# InnoDB存储引擎

如果InnoDB不是您的默认存储引擎，则可以通过在命令行上使用[--default-storage-engine=InnoDB](https://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html" \l "sysvar_default_storage_engine) 或在配置文件[default-storage-engine=innodb](https://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html" \l "sysvar_default_storage_engine) 的[mysqld]部分中 定义重新启动服务器 来确定数据库服务器或应用程序是否正常工作my.cnf。

由于更改默认存储引擎只会在创建表时影响新表，请运行所有应用程序安装和设置步骤，以确认所有安装都正确。然后运行所有应用程序功能，以确保所有的数据加载，编辑和查询功能都可以工作。如果一个表依赖于一些MyISAM特定的功能，你会收到一个错误; 将语句添加ENGINE=MyISAM到 [CREATE TABLE](https://dev.mysql.com/doc/refman/5.7/en/create-table.html" \o "13.1.18 CREATE TABLE语法)语句以避免错误。

如果您没有对存储引擎做出故意的决定，并且您只想预览某些表在下创建时如何工作InnoDB，请[ALTER TABLE table\_name ENGINE=InnoDB;](https://dev.mysql.com/doc/refman/5.7/en/alter-table.html" \o "13.1.8 ALTER TABLE语法)为每个表发出命令 。或者，为了运行测试查询和其他语句而不打扰原始表格，请制作如下所示的副本：

CREATE TABLE InnoDB\_Table （...） ENGINE=InnoDB AS SELECT \* FROM MyISAM\_Table;

要在真实的工作负载下，通过完整的应用程序获得真正的性能，请安装最新的MySQL服务器并运行基准测试。

测试完整的应用程序生命周期，从安装，大量使用和服务器重新启动。在数据库忙时模拟电源故障时，请停止服务器进程，并在重新启动服务器时验证数据是否已成功恢复。

测试任何复制配置，特别是如果您在主服务器和从属服务器上使用不同的MySQL版本和选项。

Oracle建议InnoDB作为典型数据库应用程序的首选存储引擎，从本地系统上运行的单用户维基和博客到推动性能极限的高端应用程序。在MySQL 5.7中，InnoDB是新表的默认存储引擎。

**重要**

InnoDB不能禁用。该 [--skip-innodb](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \l "option_mysqld_innodb) 选项已弃用，不起作用，其使用会导致警告。将在未来的MySQL版本中删除它。这也适用于它的同义词（--innodb=OFF， --disable-innodb等等）。

## 3、InnoDB多版本

InnoDB是一个 [多版本的存储引擎](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_mvcc" \o "MVCC)：它保留有关旧版本已更改行的信息，以支持事务功能，如并发和 [回滚](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_rollback" \o "回滚)。该信息存储在称为[回滚段](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_rollback_segment" \o "回滚段)（在Oracle中类似的数据结构之后）的数据结构中的表空间中 。InnoDB 使用回滚段中的信息执行事务回滚所需的撤消操作。它还使用这些信息来构建一行的较早版本以便 [一致性读取](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_consistent_read" \o "一致阅读)。

在内部，InnoDB为数据库中存储的每一行添加三个字段。6字节DB\_TRX\_ID字段指示插入或更新行的最后一个事务的事务标识符。此外，删除在内部被处理为更新，其中该行中的特殊位被设置为将其标记为已删除。每行还包含一个DB\_ROLL\_PTR称为滚动指针的7字节 字段。滚动指针指向写入回滚段的撤销日志记录。如果行已更新，则撤消日志记录包含在更新行之前重建该行的内容所需的信息。6字节DB\_ROW\_ID字段包含插入新行时单行增加的行ID。如果 InnoDB自动生成聚簇索引，则索引包含行ID值。 DB\_ROW\_ID

撤销回滚段中的日志分为插入和更新撤消日志。只有在事务回滚中才需要插入撤消日志，只要事务提交就可以将其删除。更新undo日志也用于一致的读取，但是只有在没有InnoDB分配了快照的事务之后，它们才能被丢弃 ，在一致的读取中可能需要更新undo log中的信息来构建数据库的早期版本行。

