15-数据库

刘亚雄

极客时间-Java 讲师

五、MySQL锁篇

5.1 MySQL锁简介

01-什么数据库的锁

在实际的数据库系统中，**每时每刻都在发生着锁**，当某个用户在修改一部分数据时，MySQL会通过锁定防止其他用户读取同一数据。

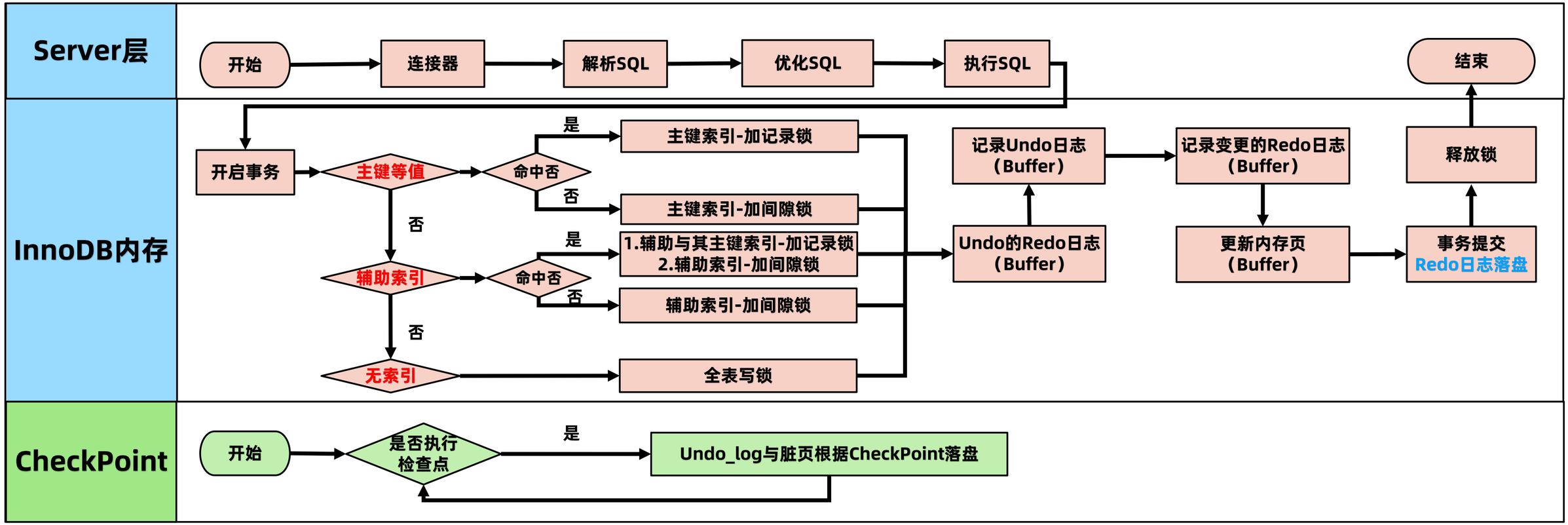
在处理并发读或者写时，通过由两种类型的锁组成的锁系统来解决问题：**共享锁、排他锁**

为什么说，数据库中每时每刻都在发生锁？

5.2 一条Update语句的执行流程

一条更新语句

```
1 update tab_user set ='曹操' where id = 1;
```



5.3 MySQL-锁分类

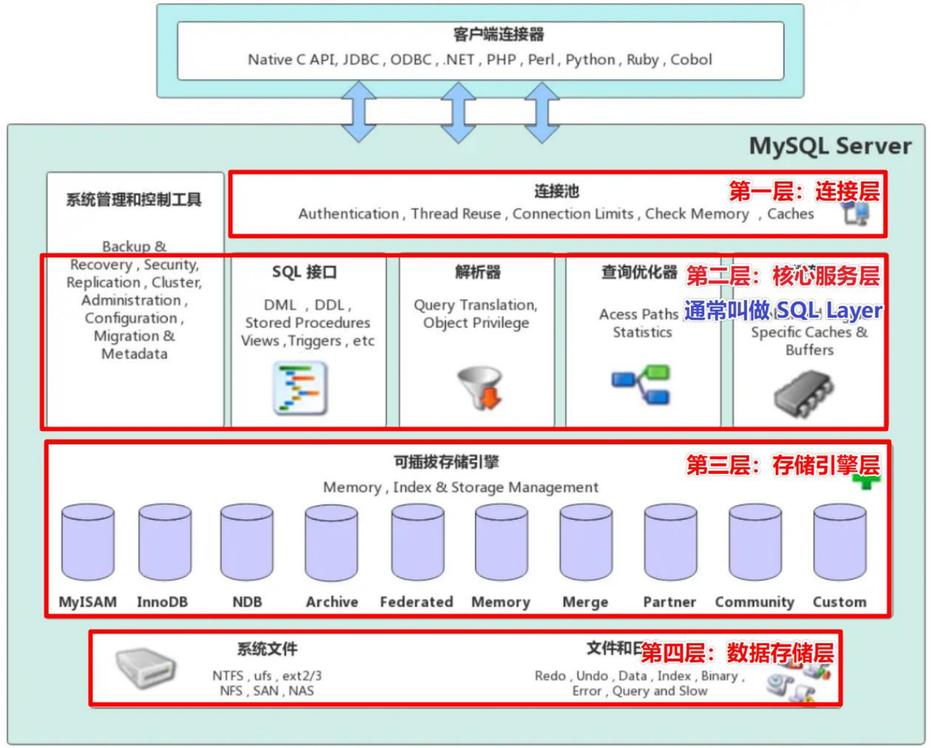
02-锁有哪些？

按锁功能划分

- **共享锁（shared lock）也叫S锁、读锁，读锁是共享的，读锁之间互相不阻塞**
  - 加锁方式：select ... lock in share mode
- **排他锁（exclusive lock）也叫X锁、写锁，写锁是排他的，写锁阻塞其他的读和写锁**
  - 加锁方式：select ... for update

