**多线程**

## 一、进程

1 进程是操作系统结构的基础；是一次程序的执行；是一个程序及其数据在处理机上顺序执行时所发生的活动。操作系统中，几乎所有运行中的任务对应一条进程（Process）。一个程序进入内存运行，即变成一个进程。进程是处于运行过程中的程序，并且具有一定独立功能。描述进程的有一句话非常经典——进程是系统进行资源分配和调度的一个独立单位。

2 进程是系统中独立存在的实体，拥有自己独立的资源，拥有自己私有的地址空间。进程的实质，就是程序在多道程序系统中的一次执行过程，它是动态产生，动态消亡的，具有自己的生命周期和各种不同的状态。进程具有并发性，它可以同其他进程一起并发执行，按各自独立的、不可预知的速度向前推进。　（注意，并发性（concurrency）和并行性（parallel）是不同的。并行指的是同一时刻，多个指令在多台处理器上同时运行。并发指的是同一时刻只能有一条指令执行，但多个进程指令被被快速轮换执行，看起来就好像多个指令同时执行一样。）

3 进程由程序、数据和进程控制块三部分组成。

## 二、线程

**1 线程，有时被称为轻量级进程(Lightweight Process，LWP）**，是程序执行流的最小单元。一个标准的线程由线程ID，当前指令指针(PC），寄存器集合和堆栈组成。另外，线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点儿在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源。一个线程可以创建和撤消另一个线程，同一进程中的多个线程之间可以并发执行。由于线程之间的相互制约，致使线程在运行中呈现出间断性。每一个程序都至少有一个线程，若程序只有一个线程，那就是程序本身。

**2 线程是程序中一个单一的顺序控制流程**。在单个程序中同时运行多个线程完成不同的工作，称为多线程。

**3 在Java Web中要注意，线程是JVM级别的**，在不停止的情况下，跟JVM共同消亡，就是说如果一个Web服务启动了多个Web应用，某个Web应用启动了某个线 程，如果关闭这个Web应用，线程并不会关闭，因为JVM还在运行，所以别忘了设置Web应用关闭时停止线程。

**4 线程的三种创建方式**

1.通过继承Thread类实现

2.通过实现Runnable接口

3.实现Callable接口

第1种方式无法继承其他类，第2，3种可以继承其他类；

第2，3种方式多线程可以共享同一个target对象，多个线程处理同一个资源；

一般使用第2，3种方式创建线程。

**5 线程的生命周期**

1.新建（new） 2.就绪(start) 3.运行(获得cpu资源) 4.阻塞（sleep,IO阻塞等）4.死亡（执行完成，Exception/Error）

**6 线程常用方法**

join() ：Thread对象调用，线程A调用join()，其他线程被阻塞，直到线程A执行完为止。

setDaemon(true) ： Thread对象调用，设置成后台线程； 当所有前台线程都死亡，后台线程自动死亡。

sleep(): 静态方法，让正在执行的线程暂停，进入阻塞状态。

yield(): 静态方法，让正在进行的线程暂停，进入就绪状态，系统的线程调度器重新调度一次；只有优先级比当前调用yield()的线程高或相同，并且处于就绪状态的线程才能获得执行。

setPriority(int newPriority), getPriority(): 设置和获取线程的优先级；newPriority范围1-10；Thread三个静态常量，MAX\_PRIORITY, 10; NORM\_PRIORITY, 5; MIN\_PRIORITY, 1；子线程和父线程具有相同的优先级，优先级高的线程比优先级低的线程执行机会更多。

**7 线程同步**

多个线程访问同一个数据时（线程调度的不确定性），很容易出现线程安全问题。解决办法：引入同步监视器，任意时刻只能有一个线程获得对同步监视器的锁定，当同步代码块执行结束后，该线程释放对该同步监视器的锁定。

“加锁”—>”修改共享资源”->”释放锁”

同步代码块

// obj就是同步监视器synchronized(obj){

同步代码块

}

**8 同步方法**

使用synchronized修饰某个方法，则该方法称为同步方法，同步方法的同步监视器是this;