定期提交您的交易，包括仅发出一致性读取的交易。否则， InnoDB不能从更新撤销日志中丢弃数据，并且回滚段可能会增长太大，从而填满表空间。

回滚段中的撤消日志记录的物理大小通常小于相应的插入或更新行。您可以使用此信息来计算回滚段所需的空间。

在InnoDB多版本化方案中，当您使用SQL语句删除一行时，不会立即从数据库中物理删除一行。InnoDB只有当物理删除相应的行及其索引记录丢弃为删除写入的更新撤消日志记录时才会物理删除。这个删除操作被称为[清除](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_purge" \o "清除)，它是相当快的，通常采取与执行删除的SQL语句相同的时间顺序。

如果您在表中以相同的速率插入和删除小批量的行，则清除线程可能会开始滞后，并且由于所有“ 死 ”行，表可能会变得越来越大 ，从而使所有磁盘绑定和非常慢。在这种情况下，调节新行操作，并通过调整[innodb\_max\_purge\_lag](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \l "sysvar_innodb_max_purge_lag)系统变量将更多资源分配给清除线程 。有关详细信息[，](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \o "14.14 InnoDB启动选项和系统变量)请参见[第14.14节“InnoDB启动选项和系统变量”](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \o "14.14 InnoDB Startup Options and System Variables)。

多版本和二级索引

InnoDB多分支并发控制（MVCC）会将二次索引与聚簇索引不同。集群索引中的记录就地更新，并且其隐藏的系统列指向可以重建早期版本的记录的日志条目。与集群索引记录不同，辅助索引记录不包含隐藏的系统列，也不会在现场进行更新。

当二级索引列被更新时，旧的二级索引记录被删除标记，插入新记录，并且最终清除删除标记的记录。当二级索引记录被删除标记或辅助索引页由更新的事务更新时，InnoDB查找聚簇索引中的数据库记录。在聚簇索引中，DB\_TRX\_ID检查记录，如果记录在读取事务启动后被修改，则会从undo日志中检索正确的记录版本。

如果二次索引记录被标记为删除，或者次索引页由更新的事务更新， 则不使用[覆盖索引](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_covering_index" \o "覆盖指数)技术。不要从索引结构返回值，而是InnoDB查找聚簇索引中的记录。

然而，如果启用了 [索引条件下推（ICP）](https://dev.mysql.com/doc/refman/5.7/en/index-condition-pushdown-optimization.html" \o "8.2.1.5索引条件下推优化)优化，并且WHERE可以仅使用索引中的字段来评估部分条件，则MySQL服务器仍将该部分WHERE条件推送到使用指数。如果没有找到匹配的记录，则避免聚类索引查找。如果找到匹配的记录，即使在删除标记的记录中， InnoDB查找聚簇索引中的记录。

## 4、InnoDB架构

本小节主要介绍InnoDB的各种组件。

（1）Buffer Pool

缓冲池是内存中的一个缓冲区， InnoDB将表和索引的热点数据缓冲在这里，这样，当访问数据库数据时，就可以直接访问内存中的数据，而无需访问磁盘，这就减少了IO，提高了速度。通常，在专用的数据库服务器上，通常将多达80％的物理内存分配给InnoDB缓冲池。为了实现大容量内存操作的效率，缓冲池被组织为页面的链表，链表使用改进的LRU算法维护。