按粒度分

- 全局锁：锁DB，由SQL Layer层实现
- 表级锁：锁Table，由SQL Layer层实现
- 行级锁：锁Row的索引，由存储引擎实现
  - 记录锁（Record Locks）：锁定索引中一条记录
  - 间隙锁（Gap Locks）：仅仅锁住一个索引区间
  - 临键锁（Next-Key Locks）：记录锁和间隙锁的组合，**解决幻读问题**
  - 插入意向锁(Insert Intention Locks)：做insert时添加的对记录id的锁
  - 意向锁：存储引擎级别的“表级”锁



5.4 MySQL-全局锁

01-什么是全局锁？

全局锁是对整个数据库实例加锁，加锁后整个实例就处于只读状态，将阻塞DML、DDL及已经更新但未提交的语句

典型应用：全库逻辑备份

02-加锁与解锁命令

加锁命令：flush tables with read lock;

释放锁命令：unlock tables;

注意：断开Session锁自动释放全局锁

全库备份怎么锁，为什么要锁定？

```
# 提交请求锁定所有数据库中的所有表，以保证数据的一致性，全局读锁
mysqldump -uroot -p --host=localhost --all-databases --lock-all-tables > /root/db.sql
# 一致性视图
mysqldump -uroot -p --host=localhost --all-databases --single-transaction > /root/db.sql
```

5.5 MySQL-表级锁

01-表级锁

- 表读锁（Table Read Lock），阻塞对当前表的写，但不阻塞读
- 表写锁（Table Write Lock），阻塞对当前表的读和写
- 元数据锁（Meta Data Lock，MDL)不需要显式指定，在访问表时会被自动加上，作用保证读写的正确性
  - 当对表做**增删改查**操作的时**加元数据读锁**
  - 当对表做**结构变更**操作的时**加元数据写锁**
- 自增锁(AUTO-INC Locks) AUTO-INC是一种特殊的表级锁，自增列事务性插入操作时产生

02-表锁相关命令

- 查看表锁定状态：show status like 'table_locks%’;
- 添加表读锁：lock table t read;
- 添加表写锁：lock table t write;
- 查看表锁情况：show open tables;
- 删除表锁：unlock tables;

```
mysql> show status like 'table%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Table_locks_immediate      | 99    | 产生表级锁定的次数
| Table_locks_waited         | 0     | 出现表级锁定争用而发生等待的次数
| Table_open_cache_hits      | 0     |
| Table_open_cache_misses    | 0     |
| Table_open_cache_overflows | 0     |
+----------------------------+-------+
5 rows in set (0.01 sec)
```

案例：锁表

5.6 MySQL-行锁【重点】

01-行级锁

- MySQL的行级锁是由存储引擎实现，**InnoDB行锁**是通过给索引上的**索引项加锁来实现**
- 特点：**只有通过索引条件检索的数据InnoDB才使用行级锁，否则InnoDB都将使用表锁**
- **按范围分：**记录锁（Record Locks）、间隙锁（Gap Locks）、临键锁（Next-Key Locks）、插入意向锁（Insert Intention Locks）
- **按功能分：**
  - 读锁：允许事务**去读**目标行，阻止其他事务**更新**。阻止其他事务加写锁，但不阻止加读锁
  - 写锁：允许事务**更新**目标行，阻止其他事务**获取或修改**。同时阻止其他事务加读锁和写锁。

02-如何加行锁？

- 对于Update、Delete和Insert语句，InnoDB会自动给涉及数据集加**写锁**
- 对于普通Select语句，**InnoDB不会加任何锁**
- 事务手动给Select记录集加读锁或写锁

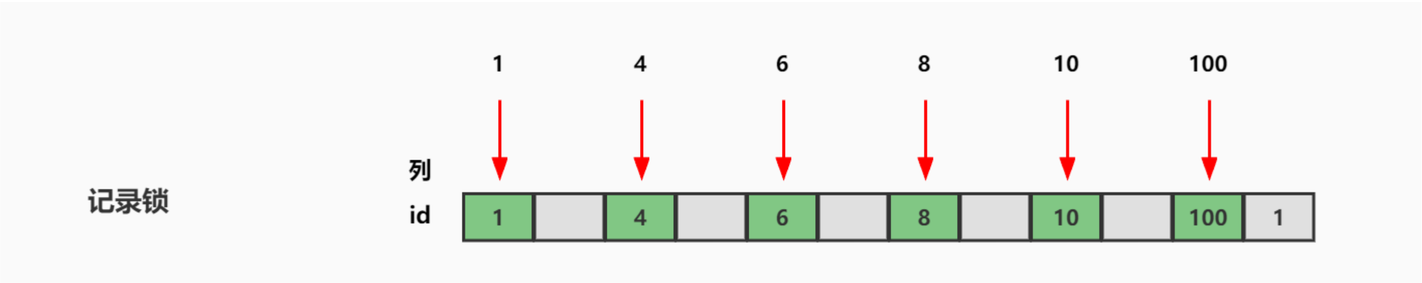
案例：Select的行读写锁

5.6 MySQL-行锁【重点】

03-行锁四兄弟：记录锁、间隙锁、临键锁、插入意向锁

➢ 记录锁（Record Locks）仅仅锁住索引记录的一行

- 记录锁锁住的永远是索引，而非记录本身，即使该表上没有任何显示索引
- 没有索引，InnoDB会创建隐藏列ROWID的聚簇索引

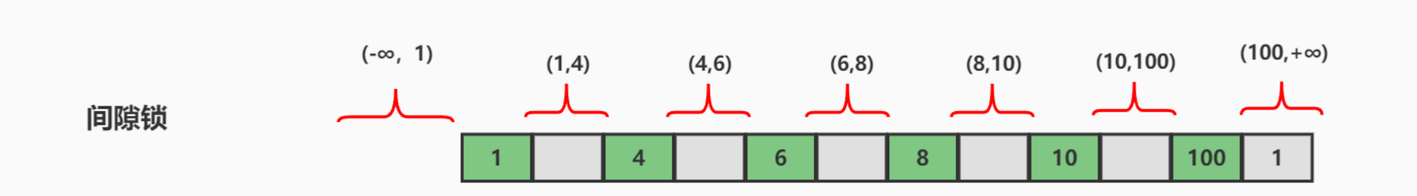



```sql
-- 加记录读锁
select * from t1_simple where id = 1 lock in share mode;
-- 加记录写锁
select * from t1_simple where id = 1 for update;
-- 新增，修改，删除加记录写锁
insert into t1_simple values (2, 22);
update t1_simple set pubtime=33 where id =2;
delete from t1_simple where id =2
```

