移动端企业IM系统优化

发表于 19小时前 | 615次阅读 | 来源 CSDN | 9 条评论 | 作者 樊星，金景，龙远智

云计算 SaaS IM imo

摘要：本文主要描述了在移动端企业IM实践过程中，对消息可靠性，时序，海量组织架构，以及语音等的优化和技术架构打造关键点。

imo在PC端IM领域有很强的积累，但在做移动端时遇到了不少的挑战。在移动端相对恶劣的运行环境加上企业IM的特殊性(高及时性，大数据量)，使得许多之前行之有效的经验水土不服，引发了若干问题，通过一些系统重构以及针对性的定位处理，问题得到了解决，本文重点介绍我们遇到的这些问题以及相应的处理经验，希望对大家有帮助。

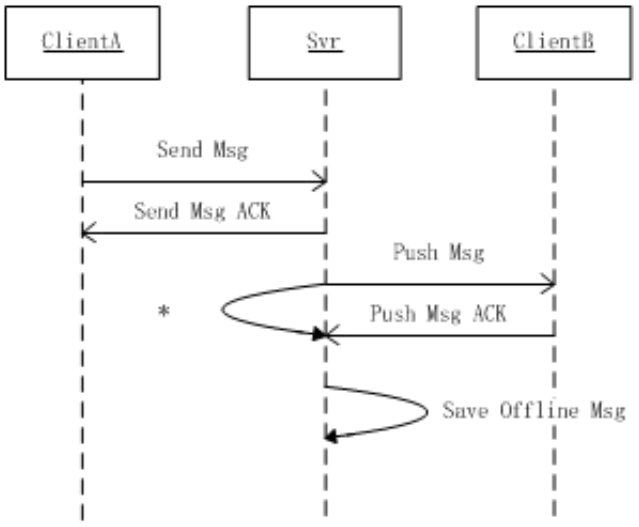
我们遇到的问题：

1. 消息收发，会出现丢消息，消息收取不及时，消息乱序等情况。
2. 巨型组织架构的更新缓慢，失败率高
3. 语音消息上传，下载失败率高，体验差
4. 内嵌web应用的速度比较慢，体验比较差

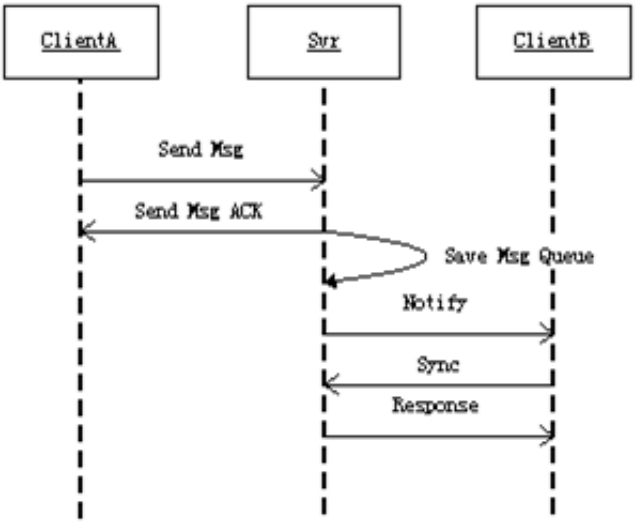
针对以上我们我们进行了逐一分析定位，重新梳理架构，流程，问题全部得到了解决，移动端IM的可靠性体验基本达到微信级别。

一. 移动端消息机制优化--->mobile环境下消息收发机制(消息传递可靠性，时序保证)

1. 传统pc端解决方案



2. Mobile环境下优化架构

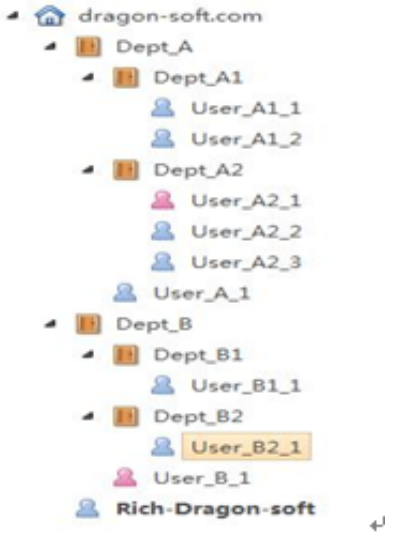


参考上面两张图，传统的PC IM架构中，由Server端从发送端ClientA收到消息后，主动将消息推送给接收端ClientB，这种方式在PC环境下行之有效，推送成功率可以达到90%，推送失败的记录离线消息，客户端会在重登录时去取离线消息。在这里的整个流程里面，server端一直处于主动角色，客户端则处于被动角色，业务逻辑有server端根据情况进行判定和处理。

