# 总结

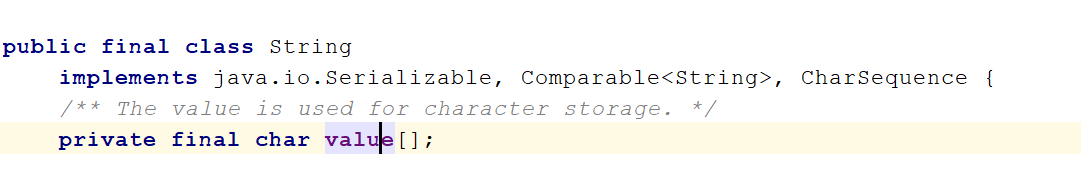
## Java基础：

### 面向过程(c)相对性能高于面向对象(java)的原因：

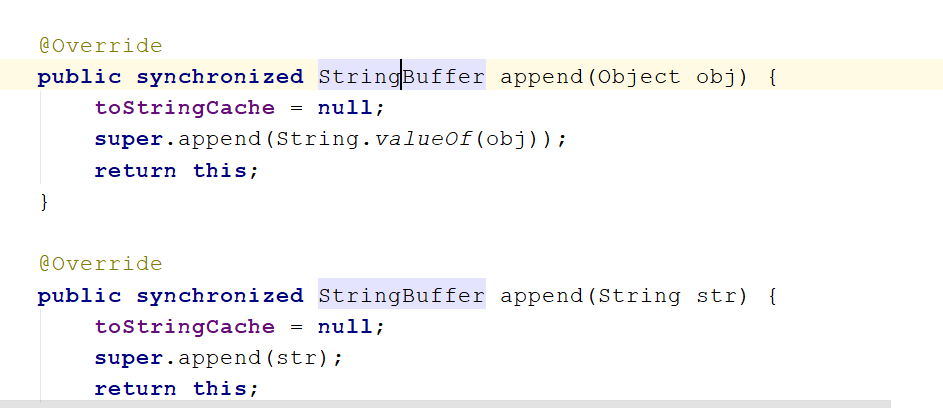
* 1. 面向过程经过一次编译为机器运行的机器码，而面向对象得先编译为字节码(.class)，然后java虚拟机转换为机器码运行。
  2. 面向过程：性能相对较好；
  3. 面向对象：易维护、易复用、易扩展。

