1.int a=10 是原子操作吗？

是的。

注意点：

i++(或++i)是非原子操作，i++是一个多步操作，而且是可以被中断的。i++可以被分割成 3 步，第一步读取 i 的值，第二步计算 i+1；第三部将最终值赋值给 i。

- int a = b;不是原子操作。从语法的级别来看，这是也是一条语句，是原子的；但是从实际执行的二进制指令来看，由于现代计算机 CPU 架构体系的限制，数据不可以直接从内存搬运到另外一块内存，必须借助寄存器中断，这条语句一般对应两条计算机指令，即将变量 b 的值搬运到某个寄存器（如 eax）中，再从该寄存器搬运到变量 a 的内存地址：

```
mov eax, dword ptr [b]
mov dword ptr [a], eax
```

既然是两条指令，那么多个线程在执行这两条指令时，某个线程可能会在第一条指令执行完毕后被剥夺 CPU 时间片，切换到另外一个线程而产生不确定的情况。

2.innodb 支持全文索引吗？

5.6 版本之后 InnoDB 存储引擎开始支持全文索引，5.7 版本之后通过使用 ngram 插件开始支持中文。之前仅支持英文，因为是通过空格作为分词的分隔符，对于中文来说是不合适的。MySQL 允许在 char、varchar、text 类型上建立全文索引。

3.innodb 支持表锁吗？

支持，补充：普通的增删改 是表锁，加入索引的增删改是行锁，执行查询时不加任何锁的。

4.HTTP 短连接怎么变成长连接。

在 header 中加入 --Connection:keep-alive。

5.调用 yeild（）会阻塞吗？

阻塞指的是暂停一个线程的执行以等待某个条件发生（如某资源就绪）。

yield() 方法：yield() 使得线程放弃当前分得的 CPU 时间，但是不使线程阻塞，即线程仍处于可执行状态，随时可能再次分得 CPU 时间。调用 yield() 的效果等价于调度程序认为该线程已执行了足够的时间从而转到另一个线程。yield()只是使当前线程重新回到可执行状态，所以执行 yield()的线程有可能在进入到可执行状态后马上又被执行。sleep()可使优先级低的线程得到执行的机会，当然也可以让同优先级和高优先级的线程有执行的机会；yield()只能使同优先级的线程有执行的机会。

6.虚拟机栈是线程共享的吗？

不是。

JVM 初始运行的时候都会分配好 Method Area（方法区） 和 Heap（堆） ，而 JVM 每遇到一个线程，就为其分配一个 Program Counter Register（程序计数器），VM Stack（虚拟机栈）和 Native Method Stack （本地方法栈）， 当线程终止时，三者（虚拟机栈，本地方法栈和程序计数器）所占用的内存空间也会被释放掉。这也是为什么我把内存区域分为线程共享和非线程共享的原因，非线程共享的那三个区域的生命周期与所属线程相同，而线程共享的区域与 JAVA 程序运行的生命周期相同，所以这也是系统垃圾回收的场所只发生在线程共享的区域（实际上对大部分虚拟机来说只发生在 Heap 上）的原因。

栈区:

每个线程包含一个栈区，栈中只保存基础数据类型的值（比如 int i=1 中 1 就是基础类型的对象）和对象的引用以及基础数据的引用

每个栈中的数据(基础数据类型和对象引用)都是私有的，其他栈不能访问。

栈分为 3 个部分：基本类型变量区、执行环境上下文、操作指令区(存放操作指令)。

堆区:

存储的全部是对象，每个对象都包含一个与之对应的 class 的信息。(class 的目的是得到操作指令) jvm 只有一个堆区(heap)被所有线程共享，堆中不存放基本类型和对象引用，只存放对象本身 。

方法区:

又叫静态区，跟堆一样，被所有的线程共享。方法区包含所有的 class 和 static 变量。

方法区中包含的都是在整个程序中永远唯一的元素，如 class，static 变量。（两者区别为堆区存放 new 出来的对象信息,方法区存放本身就具有的类信息）

7.常量存放在 JVM 的那个区域？

方法区: 又叫静态区，跟堆一样，被所有的线程共享。它用于存储已经被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。

window.postMessage() 方法可以安全地实现跨源通信。通常，对于两个不同页面的脚本，只有当执行它们的页面位于具有相同的协议（通常为 https），端口号（443 为 https 的默认值），以及主机 (两个页面的模数 Document.domain 设置为相同的值) 时，这两个脚本才能相互通信。window.postMessage() 方法提供了一种受控机制来规避此限制，只要正确的使用，这种方法就很安全。

