### @PostConstruct：

该注解由spring提供被用来修饰一个非静态的void（）方法。被@PostConstruct修饰的方法会在服务器加载Servlet的时候运行，并且只会被服务器执行一次。PostConstruct在构造函数之后执行，init（）方法之前执行。

通常我们会是在Spring框架中使用到@PostConstruct注解 该注解的方法在整个Bean初始化中的执行顺序：

Constructor(构造方法) -> @Autowired(依赖注入) -> @PostConstruct(注释的方法)

### @PreDestroy

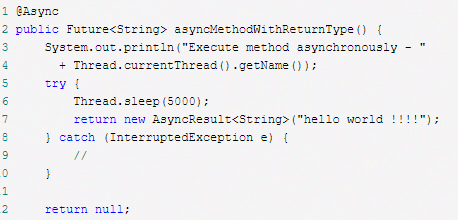
 被@PreDestroy修饰的方法会在服务器卸载Servlet的时候运行，并且只会被服务器调用一次，类似于Servlet的destroy()方法。被@PreDestroy修饰的方法会在destroy()方法之后运行，在Servlet被彻底卸载之前

### @Async spring异步注解的使用

**Jdk1.8新增的CompletableFuture实现异步调用的方式更好**

异步注解的启用，在使用类、配置类或者启动类上添加@EnableAsync注解，在业务类上添加该注解则该类的所有方法都会异步执行，或者基于xml配置的方式<task:executor id="myexecutor" pool-size="5" /><task:annotation-driven executor="myexecutor"/>，然后在方法上添加@Async注解

返回值问题，如果无返回值可以直接使用，如果需要返回值则返回值类型为线程池的Future接口，具体的结果类型为AsyncResult



异步注解与事物注解的使用

@Async与@Transactional

如果异步与事物同时使用，在其调用数据库时将无法产生事物管理控制，原因就是异步调用，解决方式为将事物注解放在异步方法的内部

@Async 底层线程池

参考<https://www.cnblogs.com/wlandwl/p/async.html>

Spring 已经实现的异步线程池：

1. SimpleAsyncTaskExecutor：不是真的线程池，这个类不重用线程，默认每次调用都会创建一个新的线程。

2. SyncTaskExecutor：这个类没有实现异步调用，只是一个同步操作。只适用于不需要多线程的地方。

3. ConcurrentTaskExecutor：Executor的适配类，不推荐使用。如果ThreadPoolTaskExecutor不满足要求时，才用考虑使用这个类。

4. SimpleThreadPoolTaskExecutor：是Quartz的SimpleThreadPool的类。线程池同时被quartz和非quartz使用，才需要使用此类。

5. ThreadPoolTaskExecutor ：最常使用，推荐。 其实质是对java.util.concurrent.ThreadPoolExecutor的包装。

**@Async默认线程池的弊端**

在线程池应用中，参考阿里巴巴java开发规范：线程池不允许使用Executors去创建，不允许使用系统默认的线程池，推荐通过ThreadPoolExecutor的方式，这样的处理方式让开发的工程师更加明确线程池的运行规则，规避资源耗尽的风险。Executors各个方法的弊端：

* newFixedThreadPool和newSingleThreadExecutor：主要问题是堆积的请求处理队列可能会耗费非常大的内存，甚至OOM。
* newCachedThreadPool和newScheduledThreadPool：主要问题是线程数最大数是Integer.MAX\_VALUE，可能会创建数量非常多的线程，甚至OOM。

    @Async默认异步配置使用的是SimpleAsyncTaskExecutor，该线程池默认来一个任务创建一个线程，若系统中不断的创建线程，最终会导致系统占用内存过高，引发OutOfMemoryError错误。针对线程创建问题，SimpleAsyncTaskExecutor提供了限流机制，通过concurrencyLimit属性来控制开关，当concurrencyLimit>=0时开启限流机制，默认关闭限流机制即concurrencyLimit=-1，当关闭情况下，会不断创建新的线程来处理任务。基于默认配置，SimpleAsyncTaskExecutor并不是严格意义的线程池，达不到线程复用的功能。

