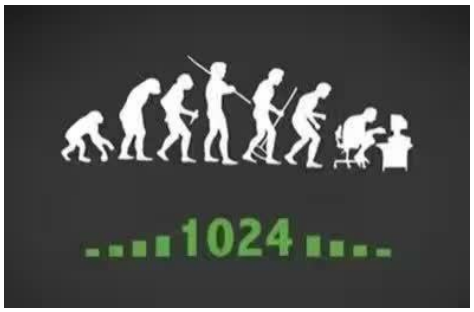
2016-10-24

Java 面试指南

【第一版续】



清风&明月

前言

话说 10.24 日是 2 的十次方，把这个特殊的日子定为程序节日，那在此祝广大码农、准码农们节日快乐！

本指南计划是作为第二个版本来写的，同时对第一个版本中的部分不足进行校正，但由于个人原因，今年浪费了不少时间，未能按计划进行，向诸位道歉。本版本中计划是收录互联面试 40 题、数据结构算法 20 题、设计模式 10 题；由于年底跳槽季，很多公司好的职位都会在年底招人，为不耽误大家使用，把版本定为“第一版续”发给大家，大家结合第一版使用。

本版本缺失的内容，大家可以闲暇时自己查找复习，互联网面试这块主要缺少一些框架原理分析和并发相关的知识点，比如“NIO、Netty、Zookeeper、Dubbo、阻塞队列、数据连接池原理”等；数据结构与算法这块主要缺常见算法（还有一些基于非内存排序的算法）“希尔排序、二叉树排序、堆排序、红黑树、大数据量检索等”；设计模式这块（观察者模式问到很多、门面、责任链都很重要），本节的内容我没有画图，写的有点难懂；附件中已经发给大家一个很有趣的电子书，大家可以参考理解。

后面再更新版本将不再以知识点的形式更新，最近了解很多面试套路，尤其是 BAT 公司的面试套路，可以模仿一下形式，我一直坚信如果能拿下 BAT 其它公司何惧哉！

下面给大家看一下面试套路之一” 技术迭代追问”，这也是下一个版本的模式，把知识点串联起来。

面试官： 对 CoreJava 中的集合了解吗？
求职者： if(true){go on..}else{换下一话题}。
面试官： 能介绍一下 HashTable 与 HashMap 的区别吗？
求职者： if(true){balabala……}else{换下一话题}
面试官： 你刚才说 HashMap 是线程不安全的，为什么不安全，如果保证安全？
求职者： if(true){balabala…… ConcurrentHashMap}else{换下一话题}
面试官： 你提到 ConcurrentHashMap，那它的底层如何实现的，线程安全的机制是什么？
求职者： if(true){balabala……分段锁}else{换下一话题}
面试官： 这样设计的好处是什么？你有没有更好的实现？
求职者： 如果 bala 不下去了，结束，否则继续。

目录

第一部分　CoreJava 相关

第 01 章　互联网面试

1.1　String 为什么是 final

解析：String、StringBuffer、StringBuilder 之间的区别，可以参考版本 I 中的比较，这里主要针对最近出现比较多互联网面试中的，String 为什么是 final 的来做一下解析。首先被 final 修饰的 String 被称为不可变类，早期接触 String 时，是把它作为八种基本数据类型外的一种特殊的处理字符串的引用类型，也可以理解成一种工具类。不同的开发人员无论在任何时候调用 String 类给我们提供的方法，同样的输入都应该有相同的输出。但如果 String 是可变的，该类的方法就可能被重写，那么得到的结果未必相同，String 将失去作者设计的初衷。另外 String 被定义成 final 类型，不能派生在安全性上比较好，编译器对 fianl 修改的类进行了优化，使得 final 修改饰的性能较高（有兴趣的可以研究一下类的编译原理）。

参考答案：1、String 类是一个不可变类，被 final 修改的类不能被继承，这样提高了 String 类使用的安全性。

2、String 类的主要变量 value[]都被设计成 private final 的，这样在多线程时，对 String 对象的访问是可以保证安全。

3、JVM 对 final 修饰的类进行了编译优化，设计成 final，JVM 不用对相关方法在虚函数表中查询，直接定位到 String 类的相关方法调用，提高了执行效率。

1.2　Class.forName 和 ClassLoader 的区别

解析：这是一个群中面试的同学反馈的面试题，虽然不是一个难题，但算是一个偏题（小概率），Class.forName 初学者在写 JDBC 连接时一定都用到过，但 ClassLoader 大家却用的不多，他们之间的区别更容易忽略，这题可以作为记忆型的题，以后有机会写框架时 ClassLoader 用的会很多，也就容易理解了。

参考答案：Class.forName 和 ClassLoader 都是用来装载类的，对于类的装载一般为分三个阶段加载、链接、编译，它们装载类的方式是有区别。

首先看一下 Class.forName(..)，forName(..)方法有一个重载方法 forName(className, boolean,ClassLoader)，它有三个参数，第一个参数是类的包路径，第二个参数是 boolean 类型，为 true 地表示 Loading 时会进行初始化，第三个就是指定一个加载器；当你调用 class.forName(..)时，默认调用的是有三个参数的重载方法，第二个参数默认传入 true,第三个参数默认使用的是当前类加载时用的加载器。

ClassLoader.loadClass()也有一个重载方法，从源码中可以看出它默认调的是它的重载方法 loadClass(name, false)，当第二参数为 false 时，说明类加载时不会被链接。**这也是两者之间最大区别，前者在加载的时候已经初始化，后者在加载的时候还没有链接。**如果你需要在加载时初始化一些东西，就要用 Class.forName 了，比如我们常用的驱动加载，

