目录

一、Java 基础面试题

1.String 能被继承吗？为什么？

不可以，因为 String 类有 final 修饰符，而 final 修饰的类是不能被继承的，实现细节不允许改变。平常我们定义的 String str=”abc”(直接赋一个字面量);其实和 String str=new String(“abc”)(通过构造器构造)还是有差异的。

String str=“abc”和 String str=new String(“abc”); 产生几个对象?

1.前者 1 或 0，后者 2 或 1，先看字符串常量池，如果字符串常量池中没有，都在常量池中创建一个，如果有，前者直接引用，后者在堆内存中还需创建一个“abc”实例对象。

2.对于基础类型的变量和常量：变量和引用存储在栈中，常量存储在常量池中。

3.为了提升 jvm（JAVA 虚拟机）性能和减少内存开销，避免字符的重复创建 项目中还是不要使用 new String 去创建字符串，最好使用 String 直接赋值。

2.String， Stringbuffer， StringBuilder 的区别。

String 字符串常量(final 修饰，不可被继承)，String 是常量，当创建之后即不能更改。(可以通过 StringBuffer 和 StringBuilder 创建 String 对象(常用的两个字符串操作类)。)

==StringBuffer 字符串变量（线程安全）,==其也是 final 类别的，不允许被继承，其中的绝大多数方法都进行了同步处理，包括常用的 Append 方法也做了同步处理(synchronized 修饰)。其自 jdk1.0 起就已经出现。其 toString 方法会进行对象缓存，以减少元素复制开销。

```
public synchronized String toString() {
    if (toStringCache == null) {
        toStringCache = Arrays.copyOfRange(value, 0, count);
    }
    return new String(toStringCache, true);
}
```

==StringBuilder 字符串变量（非线程安全）==其自 jdk1.5 起开始出现。与 StringBuffer 一样都继承和实现了同样的接口和类，方法除了没使用 synch 修饰以外基本一致，不同之处在于最后 toString 的时候，会直接返回一个新对象。

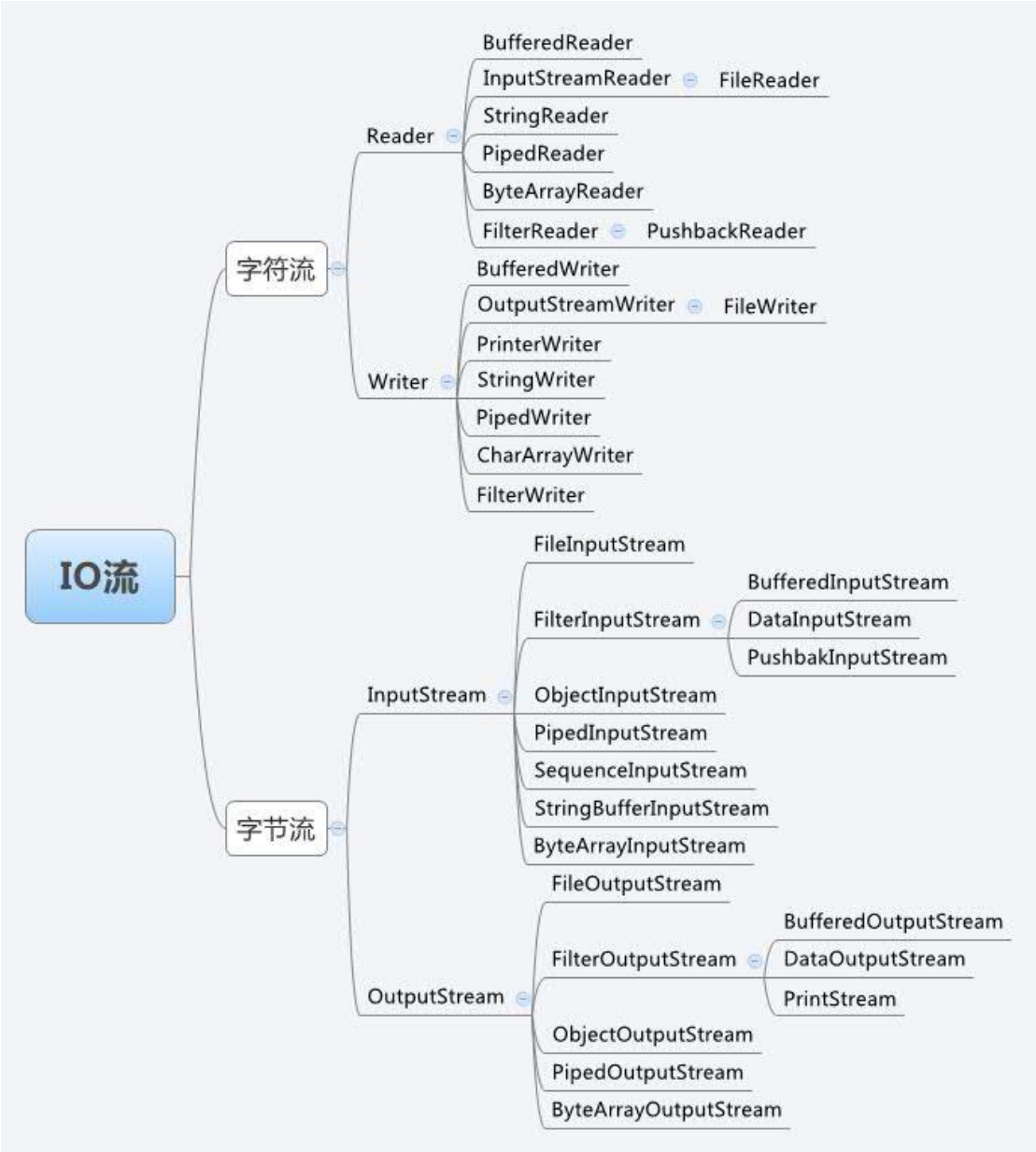
```
public String toString() {
    // Create a copy, don’ t share the array
    return new String(value, 0, count);
}
```

3. 说一说常见的输入输出流

计算机的存储器按用途可以分为主存储器和辅助存储器。

a. 主存储器又称内存，是 CPU 能直接寻址的存储空间，它的特点是存取速率快。内存一般采用半导体存储单元，包括随机存储器（RAM）、只读存储器（ROM）和高级缓存（Cache）。

b. 辅助存储器又称外存储器（简称外存），就是那些磁盘、硬盘、光盘，也就是你在电脑上看到的C、D、E、F盘。



根据处理数据类型的不同分为：字符流和字节流

字节流和字符流的区别：

读写单位不同：字节流以字节（8bit）为单位，字符流以字符为单位，根据码表映射字符，一次可能读多个字节。

处理对象不同：字节流能处理所有类型的数据（如图片、avi 等），而字符流只能处理字符类型的数据。

字节流：一次读入或读出是 8 位二进制。

字符流：一次读入或读出是 16 位二进制。

设备上的数据无论是图片或者视频，文字，它们都以二进制存储的。二进制的最终都是以一个 8 位为数据单元进行体现，所以计算机中的最小数据单元就是字节。意味着，字节流可以处理设备上的所有数据，所以字节流一样可以处理字符数据。

结论：只要是处理纯文本数据，就优先考虑使用字符流。 除此之外都使用字节流。

输入流只能进行读操作，输出流只能进行写操作，程序中需要根据待传输数据的不同特性而使用不同的流。

4.说一说 java 中的文件类 File

在 Java 语言的 java.io 包中，由 File 类提供了描述文件和目录的操作与管理方法。**File 类**

不同于输入输出流，它不负责数据的输入输出，而专门用来管理磁盘文件与目录。

File 类共提供了三个不同的构造函数，以不同的参数形式灵活地接收文件和目录名信息。构造函数：

1）File (String pathname)

例:File f1=new File("FileTest1.txt"); //创建文件对象 f1，f1 所指的文件是在当前目录下创建的 FileTest1.txt

2）File (String parent ， String child)

例:File f2 = new File(“D:\\dir1","FileTest2.txt") ; // 注意：D:\\dir1 目录事先必须存在，否则异常

3）File (File parent ，String child)

例:File f4=new File("\\dir3");

File f5=new File(f4,"FileTest5.txt"); // 在如果 \\dir3 目录不存在使用 f4.mkdir()先创建

一个对应于某磁盘文件或目录的 File 对象一经创建， 就可以通过调用它的方法来获得文件或目录的属性。

1）public boolean exists( )	判断文件或目录是否存在
2）public boolean isFile( )	判断是文件还是目录
3）public boolean isDirectory( )	判断是文件还是目录
4）public String getName( )	返回文件名或目录名
5）public String getPath( )	返回文件或目录的路径。

6）public long length( )　　获取文件的长度

7）public String[ ] list ( )　　将目录中所有文件名保存在字符串数组中返回。

File 类中还定义了一些对文件或目录进行管理、操作的方法，常用的方法有：

1） public boolean renameTo(File newFile);　　重命名文件

2） public void delete( );　　删除文件

3） public boolean mkdir( );　　创建目录

5.如何选择 IO 流：

1）确定是 输入还是输出

输入流： InputStream Reader

输出流： OutputStream Writer

2）明确操作的数据对象是否是纯文本

是: 字符流 Reader，Writer

否: 字节流 InputStream，OutputStream

3）明确具体的设备

硬盘文件:

读取：FileInputStream,, FileReader,

写入：FileOutputStream，FileWriter

内存用数组:

byte[]：ByteArrayInputStream, ByteArrayOutputStream

是 char[]：CharArrayReader, CharArrayWriter

键盘:

用 System.in（是一个 InputStream 对象）读取，用 System.out（是一个 OutoutStream 对象）打印

4）是否需要缓冲提高效率

加上 Buffered：BufferedInputStream, BufferedOuputStream, BuffereaReader, BufferedWriter

6.两个对象的 hashCode()相同，则 equals()也一定为 true，对吗？

不对，两个对象的 hashCode()相同，equals()不一定 true。

代码示例:

```
String str1 = "通话";
String str2 = "重地";
System.out.println(String.format("str1 : %d | str2 : %d",
str1.hashCode(),str2.hashCode()));
System.out.println(str1.equals(str2));
```

执行的结果：

```
str1：1179395 | str2：1179395
false
```

代码解读：很显然“通话”和“重地”的 hashCode() 相同，然而 equals() 则为 false，因为在散列表中，hashCode()相等即两个键值对的哈希值相等，然而哈希值相等，并不一定能得出键值对相等。

7.String 类的常用方法都有那些？

indexOf()：返回指定字符的索引。

charAt()：返回指定索引处的字符。

replace()：字符串替换。

trim()：去除字符串两端空白。

split()：分割字符串，返回一个分割后的字符串数组。

getBytes()：返回字符串的 byte 类型数组。

length()：返回字符串长度。

toLowerCase()：将字符串转成小写字母。

toUpperCase()：将字符串转成大写字符。

substring()：截取字符串。

equals()：字符串比较。

8.BIO、NIO、AIO 有什么区别？

BIO：Block IO 同步阻塞式 IO，就是我们平常使用的传统 IO，它的特点是模式简单使用方便，并发处理能力低。

NIO：New IO 同步非阻塞 IO，是传统 IO 的升级，客户端和服务器端通过 Channel（通道）通讯，实现了多路复用。

AIO：Asynchronous IO 是 NIO 的升级，也叫 NIO2，实现了异步非堵塞 IO ，异步 IO 的操作基于事件和回调机制。

二、Java 集合框架

1. 请先介绍一下 java 结合框架

1、List（有序、可重复）

List 里存放的对象是有序的，同时也是可以重复的，List 关注的是索引，拥有一系列和索引相关的方法，查询速度快。因为往 list 集合里插入或删除数据时，会伴随着后面数据的移动，所有插入删除数据速度慢。

