# 总结

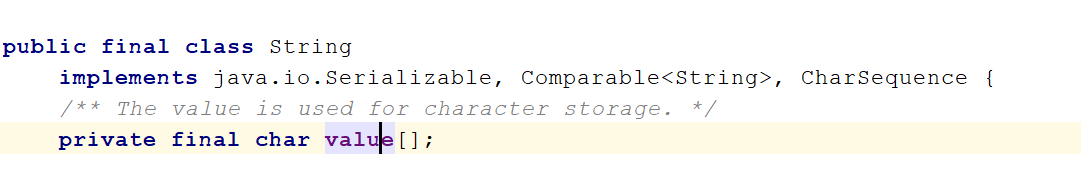
## Java基础：

### 面向过程(c)相对性能高于面向对象(java)的原因：

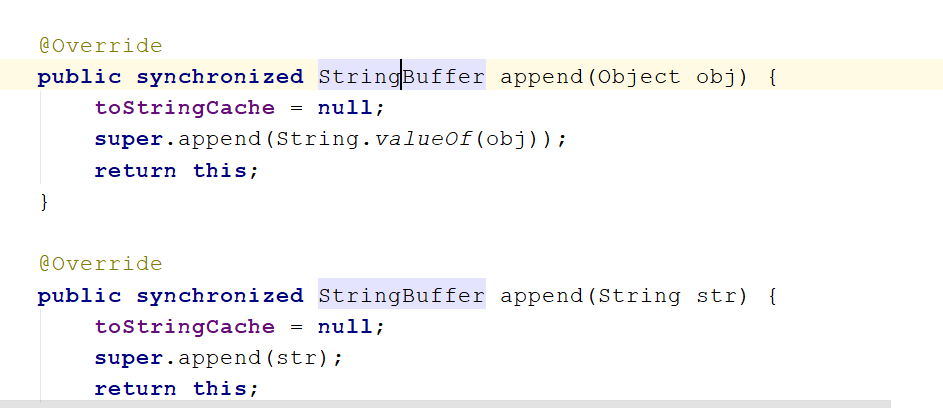
* 1. 面向过程经过一次编译为机器运行的机器码，而面向对象得先编译为字节码(.class)，然后java虚拟机转换为机器码运行。
  2. 面向过程：性能相对较好；
  3. 面向对象：易维护、易复用、易扩展。

### String、StringBuilder、StringBuffer区别：

1. String中value被加了final,具备不可变性，每一次操作都是一个新的字符串:



1. StringBuilder线程不安全，未被final修饰，每次都是操作的同一个字符串。
2. StringBuffer线程安全，被**synchronized** 修饰；未被final修饰，每次都是操作的同一个字符串。
3. StringBuilder和StringBuffer继承公共抽象父类AbstractStringBuilder

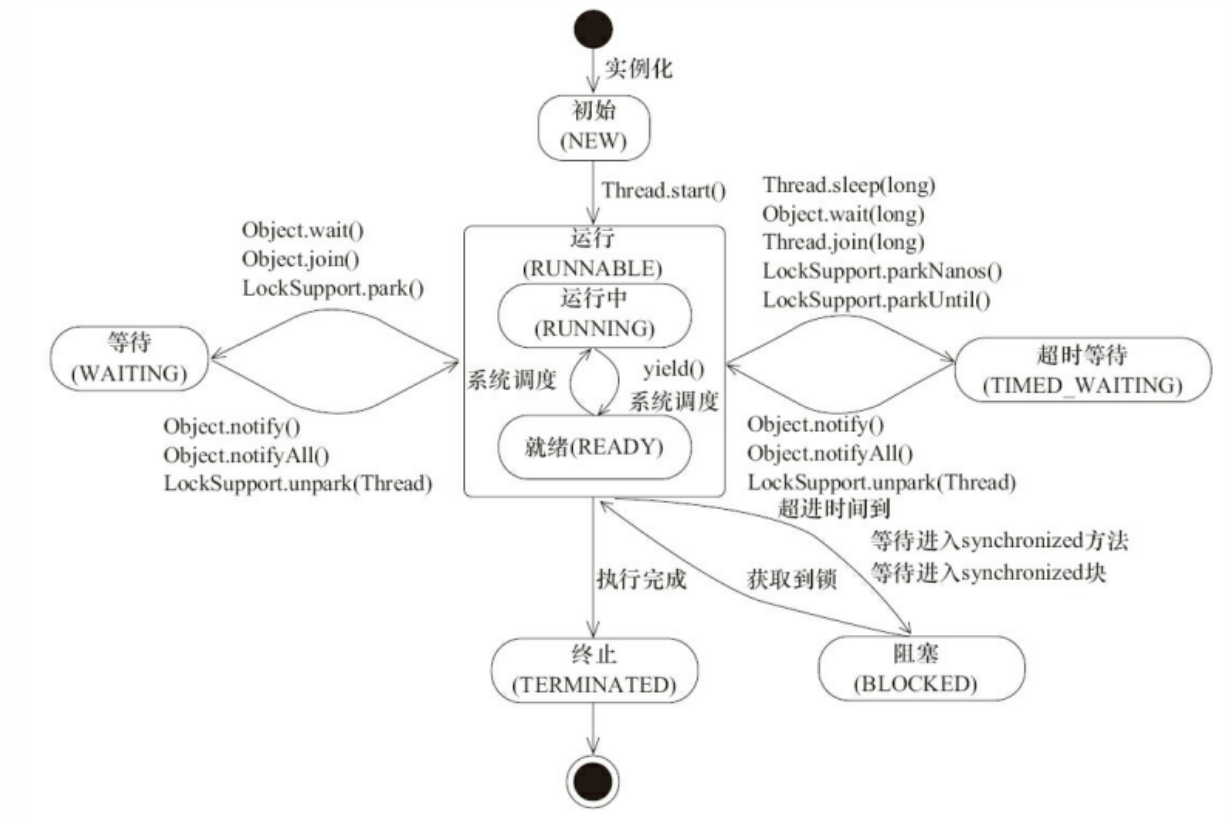


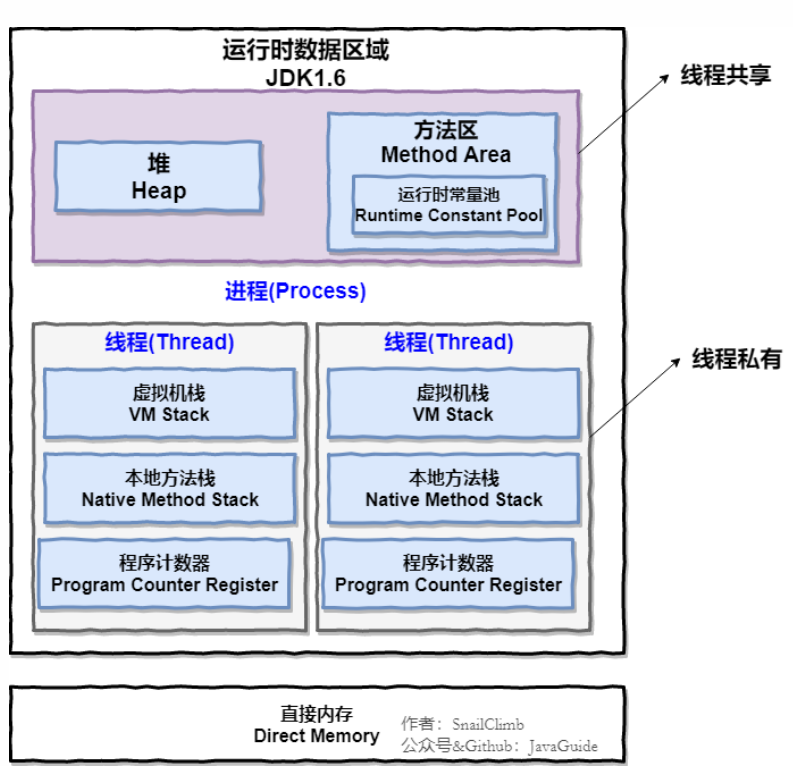
StringBuilder比StringBuffer性能高10%-15%，String适合操作少量数据；单线程操作大量数据使用StringBuilder，多线程操作大量数使用StringBuffer。

### hashCode作用：

1. 当把对象放入hashSet时，会先判断对象放入的位置，及hashCode值；
2. 获取哈希码(散列码)，通过哈希码确定对象在哈希表中的索引位置。
3. hashCode相同，两个对象不一定相等；相同的hashCode可能有多个对象，hashCode相当于缩小搜索范围。

