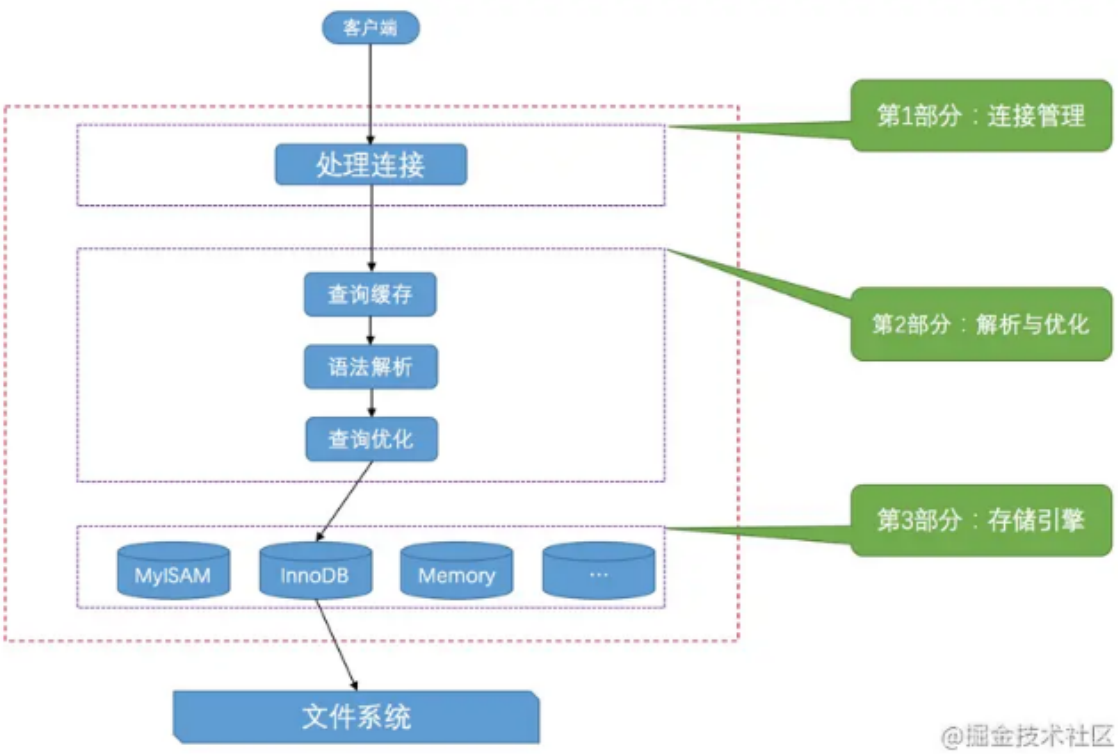
第一章：

1、客户端——服务器交互流程



2、存储引擎：表处理器

- 连接管理、缓存、语法解析、查询优化，划分为**MySQL Server**的功能；
- 数据的存储形式、读写操作，划分为**存储引擎**的功能。

	MyISAM	InnoDB	Memory
	//todo第二节内容		

3、索引分类

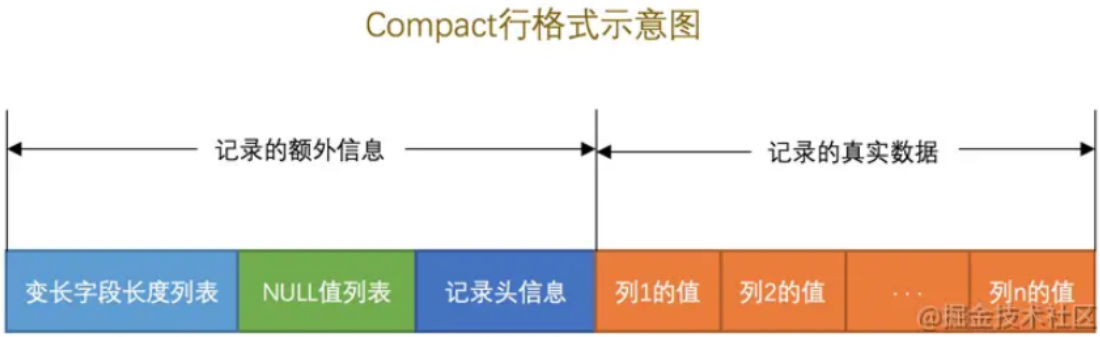
- **聚簇索引**：被索引的列必须是主键列，没有主键会指定一个唯一的为空键代替，聚簇索引是根据主键来创建的一颗B+树，叶子节点里面保存的是实际的一行数据；
- **辅助索引**：InnoDB引擎对于非主键列建立的索引，叶子节点中保存的是主键值，根据这个主键值再进行回表查找聚簇索引。
- **非聚簇索引**：MyISAM的索引结构，MyISAM引擎将索引和数据分开存储。叶子节点保存的是索引键的值和实际数据的内存地址。MyISAM把实际数据以表格形式存在一起，按照插入顺序排列。通过非聚簇索引找到数据表的行，在定位到实际的行读取数据。
- **覆盖索引**：用于建立联合索引的列，覆盖了需要查询的列，无需回表
  如：建立 name + sex + age 的索引
  slect name,sex,age from stu where name ='xzd'；是覆盖索引
  select name,birthday from stu where name = 'xzd'；不是覆盖索引