### 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。

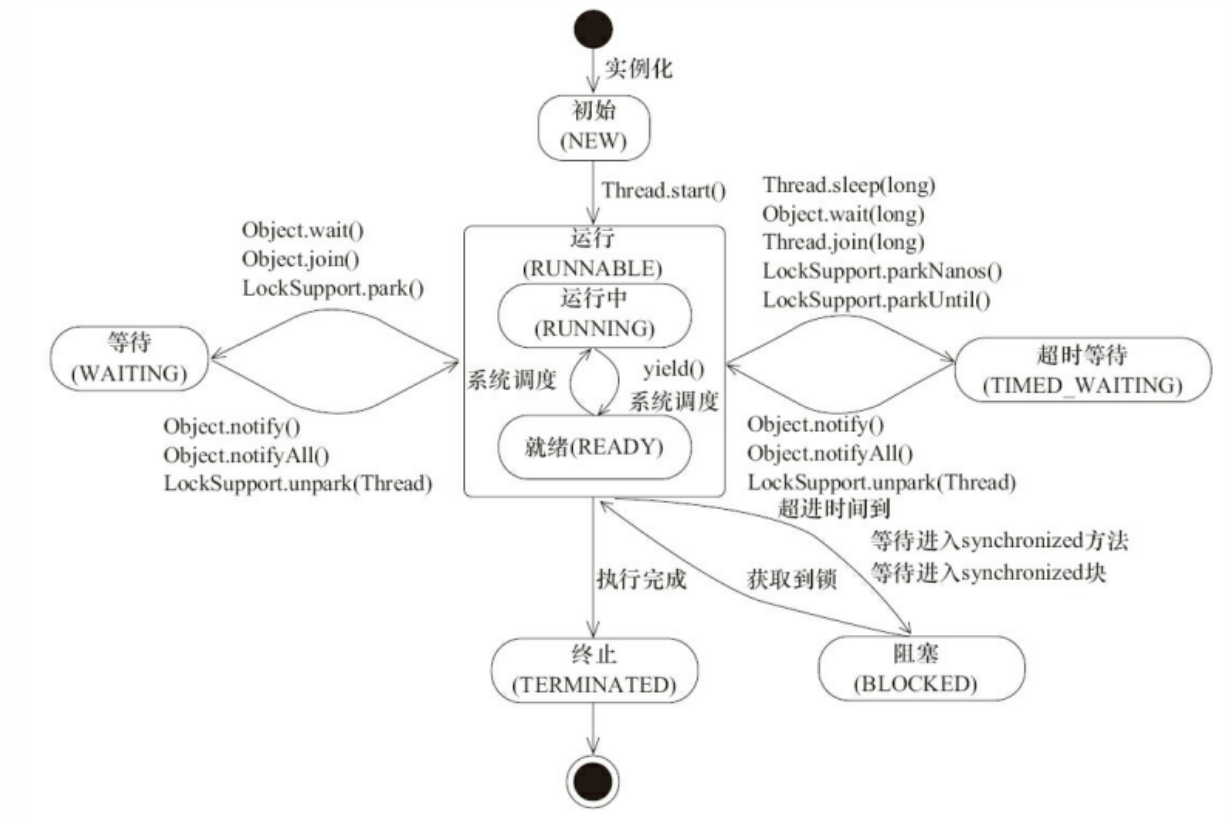


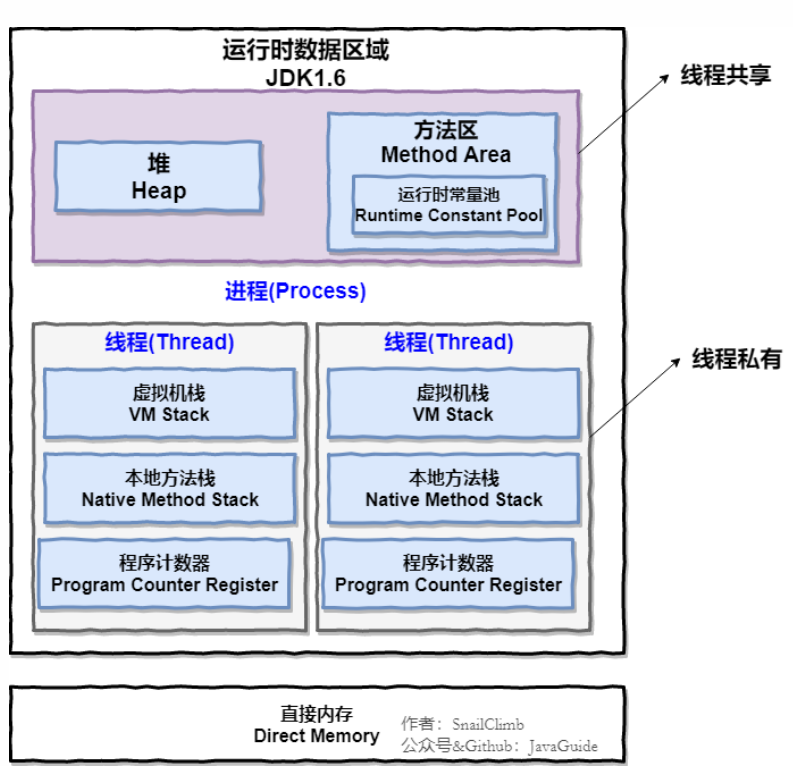
StringBuilder比StringBuffer性能高10%-15%，String适合操作少量数据；单线程操作大量数据使用StringBuilder，多线程操作大量数使用StringBuffer。

