1. 线程

需求分析：

什么是线程?

答案：

线程是程序执行的一个路径，负责当前进程中代码程序的执行，一个进程中有一个或多个线程。当一个进程中启动了多个线程去分别执行代码的时候，这个程序就是多线程程序。每一个线程都有自己的局部变量表，程序计数器（指向正在执行的指令指针）以及各自的生命周期，现代操作系统一般不止一个线程在运行，当启动了一个Java虚拟机时，从操作系统开始就会创建一个新的进程（JVM进程），JVM进程中将会派生或者创建很多线程。

2. 时间片

需求分析：

CUP中的时间片，有什么作用?

答案：

时间片即CPU分配给各个程序的时间，每个线程被分配一个时间段，称作它的时间片，即该进程允许运行的时间，使各个程序从表面上看是同时进行的。如果在时间片结束时进程还在运行，则CPU将被剥夺并分配给另一个进程。如果进程在时间片结束前阻塞或结束，则CPU当即进行切换。而不会造成CPU资源浪费。 在宏观上：我们可以同时打开多个应用程序，每个程序并行，同时运行。但在微观上：由于只有一个CPU， 一次只能处理程序要求的一部分，如何处理公平，一种方法就是引入时间片，每个程序轮流执行。

3. 进程和线程

需求分析：

进程和线程的关系是什么?

答案：

1. 一个线程必定属于也只能属于一个进程，而一个进程可以拥有多个线程，并且至少拥有一个线程
2. 属于一个进程的所有线程共享该进程的所有资源
3. 线程被称为轻量级进程，线程间切换代价小，进程间切换代价大
4. 进程是程序的一次执行，线程可以理解为程序中的一个程序片段的执行
5. 每个进程都有独立的内存空间，而线程共享其所属进程的内存空间

4. 线程分类

需求分析:

线程都有哪些分类?

答案:

- 前台线程

  又叫做执行线程、用户线程 这种线程专门用来执行用户编写的代码，地位比较高，JVM是否会停止运行，就是要看当前是否还有前台线程没有执行完，如果还剩下任意一个前台线程没有“死亡”，那么JVM就不能停止！ 例如，执行程序入口的主线程（main），就是一个前台线程，在单线程程序中，main方法执行完，就代 表main线程执行完了，这时候JVM就停止了 例如，我们在主线程创建并启动的新线程，默认情况下就是一个前台线程，用来执行用户编写的代码任务。

- 后台线程

  又叫做守护线程、精灵线程 这种线程是用来给前台线程服务的，给前台线程提供一个良好的运行环境，地位比较低，JVM是否停止运行，根本不关心后台线程的运行情况和状态。 例如，垃圾回收器，其实就一个后台线程，它一直在背后默默的执行着负责垃圾回收的代码，为我们前 台线程在执行用户代码的时候，提供一个良好的内存环境。

5. 并发和并行

需求分析:

描述什么是并发与并行?

答案:

并发:一段时间内，俩个或多个线程，使用一个CPU交替执行，单核cpu，某一时刻只能一个进程执行。但是可以通过不断切换的方式（如时间片轮巡），因为cpu执行速度非常快，所以我们看起来好像是同时进行，实际上是快速交替执行。

并行:是指在同一时刻，俩个或多个线程，各自使用一个CPU，同时进行运行。多个任务同时进行。比如洗澡的时候可以唱歌，多核CPU，多个进程可以运行在不同的物理核心上。每个CPU各司其职执行不同的任务。

6. 多线程

需求分析:

多线程的好处是什么?

