多线程

1）什么是线程？

2）线程和进程有什么区别？

进程是进程实体的运行过程，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。线程自己基本上不拥有系统资源，只拥有一点在运行中必不可少的资源（如程序计数器，一组寄存器和栈），但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源。**一个线程可以创建和撤销另一个线程，同一个进程中的多个线程之间可以并发执行。多进程**，允许多个任务同时运行；**多线程**，允许单个任务分成不同的部分运行；

进程和程序的区别：进程即运行中的程序，从中即可知，进程是在运行的，程序是非运行的，当然本质区别就是动态和静态的区别。程序可以存在外存中，也可以存在内存中。

进程和线程的关系：

（1）一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。

（2）资源分配给进程，同一进程的所有线程共享该进程的所有资源。

（3）处理机分给线程，即真正在处理机上运行的是线程。

（4）线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。线程是指进程内的一个执行单元,也是进程内的可调度实体.

进程与线程的区别：

（1）**调度**：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位

（2）**并发性**：不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行

（3）**拥有资源**：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源.

（4）**系统开销**：在创建或撤消进程时，由于系统都要为之分配和回收资源，导致系统的开销明显大于创建或撤消线程时的开销。

http://blog.csdn.net/mxsgoden/article/details/8821936 **进程和程序的区别**

3）如何在 Java 中实现线程？

Java 中实现多线程有 3 种方式，基本上采用前两种：

① **继承 Thread 类，重写 run（）方法。**Thread 类本质是 Runnable 接口的一个实例，它代表线程的一个实例。**启动线程的唯一方法是通过 Thread 类的 start（）方法。Start（）方法将启动一个线程，并执行 run（）方法（Thread 中提供的 run（）方法是一个空方法）。这种方式是通过自定义的方式继承 Thread，重写 run（）。注意：执行 start（）后并不是立即执行多线程代码，而是变成可执行状态，什么时候执行是由多线程擦操作系统决定的**

```
    public void run() {
        System. out. println("Thread body"); //线程的函数体
    }
}
public class Test {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread. start();    //开启线程
    }
}
```

②implements Runnable，实现 run（ ）方法。步骤（1）自定义实现 runnable 接口，并实现接口中的 run（ ）（2）创建 Thread 对象，用实现 Runnable 接口的对象作为参数实例化该 Thread 对象（3）调用 Thread 的 start（ ）方法

```
class MyThread implements Runnable{ //创建线程类
    public void run(){
        System. out. println("Thread body");
```

```
    }
}
public class Test {
    public static void main(String[ ] args){
        MyThread thread = new MyThread();
        Thread t = new Thread(thread);
        t. start(); //开启线程
    }
}
```

② 实现 Callable 接口，重写 call（ ）方法

Callable 和 Runnable 的区别（Callable 接口提供了比 Runnable 接口更加强大的功能），体现为 3 点：

（1） Callable 可以在任务结束后提供一个返回值，Runnable 不行

（2） Callable 中 call（ ）可以抛出异常，而 Runnable 的 run（ ）不能

```
mport Java. util. concurrent. * ;
  public class CallableAndFuture {
  //创建线程类
    public static class CallableTest implements Callable < String > {
        public String call( ) throws Exception{
            return "Hello World!";
        }
    }
    public static void main(String[ ] args) {
      ExecutorService threadPool = Executors. newSingleThreadExecutor( );
      //启动线程
      Future < String > future = threadPool. submit( new CallableTest( ) );
      try {
          System. out. println( "waiting thread to finish" );
          System. out. println( future. get( ) ); //等待线程结束,并获取返回结果
      } catch (Exception e) {
        e. printStackTrace( );
      }
    }
  }
```

上述程序的输出结果为：

```
waiting thread to finishHello World!
```

```
    public class Test extends Thread implements Runnable {
        public void run( ) {
            System. out. println( "this is run( )" );
        }
        public static void main( String args[ ]) {
            Thread t = new Thread( new Test( ) );
            t. start( );
        }
    }
```

程序运行结果为：

```
    this is run( )
```

4）用 Runnable 还是 Thread？

总结：在以上 3 种方式中，前两种方式线程执行完后都没有返回值，只有最后一种是带返回值的，**但是一般推荐用 implements Runnable。因为前两种都要重写 run**

（ ），但是继承 Thread 和 Runnable 效果相同但还要重写 Thread 类的其他方法，效率不高

5) Thread 类中的 start() 和 run() 方法有什么区别？

系统调用 start（ ）方法是用来启动线程的，但是调用完 start（ ）线程处于**待运行状态（就绪状态）**，此时等待 JVM 来调度，JVM 通过调用 run（ ）完成实际的操作，run（ ）结束后**线程终止**

如果直接调用 run( )会被当做是一个普通函数调用。即 start( )可以异步的调用 run（ ），**而直接调用 run（ ）是同步的程序中只存在主进程这一个进程无法达到多线程的目的**

6) Java 中 CyclicBarrier 和 CountDownLatch 有什么不同？

7) Java 中的 volatile 变量是什么？

Volatile 是一个类型修饰符，用来修饰被不同线程修改的对象或者变量。用 volatile 修饰的变量，系统每次用到它的时候都是从内存中获取，而不会用到缓存，这样可以保证所有线程在任何时候看到的变量值都是相同的，下面例子中是一个及时终止线程的方法。缺点：Volatile 不能保证原子性，且会阻止编译器优化代码的性能，因此能不用就不用

```
public class MyThread  implements  Runnable {
    private volatile Boolean flag;
    public void stop( ) {
        flag = false;
    }
    public void run( ) {
        while(flag)
            ;//do something
    }
}
```