### 线程的生命周期：

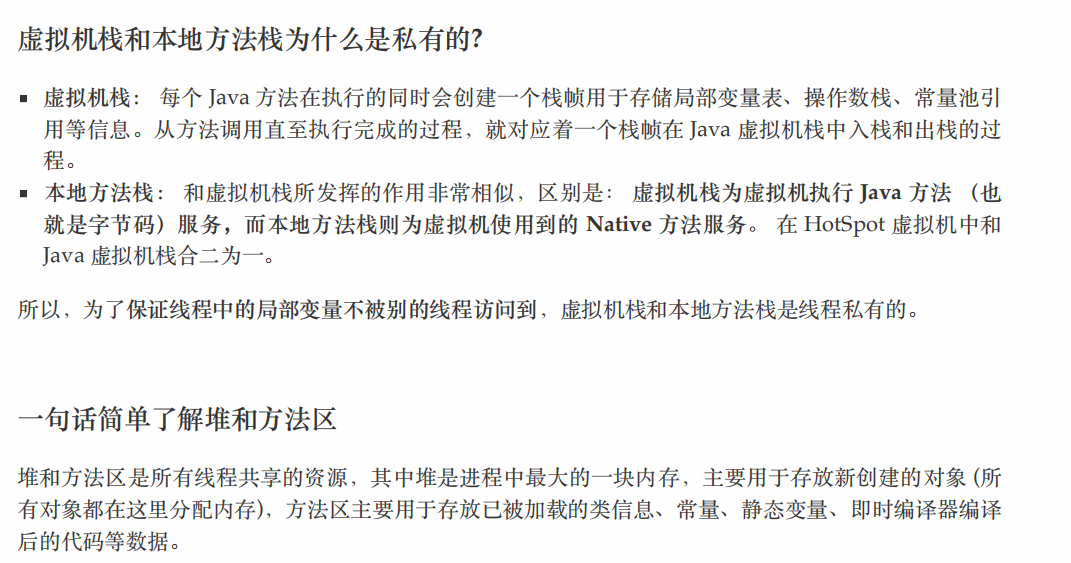




### 程序计数器为什么是线程私有的？

1. 字节码解释器通过改变程序计数器来依次读取指令，从而实现代码的流程控制；
2. 在多线程情况下，程序计数器记录当前线程执行的位置，从而当线程被切换回来时知道该线程上次被执行的位置。

总结：所以程序计数器私有是为了线程切换后能恢复到正确的执行位置。



6、

问sychonried和lock，，，就说不清楚，，没了解过，，问动态代理实现方式，，问不清楚，，3年经验，，，这个都不知道，，直接pass了

Sychonried，，是java关键字，是轻量级所，，偏向锁，，也可以锁膨胀变成重量级，可重入所，不用自己释放锁，，锁对象可以是任何，，，只有wait，异常，，或者完成，，才会释放锁，，容易造成死所

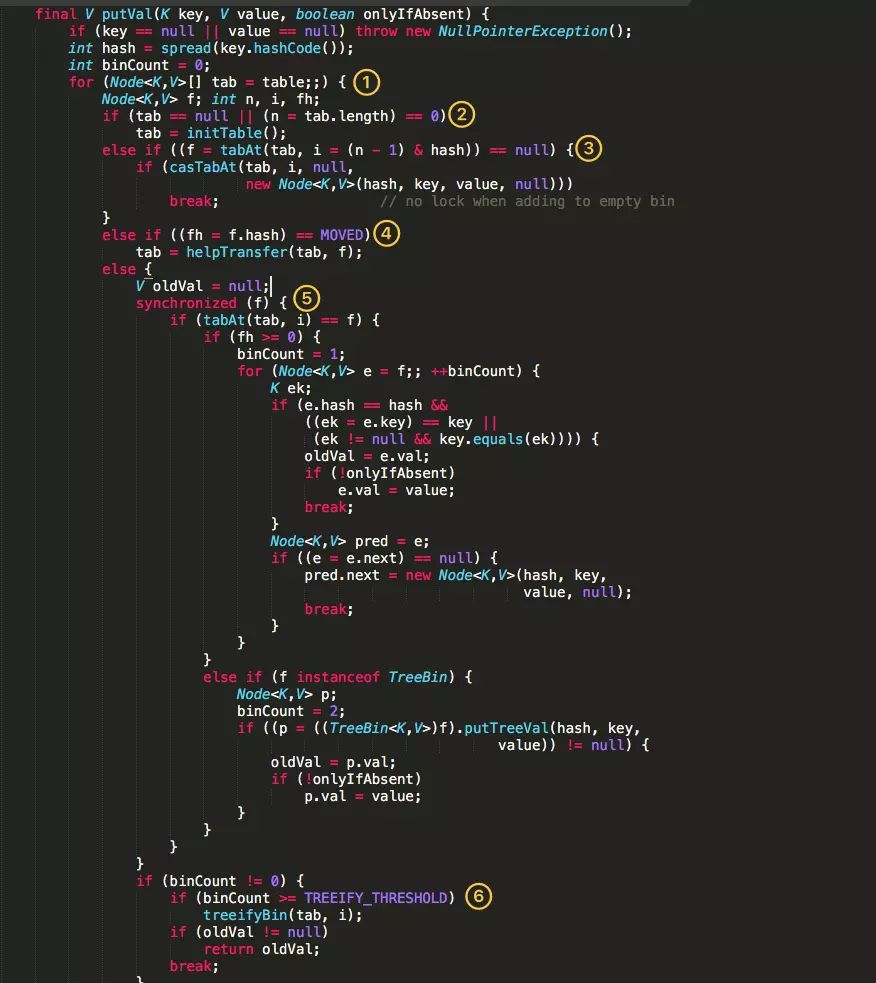
### jdk1.8新特性

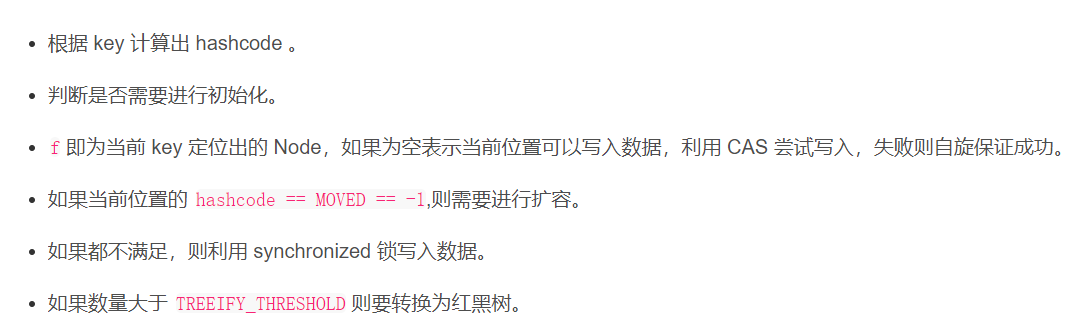
* **Lambda表达式**
* **函数式接口**
* **\*方法引用和构造器调用**
* **Stream API**
* **接口中的默认方法和静态方法**
* **新时间日期API**