答案:

多线程是指在一个程序中同时运行多个线程，这些线程可以并行执行不同的任务或处理同一任务的不同部分。多线程具有以下好处：

1. 提高程序的执行效率：多线程可以让程序同时处理多个任务，从而提高程序的执行效率。尤其是在涉及到大量数据处理或者耗时的操作时，多线程可以极大地缩短程序的执行时间。

2. 提高系统资源利用率：多线程可以充分利用计算机的多核处理器，同时利用计算机的内存、网络、磁盘等资源，从而提高系统资源的利用率。
3. 提高程序的响应速度：多线程可以使程序在处理任务时不会被阻塞，从而提高程序的响应速度，保证程序的实时性。
4. 实现复杂的交互操作：多线程可以在程序中实现复杂的交互操作，例如同时响应多个用户的请求，同时执行多个任务等。

总之，多线程技术可以提高程序的效率、资源利用率和响应速度，从而提高程序的性能和用户体验。

7. 线程启动

需求分析：

请描述start方法和run方法的区别？

答案：

run()方法一般它被我们用来在线程中执行我们的业务逻辑，也称线程体。
start()是用来启动一个线程的，执行该方法之后，线程就会处于**就绪**状态（可执行状态）。

因此，start()方法用于启动一个新的线程，而run()方法用于定义该线程要执行的任务。在多线程编程中，应该重写run()方法以定义线程的具体逻辑，而不是直接调用run()方法。

8.程序分析

需求分析：

分析代码，程序的输出结果是什么

```
public class Test08 {
    public static void main(String[] args) {
        Thread t = new Thread() {
            public void run() {
                String name = Thread.currentThread().getName();
                System.out.println(name);
            }
        };
        t.start();
        t.run();
    }
}
```

答案：

```
main
Thread-0
```

9.打印奇偶数

需求分析：

请编写一个Java程序，创建两个线程，一个打印50以内的奇数，另一个打印50以内的偶数

答案：

方式1：实现Runnable接口

```
public class Test09 {
    public static void main(String[] args) {
        Thread t = new Thread(new Runnable() {
            public void run() {
                for (int i = 1; i <= 50; i += 2) {
                    System.out.println(Thread.currentThread().getName()+"\t"+i);
                }
            }
        },"打印偶数线程");
        Thread t2 = new Thread(new Runnable() {
            public void run() {
                for (int i = 2; i <= 50; i += 2) {
                    System.out.println(Thread.currentThread().getName()+"\t"+i);
                }
            }
        },"打印奇数线程");
        t.start();
        t2.start();
    }
}
```

方式2：继承Thread类

```
public class Test09 {
    public static void main(String[] args) {
        Thread t = new Thread("打印偶数线程"){
            public void run() {
                for (int i = 1; i <= 50; i += 2) {
                    System.out.println(Thread.currentThread().getName()+"\t"+i);
                }
            }
        };
        Thread t2 = new Thread("打印奇数线程"){
            public void run() {
                for (int i = 2; i <= 50; i += 2) {
                    System.out.println(Thread.currentThread().getName()+"\t"+i);
                }
            }
        };
        t.start();
        t2.start();
    }
}
```

10.线程优先级

需求分析：

编写程序，实现创建3个线程。

1. 分别输出1~10之间的数字及对应的输出数字线程名
2. 线程优先级分别为最高、普通、最低
3. 每个线程输出一次后，休眠一秒钟
4. 思考是否每次都是优先级最高的线程先打印输出？

答案：

```
public class Test10_priority {
    public static void main(String[] args) {
        Runnable r = new Runnable() {
            public void run() {
                for (int i = 1; i <= 10; i++) {
                    System.out.println(Thread.currentThread().getName() + ": " + i);
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        };
        Thread max = new Thread(r,"max");
        Thread normal = new Thread(r,"normal");
        Thread low = new Thread(r,"min");
        //设置优先级，默认优先级为正常=5
        max.setPriority(Thread.MAX_PRIORITY);
        max.setPriority(Thread.MIN_PRIORITY);
        max.start();
        normal.start();
        low.start();
    }
}
```

否，原因：将线程优先级设置为最高并不一定就能保证让线程之前先执行，因为线程的运行顺序还受到操作系统和系统负载等因素的影响。线程优先级只是让系统更倾向于先分配时间片给优先级较高的线程。

