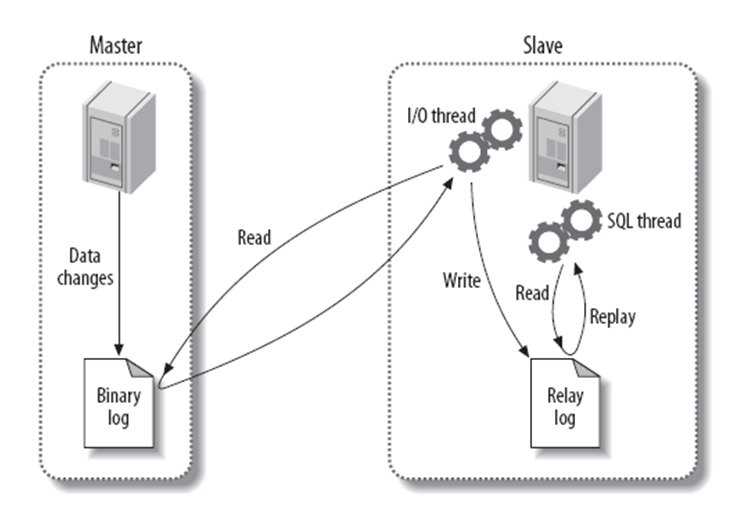
# Mysql架构及部署

# mysql主从同步介绍

主从同步使得数据可以从一个数据库服务器复制到其他服务器上，在复制数据时，一个服务器充当主服务器（master），其余的服务器充当从服务器（slave）。

MySQL数据库支持同步复制、异步复制，在复制的过程中一个服务器充当主服务，而一个或多个服务器充当从服务器。主服务器将更新写入二进制日志文件，并维护文件的一个索引以跟踪日志循环。这些日志可以记录发送到从服务器的更新。当一个从服务器连接主服务器时，它通知主服务器从服务器在日志中读取的最后一次成功更新的位置。从服务器接收从那时起发生的任何更新，然后封锁并等待主服务器通知新的更新。



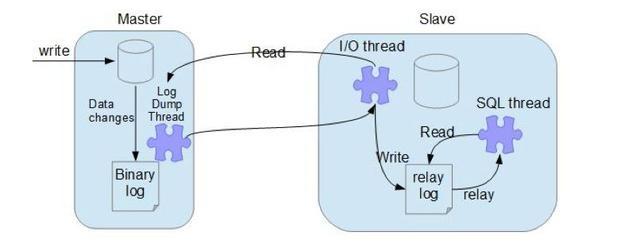
# 主从复制的作用

1. 数据实时备份，当系统中某个节点发生故障时，可以方便的故障切换
2. 实现读写分离，缓解主数据库压力，可以把读压力分摊到从数据库。
3. 高可用性和容错性 ：结合keepalive软件实现高可用性。

参考博文：<https://www.cnblogs.com/kevingrace/p/6710136.html>

# 主从同步原理

MySQL主从复制涉及到三个线程，一个运行在主节点（log dump thread），其余两个(I/O thread, SQL thread)运行在从节点，如下图所示:



**主节点 binary log dump 线程**

当从节点连接主节点时，主节点会创建一个log dump 线程，用于发送bin-log的内容。在读取bin-log中的操作时，此线程会对主节点上的bin-log加锁，当读取完成，甚至在发动给从节点之前，锁会被释放。

**从节点I/O线程**

当从节点上执行`start slave`命令之后，从节点会创建一个I/O线程用来连接主节点，请求主库中更新的bin-log。I/O线程接收到主节点binlog dump 进程发来的更新之后，保存在本地relay-log中。

**从节点SQL线程**

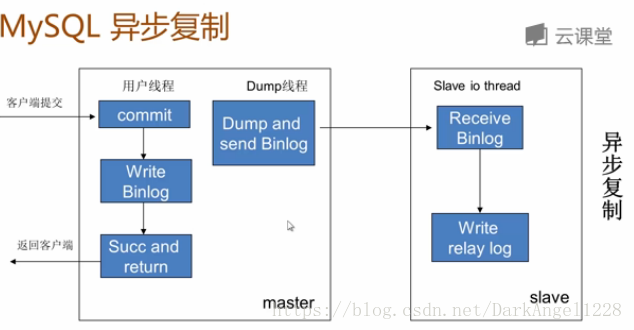
SQL线程负责读取relay log中的内容，解析成具体的操作并执行，最终保证主从数据的一致性。

