1. 基础概念：

程序:一段静态的代码，一组指令的有序集合，它本身没有任何运行的含义，它只是一个静态的实体，是应用软件执行的蓝本。

进程:是程序的一次动态执行，它对应着从代码加载，执行至执行完毕的一个完整的过程，是一个动态的实体，它有自己的生命周期。它因创建而产生，因调度而运行，因等待资源或事件而被处于等待状态，因完成任务而被撤消。反映了一个程序在一定的数据 集上运行的全部动态过程。通过进程控制块(PCB)唯一的标识某个进程。同时进程占据着相应的资源（例如包括 cpu 的使用 ，轮转时间以及一些其它设备的权限）。是系统进行资源分配和调度的一个独立单位。

程序和进程之间的主要区别在于：

名称	状态	是否具有资源	是否有唯一标识	是否具有并发性
进程	动态	有	有	有
程序	静态	无	无	无

线程: 可以理解为进程的多条执行线索，每条线索又对应着各自独立的生命周期。**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。**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。

并发和并行

并发：虚拟的同时执行多个任务；并行：同时执行多个任务

进程与线程

进程：每个独立运行的程序称为一个进程

线程：程序内部多个任务(顺序控制流)并发执行，每个任务称为一个线程。所以，线程是一个程序内部的顺序控制流

多进程：在操作系统中能同时运行多个程序

多线程：在同一应用程序中有多个顺序流同时执行

线程和进程的区别

进程拥有独立的代码和数据空间，所以进程切换的开销大

同一类的线程共享代码和数据空间，每个线程有独立的运行栈和程序计数器（PC），所以线程切换的开销小。线程又叫作轻量的进程。

2. 线程模型

多线程三个核心：虚拟处理器、执行的代码、处理的数据

A. 虚拟的处理器：

由 java.lang.Thread 类封装和虚拟；

B. 执行的代码

由 Thread 类或者 Runnable 提供的 run()方法实现

C. 处理的数据

run()处理逻辑中处理的数据

3. 多线程的实现

java.lang.Runnable

所有线程对象都必须显示或者隐式的实现该接口

java.lang.Thread

用来对多线程进行控制管理，本身也实现了 Runnable 接口

线程基础方法

public void run()

用来定义线程体，即线程可以完成的任务代码。Runnable 接口定义了该方法，Thread 类实现了 Runnable 接口并重写了该方法

注意：直接调用 run 方法并没有启动新线程，只是将 run 方法对应的线程处理逻辑插入当前线程的顺序执行流中。

public void start()

用来启动一个线程的执行，即执行该线程的 run 方法，Thread 类中定义了该方法。所以所有的线程都必须通过 Thread 的 start()方法启动执行。

注意：多次启动一个线程是非法的。特别是当线程已经结束执行后，不能再重新启动。

4. 基于继承 Thread 创建线程的步骤----**方式一**

A. 继承 Thread 类实现线程类，重写 run()方法，加入线程处理逻辑；

B. 创建 Thread 子类对象；

C. 调用线程类 start()方法，启动线程。

5. 基于实现 Runnable 接口+Thread 创建线程的步骤----**方式二**

A. 实现 Runnable 接口实现线程类，重写 run()方法加入线程处理逻辑；

B. 创建 Runnable 接口实现类的对象；

C. 创建 Thread 类对象，封装 Runnable 接口实现类型对象；

D. 调用 Thread 对象的 start()方法，启动线程。

6. 两种创建线程方式的比较：

A. 直接继承 Thread 类创建线程：

控制逻辑、处理逻辑耦合在一起

编写简单：run()方法的当前对象就是线程对象，可直接操纵

再继承？：线程体所在类不可以继承其他类

B. 使用 Runnable 接口创建线程：

实现线程控制逻辑、处理逻辑的分离

结构清晰：有利于保持程序风格的一致性

再继承？：线程体所在类还可以继承其他类

7. 线程的生命周期

线程的生命周期就是从线程的创建到线程的终止所经历的过程.

