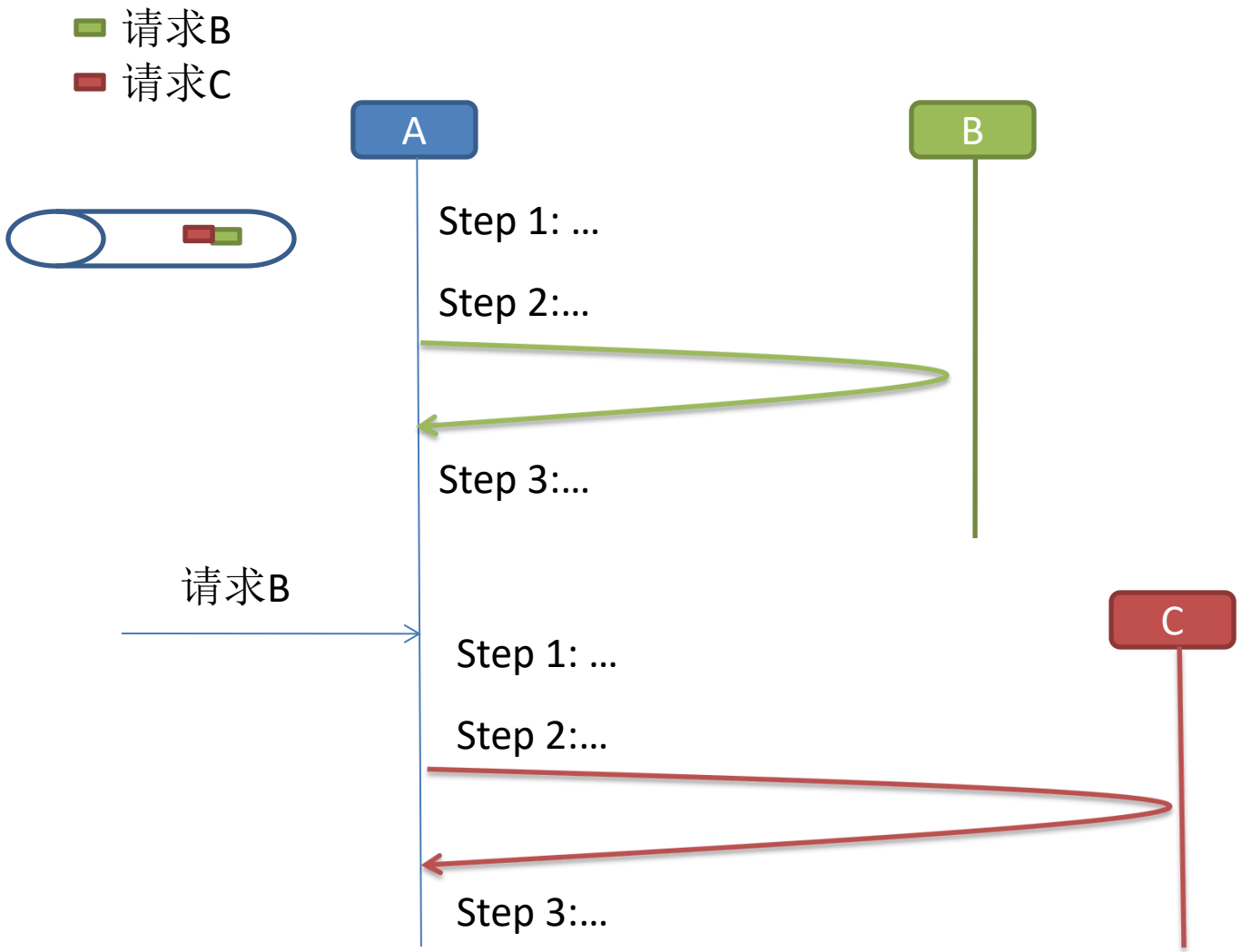
单线程同步顺序调用



单线程同步顺序调用

- 优点
  - 程序设计简单
- 缺点
  - 并发低
    - 请求A的处理时间为20ms，TPS=1000/20=50。
  - 可用性差
    - 服务B或C不可用➔服务A不可用

