http://www.dataguru.cn/thread-467043-1-1.html

原来一直从事oracle database相关的工作，看到mySQL的redo log和binlog时有点糊涂。

在Oracle里面，redo log主要用于备份恢复后的业务重演，用于把DB恢复到点或者

某个时间点，而mysql里面的redo好像不起这个作用，做恢复用的是binlog.

百度了一把，结果如下:

首先，binlog会记录所有与MySQL数据库有关的日志记录，包括InnoDB、MyISAM、Heap等其他存储引擎的日志。

而InnoDB存储引擎的重做日志只记录有关该引擎本身的事务日志。

其次，记录的内容不同。无论用户将binlog文件记录的格式设为STATEMENT还是ROW，又或是MIXED，其记录的都是

关于一个事务的具体操作内容，即该日志是逻辑日志。而InnoDB存储引擎的重做日志是关于每个页（Page）的更改的物理情况。

此外，写入的时间也不同。binlog文件仅在事务提交后进行写入，即只写磁盘一次，不论这时该事务多大。而在事务进行的过程中，

却不断有重做日志条目（redo entry）被写入到重做日志文件中。

由此可见，redo log仅仅是innodb存储引擎的一个功能，而binlog才是对应oracle的redo log.

**binlog，redo log，undo log区别**

http://blog.csdn.net/mydriverc2/article/details/50629599

1. binlog是MySQL Server层记录的日志， redo log是InnoDB存储引擎层的日志。 两者都是记录了某些操作的日志(不是所有)自然有些重复（但两者记录的格式不同）。

2. 选择binlog日志作为replication我想主要原因是MySQL的特点就是支持多存储引擎，为了兼容绝大部分引擎来支持复制这个特性，那么自然要采用MySQL Server自己记录的日志而不是仅仅针对InnoDB的redo log，因为如果采用了InnoDB redo log复制，那么其他引擎也想复制，此时改怎么办呢？对吧

binlog属于逻辑日志，是逻辑操作。innodb redo属于物理日志，是物理变更。

逻辑日志有个缺点是难以并行，而物理日志可以比较好的并行操作，所以redo复制还是有优势的，也许5.7能搞出来。

什么是binlog

binlog日志用于记录所有更新且提交了数据或者已经潜在更新提交了数据（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。

binlog作用

1.恢复使能够最大可能地更新数据库，因为二进制日志包含备份后进行的所有更新。

2.在主复制服务器上记录所有将发送给从服务器的语句。

binlog 主要参数

log\_bin

设置此参数表示启用binlog功能，并指定路径名称

innodb\_flush\_log\_at\_trx\_commit = N：

N=0 – 每隔一秒，把事务日志缓存区的数据写到日志文件中，以及把日志文件的数据刷新到磁盘上；

N=1 – 每个事务提交时候，把事务日志从缓存区写到日志文件中，并且刷新日志文件的数据到磁盘上；

N=2 – 每事务提交的时候，把事务日志数据从缓存区写到日志文件中；每隔一秒，刷新一次日志文件，但不一定刷新到磁盘上，而是取决于操作系统的调度；

sync\_binlog = N：

N>0 — 每向二进制日志文件写入N条SQL或N个事务后，则把二进制日志文件的数据刷新到磁盘上；

N=0 — 不主动刷新二进制日志文件的数据到磁盘上，而是由操作系统决定；

推荐配置组合：

N=1,1 — 适合数据安全性要求非常高，而且磁盘IO写能力足够支持业务，比如充值消费系统；

N=1,0 — 适合数据安全性要求高，磁盘IO写能力支持业务不富余，允许备库落后或无复制；

N=2,0或2,m(0<m<100) — 适合数据安全性有要求，允许丢失一点事务日志，复制架构的延迟也能接受；

N=0,0 — 磁盘IO写能力有限，无复制或允许复制延迟稍微长点能接受，例如：日志性登记业务；

Undo Log

Undo Log是为了实现事务的原子性，在MySQL数据库InnoDB存储引擎中，还用UndoLog来实现多版本并发控制(简称：MVCC)。

-事务的原子性(Atomicity)

事务中的所有操作，要么全部完成，要么不做任何操作，不能只做部分操作。如果在执行的过程中发了错误，要回滚(Rollback)到事务开始前的状态，就像这个事务从来没有执行过。

-原理

Undo Log的原理很简单，为了满足事务的原子性，在操作任何数据之前，首先将数据备份到一个地方（这个存储数据备份的地方称为UndoLo）。

然后进行数据的修改。如果出现了错误或者用户执行了ROLLBACK语句，系统可以利用UndoLog中的备份将数据恢复到事务开始之前的状态。

除了可以保证事务的原子性，Undo Log也可以用来辅助完成事务的持久化。

-事务的持久性(Durability)

事务一旦完成，该事务对数据库所做的所有修改都会持久的保存到数据库中。为了保证持久性，数据库系统会将修改后的数据完全的记录到持久的存储上。

-用Undo Log

实现原子性和持久化的事务的简化过程

假设有A、B两个数据，值分别为1,2。

A.事务开始.

B.记录A=1到undolog.

C.修改A=3.

D.记录B=2到undolog.

E.修改B=4.

F.将undolog写到磁盘。

G.将数据写到磁盘。

H.事务提交

这里有一个隐含的前提条件：‘数据都是先读到内存中，然后修改内存中的数据，最后将数据写回磁盘’。

之所以能同时保证原子性和持久化，是因为以下特点：

A.更新数据前记录Undo log。

B.为了保证持久性，必须将数据在事务提交前写到磁盘。只要事务成功提交，数据必然已经持久化。

C.Undo log

必须先于数据持久化到磁盘。如果在G,H之间系统崩溃，undo log是完整的，可以用来回滚事务。

D.如果在A-F之间系统崩溃,因为数据没有持久化到磁盘。所以磁盘上的数据还是保持在事务开始前的状态。

缺陷：每个事务提交前将数据和Undo Log写入磁盘，这样会导致大量的磁盘IO，因此性能很低。

如果能够将数据缓存一段时间，就能减少IO提高性能。但是这样就会丧失事务的持久性。因此引入了另外一种机制来实现持久化，即

Redo log

记录的是新数据的备份。在事务提交前，只要将Redo Log持久化即可，不需要将数据持久化。当系统崩溃时，虽然数据没有持久化，

但是RedoLog已经持久化。系统可以根据RedoLog的内容，将所有数据恢复到最新的状态。

-Undo+Redo

事务的简化过程

假设有A、B两个数据，值分别为1,2.

A.事务开始.

B.记录A=1到undolog.

C.修改A=3.

D.记录A=3到redolog.

E.记录B=2到undolog.

F.修改B=4.

G.记录B=4到redolog.

H.将redolog写入磁盘。

I.事务提交

-Undo+Redo

事务的特点

A.为了保证持久性，必须在事务提交前将

RedoLog持久化。

B.数据不需要在事务提交前写入磁盘，而是缓存在内存中。

C.RedoLog保证事务的持久性。

D.UndoLog保证事务的原子性。

E.有一个隐含的特点，数据必须要晚于redolog写入持久存