List 是列表类型，以线性方式存储对象，自身的方法都与索引有关，个别常用方法如下。

方法	返回值	功能描述
add(int index, Object obj)	void	用来向集合中的指定索引位置添加对象，集合的索引位置从0开始，其他对象的索引位置相对向后移一位
set(int index, E element)	Object	用指定元素替换列表中指定位置的元素，返回以前在指定位置的元素
indexOf(Object obj)	int	返回列表中对象第一次出现的索引位置，如果集合中不包含该元素则返回-1
lastIndexOf(Object obj)	int	返回列表中对象最后一次出现的索引位置，如果集合汇总不包含该元素则返回-1
listIterator()	ListIterator	用来获得一个包含所有对象的ListIterator迭代器

2、Set（无序、不能重复）

Set 里存放的对象是无序，不能重复的，集合中的对象不按特定的方式排序，只是简单地把对象加入集合中。

Set 接口常用方法如下

方法	返回值	功能描述
add(Object obj)	boolean	若集合中尚存在未指定的元素，则添加此元素
addAll(Collection col)	boolean	将参数集合中所有元素添加到集合的尾部
remove(Object obj)	boolean	将指定的参数对象移除
clear()	void	移除此Set中的所有元素
iterator()	Iterator	返回此Set中的元素上进行迭代的迭代器
size()	int	返回此Set集合中的所有元素数
isEmpty()	boolean	如果Set不包含元素，则返回true

3、Map（键值对、键唯一、值不唯一）

Map 集合中存储的是键值对，键不能重复，值可以重复。根据键得到值，对 map 集合遍历时先得到键的 set 集合，对 set 集合进行遍历，得到相应的值。

Map 接口提供了将键映射到值的对象，一个映射不能包含重复的键，每个键最多只能映射

一个值。Map 接口同样提供了 clear()、isEmpty()、size()等方法，还有一些常用方法如下：

方法	返回值	功能描述
put(key k, value v)	Object	向集合中添加指定的key与value的映射关系
get(Object key)	boolean	如果存在指定的键对象，则返回该对象对应的值，否则返回null
values()	Collection	返回该集合中所有值对象形成的Collection集合

	A	B	C	D
1	类或接口		是否有序	是否重复
2	Collection		否	是
3	List		是	是
4	Set	AbstractSet	否	否
5		HashSet		
6		TreeSet	是（用二叉排序树）	否
7	Map	AbstractMap	否	否
8		HashMap		
9		TreeMap	是（用二叉排序树）	使用key-value来映射和存储数据，key必须唯一，value可以重复
10				

1. Interface Iterable

迭代器接口，这是 Collection 类的父接口。实现这个 Iterable 接口的对象允许使用 foreach 进行遍历，也就是说，所有的 Collection 集合对象都具有"foreach 可遍历性"。这个 Iterable 接口只有一个方法: iterator()。它返回一个代表当前集合对象的泛型<T>迭代器，用于之后的遍历操作

1.1 Collection

Collection 是最基本的集合接口，一个 Collection 代表一组 Object 的集合，这些 Object

被称作 Collection 的元素。Collection 是一个接口，用以提供规范定义，不能被实例化使用

1) Set

Set 集合类似于一个罐子，"丢进"Set 集合里的多个对象之间没有明显的顺序。Set 继承自 Collection 接口，不能包含有重复元素(记住，这是整个 Set 类层次的共有属性)。

Set 判断两个对象相同不是使用"=="运算符，而是根据 equals 方法。也就是说，我们在加入一个新元素的时候，如果这个新元素对象和 Set 中已有对象进行注意 equals 比较都返回 false， 则 Set 就会接受这个新元素对象，否则拒绝。

因为 Set 的这个制约，在使用 Set 集合的时候，应该注意两点：1) 为 Set 集合里的元素的实现类实现一个有效的 equals(Object)方法、2) 对 Set 的构造函数,传入的 Collection 参数不能包含重复的元素

1.1) HashSet

HashSet 是 Set 接口的典型实现，HashSet 使用 HASH 算法来存储集合中的元素，因此具有良好的存取和查找性能。当向 HashSet 集合中存入一个元素时，HashSet 会调用该对象的 hashCode()方法来得到该对象的 hashCode 值，然后根据该 HashCode 值决定该对象在 HashSet 中的存储位置。

值得主要的是，HashSet 集合判断两个元素相等的标准是两个对象通过 equals()方法比较相等，并且两个对象的 hashCode()方法的返回值相等

1.1.1) LinkedHashSet

LinkedHashSet 集合也是根据元素的 hashCode 值来决定元素的存储位置，但和 HashSet 不同的是，它同时使用链表维护元素的次序，这样使得元素看起来是以插入的顺序保存的。当遍历 LinkedHashSet 集合里的元素时，LinkedHashSet 将会按元素的添加

顺序来访问集合里的元素。

LinkedHashSet 需要维护元素的插入顺序，因此性能略低于 HashSet 的性能，但在迭代访问 Set 里的全部元素时(遍历)将有很好的性能(链表很适合进行遍历)

1.2) SortedSet

此接口主要用于排序操作，即实现此接口的子类都属于排序的子类

1.2.1) TreeSet

TreeSet 是 SortedSet 接口的实现类，TreeSet 可以确保集合元素处于排序状态

1.3) EnumSet

EnumSet 是一个专门为枚举类设计的集合类，EnumSet 中所有元素都必须是指定枚举类型的枚举值，该枚举类型在创建 EnumSet 时显式、或隐式地指定。EnumSet 的集合元素也是有序的，它们以枚举值在 Enum 类内的定义顺序来决定集合元素的顺序

2) List

List 集合代表一个元素有序、可重复的集合，集合中每个元素都有其对应的顺序索引。List 集合允许加入重复元素，因为它可以通过索引来访问指定位置的集合元素。List 集合默认按元素 的添加顺序设置元素的索引

2.1) ArrayList

ArrayList 是基于数组实现的 List 类，它封装了一个动态的增长的、允许再分配的 Object[]数组。

2.2) Vector

Vector 和 ArrayList 在用法上几乎完全相同，但由于 Vector 是一个古老的集合，所以 Vector 提供了一些方法名很长的方法，但随着 JDK1.2 以后，java 提供了系统的集合框架，就将 Vector 改为实现 List 接口，统一归入集合框架体系中

2.2.1) Stack

Stack 是 Vector 提供的一个子类，用于模拟"栈"这种数据结构(LIFO 后进先出)

2.3) LinkedList

implements List<E>, Deque<E>。实现 List 接口，能对它进行队列操作，即可以根据索引来随机访问集合中的元素。同时它还实现 Deque 接口，即能将 LinkedList 当作双端队列 使用。自然也可以被当作"栈来使用"

3) Queue

Queue 用于模拟"队列"这种数据结构(先进先出 FIFO)。队列的头部保存着队列中存放时间最长的元素，队列的尾部保存着队列中存放时间最短的元素。新元素插入(offer)到队列的尾部， 访问元素(poll)操作会返回队列头部的元素，队列不允许随机访问队列中的元素。结合生活中常见的排队就会很好理解这个概念

3.1) PriorityQueue

PriorityQueue 并不是一个比较标准的队列实现，PriorityQueue 保存队列元素的顺序并不是按照加入队列的顺序，而是按照队列元素的大小进行重新排序，这点从它的类名也可以 看出来

3.2) Deque

Deque 接口代表一个"双端队列"，双端队列可以同时从两端来添加、删除元素，因此 Deque 的实现类既可以当成队列使用、也可以当成栈使用

3.2.1) ArrayDeque

是一个基于数组的双端队列，和 ArrayList 类似，它们的底层都采用一个动态的、可重分配的 Object[]数组来存储集合元素，当集合元素超出该数组的容量时，系统会在底层重新分配一个 Object[]数组来存储集合元素

3.2.2) LinkedList

1.2 Map

Map 用于保存具有"映射关系"的数据，因此 Map 集合里保存着两组值，一组值用于保存 Map 里的 key，另外一组值用于保存 Map 里的 value。key 和 value 都可以是任何引用类型的数据。Map 的 key 不允许重复，即同一个 Map 对象的任何两个 key 通过 equals 方法比较结果总是返回 false。

关于 Map，我们要从代码复用的角度去理解，java 是先实现了 Map，然后通过包装了一个所有 value 都为 null 的 Map 就实现了 Set 集合

Map 的这些实现类和子接口中 key 集的存储形式和 Set 集合完全相同(即 key 不能重复)

Map 的这些实现类和子接口中 value 集的存储形式和 List 非常类似(即 value 可以重复、根据索引来查找)

1) HashMap

和 HashSet 集合不能保证元素的顺序一样，HashMap 也不能保证 key-value 对的顺序。并且类似于 HashSet 判断两个 key 是否相等的标准也是：两个 key 通过 equals()方法比较返回 true、 同时两个 key 的 hashCode 值也必须相等

1.1) LinkedHashMap

LinkedHashMap 也使用双向链表来维护 key-value 对的次序，该链表负责维护 Map 的迭代顺序，与 key-value 对的插入顺序一致(注意和 TreeMap 对所有的 key-value 进行排序进行区分)

2) Hashtable

是一个古老的 Map 实现类

2.1) Properties

Properties 对象在处理属性文件时特别方便(windows 平台上的.ini 文件)，Properties 类可以把 Map 对象和属性文件关联起来，从而可以把 Map 对象中的 key-value 对写入到属性文件中，也可以把属性文件中的"属性名-属性值"加载到 Map 对象中

3) SortedMap

正如 Set 接口派生出 SortedSet 子接口，SortedSet 接口有一个 TreeSet 实现类一样，Map 接口也派生出一个 SortedMap 子接口，SortedMap 接口也有一个 TreeMap 实现类

3.1) TreeMap

TreeMap 就是一个红黑树数据结构，每个 key-value 对即作为红黑树的一个节点。TreeMap 存储 key-value 对(节点)时，需要根据 key 对节点进行排序。TreeMap 可以保证所有的 key-value 对处于有序状态。同样，TreeMap 也有两种排序方式: 自然排序、定制排序

4) WeakHashMap

WeakHashMap 与 HashMap 的用法基本相似。区别在于，HashMap 的 key 保留了对实际对象的"强引用"，这意味着只要该 HashMap 对象不被销毁，该 HashMap 所引用的对象就不会被垃圾回收。但 WeakHashMap 的 key 只保留了对实际对象的弱引用，这意味着如果 WeakHashMap 对象的 key 所引用的对象没有被其他强引用变量所引用，则这些 key 所引用的对象可能被垃圾回收，当垃圾回收了该 key 所对应的实际对象之后，WeakHashMap 也可能自动删除这些 key 所对应的 key-value 对

5) IdentityHashMap

IdentityHashMap 的实现机制与 HashMap 基本相似，在 IdentityHashMap 中，当且仅当两个 key 严格相等(key1 == key2)时，IdentityHashMap 才认为两个 key 相等

6) EnumMap

EnumMap 是一个与枚举类一起使用的 Map 实现，EnumMap 中的所有 key 都必须是单个枚举类的枚举值。创建 EnumMap 时必须显式或隐式指定它对应的枚举类。EnumMap 根据 key 的自然顺序　　(即枚举值在枚举类中的定义顺序)

2 Vector 和 ArrayList 的区别

1，vector 是线程同步的，所以它也是线程安全的，而 arraylist 是线程异步的，是不安全的。如果不考虑到线程的安全因素，一般用 arraylist 效率比较高。