详情地址：<https://blog.csdn.net/qq_29411737/article/details/80835658>

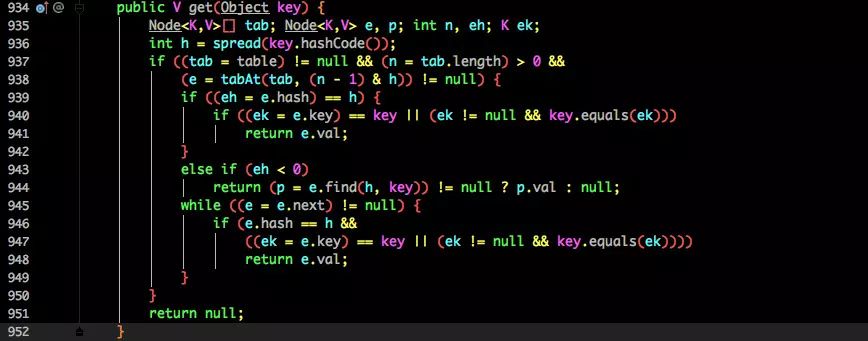
### ConcurrentHashMap原理

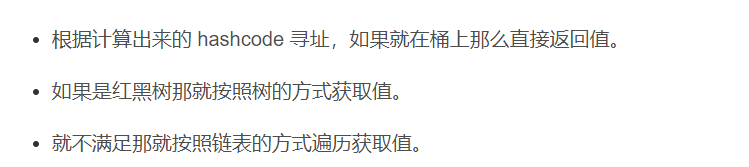
Put方法源码理解





get方法源码理解：



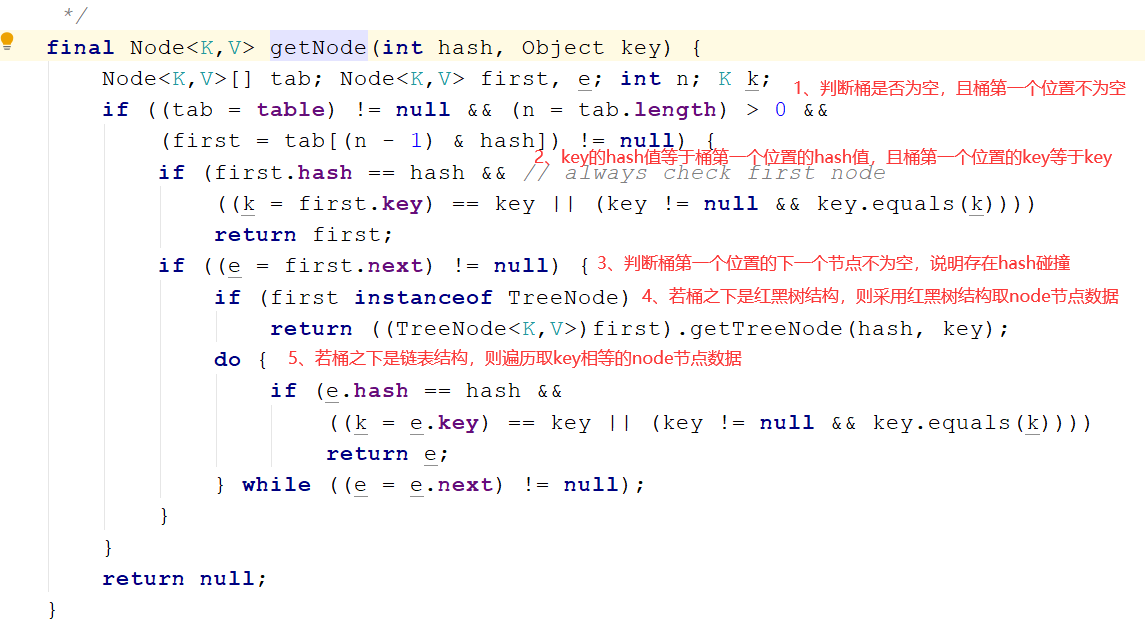


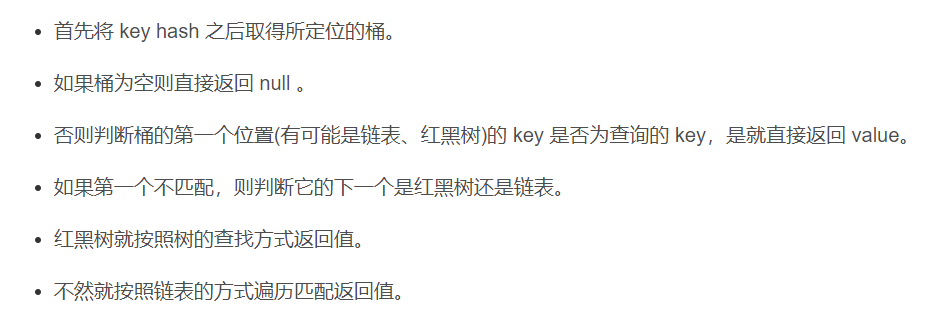
参考：<https://blog.csdn.net/tp7309/article/details/76532366>

https://blog.csdn.net/weixin\_44460333/article/details/86770169

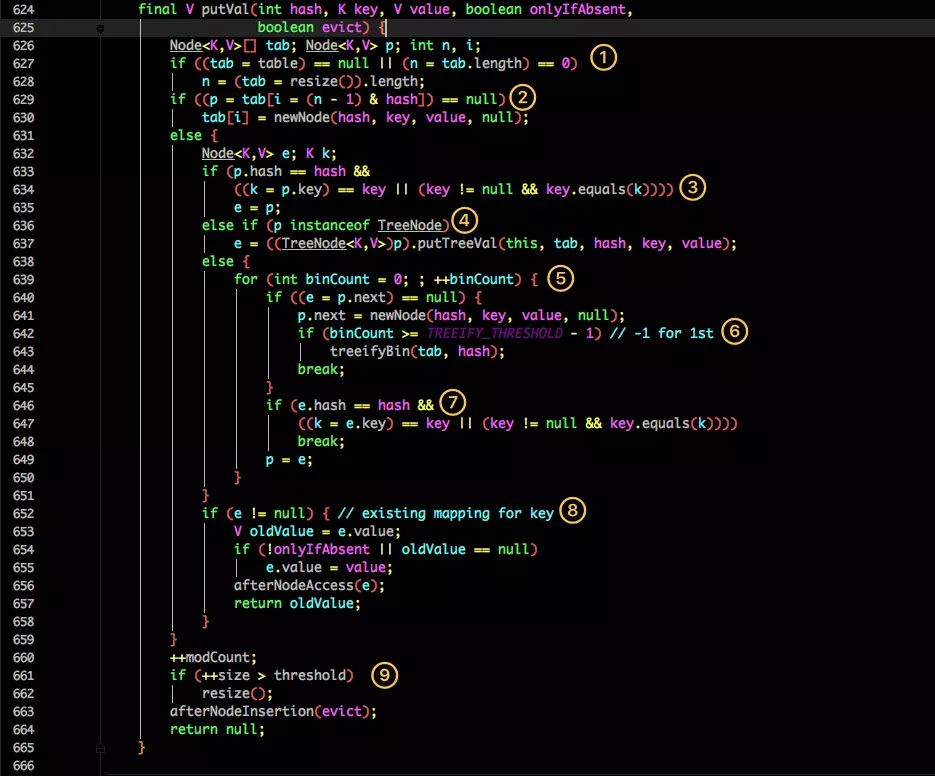
### HashMap原理

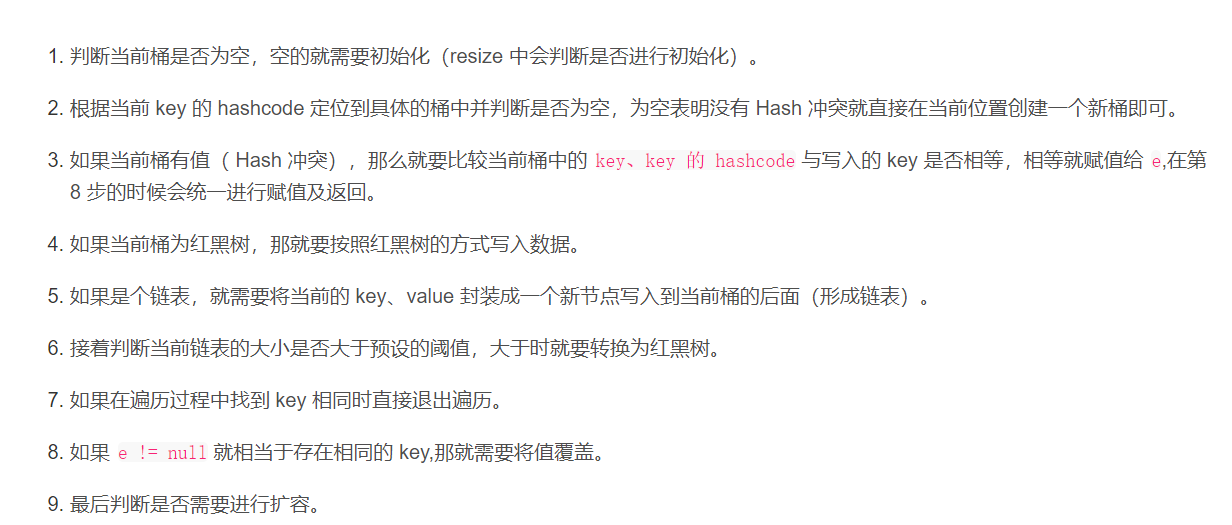
get方法源码理解：





put方法源码理解：





面试题：

#### 1、“你用过HashMap吗？” “什么是HashMap？你为什么用到它？”

答：用过；

hashMap特性：1.HashMap存储键值对实现快速存取，允许为null。key值不可重复，若key值重复则覆盖。2.非同步，线程不安全；3.底层是hash表，不保证有序(比如插入的顺序)；4、底层采用数组及链表结构或者数组及红黑树结构(当联表数量大于等于8时，自动转为红黑树)。

