荔枝架构实践与演进历程

原创：黄全 IT168企业级 1月24日

2018年10月18日,黄全 受邀参加了由IT168主办的《SACC 2018第十届系统架构师大会》,并发表了精彩演讲，以下内容根据 SACC大会实录整理。

黄全，荔枝APP架构师。拥有10年的互联网开发经验，对分布式系统、高并发解决方案有着丰富的实践经验，在国内知名互联网企业担任过资深工程师、系统架构师等职。曾就职于UC浏览器、春笋新科技。现任荔枝资深工程师，负责基础架构的设计与开发，目前专注于分布式系统、微服务、数据库中间件等技术的研究与探索。

如果你对声音互动平台有所了解，那对于荔枝APP一定不会感到陌生!

荔枝，致力于打造声音处理平台，帮助人们展现自己的声音才华。荔枝集录制、编辑、存储、收听、分享于一体，依托声音底层技术积淀，具有声音节目录制功能，可在手机内完成录音、剪辑、音频上传和语音直

播。

简单理解，荔枝APP上有很多主播，主播和用户之间可以通过声音互动。目前，荔枝APP月均活跃用户达到好几千万，月均活跃主播达到好几百万，全球注册用户和音频节目数量都已过亿。

那么，对于有着大用户量的社交APP来说，荔枝APP的背后架构该如何设计?一个时间轴可以概括整个架构的演进过程。

架构演进时间轴

- **2013年：单体架构**

这个时候的架构是V1.0版,主要特点是APP 直连服务器，服务端是单体架构。也就是说，一个服务，一个存储解决所有问题。这种架构看上去简单、粗暴，好处是快速上线、快速响应市场需求;但劣势也非常明显，APP直连服务器模式在扩展的时候非常不灵活。项目上线后3个月，用户数突破 100万，访问量上涨，服务器压力增大。

- **2014年：垂直架构**

到了2014年，荔枝APP的后台架构演进到V2.0版。这一版架构的特点是：支持水平扩展和功能拆分。首先，APP 与 app server 之间加入app代理层，用于分发请求给多台 app server，分摊压力。其次，对app server 按功能进行拆分，数据操作及业务逻辑部分由后端服务负责，并采用 netty(http) + json 方式进行交互。

虽然，这个时候已经能做到相对快速交互，但是依然存在很多问题。一个是，后端服务水平扩展时，不能动态化;另外，系统间采用 http 交互时，通讯效率较低，并且数据包较大;还有一个问题是，json 解析速度较慢、体积较大。所以，V3.0版采取了一系列措施，重点解决扩展、交互等问题。比如：引入Linux虚拟服务器集群系统 LVS，采用 TCP 取代 HTTP，通过定制私有协议取代json。

V3.0版使用 LVS 集群解决了分发请求，但是随着业务的快速发展，人力资源都投入在业务开发上，对第三方产品了解也不够深入，导致运维成了最大挑战。所以，后期考虑采用自己开发代理服务来取代 LVS。

V4.0版架构中，我们开发了代理服务，当时使用 VIP 与其他服务连接，取代LVS分发请求。这时候的整体架构依然比较简单，app server 与后端服务还是单体架构，没有做业务拆分，虽然能让后端服务支持水平扩展功能，但需要重启代理服务。另外，还有一个挑战是，随着用户量、访问量的持续上涨，系统访问压力依然很大。接下来的目标是，app server 与后端服务需要按业务垂直拆分。

演化到V5.0版架构的时候，所有服务已经能够按业务拆分，可支持水平扩展，整体架构能抗得住一定的访问压力。2014年10月，用户量曾突破 1千万。但新的问题又来了：一个是服务的配置不能做到热更;另外，不同业务的后端服务之间产生了交互需求。还有，微服务化已成为主流发展方向。所以，下一个阶段的解决方案是，开发配置中心 config server实现配置热更;采用开发分布式服务框架 lz-RPC，封装远程调用功能。

- **2015年：分布式架构**

V6.0版架构开启了分布式架构新征程，这时候的特点是app server、后端服务、代理层等，都实现了配置热更，能灵活水平扩展。到2015年9月，用户量曾突破 5千万。但是面临的问题依然很多。比如：mysql、redis 操作的重复代码太多;mysql、redis 慢操作不能及时报警;各个服务的数据源配置分散，难以管理等。另外，还涉及跨机房数据操作和数据同步问题。而分布式数据库中间件可以很好地解决这些问题。

- **2016年：分布式数据库中间件**

从2016年开始，荔枝APP的后台架构走入V7.0版时代。这时，开发团队自研了分布式数据库中间件data store服务。data store的特点是：简单易用，可减少重复代码。只需要在类上加上注解，就可以实现与数据库的交互、数据转换等功能，大大减少了开发的工作量。另外，data store具有自动维护缓存和数据库表的对应关系、自动维护缓存与数据库数据的一致性的功能。最重要的是，屏蔽了服务对数据源的管理，便于数据库的迁移和扩容等操作。

总体来看，V7.0版的最大特点是，已形成一个比较完整的分布式架构。data store 封装了常见的 mysql、redis 操作，能提供慢操作监控。后期根据业务发展，还引入了 kafka、mongoDB、zookeeper、hbase等多种第三方产品。

随着应用的增加，V7.0版架构也逐渐暴露出了一些缺陷。一是资源监控、业务监控、分布式跟踪链等功能不完善。另外，随着访问量上涨，分布式服务框架的功能也需要进行扩展。

- **2017-2018年：监控体系**

进入2017年以后，整个架构已趋于完善，重点引入第三方产品，建立监控体系，完善对服务器资源、业务、跟踪链路等的监控，同时也扩展了分布式服务框架功能。也是从这个时候开始，整个架构迎来了V8.0版。经过完善后，业务监控及基础监控功能已比较完整，分布式服务框架扩展了接口缓存、熔断、降级、过载保护等功能。

近两年踩过的“坑”以及应对措施

1、大主播开直播，访问量爆涨，影响了其他直播间的直播效果，比如出现卡顿、进入不了直播间、接口超时。举个例子：李易峰晚上8点在荔枝APP上做直播，那么从8点前开始，整个系统的访问量就会比平时要高

出很多。有用户就会出现进入直播间慢、评论出现慢以及其他体验不好的情况出现。像这样的问题，应该如何解决呢?

