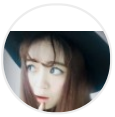
10道高频面试题整理~（持续更新）_笔试面经_牛客网

10道高频面试题整理~（持续更新）



牛妹

编辑于 2016-04-05 16:03:01

前言：给大家整理了一些面试题以及答案，但是不推荐去死记答案，只是一个参考，更多的是融入自己的理解，希望能对大家有所帮助！

1.TCP和**UDP**的差别：

参考：

TCP（Transmission Control Protocol，传输控制协议）是基于连接的协议，也就是说，在正式收发数据前，必须和对方建立可靠的连接。一个TCP连接必须要经过三次“对话”才能建立起来，其中的过程非常复杂，我们这里只做简单、形象的介绍，你只要做到能够理解这个过程即可。我们来看看这三次对话的简单过程：主机A向主机B发出连接请求数据包：“我想给你发数据，可以吗？”，这是第一次对话；主机B向主机A发送同意连接和要求同步（同步就是两台主机一个在发送，一个在接收，协调工作）的数据包：“可以，你什么时候发？”，这是第二次对话；主机A再发出一个数据包确认主机B的要求同步：“我现在就发，你接着吧！”，这是第三次对话。三次“对话”的目的是使数据包的发送和接收同步，经过三次“对话”之后，主机A才向主机B正式发送数据。

UDP（User Data Protocol，用户数据报协议）是与TCP相对应的协议。它是面向非连接的协议，它不与对方建立连接，而是直接就把数据包发送过去！

UDP适用于一次只传送少量数据、对可靠性要求不高的应用环境。比如，我们经常使用“ping”命令来测试两台主机之间TCP/IP通信是否正常，其实“ping”命令的原理就是向对方主机发送UDP数据包，然后对方主机确认收到数据包，如果数据包是否到达的消息及时反馈回来，那么网络就是通的。例如，在默认状态下，一次“ping”操作发送4个数据包（如图2所示）。大家可以看到，发送的数据包数量是4包，收到的也是4包（因为对方主机收到后会发回一个确认收到的数据包）。这充分说明了UDP协议是面向非连接的协议，没有建立连接的过程。正因为UDP协议没有连接的过程，所以它的通信效果高；但也正因为如此，它的可靠性不如TCP协议高。QQ就使用UDP发消息，因此有时会出现收不到消息的情况。

tcp协议和udp协议的差别

TCP UDP

是否连接 面向连接 面向非连接

传输可靠性 可靠 不可靠

应用场合 传输大量数据 少量数据

速度 慢 快

2.TreeMap、HashMap、LindedHashMap的区别

参考：

java为数据结构中的映射定义了一个接口java.util.Map;它有四个实现类,分别是HashMap Hashtable LinkedHashMap 和TreeMap

Map主要用于存储健值对，根据键得到值，因此不允许键重复(重复了覆盖了),但允许值重复。

Hashmap 是一个最常用的Map,它根据键的HashCode 值存储数据,根据键可以直接获取它的值，具有很快的访问速度，遍历时，取得数据的顺序是完全随机的。HashMap最多只允许一条记录的键为Null;允许多条记录的值为 Null;HashMap不支持线程的同步，即任一时刻可以有多个线程同时写HashMap;可能会导致数据的不一致。如果需要同步，可以用 Collections的synchronizedMap方法使HashMap具有同步的能力，或者使用ConcurrentHashMap。

Hashtable与 HashMap类似,它继承自Dictionary类，不同的是:它不允许记录的键或者值为空;它支持线程的同步，即任一时刻只有一个线程能写Hashtable,因此也导致了 Hashtable在写入时会比较慢。

LinkedHashMap保存了记录的插入顺序，在用Iterator遍历LinkedHashMap时，先得到的记录肯定是先插入的.也可以在构造时用带参数，按照应用次数排序。在遍历的时候会比HashMap慢，不过有种情况例外，当HashMap容量很大，实际数据较少时，遍历起来可能会比LinkedHashMap慢，因为LinkedHashMap的遍历速度只和实际数据有关，和容量无关，而HashMap的遍历速度和他的容量有关。

