【编者按】近日，InfoQ专访了阿里巴巴的研究员林昊（花名毕玄），了解了他们的数据中心异地多活项目的来龙去脉。本文在微博上引起了很多讨论。新浪微博的高级技术经理刘道儒（@liudaoru）也总结了微博平台的一些经验。

正文

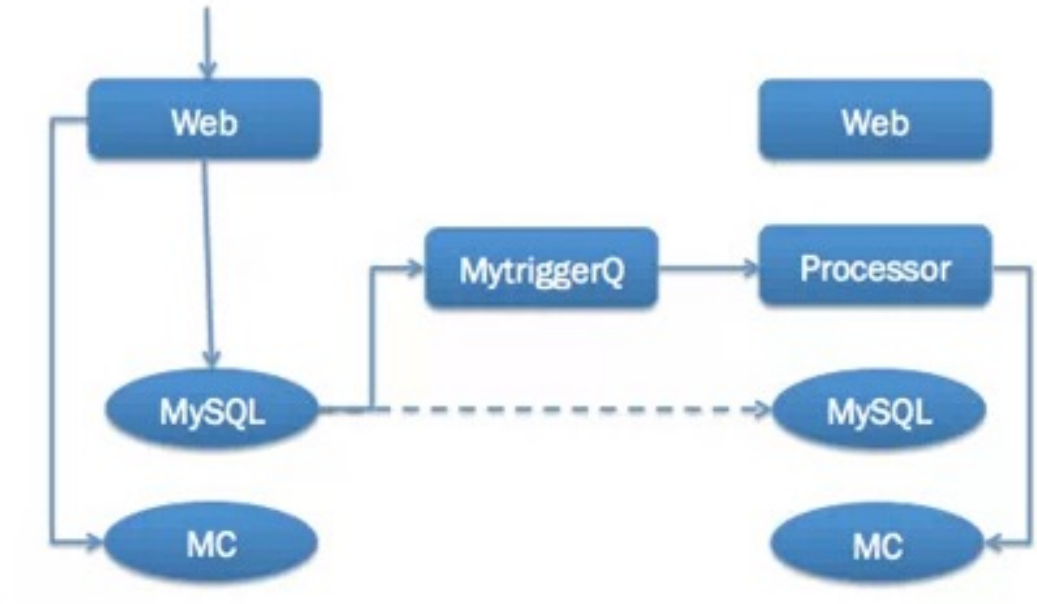
异地多活的好处阿里巴巴的同学已经充分阐述，微博的初始出发点包括异地灾备、提升南方电信用户访问速度、提升海外用户访问速度、降低部署成本（北京机房机架费太贵了）等。通过实践，我们发现优势还包括异地容灾、动态加速、流量均衡、在线压测等，而挑战包括增加研发复杂度、增加存储成本等。

微博外部历程

先说说微博外部的历程，整个过程可谓是一波多折。微博的主要机房都集中在北京，只有很小一部分业务在广州部署，2010年10月，因微博高速发展，所以准备扩大广州机房服务器规模，并对微博做异地双活部署。

第一版跨机房消息同步方案采取的是基于自研的MytriggerQ（借助MySQL从库的触发器将INSERT、UPDATE、DELETE等事件转为消息）的方案，这个方案的好处是，跨机房的消息同步是通过MySQL的主从完成的，方案成熟度高。而缺点则是，微博同一个业务会有好几张表，而每张表的信息又不全，这样每发一条微博会有多条消息先后到达，这样导致有较多时序问题，缓存容易花。

第一套方案未能成功，但也让我们认识到跨机房消息同步的核心问题，并促使我们全面下线MytriggerQ的消息同步方案，而改用基于业务写消息到MCQ（MemcacheQ，新浪自研的一套消息队列，类MC协议）的解决方案。



2011年底在微博平台化完成后，开始启用基于MCQ的跨机房消息同步方案，并开发出跨机房消息同步组件WMB（Weibo Message Broker）。经过与微博PC端等部门同学的共同努力，终于在2012年5月完成Weibo.com在广州机房的上线，实现了“异地双活”。

由于广州机房总体的机器规模较小，为了提升微博核心系统容灾能力，2013年年中我们又将北京的机房进行拆分，至此微博平台实现了异地三节点的部署模式。依托于此模式，微博具备了在线容量评估、分级上线、快速流量均衡等能力，应对极端峰值能力和应对故障能力大大提升，之后历次元旦、春晚峰值均顺利应对，日常上线导致的故障也大大减少。上线后，根据微博运营情况及成本的需要，也曾数次调整各个机房的服务器规模，但是整套技术上已经基本成熟。