实际上它的注册动作就是在加载时的一个静态块中完成的。所以它不能被 ClassLoader 加载代替。

1.3 Java 的引用类型有哪几种

解析：这些类都在 java.lang.ref 包下，这里主要是为了考察一下求职者对 JVM 结构的了解，以及对 GC 原理的深入理解；看似没有直接关联，但如不能了解 Java 对象的引用类型，你就很难理解 GC 回收的机制；往往间接考察比直接考察难度要大，因为这些问题都是平时容易忽略，这里带大家了解下概念，如果想深入掌握，大家可以自己多花点时间，不仅可以提高代码质量还能减少代码 OOM 的发生。

参考答案：Java 自从 JDK1.2 版本开始，引入四种引用的类型，它们由强到弱依次是：**强引用（StongReference）、软引用（SoftReference）、弱引用（WeakReference）、虚引用（PhantomReference）**，它们的各自特点及使用领域。

强引用：代码中最常用看到的 Object o = new Object();这里就是强引用，只要引用在，GC 就不回收它，如果 JVM 内存空间不足会抛出 OutOfMemoryError。

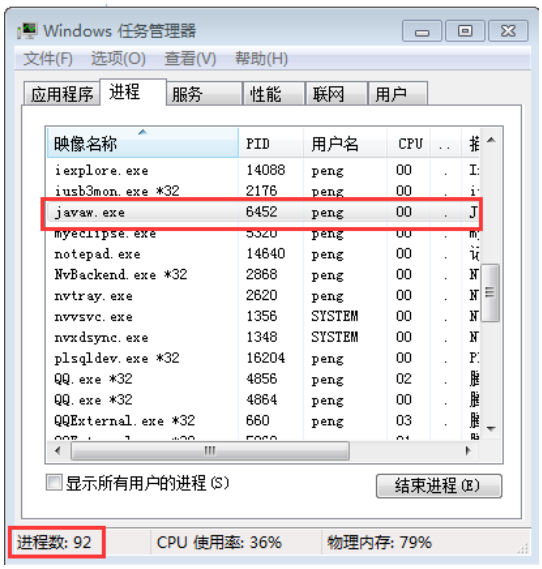
软引用：通常用描述一些有用但不是必需的对象，如果 JVM 内存不足时，会被 GC，通常被用来作为一些缓存模块的设计，而且不容易 OOM。

弱引用：比软引用还低级别的引用，软引用一般是内存不足时回收，而弱引用只要被 GC 扫描线程发现就会回收掉，即便是 JVM 内存还充足的情况下。

虚引用：如其名，虚无般的存在，完全不会影响对象的生命周期，如果一个对象仅持有虚引用，就如同没有引用一样，可能随时被回收掉，一般会与强用队列关联使用，一般只用于对象回收的事件传递。

1.4 进程和线程的区别

解析：这是一个纯概念的考察，被问到很多，这是一个同学去互联网面试回来反馈的题，而且答案也是由他提供的，这里给大家分享一下。



可以通过任务管理器看一下我们 PC 机正在运行的进程，上面 javaw.exe 就是大家最熟悉

的一个进程。

参考答案：定义：进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位。线程是进程的一个实体,是 CPU 调度和分派的基本单位,它是比进程更小的能独立运行的基本单位。

特点：1、一个进程可以拥有很多个线程，但每个线程只属于一个进程。

2、线程相对进程而言，划分尺度更小，并发性能更高。

3、进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。

4、线程必须依赖应用，在应用中调度，每个线程必须执行的入口、出口、执行序列，线程是不能够独立存在运行的。

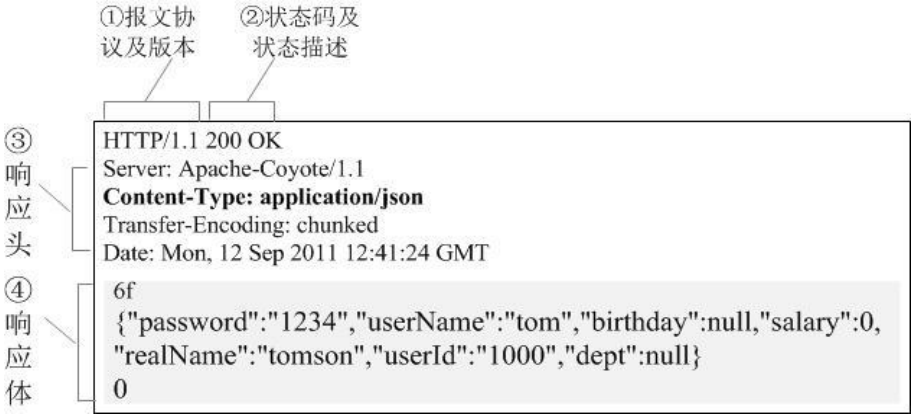
5、进程是资源分配的基本单位，线程是处理机调度的基本单位，所有的线程共享其所属进程的所有资源与代码。

6、多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。

1.5 Http 报文结构

解析：Http 协议经常会被问到，主要考察 http 协议，http 报文结构，http 常用的一些状态码理解等，难度都不算太高，但出现概率挺高。

（图片来源网络：请求报文）



(图片来源网络：响应报文)

上图为请求报文，包括“请求行①②③”、“请求头部④”、“请求包体⑤”，请求行①②③之间是用空格隔开的，面试回答时捡主要的说，没必要把所有的参数都介绍一下。

参考答案：请求报文包括**“请求行”、“请求头部”、“请求包体”**；请求行中主要包括：请求方式、请求地址、Http 版本，它们之间用空格分开。

请求头部主要包括： Accept: 告诉服务端客户端接受什么类型的响应；Accept-Language：客户端可接受的自然语言;User-Agent:请求端的浏览器以及服务器类型;Accept-Encoding：客户端可接受的编码压缩格式；Accept-Charset：可接受的应答的字符集;Host：请求的主机名，允许多个域名同处一个 IP 地址，即虚拟主机;connection：连接方式(close 或 keep-alive);Cookie：存储于客户端扩展字段，向同一域名的服务端发送属于该域的 cookie;空行：最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器以下不再有请求头;

