微服务架构下的高可用方案

一、一些概念

双机热备（Active - Standby）是指两台机器都在运行，但并不是两台机器都同时提供服务。当提供服务的一台机器出现故障的时候，另外一台机器会马上自动接管并且提供服务，并且切换的时间非常短。

HA，即高可用（High availability）。

LVS，即Linux 虚拟服务器（ Linux Virtual Server），现在 LVS 已经是 Linux 标准内核的一部分，从 Linux2.4 内核以后，已经完全内置了 LVS 的各个功能模块。

RAID，(RedundantArrayofIndependentDisk独立冗余磁盘阵列)。

二、数据库架构

在项目中，当业务规模越来越大，数据越来越多时，随之而来的就是数据库压力会越来越大，数据库层慢慢成为了整个系统的关键点和性能瓶颈，因此实现数据层的高可用就成了项目中经常要解决的问题。在保障数据层的高性能与高稳定方面，最常用的方式就是对数据进行分片、多份、冗余等，很多架构的本质其实也是基于这几点来实现的。因为无论底层是关系型数据库，还是NoSQL数据库，无论是 Mysql 还是 Redis、MongoDB，在架构设计上都是相通的。大体上，单中心双机的常见方案有以下这些：

- **主备架构（Primary - Standby Architecture）**
- **主从的架构（Master-Slave Architecture）**
- **双主架构（Master-Master Architecture）**

以上方案从上至下，依次是从简单到复杂，从基础到丰富。

1.1 主备架构（主备式）

主备架构是双机部署中最简单的一种架构，几乎市面上所有的数据库系统都会自带这个主备功能。

其思路是：将数据库部署到两台机器，其中一台机器（代号M）作为日常提供数据读写服务的机器，称为「主机」。另外一台机器（代号S）并不提供线上服务，但会实时的将「主机」的数据同步过来，称为「备机」。一旦「主机」出了故障，通过人工的方式，手动的将「主机」踢下线，将「备机」改为「主机」来继续提供服务。

优点：几乎不需要做什么开发改造，各类数据库就支持这种模式，部署维护起来也简单，并没有引入额外的系统复杂度和瓶颈。

缺点：1）当「主机」出现故障的时候，需要人工去干预啊，运维很辛苦且处理不一定及时。

2）主备架构会造成严重浪费资源，需要一台与「主机」同等配置的「备机」长期备着，但又不作为线上服务来使用。

为了解决这个资源浪费问题，出现了一个把「备机」也用起来的方案：主从式架构。

1.2主从架构（主从式）

主从式架构大体上与主备式架构差不多。区别是主备式的「备机」平时是用的，主要起到备份的作用，而主从式的「备机」改为了「从机」，平时也要提供「读」服务，但不提供「写」服务。。

主从架构中，「主机」提供「读」、「写」服务，并实时的将线上数据同步到「从机」，以保证「从机」能够正常的提供「读」服务。

这种架构相比较主备式，对资源是一种节约，并且在「主机」出现故障时，在人工介入之前，「从机」能够提供数据的「读」操作的，由于大多数业务都是「读」多「写」少，因此对稳定性又提高了一个层次。

缺点：1）架构稍微复杂了一点，毕竟「主机」和「从机」都有「读」服务，那么前端业务系统就需要用一定策略去判断该路由到哪一台去读取数据。

2）延迟问题，「主机」的数据同步到「从机」难免会有一定程度的延迟，这个延迟可能会对数据**实时性**要求较高的业务有一定影响。

综上，虽然主从架构一定程度解决了资源浪费，但是并没有解决人工干预的问题，当出现了故障后还是需要人

工去处理。如果想让架构更智能一点，就需要引入「主从双机自动切换」的功能。

主从双机自动切换：是指当主机出现故障后，从机能够自动检测发现。同时从机将自己迅速切换为主机，将原来的主机立即下线服务，或转换为从机状态。

要实现「主从双机自动切换」，有几个关键点需要考虑：

主机与从机之间的状态如何判断?