TreeMap实现SortMap接口，能够把它保存的记录根据键排序,默认是按键值的升序排序，也可以指定排序的比较器，当用Iterator 遍历TreeMap时，得到的记录是排过序的。

一般情况下，我们用的最多的是HashMap,HashMap里面存入的键值对在取出的时候是随机的,它根据键的HashCode值存储数据,根据键可以直接获取它的值，具有很快的访问速度。在Map 中插入、删除

和定位元素，HashMap 是最好的选择。

TreeMap取出来的是排序后的键值对。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。

LinkedHashMap 是HashMap的一个子类，如果需要输出的顺序和输入的相同,那么用LinkedHashMap可以实现,它还可以按读取顺序来排列，像连接池中可以应用。

3.HashMap和**ConcurrentHashMap**的区别，**HashMap**的底层源码。

参考：

有并发访问的时候用ConcurrentHashMap，效率比用锁的HashMap好 功能上可以，但是毕竟ConcurrentHashMap这种数据结构要复杂些，如果能保证只在单一线程下读写，不会发生并发的读写，那么就可以试用HashMap。ConcurrentHashMap读不加锁

HashMap类中的一些关键属性：

```
transient Entry[] table;//存储元素的实体数组
transient int size;//存放元素的个数
int threshold; //临界值  当实际大小超过临界值时，会进行扩容threshold = 加载因子*容量

final float loadFactor; //加载因子

transient int modCount;//被修改的次数
```

下面看看HashMap的几个构造方法：

```
1    public HashMap(int initialCapacity, float loadFactor) {
2        //确保数字合法
```

```
3        if (initialCapacity < 0)
4            throw new IllegalArgumentException("Illegal initial capacity: " +
5                                               initialCapacity);
6        if (initialCapacity > MAXIMUM_CAPACITY)
7            initialCapacity = MAXIMUM_CAPACITY;
8        if (loadFactor <= 0 || Float.isNaN(loadFactor))
9            throw new IllegalArgumentException("Illegal load factor: " +
10                                              loadFactor);
11
12       // Find a power of 2 >= initialCapacity
13       int capacity = 1;   //初始容量
14       while (capacity < initialCapacity)   //确保容量为2的n次幂，使capacity为大于initialCapacity的
最小的2的n次幂
15           capacity <<= 1;
16
17       this.loadFactor = loadFactor;
18       threshold = (int)(capacity * loadFactor);
19       table = new Entry[capacity];
20       init();
21   }
22
23   public HashMap(int initialCapacity) {
24       this(initialCapacity, DEFAULT_LOAD_FACTOR);
25   }
26
27   public HashMap() {
28       this.loadFactor = DEFAULT_LOAD_FACTOR;
29       threshold = (int)(DEFAULT_INITIAL_CAPACITY * DEFAULT_LOAD_FACTOR);
30       table = new Entry[DEFAULT_INITIAL_CAPACITY];
31       init();
32   }
```

4.Map、Set、List、Queue、Stack的特点与用法。

参考：

List：元素是有序的，元素可以重复。因为该集合体系有索引。

ArrayList：底层的数据结构使用的是数据结构。

查询速度很快。

增删稍慢。

线程不同步。

默认长度为10增长率为50%。

LinkedList：底层使用的链表数据结构。

增删速度很快。

查询稍慢。

Vector：底层是数组数据结构。1.0出现

线程同步

被ArrayList替代了。

长度增长率100%。

Set接口

Set接口是继承自Collection的，它不能包含有重复元素。Set中最多有一个null元素。

因为Set的这个制约，在使用Set集合的时候，应该注意：

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。

Set下有几个set类，HashSet、SortedSet、TreeSet，用的较多的是HashSet，其他两种基本不常用，以后慢慢补充该方面知识，下面说HashSet。

（1）HashSet，底层数据结构式哈希表，由哈希表支持，不保证集合的迭代顺序，特别是不保证该顺序恒久不变，此类允许使用null元素。HashSet保证元素唯一性的方法是通过元素的两个方法，hashCode和equals来完成。如果元素的HashCode值相同，才会判断equals是否为true。如果元素的hashCode值不同，不会调用equals。