5.6 MySQL-行锁【重点】

03-行锁四兄弟：记录锁、间隙锁、临键锁、插入意向锁

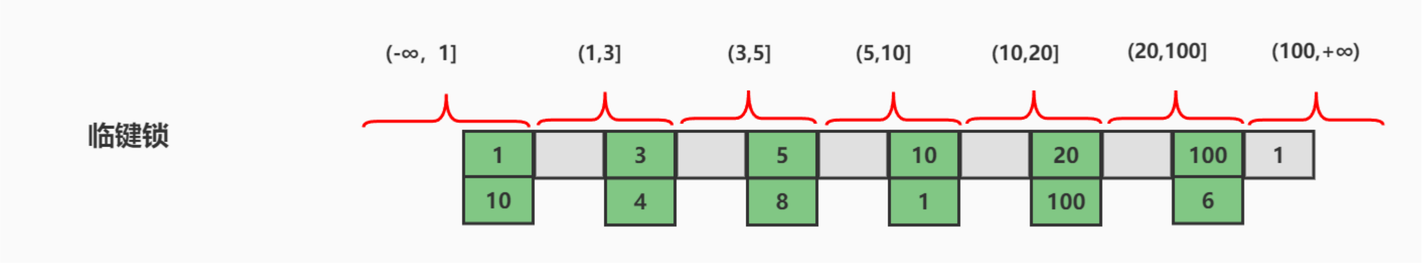
- 间隙锁（Gap Locks）仅仅锁住一个索引区间，开区间，不包括双端端点和索引记录
  - 在索引记录间隙中加锁，并不包括该索引记录本身
  - 间隙锁可用于防止幻读，保证索引间隙不会被插入数据



5.6 MySQL-行锁【重点】

03-行锁四兄弟：记录锁、间隙锁、临键锁、插入意向锁

- 临键锁（Next-Key Locks）相当于记录锁 + 间隙锁，左开右闭区间
  - 默认情况下，InnoDB使用临键锁来锁定记录，但会在不同场景中退化
  - 场景01-唯一性字段等值（=）且记录存在，退化为**记录锁**
  - 场景02-唯一性字段等值（=）且记录不存在，退化为**间隙锁**
  - 场景03-唯一性字段范围（< >），还是**临键锁**
  - 场景04-非唯一性字段，默认是**临键锁**

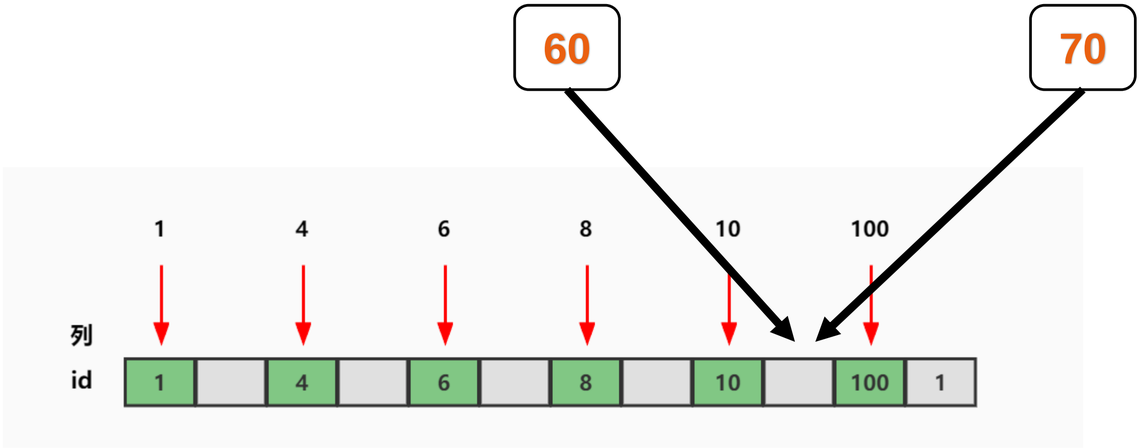


5.6 MySQL-行锁【重点】

03-行锁四兄弟：记录锁、间隙锁、临键锁、插入意向锁

- 间隙锁（Gap）可以在一定程度上解决幻读问题，但间隙锁就是最佳方案吗？还有优化空间吗？

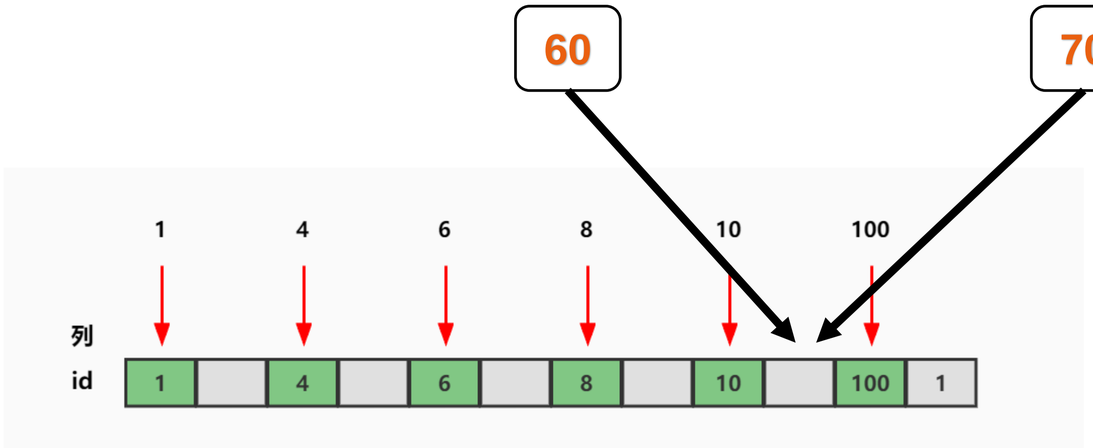
- 举个栗子：按照间隙锁的知识分析，此时间隙锁的范围是 (11,99)，意思是这个范围的 id 都不可以插入。如果是这样的话数据插入效率太低，锁范围比较大，很容易发生锁冲突怎么办？
- **插入意向锁就是用来解决这个问题的！**