异地多活面临的挑战

根据微博的实践，一般做异地多活都会遇到如下的问题：

- 机房之间的延时：微博北京的两个核心机房间延时在1ms左右，但北京机房到广州机房则有近40ms的延时。对比一下，微博核心Feed接口的总平均耗时也就在120ms左右。微博Feed会依赖几十个服务上百个资源，如果都跨机房请求，性能将会惨不忍睹；
- 专线问题：为了做广州机房外部，微博租了两条北京到广州的专线，成本巨大。同时单条专线的稳定性也很难保障，基本上每个月都会有或大或小的问题；
- 数据同步问题：MySQL如何做数据同步？HBase如何做数据同步？还有各种自研的组件，这些统统要做多机房数据同步。几十毫秒的延时，加上路途遥远导致的较弱网络质量（我们的专线每个月都会有或大或小的问题），数据同步是非常大的挑战；
- 依赖服务部署问题：如同阿里巴巴目前只做了交易单元的“异地双活”，微博部署时也面临核心服务过多依赖小服务的问题。将小服务全部部署，改造成本、维护成本过大，不部署则会遇到之前提到的机房之间延时导致整体性能无法接受的问题；
- 配套体系问题：只是服务部署没有流量引入就不能称为“双活”，而要引入流量就要求配套的服务和流程都能支持异地部署，包括预览、发布、测试、监控、降级等都要进行相应改造。

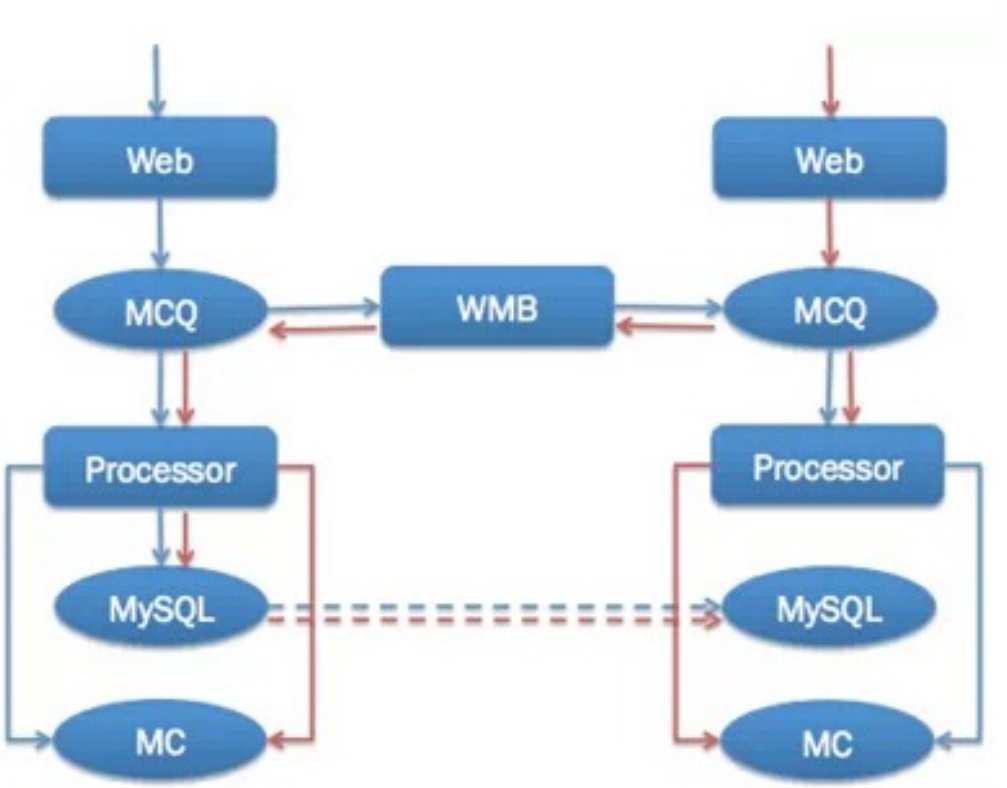
微博异地多活解决方案

由于几十毫秒的延时，跨机房服务调用性能很差，异地多活部署的主体服务必须要做数据的冗余存储，并辅以缓存等构成一套独立而相对完整的服务。数据同步有很多层面，包括消息层面、缓存层

面、数据库层面，每一个层面都可以做数据同步。由于基于MytriggerQ的方案的失败，微博后来采取的是基于MCQ的WMB消息同步方案，并通过消息对缓存更新，加上微博缓存高可用架构，可以做到即便数据库同步有问题，从用户体验看服务还是正常的。