请求包体：也是请求正文，业务报文。

响应报文包括**“状态行”、“响应头部”、“响应包体”**；状态行中包括：Http 协议版本、状态码以及状态码描述（常用状态码及描述，500：服务器内部错误，404：页面找不到，200 OK：表示请求成功返回，403：服务器收到请求但拒绝服务；其它的 1xx,2xx,3xx,4xx,5xx 系，大家可以网上查找一下，蛮重要，能说出来就行）。

响应头部，Server:响应服务器类型；Content-Type:响应数据的文档类型；Cache-Control：响应输出到客户端后，服务端通过该报文头属告诉客户端如何控制响应内容的缓存。

响应包体，真正的业务报文，也就是请求期望的返回数据。

PS：如何截图报文（以 IE 为例），打开 IE 的开发者模式（Fn+F12），然后点击网络、开始捕获 然后可以正常操作页面，获取到请求后，找到自己想观察的 URL 双击打开，可以看到上图中的名细。

1.6 Http 如何处理长连接

解析：这个问题不是很容易回答，回答这个问题必须是已经了解了 Http 协议及报文结构，到目前为止 http 主要定义了两个版本 http1.0 和 http1.1，现在大多数使用的都是 http1.1 版本，对于 http1.1 与 http1.0 的区别是，在 http1.1 中增加默认使用长连接，增加了 Host 请求头字段，提供了身份认证、状态管理和 Cache 缓存等机制相关的请求头和响应头。

参考答案： Http 目前有两个版本分别是 Http1.0 和 Http1.1，Http1.0 默认是短连接，如果需要长连接支持，需要加上 Connection: Keep-alive；Http1.1 的版本默认是支持长连接请求的方式，可以在抓取的请求中看到 Connection: Keep-alive，如果不想用长连接，需要在报文首部加上 Connection:close;对于默认的长连接可以通过 Keep-Alive:timeout=N 进行超时时间设置。

[追问]对长连接数据传输完成的识别：第一种：通过 Content-Length 指示的大小，如果传的报文长度达到了 Content-Length，则认为传输完成。

第二种：动态请求生成的文件中往往不包含 Content-Length，往往是通过分块传输入，服务器不能预先判断文件大小，这里要通过 Transfer-Encoding：chunked 模式来传输数据。Chunked 是按块进行数据传输的，这时候就要根据 chunked 编码来判断，chunked 编码的数据在最后有一个空 chunked 块，表明本次传输数据结束。

1.7 TCP 三次握手和四次挥手

解析：（下图源于网络），看到这个图画的很清晰，就直接拿来借用了，此图在学校网络

相关的课程中应该是都看到过的，TCP 协议在传统企业面试中，很少遇到；但在互联网公司面试中，基本上 80%的公司都会问到，互联网公司数据传输效率要求较高，从而对开发人员对底层协议的掌握要求也相对较高，很多互联网公司都会基于业务驱动开发一些适合自己业务的协议，如果要开发协议更要求对底层协议很清晰才行，所以如果你去互联网公司面试，请一定对此块内容好好掌握。

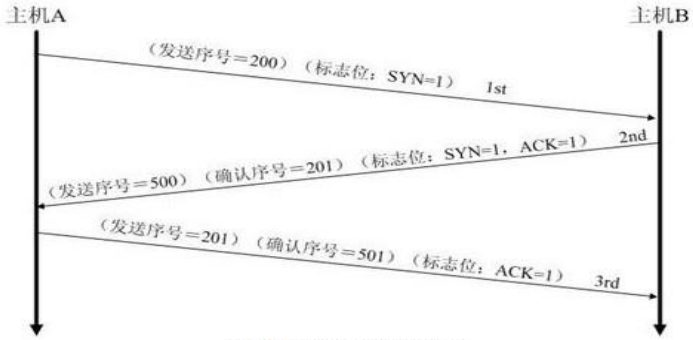


图1TCP三次握手**建立连接**

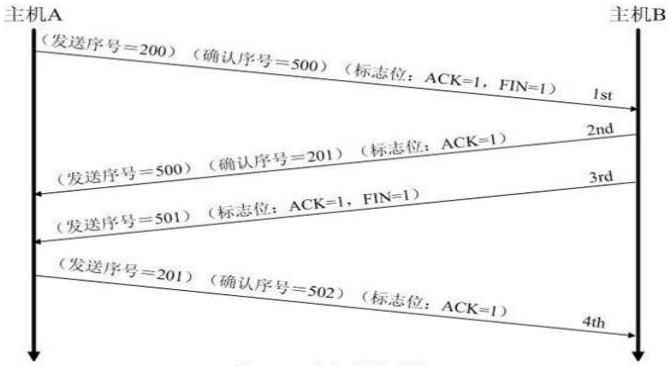


图2TCP四次挥手**关闭连接**

如果要看懂这个图，先来了解一下几个简单的概念：SYN 表示建立连接，FIN 表示关闭连接，ACK 表示响应，序号是随机产生的但作用很大，这里不详细说了，这几个关键字在面试的时候有必要先解释一下。

参考答案：TCP 建立连接和断开连接的操作有几个很重要的关键字，分别：**SYN 表示请求建立连接、ACK 表示响应、FIN 表示关闭连接请求、随机序列会随传送报文的字节数增加（SYN、FIN 都算位的，即便没有字节传送，序列也会增加）**。

TCP 建立连接的三次握手，**第一次握手：**主机 A 发送位码为 syn＝1,随机产生 seq =200 的数据包到服务器，主机 B 由 SYN=1 知道，A 要求建立联机；

第二次握手：主机 B 收到请求后要确认联机信息，向 A 发送 ack 确认序列=(主机 A 的 seq+1),syn=1,ack=1,随机产生 seq=500 的包；

第三次握手：主机 A 收到后检查 ack 确认序列是否正确，即第一次发送的 seq number+1,以及位码 ack 是否为 1，若正确，主机 A 会再发送 ack number=(主机 B 的 seq+1),ack=1，主机 B 收到后确认 seq 值与 ack=1 则连接建立成功。