4、字符集与字符编码

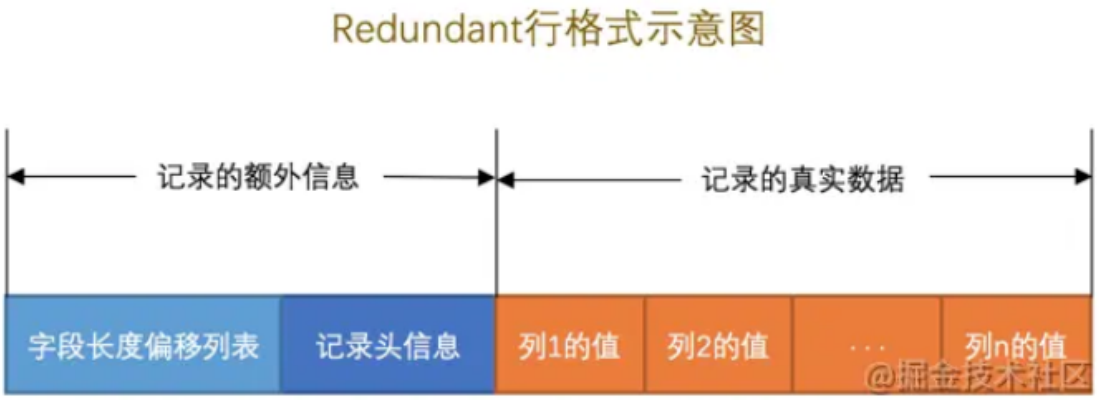
- **字符集**：所有抽象字符的集，不同的字符集收录的字符不一样，ASCII字符集只有128个字符，且没有收录中文，unicode字符集收录的字符比较全
- **字符编码**：建立字符(key)与二进制值(value)的映射关系，通过字符编码规则，可以将字符转换成二进制，保存在系统中，常用的字符编码：utf8是unicode字符集的一种编码方式，utf8使用1~4字节表示一个字符。

5、InnoDB行记录结构

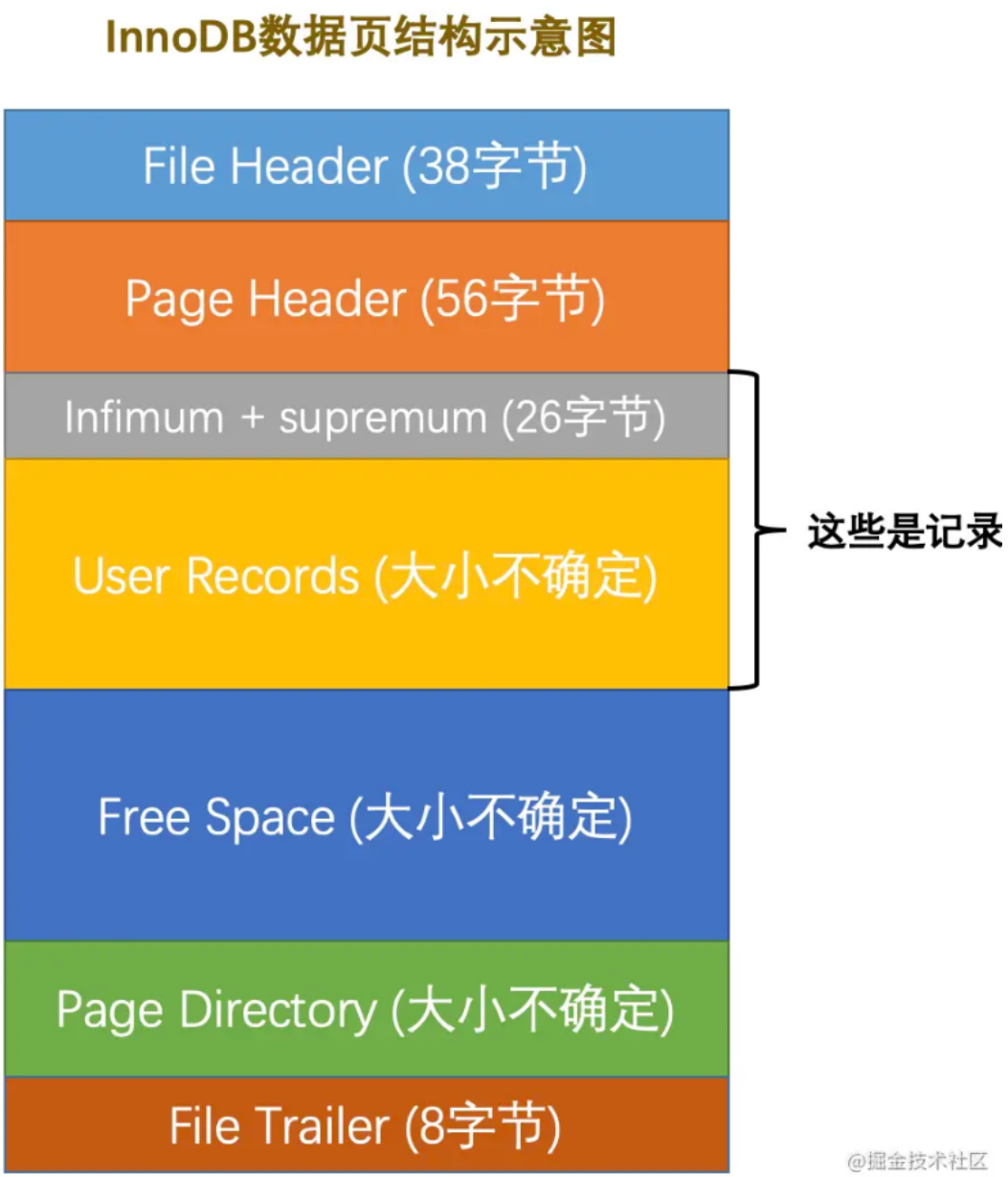
1. **COMPACT行格式**



   - 记录头信息包含：delete_mask，标记记录是否被删除；n_owned，标记当前记录拥有的记录数；next_record，标记下一条记录的位置
   - 真实数据包含：row_id，当建表没有指定主键和唯一键时，会自动添加一个隐藏主键row_id作为主键用于建立索引
2. **Redundant行格式**