8.所有的对象都分配到堆中吗？

答：不一定。

9.CopyOnWriteArrayList 是线程安全的吗？

答：是的。

CopyOnWriteArrayList 使用了一种叫写时复制的方法，当有新元素添加到

CopyOnWriteArrayList 时，先从原有的数组中拷贝一份出来，然后在新的数组做写操作，写完之后，再将原来的数组引用指向到新数组。创建新数组，并往新数组中加入一个新元素,这个时候，array 这个引用仍然是指向原数组的。当元素在新数组添加成功后，将 array 这个引用指向新数组。

CopyOnWriteArrayList 的整个 add 操作都是在锁的保护下进行的。这样做是为了避免在多线程并发 add 的时候，复制出多个副本出来,把数据搞乱了，导致最终的数组数据不是我们期望的。

```
public boolean add(E e) {
    //1、先加锁
    final ReentrantLock lock = this.lock;
    lock.lock();
    try {
        Object[] elements = getArray();
        int len = elements.length;
        //2、拷贝数组
        Object[] newElements = Arrays.copyOf(elements, len + 1);
        //3、将元素加入到新数组中
        newElements[len] = e;
        //4、将 array 引用指向到新数组
        setArray(newElements);
        return true;
    } finally {
        //5、解锁
        lock.unlock();
    }
}
```

由于所有的写操作都是在新数组进行的，这个时候如果有线程并发的写，则通过锁来控制，如果有线程并发的读，则分几种情况：

如果写操作未完成，那么直接读取原数组的数据；

如果写操作完成，但是引用还未指向新数组，那么也是读取原数组数据；

如果写操作完成，并且引用已经指向了新的数组，那么直接从新数组中读取数据。

可见，CopyOnWriteArrayList 的读操作是可以不用加锁的。

CopyOnWriteArrayList 有几个缺点：

由于写操作的时候，需要拷贝数组，会消耗内存，

如果原数组的内容比较多的情况下，可能导致 young gc 或者 full gc

不能用于实时读的场景，像拷贝数组、新增元素都需要时间，

所以调用一个 set 操作后，读取到数据可能还是旧的,

虽然 CopyOnWriteArrayList 能做到最终一致性,但是还是没法满足实时性要求；

CopyOnWriteArrayList 合适读多写少的场景，不过这类慎用

因为谁也没法保证 CopyOnWriteArrayList 到底要放置多少数据，

万一数据稍微有点多，每次 add/set 都要重新复制数组，这个代价实在太高昂了。

在高性能的互联网应用中，这种操作分分钟引起故障。

CopyOnWriteArrayList 透露的思想

读写分离，读和写分开 最终一致性 使用另外开辟空间的思路，来解决并发冲突

10.数组越界问题

一般来讲我们使用时，会用一个线程向容器中添加元素，一个线程来读取元素，而读取的操作往往更加频繁。写操作加锁保证了线程安全，读写分离保证了读操作的效率，简直完美。

如果这时候有第三个线程进行删除元素操作，读线程去读取容器中最后一个元素，读之前的时候容器大小为 i，当去读的时候删除线程突然删除了一个元素，这个时候容器大小变为了 i-1，读线程仍然去读取第 i 个元素，这时候就会发生数组越界。

测试一下，首先向 CopyOnWriteArrayList 里面塞 10000 个测试数据，启动两个线程，一个不断的删除元素，一个不断的读取容器中最后一个数据。

```
public void test(){

    for(int i = 0; i<10000; i++){
        list.add("string" + i);
    }

    new Thread(new Runnable() {
        @Override
        public void run() {
            while (true) {
                if (list.size() > 0) {
                    String content = list.get(list.size() - 1);
                }else {
                    break;
                }
            }
        }
    }).start();

    new Thread(new Runnable() {
        @Override
        public void run() {
            while (true) {
                if(list.size() <= 0){
                    break;
                }
                list.remove(0);
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }).start();
```

```
    }).start();
}
```

11.Java 接口可以多继承吗？

java 类是单继承的。classB extends classA

java 接口可以多继承。Interface3 extends Interface0, Interface1, interface……

不允许类多重继承的主要原因是，如果 A 同时继承 B 和 C，而 B 和 C 同时有一个 D 方法，A 如何决定该继承那一个呢？

但接口不存在这样的问题，接口全都是抽象方法继承谁都无所谓，所以接口可以继承多个接口。

12.(byte)300==(byte)100+(short)200?

答：false。

java 中 byte 的取值范围是-128~127，发生上溢下溢时要模 256；130>127 上溢，故模 256 得 130，仍溢出，再减 256 得-126，所以 s=-126。300>127 上溢，故模 256 得 44，44 在 byte 取值范围内，故 s=44.