11.龟兔赛跑

需求分析：

编写程序，使用多线程实现龟兔赛跑游戏。

1. 可以使用随机数取得0~1之间的随机数模拟比赛进程
2. 如果随机数在0~0.3之间代表兔子开始跑,每次跑2米。如果随机数在0.3~1之间代表乌龟开始跑，每次跑1米
3. 总距离为100米，先跑完100米者为胜利者，输出赛跑过程及谁取得冠军

答案：

```
public class Test11_rabbitAndTurtle {
    public static void main(String[] args) {
        RabbitAndTurtle rabbit = new RabbitAndTurtle("兔子");
        RabbitAndTurtle turtle = new RabbitAndTurtle("乌龟");
        rabbit.start();
        turtle.start();
    }
}
class RabbitAndTurtle extends Thread {
    // 代表比赛的状态
    public static boolean flag = true;
    // 代表比赛距离
    public int distance;

    public RabbitAndTurtle(String name) {
        super(name);
    }

    @Override
    public void run() {
        while (flag) {
            double random = Math.random();
            if (random <= 0.3 && "兔子".equals(this.getName())) {
                distance += 2;
                System.out.println("兔子跑了" + distance + "米");
            }
            if (random > 0.3 && "乌龟".equals(this.getName())) {
                distance += 1;
                System.out.println("乌龟跑了" + distance + "米");
            }
            if (distance == 100) {
                System.out.println("恭喜" + this.getName() + "取得冠军");
                flag = false;
            }
        }
    }
}
```

12. 线程状态

需求分析：

线程的状态有哪些?

答案：

- NEW，新建状态，线程被创建出来，但尚未启动时的线程状态；
- RUNNABLE，就绪状态，表示可以运行的线程状态，它可能正在运行，或者是在等待操作系统给它分配 CPU 资源；注：不一定被调用了 start()立刻会改变状态，还有一些准备工作，这个时候的状态是不确定的
- BLOCKED，阻塞等待锁的线程状态，表示处于阻塞状态的线程正在等待监视器锁，比如等待执行 synchronized 代码块或者使用 synchronized 标记的方法；

- WAITING，无限期等待，当线程调用 wait()/join()不加超时时间的方法之后所处的状态，如果没有被唤醒或等待的线程没有结束，那么将一直等待，当前状态的线程不会被分配 CPU 资源和持有锁.
- TIMED_WAITING，计时等待状态，和等待状态（WAITING）类似，它只是多了超时时间，比如调用了有超时时间设置的方法 Object.wait(long timeout) 和 Thread.join(long timeout) 等这些方法时，它才会进入此状态；
- TERMINATED，终止状态，表示线程已经执行完成。

13.输出线程状态

需求分析：

编写程序，实现创建一个线程，并调用方法实现输出这个线程状态所经历的NEW、RUNNABLE、TIMED_WAITING和TERMINATED状态。

答案：

```
public class Test10_state {
    public static void main(String[] args) throws Exception{
        Thread t = new Thread(new Runnable() {
            public void run() {
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        System.out.println("新建t线程的状态：" + t.getState());
        t.start();
        System.out.println("启动t线程后的状态：" + t.getState());
        Thread.sleep(500);
        System.out.println("t线程休眠1s的状态：" + t.getState());
        t.join();
        System.out.println("t线程执行完毕后的状态：" + t.getState());
    }
}
```