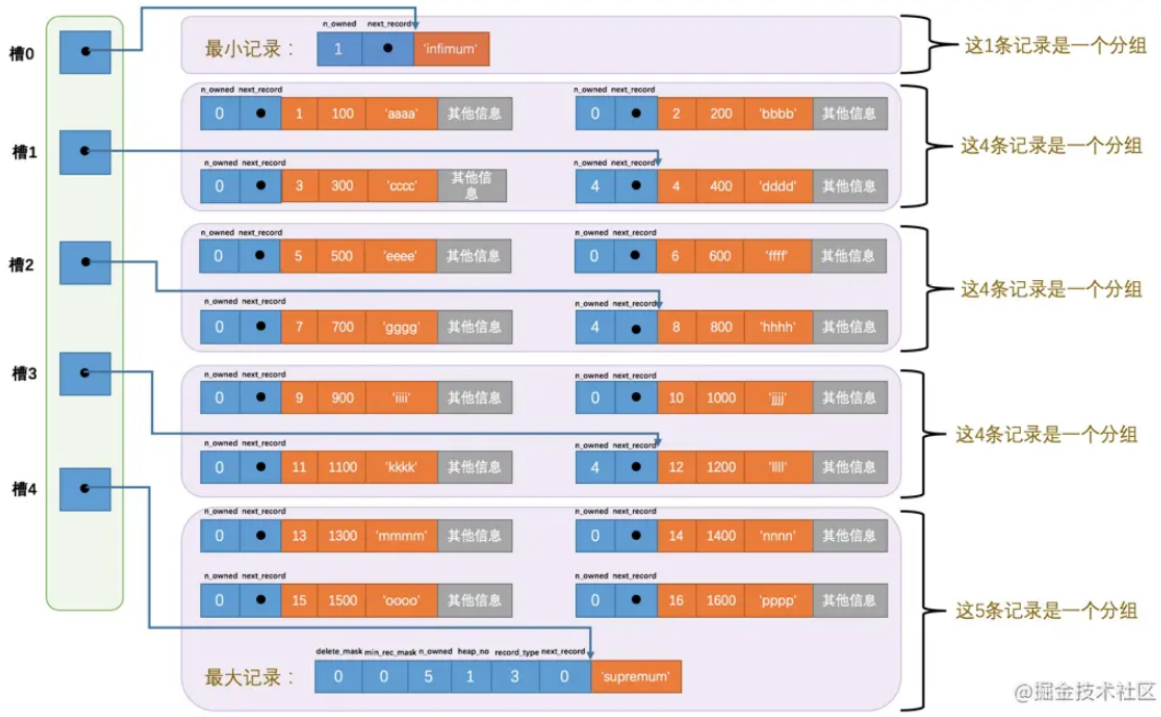


6、InnoDB数据页结构



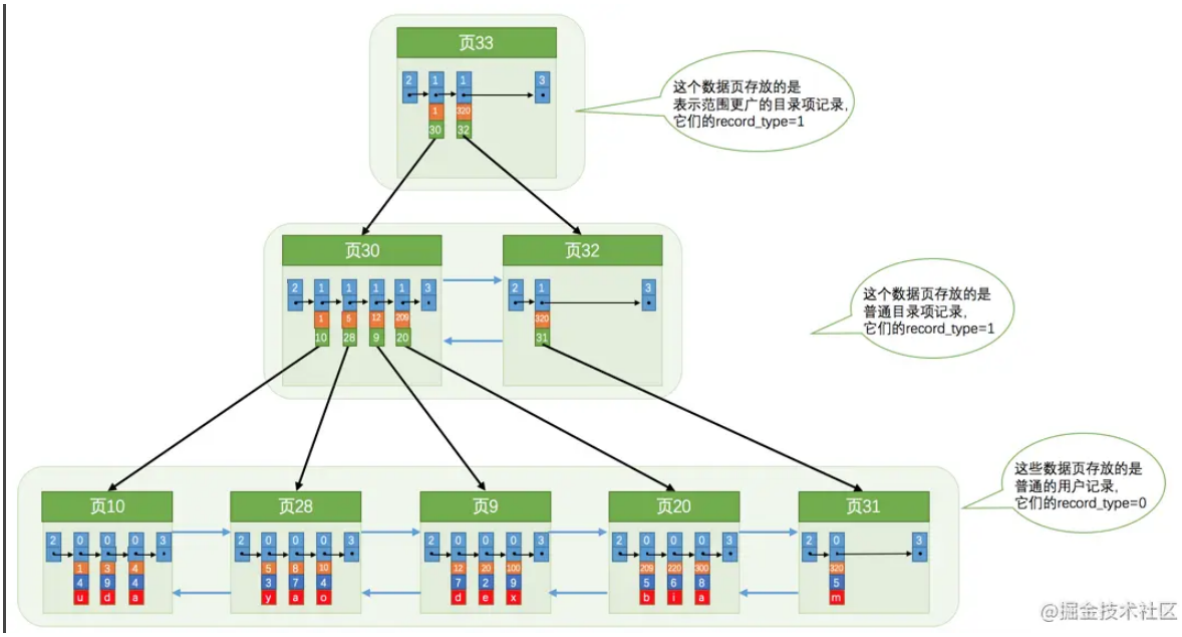
- **File Header**：页面通有信息，如：校验和（用于与Trailer的校验和对比）、上下页页号
- **Page Header**：页面专有信息，如：有多少记录，有多少槽；
- **Infimum+supremum**：最小记录和最大记录
- **User Records**：真实记录数据
- **FreeSpace**：空闲空间
- **Page Directory**：页目录，相当于书籍目录索引的页码表，由槽（表示数据在页中的位置，相当于页码）组成
- **File Trailer**：页尾，校验页是否完整