9) Java 中的同步集合与并发集合有什么区别？

10） 什么死锁，死锁产生的条件？如何避免死锁？

死锁：在两个或多个并发进程中，如果某个进程持有某种资源而又都等待别的进程释放他或者现在保持着的资源，在为改变这种状态前都不能向前推进，称这一组进程产生了死锁，通俗讲就是两个或者多个进程被无限阻塞相互等待的一种状态

产生死锁的条件：

1.互斥：一个资源每次只能被一个进程使用

2.不可抢占：进程已经获得资源，在未使用完之前，不能强行剥夺

3.占有并等待：一个进程因请求资源被阻塞时，对已获得的资源保存不释放

4.环形等待：若干进程之间形成一种首尾相接的循环等待资源关系。

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁。

11) Java 中活锁和死锁有什么区别？

活锁：就是指线程一直处于运行状态，但却是在做无用功，而这个线程本身要完成的任务却一直无法进展。就想小猫追着自己的尾巴咬，虽然一直在咬

却一直没有咬到。活锁的典型例子是某些重试机制导致一个交易（请求）被不断地重试，而每次重试都是失败的（线程在做无用功），这就导致其他失败

的交易无法得到重试的机会（任务无法进展），简单理解：就是一直尝试去获取需要的锁，不断的 try,这种情况下线程并没有阻塞，所以是活的状态，

但是在做无用功。

死锁：两个线程都处于阻塞状态，在等待其他进程释放锁

12） Java 中 synchronized 和 ReentrantLock 有什么不同？

Java 采用 sychronized 和 Lock 两种锁机制实现对某个共享资源的同步

② sychronized 使用的是 wait（）、notify（）、notifyAll（）机制

②Lock 采用的是 JDK5 新增的 Lock 接口以及它的实现类 ReentrantLock（重入锁），可以使用 Condition 进行线程之间的调度，完成 sychronized 实现的所有功能

两者的区别：

（1） 实现方式不同。Sychronized 是通过 JVM 托管实现的，Lock 是通过代码来实现的，Lock 可以实现更精确的线程语义

（2） 性能不一样，JDK5 新增了 Lock 接口的实现类 ReentrantLock，它不仅拥有和 sychronized 相同的内存和语义，还多了锁投票、定时锁、中断锁和等候等。在竞争不是很激烈的时候 sychronized 的性能优于 ReentrantLock；但

是在资源竞争很激烈的时候 sychronized 性能下降非常快 ,而 ReentrantLock 性能基本不变

（3） 锁机制不一样。Sychronized 不会因为出现异常而导致锁没有释放从而引发死锁；而 Lock 需要手动释放锁，并且必须在 finally 块中释放，否则会引发死锁

13） Java 中 ConcurrentHashMap 的并发度是什么？

14）如何在 Java 中创建 Immutable 对象？

优点：使用简单、线程安全、节省内存

当创建了这个对象的实例之后就不允许修改它的值了，Immutable 对象在多线程中更有优势，它不仅可以保证对象的状态不被改变，而且还可以不使用锁机制就能被其他线程共享。

Java 中所有基本类型的包装类都是不可变类，也包括 String，

创建 Immutable 类的原则

（1） 针对成员变量，所有的成员变量都用 private 来修饰；不允许实现成员变量的 set 方法，只提供构造函数（最好提供一个带参数的构造方法可以利用构造函数初始化这些成员变量），一次生成永不改变

（2） 如果一个类成员是不可变变量那么在成员初始化或者使用 get（）获取该参数时，需要通过 clone（）确保类的不可变性

（3） 类中所有的方法都不会被子类覆盖，可以通过把类定义为 final 或者把类中的方法定义为 final 来达到这个目的

（4） 如果有必要，可以使用覆盖 Object 类的 equals（ ）和 hashCode（ ）

15） 单例模式的双检锁是什么？（单例模式中的懒汉模式才需要用到）

利用双重检查加锁（double-checked locking），首先检查是否实例已经创建，如果尚未创建，“才”进行同步。这样以来，只有一次同步，这正是我们想要的效果。

```
public class Singleton {

    //volatile保证，当uniqueInstance变量被初始化成Singleton实例时，多个线程可以正确处理uniqueInstance变量
    private volatile static Singleton uniqueInstance;
    private Singleton() {
    }
    public static Singleton getInstance() {
        //检查实例，如果不存在，就进入同步代码块
        if (uniqueInstance == null) {
            //只有第一次才彻底执行这里的代码
            synchronized(Singleton.class) {
                //进入同步代码块后，再检查一次，如果仍是null，才创建实例
                if (uniqueInstance == null) {
                    uniqueInstance = new Singleton();
                }
            }
        }
        return uniqueInstance;
    }
}
```

16）写出 3 条你遵循的多线程最佳实践

1. 常用的线程池模式以及不同线程池的使用场景
2. 多个线程同时读写，读线程的数量远远□ 于写线程，你认为应该如何解决 并发的问题？你会选择加什么样的锁？
3. JAVA 的 AQS 是否了解，它是□ 嘛的？
4. 除了 synchronized 关键字之外，你是怎么来保障线程安全的？
5. 什么时候需要加 volatile 关键字？它能保证线程安全吗？

6. 线程池内的线程如果全部忙，提交□ 个新的任务，会发□ 什么？队列全部 塞满了之后，还是忙，再提交会发□ 什么？
7. Tomcat 本身的参数你□ 般会怎么调整？
8. synchronized 关键字锁住的是什么东□ ？在字节码中是怎么表示的？在内存中的对象上表现为什么？
9. wait/notify/notifyAll□ 法需不需要被包含在 synchronized 块中？这是为什么？
10. ExecutorService 你□ 般是怎么□ 的？是每个 service 放□ 个还是□ 个项□ □ □ 放□ 个？有什么好处？