输出结果为：

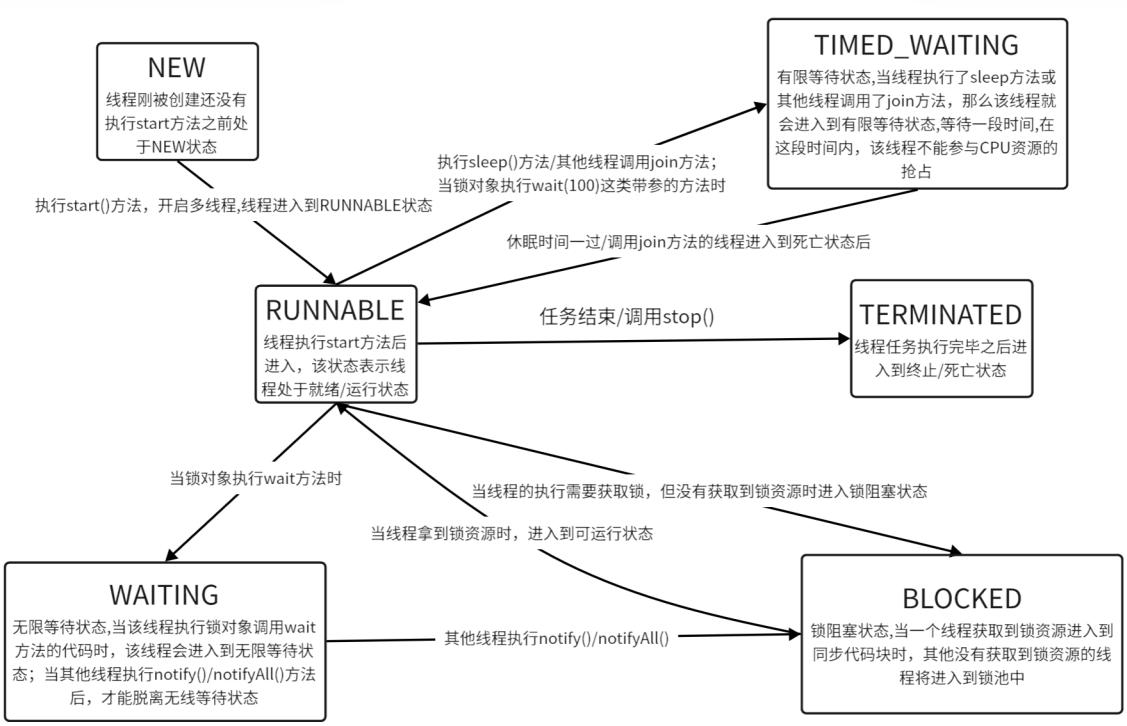
```
新建t线程的状态：NEW
启动t线程后的状态：RUNNABLE
t线程休眠1s的状态：TIMED_WAITING
t线程执行完毕后的状态：TERMINATED
```

14. 线程状态图

需求分析：

画出线程不同的状态及状态转化关系

答案：


NEW
线程刚被创建还没有执行start方法之前处于NEW状态
执行start()方法，开启多线程,线程进入到RUNNABLE状态
TIMED_WAITING
有限等待状态,当线程执行了sleep方法或其他线程调用了join方法，那么该线程就会进入到有限等待状态,等待一段时间,在这段时间内，该线程不能参与CPU资源的抢占
执行sleep()方法/其他线程调用join方法；
当锁对象执行wait(100)这类带参的方法时
休眠时间一过/调用join方法的线程进入到死亡状态后
RUNNABLE
线程执行start方法后进入，该状态表示线程处于就绪/运行状态
任务结束/调用stop()
TERMINATED
线程任务执行完毕之后进入到终止/死亡状态
当锁对象执行wait方法时
当线程的执行需要获取锁，但没有获取到锁资源时进入锁阻塞状态
当线程拿到锁资源时，进入到可运行状态
WAITING
无限等待状态,当该线程执行锁对象调用wait方法的代码时，该线程会进入到无限等待状态；当其他线程执行notify()/notifyAll()方法后，才能脱离无线等待状态
其他线程执行notify()/notifyAll()
BLOCKED
锁阻塞状态,当一个线程获取到锁资源进入到同步代码块时，其他没有获取到锁资源的线程将进入到锁池中