7、使用PageDirectory进行查找的过程



通过二分法先定位到槽2，对比主键,发现target主键>槽2的最后一个数据，那么继续二分法定位到槽1，对比主键，然后在通过遍历槽内链表结构的数据，定位到具体数据。所以，槽就相当于页码，通过页码找到数据，再看数据是否是符合要求的。

8、B+树结构



- 实际用户记录存在于最底层的叶子节点
- 用来存放目录项的节点称为非叶子节点
- 页内的记录都是按逐渐大小顺序排列成链表
- 页与页之间是双向链表

9、连接join的原理

首先，将驱动表和被驱动表的记录都取出来，依次匹配的组合加入结果集，形成一个笛卡尔积。

内连接：把不符合ON条件 或 WHERE条件的记录过滤掉；

左/右连接：把结果集中，保留符合ON条件或WHERE条件的驱动表的所有记录，对应的被驱动表中的记录若为空，则用NULL补充。

对于外连接的驱动表的记录来说，如果无法在被驱动表中找到匹配ON子句中的过滤条件的记录，那么该记录仍然会被加入到结果集中，对应的被驱动表记录的各个字段使用NULL值填充；

10、InnoDB的缓存机制Buffer pool

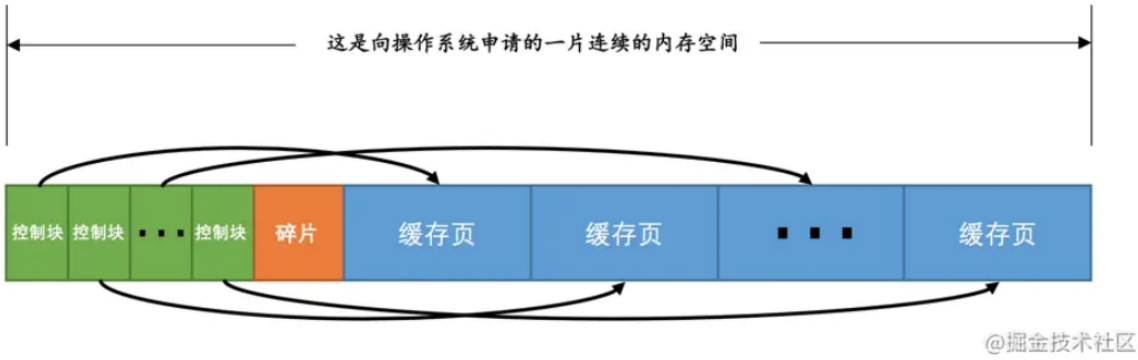
MySQL访问数据以页为基本单位，当需要访问某个页的数据时，就会把完整的页数据加载到内存中，完成访问后不急于把该页对应的内存空间释放掉，而是缓存起来。缓存的位置在内存中被叫做BufferPool。

表空间号 + 页号 作为key，缓存页作为value创建一个hash表，从而快速在buffer pool中找到缓存页。如果找不到，那就从free链表中选一个空闲页，再把磁盘中对应的缓存页加载进去。

free链表：把所有空闲的缓存页对应的控制块作为节点放到一个链表中；

flush链表：BufferPool中某个缓存页的数据被修改后，就和磁盘上的页不一样了，这样的被叫做脏页。脏页对应的控制块会被作为节点加入到一个链表中，叫做flush链表，意味着将来某个时间点会被刷新到磁盘的。

LRU链表：BufferPool中缓存页的组织形式。为了保留最经常被访问的缓存也，采用最近最少使用算法淘汰冷数据。



11、事务ACID

需要保证原子性、隔离性、一致性、持久性的一个或多个数据库操作称之为一个事务

原子性Atomicity： 一个事务中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。 事务过程中出错会回滚到事务开始前。MySQL通过undo日志来保证原子性

隔离性Isolation：数据库允许多个事务并发执行，隔离性可以防止多个事务并发时由于交叉执行而导致数据不一致。隔离级别分为读未提交、读已提交、可重复度、串行。

一致性Consistency：事务开始前和事务结束后，数据库的完整性没有被破坏。意味着事务操作必须完全符合所有预设规则，如唯一性、非空、两个账户余额总和不变。

持久性Durability：对数据库的修改，会被持久化到磁盘中。MySQL通过redo日志来保证持久性。

12、事务隔离级别

- **读未提交**：read uncommitted，最低级别，没有并发控制能力。会读到脏数据
- **读已提交**：read committed，访问到的是事务提交后的数据，避免脏数据，Oracle默认隔离级别。通过MVCC实现
- **可重复读**：repeatable read，MySql默认级别。通过MVCC实现
- **串行**：serializable，串行执行，吞吐量较低