（2）Change Buffer

存储引擎设计中的一个挑战是写入操作期间的随机I/O。在InnoDB中，一个表将具有一个聚集索引和零个或多个辅助索引。这些索引中的每一个都是一个B+树。将记录插入到表中时，首先将该记录插入到聚簇索引中，然后再插入到每个辅助索引中。因此，所产生的I/O操作将随机分布在磁盘上。对于更新和删除操作，I/O模式也是随机的。为了减轻这个问题，InnoDB存储引擎使用一种称为Change Buffer的特殊数据结构（因为以前称为*插入缓冲区*，因此你将看到内部在很多场景下用ibuf和IBUF表示Change Buffer）。Change Buffer是另一个B+树，用于缓冲对辅助索引以及辅助索引页面相关的改变。InnoDB中只有一个Change Buffer，并且保留在系统表空间中。Change Buffer树的根页面在系统表空间（空间id为0）中，固定为FSP\_IBUF\_TREE\_ROOT\_PAGE\_NO（等于4）。当服务器启动时，通过使用此固定页码来加载Change Buffer树。你可以参考函数[ibuf\_init\_at\_db\_start()](https://github.com/mysql/mysql-server/blob/5.7/storage/innobase/ibuf/ibuf0ibuf.cc" \l "L495" \t "_blank) 了解更多详细信息。Change Buffer的总大小是可配置的，旨在确保完整的Change Buffer树可以驻留在内存中。使用[innodb\_change\_buffer\_max\_size](http://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \l "sysvar_innodb_change_buffer_max_size" \o "Innodb更改缓冲区最大大小" \t "_blank)系统变量配置Change Buffer的大小。

当更改的记录所在的页面不在buff pool（缓冲池）中时，这时按道理来说就要读写磁盘了，但是正如前面分析的，如果涉及到辅助索引页的修改（INSERT, UPDATE, DELETE），将会引起随机I/O操作，为此，InnoDB将这些更改缓冲到Change Buffer中，积累一段时间之后，将这些I/O操作进行合并，这样，将避免从磁盘读入辅助索引页面所需的大量随机访问I / O以及将辅助索引页面写入磁盘的随机写I/O。

在内存中，更改缓冲区占用InnoDB缓冲池的一部分 。在磁盘上，更改缓冲区是系统表空间的一部分，因此索引更改在数据库重新启动之间保持缓冲。

监控Change Buffer：

* InnoDB的标准监视器的输出包括Change Buffer的状态信息。要查看监控数据，请发出SHOW ENGINE INNODB STATUS命令。Change Buffer的状态信息位于INSERT BUFFER AND ADAPTIVE HASH INDEX 标题下方。输出的具体含义后边解释。
* [INFORMATION\_SCHEMA.INNODB\_METRICS](https://dev.mysql.com/doc/refman/5.7/en/innodb-metrics-table.html" \o "24.31.15 INFORMATION_SCHEMA INNODB_METRICS表) 表也提供了在InnoDB的标准监视器输出中的大部分数据 ，以及其他数据。要查看Change Buffer，请发出以下查询：

mysql> SELECT NAME, COMMENT FROM INFORMATION\_SCHEMA.INNODB\_METRICS WHERE NAME LIKE '%ibuf%'\G

* [INFORMATION\_SCHEMA.INNODB\_BUFFER\_PAGE](https://dev.mysql.com/doc/refman/5.7/en/innodb-buffer-page-table.html" \o "24.31.1 INFORMATION_SCHEMA INNODB_BUFFER_PAGE表) 表提供了InnoDB缓冲池中每个页面的元数据，Change Buffer页面包括 change buffer index和change buffer bitmap两种页面。Change Buffer页面由 PAGE\_TYPE的值决定，PAGE\_TYPE值为IBUF\_INDEX 表示change buffer index页面，值为IBUF\_BITMAP表示change buffer bitmap页面由。
* 警告

查询 [INNODB\_BUFFER\_PAGE](https://dev.mysql.com/doc/refman/5.7/en/innodb-buffer-page-table.html" \o "24.31.1 INFORMATION_SCHEMA INNODB_BUFFER_PAGE表)表可能会引起显着的性能开销。为了避免影响性能，请在测试环境的实例上重现问题并解决问题。

例如，可以通过查询 [INNODB\_BUFFER\_PAGE](https://dev.mysql.com/doc/refman/5.7/en/innodb-buffer-page-table.html" \o "24.31.1 INFORMATION_SCHEMA INNODB_BUFFER_PAGE表)表以确定 IBUF\_INDEX和 IBUF\_BITMAP页面占总缓冲池页面的百分比。

SELECT

(

SELECT COUNT(\*) FROM INFORMATION\_SCHEMA.INNODB\_BUFFER\_PAGE

WHERE PAGE\_TYPE LIKE 'IBUF%'