5.6 MySQL-行锁【重点】

03-行锁四兄弟：记录锁、间隙锁、临键锁、插入意向锁

- 插入意向锁（Insert Intention Locks）是一种在 INSERT 操作之前设置的一种特殊的间隙锁。
- 插入意向锁表示了一种插入意图，即当多个不同的事务，同时往同一个索引的同一个间隙中插入数据的时候，它们互相之间无需等待，即不会阻塞。
- 插入意向锁不会阻止插入意向锁，但是插入意向锁会阻止其他**间隙写锁（排他锁）、记录锁**。



- **举个栗子：**现在有两个事务，分别尝试插入值为 60 和 70 的记录，每个事务使用插入意向锁锁定 11 和 99 之间的间隙，但是这两个事务不会相互阻塞，因为行是不冲突的！这就是插入意向锁。

5.6 MySQL-行锁【重点】

04-加锁规则

- **主键索引：**
  - 等值条件，命中加记录锁
  - 等值条件，未命中加间隙锁
  - 范围条件，命中包含where条件的临键区间加临键锁
  - 范围条件，没有命中加间隙锁
- **辅助索引：**
  - 等值条件，命中，命中记录辅助索引项，回表主键索引项加记录锁，辅助索引项两侧加间隙锁
  - 等值条件，未命中加间隙锁
  - 范围条件，命中包含where条件的临键区间加临键锁。命中记录回表主键索引项加记录锁
  - 范围条件，没有命中加间隙锁

案例：一条Update语句的执行流程

5.6 MySQL-行锁【重点】

05-存储引擎级别的表锁：意向锁

- InnoDB也实现了类似表级锁的锁，叫做意向锁（Intention Locks），意向锁是InnoDB自动控制不需要手动干预，意向锁和行锁是共存的。主要目标是为了全表更新数据时提升性能
- 作用：意向锁的存在是为了协调行锁和表锁的关系，支持行锁和表锁的共存
- 意向锁和表级S锁、X锁的兼容关系

注意：行级写锁不会因为有别的事务上了意向写锁而堵塞，MySQL允许不同行的多个行级写锁同时存在

当事务A上了如下锁					
事务B能否上		IS	IX	S	X
	IS	是	是	是	否
	IX	是	是	否	否
	S	是	否	是	否
	X	否	否	否	否

5.6 MySQL-行锁【重点】

06-锁相关参数

InnoDB所使用的**行级锁定**争用状态查看：show status like 'innodb_row_lock%';

如何查看事务、锁?

```
mysql> show status like 'innodb_row_lock%';
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| Innodb_row_lock_current_waits | 0     |
| Innodb_row_lock_time          | 0     |
| Innodb_row_lock_time_avg      | 0     |
| Innodb_row_lock_time_max      | 0     |
| Innodb_row_lock_waits         | 0     |
+-------------------------------+-------+
5 rows in set (0.00 sec)
```

当前正在等待锁定的数量

从系统启动到现在锁定总时间长度

每次等待所花平均时间

从系统启动到现在等待最常的一次所花的时间

系统启动后到现在总共等待的次数

5.7 案例：行锁分析实战

下面两条简单的SQL，他们加的什么锁？

- SQL1：**不加锁**
- SQL2：**加写锁**

```sql
-- SQL1:
select * from t1 where id = 10;
-- SQL2:
delete from t1 where id = 10;
```