必须有一个机制能监测两台机器的运行状态，以此来决定是否应该切换。

我们比较常用的状态传递方式有两种：

- 「双机互连模式」
- 「第三方中介模式」

「双机互连模式」：是指在主机和从机之间建立一条用于状态通讯的通道。通过这个通道，主机和从机之间可以共享服务状态，一旦发现对方宕机或者停止服务了，就可以立即将自己切换为主服务。不过这种方式需要关注**通道的健壮性**，一旦通道自身不稳定了，可能会导致假消息出现，比如主机并没有宕机，但是通道坏了，导致从机以为出现了异常，就将自己切换为了主机，那就出现了2个主机了，因此通道本身也是一个可能的故障点。

「第三方中介模式」：是指在主机和从机之外，再建立一个中介机器，这个中介机器专门用来维护各节点（主机/从机）状态的，主机/从机实时的将自身状态上报给中介机

器，中介机器来决定是否应该切换、何时切换。MongoDB的Replica Set就是采用的这种模式。

1.

    1. *除了状态判断，还需要考虑切换的策略是什么？即发生异常几次/多久后开始切换，是否有缓冲机制等。另外切换完成后，当原主机又恢复正常之后是否需要自动再切换回来等策略。*

    2. *另外就是需要注意在切换过程中双机数据如果发生冲突时，以哪个为准？处理机制是什么。*

这些细节都是在设计主从自动切换架构时候，要提前规划的。

1.3双主架构（互为主从式）

互为主从的架构是指两台机器自己都是主机，并且也都是作为对方的从机。两台机器都提供完整的读写服务，因此无需切换，客户机在调用的时候随机挑选一台即可，当其中一台宕机了，另外一台还可以继续服务。

互为主从架构，因为两台主机都接受写数据，那就需要将写的最新数据**实时**的同步给对方，需要将数据进行两台主机的**双向复制**。而双向复制不可避免的会在一定程度上带来**数据延迟**、极端情况下甚至有**数据丢失**等问题。在

实际业务中，有些业务数据对一致性要求是非常高的，并不能接受数据的延迟、丢失，因此这类业务也不适合互为主从的模式，比如金融业务。

1.4据库在多机集群模式下的技术架构

三、数据库一致性解决方案

一致性解决方案（Consistency Solution）：在分布式数据库系统中，一致性解决方案是指确保不同节点之间数据的一致性。

常见的一致性解决方案包括基于时间戳的复制、基于多版本并发控制（MVCC）的复制、基于Paxos协议的一致性算法、基于Raft协议的一致性算法等。这些算法都旨在保证不同节点之间数据的一致性和可靠性。

数据库架构设计也是一个重要的任务，良好的设计可以提高数据库的性能、可用性和可维护性。

1、 数据库的范式化设计：通过范式化的设计，可以减少数据冗余和数据不一致的问题。常见的范式包括第一范式（1NF）、第二范式（2NF）和第三范式（3NF）等。

2、数据库的反范式化设计：有些情况下，反范式化的设计可以提高数据库的性能。反范式化的设计包括将数据冗余存储、增加冗余索引、分区表、分片等技术。

3、 合理分配数据和索引：合理的数据和索引分配可

以提高数据库的查询性能。通常建议将索引分配到较小的表中，同时将数据均匀分配到各个节点中。

4、 优化数据库查询：通过对查询进行优化，可以提高数据库的查询性能。包括对查询进行优化，例如使用索引、避免全表扫描、使用优化的SQL语句等。

5、合理的分区和分片：分区和分片是分布式数据库中常用的技术，可以提高数据库的可扩展性。合理的分区和分片可以减少网络开销、提高查询性能、降低数据库负载等。

6、 安全性和可靠性：在数据库架构设计中，安全性和可靠性是非常重要的。包括对数据库进行备份、数据加密、访问控制、防火墙等技术的应用。

6、监控和调优：在数据库架构设计中，监控和调优也是非常重要的。包括实时监控数据库的性能、调整数据库配置参数、调整应用程序、优化数据库查询等。

四、MySQL主备应用和原理

4.1MySQL主备

如图 1 所示就是基本的主备切换流程。

binlog 的特性确保了在备库执行相同的 binlog，可以得到与主库相同的状态。因此，可以认为正常情况下主备的数据是一致的。也就是说，图 1 中 A、B 两个节点的内容是

一致的。

4.2MySQL双主

节点 A 和 B 之间总是互为主备关系。这样在切换的时候就不用再修改主备关系。但是，双 M 结构还有一个问题需要解决。业务逻辑在节点 A 上更新了一条语句，然后再把生成的 binlog 发给节点 B，节点 B 执行完这条更新语句后也会生成 binlog。那么，如果节点 A 同时是节点 B 的备库，相当于又把节点 B 新生成的 binlog 拿过来执行了一次，然后节点 A 和 B 间，会不断地循环执行这个更新语句，也就是循环复制了。

