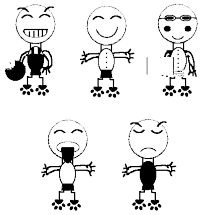


账号系统架构设计及技术选型——PC 端、移动端支持

研发一部：[redacted]　　2014.1.20——2014.1.25

根据协作部门提供的《[redacted]产品线帐号系统需求表》，客户端需要支持 PC 端、移动端、Web 端。其中 PC 端、移动端共用一套后台服务，提供原生 socket 支持。Web 端独自使用一套服务，提供 web 服务支持。决定把任务拆分为两个大方向进行：

- 支持 PC 端，移动端的架构设计及实现；
- 支持 Web 端的架构设计及实现；

主要需求提炼：

- ✧ 账号基本功能：注册、登录、密码修改、找回密码…；
- ✧ 账号状态信息：上线，在线，离线，多终端登录（同时登录或互踢）…；
- ✧ 消息推送功能：服务端主动推送消息到客户端，客户端之间的通信；
- ✧ 提供协作部门开发接口或协议：为协作部门提供通用、易用的 SDK 及接口；
- ✧ 负载考虑：随着用户量的增大提高后台的负载能力；

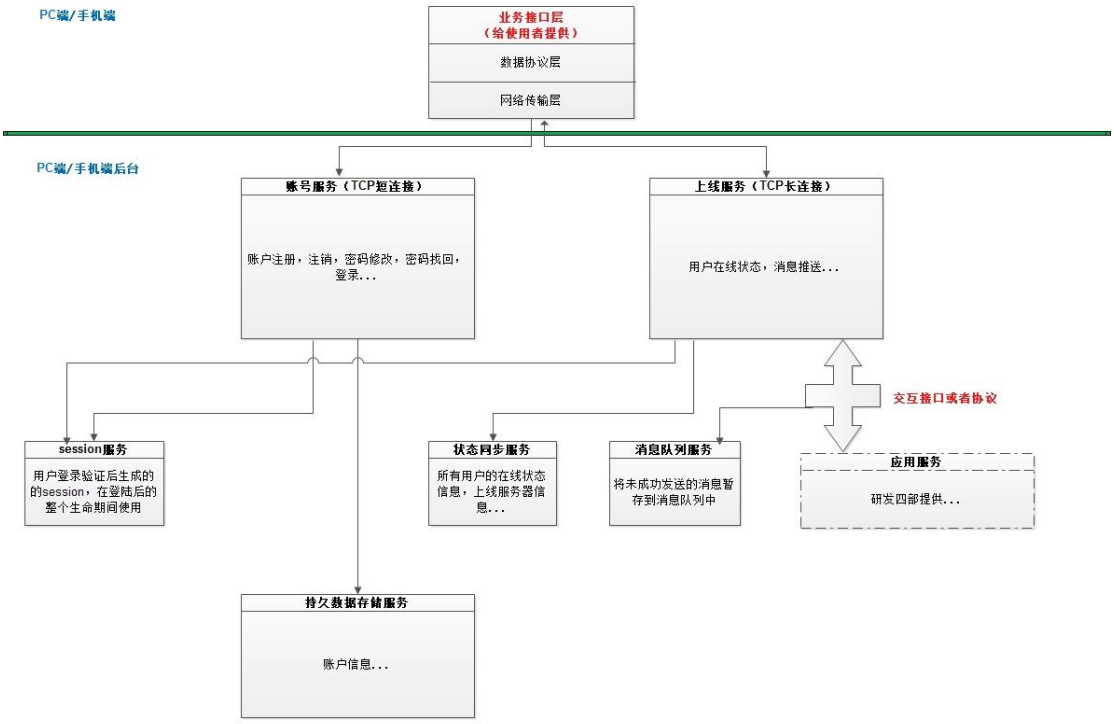
架构考量：

- 客户端与服务端的通信协议：采用可靠传输协议 TCP（短连接实现业务交互，长连接实现上线、信息推送）；

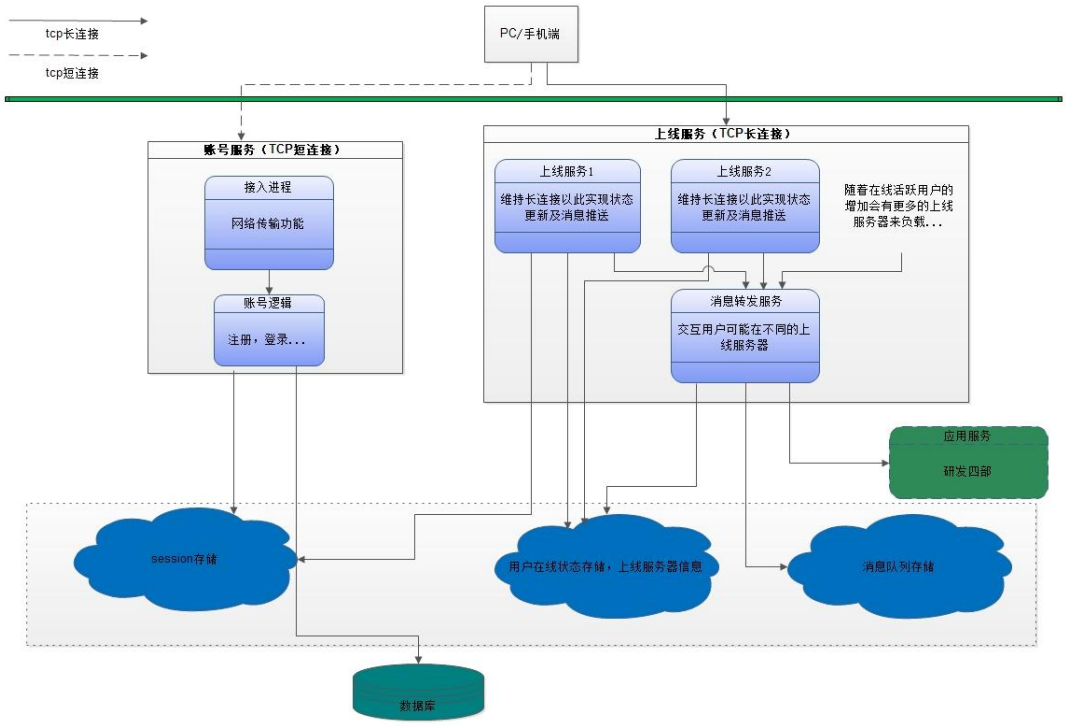