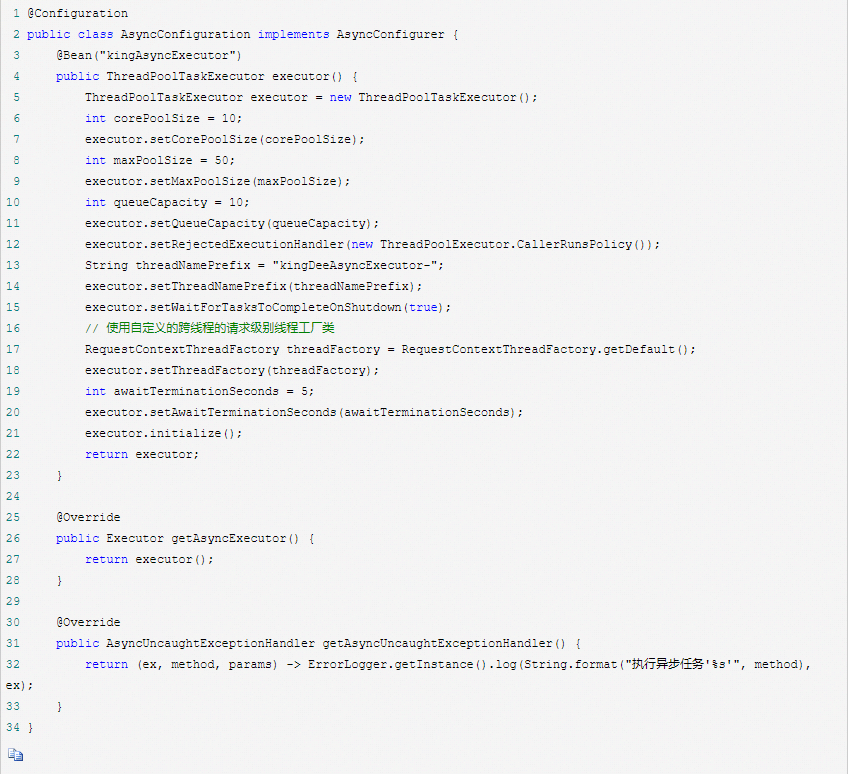
**@Async应用自定义的线程池（只能有一个线程池）**

自定义线程池，可对系统中线程池更加细粒度的控制，方便调整线程池大小配置，线程执行异常控制和处理。在设置系统自定义线程池代替默认线程池时，虽可通过多种模式设置，但替换默认线程池最终产生的线程池有且只能设置一个（不能设置多个类继承AsyncConfigurer）。自定义线程池有如下模式：

* 重新实现接口AsyncConfigurer
* 继承AsyncConfigurerSupport
* 配置由自定义的TaskExecutor替代内置的任务执行器

   通过查看Spring源码关于@Async的默认调用规则，会优先查询源码中实现AsyncConfigurer这个接口的类，实现这个接口的类为AsyncConfigurerSupport。但默认配置的线程池和异步处理方法均为空，所以，无论是继承或者重新实现接口，都需指定一个线程池。且重新实现 public Executor getAsyncExecutor()方法。

**实现接口AsyncConfigurer**



**继承AsyncConfigurerSupport**



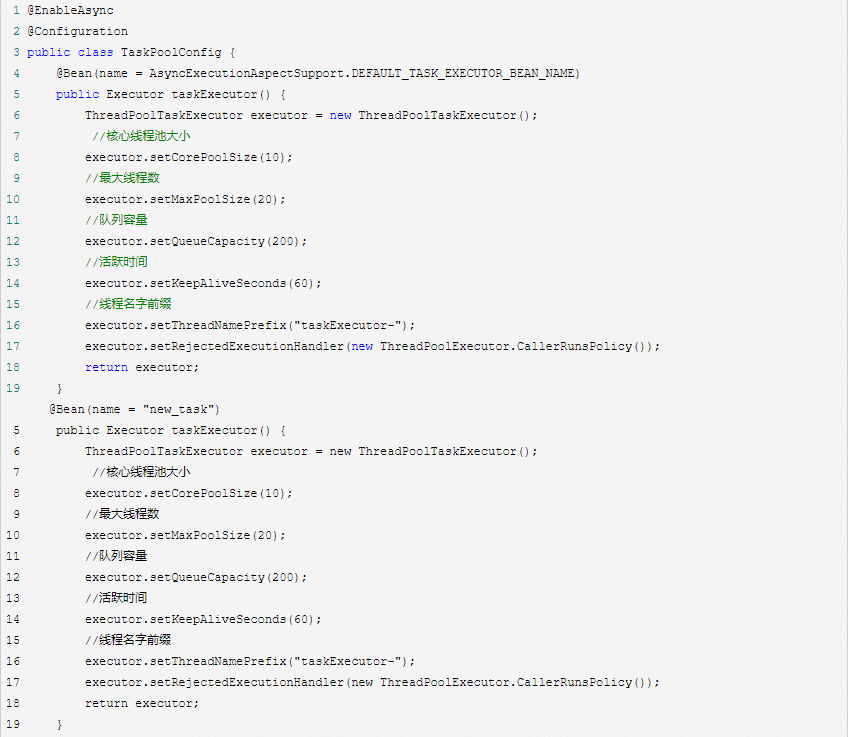
**@Async 配置自定义的TaskExecutor（可以配置多个）**

由于AsyncConfigurer的默认线程池在源码中为空，Spring通过beanFactory.getBean(TaskExecutor.class)先查看是否有线程池，未配置时，通过beanFactory.getBean(DEFAULT\_TASK\_EXECUTOR\_BEAN\_NAME, Executor.class)，又查询是否存在默认名称为TaskExecutor的线程池。所以可在项目中，定义名称为TaskExecutor的bean生成一个默认线程池。也可不指定线程池的名称，申明一个线程池，本身底层是基于TaskExecutor.class便可。

比如：

 Executor.class:ThreadPoolExecutorAdapter->ThreadPoolExecutor->AbstractExecutorService->ExecutorService->Executor（这样的模式，最终底层为Executor.class，在替换默认的线程池时，需设置默认的线程池名称为TaskExecutor）

 Executor.class:ThreadPoolTaskExecutor->SchedulingTaskExecutor->AsyncTaskExecutor->TaskExecutor（这样的模式，最终底层为TaskExecutor.class，在替换默认的线程池时，可不指定线程池名称。）



@Async注解，使用系统默认或者自定义的线程池（代替默认线程池）。可在项目中设置多个线程池，在异步调用时，指明需要调用的线程池名称，如@Async("new\_task")。