300 的 二进制是：100101100；byte 强制转换后从右往左取 8 位为：00101100；因为第八位为 0 所以为正数，又知道正数的原反补码都相同；

所以 00101100 转换为十进制是：44（32+8+4）(byte)100+(short)200，byte 和 short 的结果会自动转为 short 不会溢出。所以(byte)100+(short)200=(short)300，而(byte)300 的结果是 44.即两者不相等。

13.操作系统具有进程管理,存储管理,文件管理和设备管理的功能,下列有关描述中,哪一项是不正确的? (A)

A.进程管理主要是对程序进行管理

B.存储管理主要管理内存资源

C.文件管理可以有效的支持对文件的操作，解决文件共享、保密和保护问题

D. 设备管理是指计算机系统中除了 CPU 和内存以外的所有输入输出设备的管理

14.this 和 super 正确的是（C）：

A、都可以用在 main()方法中　B、都是指一个内存地址　C、不能用在 main()方法中　D、意义相同

public static void main(String[] args),main 方法是静态方法，不可以使用对象特有的 this 或 super 关键字。

15.引用计数法是 JVM GC 算法吗？

答：是。

16.能在 try{}catch(){}finally{}结构的 finally{}中再次抛出异常吗？

答：能。

Exception in thread “main” java.lang.Exception：异常 4

at com.test.FinallyTry.f(FinallyTry.java:16)

at com.test.FinallyTry.main(FinallyTry.java:5)

--------在 finally 中抛异常或者 return 会掩盖之前的异常

17.HTTP2 新特性？

答：减少头部的体积、添加请求优先级、服务器推送、多路复用。

18.索引可以将随机 IO 变成顺序 IO 吗？

答：对。

随机 IO：假设我们所需要的数据是随机分散在磁盘的不同页的不同扇区中的，那么找到相应的数据需要等到磁臂（寻址作用）旋转到指定的页，然后盘片寻找到对应的扇区，才能找到我们所需要的一块数据，依次进行此过程直到找完所有数据，这个就是随机 IO，读取数据速度较慢。

顺序 IO：假设我们已经找到了第一块数据，并且其他所需的数据就在这一块数据后边，那么就不需要重新寻址，可以依次拿到我们所需的数据，这个就叫顺序 IO。

19.transient 修饰的变量是临时变量吗？

答：对。

一旦变量被 transient 修饰，变量将不再是对象持久化的一部分，该变量内容在序列化后无法获得访问。

transient 关键字只能修饰变量，而不能修饰方法和类。注意，本地变量是不能被 transient 关键字修饰的。变量如果是用户自定义类变量，则该类需要实现 SERIALIZABLE 接口。

被 transient 关键字修饰的变量不再能被序列化，一个静态变量不管是否被 transient 修饰，均不能被序列化。

注意点：在 Java 中，对象的序列化可以通过实现两种接口来实现，若实现的是 SERIALIZABLE 接口，则所有的序列化将会自动进行，若实现的是 Externalizable 接口，则没有任何东西可以自动序列化，需要在 writeExternal 方法中进行手工指定所要序列化的变量，这与是否被 transient 修饰无关。

20.高、中、低三级调度。

高级调度：即作业调度，按照一定策略将选择磁盘上的程序装入内存，并建立进程。（存在与多道批处理系统中）

中级调度：即交换调度，按照一定策略在内外存之间进行数据交换。

低级调度：即 CPU 调度（进程调度），按照一定策略选择就绪进程，占用 cpu 执行。

其中低度调度是必须的。

下面那个查看 80 端口是否被占用？

方式一：ps -ef |grep 80

方式二：netstat -anp |grep :80

方式三：lsof -i:80

方式四：netstat -tunlp |grep :80

方式五：netstat -an |grep :80

21.TCP 第四次挥手后为什么要等待 2MSL 后才断开链接？等待时间为什么是 2MSL？

答：

1.为了保证客户端最后一次挥手的报文能够到达服务器，若第 4 次挥手的报文段丢失了，服务器就会超时重传第 3 次挥手的报文段，所以客户端此时不是直接进入 CLOSED，而是保持 TIME_WAIT（等待 2MSL 就是 TIME_WAIT）。当客户端再次收到服务器因为超时重传而发送的第 3 次挥手的请求时，客户端就会重新给服务器发送第 4 次挥手的报文（保证服务器能够收到客户端的回应报文）。最后，客户端、服务器才真正断开连接。说白了，等待 2MSL 就是为了确保服务器能够受到客户端最后的回应。

2.如果客户端直接 CLOSED，然后又再次向服务器发起一个新连接，谁也不能保证新发起的连接和刚关闭的连接的端口号是不同的，有可能新、老连接的端口号就是一样的。假设