只需对会改变竞争资源的方法进行同步。任何线程在进入同步代码块或同步方法前，必须获得同步监视器的锁定。 释放同步监视器的锁定的时机：

1.当前线程的同步代码块，同步方法执行结束

2.当前线程在同步代码块，同步方法中遇到break,return终止了执行

3.当前线程在同步代码块，同步方法中出现了未处理的Error或Exception

4.执行了同步监视器对象的wait()方法

**9 同步锁（Lock）**

Lock对共享资源的独占访问，每次只能一个线程对Lock对象加锁。

Lock,ReadWriteLock接口，对应的实现类ReentrantLock（可重入锁）,ReentrantReadWriteLock。

class A{

private final ReentrantLock lock = ReentrantLock();

public void method(){

// 加锁

lock.lock();

try{

// 修改共享资源

}finally{

// 释放锁

lock.unlock();

}

}

}



**10 死锁**

两个线程互相等待对方释放同步监视器，就发生了死锁。

**11 线程池**

在系统启动时，创建大量空闲的线程，将一个Runnable对象或Callable对象传给线程池，线程池会启动一个线程执行对应的run()或call()，当run()或call()执行完成后，该线程返回到线程池成为空闲状态，等待下个Runnable对象或Callable对象。

**12 创建线程池**

**Executors工厂类，提供如下静态方法创建不同的线程池：**

// 具有缓存功能的线程池，系统根据需要创建线程，这些线程会被缓存在线程池中newCachedThreadPool()// 固定数量，可重用的线程池newFixedThreadPool(int nThreads)// 单线程的线程池newSingleThreadExecutor()

// 指定线程数量，并可指定延迟时间才执行线程任务newScheduleThreadPool(int corePoolSize)// 单线程，并可指定延迟时间才执行线程任务newSingleThreadScheduleExecutor()

前三个静态方法return ExecutorService

后两个静态方法return ScheduledExecutorService

**ExecutorService代表线程池，提供3个方法：**

// 将一个Runnable对象提交给线程池，Future返回nullFuture<?> submit(Runnable task)// 将一个Runnable对象提交给线程池，Future返回指定结果resultFuture<T> submit(Runnable task, T result)// 将一个Callable对象提交给线程池，Future返回Callable对象中call()方法Future<T> submit(Callable<T> task)

**步骤：**

1.Executors静态工厂类创建ExecutorService

2.创建Runnable或Callable对象，作为线程执行体

3.调用ExecutorService实例的submit()方法提交Runnable或Callable

4.线程池关闭，调用ExecutorService实例的shutdown()方法

**ThreadLocal**

线程局部变量，把数据放在ThreadLocal中，就可以为每个线程创建一个该变量的副本，避免并发访问的线程安全问题；

private ThreadLocal<T> threadLocal = new ThreadLocal<>();

// 当前线程副本中的值

T get();// 删除void remove();// 设置当前线程副本中的值void set(T value);

**13 与同步机制的区别**

同步机制是为了同步多个线程对共享资源的并发访问，是多线程间通信的有限方式；

ThreadLocal隔离多个线程的数据共享，避免多个线程间对共享资源的竞争，不需要对多个线程进行同步。

线程安全集合

通过Collections包装成线程安全集合：

ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap等都是线程不安全；如果多个线程对这些集合读，写时，会破坏这些集合数据的完整性。

Collections提供静态方法将这些集合包装成线程安全的集合。

synchronizedCollection(Collection<T> c)

synchronizedList<List<T> list>

synchronizedMap(Map<K, V> map)

synchronizedSet(Set<T> set)

HashMap m = Collections.synchronizedMap(new HashMap);

**14 线程安全集合**

ConcurrentHashMap,

ConcurrentSkipListMap,

ConcurrentSkipListSet,

ConcurrentLinkedQueue,

ConcurrentLinkedDeque

CopyOnWriteArrayList

CopyOnWriteArraySet

三、线程状态 



 线程共包括以下5种状态。  
**1. 新建状态(New)**        : 线程对象被创建后，就进入了新建状态。此时它和其他Java对象一样，仅仅由Java虚拟机分配了内存，并初始化其成员变量值。

**2. 就绪状态(Runnable):** 也被称为“可执行状态”。线程对象被调用了该对象的start()方法，该线程处于就绪状态。Java虚拟机会为其创建方法调用栈和程序计数器。处于就绪状态的线程，随时可能被CPU调度执行，取决于JVM中线程调度器的调度。