【三次握手总结】主机 A 发 syn 给主机 B，主机 B 回 ack,syn，主机 A 回 ack，三次握手，连接成功。

四次挥手，**第一次挥手：**主机 A 发送一个 FIN，用来关闭客户 A 到服务器 B 的数据传送。

第二次挥手：主机 B 收到这个 FIN，它发回一个 ACK，确认序号为收到的序号加 1。和 SYN 一样，一个 FIN 将占用一个序号。

第三次挥手：主机 B 关闭与主机 A 的连接，发送一个 FIN 给主机 A。

第四次挥手：主机 A 发回 ACK 报文确认，并将确认序号设置为收到序号加 1。
【四次挥手总结】结束是两次 fin 的交互。主机 A 发 fin---主机 B 回 ack，主机 B 发 fin—主机 A 回 ack。

说明：这里仅仅只是针对 TCP 协议中的一个面试题来说的，对于 http,tcp 相关的协议，大家尽量多研究一些。

1.8 线程启动用 start 方法还是 run

解析：见版本一中的 5.14 线程的快速问答，这里作为补充说明，需要两部分结合看。

参考答案： 下面是 JDK 中 start()方法的源码，可以看到 start()调用的是 start0()，而 start0()是一个 native 方法，通过注释可以知道，它的作用主要是为线程分配系统资源的；而 run 只是一个普通的方法，所以线程的启动是通过 start 方法实现的。

```
    public synchronized void start() {
        if (threadStatus != 0 || this != me)
            throw new IllegalThreadStateException();
        group.add(this);
        start0();
        if (stopBeforeStart) {
        stop0(throwableFromStop);
    }}
    private native void start0();
```

1.9 多线程实现方式

解析：见版本一中的 5.14 线程的快速问答，这里作为补充说明，实现 Callable 接口，优点是可以获取线程执行的返回结果，下面给一个示例：

```
class CallableTest implements Callable<String> {
    @Override
    public String call() throws Exception {
        System.out.println("业务层:"Thread.currentThread().getName());
        return Thread.currentThread().getName();
    }
}
public class ThreadTest {
    public static void main(String[] args) {
        CallableTest ct1 = new CallableTest() ;
        CallableTest ct2 = new CallableTest() ;
        FutureTask<String> task1 = new FutureTask<String>(ct1);
        FutureTask<String> task2 = new FutureTask<String>(ct2);
        new Thread(task1).start() ;
```

```
        new Thread(task2).start() ;
        try {
            System.out.println(task1.get());   //获取返回参数
            System.out.println(task2.get());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

参考答案： 多线程实现的第四种方式是实现 Callable 接口，通常需要对需要获取线程处理结果的时候使用。

2.0 JDBC 连接步骤

解析：简单吧？一个朋友在负责公司初级程序员招聘时很喜欢问这个问题，他说可以过滤 60%的求职者；后来笔者参加一个高级开发的职位时，也被问到（当然多问了一些，后面会写出来），事后和面试人员聊天时，发现这个问题同样在高级开发的人员招聘时也能过滤 60%的求职者；惊叹之余也希望大家还是注意基础知识积累，Java 的 JDBC 连接可以说是每个人都曾经 N 次的写，虽然简单，但类似简单的知识点面试前都有必要梳理一下。这里如果能把每步操作说清楚最好，不能说清楚，就描述一个主要步骤。

参考答案： 1、注册驱动：Class.forName("…")；
2、获得连接:DiriverManager.getConnection(URL,Username,Passwored)；如果需要手动事务操作的，在这里需要关闭自动提交事务，开户手动事务处理；
3、创建执行 SQL 对象:Statement 或 PreparedStatement 对象；
4、执行语句:executeXxx 方法，如果需要手动事务操作的，记得提交事务；
5、处理返回结果；如果有异常记得回滚事务；
6、关闭资源：按打开的顺序倒序关闭法。
追问:Class.forName("…")做了哪些实现：Class.forName("")底层默认调了它的另外一个重载方法 forName(String name, boolean initialize,ClassLoader loader); initialize 这里默认传的值是 true，ClassLoader 指定时，默认用的是当前类加载时用的加载器；而 initialize 参数很重要，决定类在载入时是否连接，校验，初始化（这里主要是静态区有没有被初始化），Class.forName 在加载驱动类时，会在静态块中对服务进行注册，也就是真正的注册驱动。

2.1 线程常用的并发类及关键字

解析：前面已经说了很多和多线程有关的知识点了，但多线程的面试可以贯穿你编码职业的始终，经验职位级别不同，面试侧重点不同，之前我们提到的是必须要熟练掌握的；本

节的知识点更侧重于提高用的（如果是初级求职者，感觉到信息量太大可以以之前的为主，这个作为以后的备用复习）。常用的类 ReentrantLock、volatile、atomics、CountDownLatch、CyclicBarrier、Semaphore、FutureTask 等。

【在以后的更新中，这块依旧是重点】

参考答案： ReentrantLock 类位于 JDK 的并发包下，JDK5 后引入的（jdk5 之前的 sysnchronized 性能很差的，在版本 5 后做了很大的优化），用法语义和 sysnchronized 都很像，需要手动的在 finally 进行释放锁，这块详细可以参考版本一中的 4.8 节。

Volatile：并发类中的一个关键字，主要用来保证各个线程间的可见性（并不保证原子性）；当定义一个全局变量，它是在主存中的，各个线程读取的只是它的副本（可以认为是缓存），当某个线程修改副本时，其它线程不可见，就会导致数据的不一致性；Volatile 修饰变量后，当各个线程试图修改副本中的数据时，会先确定和主存中的数据是一致的，同时修改后会刷新到主存中，这样其它线程就可见了。

Atomics：这是一个包 java.util.concurrent.atomic，该包下主要提供了一下无锁的原子类操作，有点像 volatile 的作用。包里一共有 12 个类，四种原子更新方式，分别是原子更新基本类型，原子更新数组，原子更新引用和原子更新字段。Atomic 包里的类基本都是使用 Unsafe 实现的包装类。

CountDownLatch：是一个同步工具类，它允许一个或多个线程一直等待，直到其他线程的操作执行完后再执行。是一个倒计数的锁存器，当计数减至 0 时触发特定的事件，构造时传入 int 参数,该参数就是计数器的初始值，每调用一次 countDown()方法，计数器减 1,计数器大于 0 时，await()方法会阻塞程序继续执行。

CyclicBarrier：是一个同步工具类，允许一组线程互相等待，直到到达某个公共屏障点 (common barrier point)。在涉及一组固定大小的线程的程序中，这些线程必须不时地互相等待，此时 CyclicBarrier 很有用。因为该 barrier 在释放等待线程后可以重用，所以称它为循环 的 barrier。有点像 CountDownLatch，可以用于一个主任务多个子任务的应用场景，主任务等所有的子任务完成后再执行。

Semaphore：用来控制同时访问特定资源的线程数量，它通过 acquire()和 release()来维护一个许可证集，它会阻塞 acquire()获取许可证之前的线程，只有 release()释放后,才会允许新的线程进入。(举个例子,排队买火车票,保安 Semaphore 每次只会放进去 (给通行症) 十个人排队，每买好票走一个人（释放通行症），保安就再放进去一个；现在大家都网上买票了，不知道能否理解。)

FutureTask：它实现了 Runnable, Future, Runnable 一个是线程的标志，说明 FutureTask 可以作为一个线程实现当作线程执行;Future 就是对于具体的 Runnable 或者 Callable 任务的执行结果进行取消、查询是否完成、获取结果，可以通过 get 方法获取执行结果，该方法会阻塞直到任务返回结果，所以又可以作为 Future 得到 Callable 的返回值。

PS:虽然我想尽可能简单的描述，但概念性的东西还是决定采用通用的比较好，如果感觉学习线程类比较困难时,最好把它们放在一起比较着学习,CountDownLatch、CyclicBarrier、Semaphore 这三个都是 JDK5 引入的，学习的时候相互比较就会容易掌握。对于学有余力的同学呢，可以看一下这些同步类的实现源码，分析一下实现原理。

2.2 如果使对象 GC 后再活一次

解析：这里是间接考察了一下 finalize 用法，关于 finalize 在前面与 final、finally 的区别中已经详细说了，不熟悉的同学可以往前翻阅一下，下面写了一个小 DEMO 可以很好的说明 finalize 的用处，但不建议在开发中显示调用，它并不适合做资源关闭操作，因为它的执行很随机。