) AS change\_buffer\_pages,

(

SELECT COUNT(\*) FROM INFORMATION\_SCHEMA.INNODB\_BUFFER\_PAGE

) AS total\_pages,

(

SELECT ((change\_buffer\_pages/total\_pages)\*100)

) AS change\_buffer\_page\_percentage; +---------------------+-------------+-------------------------------+

| change\_buffer\_pages | total\_pages | change\_buffer\_page\_percentage | +---------------------+-------------+-------------------------------+

| 25 | 8192 | 0.3052 | +---------------------+-------------+-------------------------------+

Performance Schema提供了更高级的性能监控，其中包括Change Buffer 的mutex等待的指示，可以如下查询：

mysql> SELECT \* FROM performance\_schema.setup\_instruments

WHERE NAME LIKE '%wait/synch/mutex/innodb/ibuf%'; +-------------------------------------------------------+---------+-------+

| NAME | ENABLED | TIMED | +-------------------------------------------------------+---------+-------+

| wait/synch/mutex/innodb/ibuf\_bitmap\_mutex | YES | YES |

| wait/synch/mutex/innodb/ibuf\_mutex | YES | YES |

| wait/synch/mutex/innodb/ibuf\_pessimistic\_insert\_mutex | YES | YES | +-------------------------------------------------------+---------+-------+

有关监视InnoDB互斥等待的信息后边讲解。

（3）自适应哈希索引

自适应哈希索引是InnoDB基于对查询的统计分析，在内存中基于表上现有的B树索引构建的哈希索引 。 InnoDB可以根据InnoDB针对B树索引的搜索模式，为B树键的任意长度的前缀创建哈希索引。哈希索引可以仅覆盖经常访问的索引对应的那些页面。如果一个表刚好完全可以载入内存，那么自适应哈希索引通常是有益的。不过自适应哈希索引页需要维护，也要消耗资源，而且对于范围查找、LIKE运算，自适应哈希索引常常是无效的，而且，在多并发的情况下，对自适应哈希索引相关的锁的争用也比较严重，此时，关闭自适应哈希索引通常可以减少很多性能开销。由于难以预先确定此功能是否应该开启，因此请使用实际的工作负载进行基准测试来决定是启用还是禁用。由innodb\_adaptive\_hash\_index 选项启用 ，--skip-innodb\_adaptive\_hash\_index禁用。对于5.7，默认为启用。

MySQL [(none)]> SHOW variables like '%innodb\_adaptive\_hash\_index%';

+----------------------------------+-------+

| Variable\_name | Value |

+----------------------------------+-------+

| innodb\_adaptive\_hash\_index | ON |

| innodb\_adaptive\_hash\_index\_parts | 8 |

+----------------------------------+-------+

在早期，自适应哈希索引由单个锁保护，这就会导致并发性不足，为此，在5.7中，将自适应哈希索引分为多个区，每个区一个锁，通过将锁粒度变细来增加并发性。分区由innodb\_adaptive\_hash\_index\_parts 配置选项控制，默认为8，最大为512.如上语句输出。在SHOW ENGINE INNODB STATUS命令的SEMAPHORES输出部分可以监视自适应哈希索引的使用和锁的争用，如果看到很多线程正在等待锁，那么禁用自适应哈希索引通常是有益的。

（4）重做日志缓冲区

重做日志文件的主要目的是，万一实例或者介质失败（media failure），重做日志文件就能派上用场。如数据库由于所在主机掉电导致实例失败，InnoDB存储引擎会使用重做日志恢复到掉电前的时刻，以此来保证数据的完整性。默认情况下会有两个文件，名称分别为ib\_logfile0和ib\_logfile1。MySQL官方手册中将其称为InnoDB存储引擎的日志文件，不过更准确的定义应该是重做日志文件（redo log file）。重做日志缓冲区是用来保存要写入重做日志文件数据的内存区域。重做日志缓冲区大小由[innodb\_log\_buffer\_size](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \l "sysvar_innodb_log_buffer_size) 配置选项定控制。重做日志缓冲区会定期刷新到磁盘上的重做日志文件。大型重做日志缓冲区可以实现大型事务的运行，无需在事务提交之前不断的将重做日志写入磁盘。因此，如果有更新，插入或删除许多行的事务，使日志缓冲区更大，可以节省磁盘I / O。

