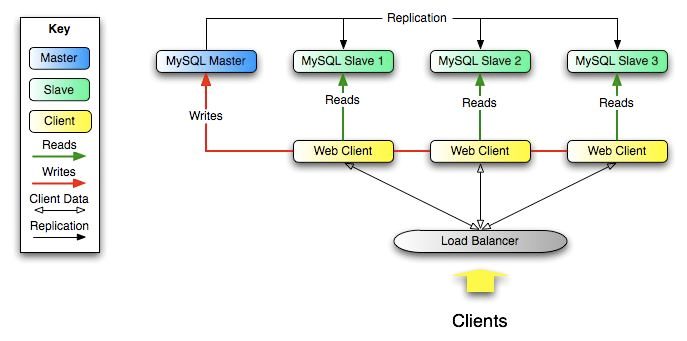
**MySQL主从同步那点事儿**

关于mysql主从同步，相信大家都不陌生，随着系统应用访问量逐渐增大，单台数据库读写访问压力也随之增大，当读写访问达到一定瓶颈时，将数据库的读写效率骤然下降，甚至不可用；为了解决此类问题，通常会采用mysql集群，当主库宕机后，集群会自动将一个从库升级为主库，继续对外提供服务；那么主库和从库之间的数据是如何同步的呢？本文针对MySQL 5.7版本进行下面的分析，下面随笔者一起探究一下mysql主从是如何同步的。

**MySQL主从复制原理**

为了减轻主库的压力，应该在系统应用层面做读写分离，写操作走主库，读操作走从库，下图为MySQL官网给出的主从复制的原理图，从图中可以简单的了解读写分离及主从同步的过程，分散了数据库的访问压力，提升整个系统的性能和可用性，降低了大访问量引发数据库宕机的故障率。



**binlog简介**

MySQL主从同步是基于binlog文件主从复制实现，为了更好的理解主从同步过程，这里简单介绍一下binlog日志文件。

binlog日志用于记录所有更新了数据或者已经潜在更新了数据（例如，没有匹配任何行的一个DELETE）的所有语句。语句以“事件”的形式保存，它描述数据更改，它是以二进制的形式保存在磁盘中。我们可以通过mysql提供的查看工具mysqlbinlog查看文件中的内容，例如 mysqlbinlog mysql-bin.00001 | more，这里注意一下binlog文件的后缀名00001，binlog文件大小和个数会不断的增加，当MySQL停止或重启时，会产生一个新的binlog文件，后缀名会按序号递增，例如mysql-bin.00002、mysql-bin.00003，并且当binlog文件大小超过 max\_binlog\_size系统变量配置时也会产生新的binlog文件。

**1. binlog日志格式**

（1）statement ： 记录每一条更改数据的sql

优点：binlog文件较小，节约I/O，性能较高。

缺点：不是所有的数据更改都会写入binlog文件中，尤其是使用MySQL中的一些特殊函数（如LOAD\_FILE()、UUID()等）和一些不确定的语句操作，从而导致主从数据无法复制的问题。

（2）row ： 不记录sql，只记录每行数据的更改细节

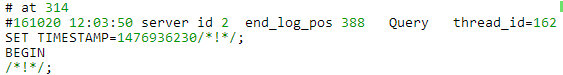
优点：详细的记录了每一行数据的更改细节，这也意味着不会由于使用一些特殊函数或其他情况导致不能复制的问题。

缺点：由于row格式记录了每一行数据的更改细节，会产生大量的binlog日志内容，性能不佳，并且会增大主从同步延迟出现的几率。

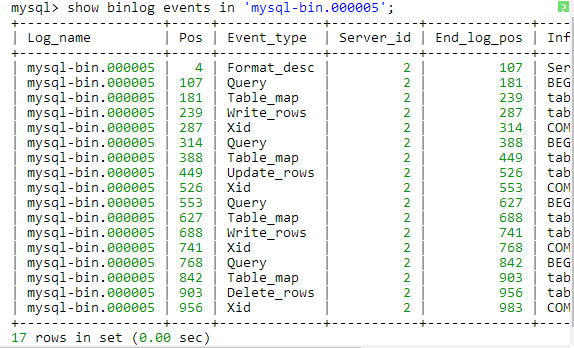
（3）mixed：一般的语句修改使用statment格式保存binlog，如一些函数，statement无法完成主从复制的操作，则采用row格式保存binlog，MySQL会根据执行的每一条具体的sql语句来区分对待记录的日志形式，也就是在Statement和Row之间选择一种。

**2. binlog日志内容**

mysqlbinlog命令查看的内容如下：



根据事件类型查看的binlog内容：



**3. binlog事件类型**

MySQL binlog记录的所有操作实际上都有对应的事件类型的，譬如STATEMENT格式中的DML操作对应的是QUERY\_EVENT类型，ROW格式下的DML操作对应的是ROWS\_EVENT类型，如果想了解更多请参考官方文档，有关binlog日志内容不在这里过多赘述，简单介绍一下是为了更好的理解主从复制的细节，下面我们进入正题。

