**[Java 异步编程](https://www.cnblogs.com/liumaowu/p/9293627.html)**

<https://www.cnblogs.com/liumaowu/p/9293627.html>

昨天头儿给的学习文档我还没看完，头儿说：“MongoDB光会简单的添删改查什么的不行，要深入了解，你们连$set和$inc使用场景都分不清。”

　　确实，学习过一年多SQL，确实对学习MongoDB有点影响。

　　不过，今天数据库的事情先翻过去，因为我在学习文档中还看到了另外一个加大加粗的标题——**异步编程**。

　　Java在Java8之前貌似（因为我也刚学，所以不对还请各位前辈指正）没有真正实现异步编程的方法，当时异步编程会使用回调或者使用其他的框架（如Netty和Guava）来实现。后来Java8借鉴了很多框架的思想，可以借助JDK原生的CompletableFuture来实现异步操作，而且用Lambda表达式来写匿名内部类大大简化了代码量。

　　那么，异步编程到底是用来干什么的呢？

　　试想这样的场景——老爸有俩孩子：小红和小明。老爸想喝酒了，他让小红去买酒，小红出去了。然后老爸突然想吸烟了，于是老爸让小明去买烟。在面对对象的思想中，一般会把买东西，然后买回来这件事作为一个方法，如果按照顺序结构或者使用多线程同步的话，小明想去买烟就必须等小红这个买东西的操作进行完。这样无疑增加了时间的开销。异步就是为了解决这样的问题。你可以分别给小红小明下达指令，让他们去买东西，然后你就可以自己做自己的事，等他们买回来的时候接收结果就可以了。

　　所以，为了尽快学习异步操作，我使用Java8的方法编写了这样一段代码来帮助了解：

[复制代码](javascript:void(0);)

1 package com.liumaowu;

2

3 import java.util.concurrent.CompletableFuture;

4 import java.util.concurrent.ExecutorService;

5 import java.util.concurrent.Executors;

6

7 public class TestFuture {

8 public static void main(String[] args) {

9 //两个线程的线程池

10 ExecutorService executor= Executors.newFixedThreadPool(2);

11 //小红买酒任务，这里的future2代表的是小红未来发生的操作，返回小红买东西这个操作的结果

12 CompletableFuture<String> future2 = CompletableFuture.supplyAsync(()-> {

13 System.out.println("爸：小红你去买瓶酒！");

14 try {

15 System.out.println("小红出去买酒了，女孩子跑的比较慢，估计5s后才会回来...");

16 Thread.sleep(5000);

17 return "我买回来了！";

18 } catch (InterruptedException e) {

19 System.err.println("小红路上遭遇了不测");

20 return "来世再见！";

21 }

22 },executor);

23

24 //小明买烟任务，这里的future1代表的是小明未来买东西会发生的事，返回值是小明买东西的结果

25 CompletableFuture<String> future1 = CompletableFuture.supplyAsync(()->{

26 System.out.println("爸：小明你去买包烟！");

27 try {

28 System.out.println("小明出去买烟了，可能要3s后回来...");

29 Thread.sleep(3000);

30 return "我买回来了!";

31 } catch (InterruptedException e) {

32 System.out.println("小明路上遭遇了不测！");

33 return "这是我托人带来的口信，我已经不在了。";

34 }

35 },executor);

36

37 //获取小红买酒结果，从小红的操作中获取结果，把结果打印

38 future2.thenAccept((e)->{System.out.println("小红说："+e);});

39 //获取小明买烟的结果

40 future1.thenAccept((e)->{System.out.println("小明说："+e);});

41

42 System.out.println("爸：loading......");

43 System.out.println("爸:我觉得无聊甚至去了趟厕所。");

44 System.out.println("爸：loading......");

45 }

46 }

[复制代码](javascript:void(0);)

　　这个运行结果如下：

[复制代码](javascript:void(0);)

爸：小红你去买瓶酒！

小红出去买酒了，女孩子跑的比较慢，估计5s后才会回来...

