1、什么是线程局部变量？

线程局部变量是局限于线程内部的变量，属于线程自身所有，不在多个线程间共享。Java 提供 ThreadLocal 类来支持线程局部变量，是一种实现线程安全的方式。但是在管理环境下（如 web 服务器）使用线程局部变量的时候要特别小心，在这种情况下，工作线程的生命周期比任何应用变量的生命周期都要长。任何线程局部变量一旦在工作完成后没有释放，Java 应用就存在内存泄露的风险。

2.用 wait-notify 写一段代码来解决生产者-消费者问题？

请参考答案中的示例代码。只要记住在同步块中调用 wait() 和 notify()方法，如果阻塞，通过循环来测试等待条件。

【生产者】

package com.edu.chapter03.test;

import java.util.Vector;

import java.util.logging.Level;

import java.util.logging.Logger;

public class Producer implements Runnable {

private final Vector sharedQueue;

private final int SIZE;

public Producer(Vector sharedQueue, int size) {

this.sharedQueue = sharedQueue;

this.SIZE = size;

}

@Override

public void run() {

// TODO Auto-generated method stub

for (int i = 0; i < 7; i++) {

System.out.println("Produced:" + i);

try {

produce(i);

} catch (InterruptedException ex) {

Logger.getLogger(Producer.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

private void produce(int i) throws InterruptedException {

//wait if queue is full

while (sharedQueue.size() == SIZE) {

synchronized (sharedQueue) {

System.out.println("Queue is full " + Thread.currentThread().getName()

+ " is waiting , size: " + sharedQueue.size());

sharedQueue.wait();

}

}

//producing element and notify consumers

synchronized (sharedQueue) {

sharedQueue.add(i);

sharedQueue.notifyAll();

}

}

}

【消费者】

package com.edu.chapter03.test;

import java.util.Vector;

import java.util.logging.Level;

import java.util.logging.Logger;

public class Consumer implements Runnable {

private final Vector sharedQueue;

private final int SIZE;

public Consumer(Vector sharedQueue, int size) {

this.sharedQueue = sharedQueue;

this.SIZE = size;

}

@Override

public void run() {

// TODO Auto-generated method stub

while (true) {

try {

System.out.println("Consumer: " + consume());

Thread.sleep(50);

} catch (InterruptedException ex) {

Logger.getLogger(Consumer.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

private int consume() throws InterruptedException {

//wait if queue is empty

while (sharedQueue.isEmpty()) {

synchronized (sharedQueue) {

System.out.println("Queue is empty " + Thread.currentThread().getName()

+ " is waiting , size: " + sharedQueue.size());

sharedQueue.wait();

}

}

//otherwise consume element and notify waiting producer

synchronized (sharedQueue) {

sharedQueue.notifyAll();

return (Integer) sharedQueue.remove(0);

}

}

}

【测试函数】

package com.edu.chapter03.test;

import java.util.Vector;

public class ProducerConsumerSolution {

public static void main(String[] args) {

Vector sharedQueue = new Vector();

int size = 4;

Thread prodThread = new Thread(new Producer(sharedQueue, size), "Producer");

Thread consThread = new Thread(new Consumer(sharedQueue, size), "Consumer");

prodThread.start();

consThread.start();

}

}

3. 用 Java 写一个线程安全的单例模式（Singleton）？

请参考答案中的示例代码，这里面一步一步教你创建一个线程安全的 Java 单例类。当我们说线程安全时，意思是即使初始化是在多线程环境中，仍然能保证单个实例。Java 中，使用枚举作为单例类是最简单的方式来创建线程安全单例模式的方式。

立即加载/饿汉式：

【在调用方法前，实例就已经被创建】

package com.weishiyao.learn.day8.singleton.ep1;

public class MyObject {

// 立即加载方式==恶汉模式

private static MyObject myObject = new MyObject();

private MyObject() {

}

public static MyObject getInstance() {

// 此代码版本为立即加载

// 此版本代码的缺点是不能有其他实例变量

// 因为getInstance()方法没有同步

// 所以有可能出现非线程安全的问题

return myObject;

}

}

【创建线程类】

package com.weishiyao.learn.day8.singleton.ep1;

public class MyThread extends Thread {

@Override

public void run() {

System.out.println(MyObject.getInstance().hashCode());

}

}

【创建运行类】

package com.weishiyao.learn.day8.singleton.ep1;

public class Run {

public static void main(String[] args) {

MyThread t1 = new MyThread();

MyThread t2 = new MyThread();