[innodb\_flush\_log\_at\_trx\_commit](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \l "sysvar_innodb_flush_log_at_trx_commit) 选项控制如何将重做日志缓冲区的内容写入日志文件。[innodb\_flush\_log\_at\_timeout](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \l "sysvar_innodb_flush_log_at_timeout) 选项控制重做日志缓冲区刷新频率。

（5）系统表空间

InnoDB系统表空间包含InnoDB的数据字典（InnoDB数据对象的元数据），并且也是doublewrite buffer, change buffer, and undo logs的存储区域，系统表空间还包含用户创建的表和索引的数据，因此系统表空间是一个共享表空间，因为它被多个表（包括不同数据库中的表）共享。

系统表空间由一个或多个数据文件表示。默认情况下，MySQL 在data目录中创建一个名为ibdata1的系统数据文件。系统数据文件的大小和数量由[innodb\_data\_file\_path](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \l "sysvar_innodb_data_file_path)启动选项控制 。

（6）InnoDB数据字典

InnoDB数据字典由一系列用于跟踪InnoDB对象，如：表，索引，和表中的列的元数据的内部系统表组成，元数据物理上位于InnoDB系统表空间中。由于历史原因，数据字典元数据在某种程度上与存储InnoDB表的元数据文件（.frm文件）中的信息重叠 。

（7）Doublewrite Buffer

InnoDB中，在将缓冲池中的数据刷新到磁盘时是以页面（InnoDB的页面，通常为16KB）为单位的，这时可能会出现部分页面写入的问题。所谓部分页面写入是指向操作系统提交的页面写入请求仅部分完成。例如，在16K 的Innodb页面中，只有第一个4KB（文件系统的块通常为4KB）的块被写入磁盘，其他部分保持原来的状态。最常见的部分页面写入一般在发生电源故障时发生。也可能发生在操作系统崩溃时。另外，如果使用软件RAID，页面可能会出现在需要多个IO请求的条带边界上。如果硬件RAID没有电池备份，电源故障时也会发生这种情况。如果对磁盘本身发出单个写入，即使电源掉电，在理论上也应完成写入，因为驱动器内部应该有足够的电源来完成它。但是真的很难检查是否总是这样，因为它不是部分页面写入的唯一原因。在Innodb Doublewrite Buffer实施之前，确实会有数据损坏。

有的人会问，数据损坏可以使用重做日志来恢复呀，但是，请注意，InnoDB并不会将整个页面的内容写入重做日志，而是记录的对页面的操作，例如将某个偏移量处的值加2，使用重做日志进行恢复的基础是表空间中的实际数据页面在内部是完整的一致的，它是哪个页面版本无关紧要 ，但是如果页面不一致，则无法继续恢复，因为你的基础数据就是不一致的。为此引入了Doublewrite Buffer来解决问题。

理解了为什么需要Doublewrite Buffer，也就不难理解Doublewrite Buffer如何工作了。具体来说就是：你可以将Doublewrite Buffer视为系统表空间中的一个短期日志文件，它包含100个页的空间。当Innodb从Innodb缓冲池中刷新页面时，InnoDB首先会将页面写入双写缓冲区（顺序），然后调用fsync（）以确保它们保存到磁盘，然后将页面写入真正的数据文件并第二次调用fsync（））。现在Innodb恢复的时候会检查表空间中数据页面的内容和Doublewrite Buffer中页面的内容。如果在双写缓冲区中的页面不一致，则简单地丢弃它，如果表空间中的数据页面不一致，则从双写缓冲区中恢复。那么会不会出现都不一致的情况呢？这个不会，以内是先写Doublewrite Buffer，后写表空间中真实的数据页面，这样，当Doublewrite Buffer中不一致时表示系统崩溃了，也就无法继续执行了，就不会收到Doublewrite Buffer是否写成功的响应，也就不会发出真实的数据页面的写操作，这样的话必然不会出现二者都损坏的情况。