（2）TreeSet：底层数据结构式二叉树。注：添加元素必须实现Comparable接口或在实例TreeSet时指定比较器。可以对Set集合中的元素进行排序。保证元素唯一性的依据：compareTo方法return 0；

Map接口

Map集成Collection接口，Map和Collection是两种不同的集合，Collection是值（value）的集合，Map是键值对（key,value）的集合。包含几种主要类和接口：HashMap、LinkedMap、WeakHashMap、SortedMap、TreeMap、HashTable等几种。

（1）Hashtable继承Map接口，实现一个key-value映射的哈希表。任何非空（non-null）的对象都可作为key或者value。添加数据使用put(key, value)，取出数据使用get(key)，这两个基本操作的时间开销为常数。

（2）WeakHashMap类，WeakHashMap是一种改进的HashMap，它对key实行“弱引用”，如果一个

key不再被外部所引用，那么该key可以被GC回收。

总结：

如果涉及到堆栈，队列等操作，应该考虑用List，对于需要快速插入，删除元素，应该使用LinkedList，如果需要快速随机访问元素，应该使用ArrayList。

如果程序在单线程环境中，或者访问仅仅在一个线程中进行，考虑非同步的类，其效率较高，如果多个线程可能同时操作一个类，应该使用同步的类。

在除需要排序时使用TreeSet,TreeMap外,都应使用HashSet,HashMap,因为他们 的效率更高。

要特别注意对哈希表的操作，作为key的对象要正确复写equals和hashCode方法。

容器类仅能持有对象引用（指向对象的指针），而不是将对象信息copy一份至数列某位置。一旦将对象置入容器内，便损失了该对象的型别信息。

尽量返回接口而非实际的类型，如返回List而非ArrayList，这样如果以后需要将ArrayList换成LinkedList时，客户端代码不用改变。这就是针对抽象编程。

注意：

1、Collection没有get()方法来取得某个元素。只能通过iterator()遍历元素。

2、Set和Collection拥有一模一样的接口。

3、List，可以通过get()方法来一次取出一个元素。使用数字来选择一堆对象中的一个，get(0)...。(add/get)

4、一般使用ArrayList。用LinkedList构造堆栈stack、队列queue。

5、Map用 put(k,v) / get(k)，还可以使用containsKey()/containsValue()来检查其中是否含有某个key/value。

HashMap会利用对象的hashCode来快速找到key。

6、Map中元素，可以将key序列、value序列单独抽取出来。

使用keySet()抽取key序列，将map中的所有keys生成一个Set。

使用values()抽取value序列，将map中的所有values生成一个Collection。

为什么一个生成Set，一个生成Collection？那是因为，key总是独一无二的，value允许重复。

java中stack的使用方法，堆栈是一种"后进先出"（LIFO） 的数据结构， 只能在一端进行插入（称为"压栈"） 或删除 （称为"出栈"）数据的操作,下面看示例吧

JAVA 中，使用 java.util.Stack 类的构造方法创建对象。

public class Stack extends vector

构造方法： public Stack() 创建一个空 Stack。

方法： 1. public push （item ） 把项 压入栈顶。其作用与 addElement (item) 相同。
参数 item 压入栈顶的项 。 返回： item 参数 ；
2. public pop () 移除栈顶对象，并作为函数的值 返回该对象。
返回：栈顶对象（Vector 对象的中的最后一项）。
抛出异常 ： EmptyStackException 如果堆栈式空的 。。。
3. public peek() 查看栈顶对象而不移除它。。
返回：栈顶对象（Vector 对象的中的最后一项）。
抛出异常 ： EmptyStackException 如果堆栈式空的 。。。
4. public boolean empty （测试堆栈是否为空。） 当且仅当堆栈中不含任何项时 返回 true，否则 返回 false.
5. public int search (object o) 返回对象在堆栈中位置， 以 1 为基数， 如果对象 o 是栈中的一项，该方法返回距离 栈顶最近的出现位置到栈顶的距离； 栈中最上端项的距离为 1 。使用equals 方法比较 o 与 堆栈中的项。

5.String、StringBuffer与StringBuilder的区别。

参考：java中String、StringBuffer、StringBuilder是编程中经常使用的字符串类，他们之间的区别也是经常在面试中会问到的问题。现在总结一下，看看他们的不同与相同。