前提一：id是不是主键？

前提二：隔离级别是什么？

前提三：如果id不是主键，那id有索引吗？

前提四：如果id有索引，那是唯一性索引吗？

前提五：两个SQL的具体执行计划是什么？走索引 还是 全表扫描

读已提交【RC】隔离级别

- 组合一：id列是主键，
- 组合二：id列是二级唯一索引
- 组合三：id列是二级非唯一索引
- 组合四：id列上没有索引

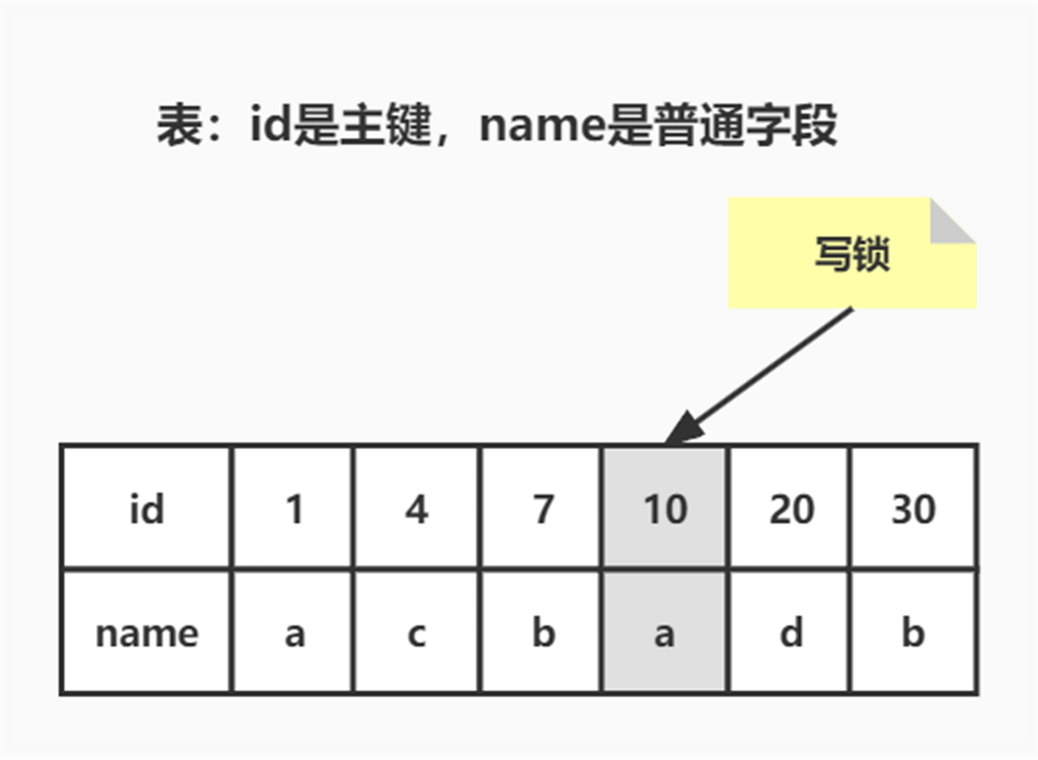
可重复读【RR】隔离级别

- 组合五：id列是主键
- 组合六：id列是二级唯一索引
- 组合七：id列是二级非唯一索引
- 组合八：id列上没有索引

5.7 案例：行锁分析实战

组合01-id是主键

- 在前面八种组合下，不论隔离级别，SQL1均不加锁，主要讨论SQL2加锁情况
- 在RC隔离级别下，给定记录加记录锁（**写锁**）




```
3  -- SQL2:
4  delete from t1 where id = 10;
```

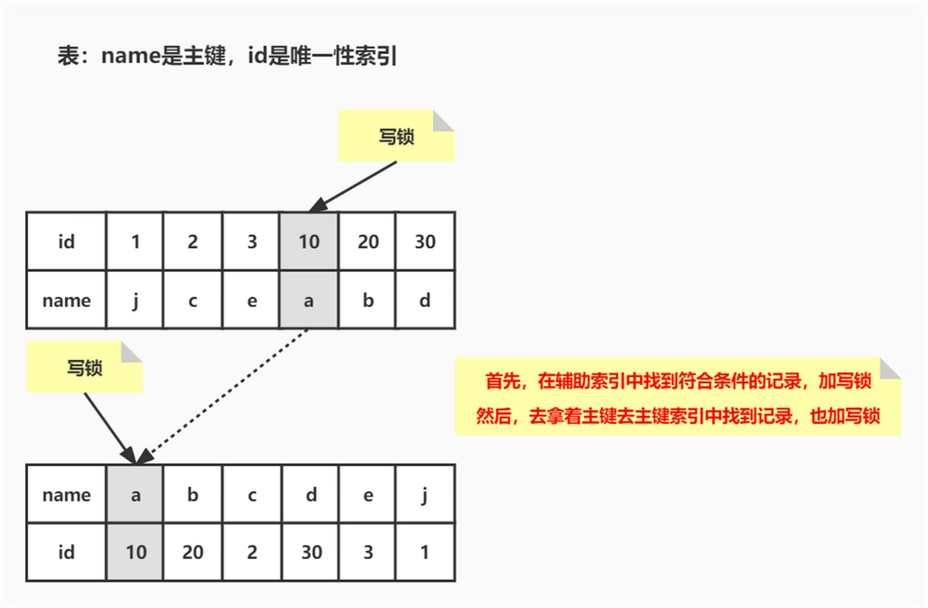
读已提交【RC】隔离级别

- 组合一：id列是主键，
- 组合二：id列是二级唯一索引
- 组合三：id列是二级非唯一索引
- 组合四：id列上没有索引

5.7 案例：行锁分析实战

组合02-id不是主键，是唯一索引

- 在RC隔离级别下，先在辅助索引树上将id=10加记录锁，然后找到id对应name='a'的主键索引项加记录锁




```
3  -- SQL2:
4  delete from t1 where id = 10;
```

读已提交【RC】隔离级别

- 组合一：id列是主键，
- 组合二：id列是二级唯一索引
- 组合三：id列是二级非唯一索引
- 组合四：id列上没有索引

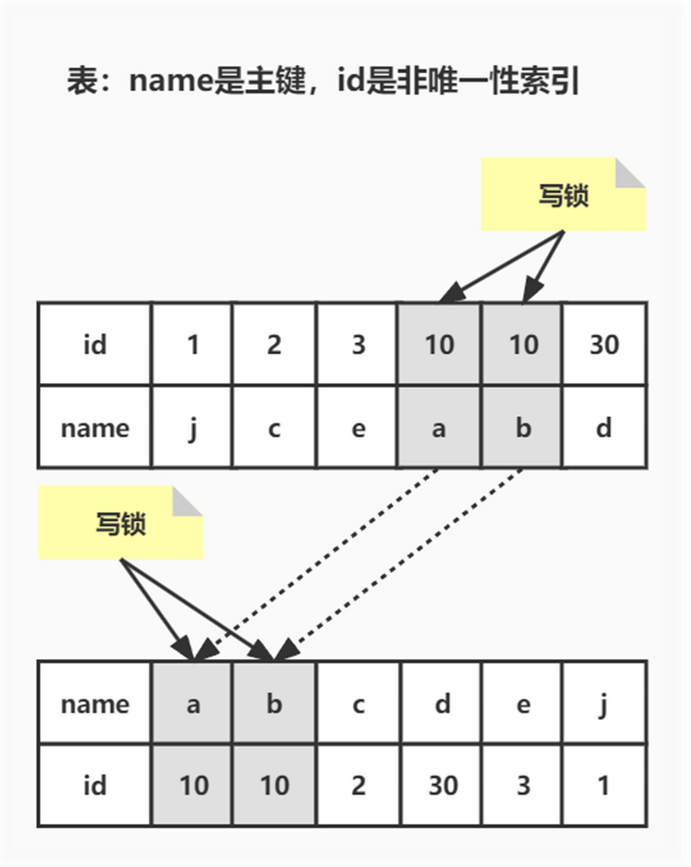
为什么聚簇索引上的记录也要加锁？

update t1 set id = 100 where name = 'a';