注意：start()方法使线程进入就绪状态，但是不立即执行。

8. 线程控制

线程的控制管理是通过 java.lang.Thread 类的相关方法来实现的. 可以查看 API 手册！

方法	功能
setDaemon(Boolean on)	将当前线程设置为设置守护线程

getPriority()/setPriority(int priority)	获得或设置线程的优先级数值
static void sleep(long millis)	让当前线程睡眠指定毫秒数
join()/join(long millis) join(long millis, int nanos)	线程合并：调用某线程的该方法，将当前线程与该线程“合并”，即：等待该线程执行结束后，再恢复当前线程的运行
static void yield()	线程让步：主动暂停当前线程让出 CPU，当前线程进入就绪队列等待调度，并执行其他线程
wait() notify()/notifyAll()	线程等待：当前线程进入对象的 wait pool，并解除对象的锁定 线程通知：唤醒对象的 wait pool 中的一个/所有等待线程
~~suspend()/resume()~~	~~挂起和恢复线程(不解除对象的锁定)。已过时(容易死锁)~~
~~stop()~~	~~线程终止：停止当前线程的执行。已过时(不安全)~~

Thread 类其他常用方法
public final boolean isAlive()
测试线程是否处于活动状态。如果线程已经启动且尚未终止，则为活动状态。
public final void setName(String name)
改变线程名称，使之与参数 name 相同。
public final String getName()
返回该线程的名称。
public static Thread currentThread()
返回对当前正在执行的线程对象的引用。

9. 守护线程

线程分类

主线程（Main Thread） ：JVM 自动程序入口方法 main()所产生的线程

子线程（Sub Thread） ：主线程中创建的线程

用户线程（User Thread）：主线程中创建的用于完成用户指定任务的线程

守护线程（DaemonThread）：在后台运行为其他线程提供服务的线程，也称为后台线程

生命周期：后台线程的生命周期与前台线程生命周期有一定关联。主要体现在：当所有的前台线程都进入死亡状态时，后台线程会自动死亡(其实这个也很好理解，因为后台线程存在的目的在于为前台线程服务的，既然所有的前台线程都死亡了，那它自己还留着有什么用...伟大啊 ！!)。

Thread 类提供的相关方法：

```
public final boolean isDaemon() //测试该线程是否为守护线程
```

public final void setDaemon(Boolean on) //将该线程标记为守护线程。当正在运行的线程都是守护线程时，Java 虚拟机退出。该方法必须在启动线程前调用。

注意：

A.如果主线程是前台，子线程是后台，子线程就算没有完成也可能因为主线程结束而结束！

在这种情况下，主线程可以调用子线程的 join()方法，等待子线程结束后在执行主线程的代码。

B.如果子线程也是前台，两者都结束才会结束！

10.线程优先级

线程优先级指的是线程被线程调度器调度和执行的优先级别。调度器倾向于让优先级高的线程先执行，但并不是优先级低的线程得不到执行，仅仅是执行的频率较低。

注意事项

A.线程的优先级用数字来表示，范围从 1 到 10。

B.主线程的缺省优先级是 5，子线程的优先级默认与其父线程相同。

Thread 类的相关方法和常量

```
public final int getPriority();
public final void setPriority(int newPriority)
Thread.MIN_PRIORITY = 1
Thread.NORM_PRIORITY = 5
Thread.MAX_PRIORITY = 10
```

11.线程合并

又称为线程串行化是指把两个或者多个线程合并为一条顺序执行的流程。在多线程程序中，**如果在一个线程运行的过程中要用到另一个线程的运行结果，则可进行线程的串行化处理。**

Thread 类提供的相关方法：

```
public final void join()
public final void join(long millis)
public final void join(long millis,int nanos)
```

让一个线程等待另一个线程完成才继续执行。如 A 线程线程执行体中调用 B 线程的 join()方法，则 A 线程被阻塞，直到 B 线程执行完为止，A 才能得以继续执行。

12.线程休眠

线程休眠是指暂停执行当前运行中的线程，使之进入阻塞状态，待经过指定的“延迟时间”后再醒来并转入到就绪状态。

Thread 类提供的相关方法：

```
public static void sleep(long millis)
public static void sleep(long millis, int nanos)
```

13.线程让步

线程让步是指让运行中的线程主动放弃当前获得的 CPU 处理机会，但不是使该线程阻塞，而是使之转入就绪状态，在就绪队列中等待，随时可能再次获得运行机会.

Thread 类提供的相关方法：

public static void yield()

14. ~~线程挂起和恢复---过时方法~~

~~线程挂起——暂时停止当前运行中的线程，使之转入阻塞状态，并且不会自动恢复运行。挂起的线程并不释放其锁定的资源~~

~~线程恢复——使得一个已挂起的线程恢复运行。~~

~~Thread 类提供的相关方法：~~

~~public final void suspend()~~

~~public final void resume()~~

~~注意：suspend()和 resume()方法已经不再提倡使用，因为挂起的线程并不释放锁定的资源，容易造成死锁.~~

15. 线程的等待和通知

线程等待——暂时停止当前运行中的线程，使之转入阻塞状态。**等待的线程将释放其锁定的资源**

线程通知——通知在等待队列中的线程恢复运行。

Thread 类提供的相关方法：

public final void wait()

public final void notify()

Public final void notifyAll()

16. 临界资源(Critical Resource)

多个线程共享的数据称为临界资源

由于线程的调度由线程调度器负责，程序员无法精确控制多线程的交替顺序。这种情况下，**多线程对临界资源的方法有时会导致数据的不一致性问题**，这样的问题称为临界资源问题----线程不安全问题.

