多线程

一、概述

- **线程(Thread)**：是一个程序内部的一条执行流程。
- **单线程程序**：如果一个程序内部只有一条执行流程，则称这个程序是一个单线程程序。
- **多线程**：是指从软硬件上实现的多条执行流程的技术（多条线程由CPU负责调度执行）。

二、创建多线程

> **Important**
>
> `java.lang.Thread`类代表线程。

2.0 注意事项

- 启动线程必须调用`start`方法，而非`run`方法。
  - 直接调用`run`方法会被当成普通方法执行，此时相当于还是单线程程序。
  - 只有调用`start`方法才是启动一个新的线程执行。
- 不要把主线程任务放在启动子线程之前，否则会导致主线程任务执行完毕后才启动子线程，失去了多线程的意义。

2.1 创建方式1：继承`Thread`类

P1 步骤

1. 创建一个子类`MyThread`并继承`Thread`类；
2. 重写`run()`方法；
3. 创建子类的实例对象；
4. 调用线程对象的`start()`方法启动线程，`start()`方法内部会去调用子类重写的`run()`方法。

P2 代码演示

- `ThreadTest1`类：

```java
package create_thread;

public class ThreadTest1 {
    // main方法是由主线程负责执行的
    public static void main(String[] args) {

        Thread t = new MyThread();
        // 启动后，实现了多线程：主线程 + t线程
        t.start();// start会去调用子类重写的run方法

        for (int i = 0; i < 10; i++) {
            System.out.println("Main Thread output: " + i);
        }
    }
}
```

- `MyThread`类：

```java
public class MyThread extends Thread {
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println("child--My Thread output: " + i);
        }
    }
}
```

- 控制台输出（每次输出结果都不同，仅截取其中某一次）：

```
D:\CS-Softwares\JDK\JDK21\bin\java.exe "-javaagent:D:\CS-Softwares\Int
child--My Thread output: 0
child--My Thread output: 1
Main Thread output: 0
Main Thread output: 1
Main Thread output: 2
Main Thread output: 3
Main Thread output: 4
Main Thread output: 5
Main Thread output: 6
Main Thread output: 7
Main Thread output: 8
child--My Thread output: 2
Main Thread output: 9
child--My Thread output: 3
child--My Thread output: 4
child--My Thread output: 5
child--My Thread output: 6
child--My Thread output: 7
child--My Thread output: 8
child--My Thread output: 9

Process finished with exit code 0
```

P3 优缺点

- **优点**：编码简单。
- **缺点**：子线程类已经继承了`Thread`类，无法再继承其他类，不利于后续功能扩展。

2.2 创建方式2：实现`Runnable`接口

P1 步骤

1. 创建一个线程任务类`MyRunnable`并实现`Runnable`接口；
2. 重写`run()`方法；
3. 把`MyRunnable`线程任务对象作为参数传递给`Thread`的一个有参构造器，从而生成一个线程对象；
4. 调用该线程对象的`start()`方法启动线程。

P2 代码演示

- `ThreadTest2`类：

```java
public class ThreadTest2 {
    public static void main(String[] args) {
        Runnable target = new MyRunnable();
        new Thread(target).start();

        for (int i = 0; i < 10; i++) {
            System.out.println("Main Thread output: " + i);
        }
    }
}
```

- `MyRunnable`类：

```java
public class MyRunnable implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println("child--My Runnable output: " + i);
        }
    }
}
```

- 控制台输出（每次输出结果都不同，仅截取其中某一次）：

```
D:\CS-Softwares\JDK\JDK21\bin\java.exe "-javaagent:D:\CS-Software
child--My Runnable output: 0
child--My Runnable output: 1
child--My Runnable output: 2
child--My Runnable output: 3
child--My Runnable output: 4
child--My Runnable output: 5
child--My Runnable output: 6
Main Thread output: 0
child--My Runnable output: 7
Main Thread output: 1
Main Thread output: 2
Main Thread output: 3
Main Thread output: 4
Main Thread output: 5
Main Thread output: 6
Main Thread output: 7
Main Thread output: 8
child--My Runnable output: 8
Main Thread output: 9
child--My Runnable output: 9

Process finished with exit code 0
```

P3 优缺点

- **优点**：线程任务类只是实现接口，可以继续继承其它类，实现其它接口，扩展性强。
- **缺点**：需要额外创建一个`Runnable`对象。

P4 使用匿名内部类简化上述代码

```
public class ThreadTest2_2 {
    public static void main(String[] args) {
        // Lambda表达式
        Runnable target = (() -> {
            for (int i = 0; i < 10; i++) {
                System.out.println("child output: " + i);
            }
        });
        new Thread(target).start();
    }
}
```