5.7 案例：行锁分析实战

组合03-id不是主键，是非唯一索引

- 在RC隔离级别下，在辅助索引树上，满足id=10的记录均加记录锁，然后找到id对应的主键记录在聚簇索引上都加记录锁



RC隔离级别，只加记录锁会有什么问题？

```
3  -- SQL2:
4  delete from t1 where id = 10;
```

读已提交【RC】隔离级别

- 组合一：id列是主键，
- 组合二：id列是二级唯一索引
- 组合三：id列是二级非唯一索引
- 组合四：id列上没有索引

5.7 案例：行锁分析实战

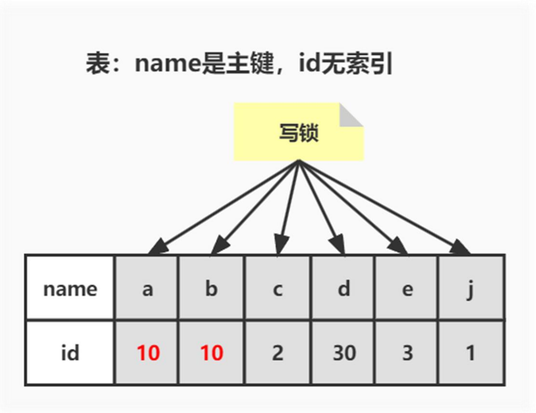
组合04-id无索引

- 在RC隔离级别下，聚簇索引name上的所有记录都加记录写锁，无论id条件是否满足

为什么不是只在满足条件的记录上加锁呢？

- id列它没索引，所以无法通过索引进行快速过滤，**只能通过聚簇索引来进行全表扫描**。为了保证在扫描的过程中数据不会被动，存储引擎将所有记录的聚簇索引加锁，然后**返回给SQL-Layer进行过滤**
- 当然这么做效率不佳，所以MySQL做了优化，对于不满足的记录会释放锁，最终只会持有满足条件的记录上的锁，但**加锁和释放锁**的动作无法省略

```
3  -- SQL2:
4  delete from t1 where id = 10;
```



读已提交【RC】隔离级别

- 组合一：id列是主键，
- 组合二：id列是二级唯一索引
- 组合三：id列是二级非唯一索引
- 组合四：id列上没有索引

5.7 案例：行锁分析实战

组合05-id是主键

- 与组合一是一致的
- 在RR隔离级别下，给定记录加记录锁（**写锁**）

组合06-id不是主键，是唯一索引

- 与组合二是一致的
- 在RR隔离级别下，先在辅助索引树上讲id=10加记录锁
- 然后找到id对应name='b'的主键索引项加记录锁

```
3  -- SQL2:
4  delete from t1 where id = 10;
```

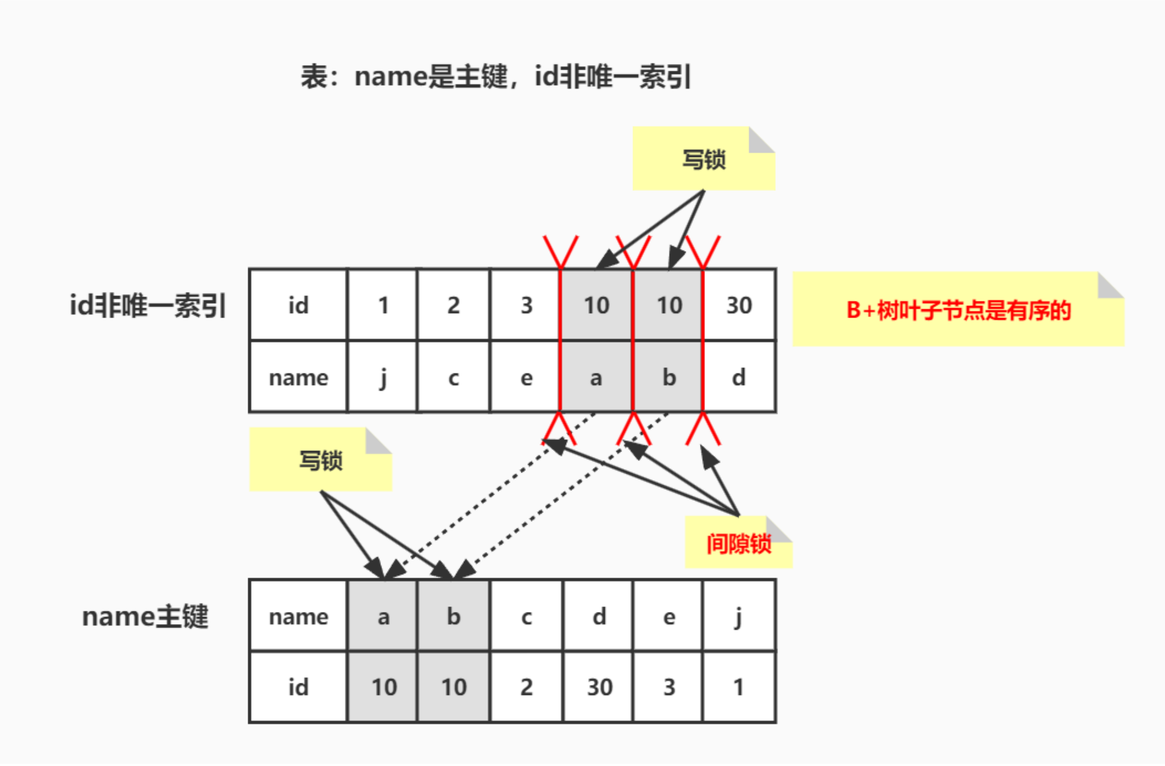
可重复读【RR】隔离级别

- 组合五：id列是主键
- 组合六：id列是二级唯一索引
- 组合七：id列是二级非唯一索引
- 组合八：id列上没有索引

5.7 案例：行锁分析实战

组合07-id不是主键，是非唯一索引

- 在RR隔离级别下，先通过辅助索引id定位到满足条件的索引项加上记录锁，然后在索引项的GAP上加间隙锁。对于辅助索引关联的聚簇索引上的记录加记录锁




```
3  -- SQL2:
4  delete from t1 where id = 10;
```

