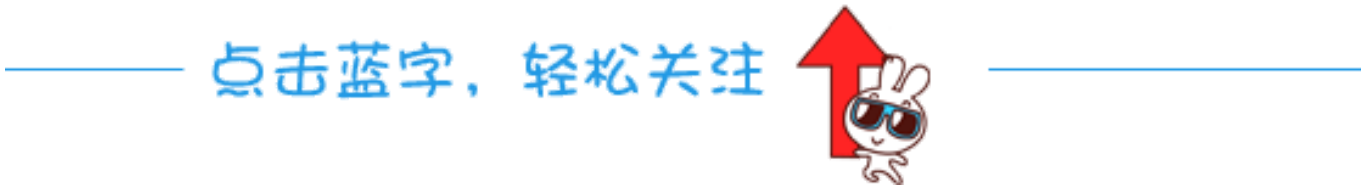
【学在GOPS】阿里游戏高可用架构设计实践

原创：李运华 高效运维 2016-07-20



> 本文根据〖GOPS2016 全球运维大会•深圳站〗李运华演讲整理而成，编辑同学为陈才@腾讯。
>
> 欢迎关注“高效运维(微信ID：greatops)”公众号，以抢先赏阅干货满满的各种原创文章。

讲师简介

> **李运华**
>
> 十余年软件设计开发经验，经历了电信行业和移动互联网行业，曾就职于华为和UCWEB，先后担任软件开发工程师、系统分析师、架构师、技术leader等角色；
>
> 现担任阿里巴巴移动事业群（原UCWEB）资深软件工程师，带领多个研发团队，承担架构设计、架构重构、技术团队管理、技术培训等职责。
>
> 技术上专注于开源技术、系统分析、架构设计，对互联网技术的特点和发展趋势有较深入的研究和理解。先后负责过游戏接入高可用项目、飞鸽事件发布订阅系统、交易平台系统解耦项目，对于系统解耦、高性能、高可用架构有丰富的经验。

大家下午好！我是来自阿里游戏的李运华，每次在做自我介绍的时候都会面临一个尴尬，包括今天也是，听到我名字的人看了我的样子之后，第一句话就是：“哇，你这么年轻！”。

但其实我已经在行业做了十多年，大家可以怀疑我的年龄，但不用担心我的实力。

今天演讲的题目是《阿里游戏高可用的架构设计实践》，演讲题目比较长，也很常见，估计大家听了没什么印象，我从整个演讲的内容中提炼出一句话，相信更加容易记住：**把运维的锅让研发去背**！也就是说，高可用的系统是设计出来的，不是靠运维保障出来的！

今天演讲的内容没有云计算，也没有大数据这些高大上的东西，但是无论是什么业务、什么行业，其实都可以借鉴，也可以比较容易落地实施。

项目背景

项目启动的背景其实跟这个图非常吻合，叫疲于奔命的运维，非常形象化地展示了当时运维的状态。

> 我们有一个月发生4次比较大的故障，包括机柜断电、交换机断机、服务器段机、程序Bug，这些故障影响的时间短的可能半个小时，长的可能就一个小时、两个小时。

对于游戏接入来说，如果游戏不能登录的话，很多玩家就打电话投诉游戏厂商或者直接打电话到阿里游戏爆粗口，影响是很大的。

分析

> **为什么说运维背了很大一个锅呢?**

因为一听到说是机柜断电，有的研发同学就长出一口气，“这个是机柜断电导致的，好象跟这个研发设计方案没有太大关系。”

所以，当时除了程序Bug以外，可能其他都是运维背黑锅了。

分析-运维太LOW?

出了这么大的问题，肯定要分析以及给出后面的应对解决方案。于是我们就成立了一个工作小组，小组第一个问题就是**怎么去分析这些问题背后真正的原因?**

- 首先想到的是不是运维太LOW了，比如说硬件质量太差，为什么这个月机柜也坏、交换机也坏，是不是到电脑城买个二手货放里面了?
- 第二想到的是不是运气不好，之前一个月、两个月才遇到一次，这个月遇到了4次，是不是你们没有在机房烧香?
- 第三个是不是测试不足，为什么这些Bug测试阶段不能发现，线上才发现？还有运维经验不足，比如交换机出现故障，有的人以为很简单，倒换一下就行了。
- 甚至有的同学提到了流程不完善，说整个流程中有很多可以改进的地方。

  比如说故障发生之后，响应机制是不是不够顺畅，故障发生之后一堆人，包括研发、测试、运维手忙脚乱，是不是要定一个全流程的处理方案，指定一堆的责任人?

