**Spring 定时任务框架加载流程**

使用@EnableScheduling注解开启定时任务，该注解会加载SchedulingConfiguration配置类，SchedulingConfiguration

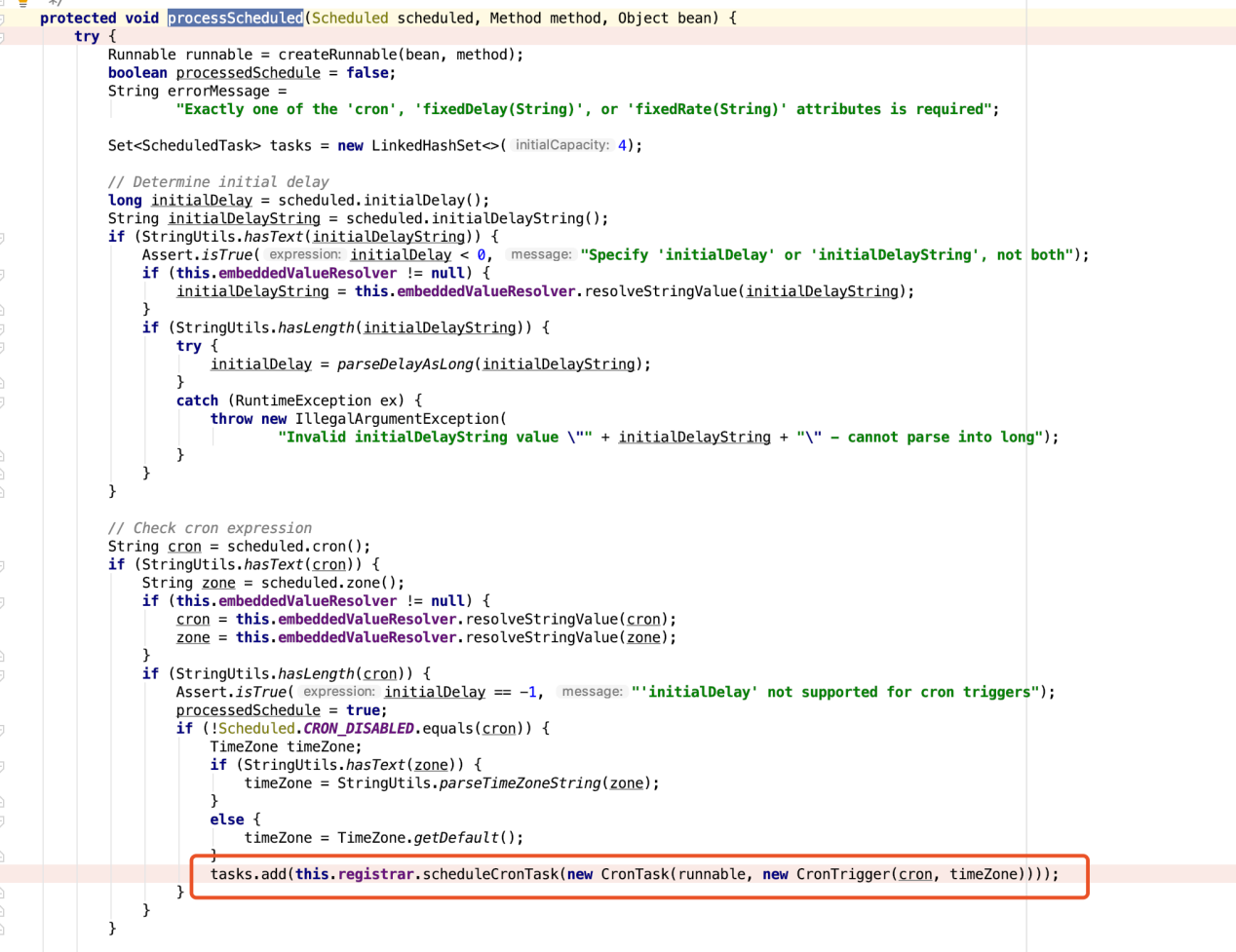
配置又会往spring容器中注入ScheduledAnnotationBeanPostProcessor

这个bean，当spring 容器中的bean初始化完毕后，会调用ScheduledAnnotationBeanPostProcessor.postProcessAfterInitialization()方法进行后置处理，代码如下：