**复制的基本过程如下：**

1. 从节点上的I/O 进程连接主节点，并请求从指定日志文件的指定位置（或者从最开始的日志）之后的日志内容；
2. 主节点接收到来自从节点的I/O请求后，通过负责复制的I/O进程根据请求信息读取指定日志指定位置之后的日志信息，返回给从节点。返回信息包含binlog日志文件和bin-log文件的position；从节点的I/O进程接收到内容后，将接收到的日志内容更新到本机的relay log中，并将读取到的binary log文件名和位置保存到master-info 文件中，以便在下一次读取的时候能够清楚的告诉Master“我需要从某个bin-log 的哪个位置开始往后的日志内容，请发给我”；
3. Slave 的 SQL线程检测到relay-log 中新增加了内容后，会将relay-log的内容解析成在祝节点上实际执行过的操作，并在本数据库中执行。

# Mysql主从同步模式

## 异步复制（Asynchronous replication）



1、逻辑上

MySQL默认的复制即是异步的，主库在执行完客户端提交的事务后会立即将结果返给给客户端，并不关心从库是否已经接收并处理，这样就会有一个问题，主如果crash掉了，此时主上已经提交的事务可能并没有传到从库上，如果此时，强行将从提升为主，可能导致新主上的数据不完整。

2、技术上

主库将事务 Binlog 事件写入到 Binlog 文件中，此时主库只会通知一下 Dump 线程发送这些新的 Binlog，然后主库就会继续处理提交操作，而此时不会保证这些 Binlog 传到任何一个从库节点上。

## 二、全同步复制（Fully synchronous replication）

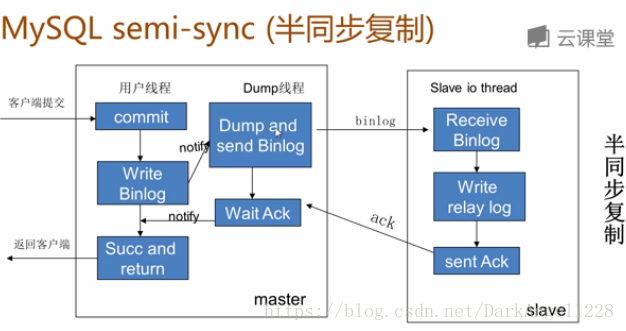
1、逻辑上

指当主库执行完一个事务，所有的从库都执行了该事务才返回给客户端。因为需要等待所有从库执行完该事务才能返回，所以全同步复制的性能必然会收到严重的影响。

2、技术上

当主库提交事务之后，所有的从库节点必须收到、APPLY并且提交这些事务，然后主库线程才能继续做后续操作。但缺点是，主库完成一个事务的时间会被拉长，性能降低。

## 三、半同步复制（Semisynchronous replication）



1、逻辑上

是介于全同步复制与全异步复制之间的一种，主库只需要等待至少一个从库节点收到并且 Flush Binlog 到 Relay Log 文件即可，主库不需要等待所有从库给主库反馈。同时，这里只是一个收到的反馈，而不是已经完全完成并且提交的反馈，如此，节省了很多时间。

2、技术上

介于异步复制和全同步复制之间，主库在执行完客户端提交的事务后不是立刻返回给客户端，而是等待至少一个从库接收到并写到relay log中才返回给客户端。相对于异步复制，半同步复制提高了数据的安全性，同时它也造成了一定程度的延迟，这个延迟最少是一个TCP/IP往返的时间。所以，半同步复制最好在低延时的网络中使用。

Mysql主从模式默认是异步复制的，而MySQL Cluster是同步复制的，从MySQL5.5开始，MySQL以插件的形式支持半同步复制。半同步复制兼顾了同步和性能的问题。

# Mysql部署

## 主从架构部署

主从架构部署参考博文：<https://www.cnblogs.com/keerya/p/7873502.html>

说明：在普通的主从架构下，如果主服务器挂了，从服务器是无法直接变主服务接收外部的请求的，需要手动切换为主服务器。参考博文：<https://blog.51cto.com/qiuyt/2148881>

## Mysql 双主+keepalived 高可用部署

在这种架构下，如果其中一台mysql实例无法对外提供服务，则另一台可以继续对外提供服务，无须手动干预。

部署参考博文：<https://gaoke.iteye.com/blog/2283890>

keepalive的作用是实现高可用,通过VIP虚拟IP的漂移实现高可用.在相同集群内发送组播包,master主通过VRRP协议发送组播包,告诉从主的状态.