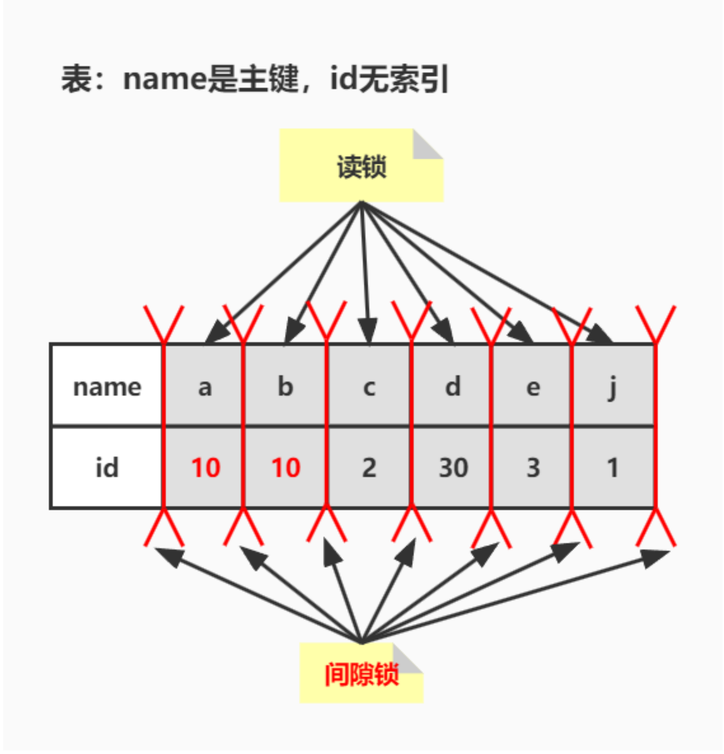
可重复读【RR】隔离级别

- 组合五：id列是主键
- 组合六：id列是二级唯一索引
- 组合七：id列是二级非唯一索引
- 组合八：id列上没有索引

5.7 案例：行锁分析实战

组合08-id无索引

- 在RR隔离级别下，如果id无索引删除，会进行全表扫描的当前读，然后锁上表中所有记录，同时会锁上聚簇索引的所有间隙，防止幻读。
- 杜绝了所有并发update/delete/insert操作
- 与**组合04**类似，MySQL也对这种情况作了优化，对于不满足条件的记录会提前释放锁




```
3  -- SQL2:
4  delete from t1 where id = 10;
```

可重复读【RR】隔离级别

- 组合五：id列是主键
- 组合六：id列是二级唯一索引
- 组合七：id列是二级非唯一索引
- 组合八：id列上没有索引

5.7 案例：行锁分析实战

组合09-Serializable

- SQL2，Serializable和RR隔离级别与**组合08**情况是一致的
- 在Serializable隔离级别下，SQL1会加读锁

```
1 -- SQL1:
2 select * from t1 where id = 10;
3 -- SQL2:
4 delete from t1 where id = 10;
```

SQL1为什么会加锁读锁？说好的读不加锁，读写不冲突呢？

- MVCC只在RR和RC隔离级别下生效
- Serializable隔离级别下，MVCC会降级为LBCC，基于锁进行并发控制

SQL太简单了？来，咱们看一个复杂的SQL

5.8 案例：复杂SQL加锁分析

再来看一个稍微复杂点的SQL：idx_t1_pu（pubtime，userid）

```
1 delete from t1 where pubtime > 1 and pubtime < 20 and userid='hero' and commit is not null;
```

查询条件构成拆分：

- **Index key：** pubtime > 1 and puptime < 20，使用组合索引范围查找
- **Index Filter：** userid = ‘hero’，会在索引上用来进行过滤
- **Table Filter：** comment is not NULL，不在组合索引中，只能在SQL-Layer上过滤

id是主键，其他列userid，blogid，pubtime，comment

id	1	4	6	8	10	100
userid	hero	yyy	hero	hero	hero	bbb
blogid	a	b	c	d	e	f
pubtime	10	3	100	5	1	20
comment				handsome		