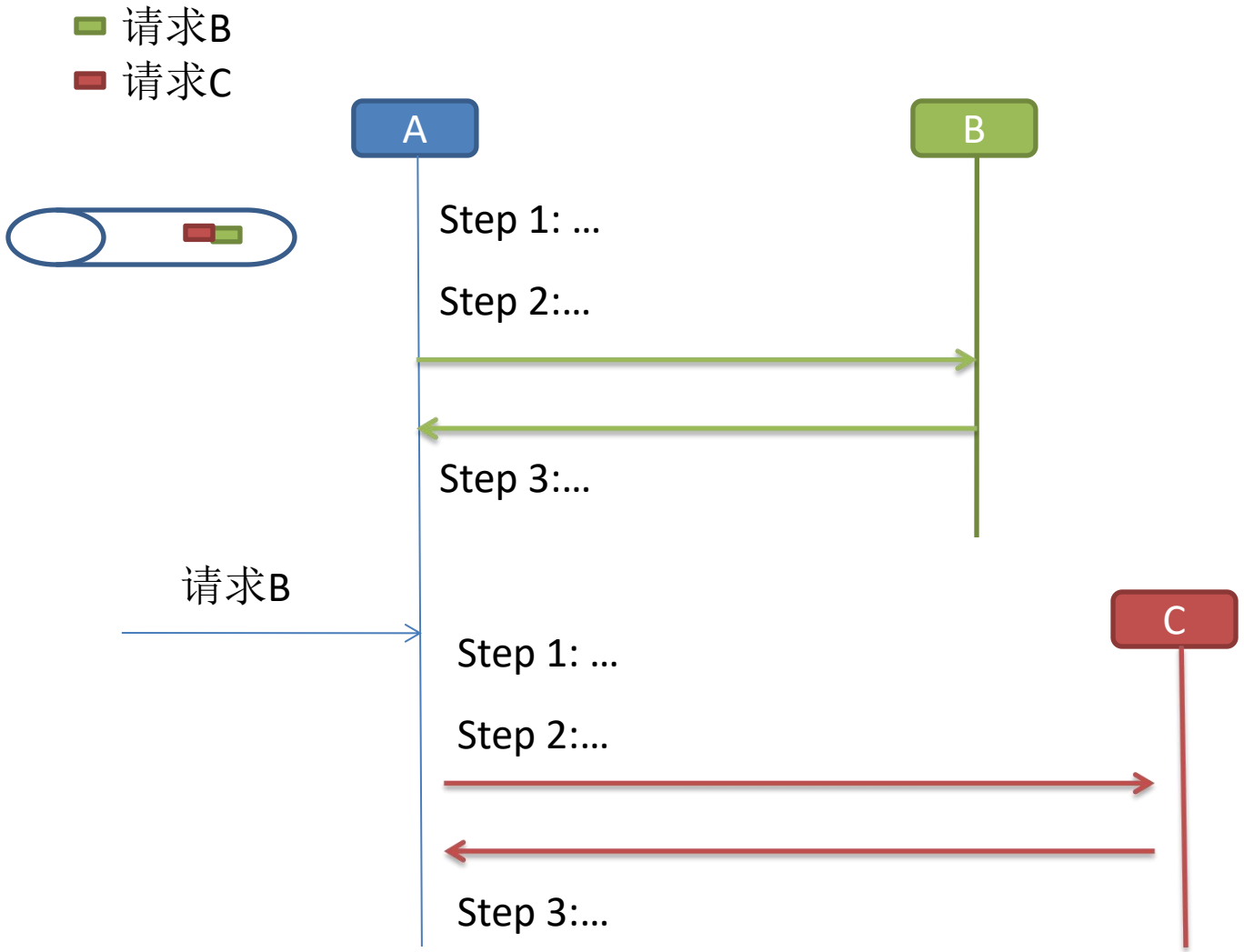
多线程同步顺序调用



多线程同步顺序调用

- 优点
  - 并发度提高
    - 请求A的处理时间为20ms，线程数为100，TPS=1000/20*100=5000。
- 缺点
  - 复杂度提高，多线程数据共享
  - 可用性有改善，但是依然差
    - 服务B或C不可用➔服务A不可用