若需要hashMap线程安全可采用以下两个方法：1、Collections.synchronizedMap(map);2、ConcurrentHashMap。

#### 谈一谈hashMap底层原理

答：基于hashing的原理，jdk8后采用数组+链表+红黑树的数据结构。我们通过put和get存储和获取对象。当我们给put()方法传递键和值时，先对键做一个hashCode()的计算来得到它在bucket数组中的位置来存储Entry对象。当获取对象时，通过get获取到bucket的位置，再通过键对象的equals()方法找到正确的键值对，然后在返回值对象。

#### 谈一谈hashMap中put方法的实现

答：1.计算关于key的hashcode值（与Key.hashCode的高16位做异或运算）

2.如果散列表为空时，调用resize()初始化散列表

3.如果没有发生碰撞，直接添加元素到散列表中去

4.如果发生了碰撞(hashCode值相同)，进行三种判断

    4.1:若key地址相同或者equals后内容相同，则替换旧值

    4.2:如果是红黑树结构，就调用树的插入方法

   4.3：链表结构，循环遍历直到链表中某个节点为空，尾插法进行插入，插入之后判断链表个数是否到达变成红黑树的阙值8；也可以遍历到有节点与插入元素的哈希值和内容相同，进行覆盖。

5.如果桶满了大于阀值，则resize进行扩容

#### 4.谈一下hashMap中什么时候需要进行扩容，扩容resize()又是如何实现的？

调用场景：

1.初始化数组table

2.当数组table的size达到阙值时即++size > load factor \* capacity 时，也是在putVal函数中

实现过程：(细讲)

1.通过判断旧数组的容量是否大于0来判断数组是否初始化过

否：进行初始化判断是否调用无参构造器，

是:使用默认的大小和阙值

否:使用构造函数中初始化的容量，当然这个容量是经过tableSizefor计算后的2的次幂数

是，进行扩容，扩容成两倍(小于最大值的情况下)，之后在进行将元素重新进行与运算复制到新的散列表中

概括的讲：扩容需要重新分配一个新数组，新数组是老数组的2倍长，然后遍历整个老结构，把所有的元素挨个重新hash分配到新结构中去。

PS：可见底层数据结构用到了数组，到最后会因为容量问题都需要进行扩容操作

#### 5.谈一下hashMap中get是如何实现的？

对key的hashCode进行hashing，与运算计算下标获取bucket位置，如果在桶的首位上就可以找到就直接返回，否则在树中找或者链表中遍历找，如果有hash冲突，则利用equals方法去遍历链表查找节点。

#### 6.谈一下HashMap中hash函数是怎么实现的？还有哪些hash函数的实现方式？

对key的hashCode做hash操作，与高16位做异或运算

还有平方取中法，除留余数法，伪随机数法

#### 7.为什么不直接将key作为哈希值而是与高16位做异或运算？

因为数组位置的确定用的是与运算，仅仅最后四位有效，设计者将key的哈希值与高16为做异或运算使得在做&运算确定数组的插入位置时，此时的低位实际是高位与低位的结合，增加了随机性，减少了哈希碰撞的次数。

HashMap默认初始化长度为16，并且每次自动扩展或者是手动初始化容量时，必须是2的幂。

#### 8、HashMap和HashTable的区别

相同点：都是存储key-value键值对的

不同点：

HashMap允许Key-value为null，hashTable不允许；

hashMap没有考虑同步，是线程不安全的。hashTable是线程安全的，给api套上了一层synchronized修饰;

HashMap继承于AbstractMap类，hashTable继承与Dictionary类。

迭代器(Iterator)。HashMap的迭代器(Iterator)是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。所以当有其它线程改变了HashMap的结构（增加或者移除元素），将会抛出ConcurrentModificationException。

容量的初始值和增加方式都不一样：HashMap默认的容量大小是16；增加容量时，每次将容量变为"原始容量x2"。Hashtable默认的容量大小是11；增加容量时，每次将容量变为"原始容量x2 + 1"；

添加key-value时的hash值算法不同：HashMap添加元素时，是使用自定义的哈希算法。Hashtable没有自定义哈希算法，而直接采用的key的hashCode()。

#### 9、平时在使用HashMap时一般使用什么类型的元素作为Key？

使用String或者Integer这样的类。这个时候可以继续追问为什么使用String、Integer呢？这些类有什么特点？如果面试者有很好的思考，可以回答出这些类经final修饰是不可变（Immutable）的，并且这些类已经很规范的覆写了hashCode()以及equals()方法。作为不可变类天生是线程安全的，而且可以很好的优化比如可以缓存hash值，避免重复计算等等，那么基本上这道题算是过关了。

参考资料：<https://blog.csdn.net/weixin_44460333/article/details/86770169>

<https://blog.csdn.net/suifeng629/article/details/82179996>

<https://www.cnblogs.com/zengcongcong/p/11295349.html>

### IO流



nio:

它是一种同步非阻塞的I/O模型，也是I/O多路复用的基础，已经被越来越多地应用到大型应用服务器，成为解决高并发与大量连接、I/O处理问题的有效方式。

传统IO基于字节流和字符流进行操作，而NIO基于Channel和Buffer(缓冲区)进行操作，数据总是从通道读取到缓冲区中，或者从缓冲区写入到通道中。Selector(选择区)用于监听多个通道的事件（比如：连接打开，数据到达）。因此，单个线程可以监听多个数据通道。

NIO和IO适用场景

如果需要管理同时打开的成千上万个连接，这些连接每次只是发送少量的数据，例如聊天服务器，这时候用NIO处理数据可能是个很好的选择。

而如果只有少量的连接，而这些连接每次要发送大量的数据，这时候传统的IO更合适。使用哪种处理数据，需要在数据的响应等待时间和检查缓冲区数据的时间上作比较来权衡选择。

参考资料：<https://blog.csdn.net/zengxiantao1994/article/details/88094910>

<https://blog.csdn.net/u013857458/article/details/82424104>

<https://blog.csdn.net/u011381576/article/details/79876754>

https://www.cnblogs.com/dolphin0520/p/3919162.html

### 常见网络攻击

#### (1)分类

1、DDoS攻击

DDoS是最常见的一种攻击方式，攻击者通过向某一个站点服务器反复发送请求，导致服务器无法承载大量的请求而产生“拒绝服务”，这就导致正常的服务无法进行，影响我们服务器的使用。

2、获取账号和密码

这类攻击会存在一定的技术性。一般来说，攻击者会利用程序来抓取数据包，获取口令和数据内容，通过侦听程序在来监视网络数据流，进而通过分析获取用户的登录账号和密码。

3、SQL注入

SQL注入的攻击手段主要是利用后台的漏洞，通过URL将关键SQL语句带入程序，在数据库中进行破坏。许多的攻击者会使用F12或者postman等拼装ajax请求，将非法的数字发送给后台，造成程序的报错，并展现在页面上，这样攻击者就会知道后台使用的语言和框架了。

4、恶意小程序

这类攻击的方式主要存在我们使用的程序上面，它们可以通过入侵修改硬盘上的文件、窃取口令等。

5、木马植入

这种攻击方式主要是通过向服务器植入木马，开启后面，获取服务器的控制权，恶意破坏服务器文件或盗取服务器数据，这类的危害都是比较大的。

#### (2)sql注入

#{}是预编译处理，${}是字符替换（字符串拼接）。 在使用 #{}时，MyBatis 会将 SQL 中的 #{}替换成“?”，配合 PreparedStatement 的 set 方法赋值，这样可以有效的防止 SQL 注入，保证程序的运行安全。