虽然Doublewrite Buffer的加入会使每次刷新数据时写两次磁盘，但是性能不会大幅下降，因为Doublewrite Buffer的写入是顺序的。所以一般来说，由于使用Doublewrite而不会超过5-10％的性能损失。但是数据是无价之宝，比起这个，这点损失可以接受。

那么Doublewrite是否可以禁用的？默认是开启的，要禁用Doublewrite，可以设置innodb\_doublewrite=0。当然了前提是你可以忍受数据丢失。

如果系统表空间文件（“ ibdata文件 ”）位于支持原子写入的Fusion-io设备上，则自动禁用Doublewrite Buffer，并将Fusion-io原子写入用于所有数据文件。因为双写缓冲区设置是全局的，因此对非Fusion-io硬件上的数据文件也将禁用Doublewrite Buffer。此功能仅在Fusion-io硬件上并且仅在Linux上启用Fusion-io NVMFS下受支持。要充分利用此功能，建议使用innodb\_flush\_method设置 O\_DIRECT。

因此最好不要禁用Doublewrite Buffer。除非可以忍受数据丢失。

（8）撤销日志

（9）每个表一个表空间文件（File-Per-Table Tablespaces）

File-Per-Table模式开启后，每一张表都会有自己的数据文件（table\_name.ibd文件）而不是将表的数据放在系统表空间中。[innodb\_file\_per\_table](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \l "sysvar_innodb_file_per_table)选项启用该模式。每个表的数据文件在数据库目录中创建。File-Per-Table模式支持DYNAMIC和 COMPRESSED行格式，从而支持诸如将可变长度数据存储在页面外和表压缩等功能。它的更多特点和优点后边分析。

（10）通用表空间

InnoDB共享的通用表空间由[CREATE TABLESPACE](https://dev.mysql.com/doc/refman/5.7/en/create-tablespace.html" \o "13.1.19 CREATE TABLESPACE Syntax) 创建，通用表空间可以创建在MySQL数据目录外的其他目录中，可以包含多个表的内容，并且支持所有的行格式。

表可以使用如下语句添加到表空间中：

[CREATE TABLE](https://dev.mysql.com/doc/refman/5.7/en/create-table.html" \o "13.1.18 CREATE TABLE Syntax)***[tbl\_name](https://dev.mysql.com/doc/refman/5.7/en/create-table.html" \o "13.1.18 CREATE TABLE Syntax)***[... TABLESPACE [=]](https://dev.mysql.com/doc/refman/5.7/en/create-table.html" \o "13.1.18 CREATE TABLE Syntax)***[tablespace\_name](https://dev.mysql.com/doc/refman/5.7/en/create-table.html" \o "13.1.18 CREATE TABLE Syntax)***

或者：

[ALTER TABLE](https://dev.mysql.com/doc/refman/5.7/en/alter-table.html" \o "13.1.8 ALTER TABLE Syntax)***[tbl\_name](https://dev.mysql.com/doc/refman/5.7/en/alter-table.html" \o "13.1.8 ALTER TABLE Syntax)***[TABLESPACE [=]](https://dev.mysql.com/doc/refman/5.7/en/alter-table.html" \o "13.1.8 ALTER TABLE Syntax)***[tablespace\_name](https://dev.mysql.com/doc/refman/5.7/en/alter-table.html" \o "13.1.8 ALTER TABLE Syntax)***

（11）undo表空间

默认情况下，undo log存储在系统表空间中，现在也可以将多个undo log存放在undo表空间（撤销表空间）中。undo表空间由[innodb\_undo\_tablespaces](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \l "sysvar_innodb_undo_tablespaces) 选项配置，如果该值为非0，那么MySQL启动时，InnoDB将会创建undo表空间。

（12）临时表空间

临时表空间用于非压缩 InnoDB临时表和相关对象。的[innodb\_temp\_data\_file\_path](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \l "sysvar_innodb_temp_data_file_path) 配置选项定义了临时表数据文件的相对路径。如果 [innodb\_temp\_data\_file\_path](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html" \l "sysvar_innodb_temp_data_file_path)未定义，ibtmp1则在数据目录中创建一个自动扩展的12MB数据文件 。在每个服务器上重新创建临时表空间，并接收动态生成的空间ID，这有助于避免与现有空间ID的冲突。临时表空间不能驻留在原始设备上。如果无法创建临时表空间，则拒绝启动。

