**@Async**

定义**：**标注某个方法或某个类里面的所有方法都是需要异步处理的。

被注解的方法被调用的时候，会在新线程中执行，而调用它的方法会在原来的线程中执行。这样可以避免阻塞、以及保证任务的实时性。适用于处理log、发送邮件、短信……等。

注：当@Async修饰方法，和修饰类时，则以方法为主。

例子：

<task:annotation-driven executor="asyncExecutor" />

 <task:executor id="asyncExecutor" pool-size="100-10000" queue-capacity="10"/>

**queue-capacity**：当最小的线程数已经被占用满后，新的任务会被放进queue里面，当这个queue的capacity也被占满之后，pool里面会创建新线程处理这个任务，直到总线程数达到了max size，这时系统会拒绝这个任务并抛出TaskRejectedException异常（缺省配置的情况下，可以通过rejection-policy来决定如何处理这种情况）

**rejection-policy**：当pool已经达到max size的时候，如何处理新任务

* ABORT（缺省）：抛出TaskRejectedException异常，然后不执行
* DISCARD：不执行，也不抛出异常
* DISCARD\_OLDEST：丢弃queue中最旧的那个任务
* CALLER\_RUNS：不在新线程中执行任务，而是有调用者所在的线程来执行

**返回值**：异步方法返回值必须为Future<>，就像Callable与Future。isDone()和isSuccess()两个方法

Spring提供的返回类型：AsyncResult。

SimpleAsyncTaskExecutor：默认情况下,Spring使用SimpleAsyncTaskExecutor来运行这些异步方法,默认的设置方式可以在两个层级上面进行覆盖 - 在应用全局配置上或在单独的方法上。

另一种实现异步的方式：配置类应该实现AsyncConfigurer接口 - 意思是getAsyncExecutor方法需要我们自己来进行实现，getAsyncUncaughtExceptionHandler()方法来自定义我们的异步异常处理类:

应用场景：1、某些耗时较长的而用户不需要等待该方法的处理结果

2、某些耗时较长的方法，后面的程序不需要用到这个方法的处理结果时