MyThread t3 = new MyThread();

t1.start();

t2.start();

t3.start();

}

}

延迟加载/懒汉式

【建对象的实例】

package com.weishiyao.learn.day8.singleton.ep2;

public class MyObject {

private static MyObject myObject;

private MyObject() {

}

public static MyObject getInstance() {

// 延迟加载

if (myObject != null) {

} else {

myObject = new MyObject();

}

return myObject;

}

}

【创建线程类】

package com.weishiyao.learn.day8.singleton.ep2;

public class MyThread extends Thread {

@Override

public void run() {

System.out.println(MyObject.getInstance().hashCode());

}

}

【创建运行类】

package com.weishiyao.learn.day8.singleton.ep2;

public class Run {

public static void main(String[] args) {

MyThread t1 = new MyThread();

t1.start();

}

}

【运行测试类】

package com.weishiyao.learn.day8.singleton.ep2;

public class Run {

public static void main(String[] args) {

MyThread t1 = new MyThread();

MyThread t2 = new MyThread();

MyThread t3 = new MyThread();

MyThread t4 = new MyThread();

MyThread t5 = new MyThread();

t1.start();

t2.start();

t3.start();

t4.start();

t5.start();

}

}

4.Java 中 sleep 方法和 wait 方法的区别？

虽然两者都是用来暂停当前运行的线程，但是 sleep() 实际上只是短暂停顿，因为它不会释放锁，而 wait() 意味着条件等待，这就是为什么该方法要释放锁，因为只有这样，其他等待的线程才能在满足条件时获取到该锁。

5.什么是不可变对象（immutable object）？Java 中怎么创建一个不可变对象？

不可变对象指对象一旦被创建，状态就不能再改变。任何修改都会创建一个新的对象，如 String、Integer及其它包装类。详情参见答案，一步一步指导你在 Java 中创建一个不可变的类。

6.我们能创建一个包含可变对象的不可变对象吗？

是的，我们是可以创建一个包含可变对象的不可变对象的，你只需要谨慎一点，不要共享可变对象的引用就可以了，如果需要变化时，就返回原对象的一个拷贝。最常见的例子就是对象中包含一个日期对象的引用。

7.Java 中应该使用什么数据类型来代表价格？

如果不是特别关心内存和性能的话，使用BigDecimal，否则使用预定义精度的 double 类型。

8.怎么将 byte 转换为 String？

可以使用 String 接收 byte[] 参数的构造器来进行转换，需要注意的点是要使用的正确的编码，否则会使用平台默认编码，这个编码可能跟原来的编码相同，也可能不同。

9.Java 中 bytes 与其他类型的转换？

public class Test {

private static ByteBuffer buffer = ByteBuffer.allocate(8);

public static void main(String[] args) {

//测试 int 转 byte

int int0 = 234;

byte byte0 = intToByte(int0);

System.out.println("byte0=" + byte0);//byte0=-22

//测试 byte 转 int

int int1 = byteToInt(byte0);

System.out.println("int1=" + int1);//int1=234

//测试 int 转 byte 数组

int int2 = 1417;

byte[] bytesInt = intToByteArray(int2);

System.out.println("bytesInt=" + bytesInt);//bytesInt=[B@de6ced

//测试 byte 数组转 int

int int3 = byteArrayToInt(bytesInt);

System.out.println("int3=" + int3);//int3=1417

//测试 long 转 byte 数组

long long1 = 2223;

byte[] bytesLong = longToBytes(long1);

System.out.println("bytes=" + bytesLong);//bytes=[B@c17164

//测试 byte 数组 转 long

long long2 = bytesToLong(bytesLong);

System.out.println("long2=" + long2);//long2=2223

}

//byte 与 int 的相互转换

public static byte intToByte(int x) {

return (byte) x;

}

public static int byteToInt(byte b) {

//Java 总是把 byte 当做有符处理；我们可以通过将其和 0xFF 进行二进制与得到它的无符值

return b & 0xFF;

}

//byte 数组与 int 的相互转换

public static int byteArrayToInt(byte[] b) {

return b[3] & 0xFF |

(b[2] & 0xFF) << 8 |

(b[1] & 0xFF) << 16 |

(b[0] & 0xFF) << 24;

}

public static byte[] intToByteArray(int a) {

return new byte[] {

(byte) ((a >> 24) & 0xFF),

(byte) ((a >> 16) & 0xFF),

(byte) ((a >> 8) & 0xFF),

(byte) (a & 0xFF)

};