### 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：

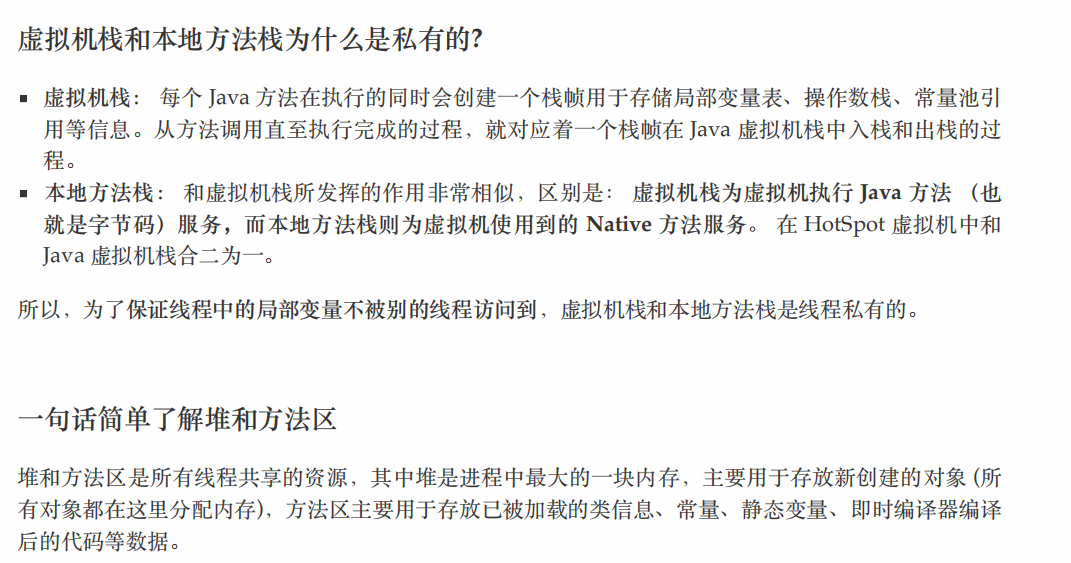




### 程序计数器为什么是线程私有的？

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。

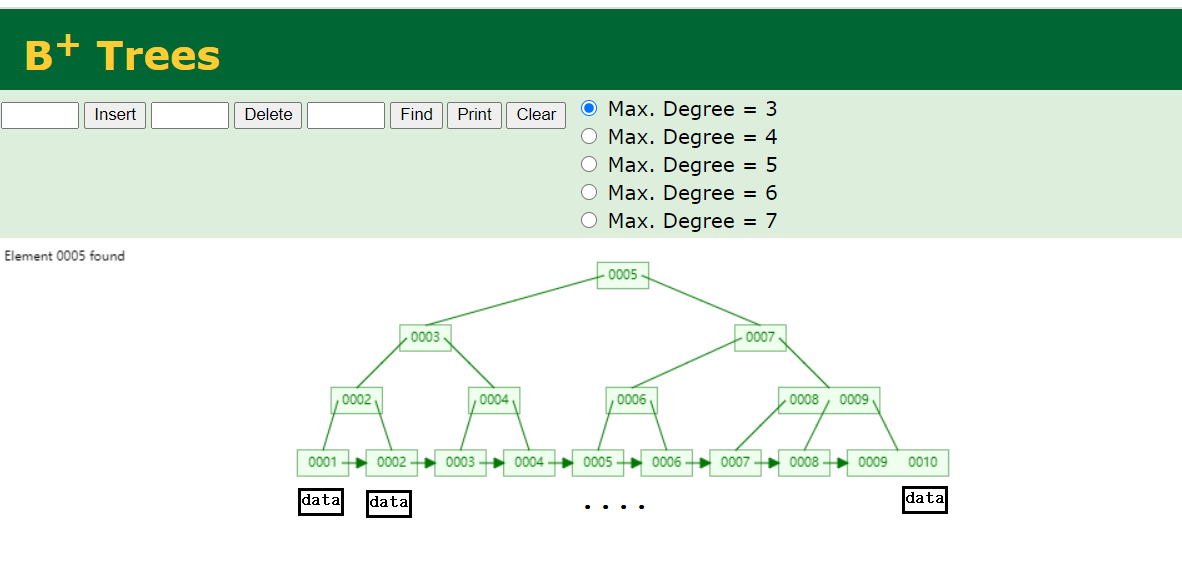
总结：所以程序计数器私有是为了线程切换后能恢复到正确的执行位置。



## 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？

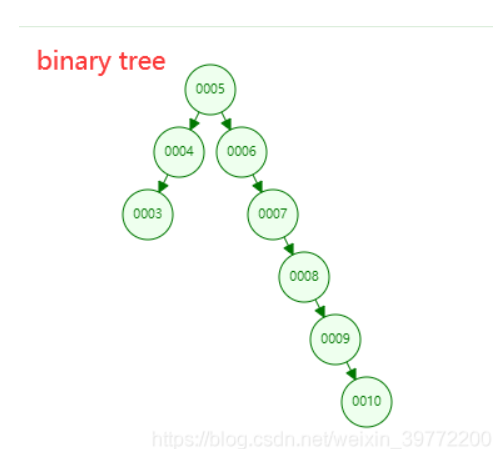
当查找的数据正好在线性表的最后一个索引时，则会进行全表扫描