1.可变与不可变

String类中使用字符数组保存字符串，如下就是，因为有“final”修饰符，所以可以知道string对象是不可变的。

private final char value[];

StringBuilder与StringBuffer都继承自AbstractStringBuilder类，在AbstractStringBuilder中也是使用字符数组保存字符串，如下就是，可知这两种对象都是可变的。

char[] value;

2.是否多线程安全

String中的对象是不可变的，也就可以理解为常量，显然线程安全。

AbstractStringBuilder是StringBuilder与StringBuffer的公共父类，定义了一些字符串的基本操作，如expandCapacity、append、insert、indexOf等公共方法。

StringBuffer对方法加了同步锁或者对调用的方法加了同步锁，所以是线程安全的。看如下源码：

复制代码

```
1 public synchronized StringBuffer reverse() {
2     super.reverse();
3     return this;
4 }
5
6 public int indexOf(String str) {
7     return indexOf(str, 0);       //存在 public synchronized int indexOf(String str, int fromIndex) 方法
8 }
```

复制代码

StringBuilder并没有对方法进行加同步锁，所以是非线程安全的。

3.StringBuilder与StringBuffer共同点

StringBuilder与StringBuffer有公共父类AbstractStringBuilder(抽象类)。

抽象类与接口的其中一个区别是：抽象类中可以定义一些子类的公共方法，子类只需要增加新的功能，不需要重复写已经存在的方法；而接口中只是对方法的申明和常量的定义。

StringBuilder、StringBuffer的方法都会调用AbstractStringBuilder中的公共方法，如super.append(...)。只是StringBuffer会在方法上加synchronized关键字，进行同步。

最后，如果程序不是多线程的，那么使用StringBuilder效率高于StringBuffer。

5.ArrayList、LinkedList、Vector的区别

> **作为一道非常经典的题目，这个没什么所谓的标准，大家心中一定要理解这个问题，很多东西不要死记硬背。**

参考：

LinkedList类

LinkedList实现了List接口，允许null元素。
此外LinkedList提供额外的get，remove，insert方法在LinkedList的首部或尾部。
LinkedList不是同步的（不是线程安全）。

实现线程安全：List list =
Collections.synchronizedList(new LinkedList(...));
增删快，查询慢。

ArrayList类
ArrayList实现了可变大小的数组。它允许null。
ArrayList没有同步。
增删慢，查询快。

Vector类
Vector线程安全。 效率低

6.问题：**Hashcode**的作用。

参考：官方大概这样定义的，hashcode方法返回该对象的哈希码值。支持该方法是为哈希表提供一些优点，例如，java.util.Hashtable 提供的哈希表。

hashCode 的常规协定是：
在 Java 应用程序执行期间，在同一对象上多次调用 hashCode 方法时，必须一致地返回相同的整数，前提是对象上 equals 比较中所用的信息没有被修改。从某一应用程序的一次执行到同一应用程序的另一次执行，该整数无需保持一致。

如果根据 equals(Object) 方法，两个对象是相等的，那么在两个对象中的每个对象上调用 hashCode 方法都必须生成相同的整数结果。
以下情况不 是必需的：如果根据 equals(java.lang.Object) 方法，两个对象不相等，那么在两个对象中

的任一对象上调用 hashCode 方法必定会生成不同的整数结果。但是，程序员应该知道，为不相等的对象生成不同整数结果可以提高哈希表的性能。

实际上，由 Object 类定义的 hashCode 方法确实会针对不同的对象返回不同的整数。（这一般是通过将该对象的内部地址转换成一个整数来实现的，但是 JavaTM 编程语言不需要这种实现技巧。）

当equals方法被重写时，通常有必要重写 hashCode 方法，以维护 hashCode 方法的常规协定，该协定声明相等对象必须具有相等的哈希码。

7.问题：**Java**的四种引用，强弱软虚，用到的场景。

参考：

java语言提供了4种引用类型：强引用、软引用(SoftReference)、弱引用（WeakReference）和幽灵引用（PhantomReference）.