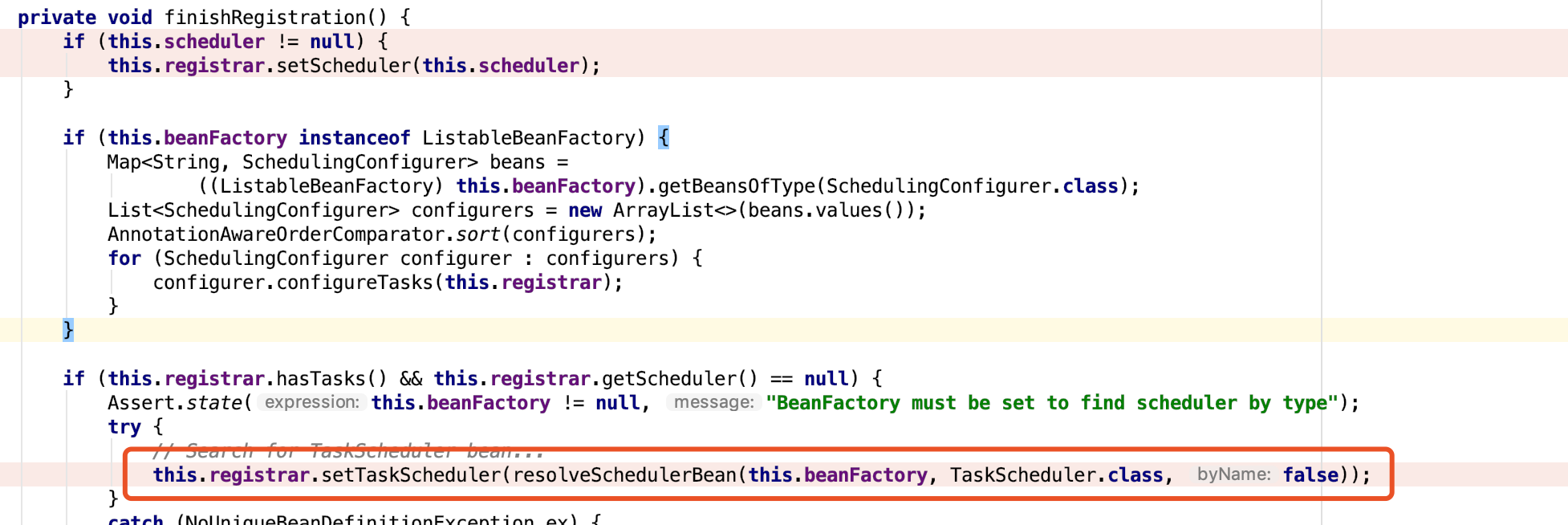


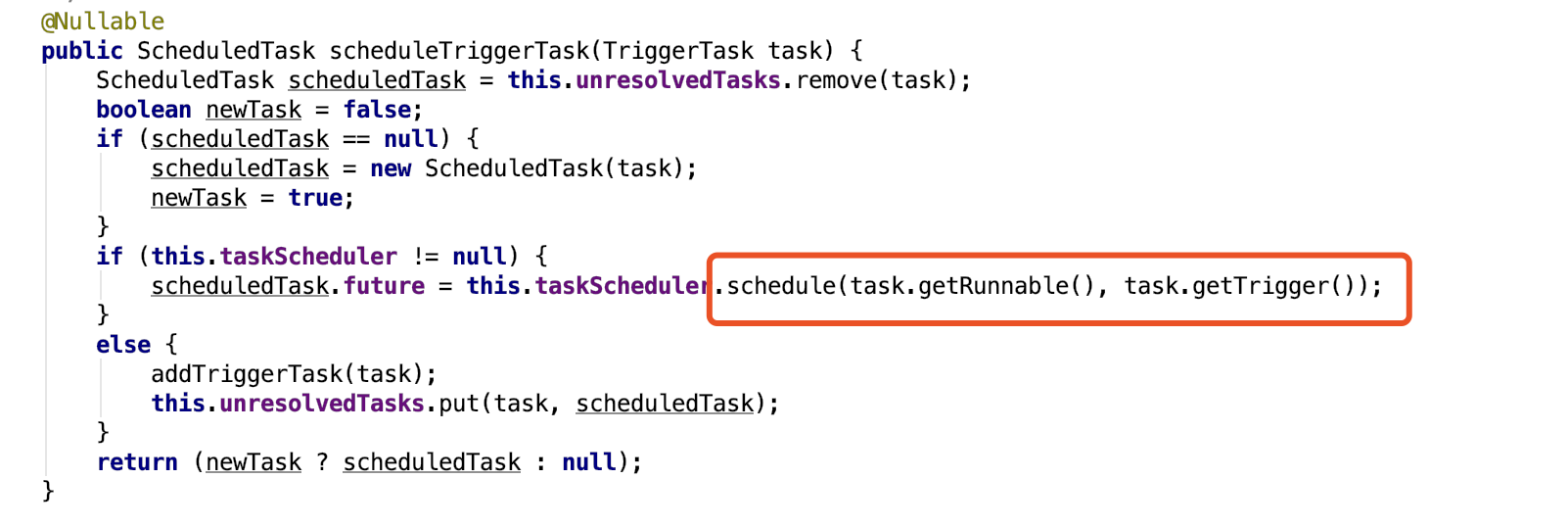
大概流程就是扫描每个bean的对应Class类型包括Class的父接口中每个方法是否有使用@Scheduled，@Schedules注解，如果有则会创建一个ScheduledMethodRunnable对象，在其run方法中，使用Java反射调用bean（如果为代理就是代理）对应Class类型对应的方法。然后根据@Scheduled，@Schedules注解上配置的不同信息，将ScheduledMethodRunnable封装为CronTask，FixedDelayTask，FixedRateTask三种任务类型，再次将这三种任务类型封装为ScheduledTask