2，如果集合中的元素的数目大于目前集合数组的长度时，vector 增长率为目前数组长度的 100%，而 arraylist 增长率为目前数组长度的 50%。如果在集合中使用数据量比较大的数据，用 vector 有一定的优势。

3，如果查找一个指定位置的数据，vector 和 arraylist 使用的时间是相同的，如果频繁的访问数据，这个时候使用 vector 和 arraylist 都可以。而如果移动一个指定位置会导致后面的元素都发生移动，这个时候就应该考虑到使用 linkedlist,因为它移动一个指定位置的数据时其它元素不移动。

ArrayList 和 Vector 是采用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，都允许直接序号索引元素，但是插入数据要涉及到数组元素移动等内存操作，所以索引数据快，插入数据慢，Vector 由于使用了 synchronized 方法（线程安全）所以性能上比 ArrayList 要差，LinkedList 使用双向链表实现存储，按序号索引数据需要进行向前或向后遍历，但是插入数据时只需要记录本项的前后项即可，所以插入数度较快。

3　arraylist 和 linkedlist 的区别

1.ArrayList 是实现了基于动态数组的数据结构，LinkedList 基于链表的数据结构。

2.对于随机访问 get 和 set，ArrayList 觉得优于 LinkedList，因为 LinkedList 要移动指针。

3.对于新增和删除操作 add 和 remove，LinedList 比较占优势，因为 ArrayList 要移动数据。 这一点要看实际情况的。若只对单条数据插入或删除，ArrayList 的速度反而优于 LinkedList。但若是批量随机的插入删除数据，LinkedList 的速度大大优于 ArrayList. 因为 ArrayList 每插入一条数据，要移动插入点及之后的所有数据。

4 HashMap 与 TreeMap 的区别

1、 HashMap 通过 hashcode 对其内容进行快速查找，而 TreeMap 中所有的元素都保持着某种固定的顺序，如果你需要得到一个有序的结果你就应该使用 TreeMap（HashMap 中元素的排列顺序是不固定的）。

2、在 Map 中插入、删除和定位元素，HashMap 是最好的选择。但如果您要按自然顺序或自定义顺序遍历键，那么 TreeMap 会更好。使用 HashMap 要求添加的键类明确定义了 hashCode()和 equals()的实现。

两个 map 中的元素一样，但顺序不一样，导致 hashCode()不一样。

同样做测试：

在 HashMap 中，同样的值的 map,顺序不同，equals 时，false;

而在 treeMap 中，同样的值的 map,顺序不同,equals 时，true，说明，treeMap 在 equals() 时是整理了顺序了的。

5 HashTable 与 HashMap 的区别

1、同步性:Hashtable 是线程安全的，也就是说是同步的，而 HashMap 是线程序不安全的，不是同步的。

2、HashMap 允许存在一个为 null 的 key，多个为 null 的 value 。

3、hashtable 的 key 和 value 都不允许为 null。

6. 常用的集合类有哪些？

Map 接口和 Collection 接口是所有集合框架的父接口：

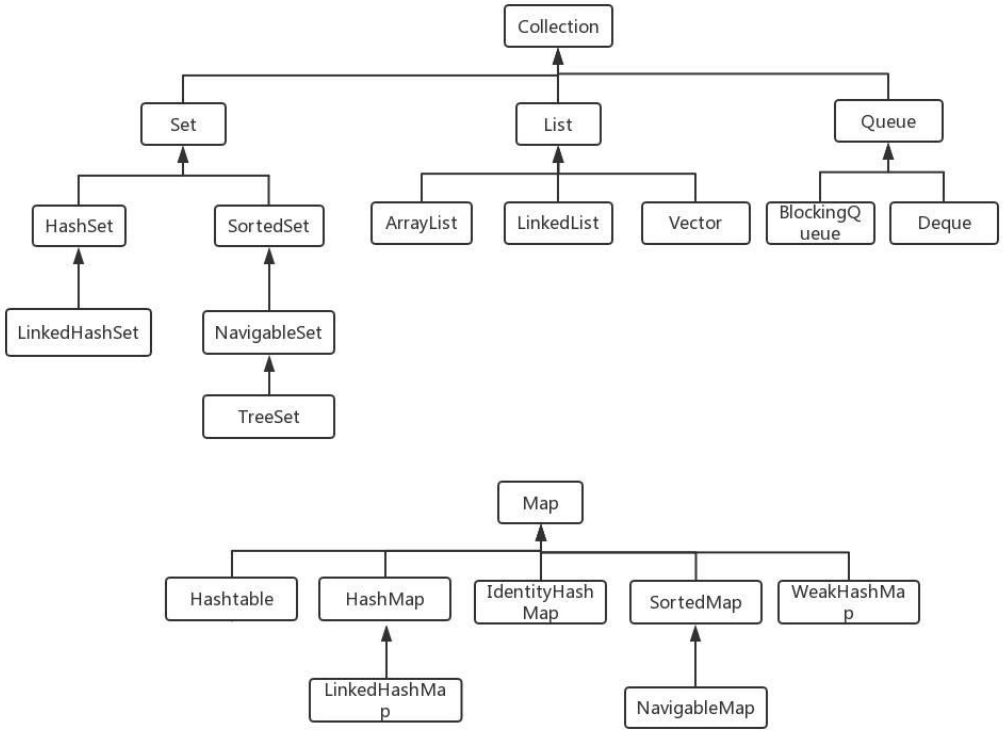
Collection 接口的子接口包括：Set 接口和 List 接口

Map 接口的实现类主要有：HashMap、TreeMap、Hashtable、ConcurrentHashMap 以及 Properties 等

Set 接口的实现类主要有：HashSet、TreeSet、LinkedHashSet 等

List 接口的实现类主要有：ArrayList、LinkedList、Stack 以及 Vector 等

7. List，Set，Map 三者的区别？List、Set、Map 是否继承自 Collection 接口？List、Map、Set 三个接口存取元素时，各有什么特点?



Java 容器分为 Collection 和 Map 两大类,Collection 集合的子接口有 Set、List、Queue 三种子接口。我们比较常用的是 Set、List，Map 接口不是 collection 的子接口。

Collection 集合主要有 List 和 Set 两大接口

List：一个有序（元素存入集合的顺序和取出的顺序一致）容器，元素可以重复，可以插入多个 null 元素，元素都有索引。常用的实现类有 ArrayList、LinkedList 和 Vector。

Set：一个无序（存入和取出顺序有可能不一致）容器，不可以存储重复元素，只允许存入一个 null 元素，必须保证元素唯一性。Set 接口常用实现类是 HashSet、LinkedHashSet 以及 TreeSet。

Map 是一个键值对集合，存储键、值和之间的映射。 Key 无序，唯一；value 不要求有序，允许重复。Map 没有继承于 Collection 接口，从 Map 集合中检索元素时，只要给出键对象，就会返回对应的值对象。

Map 的常用实现类： HashMap 、 TreeMap 、 HashTable 、 LinkedHashMap 、

ConcurrentHashMap

8. 集合框架底层数据结构

Collection

List

Arraylist： Object 数组

Vector： Object 数组

LinkedList： 双向循环链表

Set

HashSet（无序，唯一）：基于 HashMap 实现的，底层采用 HashMap 来保存元素

LinkedHashSet： LinkedHashSet 继承与 HashSet，并且其内部是通过 LinkedHashMap 来实现的。有点类似于我们之前说的 LinkedHashMap 其内部是基于 Hashmap 实现一样，不过还是有一点点区别的。

TreeSet（有序，唯一）： 红黑树(自平衡的排序二叉树。)

Map

HashMap： JDK1.8 之前 HashMap 由数组+链表组成的，数组是 HashMap 的主体，链表则是主要为了解决哈希冲突而存在的（“拉链法”解决冲突）.JDK1.8 以后在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为 8）时，将链表转化为红黑树，以减少搜索时间

LinkedHashMap：LinkedHashMap 继承自 HashMap，所以它的底层仍然是基于拉链式散列结构即由数组和链表或红黑树组成。另外，LinkedHashMap 在上面结构的基础上，增加了一条双向链表，使得上面的结构可以保持键值对的插入顺序。同时通过对链表进行相应的操作，实现了访问顺序相关逻辑。

HashTable： 数组+链表组成的，数组是 HashMap 的主体，链表则是主要为了解决哈希冲突而存在的

TreeMap： 红黑树（自平衡的排序二叉树）

9. 哪些集合类是线程安全的？

vector：就比 arraylist 多了个同步化机制（线程安全），因为效率较低，现在已经不太建议使用。在 web 应用中，特别是前台页面，往往效率（页面响应速度）是优先考虑的。

statck：堆栈类，先进后出。

hashtable：就比 hashmap 多了个线程安全。

enumeration：枚举，相当于迭代器。

10. 怎么确保一个集合不能被修改？

可以使用 Collections. unmodifiableCollection(Collection c) 方法来创建一个只读集合，这样改变集合的任何操作都会抛出 Java. lang. UnsupportedOperationException 异常。

示例代码如下：

```
List<String> list = new ArrayList<>();
list. add("x");
Collection<String> clist = Collections. unmodifiableCollection(list);
clist. add("y"); // 运行时此行报错
System. out. println(list. size());
```

11. 迭代器 Iterator 是什么？

Iterator 接口提供遍历任何 Collection 的接口。我们可以从一个 Collection 中使用迭代器方法来获取迭代器实例。迭代器取代了 Java 集合框架中的 Enumeration，迭代器允许调用者在迭代过程中移除元素。

Iterator 使用代码如下：

```
List<String> list = new ArrayList<>();
Iterator<String> it = list. iterator();
while(it. hasNext()){
   String obj = it. next();
   System. out. println(obj);
}
```

Iterator 的特点是只能单向遍历，但是更加安全，因为它可以确保，在当前遍历的集合元素被更改的时候，就会抛出 ConcurrentModificationException 异常。

12 说一下 ArrayList 的优缺点

ArrayList 的优点如下：

ArrayList 底层以数组实现，是一种随机访问模式。ArrayList 实现了 RandomAccess 接口，因此查找的时候非常快。

ArrayList 在顺序添加一个元素的时候非常方便。

ArrayList 的缺点如下：

删除元素的时候，需要做一次元素复制操作。如果要复制的元素很多，那么就会比较耗费性能。

插入元素的时候，也需要做一次元素复制操作，缺点同上。

ArrayList 比较适合顺序添加、随机访问的场景。

13. 如何实现数组和 List 之间的转换？

数组转 List：使用 Arrays. asList(array) 进行转换。

List 转数组：使用 List 自带的 toArray() 方法。

代码示例：

```
// list to array
List<String> list = new ArrayList<String>();
list.add("123");
list.add("456");
list.toArray();

// array to list
String[] array = new String[]{"123","456"};
Arrays.asList(array);
```

14. 插入数据时，ArrayList、LinkedList、Vector 谁速度较快？阐述 ArrayList、Vector、LinkedList 的存储性能和特性？

ArrayList、LinkedList、Vector 底层的实现都是使用数组方式存储数据。数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢。

Vector 中的方法由于加了 synchronized 修饰，因此 Vector 是线程安全容器，但性能上较 ArrayList 差。

LinkedList 使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但插入数

据时只需要记录当前项的前后项即可，所以 LinkedList 插入速度较快。

15. 多线程场景下如何使用 ArrayList？

ArrayList 不是线程安全的，如果遇到多线程场景，可以通过 Collections 的 synchronizedList 方法将其转换成线程安全的容器后再使用。例如像下面这样：

```
List<String> synchronizedList = Collections.synchronizedList(list);
synchronizedList.add("aaa");
synchronizedList.add("bbb");

for (int i = 0; i < synchronizedList.size(); i++) {
    System.out.println(synchronizedList.get(i));
}
```