1、强引用

强引用不会被GC回收，并且在java.lang.ref里也没有实际的对应类型，平时工作接触的最多的就是强引用。

Object obj = new Object();这里的obj引用便是一个强引用。如果一个对象具有强引用，那就类似于必不可少的生活用品，垃圾回收器绝不会回收它。当内存空 间不足，Java虚拟机宁愿抛出OutOfMemoryError错误，使程序异常终止，也不会靠随意回收具有强引用的对象来解决内存不足问题。

2、软引用

如果一个对象只具有软引用，那就类似于可有可物的生活用品。如果内存空间足够，垃圾回收器就不会回收它，如果内存空间不足了，就会回收这些对象的内存。只 要垃圾回收器没有回收它，该对象就可以被程序使用。软引用可用来实现内存敏感的高速缓存。 软引用可以和一个引用队列（ReferenceQueue）联合使用，如果软引用所引用的对象被垃圾回收，Java虚拟机就会把这个软引用加入到与之关联的引用队列中。

/** * 只有当内存不够的时候，才回收这类内存，因此在内存足够的时候，它们通常不被回收 * *

* 无论是否发送GC,执行结果都是: * java.lang.Object@f9f9d8 * null * java.lang.Object@f9f9d8 * null *

* * 可以看到:只有发送了GC,将对于从内存中释放的时候,JVM才会将reference假如引用队列 */ public

static void soft() throws Exception { Object obj = new Object(); ReferenceQueue refQueue = new ReferenceQueue(); SoftReference softRef = new SoftReference(obj, refQueue); System.out.println(softRef.get()); // java.lang.Object@f9f9d8 System.out.println(refQueue.poll());// null // 清除强引用,触发GC obj = null; System.gc(); System.out.println(softRef.get()); Thread.sleep(200); System.out.println(refQueue.poll()); }

这里有几点需要说明：

1、System.gc()告诉JVM这是一个执行GC的好时机，但具体执不执行由JVM决定（事实上这段代码一般都会执行GC)
2、Thread.sleep(200); 这是因为从对象被回收到JVM将引用加入refQueue队列，需要一定的时间。而且poll并不是一个阻塞方法，如果没有数据会返回null，所以我们选择等待一段时间。

3、弱引用
如果一个对象只具有弱引用，那就类似于可有可物的生活用品。弱引用与软引用的区别在于：只具有弱引用的对象拥有更短暂的生命周期。在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回收它的内存。不过，由于垃圾回收器是一个优先级很低的线程， 因此不一定会很快发现那些只具有弱引用的对象。 弱引用可以和一个引用队列（ReferenceQueue）联合使用，如果弱引用所引用的对象被垃圾回 收，Java虚拟机就会把这个弱引用加入到与之关联的引用队列中。

/** * 弱引用: 当发生GC的时候,Weak引用对象总是会内回收回收。因此Weak引用对象会更容易、更快被GC回收。 * Weak引用对象常常用于Map数据结构中，引用占用内存空间较大的对象 * *

* 如果不发生垃圾回收： * java.lang.Object@f9f9d8 * null * java.lang.Object@f9f9d8 * null * * 如果发生垃圾回收: * java.lang.Object@f9f9d8 * null * null * java.lang.ref.WeakReference@422ede * *

*/ public static void weak() throws Exception { Object obj = new Object(); ReferenceQueue refQueue = new ReferenceQueue(); WeakReference weakRef = new WeakReference(obj, refQueue); System.out.println(weakRef.get()); // java.lang.Object@f9f9d8 System.out.println(refQueue.poll());// null // 清除强引用,触发GC obj = null; System.gc(); System.out.println(weakRef.get()); // 这里特别注意:poll是非阻塞的,remove是阻塞的. // JVM将弱引用放入引用队列需要一定的时间,所以这里先睡眠一会儿 // System.out.println(refQueue.poll());// 这里有可能是null Thread.sleep(200); System.out.println(refQueue.poll()); // System.out.println(refQueue.poll());//这里一定是null,因为已经从队列中移除 // System.out.println(refQueue.remove()); }

这里需要注意下：

1、remove这是一个阻塞方法，类似于J.U.C并发包下的阻塞队列，如果没有队列没有数据，那么当前线程一直等待。
2、如果队列有数据，那么remove和pool都会将第一个元素出队。