但当我们把场景切换到移动端时，一切都发生了变化，我们发现，服务器主动push成功率极低，因为在服务器push时，移动客户端很少处于tcp稳定连接状态，可能处于网络不稳定，app在后台，打电话等各种场景下。针对这种情况，我们思考，如何解决解决这么多种情况，难道要针对每种情况做判断处理？这样做的话，基本是个无底洞。深入分析后，我们发现，问题的症结在于，我们应该设法从架构上规避这些场景的特殊性处理，如何做呢？我们借助有关文档，参考微软的ActiveSync机制，将消息机制从服务端push改为了客户端pull，在这种模式下，server只需要将消息高效的保存下来，客户端可以选择在任何时间点，拉取任何数量的消息，这样，我们从根本逻辑上排除了丢消息的可能性。同时，我们将心跳和notify结合起来，告知客户端在服务器上是否有新消息，也解决了消息及时性的问题。

在上面优化的基础上，我们对消息进行全局编号，全局的且有序的消息id既作为上面Sync机制的同步基准，同时也作为消息去重，以及消息排序的依据，排除了消息重复以及乱序的可能性。

至此，在新的消息机制的支持下，我们的移动端IM做到了100%的消息可靠性，健壮性和可用性得到了本质的提高。



二. 海量组织架构同步(1W以上) --- 通过足够细的UC版本控制，差量数据发送，以聚沙成塔，小粒度蚂蚁搬家的方式逐步拉到需要的数据

这个环节是企业IM独有的特殊场景。根据统计，99%的人的通讯录联系人数量在500人一下，在这个量级下，常见的个人端产品在这块并不需要花太大的精力。而企业IM环境下，1000人只是个起点，大的企业需要能支持1万人以上，甚至10万人以上的组织架构，在这个量级下，传统的解决方法基本无解。另外，这个问题我们基本没有同类产品的解决方法可以参考，只能自己探索。

我们面临的主要问题是，巨型组织架构的更新缓慢，失败率高。通过具体分析，我们发现，核心的问题是组织架构中少量信息的变化，会引起全局的变动，从而导致海量的更新数据检查，而在移动环境下，基本不会有充足的带宽和运行时间，去完成这个流程，所以，导致更新很难成功。

思路：

针对这种情况，我们发现，类似的场景其实在SVN管理大型代码库时同样会遇到，但svn却很好的解决了问题。所以，我们分析了svn的解决方法，参考它的思路，在Server保留所有版本(3个月内)的，各个粒度的版本diff结果。客户端只拉取特定粒度的diff数据，并进行数据合并。

具体流程如下：

服务器为每个（Dept）部门节点在后台维护整个版本列表，每次对部门节点的修改（增删子部门和部门成员)后的数据提交，都会导致部门原先的数据被保存为一个历史版本，并且该版本对应于修改前的uc；

当客户端用本地保存的某部门uc向服务器同步该部门数据时, 服务器比较客户端传上来的uc和该部门当前最新的uc, 如果uc不同，则计算两个版本之间的差量数据，并把这些差量数据传回给客户端；客户端收到后，将差量数据和原先保存的该部门数据一起进行计算，得出该部门最新数据，更新本地数据库数据和界面；