16. HashSet 如何检查重复？HashSet 是如何保证数据不可重复的？

向 HashSet 中 add ()元素时，判断元素是否存在的依据，不仅要比较 hash 值，同时还要结合 equles 方法比较。

HashSet 中的 add ()方法会使用 HashMap 的 put()方法。

HashMap 的 key 是唯一的，由源码可以看出 HashSet 添加进去的值就是作为 HashMap 的 key，并且在 HashMap 中如果 K/V 相同时，会用新的 V 覆盖掉旧的 V，然后返回旧的 V。所以不会重复（ HashMap 比较 key 是否相等是先比较 hashcode 再比较 equals ）。

以下是 HashSet 部分源码：

```
private static final Object PRESENT = new Object();
private transient HashMap<E,Object> map;

public HashSet() {
    map = new HashMap<>();
}

public boolean add(E e) {
    // 调用 HashMap 的 put 方法,PRESENT 是一个至始至终都相同的虚值
    return map.put(e, PRESENT)==null;
}
```

hashCode（）与 equals（）的相关规定：

如果两个对象相等，则 hashcode 一定也是相同的

两个对象相等,对两个 equals 方法返回 true

两个对象有相同的 hashcode 值，它们也不一定是相等的

综上，equals 方法被覆盖过，则 hashCode 方法也必须被覆盖

hashCode()的默认行为是对堆上的对象产生独特值。如果没有重写 hashCode(),则该 class 的两个对象无论如何都不会相等（即使这两个对象指向相同的数据）。

==与 equals 的区别

==是判断两个变量或实例是不是指向同一个内存空间 equals 是判断两个变量或实例所指向的内存空间的值是不是相同

==是指对内存地址进行比较 equals()是对字符串的内容进行比较 3.==指引用是否相同 equals()指的是值是否相同

17.BlockingQueue 是什么？

Java.util.concurrent.BlockingQueue 是一个队列，在进行检索或移除一个元素的时候，它会等待队列变为非空；当在添加一个元素时，它会等待队列中的可用空间。BlockingQueue 接口是 Java 集合框架的一部分，主要用于实现生产者-消费者模式。我们不需要担心等待生产者有可用的空间，或消费者有可用的对象，因为它都在 BlockingQueue 的实现类中被处理了。Java 提供了集中 BlockingQueue 的实现，比如 ArrayBlockingQueue、LinkedBlockingQueue、PriorityBlockingQueue,、SynchronousQueue 等。

18. 说一下 HashMap 的实现原理？

HashMap 概述： HashMap 是基于哈希表的 Map 接口的非同步实现。此实现提供所有可选的映射操作，并允许使用 null 值和 null 键。此类不保证映射的顺序，特别是它不保证该顺序恒久不变。

HashMap 的数据结构： 在 Java 编程语言中，最基本的结构就是两种，一个是数组，另外一个是模拟指针（引用），所有的数据结构都可以用这两个基本结构来构造的，HashMap 也不例外。HashMap 实际上是一个“链表散列”的数据结构，即数组和链表的结合体。

HashMap 基于 Hash 算法实现的

当我们往 Hashmap 中 put 元素时，利用 key 的 hashCode 重新 hash 计算出当前对象的元素在数组中的下标

存储时，如果出现 hash 值相同的 key，此时有两种情况。(1)如果 key 相同，则覆盖原始值；(2)如果 key 不同（出现冲突），则将当前的 key-value 放入链表中

获取时，直接找到 hash 值对应的下标，在进一步判断 key 是否相同，从而找到对应值。

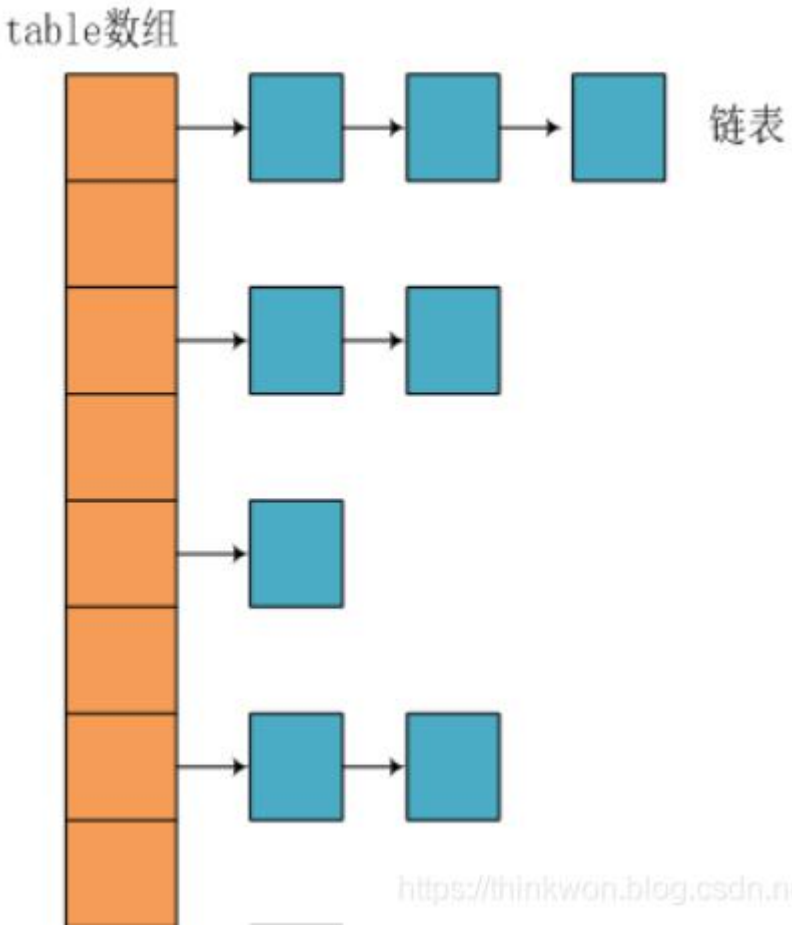
理解了以上过程就不难明白 HashMap 是如何解决 hash 冲突的问题，核心就是使用了数组的存储方式，然后将冲突的 key 的对象放入链表中，一旦发现冲突就在链表中做进一步的对比。

需要注意 Jdk 1.8 中对 HashMap 的实现做了优化，当链表中的节点数据超过八个之后，该链表会转为红黑树来提高查询效率，从原来的 O(n)到 O(logn)

19. HashMap 在 JDK1.7 和 JDK1.8 中有哪些不同？HashMap 的底层实现

在 Java 中，保存数据有两种比较简单的数据结构：数组和链表。**数组的特点是：寻址容易，插入和删除困难；链表的特点是：寻址困难，但插入和删除容易；所以我们将数组和链表结合在一起，发挥两者各自的优势，使用一种叫做拉链法的方式可以解决哈希冲突。**

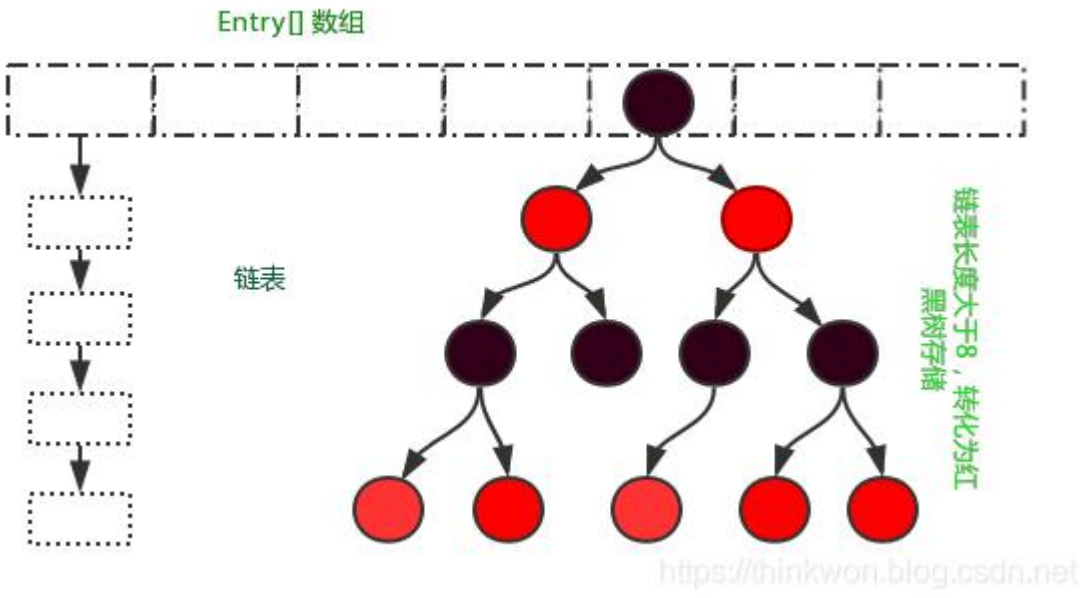
JDK1.8 之前



JDK1.8 之前采用的是拉链法。**拉链法**：将链表和数组相结合。也就是说创建一个链表数组，数组中每一格就是一个链表。若遇到哈希冲突，则将冲突的值加到链表中即可。

JDK1.8 之后

相比于之前的版本，jdk1.8 在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为 8）时，将链表转化为红黑树，以减少搜索时间。



JDK1.7 VS JDK1.8 比较

JDK1.8 主要解决或优化了一下问题：

resize 扩容优化

引入了红黑树，目的是避免单条链表过长而影响查询效率，红黑树算法请参考

解决了多线程死循环问题，但仍是非线程安全的，多线程时可能会造成数据丢失问题。

不同	JDK 1.7	JDK 1.8
存储结构	数组 + 链表	数组 + 链表 + 红黑树
初始化方式	单独函数：inflateTable()	直接集成到了扩容函数resize()中
hash值计算方式	扰动处理 = 9次扰动 = 4次位运算 + 5次异或运算	扰动处理 = 2次扰动 = 1次位运算 + 1次异或运算
存放数据的规则	无冲突时，存放数组；冲突时，存放链表	无冲突时，存放数组；冲突 & 链表长度 < 8：存放单链表；冲突 & 链表长度 > 8：树化并存放红黑树
插入数据方式	头插法（先讲原位置的数据移到后1位，再插入数据到该位置）	尾插法（直接插入到链表尾部/红黑树）
扩容后存储位置的计算方式	全部按照原来方法进行计算（即hashCode ->> 扰动函数 ->> (h&length-1)）	按照扩容后的规律计算（即扩容后的位置=原位置 or 原位置 + 旧容量）

20. HashMap 是怎么解决哈希冲突的？

答：在解决这个问题之前，我们首先需要知道**什么是哈希冲突**，而在了解哈希冲突之前我们还要知道**什么是哈希**才行；

什么是哈希？

Hash，一般翻译为“散列”，也有直接音译为“哈希”的，这就是把任意长度的输入通过散列算法，变换成固定长度的输出，该输出就是散列值（哈希值）；这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，所以不可能从散列值来唯一的确定输入值。**简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。**