17. 互斥锁：

在 Java 语言中，为了保证共享数据操作的完整性，引入了对象互斥锁.

互斥锁

每个对象都对应于一个可称为“互斥锁(监视器)”的标记，**这个标记用来保证在任一时刻，只能有一个线程访问该对象。**这样的互斥锁会降低程序的执行效率，有时也会引起死锁，所以 Java 对象默认是不启用互斥锁的.

互斥锁的实现

关键字 synchronized 来与对象的对象在任一时刻只能由一个线程访问.

synchronized 用法

A. 修饰方法：在当前方法执行的过程中，该方法所属的对象（当前对象）被当前线程锁定

B. 修饰语句块：**此时要标记一个对象**，并修饰一段语句块。在当前语句被执行的过程中，标记的对象（不一定是当前对象）被当前线程锁定

解决不安全问题：

(1)同步： ----方法的同步 synchronized 关键字修饰方法

----对象的同步 synchronized（object）{}； //通常在方法内部

(2) 同步 run 方法 public sysnchronized void run(){};

(3) lock: lock();方法和 unlock();方法 ----需要进行属性的声明，并且实例化一个实例对象

// 显示定义 Lock 同步锁对象，此对象与共享资源具有一对一关系

private final Lock lock = new ReentrantLock();

```
public void m(){
        // 加锁
        lock.lock();
        //...    需要进行线程安全同步的代码
      // 释放 Lock 锁
        lock.unlock();
  }
```

18.死锁

并发运行的多个线程间彼此等待对方锁定的临界资源、都无法运行的状态称为线程死锁.

死锁产生四个必要条件：

- **互斥条件 某一资源在一段时间内只能由一个进程占用 不能同时被两个或两个以上的进程占用**
- **不可抢占条件 进程所获的的资源在未使用完之前，自愿申请者不能强行从资源占有者手中夺**
- **占有且申请条件 进程至少占有一个资源 但又申请新的资源由于该资源已被另外进程占有，此时进程阻塞**
- **循环等待 形成一个进程循环等待圈**

进入死锁:打破死锁产生的四个条件其中之一 ！ //破解死锁！

预防死锁:在死锁产生前 提前预知资源抢占 死锁的预防是保证系统不进入死锁状态的一种策略。它的基本思想是**要求进程申请资源时遵循某种协议，从而打破产生死锁的四个必要条件中的一个或几个，保证系统不会进入死锁状态。**

19.线程同步

为避免死锁，在线程进入阻塞状态时应尽量释放其锁定的资源，以为其他的线程提供运行的机会，称为线程同步

相关方法

```
public final void wait()
public final void notify()/notifyAll()
```

20.生产者消费者问题：例如**存钱取钱程序**！流水线场景、收快递场景

“生产者-消费者-仓储”模型，离开了仓储，生产者消费者模型就显得没有说服力了。对于此模型：

仓库有数据属性和仓库状态标识属性(例如，isFull 属性)，生产者和消费者操作仓库之前，都要判断仓库状态(true 还是 false)。最好将生产与消费的方法封装到仓库里，并使用同步关键字！

A.生产者仅仅在仓储未满时候生产，仓满则停止生产。 ---isFull 判断仓库状态，满仓则调用 wait();

B.消费者仅仅在仓储有产品时候才能消费，仓空则等待。 ---isFull 判断仓库状态，苍空则调用 wait();

C.当消费者发现仓储没产品可消费时候会通知生产者生产。 ---空仓 notifyAll()生产者;

D.生产者在生产出可消费产品时候，应该通知等待的消费者去消费。 ---满仓 notifyAll()消费者;

22.A.main 方法的线程 id 是 1,优先级是 5.

B.如何让线程进入就绪状态--创建线程对象 调用 Start（）方法

C. 如何手动停止一个线程;

(1)使用 Boolean 型的变量，作为线程的一个属性

(2)线程 run 方法中，进行这个属性值的判断，如过条件满足继续执行，

不满足则退出，通过改变 Boolean 型的变量来控制这个线程

D.让线程进入后台运行：setDaemon(true) ;然后再 start() ;

小结：

A.一类两接口：

继承 Thread 类,

实现 Runnable 接口(无返回值)，实现 Callable 接口(有返回值)

B.线程不安全与死锁;

C.生产者消费者问题。