这个要怎么解决呢？从上面的图 6 中可以看到，MySQL 在 binlog 中记录了这个命令第一次执行时所在 实例的server id。因此，可以用下面的逻辑，来解决两个节点间的循环复制的问题：

1. 规定两个库的 server id 必须不同，如果相同，则它们之间不能设定为主备关系；
2. 一个备库接到 binlog 并在重放的过程中，生成与原 binlog 的 server id 相同的新的 binlog；
3. 每个库在收到从自己的主库发过来的日志后，先判断 server id，如果跟自己的相同，表示这个日志是自己生成的，就直接丢弃这个日志。

原文链接：
https://blog.csdn.net/qq_43284469/article/details/129017

4.3MySQL+Keepalived双主热备高可用环境部署

MySQL双主复制，即互为Master-Slave(只有一个Master提供写操作)，可以实现数据库服务器的热备，但是一个Master宕机后不能实现动态切换，实际生产上使用比较多的是双 M 结构。

使用Keepalived，可以通过虚拟IP，实现双主对外的统一接口以及自动检查、失败切换机制，从而实现MySQL数据库的高可用方案。

环境概述：
1）先实施Master->Slave的主主同步。主主是数据双向同步，主从是数据单向同步。一般情况下，主库宕机后，需要手动将连接切换到从库上。（但是用keepalived就可以自动切换）

2）再结合Keepalived的使用，通过浮动IP实现Mysql双主对外连接的统一接口。即客户端通过浮动IP连接数据库；当其中一台宕机后，浮动IP会漂移到另一台上，这个过程对于客户端的数据连接来说几乎无感觉，从而实现高可用。

————————————————————

原文链接：
https://blog.csdn.net/fyh360345192/article/details/84620473

五、高可用浅析

高可用（High availability，即 HA）主要是针对架构而言，第一步一般会采用分层的思想将一个庞大的应用系统拆分成应用层、中间件、数据存储层等独立的层，每一层再拆分成更细粒度的组件；
第二步就是让每个组件对外提供服务，毕竟每个组件都不是孤立存在的，都需要互相协作，对外提供服务才有意义；
第三步就是保证架构中所有组件以及对外暴露服务都要做到高可用，任何一个组件或其服务没做高可用，都意味着系统存在风险。

5.1高可用方案

任何组件要做高可用，都离不开「冗余」和「自动故障转移」。

组件一般是以集群（至少两台机器）的形式存在的，只要某台机器出现问题，集群中的其他机器就可以随时顶替，这就是「冗余」。

简单计算一下，假设一台机器的可用性为 90%，则两台机器组成的集群可用性为 1-0.1*0.1 = 99%，所以显然冗余的机器越多，可用性越高。

但光有冗余还不够，如果机器出现问题，需要人工切换的话也是费时费力，而且容易出错，所以我们还需要借助第三方工具（即仲裁者）的力量来实现「自动」的故障转移，以达到实现近实时的故障转移的目的，**近实时的故障转移**才是高可用的主要意义。
在业界一般用几个九来衡量系统的可用性，如下：

互联网采用的微服务架构示意如下（简单架构）：

可见架构主要分为以下几层：

接入层：主要由 F5 硬件或 LVS 软件来承载所有的流量入口；
反向代理层：Nginx，主要负责根据 url 来分发流量，限流等；
网关：主要负责流控，风控，协议转换等；
服务层：主要业务服务；
存储层：也就是 DB，如 MySQL，Oracle 等，一般由服务层调用返回；
中间件：ZK，Redis，Kafka 等，主要起到加速访问数据等功能；

5.2高可用实现

1）接入层&反向代理层

可用通过引入Keepalived实现，类似如下

两个 nginx 以主备的形式对外提供服务，注意只有 master 在工作（即此时的 VIP 在 master 上生效），另外一个 backup 在 master 宕机之后会接管 master 的工作。在主备机器上都装上 keepalived 软件，通过「IP漂移」的方式即解决了nginx的高可用。