**3. 运行状态(Running) :** 线程获取CPU权限进行执行。需要注意的是，线程只能从就绪状态进入到运行状态。

**4. 阻塞状态(Blocked)  :** 阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：  
    (01) 等待阻塞 -- 通过调用线程的wait()方法，让线程等待某工作的完成。  
    (02) 同步阻塞 -- 线程在获取synchronized同步锁失败(因为锁被其它线程所占用)，它会进入同步阻塞状态。  
    (03) 其他阻塞 -- 通过调用线程的sleep()或join()或发出了I/O请求时，线程会进入到阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

**5. 死亡状态(Dead)    :** 线程执行完了、因异常退出了run()方法或者直接调用该线程的stop()方法（容易导致死锁，现在已经不推荐使用），该线程结束生命周期。

## 四、wait()、notify()、nofityAll()方法

　　在Object.java中，定义了wait(), notify()和notifyAll()等方法。

　　wait()的作用是让当前线程进入等待状态，同时，wait()也会让当前线程释放它所持有的锁。

　　而 notify()和notifyAll()的作用，则是唤醒当前对象上的等待线程；notify()是唤醒单个线程，而notifyAll()是唤醒所有的线程。

Object类中关于等待/唤醒的API详细信息如下：

**notify()**

-- 唤醒在此对象监视器上等待的单个线程，使其进入“就绪状态”。

**notifyAll()**

**-**- 唤醒在此对象监视器上等待的所有线程，使其进入“就绪状态”。

**wait()**

-- 让当前线程处于“等待(阻塞)状态”，“直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法”，当前线程被唤醒(进入“就绪状态”)。

**wait(long timeout)**

-- 让当前线程处于“等待(阻塞)状态”，“直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者超过指定的时间量”，当前线程被唤醒(进入“就绪状态”)。

**wait(long timeout, int nanos)**

**--** 让当前线程处于“等待(阻塞)状态”，“直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量”，当前线程被唤醒(进入“就绪状态”)。

**wait()**的作用是让“当前线程”等待（会释放锁），而“当前线程”是指正在cpu上运行的线程！

五、yield()、sleep()、join()和interrupt()方法

**1、yield()**

　 yield()是Thread类的静态方法。它能让当前线程暂停，但不会阻塞该线程，而是由“运行状态”进入到“就绪状态”，从而让 其它具有相同优先级的等待线程获取执行权；但是，并不能保证在当前线程调用yield()之后，其它具有相同优先级的线程就一定能获得执行权；也有可能是 当前线程又进入到“运行状态”继续运行！值得注意的是，yield()方法不会释放锁。

**2、sleep()**

　　sleep()是Thread类的静态方法。该方法声明抛出了InterrupedException异常。所以使用时，要么捕捉，要么声明抛出。

　　有2种重载方式：

——static void sleep(long millis)　　:　　让当前正在执行的线程暂停millis毫秒，并进入阻塞状态，该方法受到系统计时器和线程调度器的精度和准度的影响。

——static void sleep(long millis , int nanos)　　：　　让当前正在执行的线程暂停millis毫秒加nanos微秒，并进入阻塞状态，该方法受到系统计时器和线程调度器的精度和准度的影响。

　　sleep() 的作用是让当前线程休眠，即当前线程会从“[运行状态](http://www.cnblogs.com/skywang12345/p/3479024.html)”进入到“[休眠(阻塞)状态](http://www.cnblogs.com/skywang12345/p/3479024.html)”。sleep()会指定休眠时间，线程休眠的时间会大于/等于该休眠时间；在线程重新被唤醒时，它会由“[阻塞状态](http://www.cnblogs.com/skywang12345/p/3479024.html)”变成“[就绪状态](http://www.cnblogs.com/skywang12345/p/3479024.html)”，从而等待cpu的调度执行。常用来暂停程序的运行。

　　同时注意，sleep()方法不会释放锁。

**3、join()**

　　join() 是Thread的一个实例方法。表示，当某个程序执行流中调用其他线程的join方法时，调用线程将被阻塞，直到被join的线程执行完毕。

有3种重载的形式：

**——join()　　:**　等待被join的线程执行完成

**——join(long millis)　　：**　等待被join的线程的时间最长为millis毫秒，若在millis毫秒内，被join的线程还未执行结束，则不等待。

**——join(long millis , int nanos)　　:**　等待被join的线程的时间最长为millis毫秒加nanos微秒，若在此时间内，被join的线程还未执行结束，则不等待。

即当前线程内，用某个线程对象调用join()后，会使当前线程等待，直到该线程对象的线程运行完毕，原线程才会继续运行。

**4、interrupt()**

　　我们经常通过判断线程的中断标记来控制线程。

　　interrupt()是Thread类的一个实例方法，用于中断本线程。这个方法被调用时，会立即将线程的中断标志设置为“true”。所以当中断处于“阻塞状态”的线程时，由于处于阻塞状态，中断标记会被设置为“false”，抛出一个 InterruptedException。所以我们在线程的循环外捕获这个异常，就可以退出线程了。