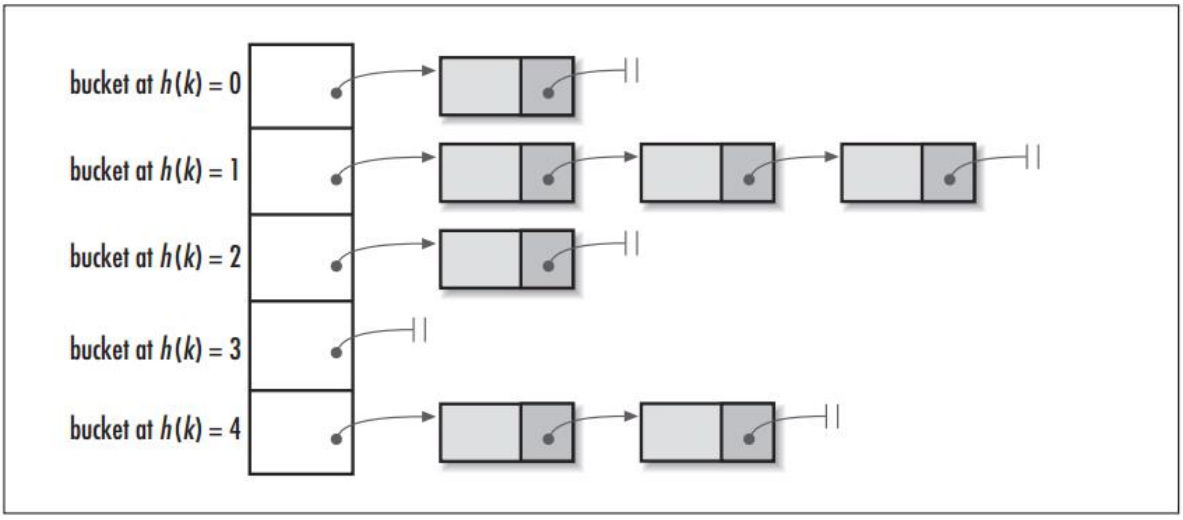
所有散列函数都有如下一个基本特性**：根据同一散列函数计算出的散列值如果不同，那么输入值肯定也不同。但是，根据同一散列函数计算出的散列值如果相同，输入值不一定相同**。

什么是哈希冲突？

当两个不同的输入值，根据同一散列函数计算出相同的散列值的现象，我们就把它叫做碰撞（哈希碰撞）。

HashMap 的数据结构

在 Java 中，保存数据有两种比较简单的数据结构：数组和链表。**数组的特点是：寻址容易，插入和删除困难；链表的特点是：寻址困难，但插入和删除容易**；所以我们将数组和链表结合在一起，发挥两者各自的优势，使用一种叫做链地址法的方式可以解决哈希冲突：



这样我们就可以将拥有相同哈希值的对象组织成一个链表放在 hash 值所对应的 bucket 下，**但相比于 hashCode 返回的 int 类型，我们 HashMap 初始的容量大小 DEFAULT_INITIAL_CAPACITY = 1 << 4（即 2 的四次方 16）要远小于 int 类型的范围，所以我们如果只是单纯的用 hashCode 取余来获取对应的 bucket 这将会大大增加哈希碰撞的概率，并且最坏情况下还会将 HashMap 变成一个单链表**，所以我们还需要对 hashCode 作一定的优化

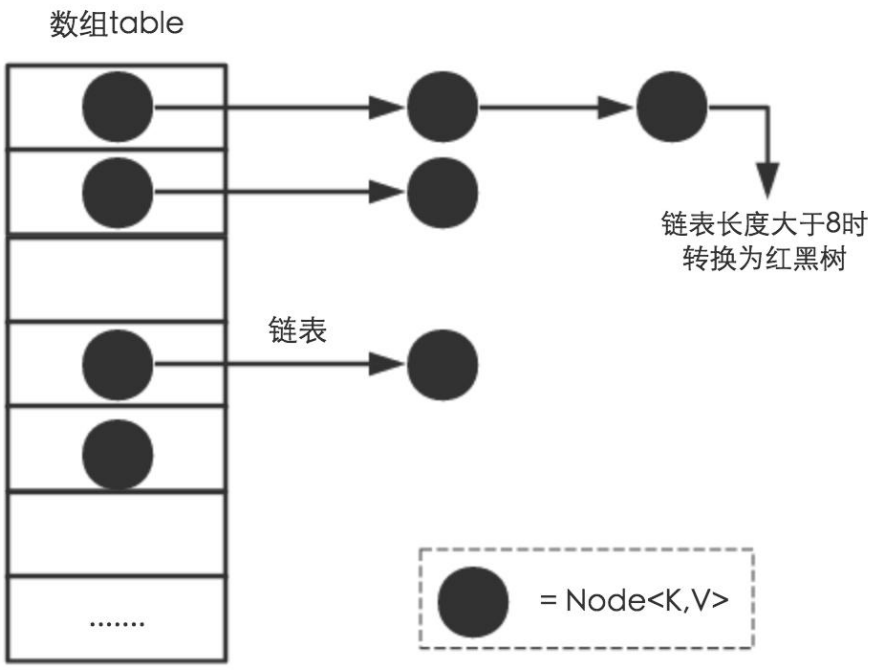
hash()函数

上面提到的问题，主要是因为如果使用 hashCode 取余，那么相当于**参与运算的只有 hashCode 的低位**，高位是没有起到任何作用的，所以我们的思路就是让 hashCode 取值出的高位也参与运算，进一步降低 hash 碰撞的概率，使得数据分布更平均，我们把这样的操作称为扰动，在 JDK 1.8 中的 hash()函数如下：

```
static final int hash(Object key) {
    int h;
    return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);// 与自己右移 16 位进行异或运算（高低位异或）
}
```

这比在 JDK 1.7 中，更为简洁，**相比在 1.7 中的 4 次位运算，5 次异或运算（9 次扰动），在 1.8 中，只进行了 1 次位运算和 1 次异或运算（2 次扰动）；**

JDK1.8 新增红



通过上面的链地址法（使用散列表）和扰动函数我们成功让我们的数据分布更平均，哈希碰撞减少，但是当我们的 HashMap 中存在大量数据时，加入我们某个 bucket 下对应的链表有 n 个元素，那么遍历时间复杂度就为 O(n)，为了针对这个问题，JDK1.8 在 HashMap 中新增了红黑树的数据结构，进一步使得遍历复杂度降低至 O(logn)；

总结

简单总结一下 HashMap 是使用了哪些方法来有效解决哈希冲突的：

1. 使用链地址法（使用散列表）来链接拥有相同 hash 值的数据；
2. 使用 2 次扰动函数（hash 函数）来降低哈希冲突的概率，使得数据分布更平均；
3. 引入红黑树进一步降低遍历的时间复杂度，使得遍历更快；

21. 如果使用 Object 作为 HashMap 的 Key，应该怎么

办呢？

答：重写 hashCode()和 equals()方法

重写 hashCode()是因为需要计算存储数据的存储位置，需要注意不要试图从散列码计算中排除掉一个对象的关键部分来提高性能，这样虽然能更快但可能会导致更多的 Hash 碰撞；

重写 equals()方法，需要遵守自反性、对称性、传递性、一致性以及对于任何非 null 的引用值 x，x.equals(null)必须返回 false 的这几个特性，目的是**为了保证 key 在哈希表中的唯一性；**

22. HashMap 与 HashTable 有什么区别？

线程安全： HashMap 是非线程安全的，HashTable 是线程安全的；HashTable 内部的方法基本都经过 synchronized 修饰。（如果你要保证线程安全的话就使用 ConcurrentHashMap 吧！）；

效率： 因为线程安全的问题，HashMap 要比 HashTable 效率高一点。另外，HashTable 基本被淘汰，不要在代码中使用它；

对 Null key 和 Null value 的支持： HashMap 中，null 可以作为键，这样的键只有一个，可以有一个或多个键所对应的值为 null。但是在 HashTable 中 put 进的键值只要有一个 null，直接抛 NullPointerException。

**初始容量大小和每次扩充容量大小的不同 **： ①创建时如果不指定容量初始值，Hashtable 默认的初始大小为 11，之后每次扩充，容量变为原来的 2n+1。HashMap 默认的初始化大小为 16。之后每次扩充，容量变为原来的 2 倍。②创建时如果给定了容量初

始值，那么 Hashtable 会直接使用你给定的大小，而 HashMap 会将其扩充为 2 的幂次方大小。也就是说 HashMap 总是使用 2 的幂作为哈希表的大小，后面会介绍到为什么是 2 的幂次方。

底层数据结构： JDK1.8 以后的 HashMap 在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为 8）时，将链表转化为红黑树，以减少搜索时间。Hashtable 没有这样的机制。

推荐使用：在 Hashtable 的类注释可以看到，Hashtable 是保留类不建议使用，推荐在单线程环境下使用 HashMap 替代，如果需要多线程使用则用 ConcurrentHashMap 替代。

23. ConcurrentHashMap 和 Hashtable 的区别？

ConcurrentHashMap 和 Hashtable 的区别主要体现在实现线程安全的方式上不同。

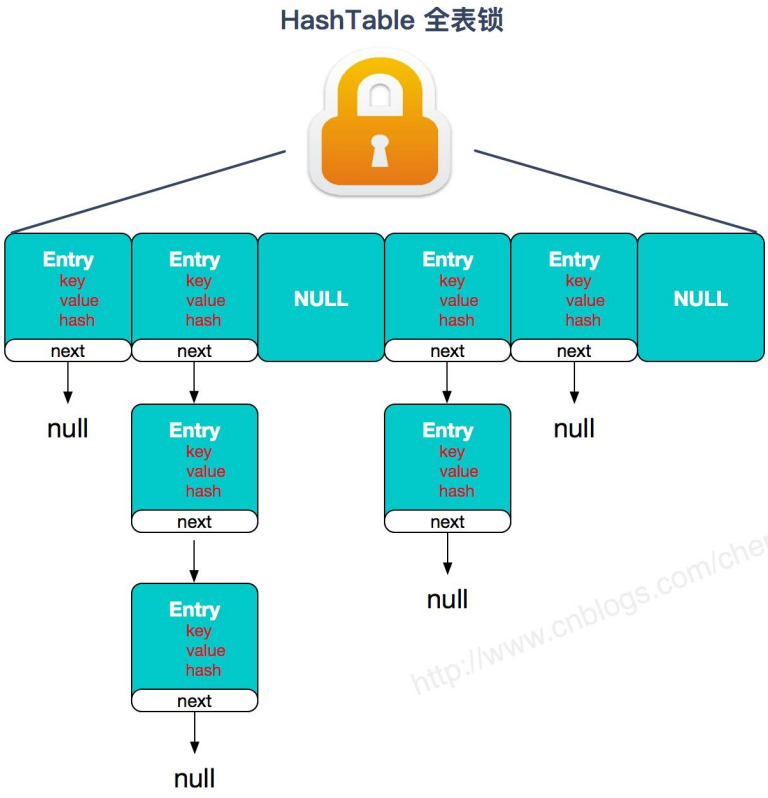
底层数据结构： JDK1.7 的 ConcurrentHashMap 底层采用 分段的数组+链表 实现，JDK1.8 采用的数据结构跟 HashMap1.8 的结构一样，数组+链表/红黑二叉树。Hashtable 和 JDK1.8 之前的 HashMap 的底层数据结构类似都是采用 数组+链表 的形式，数组是 HashMap 的主体，链表则是主要为了解决哈希冲突而存在的；

实现线程安全的方式（重要）： ① 在 JDK1.7 的时候，ConcurrentHashMap（分段锁）对整个桶数组进行了分割分段(Segment)，每一把锁只锁容器其中一部分数据，多线程访问容器里不同数据段的数据，就不会存在锁竞争，提高并发访问率。（默认分配 16 个 Segment，比 Hashtable 效率提高 16 倍。） 到了 JDK1.8 的时候已经摒弃了 Segment 的概念，而是直接用 Node 数组+链表+红黑树的数据结构来实现，并发控制使用 synchronized 和