总体架构

经过深入的讨论和分析，最后我们认为，根本原因还是**系统设计方案**有问题，也就是说，技术上是比较弱的。

得到这个根本原因之后，解决策略就比较明显，总结一句话就是“**把运维的锅让研发去背**”，不要指望你的运气好或者烧香、提高某某人的运维能力，而是假设所有的这些问题都有可能发生，发生之后你的系统是不是有应对的措施和快速的解决方案？最终我们决定**从技术方面来解决这个可靠性的问题**。

高可用目标-传统方法

确定这个方向之后我们就需要定一个目标，首先确定一个目标。高可用其实都是指几个9，5个9的话可能就是电信级或者金融级的，互联网大部分是3个9到4个9。

刚开始也想用这个指标作为一个目标，比如说想达到4个9或者5个9，但是这个指标的优点是业界通用，搞技术的大家都接受了这个指标。

但是有一个缺点，除了技术人员，其他同学不是很好理解，他们没有办法将4个9或者5个9转换成直观的理解。所以，我们当时在定项目目标的时候并没有这样去定。

高可用目标-面向业务

我们最终确定的目标跟几个9的目标有一个比较大的区别，几个9的目标主要是从系统的角度去考虑的，就是说这个系统的可靠性是几个9。

而我们的目标是面向整个业务，这个目标就是**3分钟来定位问题，5分钟能恢复业务**，而且问题的发生频率不能太高，2个月发生一次。

这个目标的优点：

1、聚焦业务。

2、容易分解。目标本身就是我们的工作方向，首先要定位问题，怎么定位问题？我们就可以想一个办法，其次是恢复业务，第三是故障的频率不能太高；

3、容易衡量。我们后来再做方案的时候，很多方案只要拿这个标准一套，基本上就能够判断这个方案是否可行。

整个目标最后折算下来，对应的差不多是4个9左右，比4个9高一点点。

高可用整体架构

> 整体架构一共分为四层：用户层、网络层、服务层、运维层。

整个架构其实跟目标是一样的，我们是面向整个业务的，没有说哪个系统应该具备几个9的高可用，而是站在整个业务的全流程来看假设要达到目标，每一个应该怎么去做。

每一层都需要做一些应对的方案，才能达到目标。

接下来我就详细给大家介绍一下，每一个方案的基本思路和做法。

HTTP-DNS

客户端重试

在游戏接入阶段，游戏里会嵌入我们的SDK，对于SDK内的产品来说，如果遇到后端故障，最快、对用户影响最小的解决方式是立刻去重试，服务器1有故障的话，我们重新发一个请求到服务器2，就能够正确处理业务。

重试里面有一个关键点需要特别注意，重试的时候必须保证这个请求不要再发到原来有问题的服务器上面，否则这个重试只是浪费时间。

传统DNS问题

在重试的时候，要尽量保证重试的服务器跟原来的服务器不一样，但是传统的DNS正好在这方面存在天然的缺陷，DNS存在三个方面的问题：

> 1、DNS劫持
>
> 2、DNS污染
>
> 3、DNS缓存

比如这个截图就是我们的用户的机器上**DNS**已经被运营商给**劫持**。对于**DNS的污染**来说，在国内大区域网络应该是每时每刻都在发生的。**DNS缓存**，包括操作系统，为了提高性能，都把DNS解析的结果缓存在本地，这样就会带来一个问题，我们在重试的时候，有可能第二次重试还是发到原来有问题的机器上。

HTTP-DNS

我们做了一套HTTP-DNS方案，用户可以向HTTP-DNS系统请求，其实也是域名和服务器的对应关系，但是它走的是HTPP私有的协议，而不是走传统的DNS协议。

我们的运维人员可以直接在这个HTTP-DNS服务器上去修改配置，比如说某台机器有故障，运维一键就把它下掉了。我们的服务器也可以自己上报一些状态和监控给HTTP-DNS系统。