**Keepalived高可用故障切换转移原理**

Keepalived高可用服务对之间的故障切换转移，是通过 VRRP (Virtual Router Redundancy Protocol ，虚拟路由器冗余协议）来实现的。在 Keepalived服务正常工作时，主Master节点会不断地向备节点发送（多播的方式）心跳消息，用以告诉备Backup节点自己还活看，当主Master节点发生故障时，就无法发送心跳消息，备节点也就因此无法继续检测到来自主Master节点的心跳了，于是调用自身的接管程序，接管主Master节点的 IP资源及服务。而当主Master节点恢复时，备Backup节点又会释放主节点故障时自身接管的IP资源及服务，恢复到原来的备用角色。

简单来讲：加了keepalive以后 ，外部访问mysql不是直接通过他们的ip，而是通过keepalive提供的虚拟ip来访问mysql数据库。Keepalive默认为其中一台mysql实例提供虚拟ip，这是外部通过虚拟ip访问的实例就是这一台实例。如果这个mysql实例节点挂了，keepalive会把这个虚拟ip提供给另外一台mysql使用，从而达到继续对外提供服务的能力。

## MySQL主从复制——主库已有数据的解决方案

由单机架构切换到一主一从或一主多从，在增加从库节点前，主库可能已经运行过一段时间，这种情况在实际业务中很常见。那么如何应对开启主从复制前主库有数据的场景呢？

第一种方案是选择忽略主库之前的数据，不做处理。这种方案只适用于不重要的可有可无的数据，并且业务上能够容忍主从库数据不一致的场景。

第二种方案是对主库的数据进行备份，然后将主数据库中导出的数据导入到从数据库，然后再开启主从复制，以此来保证主从数据库数据一致。

参考博文：<https://blog.csdn.net/qq_42606051/article/details/81279466>

# Mysql主从同步读写分离数据不一致性问题

## 读写分离

MySQL的主从复制和MySQL的读写分离两者有着紧密联系，首先部署主从复制，只有主从复制完了，才能在此基础上进行数据的读写分离。

简单来说，读写分离就是只在主服务器上写，只在从服务器上读，基本的原理是让主数据库处理事务性操作，而从数据库处理非事务性操作，然后再采用主从复制来把master上的事务性操作同步到slave数据库中。

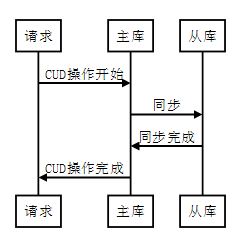
## 数据不一致性问题

主从同步架构下，从数据库需要不断向主数据库同步数据，由于网络延时等原因，造成主从数据库数据不一致问题

## 数据不一致解决方案

1、半同步复制

主从不一致的原因是延时引起的,所以要消除这个延时的影响，可以从主库进行CUD操作时进行规避，办法就是等主从同步完成之后，主库上的写请求再返回，就是大家常说的“半同步复制”semi-sync。

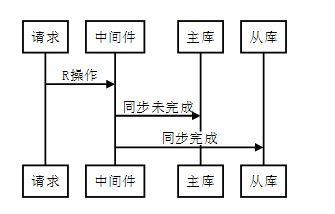


方案优点：利用数据库原生功能，比较简单

方案缺点：主库的写请求时延会增长，吞吐量会降低

2、数据库中间件

由中间件控制读操作时选择从主库查找还是从从库查找。



方案优点：能保证绝对一致

方案缺点：数据库中间件的成本比较高

3、缓存记录写key法

CUD操作

（1）将某个库上的某个key要发生写操作，记录在cache里，并设置“经验主从同步时间”的cache超时时间，例如500ms

（2）修改数据库

R操作

（1）先到cache里查看，对应库的对应key有没有相关数据

（2）如果cache hit，有相关数据，说明这个key上刚发生过写操作，此时需要将请求路由到主库读最新的数据

（3）如果cache miss，说明这个key上近期没有发生过写操作，此时将请求路由到从库，继续读写分离

方案优点：相对数据库中间件，成本较低

方案缺点：方案缺点：为了保证“一致性”，引入了一个cache组件，并且读写数据库时都多了一步cache操作

show status，Seconds\_Behind\_Master，你可以看到从库复制主库的数据落后了几ms