　　interrupt()并不会中断处于“运行状态”的线程，它会把线程的“中断标记”设置为true，所以我们可以不断通过isInterrupted()来检测中断标记，从而在调用了interrupt()后终止线程，这也是通常我们对interrupt()的用法。

　　Interrupted()是Thread类的一个静态方法，它返回一个布尔类型指明当前线程是否已经被中断，isInterrupted()是Thread类的实例方法，返回一个布尔类型来判断线程是否已经被中断。它们都能够用于检测对象的“中断标记”。区别是，interrupted()除了返回中断标记之外，它还会清除中断标记(即将中断标记设为false)；而isInterrupted()仅仅返回中断标记。

六、 Synchronized关键字

**1、原理**

在java中，每一个对象有且仅有一个同步锁。这也意味着，同步锁是依赖于对象而存在。当当前线程调用某对象的synchronized方法时，就获取了该对象的同步锁。例如，synchronized(obj)，当前线程就获取了“obj这个对象”的同步锁。不同线程对同步锁的访问是互斥的。也就是说，某时间点，对象的同步锁只能被一个线程获取到！通过同步锁，我们就能在多线程中，实现对“对象/方法”的互斥访问。 例如，现在有个线程A和线程B，它们都会访问“对象obj的同步锁”。假设，在某一时刻，线程A获取到“obj的同步锁”并在执行一些操作；而此时，线程B也企图获取“obj的同步锁” —— 线程B会获取失败，它必须等待，直到线程A释放了“该对象的同步锁”之后线程B才能获取到“obj的同步锁”从而才可以运行。

**2、基本规则**

第1条 :  当一个线程访问“某对象”的“synchronized方法”或者“synchronized代码块”时，其他线程对“该对象”的该“synchronized方法”或者“synchronized代码块”的访问将被阻塞。  
第2条 :  当一个线程访问“某对象”的“synchronized方法”或者“synchronized代码块”时，其他线程仍然可以访问“该对象”的非同步代码块。  
第3条 :  当一个线程访问“某对象”的“synchronized方法”或者“synchronized代码块”时，其他线程对“该对象”的其他的“synchronized方法”或者“synchronized代码块”的访问将被阻塞。

**3、实例锁和全局锁**

**实例锁 --** 锁在某一个实例对象上。如果该类是单例，那么该锁也具有全局锁的概念。  
 实例锁对应的就是synchronized关键字。

**全局锁 --** 该锁针对的是类，无论实例多少个对象，那么线程都共享该锁。  
全局锁对应的就是static synchronized（或者是锁在该类的class或者classloader对象上）。

就是说，一个非静态方法上的synchronized关键字，代表该方法依赖其所属对象。一个静态方法上synchronized关键字，代表该方法依赖这个类本身。

## 七、线程优先级和守护线程

**1、线程优先级**

java中的线程优先级的范围是1～10，默认的优先级是5。每个线程默认的优先级都与创建它的父线程具有相同的优先级。默认情况下，mian线程具有普通优先级。“高优先级线程”会优先于“低优先级线程”执行。Thread提供了setPriority(int newPriority)和getPriority()方法来设置和返回线程优先级。

　　Thread类有3个静态常量：

——MAX\_PRIORITY = 10

——MIN\_PRIORITY = 1

——NORM\_PRIORITY = 5

**2、守护线程**

java 中有两种线程：用户线程和守护线程。可以通过isDaemon()方法来区别它们：如果返回false，则说明该线程是“用户线程”；否则就是“守护线程”。用户线程一般用户执行用户级任务，而守护线程也就是“后台线程”，一般用来执行后台任务。需要注意的是：Java虚拟机在“用户线程”都结束后会后退出。守护线程又称“后台线程”、“精灵线程”，它有一个特征——如果所有前台线程都死亡，后台线程自动死亡。通过setDaemon(true)来设置一个线程。