2.3 创建方式3：利用`Callable`接口和`FutureTask`类

P0 引入背景

前两种创建方式都存在一个问题：假如线程执行完毕后有一些数据需要返回，由于`run()`方法的返回值为空，所以均不能直接返回结果。

因此，`JDK5.0`提供了`Callable`接口和`FutureTask`类来解决这个问题。

P1 步骤

1. 创建一个类`MyCallable`并实现`Callable`接口；
2. 重写`call`方法，封装线程任务和要返回的数据；
3. 把`Callable`对象封装成`FutureTask`对象（线程任务对象）。
4. 把该线程任务对象作为参数传递给`Thread`的一个有参构造器，从而生成一个线程对象；

5. 调用该线程对象的`start()`方法启动线程；
6. 线程执行完毕后，可以通过`FutureTask`对象的`get()`方法去获取线程任务的执行结果。

> **Tip**
>
> 可以采用匿名内部类简化步骤。

P2 代码演示

- `ThreadTest3`类：

```java
import java.util.concurrent.Callable;
import java.util.concurrent.FutureTask;

public class ThreadTest3 {
    private static int n = 100;

    public static void main(String[] args) throws Exception {

        Callable<String> myCallable = (() -> {
            int sum = 0;
            for (int i = 1; i <= n; i++) {
                sum += i;
            }
            return "child calculate the sum from 1 to " + n + ": "
+ sum;
        });

        FutureTask<String> futureTask = new FutureTask<>
(myCallable);
        new Thread(futureTask).start();

        String result = futureTask.get();
        System.out.println(result);
    }
}
```

- 控制台输出：

```
D:\CS-Softwares\JDK\JDK21\bin\java.exe "-javaagent:D:\C
child calculate the sum from 1 to 100: 5050

Process finished with exit code 0
```

P3 优缺点

- **优点**：
  - 线程任务类只是实现接口，可以继续继承其它类，实现其它接口，扩展性强。
  - 可以在线程执行完毕后获取到线程的执行结果。
- **缺点**：编码相对复杂一些。

三、`Thread`类的常用API

3.1 常用构造器

序号	构造器	说明
01	`public Thread(String name)`	创建一个指定名称的线程对象。
02	`public Thread(Runnable target)`	封装`Runnable`对象成为线程对象。
03	`public Thread(Runnable target, String name)`	封装`Runnable`对象成为线程对象，同时指定线程名称。

3.2 常用方法

序号	方法	说明
01	`void run()`	线程的任务方法。
02	`void start()`	启动线程。
03	`String getName()`	获取当前线程名称，默认是`Thread-索引`。
04	`void setName(String name)`	为线程设置名称，建议在启动线程之前。
05	`static Thread currentThread()`	获取当前执行的线程对象。
06	`static void sleep(long time)`	让当前执行的线程休眠一定毫秒数后，再继续执行。
07	`final void join()`	调用此方法的线程会优先执行完毕，合理使用此方法可以安排线程执行顺序。

四、线程安全

4.1 线程安全问题

- 多个线程同时访问并修改同一个共享资源时，可能会出现业务安全问题。
- 比如A、B线程同时使用打印机，会导致打印出的内容混杂错乱。

4.2 用程序模拟线程安全问题

P1 需求

小明和小红是一对夫妻，他们有一个共同银行账户，余额是10万元，现在模拟二人同时取出10万元的操作。

P2 代码演示

- `Account`：

```
/**
 * 账户类，代表小明和小红的共同银行账户
 * 单例模式
 */
public class Account {
    private double money;// 账户余额

    private static Account account = new Account(100000);
    private Account(double money) {
        this.money = money;
    }
    public static Account getAccount() {
        return account;
    }

    public void drawMoney(double moneyToDraw) {
        String threadName = Thread.currentThread().getName();
        if (money >= moneyToDraw) {
            System.out.println(threadName + moneyToDraw + "成功！");
            money -= moneyToDraw;
            System.out.println(threadName + "后，余额变更为：" +
money);
        } else {
            System.out.println(threadName + "：余额不足！");
        }
    }

    public double getMoney() {
        return money;
    }
    public void setMoney(double money) {
        this.money = money;
```

```
    }
}
```

- `CashWithdrawalThread`:

```
/**
 * 取钱线程类
 */
public class CashWithdrawalThread extends Thread {
    private Account account;

    public CashWithdrawalThread(String name, Account account) {
        super(name);
        this.account = account;
    }

    @Override
    public void run() {
        // 取钱
        account.drawMoney(100000);
    }
}
```

- `Test`:

```
public class Test {
    public static void main(String[] args) {
        Account account = Account.getAccount();
        Thread xiaoMing = new CashWithdrawalThread("小明取钱",
account);
        Thread xiaoHong = new CashWithdrawalThread("小红取钱",
account);
        xiaoMing.start();
        xiaoHong.start();
    }
}
```

P3 控制台输出（每次输出结果都不同，仅截取其中某一次）

```
D:\CS-Softwares\JDK\JDK21\bin\java.exe "-javaagent:D:\
小红取钱100000.0成功!
小明取钱100000.0成功!
小红取钱后，余额变更为：0.0
小明取钱后，余额变更为：-100000.0

Process finished with exit code 0
```

五、线程同步

六、线程通信

七、线程池