多线程异步调用



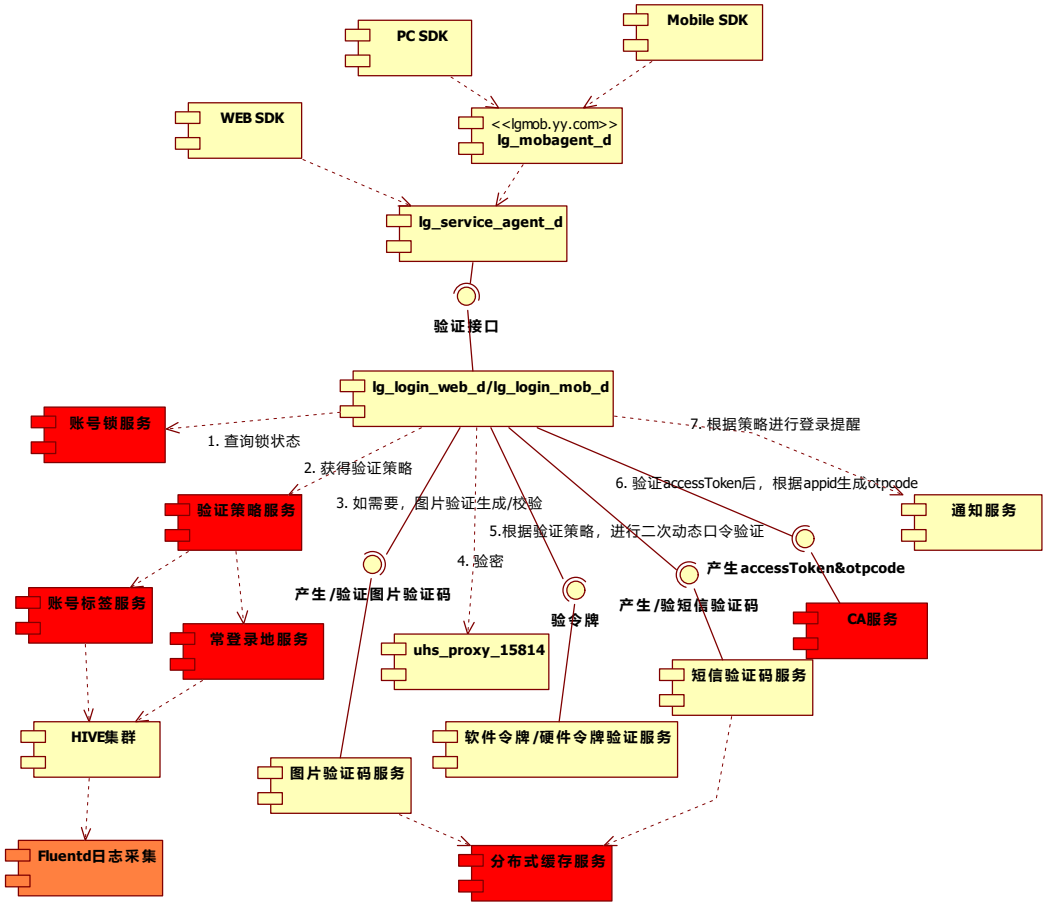
多线程异步调用

- 优点
  - 并发性提高
    - 请求的处理时间与并发度无关
      - 请求A的处理时间为20ms，服务A单线程每秒能够处理的消息数>>1000/20=50。
      - 服务A每秒能够处理的消息数：1000/∑(t1+t2+…+tn)*CPU数。
  - 可用性提高
    - 服务B不可用只影响到需要访问服务A的请求。
    - 服务A正常对外提供业务
- 缺点
  - 复杂性进一步提高，顺序编程变成消息驱动
  - 不同的业务请求异步调用同一基础服务时，对程序设计将会是一个噩梦？

基于状态机的异步框架

- 框架完成
  - 状态机上下文的管理
  - 超时机制
  - 状态机自动扭转
  - 数据统计
- 业务开发
  - 状态机的定义
    - 主要的设计工作
  - 状态事件处理函数的实现
    - 复杂度降低

异步状态编程示例-策略验证服务



状态机设计

STATE	EVENT	NEXT STATE	Processing
INIT	Load User Data Failed（系统错误）	END	返回响应（系统错误）
	Load User Data Failed（用户不存在）	Policy Querying	发送SP Query请求
	Load User Data Suc	AC Querying	初始化用户数据 发送AC Query请求
AC Querying	Received AC Response（Locked）	END	返回响应（账号被锁）
	Received AC Response（Unlocked）	Policy Querying	发送 SP Query请求
	Timeout	Policy Querying	发送 SP Query请求
Policy Querying	Received SP Response	Policy Processing	
	Timeout	Policy Timeout Processing	
Policy Processing	防刷策略=冻结	END	返回响应（冻结）
	防刷策略=图片验证码	PicCode Getting	发送PicCode Get请求
	验密失败	END	返回响应（密码错）
	账号不存在	END	返回响应（密码错）
	动态策略中含有密保问题	SQ Querying	发送SQ Query请求
	动态策略中无密保问题	END	返回响应（二次验证）
	无动态策略	CA Getting	发送CA Get请求
Policy Timeout Processing	验密成功	CA Getting	发送CA Get请求
	验密失败	END	返回响应（密码错）
SQ Querying	Recived SQ Reponse	END	返回响应（二次验证）
	Timeout &动态验证中仅有密保问题	CA Getting	发送 CA Get请求
	Timeout &动态验证中有其他动态策略	END	返回响应（二次验证移出密保问题）
PicCode Getting	Received PicCode Response	END	返回响应（图片验证码）
	Timeout &验密成功 &动态策略=无	CA Getting	发送CA Get请求
	Timeout &验密成功 &动态策略含密保问题	SQ Querying	发送SQ Query请求
	Timeout &验密成功 &动态策略无密保问题	END	返回响应（二次验证）
	Timeout &验密失败	END	返回响应（密码错）
CA Getting	CA Response	END	返回响应（登陆成功）
	Timeout	END	返回响应（系统错误）