#### (3)简述 tcp（传输控制协议） 和 udp的区别？

tcp 和 udp 是 OSI 模型中的运输层中的协议。tcp 提供可靠的通信传输，而 udp 则常被用于让广播和细节控制交给应用的通信传输。

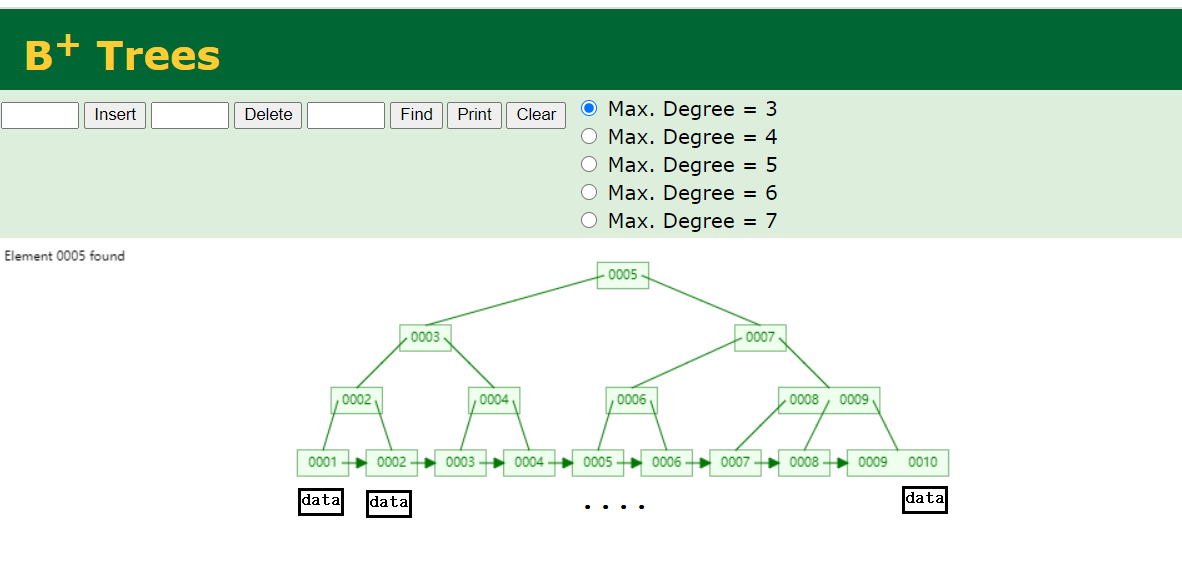
两者的区别大致如下：

* tcp 面向连接，udp 面向非连接即发送数据前不需要建立链接；
* tcp 提供可靠的服务（数据传输），udp 无法保证；
* tcp 面向字节流，udp 面向报文；
* tcp 数据传输慢，udp 数据传输快；
* tcp占用系统资源多，udp较少；

## Mysql

### mysql为什么用b+树作为数据结构。

注：InnoDB存储引擎中data存放的是这一行的真实数据；myisam存放的是真实数据的磁盘地址。



使用b+树的优点：非叶子节点只保存了索引的key值，不会保存索引的value值，树结构每层可以保存更多的索引，mysql定义的非叶子节点可以保存16kb的数据，以bigint为列：索引占据8个字节，两个索引之间指针占据6个字节，一层数可存放16kb/14b个索引，约为1143个索引，一般b+数三层可存储2000多万数据。并且，叶子节点为单向链表结构，比起b树而言，支持范围查找。

#### 为什么不用线性表？

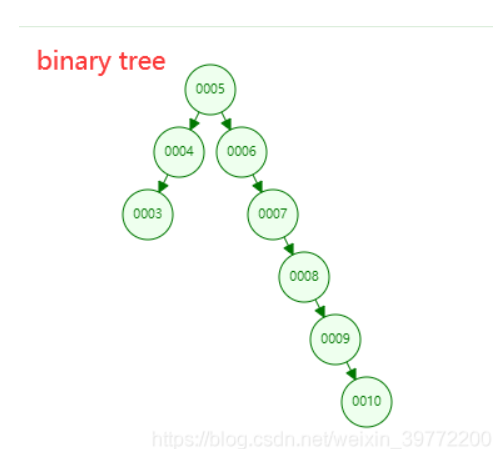
当查找的数据正好在线性表的最后一个索引时，则会进行全表扫描

#### （2）、为什么不用二分查找？

二分查找本身有排序，排序无论用什么算法都会消耗资源，尤其是查找时可能还会有insert、update、delete，这可能会导致数据的索引发生变化，极端情况查询开销也非常大

#### （3）、为什么不用二叉数查找？

二叉树：极端情况单分支变成线性表，查询开销也非常大。eg：右侧分支有几十万甚至上百万条数据，最坏的情况是数据在右侧分支的最后一个节点，此时查询效率和线性表无二。



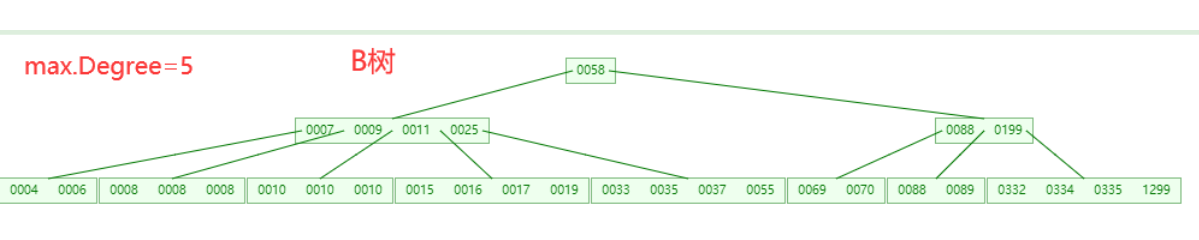
#### （4）为什么不用红黑树

使用红黑树，当数据越多，树的深度越深，查询耗时也越多，性能越差。

#### （5）、为什么不是哈希查找？

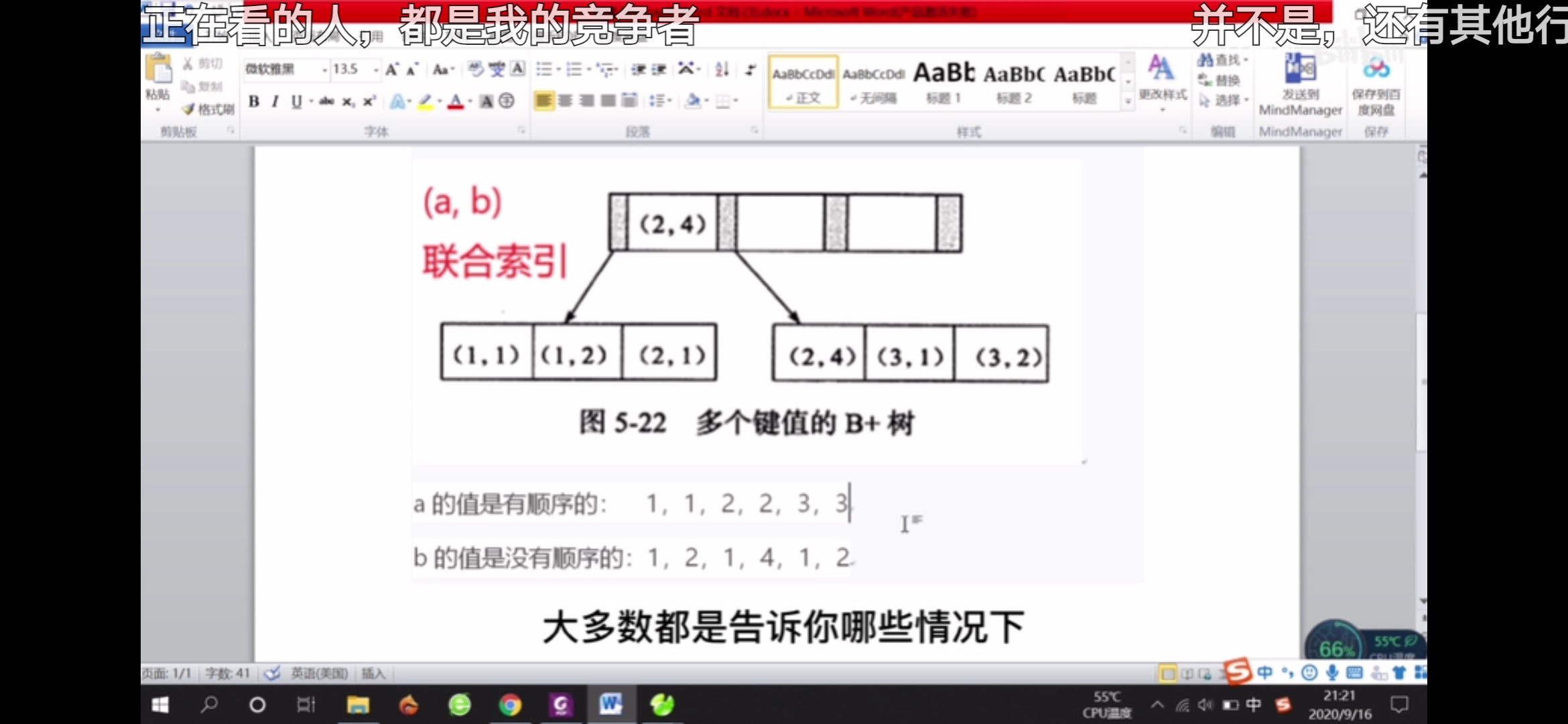
哈希算法虽然高，但也有它的局限性  
 1、 哈希索引只能操作：in、=，不支持范围查询，eg：>,<,!=  
原因：哈希索引比较的是经过哈希算法之后的哈希值，而哈希索引是无序的，用范围值的结果计算出来的哈希值并不一定和哈希索引中的排序是一样的。还有可能发生哈希碰撞。  
 2、 由于哈希索引本身是不排序的，所以任何时候都避免不了索引全表扫描。

