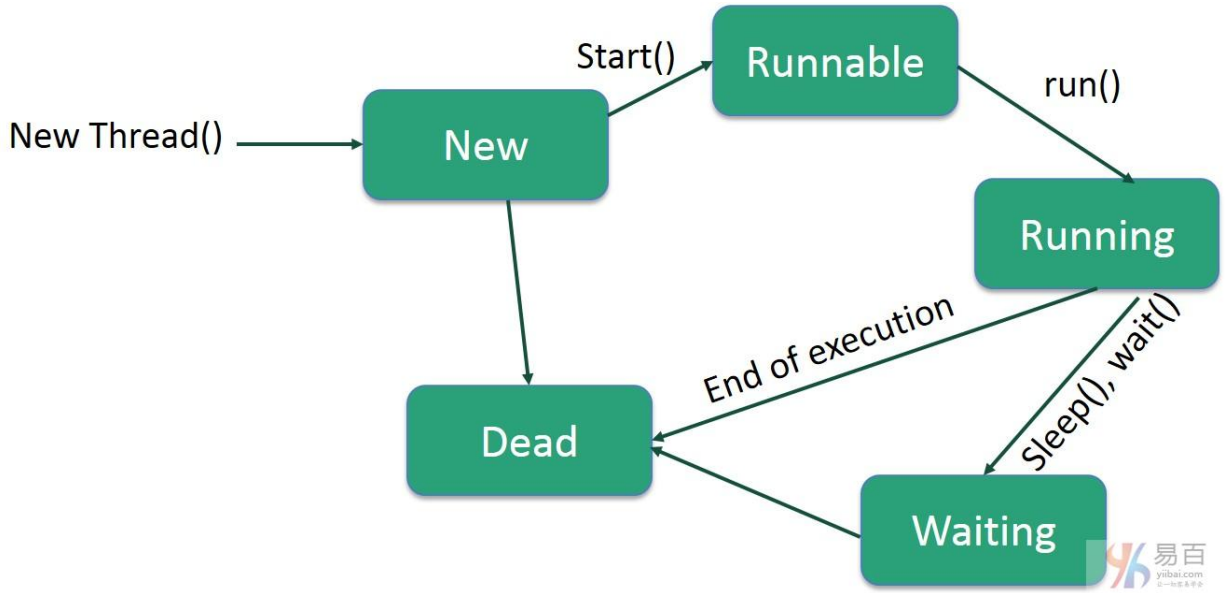
Java是一种多线程编程语言，我们可以使用Java来开发多线程程序。 多线程程序包含两个或多个可同时运行的部分，每个部分可以同时处理不同的任务，从而能更好地利用可用资源，特别是当您的计算机有多个CPU时。多线程使您能够写入多个活动，可以在同一程序中同时进行操作处理。

根据定义，多任务是当多个进程共享，如CPU处理公共资源。 多线程将多任务的概念扩展到可以将单个应用程序中的特定操作细分为单个线程的应用程序。每个线程可以并行运行。操作系统不仅在不同的应用程序之间划分处理时间，而且在应用程序中的每个线程之间划分处理时间。

多线程能够在同一程序同中，进行多个活动的方式进行写入。

线程的生命周期

线程在其生命周期中经历各个阶段。 例如，线程诞生，启动，运行，然后死亡。 下图显示了线程的完整生命周期。



以下是线程生命周期的阶段 -

- **新线程(New) - 新线程在新的状态下开始其生命周期。直到程序启动线程为止，它保持在这种状态。它也被称为出生线程。**
- **可运行(Runnable) - 新诞生的线程启动后，该线程可以运行。状态的线程被认为正在执行其任务。**
- **等待(Waiting) - 有时，线程会转换到等待状态，而线程等待另一个线程执行任务。 只有当另一个线程发信号通知等待线程才能继续执行时，线程才转回到可运行状态。**
- **定时等待(Timed Waiting) - 可运行的线程可以在指定的时间间隔内进入定时等待状态。 当该时间间隔到期或发生等待的事件时，此状态的线程将转换回可运行状态。**
- **终止(Dead) - 可执行线程在完成任务或以其他方式终止时进入终止状态。**

线程优先级

每个Java线程都有一个优先级，可以帮助操作系统确定安排线程的顺序。Java线程优先级在MIN_PRIORITY(常数为1)和MAX_PRIORITY(常数为10)之间的范围内。 默认情况下，每个线程都被赋予优先级NORM_PRIORITY(常数为5)。

具有较高优先级的线程对于一个程序来说更重要，应该在低优先级线程之前分配处理器时间。 然而，线程优先级不能保证线程执行的顺序，并且依赖于平台。

通过实现Runnable接口创建一个线程

如果想让一个类作为线程执行，那么您可以通过实现Runnable接口来实现此目的。 您将需要遵循三个基本步骤 -

第1步

在第一步中，您需要实现由Runnable接口提供的run()方法。 该方法为线程提供了一个入口点，您可将把完整的业务逻辑放在此方法中。 以下是run()方法的简单语法 -

```
public void run( )
```

第2步

在第二步中，您将使用以下构造函数实例化一个Thread对象 -

```
Thread(Runnable threadObj, String threadName);
```

其中，threadObj是实现Runnable接口的类的实例，threadName是给予新线程的名称。

第3步

当创建了一个线程对象，可以通过调用start()方法启动它，该方法执行对run()方法的调用。 以下是一个简单的语法start()方法 -

```
void start();
```

实例

这是一个创建一个新线程并开始运行的示例 -

```
class RunnableDemo implements Runnable {
   private Thread t;
   private String threadName;

   RunnableDemo( String name) {
      threadName = name;
      System.out.println("Creating " +  threadName );
   }

   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
      }catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");
      }
      System.out.println("Thread " +  threadName + " exiting.");
   }
```

```
   public void start () {
      System.out.println("Starting " +  threadName );
      if (t == null) {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}

public class TestThread {

   public static void main(String args[]) {
      RunnableDemo R1 = new RunnableDemo( "Thread-1");
      R1.start();

      RunnableDemo R2 = new RunnableDemo( "Thread-2");
      R2.start();
   }
}
```

这将产生以下结果 -

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

通过扩展Thread类创建一个线程

创建线程的第二种方法是创建一个新类，使用以下两个简单的步骤来扩展Thread类。 这种方法在处理使用Thread类中可用的方法创建的多个线程时提供了更多的灵活性。

第1步

需要覆盖Thread类中的run()方法。 该方法为线程提供了一个入口点，您将把完整的业务逻辑放在此方法中。 以下是run()方法的简单语法 -

```
public void run( )
```

第2步

当创建了Thread对象，您可以通过调用start()方法启动它，该方法执行对run()方法的调用。以下是一个简单的语法start()方法 -

```
void start( );
```

示例

这是上面示例程序中重写扩展Thread代码如下 -

```
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;

   ThreadDemo( String name) {
      threadName = name;
      System.out.println("Creating " +  threadName );
   }

   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
      }catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");
      }
      System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start () {
      System.out.println("Starting " +  threadName );
      if (t == null) {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
```

```
public class TestThread {

   public static void main(String args[]) {
      ThreadDemo T1 = new ThreadDemo( "Thread-1");
      T1.start();

      ThreadDemo T2 = new ThreadDemo( "Thread-2");
      T2.start();
   }
}
```

运行上面代码，这将产生以下结果 -

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```