- 对于负载的充分考虑：后台架构的灵活性、可扩展性，支持分布式部署——把网络层、业务逻辑层、数据层分离，网络层和业务层支持负载均衡策略、数据层支持分布式存储。

- 提供的开发 SDK 的易用性：尽量减少与协作部门业务服务的交叉，提供统一、易用的业务接口或者公共的协议。

以下是支持 PC 端、移动端的架构设计图


PC端/手机端
业务接口层
（给使用者提供）
数据协议层
网络传输层
PC端/手机端后台
账号服务（TCP短连接）
账户注册，注销，密码修改，密码找回，登录...
上线服务（TCP长连接）
用户在线状态，消息推送...
交互接口或者协议
session服务
用户登录验证后生成的的session，在登陆后的整个生命期间使用
状态同步服务
所有用户的在线状态信息，上线服务器信息...
消息队列服务
将未成功发送的消息暂存到消息队列中
应用服务
研发四部提供...
持久数据存储服务
账户信息...

架构示意图


tcp长连接
tcp短连接
PC/手机端
账号服务（TCP短连接）
接入进程
网络传输功能
账号逻辑
注册，登录...
上线服务（TCP长连接）
上线服务1
维持长连接以此实现状态更新及消息推送
上线服务2
维持长连接以此实现状态更新及消息推送
随着在线活跃用户的增加会有更多的上线服务器来负载...
消息转发服务
交互用户可能在不同的上线服务器
应用服务
研发四部
session存储
用户在线状态存储，上线服务器信息
消息队列存储
数据库