**MySQL主从复制原理**

mysql主从复制需要三个线程，master（binlog dump thread）、slave（I/O thread 、SQL thread）。

**master**

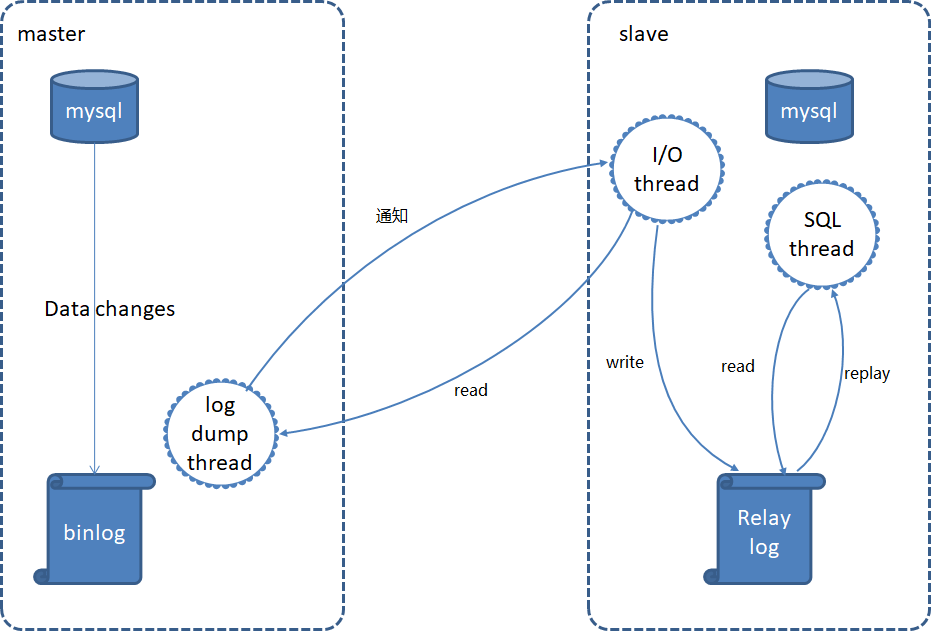
（1）binlog dump线程：当主库中有数据更新时，那么主库就会根据按照设置的binlog格式，将此次更新的事件类型写入到主库的binlog文件中，此时主库会创建log dump线程通知slave有数据更新，当I/O线程请求日志内容时，会将此时的binlog名称和当前更新的位置同时传给slave的I/O线程。

**slave**

（2）I/O线程：该线程会连接到master，向log dump线程请求一份指定binlog文件位置的副本，并将请求回来的binlog存到本地的relay log中，relay log和binlog日志一样也是记录了数据更新的事件，它也是按照递增后缀名的方式，产生多个relay log（ host\_name-relay-bin.000001）文件，slave会使用一个index文件（ host\_name-relay-bin.index）来追踪当前正在使用的relay log文件。

（3）SQL线程：该线程检测到relay log有更新后，会读取并在本地做redo操作，将发生在主库的事件在本地重新执行一遍，来保证主从数据同步。此外，如果一个relay log文件中的全部事件都执行完毕，那么SQL线程会自动将该relay log 文件删除掉。

下面是整个复制过程的原理图：



**主从同步延迟**

mysql的主从复制都是单线程的操作，主库对所有DDL和DML产生binlog，binlog是顺序写，所以效率很高，slave的I/O线程到主库取日志，效率也比较高，但是，slave的SQL线程将主库的DDL和DML操作在slave实施。DML和DDL的IO操作是随即的，不是顺序的，成本高很多，还可能存在slave上的其他查询产生lock争用的情况，由于SQL也是单线程的，所以一个DDL卡住了，需要执行很长一段事件，后续的DDL线程会等待这个DDL执行完毕之后才执行，这就导致了延时。当主库的TPS并发较高时，产生的DDL数量超过slave一个sql线程所能承受的范围，延时就产生了，除此之外，还有可能与slave的大型query语句产生了锁等待导致。

由于主从同步延迟是客观存在的，我们只能从我们自己的架构上进行设计， 尽量让主库的DDL快速执行。下面列出几种常见的解决方案：

1.业务的持久化层的实现采用分库架构，mysql服务可平行扩展，分散压力；

2.服务的基础架构在业务和mysql之间加入memcache或者Redis的cache层。降低mysql的读压力；

3.使用比主库更好的硬件设备作为slave；

4.sync\_binlog在slave端设置为0；

5.–logs-slave-updates 从服务器从主服务器接收到的更新不记入它的二进制日志；

6.禁用slave的binlog。