这套方案中，每个机房的缓存是完全独立的，由每个机房的Processor（专门负责消息处理的程序，类Storm）根据收到的消息进行缓存更新。由于消息不会重复分发，而且信息完备，所以MytriggerQ方案存在的缓存更新脏数据问题就解决了。而当缓存不存在时，会穿透到MySQL从库，然后进行回种。可能出现的问题是，缓存穿透，但是MySQL从库如果此时出现延迟，这样就会把脏数据种到缓存中。我们的解决方案是做一个延时10分钟的消息队列，然后由一个处理程序来根据这个消息做数据的重新载入。一般从库延时时间不超过10分钟，而10分钟内的脏数据在微博的业务场景下也是可以接受的。

微博的异地多活方案如下图（三个节点类似，消息同步都是通过WMB）：



跟阿里巴巴遇到的问题类似，我们也遇到了数据库同步的问题。由于微博对数据库不是强依赖，加上数据库双写的维护成本过大，我们选择的方案是数据库通过主从同步的方式进行。这套方案可能的缺点是如果主从同步慢，并且缓存穿透，这时可能会出现脏数据。这种同步方式已运行了三年，整体上非常稳定，没有发生因为数据同步而导致的服务故障。从2013年开始，微博启用HBase做在线业务的存储解决方案，由于HBase本身不支持多机房部署，加上早期HBase的业务比较小，且有单独接口可以回调北京机房，所以没有做异地部署。到今年由于HBase支撑的对象库服务已经成为微博非常核心的基础服务，我们也在规划HBase的异地部署方案，主要的思路跟MySQL的方案类似，同步也在考虑基于MCQ同步的双机房HBase独立部署方案。

阿里巴巴选择了单元化的解决方案，这套方案的优势是将用户分区，然后所有这个用户相关的数据都在一个单元里。通过这套方案，可以较好的控制成本。但缺点是除了主维度（阿里巴巴是买家维度），其他所有的数据还是要做跨机房同步，整体的复杂度基本没降低。另外就是数据分离后由于

拆成了至少两份数据，数据查询、扩展、问题处理等成本均会增加较多。总的来讲，个人认为这套方案更适用于WhatsApp、Instagram等国外业务相对简单的应用，而不适用于国内功能繁杂、依赖众多的应用。

数据同步问题解决之后，紧接着就要解决依赖服务部署的问题。由于微博平台对外提供的都是Restful风格的API接口，所以独立业务的接口可以直接通过专线引流回北京机房。但是对于微博Feed接口的依赖服务，直接引流回北京机房会将平均处理时间从百毫秒的量级直接升至几秒的量级，这对服务是无法接受的。所以，在2012年我们对微博Feed依赖的主要服务也做了异地多活部署，整体的处理时间终于降了下来。当然这不是最优的解决方案，但在当时微博业务体系还相对简单的情况下，很好地解决了问题，确保了2012年5月的广州机房部署任务的达成。

而配套体系的问题，技术上不是很复杂，但是操作时却很容易出问题。比如，微博刚开始做异地多活部署时，测试同学没有在上线时对广州机房做预览测试，曾经导致过一些线上问题。配套体系需要覆盖整个业务研发周期，包括方案设计阶段的是否要做多机房部署、部署阶段的数据同步、发布预览、发布工具支持、监控覆盖支持、降级工具支持、流量迁移工具支持等方方面面，并需开发、测试、运维都参与进来，将关键点纳入到流程当中。

关于为应对故障而进行数据冗余的问题，阿里巴巴的同学也做了充分的阐述，在此也补充一下我们的一些经验。微博核心池容量冗余分两个层面来做，前端Web层冗余同用户规模成正比，并预留日常峰值50%左右的冗余度，而后端缓存等资源由于相对成本较低，每个机房均按照整体两倍的规模进行冗余。这样如果某一个机房不可用，首先我们后端的资源是足够的。接着我们首先会只将核心接口进行迁移，这个操作分钟级即可完成，同时由于冗余是按照整体的50%，所以即使所有的核心接口流量全部迁移过来也能支撑住。接下来，我们会把其他服务池的前端机也改为部署核心池前端机，这样在一小时内即可实现整体流量的承接。同时，如果故障机房是负责数据落地的机房，DBA会将从库升为主库，运维调整队列机开关配置，承接数据落地功能。而在整个过程中，由于我们核心缓存可以脱离数据库支撑一个小时左右，所以服务整体会保持平稳。

异地多活最好的姿势

以上介绍了一下微博在异地多活方面的实践和心得，也对比了一下阿里巴巴的解决方案。就像没有完美的通用架构一样，异地多活的最佳方案也要因业务情形而定。如果业务请求量比较小，则根本没有必要做异地多活，数据库冷备足够了。不管哪种方案，异地多活的资源成本、开发成本相比与单机房部署模式，都会大大增加。