#### （6）、为什么不是B树查找？



B树的优点：一个节点可以保存多个索引，通过度解决了红黑树的树深问题  
 缺点：由于非叶子节点也保存数据，导致一次IO读不到更多的索引

### mysql联合索引为什么会失效



如图：

关键原理：a索引是有序的，b索引是无序的。但在a索引相同的情况下，b索引是有序的

原因：遵循最佳左前缀原则。

注意：最佳左前缀法则在全是and=的时候不适用，mysql会自动调整为创建索引的顺序。

例：

EXPLAIN SELECT \* FROM contract\_management WHERE apply\_person ='唐超' AND STATUS=2

EXPLAIN SELECT \* FROM contract\_management WHERE STATUS=2 AND apply\_person ='唐超'

这两者效果一样，但是建议按照索引的顺序来书写

例1：select name,age from user where name=? and age=?;

联合索引name,和age,b+树是先根据第一个索引去查询到满足的数据索引位置，再根据第二个索引去查询最终满足的数据。

例2：select name,age from user where name=? and age>?and gener=?;

此联合索引范围查询之后的索引不会使用，第三个索引变得无序。当范围查找后，第三个索引变得无序，只能进行全表扫描。

例3：select name,age from user where like name “%name%”;

当左边%存在，不能使用索引，当左边%不存在，可以用到索引。



### mysql的索引原理和数据结构功能

Mysql采用B+树存储索引，索引的key存放于树的非叶子节点，数据保存于叶子节点data域中，当执行查询时，通过二分法查找索引key，查询出对应的叶子节点数据。

### B树和B+树的区别

区别：

1. B树每个节点都存放了索引数据和当前所有对应的表数据，而B+树每个非叶子节点只存储索引的key,而在叶子节点存放表数据；
2. B树由于每个节点既存索引数据又存表数据，导致每个节点所存索引更少，数的深度更深，查询时相对B+树更慢。
3. B树不支持范围查询，而B+树叶子节点为单向链表结构，支持范围查询。

### 5、mysql聚簇索引和非聚簇索引的区别

存储引擎MyISAM使用非聚簇索引，InolDB主键索引使用聚簇索引。

聚簇索引就是索引和数据存放一个文件，如InnoDB使用的是聚簇索引，将主键组织到一棵B+树中，而行数据就储存在叶子节点data域上，若使用"where id = 14"这样的条件查找主键，则按照B+ 树的检索算法即可查找到对应的叶节点，之后获得行数据。

非聚簇索引索引和数据文件分别在不同的表中，如MyISAM使用非聚簇索引，在B+树中，每个叶子节点的data域都存放数据磁盘位置的指针，指向另一个文件的具体文件位置。