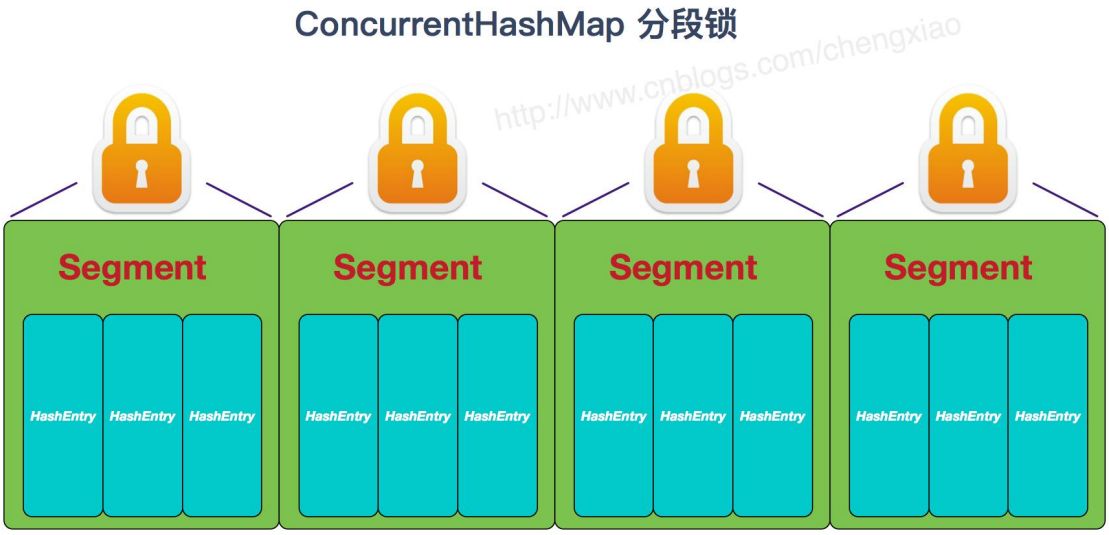
CAS 来操作。（JDK1.6 以后 对 synchronized 锁做了很多优化） 整个看起来就像是优化过且线程安全的 HashMap，虽然在 JDK1.8 中还能看到 Segment 的数据结构，但是已经简化了属性，只是为了兼容旧版本；② Hashtable(同一把锁) :使用 synchronized 来保证线程安全，效率非常低下。当一个线程访问同步方法时，其他线程也访问同步方法，可能会进入阻塞或轮询状态，如使用 put 添加元素，另一个线程不能使用 put 添加元素，也不能使用 get，竞争会越来越激烈效率越低。

两者的对比图：

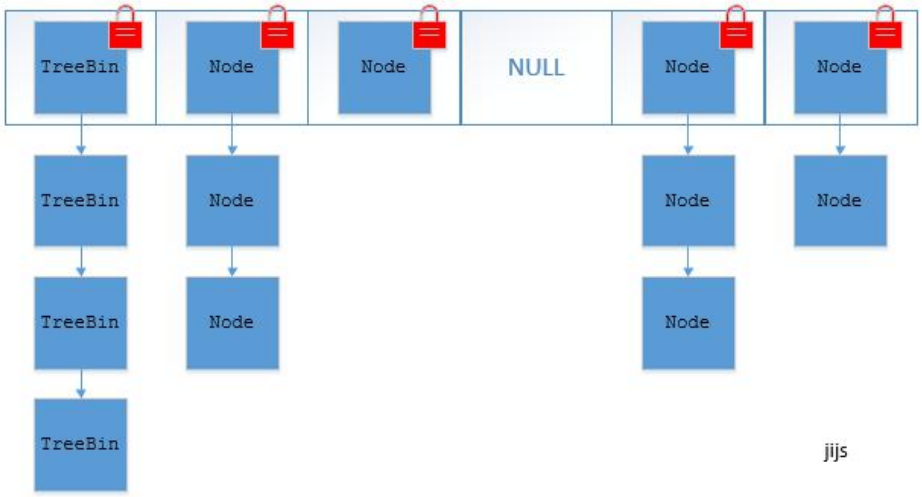
HashTable:



JDK1.7 的 ConcurrentHashMap：



JDK1.8 的 ConcurrentHashMap（TreeBin: 红黑二叉树节点 Node: 链表节点）：



答：

ConcurrentHashMap 结合了 HashMap 和 HashTable 二者的优势。

HashMap 没有考虑同步，HashTable 考虑了同步的问题。

但是 HashTable 在每次同步执行时都要锁住整个结构。 ConcurrentHashMap 锁的方式是稍微细粒度的。

24. TreeMap 和 TreeSet 在排序时如何比较元素？Collections 工具类中的 sort()方法如何比较元素？

TreeSet 要求存放的对象所属的类必须实现 Comparable 接口，该接口提供了比较元素的 compareTo()方法，当插入元素时会回调该方法比较元素的大小。TreeMap 要求存放的键值对映射的键必须实现 Comparable 接口从而根据键对元素进行排序。

Collections 工具类的 sort 方法有两种重载的形式，

第一种要求传入的待排序容器中存放的对象比较实现 Comparable 接口以实现元素的比较；

第二种不强制性的要求容器中的元素必须可比较，但是要求传入第二个参数，参数是 Comparator 接口的子类型（需要重写 compare 方法实现元素的比较），相当于一个临时定义的排序规则，其实就是通过接口注入比较元素大小的算法，也是对回调模式的应用（Java 中对函数式编程的支持）。

三、Linux 常用指令

1. 常见命令

```
ctrl c 退出当前执行

cd .. 返回到上一级目录中

cd ~ 返回到根目录中 或者是 cd (加个空格)

cd - 返回进入此目录之前所在的目录

exit 退出
```

2. 常见的操作文件，文件夹的命令

```
# ls — List ： 列举出当前工作目录的内容（文件或文件夹）

ls -l # 查看文件名称和 文件的权限

ls -a # 显示全部的文件，包括隐藏的文件


# clear ： 清除当前命令


# mkdir — Make Directory ： 创建一个新文件夹


# rmdir— Remove Directory ： 删除一个文件夹


# pwd — Print Working Directory ： 显示当前目录
```

cd — Change Directory ： 切换文件路径，cd 将给定的文件夹（或目录）设置成当前工作目录

rm— Remove ： 删除指定的文件。 在 linux 没有回收站，删除之后无法恢复

rm -rf 文件（r 递归删除，f 直接强行删除）

sudo rm -r hadoop # 删除文件夹下的所有文件

vi ： 创建一个文件 vi test.txt

一种比较方便打开文件的方式

sudo gedit + 文件

>是覆盖，>> 是追加

echo >： 向已有的文件中增加内容， echo "added contents" >>test.txt

echo >>： 覆盖文件内容，若不存在则创建一个新的文件 echo "new file" >newFile.txt

cat— concatenate and print files ： 显示一个文件 cat test.txt

less, more ： 如果文件太大，分页显示一个文件，按 Q 结束浏 览

tail ： 显示文件的最后 10 行， tail -n 5 test.txt 显示最后 5 行

touch ： 创建一个新的空的文件，不直接进行编辑

cp— Copy ： 对文件进行复制 ；例子： cp test.txt ../jun2

mv— Move ： 对文件或文件夹进行移动

mv hello.csv ./python：把当前目录的 hello.csv 剪切到当前目的 python 文件夹里

grep ： 在文件中查找指定的字符; grep fun test.txt

tar ： tar 命令能创建、查看和提取 tar 压缩文件。

tar -cvf 是创建对应压缩文件， tar -cvf test.tar test.txt # 将 test.txt 压缩为 test.tar

tar -tvf 来查看对应压缩文件，

tar -xvf 来提取对应压缩文件

mkdir — Make Directory ： 创建一个新目录

mkdir — Make Directory ： 创建一个新目录

mkdir — Make Directory ： 创建一个新目录

3. 软件下载安装

```
sudo apt-get update     # 更新一下软件源，获取最新软件的列表

sudo apt-get install gedit(软件名)     # 安装软件
```

4. 系统重启和关机指令

```
#  reboot ：  立刻进行重启

#  shutdown -r now ：  立刻进行重启

#  shutdown -r 10 ：  10 分钟后进行重启


# 关机命令

#  halt ：  立刻关机

#  poweroff ：  立刻关机

#  shutdown -h now ：  立刻关机
```

5. 文件模式和访问权限

```
#  chmod (change mode) ：命令来改变文件或目录的访问权限
```

-rwxr-xr-- ：含义表示

第一列的字符可以分为三组，每一组有三个，每个字符都代表不同的权限，分别为读取(r)、写入(w)和执行(x)：

u 第一组字符(2-4)表示文件所有者的权限，-rwxr-xr-- 表示所有者拥有读取(r)、写入(w)和执行(x)的权限。

g 第二组字符(5-7)表示文件所属用户组的权限，-rwxr-xr-- 表示该组拥有读取(r)和执行(x)的权限，但没有写入权限。

o 第三组字符(8-10)表示所有其他用户的权限，rwxr-xr-- 表示其他用户只能读取(r)文件。

目录的访问模式为：

读取：用户可以查看目录中的文件

写入：用户可以在当前目录中删除文件或创建文件

执行：执行权限赋予用户遍历目录的权利，例如执行 cd 和 ls 命令。（只是对于目录而言）

chmod o=rwx anoterDir ： 设置其他用户的权限，o,

- 表示删除权限， + 表示增加权限

6. 环境变量

常见的环境环境变量。 输入的形式为： echo $Home

#--- 使用 env 显示所有的环境变量

```
HOME：指定用户的主工作目录 ，如： echo $Home

LOGNAME：指当前用户的登录名

HOSTNAME：指主机的名称

LANG/LANGUGE：和语言相关的环境变量

vim ~/.bashrc       #进行环境配置

source ~/.bashrc   # 使配置立即生效

export 设置一个新的环境变量 export HELLO="hello" (可以无引号)

unset 清除环境变量 unset HELLO
```

7. ubuntu 登陆到 mysql

```
(登陆到 mysql 中)

#---启动和终止 mysql-----

service mysql start


service mysql restart

service mysql stop


#---打开 mysql 的 shell 命令

mysql -u root -p
```

四、MySQL 基础面试

1. 三个范式是什么

第一范式（1NF）：数据库表中的字段都是单一属性的，不可再分。这个单一属性由基本类型构成，包括整型、实数、字符型、逻辑型、日期型等。

第二范式（2NF）：数据库表中不存在非关键字段对任一候选关键字段的部分函数依赖（部分函数依赖指的是存在组合关键字中的某些字段决定非关键字段的情况），也即所有非关键字段都完全依赖于任意一组候选关键字。

第三范式（3NF）：在第二范式的基础上，数据表中如果不存在非关键字段对任一候选关键字段的传递函数依赖则符合第三范式。所谓传递函数依赖，指的是如果存在"A → B → C"的决定关系，则 C 传递函数依赖于 A。因此，满足第三范式的数据库表应该不存在如下依赖关系： 关键字段 → 非关键字段 x → 非关键字段 y