实现模型

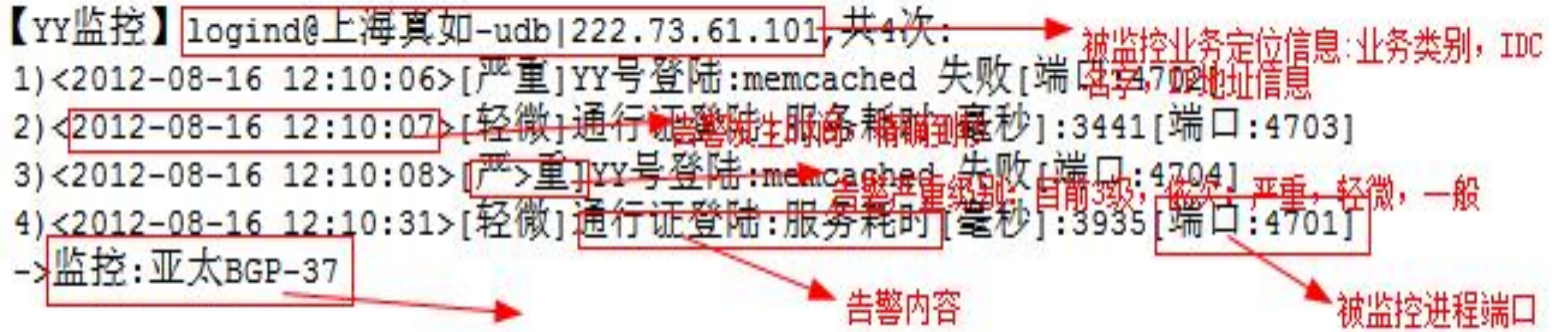
```
LOGIN_EVENT_ENTRY(STATE_START, DPL_EV_PROCESS, init_request, STATE_START),
LOGIN_EVENT_ENTRY(STATE_START, DPL_EV_SYSTEM_ERR, on_error_system, STATE_END),
LOGIN_EVENT_ENTRY(STATE_START, DPL_EV_INIT_USER_INVALID, on_error_invalid_user, STATE_END),
LOGIN_EVENT_ENTRY(STATE_START, DPL_EV_INIT_USER_AUTH_FAIL, on_error_auth_fail, STATE_END),
LOGIN_EVENT_ENTRY(STATE_START, DPL_EV_INIT_USER_VER_SQ, on_start_ver_sq, STATE_VER_SQ),
LOGIN_EVENT_ENTRY(STATE_START, DPL_EV_INIT_USER_VER_MOBTOKEN, on_start_ver_mobtoken, STATE_VER_
LOGIN_EVENT_ENTRY(STATE_START, DPL_EV_INIT_USER_VER_HWTOKEN, on_start_ver_hwtoken, STATE_VER_H
LOGIN_EVENT_ENTRY(STATE_START, DPL_EV_INIT_USER_VER_SMSCODE, on_start_ver_smscode, STATE_VER_SM

LOGIN_EVENT_ENTRY(STATE_VER_MOBTOKEN, DPL_EV_PROCESS, on_ver_mobtoken_resp, STATE_VER_MOBTOK
LOGIN_EVENT_ENTRY(STATE_VER_MOBTOKEN, DPL_EV_SYSTEM_ERR, on_error_system, STATE_END),
LOGIN_EVENT_ENTRY(STATE_VER_MOBTOKEN, DPL_EV_MOBTOKEN_FAIL, on_ver_mobtoken_fail, STATE_END),
LOGIN_EVENT_ENTRY(STATE_VER_MOBTOKEN, DPL_EV_MOBTOKEN_TIMEOUT, on_start_get_ca, STATE_GET_CA)
LOGIN_EVENT_ENTRY(STATE_VER_MOBTOKEN, DPL_EV_MOBTOKEN_SUC, on_start_get_ca, STATE_GET_CA),
//LOGIN_EVENT_ENTRY(STATE_VER_MOBTOKEN, DPL_EV_MOBTOKEN_TIMEOUT, on_ver_mobtoken_timeout, STATE_GET_CA),
//LOGIN_EVENT_ENTRY(STATE_VER_MOBTOKEN, DPL_EV_MOBTOKEN_SUC, on_ver_mobtoken_suc, STATE_GET_CA),

LOGIN_EVENT_ENTRY(STATE_VER_HWTOKEN, DPL_EV_PROCESS, on_ver_hwtoken_resp, STATE_VER_HWTOKEN)
LOGIN_EVENT_ENTRY(STATE_VER_HWTOKEN, DPL_EV_SYSTEM_ERR, on_error_system, STATE_END),
LOGIN_EVENT_ENTRY(STATE_VER_HWTOKEN, DPL_EV_HWTOKEN_FAIL, on_ver_hwtoken_fail, STATE_END),
LOGIN_EVENT_ENTRY(STATE_VER_HWTOKEN, DPL_EV_HWTOKEN_TIMEOUT, on_start_get_ca, STATE_GET_CA),
LOGIN_EVENT_ENTRY(STATE_VER_HWTOKEN, DPL_EV_HWTOKEN_SUC, on_start_get_ca, STATE_GET_CA),
//LOGIN_EVENT_ENTRY(STATE_VER_HWTOKEN, DPL_EV_HWTOKEN_TIMEOUT, on_ver_hwtoken_timeout, STATE_GET_CA),
//LOGIN_EVENT_ENTRY(STATE_VER_HWTOKEN, DPL_EV_HWTOKEN_SUC, on_ver_hwtoken_suc, STATE_GET_CA),

LOGIN_EVENT_ENTRY(STATE_VER_SMSCODE, DPL_EV_PROCESS, on_ver_smscode_resp, STATE_VER_SMSCODE),
```