### 6、使用mysql索引都有什么原则

1、 对于查询频率高的字段创建索引；

2、 对排序、分组、联合查询频率高的字段创建索引；

3、 索引的数目不宜太多 原因：

a、每创建一个索引都会占用相应的物理控件；

b、过多的索引会导致insert、update、delete语句的执行效率降低；

4、选择唯一性索引 唯一性索引的值是唯一的，可以更快速的通过该索引来确定某条记录。

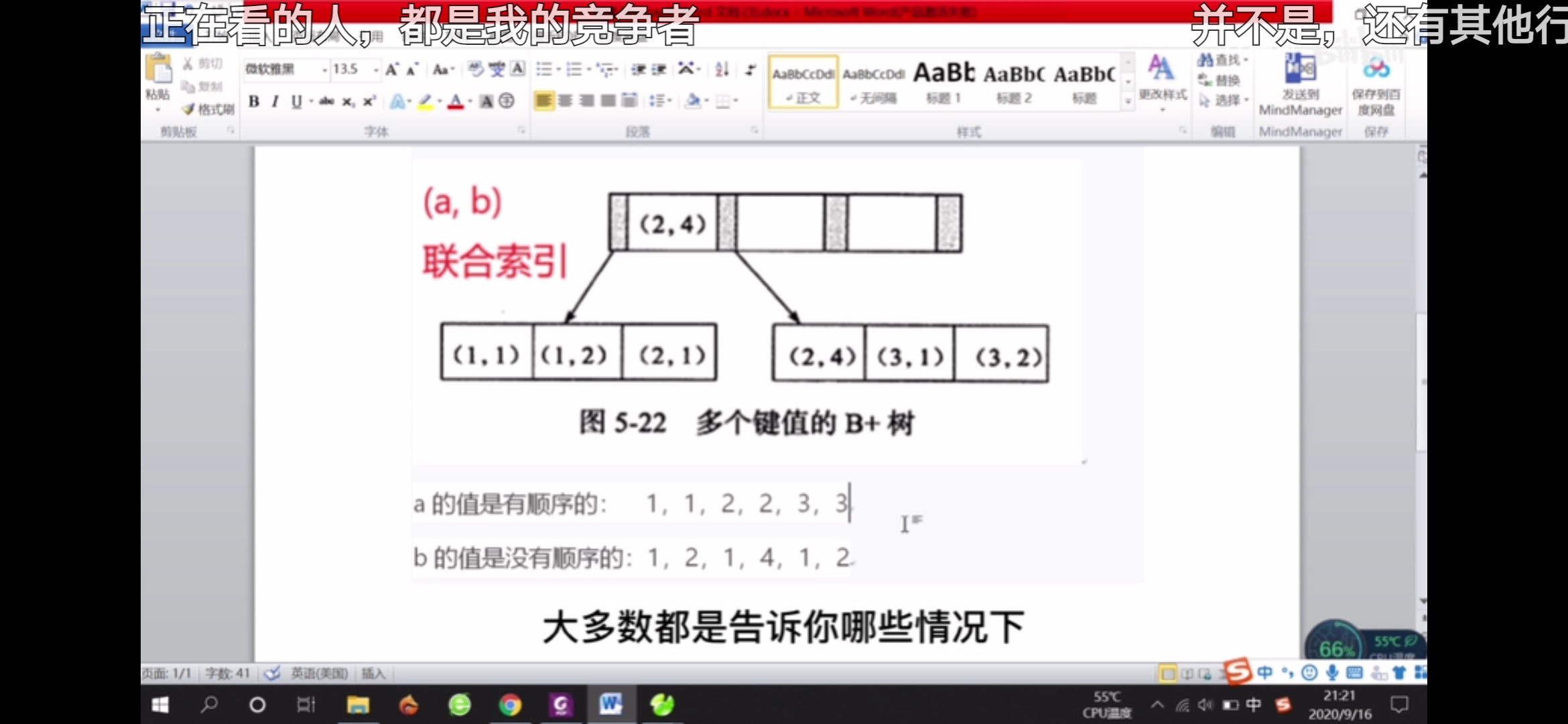
5、尽量使用数据量少的索引 如果索引的值很长，那么查询的速度会受到影响。

6、尽量使用前缀来索引 如果索引字段的值很长，最好使用值的前缀来索引。

7、删除不再使用或者很少使用的索引. 表中的数据被大量更新，或者数据的使用方式被改变后，原有的一些索引可能不再需要。

### 7、不同的存储引擎是如何进行实际存储的

### 8、mysql的组合索引结构是什么样的

如图所示：先第一个索引a查找，在找到a位置后，再对b进行查找。

### mysql索引如何进行优化

查询慢的原因：

1.硬件问题。如网络速度慢，内存不足，I/O吞吐量小，磁盘空间满了等。

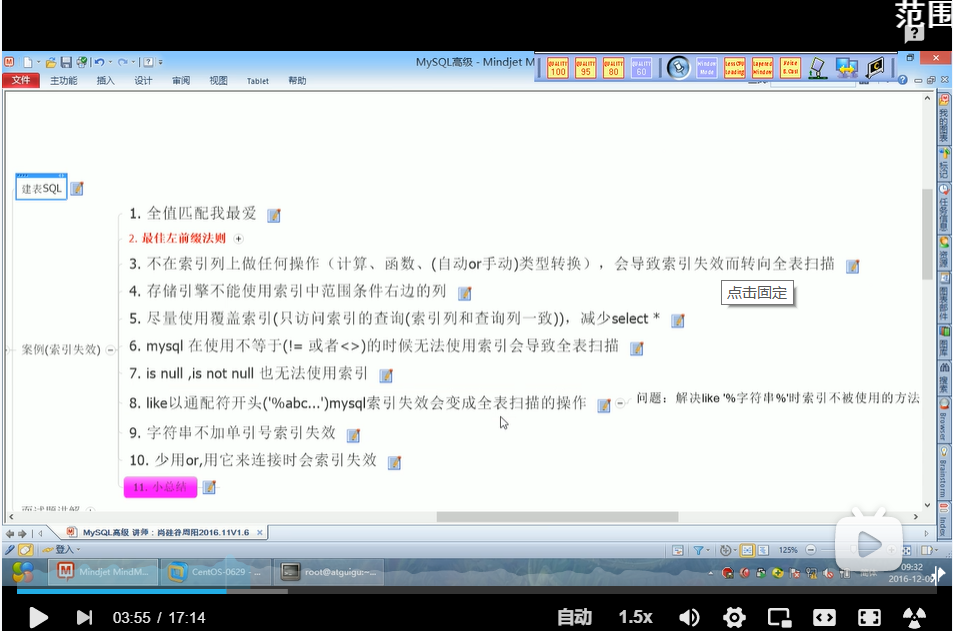
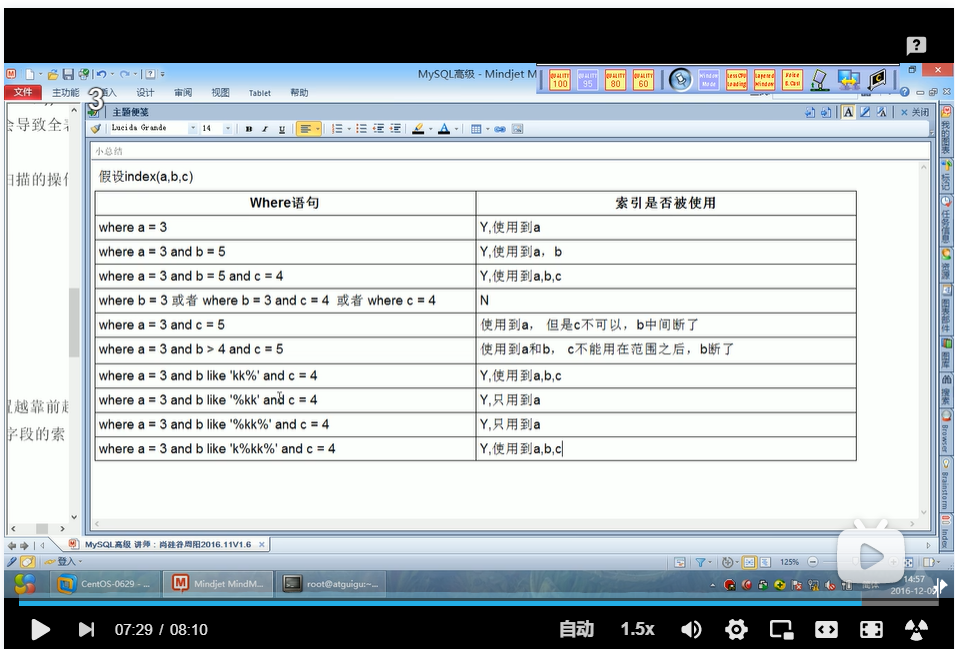
2.没有索引或者索引失效。（一般在互联网公司，DBA会在半夜把表锁了，重新建立一遍索引，因为当你删除某个数据的时候，索引的树结构就不完整了。所以互联网公司的数据做的是假删除.一是为了做数据分析,二是为了不破坏索引 ）

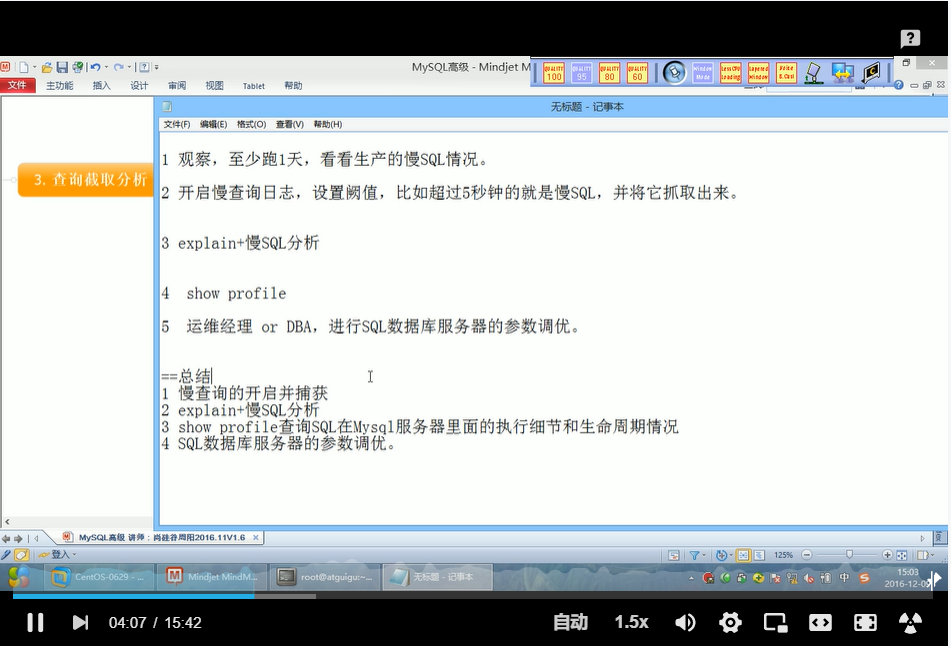
3.数据过多（分库分表）

4.服务器调优及各个参数设置（调整my.cnf）

10、覆盖索引

就是select的数据列只用从索引中就能够取得，不必从数据表中读取，换句话说查询列要被所使用的索引覆盖。

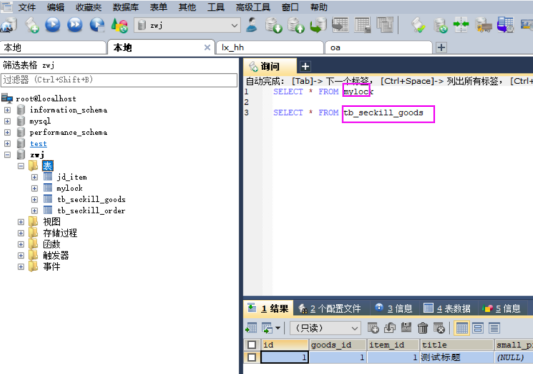
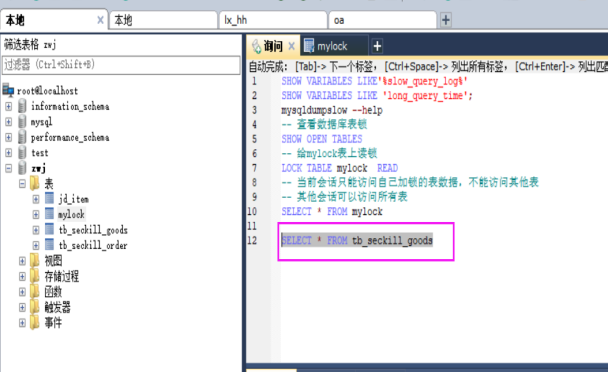


