[**并发、并行、同步、异步、多线程的区别？**](https://www.cnblogs.com/xqn2017/p/8029472.html)

并发：同一时间段有几个程序都处于已经启动到运行完毕之间，并且这几个程序都在同一个处理机上运行，并发的两种关系是同步和互斥；  
互斥：进程之间访问临界资源时相互排斥的现象；  
同步：进程之间存在依赖关系，一个进程结束的输出作为另一个进程的输入。具有同步关系的一组并发进程之间发送的信息称为消息或者事件；  
并行：单处理器中进程被交替执行，表现出一种并发的外部特征；在多处理器中，进程可以交替执行，还能重叠执行，实现并行处理，并行就是同事发生的多个并发事件，具有并发的含义，但并发不一定是并行，也就是说事件之间不一定要同一时刻发生；  
多线程：多线程是进程中并发运行的一段代码，能够实现线程之间的切换执行；（起多个线程，同时执行同一个代码（方法）），这样就起到了异步效果，  
异步：和同步相对，同步是顺序执行，而异步是彼此独立，在等待某个事件的过程中继续做自己的事，不要等待这一事件完成后再工作。线程是实现异步的一个方式，异步是让调用方法的主线程不需要同步等待另一个线程的完成，从而让主线程干其他事情。（异步就是，执行同一段代码时，不需要）  
异步和多线程：不是同等关系，异步是目的，多线程只是实现异步的一个手段，实现异步可以采用多线程技术或者交给其他进程来处理。

#### 2.并发和并行的区别：

     所有的并发处理都有排队等候，唤醒和执行这三个步骤，所以并发是宏观的观念，在微观上他们都是序列被处理的，只不过资源不会在某一个上被阻塞（一般是通过时间片轮转），所以在宏观上多个几乎同时到达的请求同时在被处理。如果是同一时刻到达的请求也会根据优先级的不同，先后进入队列排队等候执行。

     并发与并行是两个既相似但是却不相同的概念：

         并发性：又称共行性，是指处理多个同时性活动的能力，。

         并行：指同时发生两个并发事件，具有并发的含义。并发不一定并行，也可以说并发事件之间不一定要同一时刻发生。

     并发的实质是一个物理CPU（也可以是多个物理CPU）在若干个程序之间多路复用，并发性是对有限物理资源强制行使 多用户共享以提高效率。

     并行指两个或两个以上事件或活动在同一时刻发生，在多道程序环境下，并行使多个程序同一时刻可在不同CPU上同时执行。

     并发是在同一个cpu上同时（不是真正的同时，而是看来是同时，因为CPU要在多个程序之间切换）运行多个程序。

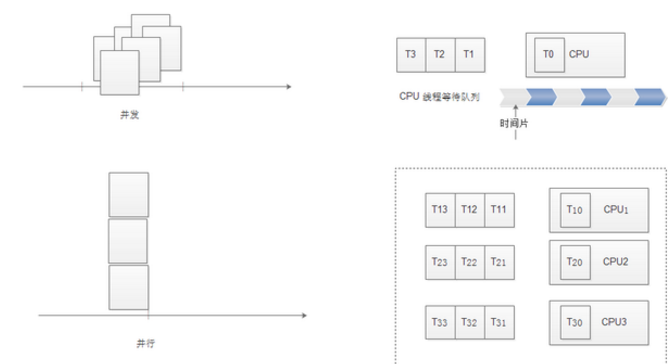
     并行是每一个CPU运行一个程序。

     打个比方：并发就像一个人（CPU）喂两个小孩（程序）吃饭，表面上是两个小孩在吃饭，实际是一个人在喂。

                   并行就是两个人喂两个小孩子吃饭。

#### 3.并发、并行和多线程的关系：都是起了多个线程，并发指的是多个线程同时让一个处理器处理，通过时间片处理，多线程得并发（同时执行同一段代码） 可以达到异步的效果（不会等程序处理完），还可以通过@Async标记方法，表示可以异步执行，即主线程不阻塞，具体执行交给其他线程（类似多线程的线程）

     并行需要两个或两个以上的线程跑在不同的处理器上，并发可以跑在一个处理器上通过时间片进行切换。



# [Spring中@Async](https://www.cnblogs.com/wihainan/p/6516858.html)（异步执行）

在[Java](http://lib.csdn.net/base/javase)应用中，绝大多数情况下都是通过同步的方式来实现交互处理的；但是在处理与第三方系统交互的时候，容易造成响应迟缓的情况，之前大部分都是使用多线程来完成此类任务，其实，在[spring](http://lib.csdn.net/base/javaee) 3.x之后，就已经内置了@Async来完美解决这个问题

**1.  何为异步调用？**

    在解释异步调用之前，我们先来看同步调用的定义；同步就是整个处理过程顺序执行，当各个过程都执行完毕，并返回结果。 异步调用则是只是发送了调用的指令，调用者无需等待被调用的方法完全执行完毕；而是继续执行下面的流程。

     例如， 在某个调用中，需要顺序调用 A, B, C三个过程方法；如他们都是同步调用，则需要将他们都顺序执行完毕之后，方算作过程执行完毕； 如B为一个异步的调用方法，则在执行完A之后，调用B，并不等待B完成，而是执行开始调用C，待C执行完毕之后，就意味着这个过程执行完毕了。

**2.  常规的异步调用处理方式**

    在Java中，一般在处理类似的场景之时，都是基于创建独立的线程去完成相应的异步调用逻辑，通过主线程和不同的线程之间的执行流程，从而在启动独立的线程之后，主线程继续执行而不会产生停滞等待的情况。或是使用TaskExecutor执行异步线程，参看http://www.cnblogs.com/wihainan/p/6098970.html。