自动运维

- 离线监控
- 鹰眼统计
- 自动部署（包发布）

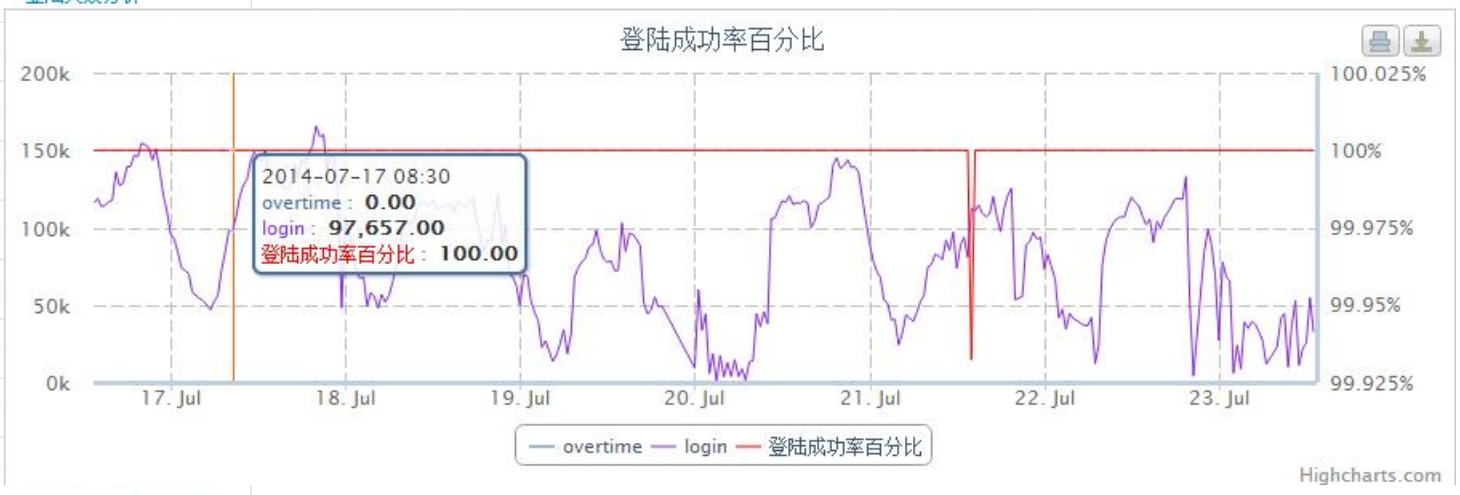
离线监控

- 周期性仿真调用核心接口
- 接入层/逻辑层不同的探测策略
- 告警抑制
  - 去除网络波动带来的误报