爸：小明你去买包烟！

小明出去买烟了，可能要3s后回来...

爸：loading......

爸:我觉得无聊甚至去了趟厕所。

爸：loading......

小明说：我买回来了!

小红说：我买回来了！

[复制代码](javascript:void(0);)

　　因为sleep时间调的挺大，所以可以很明显的看到小明买回来的结果是在等待之后一会儿才跳出来，小红买回来的结果也等待了一会儿才跳出来。因为是先让小红买，再让小明出去的，但是小明的结果先回来，这说明什么时候取到结果取决于操作的时间长度。

　　之后我把取futrue的结果的操作放到了老爸等待语句的下面，如下：

[复制代码](javascript:void(0);)

1 /\*

2 上面的代码不动

3 \*/

4 System.out.println("爸：loading......");

5 System.out.println("爸:我觉得无聊甚至去了趟厕所。");

6 System.out.println("爸：loading......");

7 //获取小红买酒结果

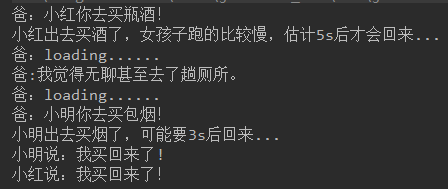
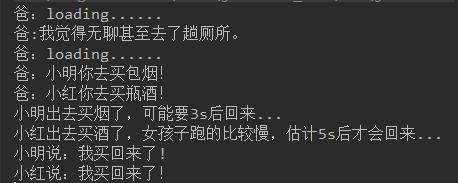
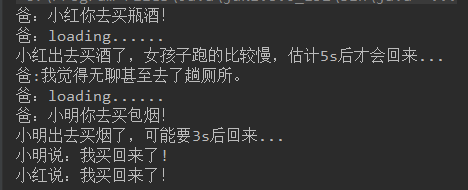
8 future2.thenAccept((e)->{System.out.println("小红说："+e);});

9 //获取小明买烟的结果

10 future1.thenAccept((e)->{System.out.println("小明说："+e);});

[复制代码](javascript:void(0);)

　　多次运行之后结果就比较有意思了：

结果1：结果2：结果3

　　这种解决异步的方式还是使用到了多线程，在获取结果之前，操作的顺序取决于线程操作到了哪一步，比如结果2，因为小红线程启动的慢了，所以小明先出去了。但是如果先取结果的话，会确保这个线程已经启动了，所以老爸loading始终显示在老爸发出买东西命令之后。这也算先强制线程启动，再进行主操作。

在看了异步之后，不可避免地会把**同步/异步/多线程**弄混淆，那么我们再来总结下几个东西的**区别**：

**一般情况**：顺序结构，必须等待前面的操作完成（两个人说话，a把所有话说完，b才能继续说）

**并发**：同一时间段处理多个任务的能力（两人说话，支持你一言我一语的交流，两人在一个时间段内都有说话，是基于时间段内的同时发生）

**并发又有同步和互斥**

**互斥**：不能同时使用临界资源（有一个共享资源--话筒，两人必须用话筒说话，但同时只能有一个人用这个话筒，保证了只有一个人在说话）

**同步：**前一个处理的结果作为下一个处理的资源。大多数情况下，同步已经实现了互斥。（两人你一言我一语的交流，我必须知道你说了啥我才能接上你的话）

**并行**：同一时刻处理多个任务的能力（两人合唱，同时出声）

**异步**：不用等待一个结果出来，可以继续其他操作（两个人不说话了，寄信，a把信拿到邮局就不用管了，回家可以想干嘛就干嘛，等b回信到了，取邮局接收一下结果--b的回信就可以了）

**多线程**：如果说同步和异步是对如何处理事情的**要求**，那么多线程就是实现这些要求的**方法**。

<https://www.jianshu.com/p/dade70be4edd>

<https://colobu.com/2016/02/29/Java-CompletableFuture/>