第一个方案，也是最简单的方法，是“隔离”。在 data store 中，针对 redis 存储开发，按前缀分片的功能，对大主播的直播数据进行隔离，避免影响其他主播的直播效果。

第二个方案是，在高访问量期间，结合分布式服务框架中开发的熔断、降级、过载保护等功能，采取对部分非关键服务做降级措施，避免服务器因访问量过高发生雪崩。

2、在高并发环境下，Mysql 查询性能成为瓶颈。当数据量呈现爆发式增长，Mysql 查询速度变慢。我们对分布式数据库中间件作了扩展，在操作mysql时，在数据库上层加入缓存memcached后，大大提高了查询性能,并且自动维护缓存和数据库数据的一致性。

3、访问量上涨，受日志文件的IO影响，服务出现长GC(stop the world，阻塞业务线程)。类似服务出现长GC的问题，很多互联网公司都会遇到。在GC的整个回收过程中，会有两个步骤涉及到IO操作。第一个操作就是写 perf 文件;第二个是写 gc log的时候。一台服务器一般会部署多个服务，这些服务在运行的过程中，会不断输出日志，这时容易出现与其他服务的GC线程发生抢占IO资源的冲突，而导致GC线程阻塞等待，最终导致整个GC过程耗时较长，影响了服务的响应和稳定。为了解决这些问题，采取了两个方案来解决：第一，不生成 perf 文件，在服务启动脚本里，加上参数 -XX:+PerfDisableSharedMem就可以了;第二，将 GC 日志保存到内存盘中(tmpfs 或 ramfs)，在服务启动脚本里加上 -Xloggc:/dev/shm/lz-app-gc.log参数就可以解决了。

4、随着业务的发展，系统的整体访问量越来越大，后端服务接口调用耗时越来越长，导致经常超时。经过分析和统计发现，整个平台实际上以“读多写少”的场景居多。有没有一个兼顾全局的解决方案呢?在分布式服务框架中开发“缓存接口”功能，解决了这个问题。其实有很多场景，我们是不需要实时看到最新数据的，即使新数据晚了30秒或者1分钟用户才看到，也是可以接受的。

5、系统间异步消息通知功能不完善。之前，是通过redis来做异步消息通知，好处是比较轻量化，但是随着数据量增加，大数据传输增多，出现多个消费方需要消费相同消息的时候，redis 就不是很适用了。这时，使用 kafka可以满足系统间消息通知、大数据量传输、多个消费者消费相同消息的场景。

6、当服务框架中的各种功能都比较完善后，却发现缺少一个报警功能。比如在请求失败/超时/异常等，如果有监控机制，就可以找到具体的问题点。借助监控系统，我们可以看到服务器负载、物理内存、swap、磁盘等信息，也能监控到GC信息的回收时间、次数以及JVM堆信息等，还可以对异常请求进行统计。

7、随着服务的增多，每个服务都有很多实例，导致整个架构的调用链路不清晰，也不能预知整体架构存在的瓶颈。引入skywalking 实现调用链跟踪功能后，能快速定位到线上故障和整个架构的性能瓶颈。

8、主要是更新服务的问题，上线/重启服务操作很原始，之前都是人工在本地打包，再上传到服务器。服务不多的时候还能支撑，但是服务实例数量开始增多的情况下，这种方式就需要改进。荔枝的做法是开发一个

自动发布平台，一键式操作。另外，就是通过jenkins + gitlab，接入自动发布平台，实现自动打包、一键发布。

9、服务发布流程不够规范。过去的部署流程很简单，先是在本地测试，测试通过后，打包部署到线上，再观察服务的运行日志。但是随着团队人员和系统越来越多后，这种做法是不合适的。要想保证整个业务顺利上线，必须把服务发布流程规范化。从预发布测试到影响评估，到回滚步骤，再到灰度发布、线上验证，每一个环节都要标准化。预发布测试包含业务流程测试、新功能测试、SQL 验证、代码审查;影响评估包含对业务系统的影响和对交互系统的影响;回滚方案包含回滚步骤和回滚版本号，灰度发布则要按照按流量百分比、按设备类型和按 app 版本号等来操作;线上验证要做功能回归测试，以及观察异常日志、报警信息等。

10、研发规范不够标准。一个技术团队从10几个人发展到几百人甚至上千人的时候，规范很重要。为了提高效率，公司制定了各种标准的开发/操作规范，包括客户端开发规范、服务端开发规范、测试规范、运维规范、mysql、redis、kafka、mongoDB 等的使用规范。

未来对整个架构会有多个优化的目标，例如通过“微服务+容器化”实现服务实例的动态扩容与缩容、Service Mesh架构改造、业务级别的调用链跟踪功能等等。

最后，引用大家常说的一句话：好的系统不是设计出来的，而是演进出来的。未来，荔枝的系统架构会更加完善，在不断探索中更加精进。

- End -

作者：黄全

编辑：李代丽