13、并发下的事务问题

- **脏读：**读了未提交的数据。事务A修改了数据，但未提交，此时事务B来访问，读到了事务A未提交的脏数据。
- **不可重复读：**读了已提交的数据。一个事务还未执行完成，但过程中访问的数据被另一个事务更改：事务A读取数据，此时事务B对这个数据进行修改并提交成功，事务A再来读这个数据，两次读

取的数据不一样，重复读的数据不一致。可通过行锁解决

- **幻读：**涉及到插入/删除动作。一个事务还未执行完成，但过程中其他事务进行了插入/删除；事务A第一次访问数据，数据为空（或数据有N条），此时事务B进行插入（或删除）动作，事务A再来读取，发现数据多了（或是少了）。通过行锁解决不了

不可重复读针对update操作，使用行级锁即可解决；幻读针对insert与delete操作，需要使用表级锁。

14、事务隔离级别的并发问题

	脏读	不可重复读	幻读
Read uncommitted	存在	存在	存在
Read committed	不存在	存在	存在
Repeatable read	不存在	不存在	存在
Serializable	不存在	不存在	不存在

15、redo日志

概念：InnoDB中的重做日志，用于实现事务的持久性，日志文件由两部分组成：内存中的redo日志缓存 + 磁盘中的redo日志文件。redo是 `物理日志`，记录的是“在某个数据页上做了什么修改”

作用：MySQL为了提升性能，数据的修改不会立即同步到磁盘，而是保存在bufferPool中，为了防止宕机造成数据丢失，引入redo log机制，对数据进行修改前会记录redo日志，在事务commit之前会先把redo日志刷到磁盘中。系统崩溃后，根据磁盘中的redo日志可以恢复最新的数据

16、undo日志

概念：InnoDB中的回滚日志，就是对用于实现事务原子性和MVCC，undo日志主要记录的是数据的逻辑变化，为了在发生错误时回滚之前的操作， 会将修改之前的操作都记录下来， 然后在发生错误时才可以回滚。 undo日志是 `逻辑日志` ，简单来说就是记录sql语句，如一条Insert语句，对应一条Delete的undo log。

作用：记录事务修改之前的数据信息，因此加入由于系统错误或者rollback而回滚，可以根据undo日志的信息来恢复之前的数据状态。

分类：

insert undo log：Insert操作记录没有历史版本，只对当前事务本身可见。在事务提交后直接删除

update undo log：Update操作和Delete操作产生的log，delete操作相当于打个标记并没有实际删除。

17、undo及redo事务简化过程

假设有A、B两个数据，值分别为1、2
开始一个事务，操作内容为：把1修改为3 ， 2修改为4，那么实际的记录如下：

1.事务开始；

2.undo:记录A = 1；
3.修改A = 3；
4.redo: 记录A = 3；

5.undo: 记录B = 2；
6.修改B = 4；
7.redo: 记录B = 4；

8.写binlog；
9.将redo log写入磁盘；
9.事务提交；

18、redo log 和 binlog的区别

	redo log	binlog
日志类型	物理日志，记录值的变化	逻辑日志，记录的是sql
文件大小	redo log大小是固定的	可通过
功能所属	InnoDB引擎层	MySQL Server层
记录方式	采用循环写的方式记录，当写到结尾时，会回到开头循环写日志	通过追加的方式记录，当文件大小大于给定值后，会创建新的文件
适用场景	用于崩溃恢复 crash-safe	用于主从复制和数据恢复

由 `binlog` 和 `redo log` 的区别可知：`binlog` 日志只用于归档，只依靠 `binlog` 是没有 `crash-safe` 能力的。但只有 `redo log` 也不行，因为 `redo log` 是 `InnoDB` 特有的，且日志上的记录落盘后会被覆盖掉。因此需要 `binlog` 和 `redo log` 二者同时记录，才能保证当数据库发生宕机重启时，数据不会丢失。