客户端可以选择性的更新特定dept，不同dept之间没有强依赖。

通过上面的方法，无论组织架构有多大，无论什么样的更新，我们的移动客户端几乎都可以顺利的完成更新，

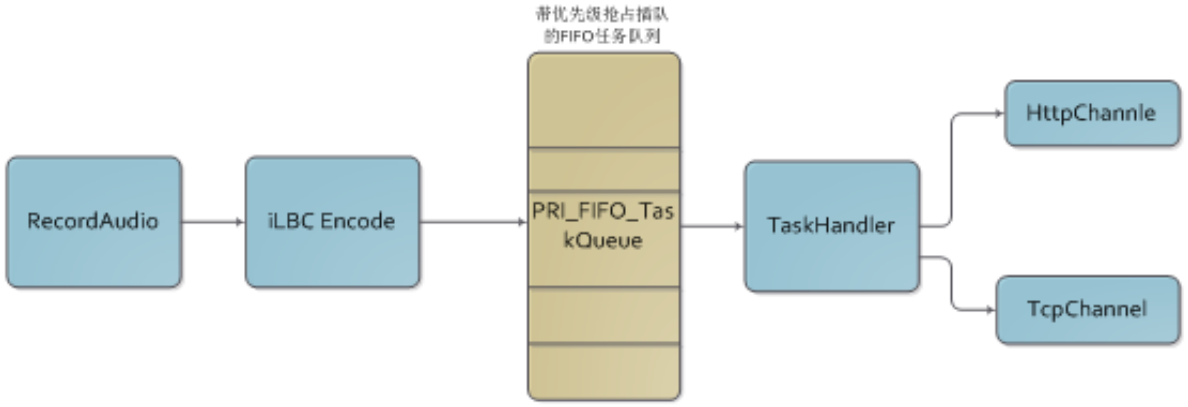
基本解决了企业IM环境下海量组织架构带来的问题。

三. 语音优化

经典IM的解决方法，把语音作为2进制文件处理，先录音，存为文件，再通过http上传。

第一版按这个方案做的，语音上传失败率高，尤其在进行连续语音发送时, 经过分析我们定位到下面一些点：

1. 语音采用系统内置的amr编码，对于voice message 而言，码率过高，我们的一分钟语音的数据量偏大。
2. 连续语音发送时，存在多个上行通道，对带宽抢占过大
3. http上传，在移动端，ios，android提供的http库不能对http长连接进行可靠的保持，大多数情况下，http请求都要新建tcp连接。
4. 单次tcp连接建立代价高
5. 用户发完语音后会很快退出对话界面或者将应用切入后台，app的可执行时间偏短



语音任务流程图

针对上面的点，我们分别从编解码，流程，传输通道三个方面进行了优化：

a. 编解码：经过测试，对比，综合音质和数据量的考虑，选用了iLBC作为跨平台语音编码，iLBC为通话语音做了专门的优化，非常适合窄带环境(移动端)语音通信，qq的超级语音的编解码就是以这个为基础的，google在webrtc里面对iLBC编码做了开源。iLBC使我们的语音能在相对保证音质的情况下实现较小的数据量。

b. 流程：原有流程会出现多通道抢占带宽，这个情况类似高速公路，大家一起抢着跑的结果是大家都跑不了。移动端的带宽在gprs仅有几K，这样的带宽仅能勉强撑起一路的上传。针对这个情况，我们设计了一个带优先级任务队列，在同一时刻保证只有一个任务在上传，按照FIFO原则进入队列，同时提供优先级插队能力。如上面的流程图，这样基本解决了通道阻塞情况，只要有网络，上传任务迟早都能完成。

c. 通道：在GPRS环境下一次TCP连接建立的平均时间是5s，这样的代价我们无法承受，但我们同时发现，我们进行IM信令通讯的的TCP长连接通道非常稳定。鉴于此，我们尝试将上传任务通过IM的TCP、长连接来执行，同时采取边录边传的方式。通过AB Test，我们发现这种优化非常有效，语音任务的单次完成率有了很大的提升。同时，语音的数据量相对并不大，所以，也没有对我们的信令通道产生大的影响。另外，考虑到一些特殊情况，我们仍然保留了http通道，由TaskHandler根据实际的业务场景来进行选择。

结束语：

除了上面的几个方面之外，在具体实践过程中，还做了很多细节的优化，限于篇幅，这里就不一一赘述。Imo的移动客户端在经过上述优化之后，稳定性，可靠性大幅提升，形成了一个比较夯实的基础，已经逐渐体现出替代和补充PC客户端的趋势。

移动端的技术日新月异，林林总总的坑也无数，希望上面的优化经验能对大家有所帮助。