```
class User {
    public static User user = null;
    @Override
    protected void finalize() throws Throwable {
        System.out.println("hi,finalize被调用..");
        super.finalize();
        user = this; // 再给你一次生命
    }
}

public class TestFinalize {
    public static void main(String[] args) throws InterruptedException {
        // TODO Auto-generated method stub
        User u = new User();
        u = null;
        System.gc(); // 垃圾回收执行
        Thread.sleep(1000); // 保证回收
        // finalize 调用后
        u = User.user;
        System.out.println(u == null ? "彻底死了" : "又活了一次");
        System.gc();
        Thread.sleep(1000); // 保证回收

        u = User.user;
        u = null;
        System.out.println(u == null ? "彻底死了" : "又活了一次");
        Thread.sleep(1000); // 保证回收
    }
```

参考答案：在 java 中如果对象被 GC 后，可以使对象再活一次，这可以充分利用 finalize 关键字的作用，它在垃圾回收中被调用，而且仅调用一次。

2.3 ThreadLocal 的基本原理

解析：下面两个问题是一个同学面试的时候遇到的，网上也能看到，问题不难但平时不留意也不太容易回答。1、每个线程的变量副本是存储在哪里的？2、threadlocal 是何时初始化的？变量副本是如何为共享的那个变量赋值的？回答这样的问题，建议大家看一下 JDK 相关 threadlocal 部分的源码，下面只引用部分源码来解释说明。

参考答案：ThreadLocal 并非是线程的本地实现，而是线程的本地变量，它归附于具体的线程，为每个使用该变量的线程提供一个副本，每个线程都可以独立的操作这个副本，在外面看来，貌似每个线程都有一份变量。

线程的变量存在哪里，这里可以结果 ThreadLocal 的源码说明，这里看一下 get 实现

```
public T get() {
    Thread t = Thread.currentThread();
    ThreadLocalMap map = getMap(t);
    if (map != null) {
        ThreadLocalMap.Entry e = map.getEntry(this);
        if (e != null)
            return (T)e.value;
    }
    return setInitialValue();
}
```

在 get 方法中，会先获得当前线程对象，然后传到 getMap()中获取 ThreadLocalMap 对象，我们要的变量副本是从 ThreadLocalMap 对象中取出来的，可见每个线程的变量副本是保存在 ThreadLocalMap 对象里，而跟一下代码可以看到 ThreadLocalMap 是在 Thread 中声明实现的，所以**每个线程的变量副本就保存在相应线程的 ThreadLocalMap 对象中**。

第二个问题，可以理解 ThreadLocal 如何把变量的副本复制并且初始化的（声明和初始化），这里看一下源码中的 set 方法实现