在binlog写入磁盘之前，redo log已经准备好在内存中；
在binlog写入磁盘之后，redo log才开始刷盘；

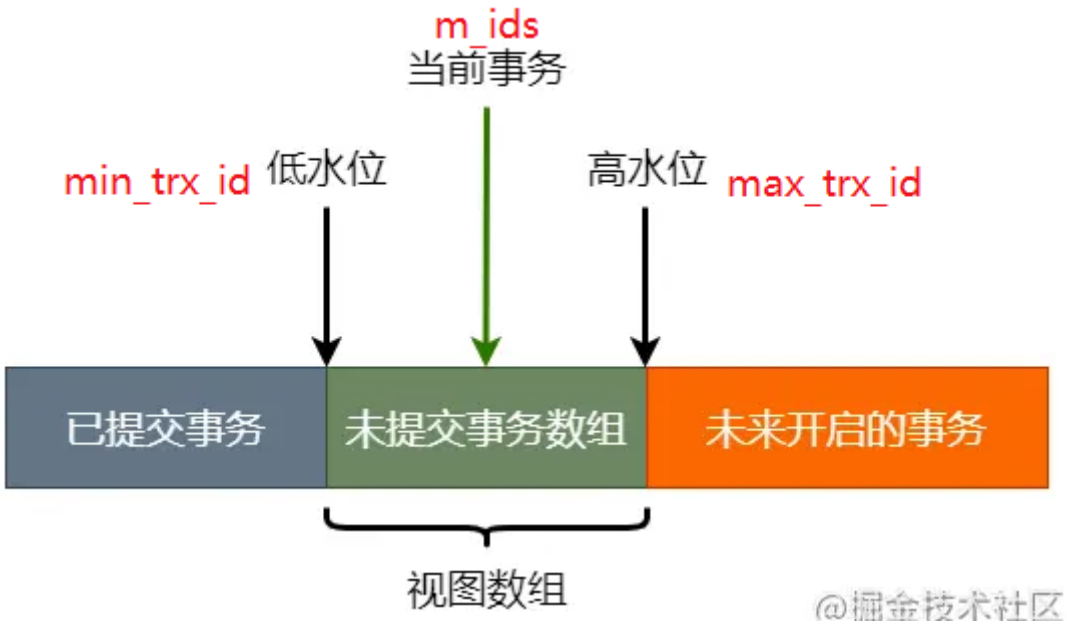
19、MVCC原理

概念：
多版本并发控制Multi-Version Concurrency Control。
指的就是在使用ReadCommitted、RepeatableRead这两种隔离级别的事务在执行普通SELECT操作（快照读）时，访问记录的版本链的过程。该过程依赖undo log 和视图ReadView。

原理：
通过ReadView来确定数据的可见性，使用undo log来读取旧版本数据。
每条记录有一个trx_id字段和roll_pointer字段，trx_id记录最近修改了这条数据的事务ID，rool_pointer指向undo日志，用于遍历历史版本的数据。

作用：
1.解决**读-写冲突**的无锁控制（乐观锁解决**写-写冲突**的无锁控制）。每个修改保存一个版本，版本与事务id关联，读操作只读readView开始前的数据库的快照。
2.实现事务隔离级别RC 、RR。

ReadView视图:
记录当前活跃的事务，用于确定数据版本的可见性。RC级别和RR级别生成ReadView的时机不一样。
包含 `m_ids` 活跃的事务id列表、`min_trx_id` 活跃事务中最小的事务ID、`max_trx_id` 应该分配给下一个事务的ID值、`creator_trx_id` 生成这个ReadView的事务ID。



使用ReadView进行快照读的规则：

1.被访问数据的trx_id与Read_View的creator_trx_id相同，说明是自己的数据，可见；
2.被访问数据的trx_id < min_trx_id说明早已经提交，可见；
3.被访问数据的trx_id >= max_trx_id，说明该事务数据在生成ReadView之后开启的，不可见；
4.被访问数据的trx_id < max_trx_id 并 trx_id > min_trx_id，判断trx_id是否在m_ids中：
4.1如果在，说明事务还在活跃，不可见；
4.2如果不在，说明该版本已提交，可见。

不同的事务隔离级别生成ReadView的时机：ReadView的时机决定不同max_trx_id和m_ids列表。

Read Committed：每次select都会生成一个ReadView。保证每次都读取到最新的已提交数据。

Repeatable Read：在第一次select时生成ReadView ，整个事务期间只有一个ReadView。查询值承认在事务启动前就已经提交完成的数据。

快照读&当前读：

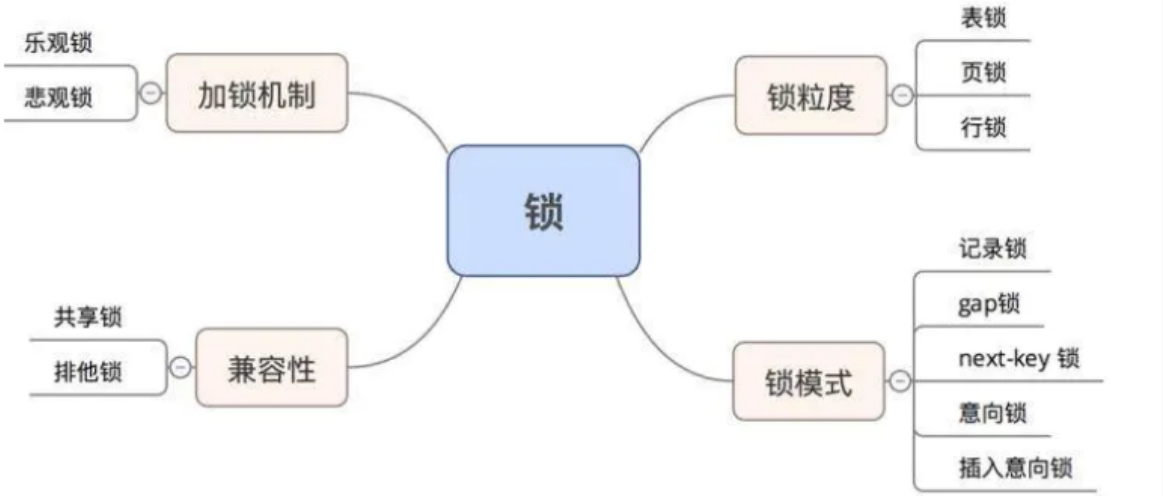
1.MVCC中的快照读，通过MVCC机制读取快照数据，是属于乐观锁的读取方式。
2.MVCC中的当前读，通过加锁来进行控制，读的是最新版本，并且对读取的记录加锁，阻塞其他事务。