HTTP-DNS系统有三个优点：

1、**灵活**。因为这个走的是HTTP的协议，而且是私有的，可以基于自己的业务特点做很多个性化的东西。

2、**快速**。因为它不存在缓存这个问题，一旦运维人员甚至是测试同学把这个服务器下掉后，用户这个请求能够立刻拿到最新的结果，避免重复返回到原来有故障的机器。

3、**方便**。这个系统是我们自己维护的，想做什么样的操作都可以，如果是公共的DNS那就做不了。

客户端重试+HTTP-DNS

结合客户端重试+HTTP-DNS之后，整体的解决方案就是这样的，正常的时候走的传统DNS，异常情况下就走只有的HTTP-DNS。传统的DNS成功了，那业务就没有问题，如果失败了，再走HTTP-DNS请求。

> **为什么正常的要走传统的DNS呢?**

这是为了性能考虑，因为HTTP-DNS为了能够快速地处理故障，它其实是不存在缓存机制的。所以，正常情况下走传统DNS有利于我们保证业务服务的性能。

架构解耦

业务分离

下图是原来的架构，这个系统把所有的功能都包含了，比如说登录、注册、参数下发、消息、日志、更新。

其实对于玩家来玩游戏来说，真正强相关的只有登录注册和参数下发，消息和日志、更新其实并不是玩家玩游戏必须具备或者强相关的。

所以，业务分离的做法就是把核心业务和非核心业务分拆到不同的系统中，把两个系统之间通过接口调用，互相访问。

这样做的好处，假设非核心业务系统出现故障，它并不影响核心业务系统，因为它们之间是通过接口调用的，并不共享相同的资源。

举一个最简单的例子。

> 现在因为运营的误操作，下发了大量的消息，最后把整个系统冲死了，玩家登录不了，游戏玩不了。
>
> 拆分之后，这块系统自己能够完全挂起，跟玩游戏相关的登录、注册都不受影响，他可以继续访问，只要玩家只要能够登录游戏进行玩，就不会有太大的问题。

当然，这个架构对系统功能实现还是有一定要求的，毕竟有接口访问，核心系统和非核心系统还是有一定依赖的。

所以，对于核心业务系统来说，涉及到非核心业务系统的访问就需要做一定的容错处理。如果接口调用失败后，你自己的业务就失败了，就起不到业务分离的作用。

服务中心

服务中心类似于DNS，是实现整个内部系统之间服务调用时候的调度功能，服务中心是一个类似于服务的名字系统。

比如说有一个A业务想访问其他系统提供的业务。它首先并不是直接访问到另外一个系统，而是要到服务中心访问一下。

比如说我需要X服务，服务中心告诉A：你去访问Host1+port1的xxx接口。服务中心有一个配置和状态上报，可以根据一些状态、算法、配置，选择一个最优秀的服务器告诉A业务。

那么，A业务收到之后就按照这个指示去访问真正提供业务的机器，比如B系统里的Host1+ port1这个机器。服务中心的作用跟HTTP-DNS的作用差不多，就是希望在内部某个系统故障的时候可以快速处理或者切换。

假设B系统某台机器有问题了，我们可以通过自动或者人工的方式在服务中心把这台机器直接下掉，A业务请求的时候，它就不会再请求到这个有问题的机器上，这一台机器的故障就不影响A业务。

业务降级

整个系统拆分成核心业务系统和非核心业务系统，在一些紧急情况下，比如说非核心业务系统重启也没有办法，甚至说某个数据库搞挂了，它又影响业务核心系统。

这个时候，接口是可以访问的，但是响应时间特别慢，核心系统就有点被拖慢。

那么，在这种比较极端的情况下，我们可以通过人工的方式下发降级指令，把这个非核心业务系统的功能给停掉，这个停掉并不是把程序停掉，而是说把其中的一个接口或者url停掉，核心系统去访问的时候就得到一个500或者503错误。

我们做了一个专门的降级系统，降级系统可以去下发这些降级指令。一般情况下由降级系统给非核心业务系统下发降级指令，如果到了一些关键时刻，其实核心业务系统中有的接口也是可以进行降级的。

也就是说，我们降级的时候并不是对整个系统或者整个功能进行降级，我们可以做到接口或者url这个级别的降级。通过牺牲非核心业务系统的功能，我们尽最大努力地去保证核心业务系统提供的业务。