```
public void set(T value) {
    Thread t = Thread.currentThread();
    ThreadLocalMap map = getMap(t);
    if (map != null)
        map.set(this, value);
    else
        createMap(t, value);
}
```

当第一次调用 set 方法时，获取的 ThreadLocalMap 对象为空，这里会调用 createMap 方法创建一个 ThreadLocalMap 对象，并且完成相应的初始，将 value 值存放进去。后面再次调用将会直接从线程中获取 ThreadLocalMap 对象，然后将副本保存进去。

2.4 Java 内存泄露原因有哪些

解析：这个问题看似简单，却用一个问题考察了 JVM 很多个相关的知识点，回答这个问题你首先要了解 JVM 的结构、对象的分配与存储、GC 的原理等，但你看到此的时候如果对上面的知识点还不是很熟悉的话，先翻开其相关知识点，之前都已经谈到过，然后这个问题就容易回答了。

JVM 结构上一般分为堆内存、栈内存、方法区，内存泄露可能会发生在任何一个位置；在 JVM 划分上方法区通常划给堆，所以这两块可以一起。而栈内存通常是用来存放普通变量和对象引用，回收速度速度快，一般不会造成内存泄露，一旦溢出通常是栈内存大小分配不合理，或者可能显示的将对象空间分配到栈内存来追求效率造成的。下面我们重点探讨堆内存的溢出（根据面试的场景来判断有没有谈栈内存这块）。

参考答案： 首先造成 Java JVM 泄露的主要原因：JVM 未及时的对垃圾进行回收造成的；当对象失去引用且持续占有内存或无用对象的内存得不到及时释放，从而造成内存空间的浪费称为内存泄漏。造成这种对象无法及时释放导致内存泄露的原因，可以简单的为归分两类。

一是基于设计方面：1、对应用加载数据级别判断失误，从而导致 JVM 内存分配不合理（企业单机部署应用常见到）。2、应用请求的常连接设计，常连接会一直占用后台资源，不能及时释放。3、数据库操作时，存在很多耗时连接，导致大量资源不能释放。4、大量的监听设计等。

二是基于开发方面：1、大量静态变量的使用（静态变量的生成周期与应用一致），如果静态引用指向的是集合或者数据，会一直占用资源。2、不合理的方法使用，比如 jdk6 之前的 substring 就可能导致内存泄露。3、数据库连接未能及时关闭，刚工作不久的同学容易忽略。4、单例模式使用，单例通常用来加载资源信息，但如果加载信息里有大量的集合、数组等对象，这些资源会一直驻留内存中，不易释放。5、在循环中创建复杂对象、一次性读取加载大量信息到内存中，都有可能造成内存泄露。

内存溢出与内存泄露是一回事吗？内存溢出主要是因为内存分配不足，在加载的时候溢出；泄露主要是 GC 回收不及时造成的。

2.5 GC 如何判断对象失去引用

解析：先了解这块再去看 GC 回收原理，会简单些；这里主要谈一下 GC 是如何来判断对象的引用失去而进行垃圾回收的，GC 判断引用是否可达是从 root 开始的，这里主要解释清 JVM 哪些是可以作为 root 判断便可。

参考答案： GC 判断对象引用是否可达是从 Root 根目录开始判断的，在 JVM 中可以作为 gc root 的有：**JVM 栈中对象引用（主要）、方法区中引用的对象、JNI 方法引用的对象、静态区域中引用的对象（一般不回收）**。对于多层结构的，则已递归的方式查找，能到达的都是不会被 GC 的对象。

2.6 OO 的设计原则

解析：上面淋淋总总的谈了很多知识点了，一直希望轻松着装深度备战，可好的待遇与前期的负出总是深深相关，这里放松一下回到面向对象的设计上，来关注一下面向对象设计的五大原则，这是一个非常重要的知识点也容易被忽视；本问题对初级高级架构级别面试都可考察，值得掌握。

OO 的设计原则，通常总成五类 SRP、OCP、LSP、DIP、ISP，属于记忆型面试题，难度不大。

参考答案： 面向对象设计原则通常归结为五大类，

第一“单一职责原则”（SRP）：一个设计元素只做一件事，不要随意耦合，多管闲事；

第二“开放封闭原则”（OCP）：对变更关闭、对扩展开放，提倡基于接口设计，新的需要最好不要去变更已经完成的功能，可以灵活通过接口扩展新功能；

第三“里氏替换原则”（LSP）：子类可以替换父类并且出现在父类能够出现的任何地方，这个也是提倡面向接口编程思想的；

第四“依赖倒置原则”（DIP）：要依赖于抽象，不要依赖于具体，简单的说就是面对抽象编程，不要过于依赖于细节；

第五“接口隔离原则”（ISP）：使用多个专门的接口比使用单个接口要好，在设计中

不要把各种业务类型的东西放到一个接口中造成臃肿。

PS: 面试时，参考答案中的解释性东西可以不问不回答；复习完五大设计原则，可以顺便回忆下面向对象的三大特性。

第 02 章　数据结构与算法

对于算法这节，我会把 Java 示例代码发给大家，大家作为参考，慢慢体会，算法通常在面试中是让你写出来，所以大家最好能在理解的基础上掌握。

2.1　冒泡排序

解析：在要排序的一组数中，对当前还未排好序的范围内的全部数，相邻两个数依次比较，将小的或者大的数向一端冒出。

参考答案：

```
public void sort() throws Exception {
        int[] arr = SortUtil.getIntarrayay(); //生成随机数组
        SortUtil.display(arr); //排序前
        for (int i = 0; i < arr.length - 1; i++) {
            for (int j = 0; j < arr.length - i - 1; j++) {
                if (arr[j] > arr[j + 1]) { //如果前面的数大于后面
                    arr[j] = arr[j] ^ arr[j + 1];
                    arr[j + 1] = arr[j] ^ arr[j + 1];
                    arr[j] = arr[j] ^ arr[j + 1];
                }
            }
        }
        SortUtil.display(arr); //排序后
}
```

2.2　选择排序

解析：在要排序的一组数中，选出最小的一个数与第一个位置的数交换；然后在剩下的数当中再找最小的与第二个位置的数交换，如此循环到倒数第二个数和最后一个数比较为止。

参考答案：

```
public void sort() throws Exception {
        int[] arr = SortUtil.getIntarrayay();
        SortUtil.display(arr);
        int k = 0;
        int temp = 0;
        for (int i = 0; i < arr.length - 1; i++) {
```