上面的文字我们肯定是看不懂的，也不愿意看下去的。接下来我就总结一下：

首先要明确的是：**满足着第三范式，那么就一定满足第二范式、满足着第二范式就一定满足第一范式**

第一范式：**字段是最小的的单元不可再分**

学生信息组成学生信息表，有年龄、性别、学号等信息组成。这些字段都不可再分，所以它是满足第一范式的

第二范式：**满足第一范式,表中的字段必须完全依赖于全部主键而非部分主键。**

其他字段组成的这行记录和主键表示的是同一个东西，而主键是唯一的，它们只需要依赖于主键，也就成了唯一的

学号为 1024 的同学，姓名为 Java3y，年龄是 22 岁。姓名和年龄字段都依赖着学号主键。

第三范式：满足第二范式，**非主键外的所有字段必须互不依赖**

就是数据只在一个地方存储，不重复出现在多张表中，可以认为就是消除传递依赖

比如，我们大学分了很多系（中文系、英语系、计算机系……），这个系别管理表信息有以下字段组成：系编号，系主任，系简介，系架构。那我们能不能在学生信息表添加系编号，系主任，系简介，系架构字段呢？不行的，因为这样就冗余了，非主键外的字段形成了依赖关系(依赖到学生信息表了)！正确的做法是：学生表就只能增加一个系编号字段。

2. 什么是事务？

事务简单来说：**一个 Session 中所进行所有的操作，要么同时成功，要么同时失败**

ACID — 数据库事务正确执行的四个基本要素

包含：原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)、持久性(Durability)。

一个支持事务（Transaction）中的数据库系统，必需要具有这四种特性，否则在事务过程（Transaction processing）当中无法保证数据的正确性，交易过程极可能达不到交易。

举个例子:A 向 B 转账，转账这个流程中如果出现问题，事务可以让数据恢复成原来一样【A 账户的钱没变，B 账户的钱也没变】。

事例说明：

```
        /*
        * 我们来模拟 A 向 B 账号转账的场景
        *    A 和 B 账户都有 1000 块，现在我让 A 账户向 B 账号转 500 块钱
        *
        * */
            //JDBC 默认的情况下是关闭事务的，下面我们看看关闭事务去操作转账操作有什么问题

            //A 账户减去 500 块
            String sql = "UPDATE a SET money=money-500 ";
            preparedStatement = connection.prepareStatement(sql);
            preparedStatement.executeUpdate();

            //B 账户多了 500 块
            String sql2 = "UPDATE b SET money=money+500";
            preparedStatement = connection.prepareStatement(sql2);
            preparedStatement.executeUpdate();
```

从上面看，我们的确可以发现 **A 向 B 转账，成功了。可是如果 A 向 B 转账的过程中出现了问题呢？** 下面模拟一下

```
//A 账户减去 500 块
String sql = "UPDATE a SET money=money-500 ";
preparedStatement = connection.prepareStatement(sql);
preparedStatement.executeUpdate();

//这里模拟出现问题
int a = 3 / 0;



String sql2 = "UPDATE b SET money=money+500";
preparedStatement = connection.prepareStatement(sql2);
preparedStatement.executeUpdate();
```

显然，上面**代码是会抛出异常的**，我们再来查询一下数据。**A 账户少了 500 块钱，B 账户的钱没有增加。这明显是不合理的。**

我们可以通过事务来解决上面出现的问题

```
//开启事务,对数据的操作就不会立即生效。
connection.setAutoCommit(false);
```

```
            //A 账户减去 500 块
            String sql = "UPDATE a SET money=money-500 ";
            preparedStatement = connection.prepareStatement(sql);
            preparedStatement.executeUpdate();

            //在转账过程中出现问题
            int a = 3 / 0;

            //B 账户多 500 块
            String sql2 = "UPDATE b SET money=money+500";
            preparedStatement = connection.prepareStatement(sql2);
            preparedStatement.executeUpdate();

            //如果程序能执行到这里，没有抛出异常，我们就提交数据
            connection.commit();

            //关闭事务【自动提交】
            connection.setAutoCommit(true);


        } catch (SQLException e) {
            try {
```

```
                //如果出现了异常，就会进到这里来，我们就把事务回滚【将数据变成原来那样】
                connection.rollback();

                //关闭事务【自动提交】
                connection.setAutoCommit(true);
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
```

上面的程序也一样抛出了异常，A 账户钱没有减少，B 账户的钱也没有增加。

注意：当 Connection 遇到一个未处理的 SQLException 时，系统会非正常退出，事务也会自动回滚，但**如果程序捕获到了异常，是需要在 catch 中显式回滚事务的。**

3. 事务隔离级别

数据库定义了 4 个隔离级别：

Serializable【可避免脏读，不可重复读，虚读】

Repeatable read【可避免脏读，不可重复读】

Read committed【可避免脏读】

Read uncommitted【级别最低，什么都避免不了】

分别对应 Connection 类中的 4 个常量

TRANSACTION_READ_UNCOMMITTED

TRANSACTION_READ_COMMITTED

TRANSACTION_REPEATABLE_READ

TRANSACTION_SERIALIZABLE

脏读：**一个事务读取到另外一个事务未提交的数据**

例子：**A 向 B 转账，A 执行了转账语句，但 A 还没有提交事务，B 读取数据，发现自己账户钱变多了！**B 跟 A 说，我已经收到钱了。A 回滚事务【rollback】，等 B 再查看账户的钱时，发现钱并没有多。

不可重复读：**一个事务读取到另外一个事务已经提交的数据，也就是说一个事务可以看到其他事务所做的修改**

注：**A 查询数据库得到数据，B 去修改数据库的数据，导致 A 多次查询数据库的结果都不一样【危害：A 每次查询的结果都是受 B 的影响的，那么 A 查询出来的信息就没有意思了】**

虚读(幻读)：**是指在一个事务内读取到了别的事务插入的数据，导致前后读取不一致。**

注：**和不可重复读类似，但虚读(幻读)会读到其他事务的插入的数据，导致前后读取不一致**

简单总结：**脏读是不可容忍的，不可重复读和虚读在一定的情况下是可以的【做统计的肯定就不行】。**

4. 数据库的乐观锁和悲观锁是什么？

确保在多个事务同时存取数据库中同一数据时不破坏事务的隔离性和统一性以及数据库的统一性，**乐观锁和悲观锁是并发控制主要采用的技术手段。**

悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作

在查询完数据的时候就把事务锁起来，直到提交事务

实现方式：使用数据库中的锁机制

乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。

在修改数据的时候把事务锁起来，通过 version 的方式来进行锁定

实现方式：使用 version 版本或者时间戳

悲观锁：

1.悲观锁

select * from eb_sku where sku_id = 1001 for udpate

update eb_sku set stock = 100 - #{quantity} where sku_id = 1001

缺点：性能低

乐观锁：

1.乐观锁

	sku_id	stock	version
a(查询的数据)	1001	100	1
	1001	98	2
b(查询的数据)	1001	100	1

```
select * from eb_sku where sku_id  = 1001

update eb_sku set stock = 100 - #{quantity},  version = #{version} +1 where sku_id = #{skuId} and
version = #{version} and stock >= #{quantity}
```

可以参考：

http://www.open-open.com/lib/view/open1452046967245.html

5. 超键、候选键、主键、外键分别是什么？

超键：**在关系中能唯一标识元组的属性集称为关系模式的超键。**一个属性可以为作为一个超键，多个属性组合在一起也可以作为一个超键。**超键包含候选键和主键。**

候选键(候选码)：是最小超键，即没有冗余元素的超键。

主键(主码)：数据库表中对储存数据对象予以唯一和完整标识的数据列或属性的组合。一个数据列只能有一个主键，且主键的取值不能缺失，即不能为空值（Null）。

外键：在一个表中存在的另一个表的主键称此表的外键。

候选码和主码：

例子：邮寄地址（城市名，街道名，邮政编码，单位名，收件人）

它有两个候选键:{城市名，街道名} 和 {街道名，邮政编码}

如果我选取{城市名，街道名}作为唯一标识实体的属性，那么{城市名，街道名} 就是主码(主键)

6. SQL 约束有哪几种？

NOT NULL: 用于控制字段的内容一定不能为空（NULL）。

UNIQUE: 控件字段内容不能重复，一个表允许有多个 Unique 约束。

PRIMARY KEY: 也是用于控件字段内容不能重复，但它在一个表只允许出现一个。

FOREIGN KEY: 用于预防破坏表之间连接的动作，也能防止非法数据插入外键列，因为它必须是它指向的那个表中的值之一。

CHECK: 用于控制字段的值范围。

7. drop、delete 与 truncate 分别在什么场景之下使用？

我们来对比一下他们的区别：

drop table

1)属于 DDL

2)不可回滚

3)不可带 where

4)表内容和结构删除

5)删除速度快

truncate table

1)属于 DDL

2)不可回滚

3)不可带 where

4)表内容删除

5)删除速度快

delete from

1)属于 DML

2)可回滚

3)可带 where

4)表结构在，表内容要看 where 执行的情况

5)删除速度慢,需要逐行删除

不再需要一张表的时候，用 drop

想删除部分数据行时候，用 delete，并且带上 where 子句

保留表而删除所有数据的时候用 truncate

8. 索引特点

索引的特点

(1) 索引一旦建立, Oracle 管理系统会对其进行自动维护，而且由 O**racle 管理系统决定何时使用索引**

(2) 用户不用在查询语句中指定使用哪个索引

(3) **在定义 primary key 或 unique 约束后系统自动在相应的列上创建索引**

(4) 用户也能按自己的需求，对指定单个字段或多个字段，添加索引

需要注意的是：**Oracle 是自动帮我们管理索引的，并且如果我们指定了 primary key 或者 unique 约束，系统会自动在对应的列上创建索引..**

什么时候【要】创建索引

(1) 表经常进行 SELECT 操作

(2) 表很大(记录超多)，记录内容分布范围很广

(3) 列名经常在 WHERE 子句或连接条件中出现

什么时候【不要】创建索引

(1) 表经常进行 INSERT/UPDATE/DELETE 操作

(2) 表很小(记录超少)

(3) 列名不经常作为连接条件或出现在 WHERE 子句中

索引优缺点：

索引加快数据库的检索速度

索引降低了插入、删除、修改等维护任务的速度(虽然索引可以提高查询速度，但是它们也会导致数据库系统更新数据的性能下降，**因为大部分数据更新需要同时更新索引)**

唯一索引可以确保每一行数据的唯一性，通过使用索引，可以在查询的过程中使用优化隐藏器，提高系统的性能

索引需要占物理和数据空间

索引分类：

唯一索引：唯一索引不允许两行具有相同的索引值

主键索引：为表定义一个主键将自动创建主键索引，主键索引是唯一索引的特殊类型。主键索引要求主键中的每个值是唯一的，并且不能为空

聚集索引(Clustered)：表中各行的物理顺序与键值的逻辑（索引）顺序相同，每个表只能有一个

非聚集索引(Non-clustered)：非聚集索引指定表的逻辑顺序。数据存储在一个位置，索引存储在另一个位置，索引中包含指向数据存储位置的指针。可以有多个，小于 249 个

9. 非关系型数据库和关系型数据库区别，优势比较?