}

//byte 数组与 long 的相互转换

public static byte[] longToBytes(long x) {

buffer.putLong(0, x);

return buffer.array();

}

public static long bytesToLong(byte[] bytes) {

buffer.put(bytes, 0, bytes.length);

buffer.flip();//need flip

return buffer.getLong();

}

}

10.我们能将 int 强制转换为 byte 类型的变量吗？如果该值大于 byte 类型的范围，将会出现什么现象？是的，我们可以做强制转换，但是 Java 中 int 是 32 位的，而 byte 是 8 位的，所以，如果强制转化是，int 类型的高 24 位将会被丢弃，byte 类型的范围是从 -128 到 128。

11.存在两个类，B 继承 A，C 继承 B，我们能将 B 转换为 C 么？如 C = (C) B；

可以，向下转型。但是不建议使用，容易出现类型转型异常。

12.哪个类包含 clone 方法？是 Cloneable 还是 Object？

java.lang.Cloneable 是一个标示性接口，不包含任何方法，clone 方法在 object 类中定义。并且需要知道 clone() 方法是一个本地方法，这意味着它是由 c 或 c++ 或 其他本地语言实现的。

13.Java 中 ++ 操作符是线程安全的吗？

不是线程安全的操作。它涉及到多个指令，如读取变量值，增加，然后存储回内存，这个过程可能会出现多个线程交差。

14.a = a + b 与 a += b 的区别

+= 隐式的将加操作的结果类型强制转换为持有结果的类型。如果两这个整型相加，如 byte、short 或者 int，首先会将它们提升到 int 类型，然后在执行加法操作。如果加法操作的结果比 a 的最大值要大，则 a+b 会出现编译错误，但是 a += b 没问题，如下：

byte a = 127;

byte b = 127;

b = a + b; // error : cannot convert from int to byte

b += a; // ok

（译者注：这个地方应该表述的有误，其实无论 a+b 的值为多少，编译器都会报错，因为 a+b 操作会将 a、b 提升为 int 类型，所以将 int 类型赋值给 byte 就会编译出错）

15.我能在不进行强制转换的情况下将一个 double 值赋值给 long 类型的变量吗？

不行，你不能在没有强制类型转换的前提下将一个 double 值赋值给 long 类型的变量，因为 double 类型的范围比 long 类型更广，所以必须要进行强制转换。

16. 3\*0.1 == 0.3 将会返回什么？true 还是 false？

false，因为有些浮点数不能完全精确的表示出来。

17.int 和 Integer 哪个会占用更多的内存？

Integer 对象会占用更多的内存。Integer 是一个对象，需要存储对象的元数据。但是 int 是一个原始类型的数据，所以占用的空间更少。

18.为什么 Java 中的 String 是不可变的（Immutable）？

Java 中的 String 不可变是因为 Java 的设计者认为字符串使用非常频繁，将字符串设置为不可变可以允许多个客户端之间共享相同的字符串。更详细的内容参见答案。

19.我们能在 Switch 中使用 String 吗？

从 Java 7 开始，我们可以在 switch case 中使用字符串，但这仅仅是一个语法糖。内部实现在 switch 中使用字符串的 hash code。

20.Java 中的构造器链是什么？

当你从一个构造器中调用另一个构造器，就是Java 中的构造器链。这种情况只在重载了类的构造器的时候才会出现。

21.64 位 JVM 中，int 的长度是多数？

Java 中，int 类型变量的长度是一个固定值，与平台无关，都是 32 位。意思就是说，在 32 位 和 64 位 的Java 虚拟机中，int 类型的长度是相同的。

22.Serial 与 Parallel GC之间的不同之处？

Serial 与 Parallel 在GC执行的时候都会引起 stop-the-world。它们之间主要不同 serial 收集器是默认的复制收集器，执行 GC 的时候只有一个线程，而 parallel 收集器使用多个 GC 线程来执行。

23. 32 位和 64 位的 JVM，int 类型变量的长度是多数？

32 位和 64 位的 JVM 中，int 类型变量的长度是相同的，都是 32 位或者 4 个字节。

24.Java 中 WeakReference 与 SoftReference的区别？

虽然 WeakReference 与 SoftReference 都有利于提高 GC 和 内存的效率，但是 WeakReference ，一旦失去最后一个强引用，就会被 GC 回收，而软引用虽然不能阻止被回收，但是可以延迟到 JVM 内存不足的时候。

25.WeakHashMap 是怎么工作的？

WeakHashMap 的工作与正常的 HashMap 类似，但是使用弱引用作为 key，意思就是当 key 对象没有任何引用时，key/value 将会被回收。