```
            k = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[k]) {
                    k = j;
                }
            }
            temp = arr[i];
            arr[i] = arr[k];
            arr[k] = temp;
        }
        SortUtil.display(arr);
}
```

2.3 插入排序

解析：在要排序的一组数中，假设前面(n-1)[n>=2] 个数已经是排好顺序的，现在要把第 n 个数插到前面的有序数中，使得这 n 个数也是排好顺序的。如此反复循环，直到全部排好顺序。

参考答案：

```
public void sort() throws Exception {
        int[] arr = SortUtil.getIntarrayay();
        SortUtil.display(arr);
        int currentValue = 0;
        int k = 0;// 当前位置
        for (int i = 1; i <= arr.length - 1; i++) {
            currentValue = arr[i]; // 当前值
            k = i;
            for (int j = i - 1; j >= 0; j--) {
                if (arr[j] > currentValue) { // 已排好中的数大于当前值
                    arr[j + 1] = arr[j]; // 就把排序中的数向后移动
                    k--;// 下标向前移动一位
                } else {
                    break;
                }
            }
            arr[k] = currentValue;
        }
        SortUtil.display(arr);
}
```

2.4 快速排序

解析：选择一个基准元素,通常选择第一个元素或者最后一个元素,通过一趟扫描，将待排序列分成两部分,一部分比基准元素小,一部分大于等于基准元素,此时基准元素在其排好序后的正确位置,然后再用同样的方法递归地排序划分的两部分。

参考答案：

```
public void sort() throws Exception {
        int[] arr = SortUtil.getIntarrayay();
        SortUtil.display(arr);
        diguiSort(arr,0,arr.length-1);
        SortUtil.display(arr);
    }

    public  void diguiSort(int[] arr,int L,int R){
        int js = arr[(L+R)/2];
        int i = L ;
        int j = R ;
        int temp = 0;
        while(i<=j){
            while (arr[i] < js) {
                i++;
            }
            while (arr[j] > js) {
                j--;
            }
            if(i<=j){
                temp = arr[j];
                arr[j] = arr[i];
                arr[i] = temp;
                i++;
                j--;
            }
        }
        if(L<j){
            diguiSort(arr,L,j);
        }
        if(i<R){
            diguiSort(arr,i,R);
        }
}
```

2.5 二分法查找

解析：BAT 公司的面试题，大家可以作为了解；算法的思想是对于已经有序的数组，查找一个随机数字时，每次取数组的中间数与目标数比较便可。

参考答案：

```
public static int find(int[] array, int objKey) {
    int start = 0;
    int end = array.length - 1;
    while (start <= end) {
        int middle = (start + end) / 2; //找出中间位
        if (objKey < array[middle]) {
            end = middle - 1;
        } else if (objKey > array[middle]) {
            start = middle + 1;
        } else {
            return middle;
        }
    }
    return -1;
}
```

2.6 递归逆序输出一个 int 类型数

原题：用递归算法对输入一个整形数，然后逆序输出，输出的必须是字符串。

解析：这是一个 BAT 公司面试题，感兴趣的可以了解一下。

参考答案：

```
public static String reverse(int a) {
        if(a<0){
            return "请输入一个正整数..";
        }
        if (a < 10)
            return Integer.toString(a);
        int last = a - (a / 10) * 10;// 获取末位
        return Integer.toString(last) + reverse(a / 10);// 递归输出最后一位和前面的倒序数字
}
```

第 03 章 设计模式

设计模式基本是面试必问内容，在给大家发第一个版的面试指南时，有一个“24 种设计

模式介绍及 6 大设计原则.pdf”文档，里面把常用的设计模式以一种很幽默的方式都介绍了一遍，大家如果想全面研究记得多看一下；这里就浅尝辄止的介绍几个大家必须掌握的，以便面试机动应答。

3.1 单例模式

解析：单例模式作为系统设计者一般都会用到的设计模式，即一个类只有一个实例，而且这个实例通常一旦创建就一直活着，不会被 GC。单例模式通常有两种写法，就是我们常说的懒汉式和饿汉式。

参考答案：单例模式指一个类通常只被实例化一次，通常用于系统资源实始化加载；常用两种写法为懒汉式和饿汉式。