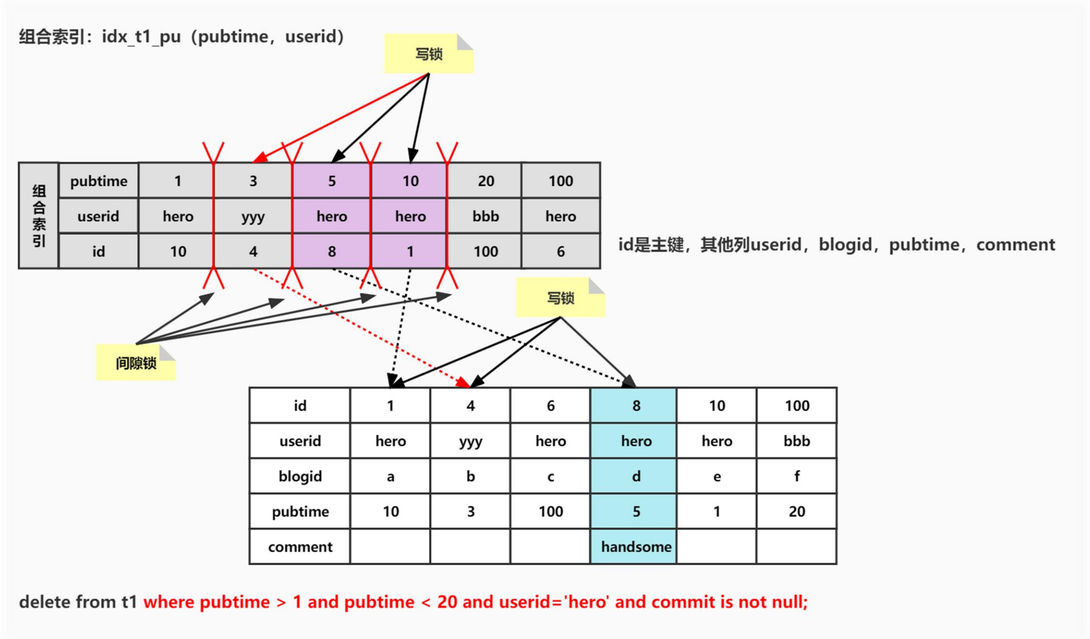
5.8 案例：复杂SQL加锁分析

再来看一个稍微复杂点的SQL：idx_t1_pu（pubtime，userid）

```
1 delete from t1 where pubtime > 1 and pubtime < 20 and userid='hero' and commit is not null;
```

加锁情况：

- 在RR隔离级别下，由Index Key所确定的范围，被加上了间隙锁
- Index Filter条件确定的记录加锁情况，会依据ICP来确定，支持ICP则无需加写锁，不支持要加写锁
- Table Filter过滤条件，则需要在聚簇索引上加写锁

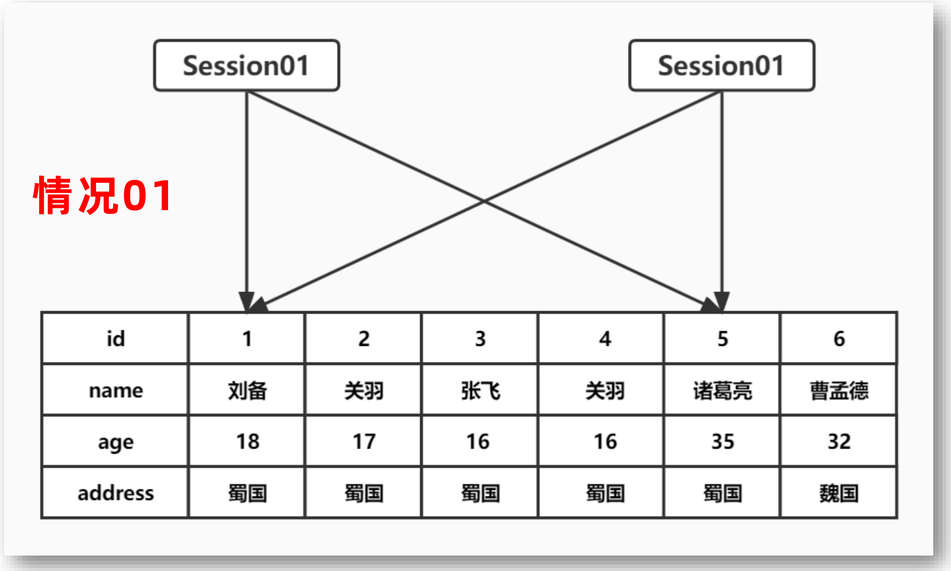


5.9 MySQL死锁

01-什么是死锁？

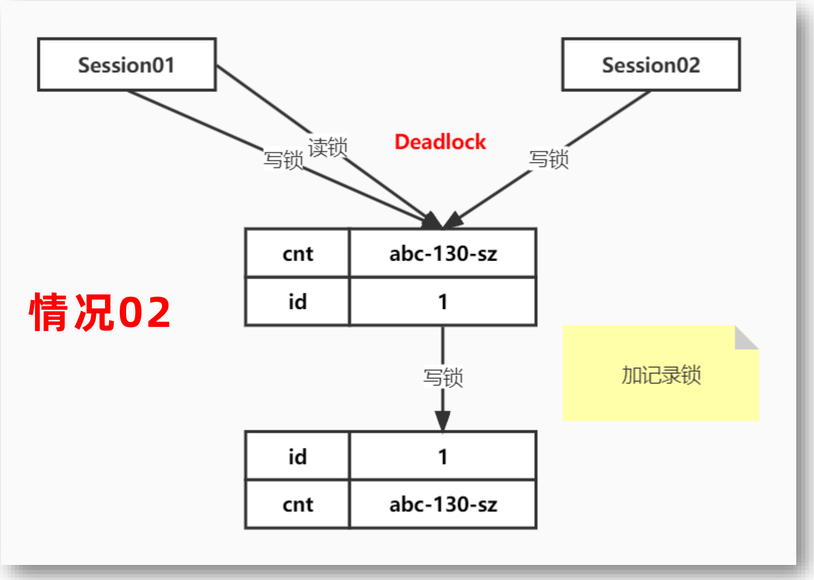
02-为什么学习死锁？

- 写出不会发生死锁的SQL
- 快速定位出线上产生死锁的原因
- 透过现象看本质：理解数据库层面阻塞执行的根本原因



03-如何避免死锁呢？

- **注意程序逻辑：**根本原因是程序逻辑的顺序交叠
- **保持事务轻量：**越是轻量的事务，占有越少的锁资源，这样发生死锁的几率就越小
- **提高运行速度：**避免使用子查询，尽量使用主键等等
- **尽量快提交事务，减少持有锁的时间：**越早提交事务，锁就越早释放



注意：MySQL会主动探知死锁，并回滚某一个影响最小的事务，等另一事务执行完后，再重新执行该事务！

整个案例：强人锁男

THANKS

教育不是注满一桶水，而是点燃一把火