#### （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。



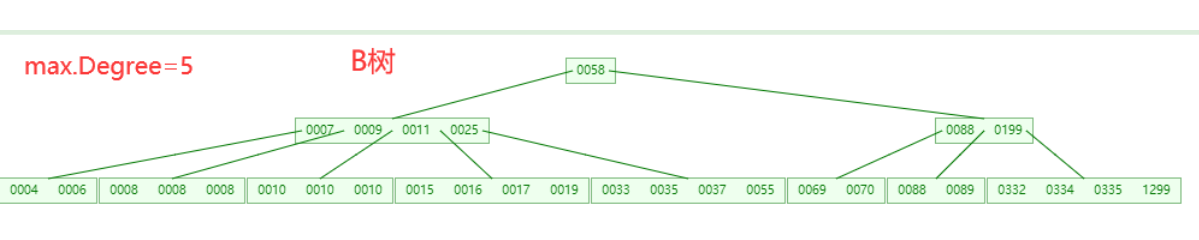
#### （4）为什么不用红黑树

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？

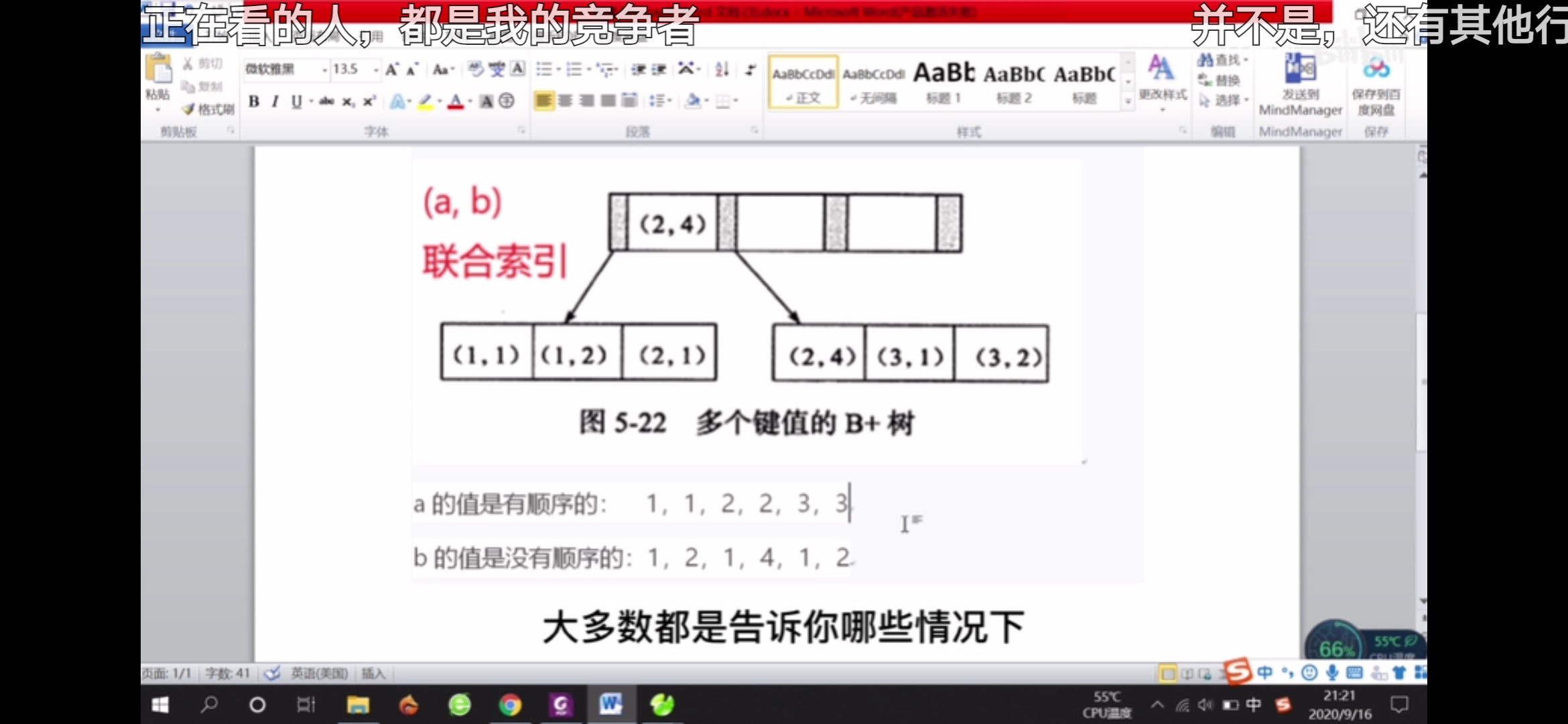
哈希算法虽然高，但也有它的局限性  
 1、 哈希索引只能操作：in、=，不支持范围查询，eg：>,<,!=  
原因：哈希索引比较的是经过哈希算法之后的哈希值，而哈希索引是无序的，用范围值的结果计算出来的哈希值并不一定和哈希索引中的排序是一样的。还有可能发生哈希碰撞。  
 2、 由于哈希索引本身是不排序的，所以任何时候都避免不了索引全表扫描。