**3. @Async介绍**

   在Spring中，基于@Async标注的方法，称之为异步方法；这些方法将在执行的时候，将会在独立的线程中被执行，调用者无需等待它的完成，即可继续其他的操作。

   如何在Spring中启用@Async。基于Java配置的启用方式：

1 @Configuration

2 @EnableAsync

3 public class SpringAsyncConfig { ... }

**4. 基于@Async无返回值调用**

使用的方式非常简单，一个标注即可解决所有的问题：

1 @Async //标注使用

2 public void asyncMethodWithVoidReturnType() {

3 System.out.println("Execute method asynchronously. "

4 + Thread.currentThread().getName());

5 }

**5. 基于@Async返回值的调用**

[复制代码](javascript:void(0);)

1 @Async

2 public Future<String> asyncMethodWithReturnType() {

3 System.out.println("Execute method asynchronously - " + Thread.currentThread().getName());

4 try {

5 Thread.sleep(5000);

6 return new AsyncResult<String>("hello world !!!!");

7 } catch (InterruptedException e) {

8 //

9 }

10

11 return null;

12 }

[复制代码](javascript:void(0);)

  以上示例可以发现，返回的数据类型为Future类型，其为一个接口。具体的结果类型为AsyncResult,这个是需要注意的地方。

 调用返回结果的异步方法示例

[复制代码](javascript:void(0);)

1 public void testAsyncAnnotationForMethodsWithReturnType()

2 throws InterruptedException, ExecutionException {

3 System.out.println("Invoking an asynchronous method. " + Thread.currentThread().getName());

4 Future<String> future = asyncAnnotationExample.asyncMethodWithReturnType();

5

6 while (true) { ///这里使用了循环判断，等待获取结果信息

7 if (future.isDone()) { //判断是否执行完毕

8 System.out.println("Result from asynchronous process - " + future.get());

9 break;

10 }

11 System.out.println("Continue doing something else. ");

12 Thread.sleep(1000);

13 }

14 }

[复制代码](javascript:void(0);)

这些获取异步方法的结果信息，是通过不停的检查Future的状态来获取当前的异步方法是否执行完毕来实现的。

**6. 基于@Async调用中的异常处理机制**

 在异步方法中，如果出现异常，对于调用者caller而言，是无法感知的。如果确实需要进行异常处理，则按照如下方法来进行处理：

    1.  自定义实现AsyncTaskExecutor的任务执行器

         在这里定义处理具体异常的逻辑和方式。

    2.  配置由自定义的TaskExecutor替代内置的任务执行器

    示例步骤1，自定义的TaskExecutor

[复制代码](javascript:void(0);)

1 public class ExceptionHandlingAsyncTaskExecutor implements AsyncTaskExecutor {

2 private AsyncTaskExecutor executor;

3 public ExceptionHandlingAsyncTaskExecutor(AsyncTaskExecutor executor) {

4 this.executor = executor;

5 }

6 ////用独立的线程来包装，@Async其本质就是如此

7 public void execute(Runnable task) {

8 executor.execute(createWrappedRunnable(task));

9 }

10 public void execute(Runnable task, long startTimeout) {

11 /用独立的线程来包装，@Async其本质就是如此

12 executor.execute(createWrappedRunnable(task), startTimeout);

13 }

14 public Future submit(Runnable task) { return executor.submit(createWrappedRunnable(task));

15 //用独立的线程来包装，@Async其本质就是如此。

16 }

17 public Future submit(final Callable task) {

18 //用独立的线程来包装，@Async其本质就是如此。

19 return executor.submit(createCallable(task));

20 }

21

22 private Callable createCallable(final Callable task) {

23 return new Callable() {

24 public T call() throws Exception {

25 try {

26 return task.call();

27 } catch (Exception ex) {

28 handle(ex);

29 throw ex;

30 }

31 }

32 };

33 }

34

35 private Runnable createWrappedRunnable(final Runnable task) {

36 return new Runnable() {

37 public void run() {

38 try {

39 task.run();

40 } catch (Exception ex) {

41 handle(ex);

42 }

43 }

44 };

45 }

46 private void handle(Exception ex) {

47 //具体的异常逻辑处理的地方

48 System.err.println("Error during @Async execution: " + ex);

49 }

50 }

[复制代码](javascript:void(0);)

 分析： 可以发现其是实现了AsyncTaskExecutor, 用独立的线程来执行具体的每个方法操作。在createCallable和createWrapperRunnable中，定义了异常的处理方式和机制。

 handle()就是未来我们需要关注的异常处理的地方。 配置文件中的内容：

1 <task:annotation-driven executor="exceptionHandlingTaskExecutor" scheduler="defaultTaskScheduler" />

2 <bean id="exceptionHandlingTaskExecutor" class="nl.jborsje.blog.examples.ExceptionHandlingAsyncTaskExecutor">

3 <constructor-arg ref="defaultTaskExecutor" />

4 </bean>

5 <task:executor id="defaultTaskExecutor" pool-size="5" />

6 <task:scheduler id="defaultTaskScheduler" pool-size="1" />

**7. @Async调用中的事务处理机制**

    在@Async标注的方法，同时也适用了@Transactional进行了标注；在其调用[数据库](http://lib.csdn.net/base/mysql)操作之时，将无法产生事务管理的控制，原因就在于其是基于异步处理的操作。

     那该如何给这些操作添加事务管理呢？可以将需要事务管理操作的方法放置到异步方法内部，在内部被调用的方法上添加@Transactional.

    例如：  方法A，使用了@Async/@Transactional来标注，但是无法产生事务控制的目的。

          方法B，使用了@Async来标注，  B中调用了C、D，C/D分别使用@Transactional做了标注，则可实现事务控制的目的。

不积跬步无以至千里