具体参考《浅入浅出keepalived+nginx实现高可用双机热备》。

2）服务层

采用的是微服务架构，微服务架构体系自带高可用机制，此处不再展开。

3）中间件

Redis 的高可用需要根据它的部署模式来看看，主要分为「主从模式」、「哨兵模式」和「集群模式」，具体参考《Redis之高可用方案浅析》。

Kafka集群，每个 Topic 的 Partition 都分布式存储在其它消息服务器上，这样一旦某个 Partition 不可用，可以从 follower 中选举出 leader 继续服务，不过与 ES 中的数据分片不同的是，follower Partition 属于冷备，也就是说在正常情况下不会对外服务，只有在 leader 挂掉之后从 follower 中选举出 leader 后它才能对外提供服务。

4）存储层

mysql的高可用设计也是引用Keepalived来实现高可用。大家在工程配置的 MySQL 地址一般是 VIP 以保证高可用。数据量大了之后就要分库分表了，于是就有了多

主，就像 Redis 的分片集群一样，需要针对每个‘主’配备多个‘从’，具体参考《Mysql之高可用方案浅析》。

————————————————————

5.3SpringCloud应用双节点高可用部署测试

步骤：方案设计->注册中心HA部署->Nginx代理部署->Keepalived双机热备部署->微服务其他应用部署->MySQL主从配置->Redis主从配置->MongoDB RepilcaSet配置->RabbitMQ集群配置->HA测试

六、LVS、Nginx、HAProxy、keepalive 的工作原理

在 Linux 环境下，常用的负载均衡解决方案包括 HAProxy、Nginx 和 Keepalived。

实际应用中，在应用服务器集群之前会有负载均衡服务器，负载均衡设备的任务就是作为应用服务器流量的入口，挑选一台 Web 服务器将客户端的请求转发给它处理，实现客户端到真实服务端的透明转发。

- LVS、Nginx、HAProxy 是目前使用最广泛的三种软件负载均衡软件。

一般对负载均衡的使用是根据不同的规模来使用不同的技术。如果是中小型的 Web 应用，比如日 PV 小于1000万，用 Nginx 就完全可以了；如果机器不少，可以用 DNS 轮询，LVS 所耗费的机器还是比较多的；大型网站或重要的服务，且服务器比较多时，可以考虑用 LVS。

目前关于网站架构一般比较合理流行的架构方案：

- Web 前端采用 Nginx/HAProxy+Keepalived 作负载均衡器；
- 后端采用 MySQ L数据库一主多从和读写分离，采用 LVS+Keepalived 的架构。

HAProxy---高性能负载均衡软件 官网：haproxy.org/

高可用——Keepalived安装部署使用详解

使用keepalived 与 HAProxy 给两台服务器设置容灾故障转移的配置文档

作者：jiangmo
链接：https://www.jianshu.com/p/16e9c84fdb3c
来源：简书

七、数仓 | 几种常见的数据同步方式

数据仓库的特性之一是集成，即首先把未经过加工处理的、不同来源的、不同形式的数据同步到ODS（Operational Data Store，原始数据）层，一般情况下，

这些ODS层数据包括日志数据和业务DB数据。

对于业务DB数据而言(比如存储在MySQL中)，将数据采集并导入到数仓中(通常是Hive或者MaxCompute)是非常重要的一个环节。

那么，该如何将业务DB数据高效准确地同步到数仓中呢?一般企业会使用两种方案：直连同步与实时增量同步(数据库日志解析)。其中直连同步的基本思路是直连数据库进行SELECT，然后将查询的数据存储到本地文件作为中间存储，最后把文件Load到数仓中。这种方式非常的简单方便，但是随着业务的发展，会遇到一些瓶颈，具体见下文分析。

为了解决这些问题，一般会使用实时增量的方式进行数据同步，其基本原理是CDC (Change Data Capture) + Merge，即实时Binlog采集 + 离线处理Binlog还原业务数据这样一套解决方案。

本文主要包括以下内容，希望对你有所帮助

-
    - 常见数据同步方式
    - 流式数据集成

参考资料

1、架构设计之「数据库从主备到主主的高可用方案」

2、数据库架构：主备、双主、主从架构、一致性解决方案

3、MySQL是怎么保证主备一致的

4、Mysql+Keepalived双主热备高可用环境部署

5、高可用性和双机热备浅析

6、SpringCloud应用双节点高可用部署测试

7、

8、在 Linux 中如何使用 HAProxy、Nginx 和 Keepalived 进行负载均衡?

9、Nginx实现高可用集群构建(Keepalived+Haproxy+Nginx)

5、数仓 | 几种常见的数据同步方式