这个业界有很多叫法，比如有损服务、可损服务，其实我们这个也是一个可损服务，功能上的可损，不是流量上的可损。

异地多活

原来的老架构中分两个集群：A集群、B集群。两个集群共用同一个数据库主库，部署在A集群，所有的写操作都放在A集群主库，由A集群主库同步到A集群从库、B集群从库，各集群的读操作都直接读集群本地的从库。

它的缺点是存在全局单点的问题，假如说A集群主库挂了，两个集群的业务基本上全部不能提供了。

第二个问题是跨机房同步时延的问题，从A集群同步到B集群，我们是直接通过MySQL底层的复制机制去同步的，在一些比较极端的情况下，它的时延可能达到半小时甚至一个小时，这么大的时延的情况下，提供的业务可能就有问题。

为了解决全局单点、跨机房同步时延的问题，我们做了一套新架构，如下图：

这个新架构和老架构相比。**最大的特点**：

1、**业务层数据同步**。现在有两个主库了，每个集群读写自己的库。那么，两个集群的数据怎么同步呢？我们其实是通过应用层进行数据同步的，而没有采用数据库底层的机制读库。

这样做有一个好处，我们能够尽量通过程序去保证同步的有效性和及时性，而不依赖于底层，用程序可以做多种动作。

2、**二次读取**。虽然程序进行同步能够尽最大可能缓解同步时延的问题，但是在一定极端情况下其实还是有一定时延的问题。所以，我们又做了两个集群直接互相读取的功能，叫二次读取。

也就是说，有的数据在A集群还没有同步到B集群，B集群此时是读取不到的，这时B集群的系统会通过程序的接口再去访问A集群的系统，意思是：A集群，你这边有没有这个数据？如果有的话，A集群就要返回数据给B集群，B集群再把数据再写入本地。

3、**可重复生成全局唯一数据**。我们的游戏数据中有一个数据是要求全局唯一的，不管在哪个集群，不管什么时候生成，对于同一款游戏、同一个用户来说，这一个值都是唯一的。

最开始的时候，我们是通过redis两个集群分段来实现的，但是通过redis分段就有一个问题，假设其中一个集群宕机之后，另一个集群并不能把原来集群的业务全部接过去，只是能够保证自己集群的业务没有问题。

我们为了能够解决两个集群互相接管对方的业务，就做了可重复生成全局唯一数据方案。

360度监控

一体化

> 整体方案从上到下分为五层：业务层、应用服务层、接口调用层、基础组件层、基础设施层。

1、**业务层：**就是我们业务上的打点，根据这些打点进行机型统计或者分析；

2、**应用服务层：**简单来说就是我们url的一个访问情况；

3、**接口调用层：**就是我们自己系统对外部依赖的那些接口的访问情况，比如A系统调用B系统的一个接口，在A系统里统计或者监控调用B系统接口的情况，包括时延、错误次数之类；

4、**基础组件层：**其实就是我们使用的一些组件，包括MySQL等；

5、**基础设施层：**就是最底层的，包括操作系统、网络、磁盘、IO这些设备。

整个监控是分层的，在我们出现问题的时候，定位问题需要的关键信息全部包括在这里面的。

自动化

自动化的监控方案是业界现在最常见的ELK解决方案，整个方案是实时采集和分析的，并不需要人工参与。

> **为什么特别要做自动化日志分析呢?**

我们来看看在没有做自动化之前我们是怎么处理线上问题和故障的。

- 首先，研发测试同学问一下运维同学这些IP是多少，再一台一台机器登录上去，再把日志下载下来，再由研发或者测试同学去写脚本去分析。

整个流程非常低效，从生产网到办公网的网速很慢，要下载1G日志可能需要半小时、一小时，如果下载10台机器，我们之前遇到一次把10台机器日志下来用了三个小时。

- 其次，现场由测试同学和研发同学去写AWK这些脚本的时候，经常会出现脚本写错了，毕竟研发和测试同学写脚本的经验也不会太多，之前就出现过分析了半天最后发现这个脚本写错了。

要知道，在处理线上故障的时候是争分夺秒，你下载日志半小时、写个脚本写错了调试10分钟、把脚本和日志跑一下半个小时，一个多小时过去了，用户的投诉电话已经打到客服顶不住了。

为了能够快速处理这个故障，我们进行了详细的分析，把真正出故障的时候要关注哪些信息、关注哪些指标，都通过预先的日志采集、计算、分析完成，放在那里等我们要处理故障或者问题的时候，直接看就可以了。