非关系型数据库的优势：

性能：NOSQL 是基于键值对的，可以想象成表中的主键和值的对应关系，而且不需要经过 SQL 层的解析，所以性能非常高。

可扩展性：同样也是因为基于键值对，数据之间没有耦合性，所以非常容易水平扩展。

关系型数据库的优势：

复杂查询：可以用 SQL 语句方便的在一个表以及多个表之间做非常复杂的数据查询。

事务支持：使得对于安全性能很高的数据访问要求得以实现。

其他：

1.对于这两类数据库，对方的优势就是自己的弱势，反之亦然。

2.NOSQL 数据库慢慢开始具备 SQL 数据库的一些复杂查询功能，比如 MongoDB。

3.对于事务的支持也可以用一些系统级的原子操作来实现例如乐观锁之类的方法来曲线救国，比如 Redis set nx。

10. MYSQL 的两种存储引擎区别（事务、锁级别等等），各自的适用场景

引擎	特性
MYISAM	不支持外键，表锁，插入数据时，锁定整个表，查表总行数时，不需要全表扫描
INNODB	支持外键，行锁，查表总行数时，全表扫描

11. 索引有 B+索引和 hash 索引

索引	区别
Hash	hash索引，等值查询效率高，不能排序,不能进行范围查询
B+	数据有序,范围查询

12 为什么设计红黑树

红黑树通过它规则的设定，确保了插入和删除的最坏的时间复杂度是 O(log N) 。

红黑树解决了 AVL 平衡二叉树的维护起来比较麻烦的问题，红黑树，读取略逊于 AVL，维护强于 AVL，每次插入和删除的平均旋转次数应该是远小于平衡树。

因此：

相对于要求严格的 AVL 树来说，红黑树的旋转次数少，所以对于插入、删除操作较多的情况下，我们就用红黑树。但是，只是对查找要求较高,那么 AVL 还是较优于红黑树.

13 B 树的作用

B 树大多用在磁盘上用于查找磁盘的地址。因为磁盘会有大量的数据，有可能没有办法一次将需要的所有数据加入到内存中，所以只能逐一加载磁盘页，每个磁盘页就对应一个节点，而对于 B 树来说，B 树很好的将树的高度降低了，这样就会减少 IO 查询次数，虽然一次加载到内存的数据变多了，但速度绝对快于 AVL 或是红黑树的。

14 B 树和 B+树的区别

B/B+树用在磁盘文件组织、数据索引和数据库索引中。其中 B+树比 B 树更适合实际应用中操作系统的文件索引和数据库索引，因为：

1、B+树的磁盘读写代价更低

B+树的内部结点并没有指向关键字具体信息的指针。因此其内部结点相对 B 树更小。如果把所有同一内部结点的关键字存放在同一盘块中，那么盘块所能容纳的关键字数量也越多。一次性读入内存中的需要查找的关键字也就越多。相对来说 IO 读写次数也就降低了。

举个例子，假设磁盘中的一个盘块容纳 16bytes，而一个关键字 2bytes，一个关键字具体信息指针 2bytes。一棵 9 阶 B-tree(一个结点最多 8 个关键字)的内部结点需要 2 个盘快。而 B+ 树内部结点只需要 1 个盘快。当需要把内部结点读入内存中的时候，B 树就比 B+ 树多一次盘块查找时间(在磁盘中就是盘片旋转的时间)。

2、B+-tree 的查询效率更加稳定

由于非终结点并不是最终指向文件内容的结点，而只是叶子结点中关键字的索引。所以任何关键字的查找必须走一条从根结点到叶子结点的路。所有关键字查询的路径长度相同，导致每一个数据的查询效率相当。

3、B 树在元素遍历的时候效率较低

由于 B+树的数据都存储在叶子结点中，分支结点均为索引，方便扫库，只需要扫一遍叶子结点即可，但是 B 树因为其分支结点同样存储着数据，我们要找到具体的数据，需要进行一次中序遍历按序来扫，所以 B+树更加适合在区间查询的情况，所以通常 B+树用于数据库索引。在数据库中基于范围的查询相对频繁，所以此时 B+树优于 B 树。

15 B 树和红黑树的区别

最大的区别就是树的深度较高，在磁盘 I/O 方面的表现不如 B 树。

要获取磁盘上数据，必须先通过磁盘移动臂移动到数据所在的柱面，然后找到指定盘面，接

着旋转盘面找到数据所在的磁道，最后对数据进行读写。磁盘 IO 代价主要花费在查找所需的柱面上，树的深度过大会造成磁盘 IO 频繁读写。根据磁盘查找存取的次数往往由树的高度所决定。

所以，在大规模数据存储的时候，红黑树往往出现由于树的深度过大而造成磁盘 IO 读写过于频繁，进而导致效率低下。在这方面，B 树表现相对优异，B 树可以有多个子女，从几十到上千，可以降低树的高度。

16 AVL 树和红黑树的区别

红黑树的算法时间复杂度和 AVL 相同，但统计性能比 AVL 树更高。

1、红黑树和 AVL 树都能够以 O(log2 n)的时间复杂度进行搜索、插入、删除操作。

2、由于设计，红黑树的任何不平衡都会在三次旋转之内解决。AVL 树增加和删除可能需要通过一次或多次树旋转来重新平衡这个树。

在查找方面：

红黑树的性质(最长路径长度不超过最短路径长度的 2 倍)，其查找代价基本维持在 O(logN)左右，但在最差情况下(最长路径是最短路径的 2 倍少 1)，比 AVL 要略逊色一点。

AVL 是严格平衡的二叉查找树（平衡因子不超过 1）。查找过程中不会出现最差情况的单支树。因此查找效率最好，最坏情况都是 O(logN)数量级的。

所以，综上：

AVL 比 RBtree 更加平衡，但是 AVL 的插入和删除会带来大量的旋转。 所以如果插入

和删除比较多的情况，应该使用 RBtree，如果查询操作比较多，应该使用 AVL。

AVL 是一种高度平衡的二叉树，维护这种高度平衡所付出的代价比从中获得的效率收益还大，故而实际的应用不多，更多的地方是用追求局部而不是非常严格整体平衡的红黑树。当然，如果场景中对插入删除不频繁，只是对查找特别有要求，AVL 还是优于红黑的。

17 数据库为什么使用 B 树，而不使用 AVL 或者红黑树

我们假设 B+树一个节点可以有 100 个关键字，那么 3 层的 B 树可以容纳大概 1000000 多个关键字（100+101*100+101*101*100）。而红黑树要存储这么多至少要 20 层。所以使用 B 树相对于红黑树和 AVL 可以减少 IO 操作

18 mysql 的 Innodb 引擎为什么采用的是 B+树的索引方式

B+树只有叶子节点存放数据，而其他节点只存放索引，而 B 树每个节点都有 Data 域。所以相同大小的节点 B+树包含的索引比 B 树的索引更多（因为 B 树每个节点还有 Data 域）还有就是 B+树的叶子节点是通过链表连接的，所以找到下限后能很快进行区间查询，比 B 树中序遍历快

19 红黑树 和 b+树的用途有什么区别？

红黑树多用在内部排序，即全放在内存中的，STL 的 map 和 set 的内部实现就是红黑树。

B+树多用于外存上时，B+也被成为一个磁盘友好的数据结构。

20 为什么 B+树比 B 树更为友好

磁盘读写代价更低

树的非叶子结点里面没有数据，这样索引比较小，可以放在一个 blcok（或者尽可能少的 blcok）里面。避免了树形结构不断的向下查找，然后磁盘不停的寻道，读数据。这样的设计，可以降低 io 的次数。

查询效率更加稳定

非终结点并不是最终指向文件内容的结点，而只是叶子结点中关键字的索引。所以任何关键字的查找必须走一条从根结点到叶子结点的路。所有关键字查询的路径长度相同，导致每一个数据的查询效率相当。

遍历所有的数据更方便

B+树只要遍历叶子节点就可以实现整棵树的遍历，而其他的树形结构 要中序遍历才可以访问所有的数据。

21. 数据库优化

在我们书写 SQL 语句的时候，其实书写的顺序、策略会影响到 SQL 的性能，虽然实现的功

能是一样的，但是它们的性能会有些许差别。

因此，下面就讲解在书写 SQL 的时候，怎么写比较好。

①选择最有效率的表名顺序

数据库的解析器**按照从右到左的顺序处理 FROM 子句中的表名，FROM 子句中写在最后的表将被最先处理**

在 FROM 子句中包含多个表的情况下：

如果三个表是完全无关系的话，将记录和列名最少的表，写在最后，然后依次类推

也就是说：选择记录条数最少的表放在最后

如果有 3 个以上的表连接查询：

如果三个表是有关系的话，将引用最多的表，放在最后，然后依次类推。

也就是说：被其他表所引用的表放在最后

例如：查询员工的编号，姓名，工资，工资等级，部门名

emp 表被引用得最多，记录数也是最多，因此放在 form 字句的最后面

```
select emp.empno,emp.ename,emp.sal,salgrade.grade,dept.dname
from salgrade,dept,emp
where (emp.deptno = dept.deptno) and (emp.sal between salgrade.losal and salgrade.hisal)
```

②WHERE 子句中的连接顺序

数据库采用**自右而左的顺序解析** WHERE 子句,根据这个原理,**表之间的连接必须写在其他 WHERE 条件之左,那些可以过滤掉最大数量记录的条件必须写在 WHERE 子句的之右。**

emp.sal 可以过滤多条记录，写在 WHERE 字句的最右边

```
select emp.empno,emp.ename,emp.sal,dept.dname
from dept,emp
where (emp.deptno = dept.deptno) and (emp.sal > 1500)
```

③SELECT 子句中避免使用*号

我们当时学习的时候，“*”号是可以获取表中全部的字段数据的。

但是它要通过查询数据字典完成的，这意味着将耗费更多的时间

使用*号写出来的 SQL 语句也不够直观。

④用 TRUNCATE 替代 DELETE

这里仅仅是：**删除表的全部记录，除了表结构才这样做。**

DELETE 是一条一条记录的删除，而 Truncate 是将整个表删除，保留表结构，这样比 DELETE 快

⑤多使用内部函数提高 SQL 效率

例如使用 mysql 的 concat()函数会比使用||来进行拼接快，因为 concat()函数已经被 mysql 优化过了。

⑥使用表或列的别名

如果表或列的名称太长了，使用一些简短的别名也能稍微提高一些 SQL 的性能。毕竟要扫描的字符长度就变少了。。。

⑦多使用 commit

comiit 会释放回滚点...

⑧善用索引

索引就是为了提高我们的查询数据的，当表的记录量非常大的时候，我们就可以使用索引了。

⑨SQL 写大写

我们在编写 SQL 的时候，官方推荐的是使用大写来写关键字，**因为 Oracle 服务器总是先将小写字母转成大写后，才执行**

⑩避免在索引列上使用 NOT

因为 Oracle 服务器遇到 NOT 后，他就会停止目前的工作，转而执行全表扫描

①①避免在索引列上使用计算

WHERE 子句中，**如果索引列是函数的一部分，优化器将不使用索引而使用全表扫描，这样会变得变慢**

①②用 >= 替代 >

```
低效:

SELECT * FROM EMP WHERE DEPTNO > 3

首先定位到 DEPTNO=3 的记录并且扫描到第一个 DEPT 大于 3 的记录

高效:

SELECT * FROM EMP WHERE DEPTNO >= 4
```

直接跳到第一个 DEPT 等于 4 的记录

①③用 IN 替代 OR

```
select * from emp where sal = 1500 or sal = 3000 or sal = 800;

select * from emp where sal in (1500,3000,800);
```

①④总是使用索引的第一个列

如果索引是建立在多个列上，只有在它的第一个列被 WHERE 子句引用时，优化器才会选择使用该索引。 当只引用索引的第二个列时，不引用索引的第一个列时，优化器使用了全表扫描而忽略了索引

```
create index emp_sal_job_idex

on emp(sal,job);

---------------------------------

select *

from emp

where job != 'SALES';
```

上边就不使用索引了。

数据库结构优化

1）范式优化： 比如消除冗余（节省空间。。）

2）反范式优化：比如适当加冗余等（减少 join）

3）拆分表： 垂直拆分和水平拆分

本文为 CSDN 博主「蜘蛛侠不会飞」的原创文章

原文链接：https://blog.csdn.net/qq_40587575/article/details/114488339