1.多线程

1.1 进程和线程

进程: 正在运行的程序就是一个进程,单一的顺序控制流,有独立的代码运行空间。

线程: 轻量级的进程,单一的顺序控制,有独立的代码运行空间。

进程之间的通信开销比较大,线程之间的通信(进程内部的通信)开销比较小。

1.2 创建线程、多线程

1.2.1 创建线程

- 1. 实现Runnable接口, 重写run方法(业务)
- 2. 创建线程对象
- 3. 启动线程 start();

```
public class MyRunnable implements Runnable {
    @Override
    public void run() {
        // TO DO 业务
        System.out.println("子线程: " + Thread.currentThread().getName() + ":" +
    new Date());
    }
}
```

```
public class Test {

   public static void main(String[] args) {

      // 获取当前线程的名称
      System.out.println("主线程:" + Thread.currentThread().getName());

      MyRunnable mr = new MyRunnable();
      Thread t = new Thread(mr);
      t.start();

}
```

1.2.2创建多线程

```
public class MyRunnable implements Runnable {
    @override
    public void run() {
        // TO DO 业务
        System.out.println("子线程: " + Thread.currentThread().getName() + ":" +
    new Date());
    }
}
```

```
public class Test {

public static void main(String[] args) {

// 业务接口

MyRunnable mr = new MyRunnable();

// 启动多线程

Thread t1 = new Thread(mr, "A");
 Thread t2 = new Thread(mr, "B");
 Thread t3 = new Thread(mr, "C");

t1.start(); // 告诉 JVM 我们需要一个子线程,由JVM向CPU去申请一个线程 t2.start();
 t3.start();

// 现在在执行后会自动销毁

// t1.start(); // 抛出异常 IllegalThreadStateException
}
```

线程的执行顺序和线程的启动顺序无关。

```
> 🖿 out
                        15
                                      // 启动多线程
∨ src
   ∨ 🖿 com
                                      Thread t1 = new Thread(mr,
                        16
     ∨ 🖿 iweb
                        17
                                      Thread t2 = new Thread(mr,
                                                                "B")
       ∨ 🖿 lesson01
                                      Thread t3 = new Thread(mr,
                                                                "C")
                        18
           c MyRunnable 19
            c Test
                                       t1.start(): // 告诉 JVM 我们需要一个子线程,由JVM向CPU去申请一个线料
                        20
Jn: \equiv Test (1) 	imes
D:\too<u>ls</u>\java\jdk1.8.0_181\bin\java.exe ...
                                                              线程的执行顺序程序员不能指定
     子线程: C:Thu Apr 29 10:51:43 CST 2021
     子线程: B:Thu Apr 29 10:51:43 CST 2021
1 1
     子线程: A: Thu Apr 29 10:51:43 CST 2021
ķ ↓
```

使用Thread类来创建线程:编写方便,扩展性没有 Runnable接口号。

```
public class MyThread extends Thread {
    @Override
    public void run() {
        System.out.println(Thread.currentThread().getName());
    }
}
```

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

1.3 线程调度

sleep(long time): 休眠,让出CPU的资源一定时间后,重新获取线程资源。

编写一个业务类

```
public class MyRunnable implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentThread().getName());
        }
    }
}</pre>
```

测试类

```
public class Test {

public static void main(String[] args) {
    // 启动2个线程来执行任务
    MyRunnable mr = new MyRunnable();

Thread t1 = new Thread(mr, "A");
    Thread t2 = new Thread(mr, "B");

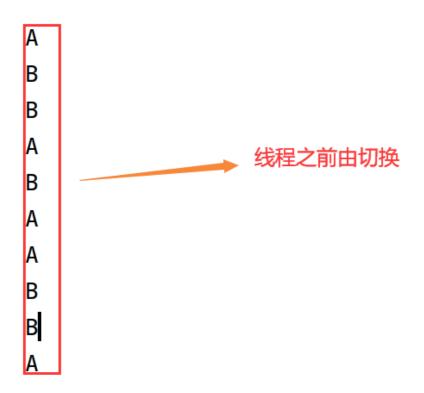
t1.start();
    t2.start();
}
```

```
demo2 C:\Users\jack\Desktop 13
 > <u></u>idea
                            14
                                            Thread t1 = new Thread(mr, "A");
 > out
                           15
                                            Thread t2 = new Thread(mr, "B");
  ∨ 📄 src
                            16
    ∨ D com
                            17
                                            t1.start();
      ∨ 🖿 iweb
                                            t2.start();
                            18
         > lesson01
Run: \blacksquare Test (2) 	imes
       В
■ †:
                         在B线程执行的过程中,一直持有CPU的资源
       В
薪
       В
       В
   ₽
       la
   <u>=</u>↓
       Α
```

代码修改

```
for (int i = 0; i < 5; i++) {
    System.out.println(Thread.currentThread().getName());

    try {
        Thread.sleep(1); // 当前线程体服1 ns, 这个1ms期间其他线程有机会获取CPU的资源
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```



yeid(): 让步,让出CPU的资源一个计算频率。重新获取线程资源。

```
for (int i = 0; i < 5; i++) {
             System.out.println(Thread.currentThread().getName());
                              Thread.yield();
   Test (2) \times
   D:\tools\java\jdk1.8.0_181\bin\java.exe ...
   В
   В
  В
                                    因为让出的时间比较短,效果不是很明显
5
  В
<u>+</u>
   В
   Α
-
   Α
ì
   Α
```

join(): 线程的加入,一个线程强行加入另外一个线程,直到加入的线程执行完毕后才释放线程资源。

```
public class MyRunnable1 implements Runnable {
    private Thread thread;
    public void setThread(Thread thread) {
        this.thread = thread;
    }
    @override
    public void run() {
        // 当执行 i = 3 的时候 加入一个线程
        for (int i = 0; i < 5; i++) {
           if (i == 3) {
               try {
                    // 加入线程
                   thread.start();
                   thread.join();
               } catch (InterruptedException e) {
                    e.printStackTrace();
               }
            }
           System.out.println(Thread.currentThread().getName());
        }
   }
}
```

```
public class MyRunnable2 implements Runnable {

    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentThread().getName());
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                 e.printStackTrace();
            }
}</pre>
```

```
}
}
}
```

```
public class Test {

public static void main(String[] args) {

    MyRunnable1 m1 = new MyRunnable1();
    MyRunnable2 m2 = new MyRunnable2();

    Thread t1 = new Thread(m1, "A");
    Thread t2 = new Thread(m2, "B");

    m1.setThread(t2);

    t1.start();
}
```

- 1.4 线程交互
- 1.5 线程的生命周期
- 1.6 守护线程
- 1.7 线程组 (了解)
- 1.8 数据共享
- 1.9 线程同步与锁
- 2.0 Lock锁类
- 2.1 线程池
- 2.2 生产者消费者 (设计模式)
- 2.3 定时任务