以下是方案选型时需要考虑的一些维度：

- 能否整业务迁移：如果机器资源不足，建议优先将一些体系独立的服务整体迁移，这样可以为核心服务节省出大量的机架资源。如果这样之后，机架资源仍然不足，再做异地多活部署。
- 服务关联是否复杂：如果服务关联比较简单，则单元化、基于跨机房消息同步的解决方案都可以采用。不管哪种方式，关联的服务也都要做异地多活部署，以确保各个机房对关联业务的请求都落在本机房内。
- 是否方便对用户分区：比如很多游戏类、邮箱类服务，由于用户可以很方便地分区，就非常适合单元化，而SNS类的产品因为关系公用等问题不太适合单元化。
- 谨慎挑选第二机房：尽量挑选离主机房较近（网络延时在10ms以内）且专线质量好的机房做第二中心。这样大多数的小服务依赖问题都可以简化掉，可以集中精力处理核心业务的异地多活问题。同时，专线的成本占比也比较小。以北京为例，做异地多活建议选择天津、内蒙古、山西等地的机房。

- 控制部署规模：在数据层自身支持跨机房服务之前，不建议部署超过两个的机房。因为异地两个机房，异地容灾的目的已经达成，且服务器规模足够大，各种配套的设施也会比较健全，运维成本也相对可控。当扩展到三个点之后，新机房基础设施磨合、运维决策的成本等都会大幅增加。
- 消息同步服务化：建议扩展各自的消息服务，从中间件或者服务层面直接支持跨机房消息同步，将消息体大小控制在10k以下，跨机房消息同步的性能和成本都比较可控。机房间的数据一致性只通过消息同步服务解决，机房内部解决缓存等与消息的一致性问题。跨机房消息同步的核心点在于消息不能丢，微博由于使用的是MCQ，通过本地写远程读的方式，可以很方便的实现高效稳定的跨机房消息同步。

异地多活的新方向

时间到了2015年，新技术层出不穷，之前很多成本很高的事情目前都有了很好的解决方案。接下来我们将在近五年异地多活部署探索的基础上，继续将微博的异地多活部署体系化。

升级跨机房消息同步组件为跨机房消息同步服务。面向业务隔离跨机房消息同步的复杂性，业务只需要对消息进行处理即可，消息的跨机房分发、一致性等由跨机房同步服务保障。且可以作为所有业务跨机房消息同步的专用通道，各个业务均可以复用，类似于快递公司的角色。

推进Web层的异地部署。由于远距离专线成本巨大且稳定性很难保障，我们已暂时放弃远程异地部署，而改为业务逻辑近距离隔离部署，将Web层进行远程异地部署。同时，计划不再依赖昂贵且不稳定的专线，而借助于通过算法寻找较优路径的方法通过公网进行数据传输。这样我们就可以将Web层部署到离用户更近的机房，提升用户的访问性能。依据我们去年做微博Feed全链路的经验，中间链路占掉了90%以上的用户访问时间，将Web层部署的离用户更近，将能大大提升用户访问性能和体验。

借助微服务解决中小服务依赖问题。将对资源等的操作包装为微服务，并将中小业务迁移到微服务架构。这样只需要对几个微服务进行异地多活部署改造，众多的中小业务就不再需要关心异地部署问题，从而可以低成本完成众多中小服务的异地多活部署改造。

利用Docker提升前端快速扩容能力。借助Docker的动态扩容能力，当流量过大时分钟级从其他服务池摘下一批机器，并完成本服务池的扩容。之后可以将各种资源也纳入到Docker动态部署的范围，进一步扩大动态调度的机器源范围。

以上是对微博异地多活部署的一些总结和思考，希望能够对大家有所启发，也希望看到更多的同学分享一下各自公司的异地多活架构方案。

编后语

为了更好地向读者输出更优质的内容，InfoQ将精选来自国内外的优秀文章，经过整理审校后，发布到网站。本文首发于“微博平台架构”微信公众号，已由原作者授权InfoQ中文站转载。

关于作者

刘道儒，毕业于北京信息工程学院，现任微博平台研发高级技术经理、Feed项目技术负责人，负责Feed体系后端研发，先后在TRS、搜狗、新浪微博从事社区&社交业务研发，专注于大规模分布式系统的研发、高可用等领域。他曾代表平台在2012年主导了微博广州机房部署项目以及北京双机房部署项目。

感谢臧秀涛对本文的审校。

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至editors@cn.infoq.com。也欢迎大家通过新浪微博（@InfoQ，@丁晓昀），微信（微信号：InfoQChina）关注我们，并与我们的编辑和其他读者朋友交流。