当前读的情况：select ... lock in share mode; select ... for update; update; delete; insert

当前读操作都是先读后写（读是更新语句执行，不是我们手动执行），读的就是当前版本的值，叫当前读；而我们普通的查询语句就叫快照读。

20、MySQL的锁

锁的分类：



锁的分类

兼容性：

共享锁：也叫S锁、读锁。共享锁与共享锁兼容，与排它锁不兼容

排它锁：也叫X锁、写锁。与共享锁和排它锁皆不兼容

意向共享锁、意向排它锁： 为了允许行锁和表锁共存，实现多粒度锁机制 。如果不在**表**上加意向锁，对**表**加锁的时候，都要去检查表中的**某一行**上是否加有行锁，意图锁的主要目的是某个事务正在锁定表中的一行，或者将要锁定表中的一行。

锁粒度：

表锁：MyISAM只支持表锁。

行锁：InnoDB行锁是通过给索引上的索引项加锁来实现的，只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB将使用表锁。默认情况下Innodb不会上行锁。可通过语句指定加锁：select... lock in share mode、select...for update。

加锁机制：

乐观锁：一种用来解决**写-写冲突**的无锁并发控制。数据记录会有一个版本号，每次对数据的修改都会版本号加一。事务在提交修改前，会检查版本号是否被改变，如果被改变则重试或回滚。

悲观锁：排它锁。使用select...for update或者update语句会触发悲观锁。

锁模式：

间隙锁GAP：用范围条件检索数据请求共享或排他锁时，InnoDB会给符合范围条件的已有记录的索引项加锁。

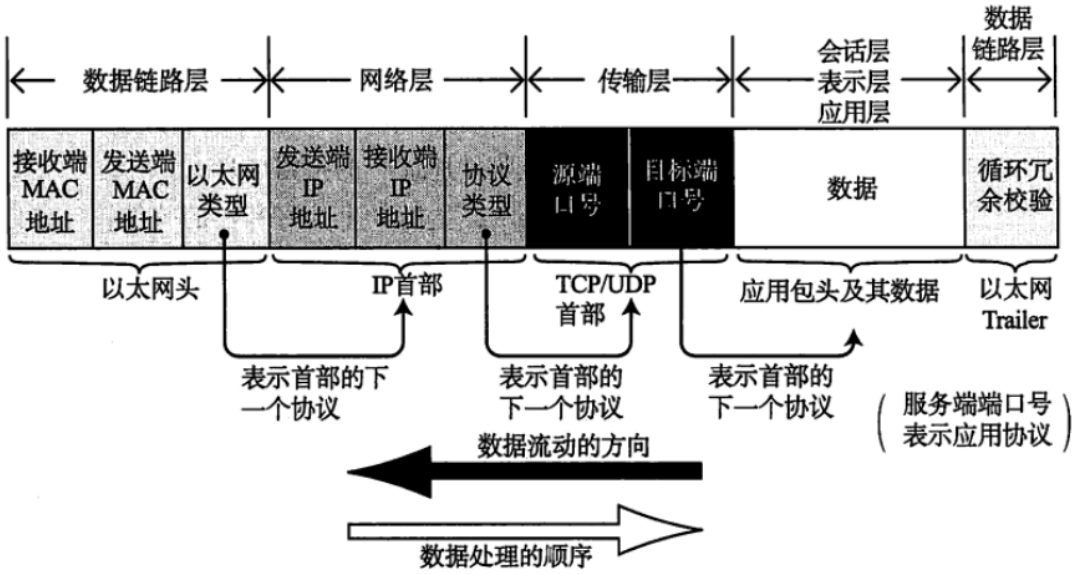
意向锁：意图锁的主要目的是某个事务正在锁定表中的一行，或者将要锁定表中的一行。事务给某一行加锁的时候，需要先取得相应的意向锁。