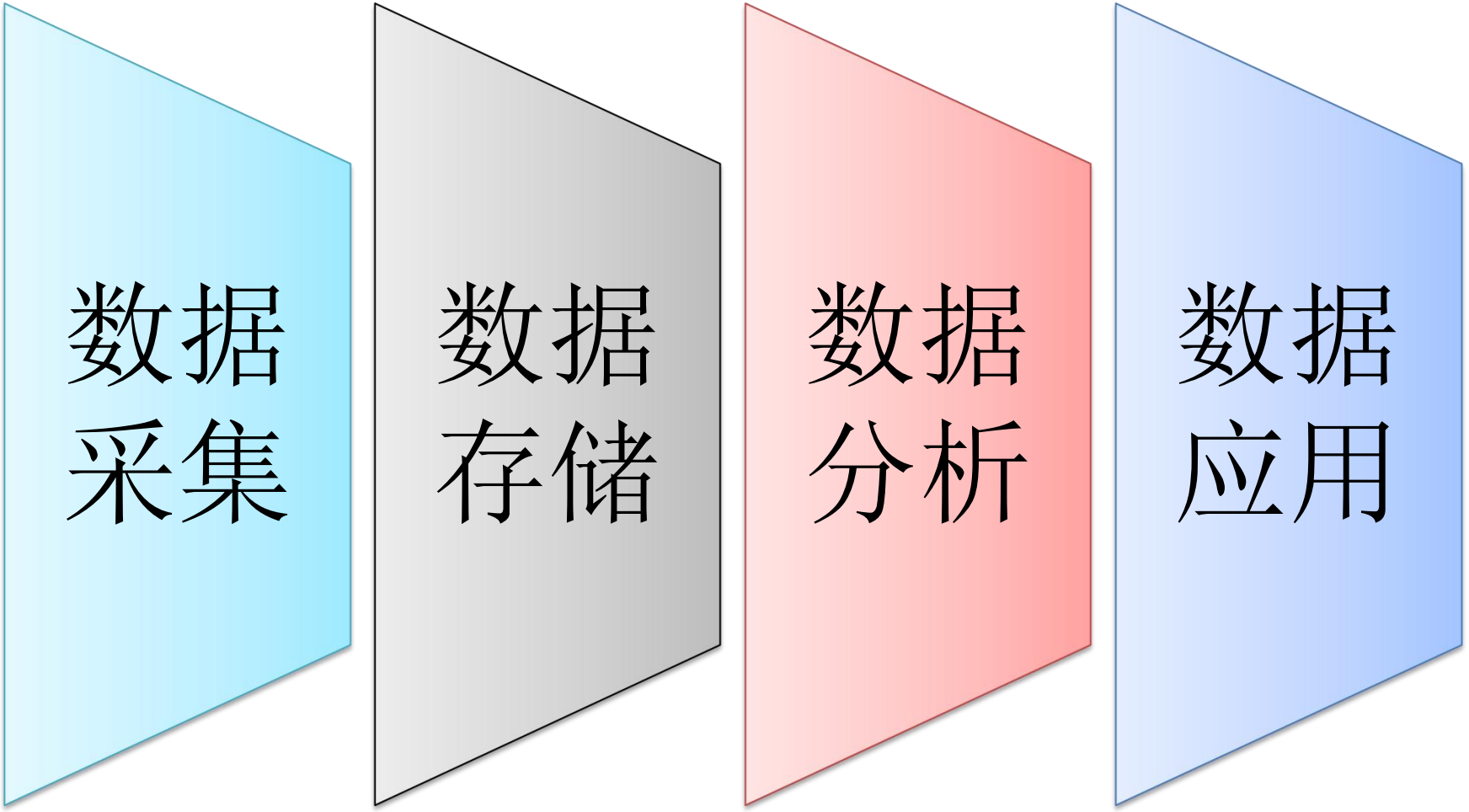


鹰眼统计

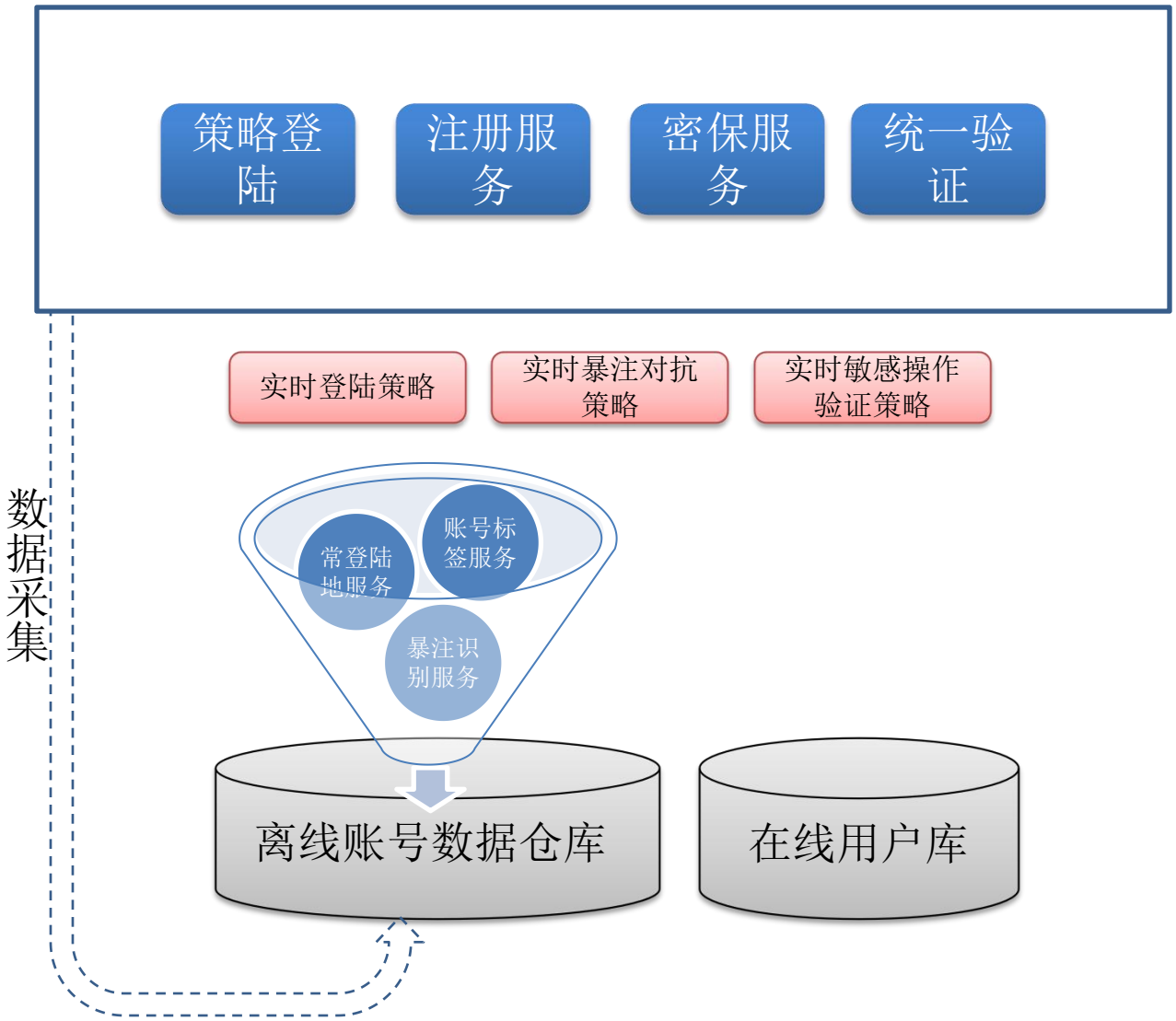
- 登录
  - 登陆可用性
  - 登陆TPS
  - 登陆失败分析



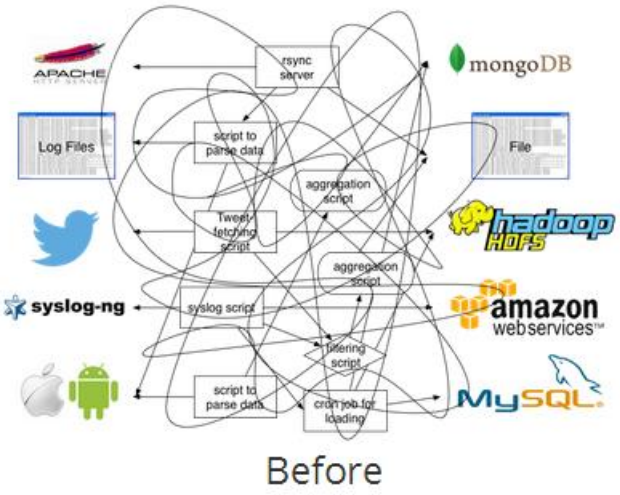
基于数据分析的账号平台



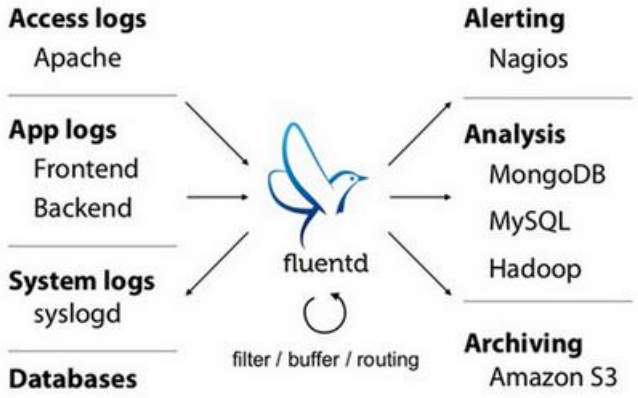
技术方案


策略登陆
注册服务
密保服务
统一验证
实时登陆策略
实时暴注对抗策略
实时敏感操作验证策略
数据采集
常登陆地服务
账号标签服务
暴注识别服务
离线账号数据仓库
在线用户库