4、幽灵引用(虚引用)
虚引用主要用来跟踪对象被垃圾回收器回收的活动。虚引用与软引用和弱引用的一个区别在于：虚引用必须和引用队列 （ReferenceQueue）联合使用。当垃圾回收器准备回收一个对象时，如果发现它还有虚引用，就会在回收对象的内存之前，把这个虚引用加入到与之关联的引用队列中。程序可以通过判断引用队列中是否已经加入了虚引用，来了解被引用的对象是否将要被垃圾回收。如果程序发现某个虚引用已经被加入到引用队列，那么就可以在所引用的对象的内存被回收之前采取必要的行动。由于Object.finalize()方法的不安全性、低效性，常常使用虚引用完成对象回收前的资源释放工作。

/** * 当GC一但发现了虚引用对象，将会将PhantomReference对象插入ReferenceQueue队列. * 而此时PhantomReference所指向的对象并没有被GC回收，而是要等到ReferenceQueue被你真正的处理后才会被回收. * *

这里特别需要注意：当JVM将虚引用插入到引用队列的时候，虚引用执行的对象内存还是存在的。但是PhantomReference并没有暴露API返回对象。所以如果我想做清理工作，需要继承PhantomReference类，以便访问它指向的对象。如NIO直接内存的自动回收，就使用到了sun.misc.Cleaner。

8.Object有哪些公用方法？

> 在面试的过程中经常会问到这个问题，有可能你答完所有的方法就可以了，也有可能还会突然就某一个方法让你进行讲解，所以都要掌握

参考答案：

Object是所有类的父类，任何类都默认继承Object。
clone
保护方法，实现对象的浅复制，只有实现了Cloneable接口才可以调用该方法，否则抛出CloneNotSupportedException异常
equals
在Object中与==是一样的，子类一般需要重写该方法
hashCode

该方法用于哈希查找，重写了equals方法一般都要重写hashCode方法。这个方法在一些具有哈希功能的Collection中用到

getClass

final方法，获得运行时类型

wait

使当前线程等待该对象的锁，当前线程必须是该对象的拥有者，也就是具有该对象的锁。wait()方法一直等待，直到获得锁或者被中断。wait(long timeout)设定一个超时间隔，如果在规定时间内没有获得锁就返回。

调用该方法后当前线程进入睡眠状态，直到以下事件发生：

1. 其他线程调用了该对象的notify方法
2. 其他线程调用了该对象的notifyAll方法
3. 其他线程调用了interrupt中断该线程
4. 时间间隔到了

此时该线程就可以被调度了，如果是被中断的话就抛出一个InterruptedException异常

notify

唤醒在该对象上等待的某个线程

notifyAll

唤醒在该对象上等待的所有线程

toString

转换成字符串，一般子类都有重写，否则打印句柄

9.Switch能否用**string**做参数？

参考答案：　在 Java 7之前，switch 只能支持 byte、short、char、int或者其对应的封装类以及 Enum 类型。在 Java 7中，String支持被加上了。

```
switch (ctrType) {
case "01" :
exceptionType = "读FC参数数据";
break;
case "03" :
exceptionType = "读FC保存的当前表计数据";
break;
default:
exceptionType = "未知控制码："+ctrType;
}
```

其中ctrType为字符串。

如在jdk 7 之前的版本使用, 会提示如下错误：

Cannot switch on a value of type String for source level below 1.7. Only convertible int values or enum variables are permitted

意为jdk版本太低，不支持。

10.java中九种基本数据类型的大小，以及他们的封装类。

参考答案

java中有八种基本数据类型，int ,double ,long ,float, short,byte,character,boolean;

和他们对应的封装类型是：Integer ,Double ,Long ,Float, Short,Byte,Character,Boolean;

数据类型	Bit---位	范围	数据类型	Bit---位	范围
boolean	1		Char（2字节）	16	
Byte（1字节）	8	-128-127	Short（2字节）	16	
Int（4字节）	32	-2的15次幂-2的15次幂-1	Long（8字节）	64	-2的63次幂-2的63次幂-1
Float（4字节）	32	-3.403E38-3.403E38	Double（8字节）	64	-1.798E308