```
//懒汉式
public class SingletonLazy {
    private static SingletonLazy instance;
    private SingletonLazy() {
    }
    public static SingletonLazy getInstance() {
        if (instance == null) { //在第一次调用时初始化，只实例化一次
            instance = new SingletonLazy();
        }
        return instance;
    }
}
//饿汉式
public class SingletonHungry {
    //系统加载时初始化，只初始化一次
    private static SingletonHungry instance = newSingletonHungry();
    private SingletonHungry() {
    }
    public static SingletonHungry getInstance() {
        return instance;
    }
}
//JDK 中有哪些类是基于单例实现的，大家看一下 Runtime
```

扩展：如果在并发环境下使用单例模式，以上代码肯定会存在并发问题，可能创建多个实例出来，大家结合多线程锁的概念，自己完善一下安全的单例写法(加下锁)。

3.2 工厂模式

解析：工厂模式应该是 Java 学习者接触的第一个设计模式，工厂模式主要分为“简单工厂模式”、“工厂方法模式”、“抽象工厂模式”，后面两种工厂模式在【24 种设计模式介绍及 6 大设计原则.pdf】文档中已经有很详细的描述，请参考，这里省略示例代码。

参考答案：根据工作中常用设计模式归类，工厂模式可以分为三类**“简单工厂模式”、“工厂方法模式”、“抽象工厂模式”**，其中“简单工厂模式”并未归到 23 种 GOF 设计模式之一，它是另外两种工厂模式的基础，而且在小项目开发中使用广泛。

简单工厂模式：属于创建型模式，又叫做静态工厂方法（Static Factory Method）模式，是由一个工厂对象决定创建出哪一种产品类的实例。简单工厂模式是工厂模式家族中最简单实用的模式。

优点：容易实现、使用者只需要关注调用的接口，不需要关心具体实现。

缺点：实现类的增减都会牵扯到工厂类的修改，工厂类偶合了实例的业务逻辑，不符合高内聚低耦合的原则，也违背了开放封闭原则。

工厂方法模式：就是我们通常简称的工厂模式，它属于类创建型模式，也是处理在不指定对象具体类型的情况下创建对象的问题。实质（使用）定义一个创建对象的接口，但让实现这个接口的类来决定实例化哪个类。工厂方法让类的实例化推迟到子类中进行。”

优点：实现简单，符合开闭原则 。

缺点：新增实现，会导到工厂类大量增加，不符合优化原则 。

抽象工厂模式：提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。

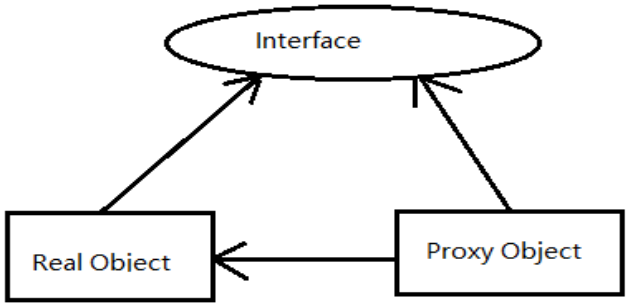
优点：隔离了具体类的生成，使得客户并不需要知道什么被创建，而且每次可以通过具体工厂类创建一个产品族中的多个对象，增加或者替换产品族比较方便。

缺点：结构复杂，实现难度相对来说高。

说明：各位单看上面的概念，是不是云里雾里，如果单看概念肯定一点实质感没有，建议结合实例理解的基础参考上面的概念性东西，组织一下在面试中用自己理解的话来回答。

3.3 代理模式

解析：代理模式分为”静态代理”和“动态代理”，而动态代理又分为“JDK 动态代理”和“CGLIB 动态代理”。静态代理在给大家的文档中，看过的对潘金莲那章节应该印象深刻，这里主要说一下 JDK 和 GCLIB 两种方式（在前面 AOP 中已经提到过，这里再强调一下），也是 AOP 的核心。



（静态代理实现图）

参考答案：动态代理可以分为”静态代理”和“动态代理”，动态代理又分为“JDK 动态代理”和“CGLIB 动态代理”实现。

静态代理：（如上图）代理对象和实际对象都继续了同一个接口，在代理对象中指向的是实际对象的实例，这样对外暴露的是代理对象而真正调用的是 Real Object。

优点：可以很好的保护实际对象的业务逻辑对外暴露，从而提高安全性。

缺点：不同的接口要有不同的代理类实现，会很冗余。

JDK 动态代理:为了解决静态代理中，生成大量的代理类造成的冗余；JDK 动态代理只需要实现 InvocationHandler 接口，重写 invoke 方法便可以完成代理的实现，从代理类的 Proxy.newProxyInstance(targetObject.getClass().getClassLoader(),***targetObject.getClass().getInterfaces()***, this);获取方式来看，必须传入实现的接口。

优点：解决了静态代理中冗余的代理实现类问题。

缺点：JDK 动态代理是基于接口设计实现的，如果没有接口，但会抛异常。

CGLIB 代理：由于 JDK 动态代理限制了只能基于接口设计，而对于没有接口的情况，JDK 方式解决不了；CGLib 采用了非常底层的字节码技术，其原理是通过字节码技术为一个类创建子类，并在子类中采用方法拦截的技术拦截所有父类方法的调用，顺势织入横切逻辑，来完成动态代理的实现。实现方式实现 MethodInterceptor 接口，重写 intercept 方法，通过 Enhancer 类的回调方法来实现。

优点：没有接口也能实现动态代理，而且采用字节码增强技术，性能也不错。

缺点：技术实现相对难理解些。

说明：建议大家自己写一下 JDK 和 CGLIB 代理的求你，好好理解一下。

3.4 适配模式

解析：本想介绍一下策略模式，但策略模式很 Easy 大家看一眼就能理解了，参见一下我们附件中【24 种设计模式介绍及 6 大设计原则.pdf】，结合实际工作需要来说，适配器模式用到的还是比较多，尤其是老项目扩充新功能中常遇见。

场景举例：我们管理中心对外暴露一个用户上传的接口给企业上报用户使用，但企业用户的信息存储与我们的不符合，比如我们地域存的是省市县三级三个字段，他们采用的是空格隔开的一个字段，我们要不修改业务逻辑情况下保存他们上报过来的数据，就需要在我们原有的接口之上加一级适配，将他们的数据转换成我们业务可以使用的格式。

参考答案：适配模式可以分为“类适配”和“对象适配”两种方式，这里要介绍场景举例说明（要不用白板画图说明）以上场景为例：管理中心的接口国 target（源接口），对外提供的用户上传接口为 outOjb 接口，实现类 adaptee，适配类 adapter。

类适配：适配类 adapter 需要继承 adaptee 类同时实现 target 源接口，在适配类中完成数据格式转化。

对象适配：与类适配不一样的是，不需要继续 **adaptee，**适配类 adapter 只要实现 target 源接口，在 adapter 类中传入 outojb 的实现实例 adaptee 对象便可。

说明：由于时间紧，来不急画图说明，大家参考文档中的类关系图理解，面试的时候也要结合实例说明。

【下面分享群中同学买的视频，与本面试指南无关，可以忽略】

群中王同学买了一套视频，原价 600 多，想找人合购分担下，里面主要是一下分布式大数据内容（dubbo、zookeeper、java8、redis、Git、Java 版数据结构、NIO、JVM、storm 等 300G 左右），大家有兴趣可以看下（大家如果有很好的东西也可以帮助代挂一下）。
https://item.taobao.com/item.htm