数据采集



Before



After Fluentd

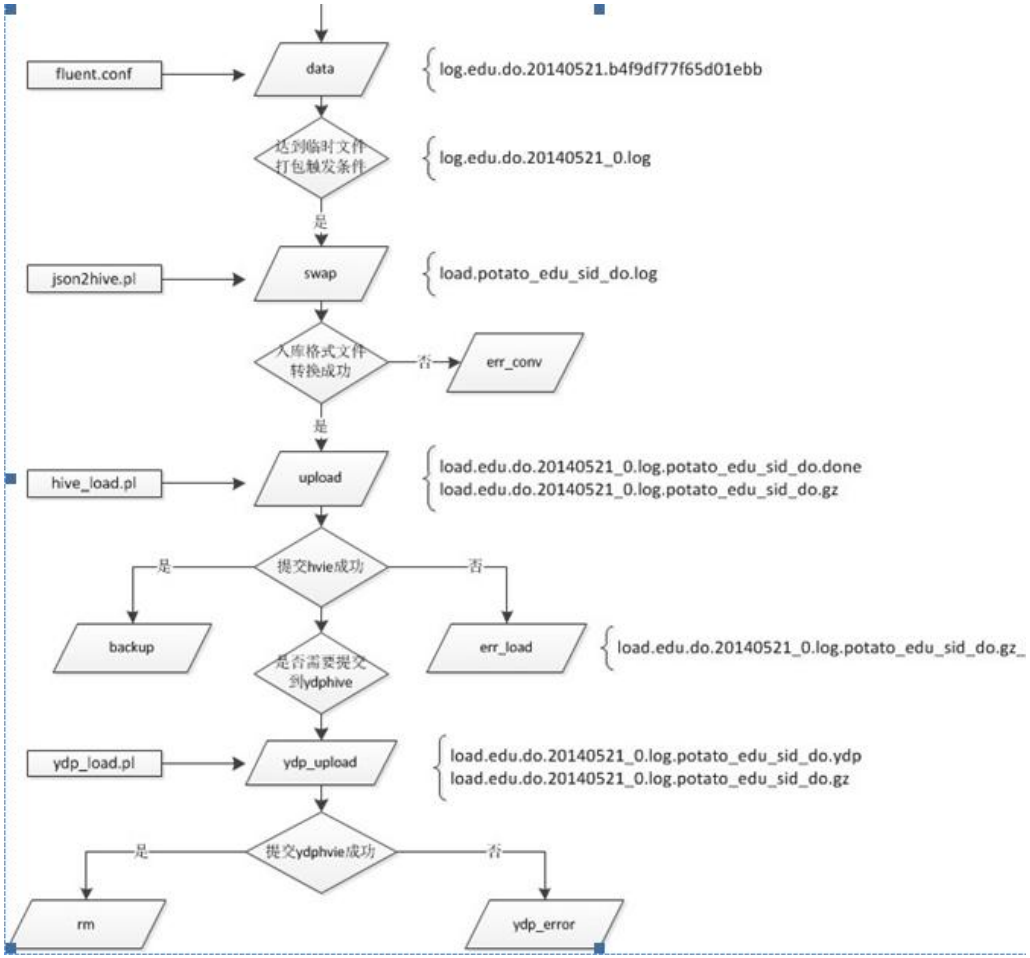
Fluentd 优势

- 无侵入性
- 具备容灾能力
- 扩展性好
  - 插件方式支持各种存储
- 用JSON结构化日志输出

相关开源日志采集系统

	scribe	Chukwa	Kafka	Flume
公司	facebook	apache/yahoo	LinkedIn	Cloudera
开源时间	2008 年 10 月	2009 年 11 月	2010 年 12 月	2009 年 7 月
实现语言	C/C++	JAVA	SCALA	JAVA
框架	push/push	push/push	push/pull	push/push
容错性	collector 和 store 之间有容错机制，而 agent 和 collector 之间的容错需用户自己实现	Agent 定期记录已送给 collector 的数据偏移量，一旦出现故障后，可根据偏移量继续发送数据。	Agent 可用通过 collector 自动识别机制获取可用 collector。store 自己保存已经获取数据的偏移量，一旦 collector 出现故障，可根据偏移量继续获取数据。	Agent 和 collector，collector 和 store 之间均有容错机制，且提供了三种级别的可靠性保证。
负载均衡	无	无	使用 zookeeper	使用 zookeeper
可扩展性	好	好	好	好
agent	Thrift client，需自己实现	自带一些 agent，如获取 hadoop logs 的 agent	用户需根据 Kafka 提供的 low-level 和 high-level API 自己实现。	提供了各种非常丰富的 agent
collector	实际上是一个 thrift server	--	使用了 sendfile，zero-copy 等技术提高性能	系统提供了很多 collector，直接可以使用。
store	直接支持 HDFS	直接支持 HDFS	直接支持 HDFS	直接支持 HDFS
总体评价	设计简单，易于使用，但容错和负载均衡方面不够好，且资料较少。	属于 hadoop 系列产品，直接支持 Hadoop，目前版本升级比较快，但还有待完	设计架构（push/pull）非常巧妙，适合异构集群，但产品较新，其稳定性有待验证。	非常优秀

数据存储


fluent.conf
data
log.edu.do.20140521.b4f9df77f65d01ebb
达到临时文件打包触发条件
log.edu.do.20140521_0.log
是
json2hive.pl
swap
load.potato_edu_sid_do.log
入库格式文件转换成功
否
err_conv
是
hive_load.pl
upload
load.edu.do.20140521_0.log.potato_edu_sid_do.done
load.edu.do.20140521_0.log.potato_edu_sid_do.gz
提交hvie成功
是
否
backup
err_load
load.edu.do.20140521_0.log.potato_edu_sid_do.gz_
是否需要提交到ydphive
ydp_load.pl
ydp_upload
load.edu.do.20140521_0.log.potato_edu_sid_do.ydp
load.edu.do.20140521_0.log.potato_edu_sid_do.gz
提交ydphvie成功
是
否
rm
ydp_error

数据应用

- 外挂对抗
- 暴力注册
- 策略登陆
- 统一验证
- 提高重点账号的安全性


Thanks