第二章：

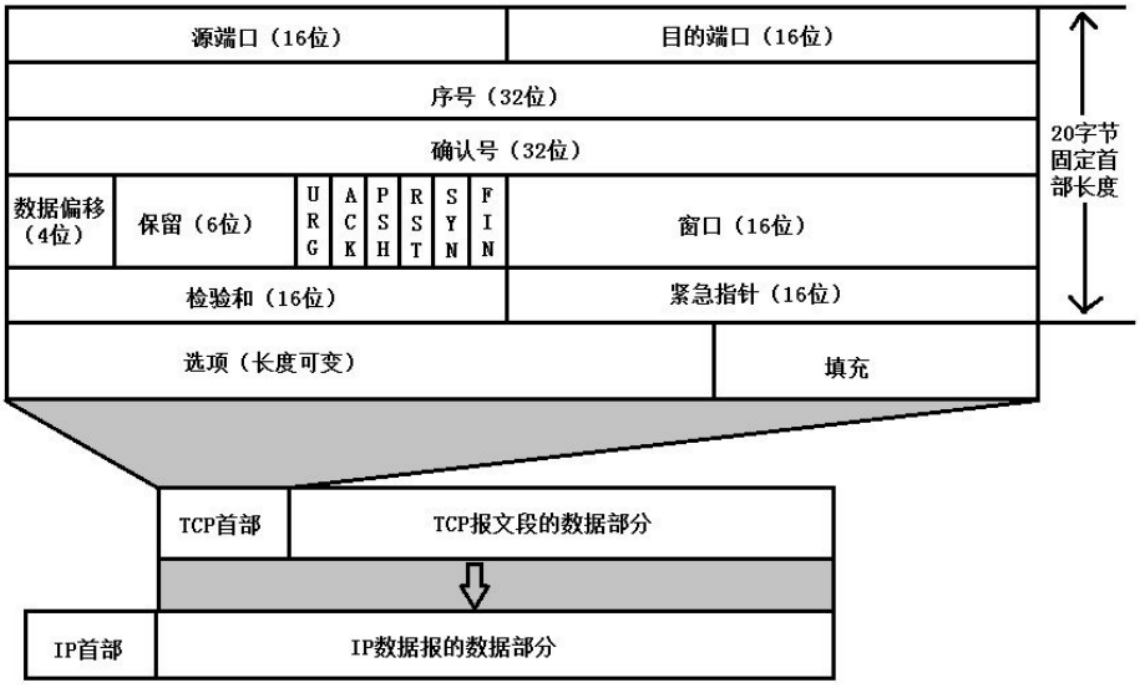
1、OSI七层模型

2、TCP/IP四层

3、数据包结构

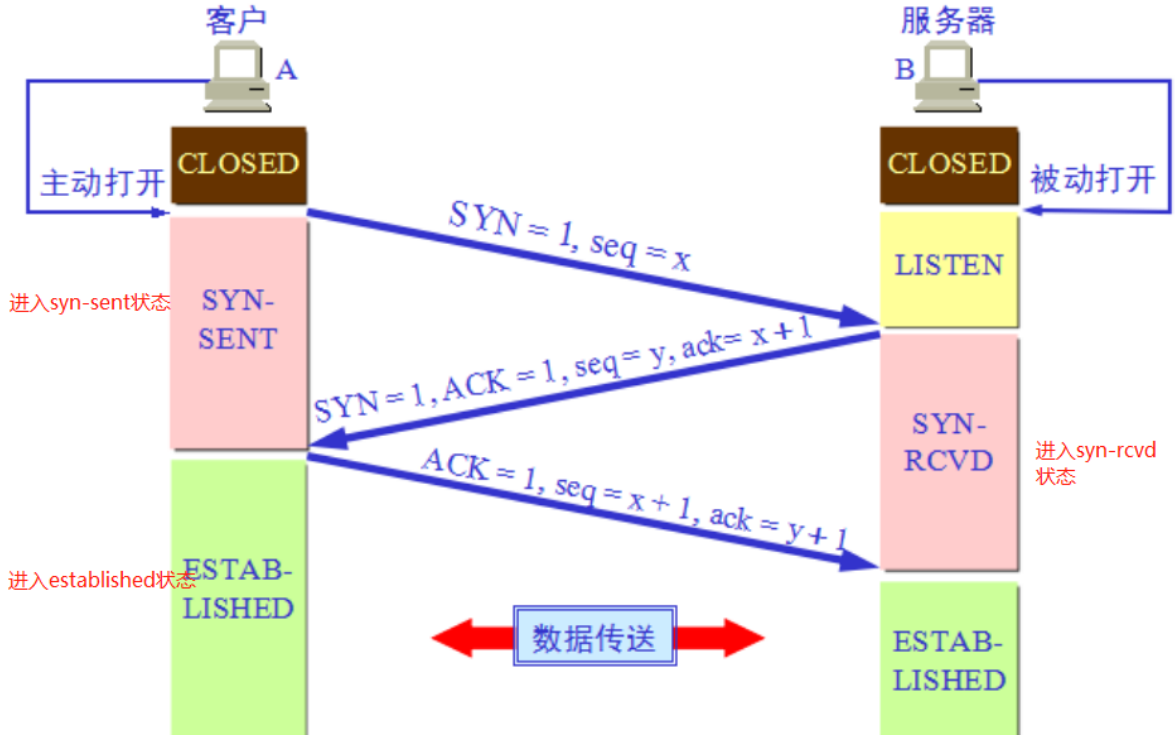


4、TCP首部结构



- 端口号：用来标记应用进程。
- 序列号seq：标识当前传输报文段中的第一个数据字节，随机产生seq并在三握时通过SYN传给接收端。保证TCP传输的有序性。
- 确认应答号ack：即acknumber，表明下一个期待收到的字节序号。
- 数据偏移：表示TCP数据部分应该从哪个位算起，可看成TCP首部的长度。
- 控制位：控制标志，ACK=1表示确认号有效，SYN=1表示建立连接，FIN=1表示释放连接。
- 窗口：滑动窗口大小，用于告知发送端的缓存大小，用于流量控制。
- 校验和：校验TCP报文段。

5、三次握手



为什么不能用两次握手：

因为三次握手完成两个事：

1.协商交换序列号seq，这个seq在握手中被发送和确认。

2.通知彼此都已准备好，防止ACK丢失导致S端资源占用。如果两次握手，S端应答ACK后进入就绪状态，但ACK丢失，C端不知道已经连接成功，会占用服务器资源。

1.微服务、分布式、集群。通俗理解

微服务是设计层面的，分布式是部署层面的。分布式是微服务的实现方式。

1.假设饭店是一个完整的系统，那么饭店的工作细分为切菜、烧菜、端菜、洗碗，这些是不同的服务（**微服务**）。

2.这些服务由不同的人（服务器）来干，就是**分布式**（不同服务在不同服务器上面）。

3.烧菜的一个不够就多招几个人来干，就是**集群。**

分布式实现了服务之间互不影响，专职专责。集群是为了降低同一份工作的压力。