新、老连接端口号一致，若老连接的一些数据仍滞留在网络中，这些滞留数据在新连接建立后才到达服务器，鉴于前后端口号一致，TCP 协议就默认这些数据属于新连接，于是数据就这样乱成一锅粥了。所以 TCP 连接还要在 TIME_WAIT 状态下等待 2MSL，确保所有老连接的数据都在网络中消失！

3.首先说明什么是 MSL，MSL 是 Maximum Segment Lifetime 的缩写，译为报文最大生存时间，也就是任何报文在网络上存活的最大时间，一旦超过该时间，报文就会被丢弃。2MSL 也就是指的 2 倍 MSL 的时间。 为什么是 2 倍呢？

主动断开的一侧为 A，被动断开的一侧为 B。

第一个消息：A 发 FIN

第二个消息：B 回复 ACK

第三个消息：B 发出 FIN 此时此刻：B 单方面认为自己与 A 达成了共识，即双方都同意关闭连接。此时，B 能释放这个 TCP 连接占用的内存资源吗？不能，B 一定要确保 A 收到自己的 ACK、FIN。所以 B 需要静静地等待 A 的第四个消息的到来：

第四个消息：A 发出 ACK，用于确认收到 B 的 FIN

当 B 接收到此消息，即认为双方达成了同步：双方都知道连接可以释放了，此时 B 可以安全地释放此 TCP 连接所占用的内存资源、端口号。所以被动关闭的 B 无需任何 wait time，直接释放资源。但，A 并不知道 B 是否接到自己的 ACK，A 是这么想的：

1）如果 B 没有收到自己的 ACK，会超时重传 FiN 那么 A 再次接到重传的 FIN，会再次发送 ACK
2）如果 B 收到自己的 ACK，也不会再发任何消息，包括 ACK 无论是 1 还是 2，A 都需要等待，要取这两种情况等待时间的最大值，以应对最坏的情况发生，
这个最坏情况是：
去向 ACK 消息最大存活时间（MSL) + 来向 FIN 消息的最大存活时间(MSL)。这恰恰就是 2MSL(Maximum Segment Life)。等待 2MSL 时间，A 就可以放心地释放 TCP 占用的资源、端口号，此时可以使用该端口号连接任何服务器。同时也能保证网络中老的链接全部消失。

22.进程有那些状态，并简单描述一下？

进程其基本状态有 5 种，即创建状态、就绪状态、运行状态、阻塞状态、终止状态。创建状态:进程在创建时需要申请一个空白 PCB，向其中填写控制和管理进程的信息，完成资源分配。

就绪状态:进程已经准备好，已分配到所需资源，只要分配到 CPU 就能够立即运行。

执行状态:进程处于就绪状态被调度后，进程进入执行状态。

阻塞状态:正在执行的进程由于某些事件而暂时无法运行，进程受到阻塞。

终止状态:进程结束，或出现错误，或被系统终止，进入终止状态，无法再执行。

进程是指计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位。

进程状态是指一个进程的生命周期可以划分为一组状态，这些状态刻画了整个进程，进程状态即体现一个进程的生命状态。

23.创建 NIO 客户端代码

```
package com.cn.niochat;

import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.channels.SocketChannel;
import java.nio.charset.Charset;
import java.util.Scanner;
```

```
/**
 * 用 Java 实现 nio 的客户端
 */
public class NioClient {

    public void start() throws IOException {
        /**
         * 链接服务器端
         */
        SocketChannel socketChannel = SocketChannel.open(new
InetSocketAddress("127.0.0.1",8000));

        //向服务器端发送数据
        //从命令行获取数据，获取键盘的输入
        Scanner scanner = new Scanner(System.in);
        while (scanner.hasNextLine()){
            //获取这一行数据
           String request =  scanner.nextLine();
            //如果有数据，则发送，且数据不为空
            if(request != null && request.length() > 0){
                socketChannel.write(Charset.forName("UTF-
8").encode(request));
            }
        }
    }

    public static void main(String[] args) throws IOException {
        NioClient nioClient = new NioClient();
        nioClient.start();
    }
}
```

24.获取一个类的 class 实例的方法有那些？

（1）.调用运行时类本身的.class 属性

```
Class clazz = String.class;
```

（2）,通过运行时类的对象获取

```
public final Class<?> getClass()是非静态方法.
Person p = new Person();
Class clazz = p.getClass();
```

（3）通过 Class 的静态方法获取:体现反射的动态性

```
String className = “java.util.commons”;
Class clazz = Class.forName(className);
```

（4）通过类的加载器

```
String className = “java.util.commons”;
ClassLoader classLoader = this.getClass().getClassLoader();
Class claz = classLoader.loadClass(className);
```