架构细化图

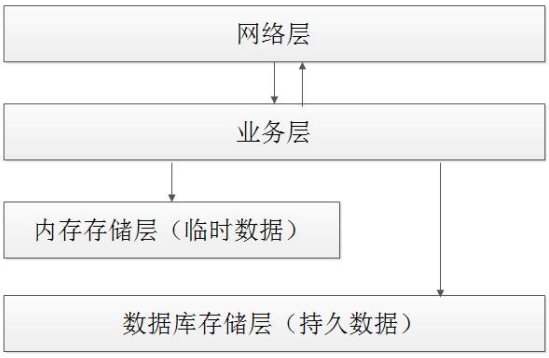
针对部分重点、难点进行说明：

- 与协作部门的协作开发问题：从<架构示意图>中可以看出，协作部门需要关注的仅仅是标红的客户端业务接口和服务端标红的交互接口。

- 后期可能涉及到的海量用户并发上线问题：
  - 从<架构细化图>中可以看出对于上线服务由于建立的是 TCP 长连接，对于单台服务器往往由于硬件资源、系统资源、网络资源的限制无法做到海量用户的同时在线，所以设计为根据服务器负载支持多服务器上线，同时由于多服务器上线造成了对整个系统交互（不同的客户端的交互，协作部门应用服务和客户的交互）的分割，引入消息转发服务器作为粘合点。另外对于多服务器上线造成的统一账户信息（在线状态，消息）数据的分割，引入统一的数据层（内存存储层：session、状态信息存储、消息队列存储；数据库：账号信息存储）做到业务和数据的分离，也就做到了支持分布式部署。

  - 对于账户服务：做到网络层、业务层、数据层的完全分离。首先对于 TCP 短连接来说不会如长连接那般消耗资源，即使后期遇到海量的并发访问请求依然可以从容的通过负载均衡策略和数据分布式部署策略进行解决。

根据以上的描述解析，可以把服务端架构进一步简化表现为下图所示：



服务端平台及技术选型：

- 系统开发平台：CentOS——Linux 发行版的一种，稳定可靠、可定制优化、支持丰富；
- 网络支撑层：libevent——减小开发成本，增强稳定性；
- 内存存储层：Redis——支持丰富的存储结构，支持分布式存储；
- 数据库：MySQL——最适合互联网的数据库，免授权、高效稳定、可控性高；

✧ 开发语言： C/C++

热点问题考量：

- 系统性能考量：
  - 编码角度：采用高效的网络模型，线程模型，I/O 处理模型，合理的数据库设计和操作语句的优化；
  - 垂直扩展：通过提高单服务器的硬件资源或者网络资源来提高性能；
  - 水平扩展：通过合理的架构设计和运维方面的负载均衡策略将负载分担，有效提高性能；
  - 后期甚至可以考虑加入数据缓存层，突破 IO 瓶颈；

- 系统的高可用性：（防止单点故障）
  - 在架构设计时做到业务处理和数据的分离，从而依赖分布式的部署使得在单点故障时能保证系统可用。
  - 对于关键独立节点可以采用双机热备技术进行切换。
  - 数据库数据的安全性可以通过磁盘阵列的冗余配置和主备数据库来解决。

- 账号同步问题：
  - 对于一个设计良好的 MySQL 数据库来说单库百万数据是几乎没有压力的，所以理想情况下是把所有账号存储在一个集中式的数据库中，通过集中式的性能提高方案来解决用户访问慢的问题。但是针对可能出现的跨运营商网络、跨国网络的访问非常缓慢的问题，当出现多 IDC 时就必须要考虑数据的同步问题，可以抽象为分布式缓存 Redis，MySQL 数据库的同步问题，相应的他们也都有成熟的数据同步方案，所以后期更多是运维方面的考虑。

技术把控的考虑：

- 所采用技术的通用性，稳定性：CentOS，libevent，Redis，MySQL 都是业界成熟稳定的技术；在开源世界、商业上也多有应用；也没有授权的忧虑；有关于这些技术的文档、书籍也很丰富；中英文讨论组也很活跃。在有能力的技术人员的掌控下不用担心后期的技术风险。

- 技术的把控力：在研究院过去一年多的平台建设中均采用了以上技术方案，已经具备一定的经验和生产力。

- 系统后期要投入的力量：对于高可用性、分布式、负载均衡、同步处理除去架构的支持外，最主要的是运维的支持，在这方面目前现有成员能掌握的是集中式的实际经验，分布式部署等的理论经验，后期会在条件具备的情况下进行实践，同时希望增加其他有实践经验的人员投入。