#### （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。

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。

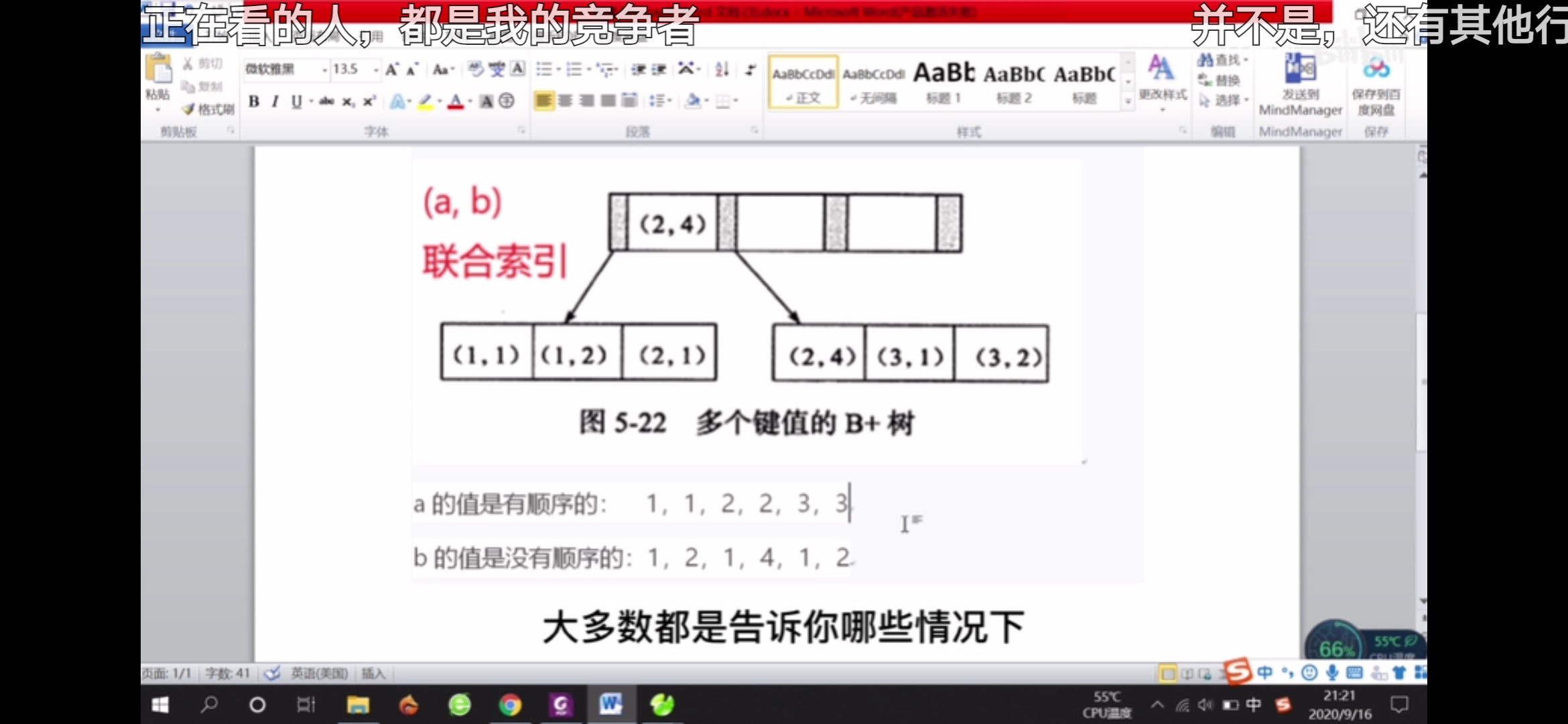
5、尽量使用数据量少的索引 如果索引的值很长，那么查询的速度会受到影响。

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）