26.JVM 选项 -XX:+UseCompressedOops 有什么作用？为什么要使用？

当你将你的应用从 32 位的 JVM 迁移到 64 位的 JVM 时，由于对象的指针从 32 位增加到了 64 位，因此堆内存会突然增加，差不多要翻倍。这也会对 CPU 缓存（容量比内存小很多）的数据产生不利的影响。因为，迁移到 64 位的 JVM 主要动机在于可以指定最大堆大小，通过压缩 OOP 可以节省一定的内存。通过 -XX:+UseCompressedOops 选项，JVM 会使用 32 位的 OOP，而不是 64 位的 OOP。

27.怎样通过 Java 程序来判断 JVM 是 32 位 还是 64 位？

你可以检查某些系统属性如 sun.arch.data.model 或 os.arch 来获取该信息。

28.32 位 JVM 和 64 位 JVM 的最大堆内存分别是多数？

理论上说上 32 位的 JVM 堆内存可以到达 2^32，即 4GB，但实际上会比这个小很多。不同操作系统之间不同，如 Windows 系统大约 1.5 GB，Solaris 大约 3GB。64 位 JVM允许指定最大的堆内存，理论上可以达到 2^64，这是一个非常大的数字，实际上你可以指定堆内存大小到 100GB。甚至有的 JVM，如 Azul，堆内存到 1000G 都是可能的。

29.JRE、JDK、JVM 及 JIT 之间有什么不同？

JRE 代表 Java 运行时（Java run-time），是运行 Java 引用所必须的。JDK 代表 Java 开发工具（Java development kit），是 Java 程序的开发工具，如 Java 编译器，它也包含 JRE。JVM 代表 Java 虚拟机（Java virtual machine），它的责任是运行 Java 应用。JIT 代表即时编译（Just In Time compilation），当代码执行的次数超过一定的阈值时，会将 Java 字节码转换为本地代码，如，主要的热点代码会被准换为本地代码，这样有利大幅度提高 Java 应用的性能。

30.解释 Java 堆空间及 GC？

当通过 Java 命令启动 Java 进程的时候，会为它分配内存。内存的一部分用于创建堆空间，当程序中创建对象的时候，就从对空间中分配内存。GC 是 JVM 内部的一个进程，回收无效对象的内存用于将来的分配。

31.你能保证 GC 执行吗？

不能，虽然你可以调用 System.gc() 或者 Runtime.gc()，但是没有办法保证 GC 的执行。

32.怎么获取 Java 程序使用的内存？堆使用的百分比？

可以通过 java.lang.Runtime 类中与内存相关方法来获取剩余的内存，总内存及最大堆内存。通过这些方法你也可以获取到堆使用的百分比及堆内存的剩余空间。

Runtime.freeMemory() 方法返回剩余空间的字节数，Runtime.totalMemory() 方法总内存的字节数，Runtime.maxMemory() 返回最大内存的字节数。

33.Java 中堆和栈有什么区别？(答案)

JVM 中堆和栈属于不同的内存区域，使用目的也不同。栈常用于保存方法帧和局部变量，而对象总是在堆上分配。栈通常都比堆小，也不会在多个线程之间共享，而堆被整个 JVM 的所有线程共享。

34. “a==b”和”a.equals(b)”有什么区别？

如果 a 和 b 都是对象，则 a==b 是比较两个对象的引用，只有当 a 和 b 指向的是堆中的同一个对象才会返回 true，而 a.equals(b) 是进行逻辑比较，所以通常需要重写该方法来提供逻辑一致性的比较。例如，String 类重写 equals() 方法，所以可以用于两个不同对象，但是包含的字母相同的比较。

35.a.hashCode() 有什么用？与 a.equals(b) 有什么关系？

hashCode() 方法是相应对象整型的 hash 值。它常用于基于 hash 的集合类，如 Hashtable、HashMap、LinkedHashMap等等。它与 equals() 方法关系特别紧密。根据 Java 规范，两个使用 equal() 方法来判断相等的对象，必须具有相同的 hash code。

36.final、finalize 和 finally 的不同之处？

final 是一个修饰符，可以修饰变量、方法和类。如果 final 修饰变量，意味着该变量的值在初始化后不能被改变。finalize 方法是在对象被回收之前调用的方法，给对象自己最后一个复活的机会，但是什么时候调用 finalize 没有保证。finally 是一个关键字，与 try 和 catch 一起用于异常的处理。finally 块一定会被执行，无论在 try 块中是否有发生异常。