Sql慢查询的开启并捕获：<https://blog.csdn.net/qq_39291929/article/details/81561388>

### mysql读写锁

读锁：

* 1. 当mysql为一个表开启读锁的时候,其他进程包括进程本身没有权利去修改这张表的内容,也没有权利更新,但是可以读取表里面的内容



-- 查看数据库表锁

SHOW OPEN TABLES

-- 给mylock表上读锁

LOCK TABLE mylock READ

-- 当前会话只能访问自己加锁的表数据，不能访问其他表

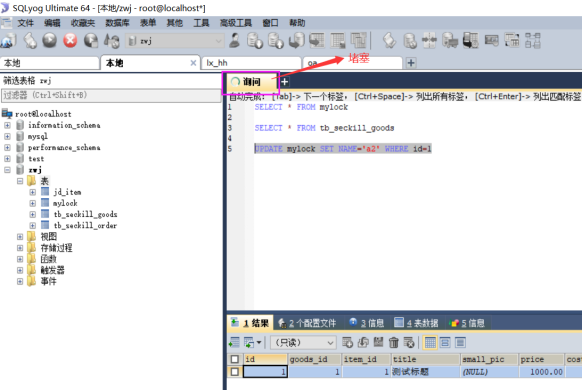
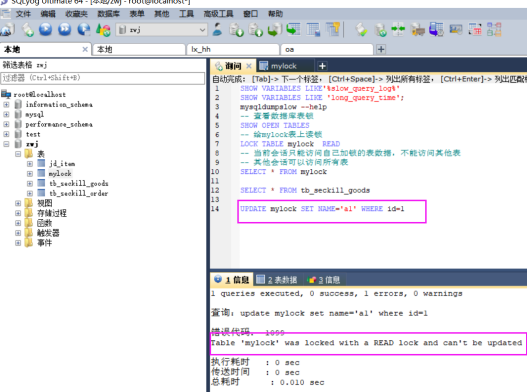
-- 其他会话可以访问所有表

SELECT \* FROM mylock

SELECT \* FROM tb\_seckill\_goods

-- 释放读锁

UNLOCK TABLES



本会话不能对加读锁的表进行增删改写操作；其他会话写操作会堵塞，知道加锁的表释放锁。

网上参考：<https://blog.csdn.net/qq_44766883/article/details/105879308>

写锁：

-- mylock加写锁

LOCK TABLE mylock WRITE

只有当前会话能增删改查，其他会话无法任何操作（阻塞，等待加锁会话释放锁）

网上参考：<https://www.cnblogs.com/CyLee/p/5575152.html>

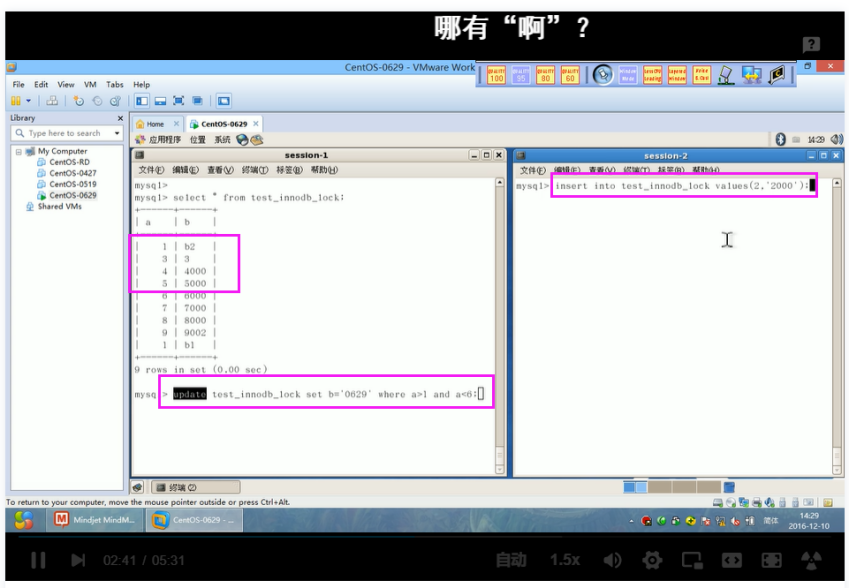
查看锁竞争情况：SHOW STATUS LIKE 'table%'

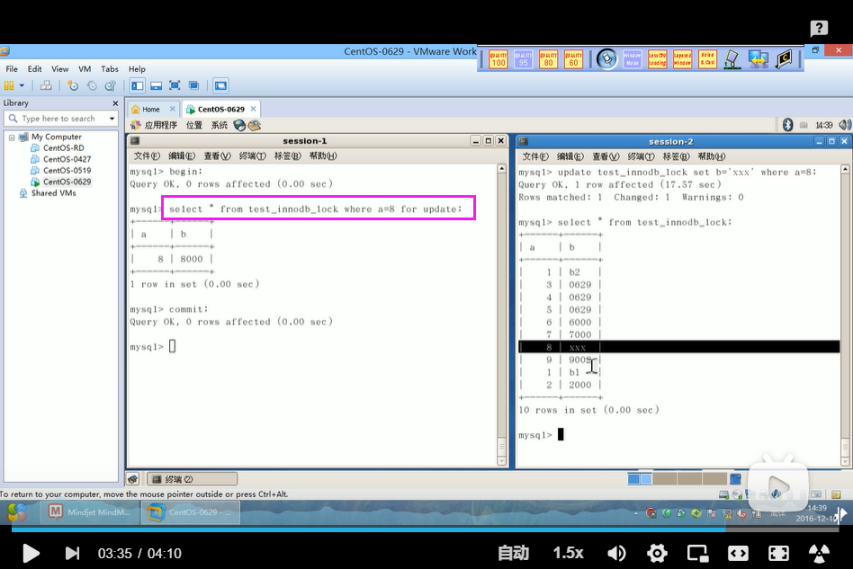
### 行锁

* 1. 索引失效，行锁变表锁

例子：如varchar类型的字段，主动不加’’,致索引失效，当前会话可以增删改，未提交完成的话(可设置手动提交 set commit 0效果更明显)，其他会话会阻塞。

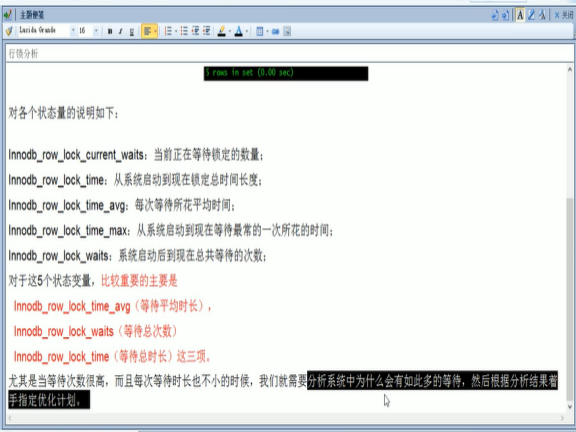
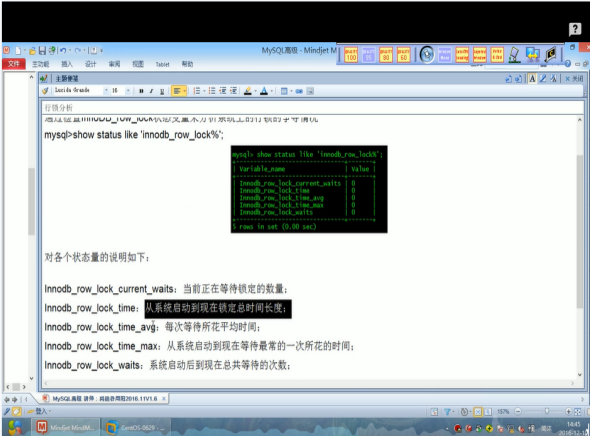
间隙锁的危害





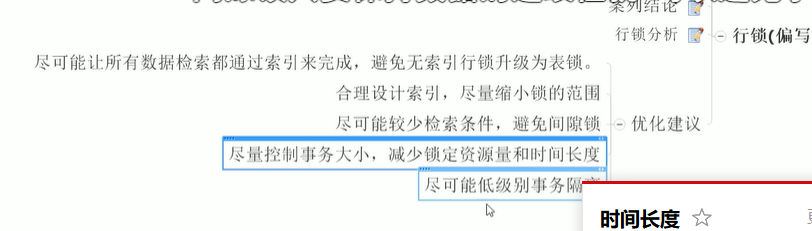
人为给表的某行加锁，select \* from table(表明) where id= 1 for update;

若当前会话未提交完成，其他会话阻塞



查看innodb锁情况

SHOW STATUS LIKE 'innodb\_row\_lock%'



### mysql缓存池

学习地址：一起看更容易懂

<https://mp.weixin.qq.com/s/AbiYMbDUJsMWK-_g0mFNwg>

https://blog.csdn.net/wuhenyouyuyouyu/article/details/93377605

## 消息队列

### RabbitMQ

### RabbitMQ 的使用场景有哪些？

* 抢购活动，削峰填谷，防止系统崩塌。
* 延迟信息处理，比如 10 分钟之后给下单未付款的用户发送邮件提醒。

解耦系统，对于新增的功能可以单独写模块扩展，比如用户确认评价之后，新增了给用户返积分的功能，这个时候不用在业务代码里添加新增积分的功能，只需要把新增积分的接口订阅确认评价的消息队列即可，后面再添加任何功能只需要订阅对应的消息队列即可。