可视化

每一层的指标，都通过一个系统可视化展现出来。比如，今日的访问量、今日的正确率、最近一分钟的平均响应时间、错误率。

整个立体化监控的这一套系统出来之前，我们定位故障或者处理问题都需要最顶级、最资深、最核心的员工放下手头的工作来处理，**为什么？**

因为那些日志，其他人如果你没有做过的话，你不知道分析哪个日志，只能找他们来处理和定位。

有了这套系统以后，只要是做这个业务的同学，不管是研发还是测试，他都能够通过这套系统快速地定位问题。

有了这套系统以后，有的业务部门确定了一个值班机制，每周都安排几个人去检查系统状态，新同学也可以，研发也可以，测试也可以，甚至项目经理都可能上去看一下系统的状况。

整个系统的状态一目了然，基本上识字就能够看懂整个系统的状态。

除了发现比较大的问题外，一些很小的问题，我们也可以通过这个系统发现。

> 我举一个最简单的例子，比如我们有一个系统，上了这个系统一看，每天到了零点，请求量就爆涨。我们的业务比较特殊，12点没有什么人来真正要去刷或者签到、领礼包之类的；
>
> 但是奇怪的是到了12点就暴涨，后来通过可视化的监控系统把每天12点的访问量排行一看，原来是有一个日志接口，代码写成“定时上报，定时12点”，结果在12点大量客户端同时上报日志。

定位问题后其实修改方式比较简单，改一下，在12点左右就行了。

设计哲学

1、**面向业务**。不单单关注某一个系统某一个模块的可靠性和高可用，而是站在**整个业务流程**的角度去分析一下，为了保证这个业务的高可用，每一个步骤、每一个环节、每一个系统应该怎么做，需要做哪些改进和优化。

2、**技术驱动**。我们并没有说制定完善的流程、提高人的素质或者采购更贵的硬件、更好的硬件，整体的方案都从技术的角度去考虑。

3、**关注核心**。非核心业务我们可以紧急的时候停掉或者关掉。

4、**可量化**。360度监控里的这些指标、分析都是可以量化的，整个项目目标也是可以量化的。

整体方案的效果

做这个方案前，几乎是每个月有一次故障，最极端的是有一个月有4次比较大的故障。可用性指标大概是3个9。

做完这个系统之后，到目前为止没有一次大的线上故障，可用性指标达到了4个9。**是不是我们的运气好呢?**

其实不是，做了这套方案后，机房的线路也挂过，交换机也断过，就在3月份的时候我们刚刚有一台核心交换机的缓存模块出现故障，最后厂商定位说是他们的Bug，交换机花了半个小时才定位恢复，但是对我们的业务其实没有影响，因为我们把整个业务切到另外一个机房去了。

愿景

天下无贼，运维无锅。研发、测试、运维，大家一起来设计高可用性。

谢谢大家！

Q&A

提问1：我们知道，DNS请求不需要有端，但是HTTP-DNS是有端，在端上找到你们的服务器，你们的服务器出现故障是怎么切换的呢?

> 李运华：服务器出现故障的话，可以通过人工，也可以通过自动，来修改HTTP-DNS上面的配置。

提问2：不是，我是说HTTP-DNS如果出现故障，端的处理，怎么在端切换的?

> 李运华：我们整体流程，正常情况下我们走传统DNS，传统DNS有问题的时候，我们再走HTTP-DNS。如果说，正常的DNS也出现问题了，HTTP-DNS也出现问题了，这个就没有办法了。

> 当然，其实我们有几套HTTP-DNS。

提问3：如果这一台故障了，端那边没有办法处置到另一台吗?

> 李运华：我们有几个HTTP-DNS系统。

提问4：HTTP-DNS上面的故障和DNS的故障，你们是自动调度过去，两边都会生效，只是说DNS有缓存。

> 李运华：对。

提问5：刚才讲到的服务中心数据是不是相当于Load Balance

> 李运华：服务中心和Load Balance还是有区别，虽然它的作用也是Load Balance的作用，但服务中心并不直接接受真正的接口请求，它类似于一个DNS。

提问6：更多的作用还是在于DNS里面。

> 李运华：对，你可以理解为它是一个服务的名字系统。谢谢。

我是华丽的分割线

↓↓↓ 点击"阅读原文" 【直接报名】

阅读原文