临时表空间在正常关机或中止初始化时被删除。发生崩溃时，临时表空间不会被删除。在这种情况下，数据库管理员可以手动删除临时表空间或重新启动具有相同配置的服务器，从而删除并重新创建临时表空间。

**临时表撤消日志：**

临时表撤销日志用于临时表和相关对象。此类型的[撤销日志](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_undo_log" \o "撤销日志)不是重做日志，因为临时表在崩溃恢复期间不会恢复，并且不需要重做日志。但是，临时表撤销日志用于服务器运行时的回滚。这种特殊类型的非重做撤销日志通过避免临时表和相关对象的重做日志记录I / O来提高性能。临时表撤销日志驻留在临时表空间中。32个 [回滚段](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_rollback_segment" \o "回滚段) 保留用于修改临时表和相关对象的事务的临时表 [撤销日志](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_undo_log" \o "撤销日志)。

（13）redo log（重做日志）

重做日志是在崩溃恢复期间使用的基于磁盘的数据结构，以纠正不完整事务写入的数据。在正常操作期间，重做日志编码要更改InnoDB由SQL语句或低级API调用导致的表数据的请求 。在初始化期间和连接被接受之前，会自动重播在意外关闭之前未完成更新数据文件的修改。有关重做日志在崩溃恢复中的作用的信息，请参见[第14.18.2节“InnoDB恢复”](https://dev.mysql.com/doc/refman/5.7/en/innodb-recovery.html" \o "14.18.2 InnoDB恢复)。

默认情况下，重做日志物理磁盘上的一组文件，命名为代表的ib\_logfile0和 ib\_logfile1。MySQL以循环方式写入重做日志文件。重做日志中的数据以影响的记录编码; 该数据统称为重做。通过重做日志的数据通过越来越多的[LSN](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_lsn" \o "LSN)值来表示。

有关相关信息，请参阅：

* [第14.6.1节“InnoDB启动配置”](https://dev.mysql.com/doc/refman/5.7/en/innodb-init-startup-configuration.html" \o "14.6.1 InnoDB启动配置)
* [第8.5.4节“优化InnoDB重做日志记录”](https://dev.mysql.com/doc/refman/5.7/en/optimizing-innodb-logging.html" \o "8.5.4优化InnoDB重做日志记录)
* [第14.7.2节“更改InnoDB重做日志文件的数量或大小”](https://dev.mysql.com/doc/refman/5.7/en/innodb-data-log-reconfiguration.html" \o "14.7.2更改InnoDB重做日志文件的数量或大小)
* [InnoDB崩溃恢复](https://dev.mysql.com/doc/refman/5.7/en/innodb-recovery.html" \l "innodb-crash-recovery" \o "InnoDB崩溃恢复)

**重做日志刷新的组提交**

InnoDB像任何其他与 [ACID](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_acid" \o "酸)兼容的数据库引擎一样，在事务提交之前刷新[重做日志](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_redo_log" \o "重做日志)。InnoDB 使用[组提交](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_group_commit" \o "组提交) 功能将多个这样的刷新请求组合在一起，以避免每次提交一次刷新。使用组提交时，会 InnoDB发出单次写入日志文件以对同时提交的多个用户事务执行提交操作，从而显着提高吞吐量。

有关性能COMMIT和其他事务操作的更多信息 ，请参见[第8.5.2节“优化InnoDB事务管理”](https://dev.mysql.com/doc/refman/5.7/en/optimizing-innodb-transaction-management.html" \o "8.5.2优化InnoDB事务管理)。

## 5、InnoDB锁和事务模型

要实现大规模、繁忙或高度可靠的数据库应用程序，并调整MySQL的性能，那么了解InnoDB锁和InnoDB 事务模型将是非常重要的 。

（1）InnoDB锁

本小节介绍InnoDB使用的各种锁类型。

共享锁和独占锁

6、InnoDB配置指南

本节详细介绍InnoDB相关的配置，包括：初始化、启动、各个功能及组件的配置等。

（1）InnoDB启动配置