37.Java 中的编译期常量是什么？使用它又什么风险？

公共静态不可变（public static final ）变量也就是我们所说的编译期常量，这里的 public 可选的。实际上这些变量在编译时会被替换掉，因为编译器知道这些变量的值，并且知道这些变量在运行时不能改变。这种方式存在的一个问题是你使用了一个内部的或第三方库中的公有编译时常量，但是这个值后面被其他人改变了，但是你的客户端仍然在使用老的值，甚至你已经部署了一个新的jar。为了避免这种情况，当你在更新依赖 JAR 文件时，确保重新编译你的程序。

38.List、Set、Map 和 Queue 之间的区别

List 是一个有序集合，允许元素重复。它的某些实现可以提供基于下标值的常量访问时间，但是这不是 List 接口保证的。Set 是一个无序集合。

39.poll() 方法和 remove() 方法的区别？

poll() 和 remove() 都是从队列中取出一个元素，但是 poll() 在获取元素失败的时候会返回空，但是 remove() 失败的时候会抛出异常。

40.Java 中 LinkedHashMap 和 PriorityQueue 的区别是什么？

PriorityQueue 保证最高或者最低优先级的的元素总是在队列头部，但是 LinkedHashMap 维持的顺序是元素插入的顺序。当遍历一个 PriorityQueue 时，没有任何顺序保证，但是 LinkedHashMap 课保证遍历顺序是元素插入的顺序。

41.ArrayList 与 LinkedList 的不区别？

最明显的区别是 ArrrayList 底层的数据结构是数组，支持随机访问，而 LinkedList 的底层数据结构书链表，不支持随机访问。使用下标访问一个元素，ArrayList 的时间复杂度是 O(1)，而 LinkedList 是 O(n)。更多细节的讨论参见答案。

42.用哪两种方式来实现集合的排序？

你可以使用有序集合，如 TreeSet 或 TreeMap，你也可以使用有顺序的的集合，如 list，然后通过 Collections.sort() 来排序。

43.Java 中怎么打印数组？(answer)

你可以使用 Arrays.toString() 和 Arrays.deepToString() 方法来打印数组。由于数组没有实现 toString() 方法，所以如果将数组传递给 System.out.println() 方法，将无法打印出数组的内容，但是 Arrays.toString() 可以打印每个元素。

44.Java 中的 LinkedList 是单向链表还是双向链表？

是双向链表，你可以检查 JDK 的源码。在 Eclipse，你可以使用快捷键 Ctrl + T，直接在编辑器中打开该类。

45.Java 中的 TreeMap 是采用什么树实现的？

Java 中的 TreeMap 是使用红黑树实现的。

46. Hashtable 与 HashMap 有什么不同之处？

这两个类有许多不同的地方，下面列出了一部分：

a) Hashtable 是 JDK 1 遗留下来的类，而 HashMap 是后来增加的。

b）Hashtable 是同步的，比较慢，但 HashMap 没有同步策略，所以会更快。

c）Hashtable 不允许有个空的 key，但是 HashMap 允许出现一个 null key。

更多的不同之处参见答案。

47.Java 中的 HashSet，内部是如何工作的？

HashSet 的内部采用 HashMap来实现。由于 Map 需要 key 和 value，所以所有 key 的都有一个默认 value。类似于 HashMap，HashSet 不允许重复的 key，只允许有一个null key，意思就是 HashSet 中只允许存储一个 null 对象。

48.写一段代码在遍历 ArrayList 时移除一个元素？

该问题的关键在于面试者使用的是 ArrayList 的 remove() 还是 Iterator 的 remove()方法。这有一段示例代码，是使用正确的方式来实现在遍历的过程中移除元素，而不会出现 ConcurrentModificationException 异常的示例代码。

49.我们能自己写一个容器类，然后使用 for-each 循环码？

可以，你可以写一个自己的容器类。如果你想使用 Java 中增强的循环来遍历，你只需要实现 Iterable 接口。如果你实现 Collection 接口，默认就具有该属性。

50.ArrayList 和 HashMap 的默认大小是多数？

在 Java 7 中，ArrayList 的默认大小是 10 个元素，HashMap 的默认大小是16个元素（必须是2的幂）。这就是 Java 7 中 ArrayList 和 HashMap 类的代码片段：

// from ArrayList.java JDK 1.7

private static final int DEFAULT\_CAPACITY = 10;

//from HashMap.java JDK 7

static final int DEFAULT\_INITIAL\_CAPACITY = 1 << 4; // aka 16