并注册到ScheduledTaskRegistrar中代码如下：



ScheduledAnnotationBeanPostProcessor实现了ApplicationListener接口监听spring容器启动完成事件，然后将ThreadPoolTaskScheduler实例注入到ScheduledTaskRegistrar中，并且使用ThreadPoolTaskScheduler开始调度任务。代码如下：





ThreadPoolTaskScheduler由自动配置中的task自动配置进行加载，具体在TaskSchedulingAutoConfiguration中，默认线程池最大任务数为1，可以通过TaskSchedulingProperties进行修改。代码如下：



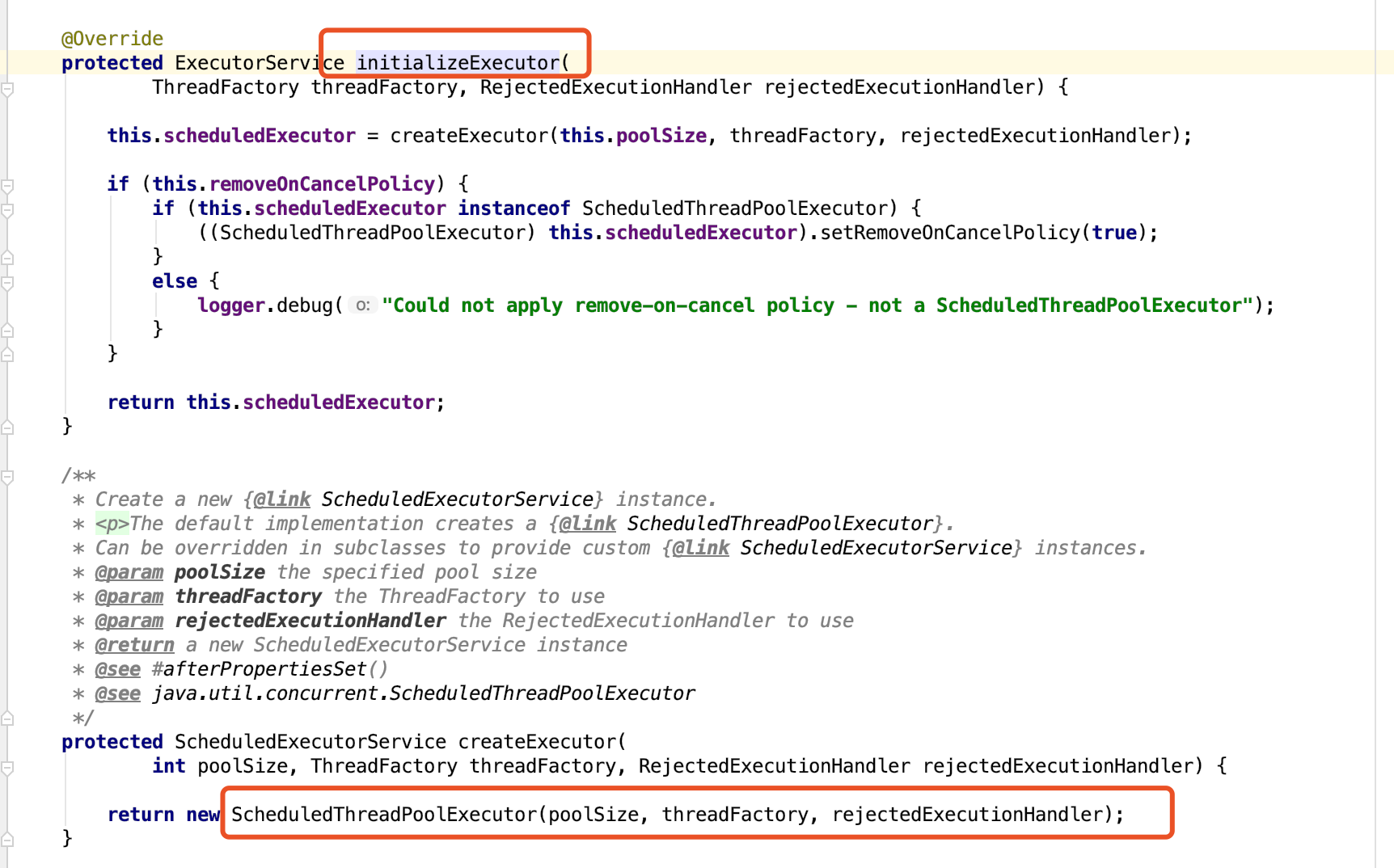


Spring提供的调度任务接口为TaskScheduler.class 其实现类有[ConcurrentTaskScheduler](https://docs.spring.io/spring/docs/5.1.5.RELEASE/javadoc-api/org/springframework/scheduling/concurrent/ConcurrentTaskScheduler.html" \o "class in org.springframework.scheduling.concurrent), [DefaultManagedTaskScheduler](https://docs.spring.io/spring/docs/5.1.5.RELEASE/javadoc-api/org/springframework/scheduling/concurrent/DefaultManagedTaskScheduler.html" \o "class in org.springframework.scheduling.concurrent), [ThreadPoolTaskScheduler](https://docs.spring.io/spring/docs/5.1.5.RELEASE/javadoc-api/org/springframework/scheduling/concurrent/ThreadPoolTaskScheduler.html" \o "class in org.springframework.scheduling.concurrent), [TimerManagerTaskScheduler](https://docs.spring.io/spring/docs/5.1.5.RELEASE/javadoc-api/org/springframework/scheduling/commonj/TimerManagerTaskScheduler.html" \o "class in org.springframework.scheduling.commonj)。

由于ThreadPoolTaskScheduler继承ExecutorConfigurationSupport

，ExecutorConfigurationSupport实现InitializingBean

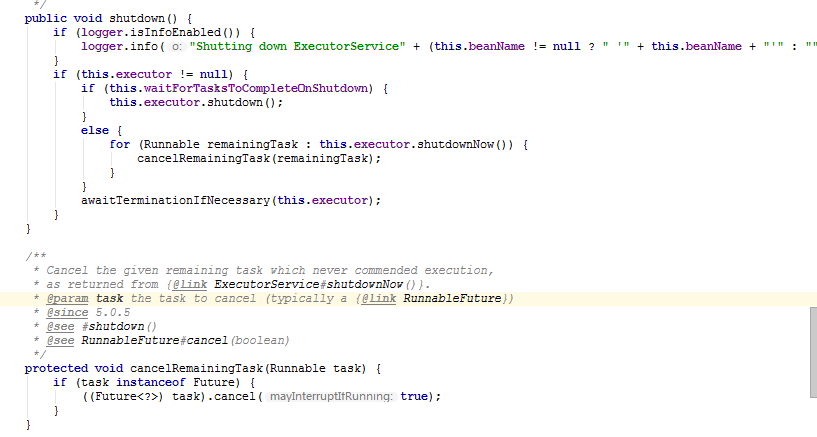
接口，在注入到springring容器后调用afterPropertiesSet方法，最终调用initializeExecutor方法，创建一个指定线程池大小，指定线程工程，的定时任务线程池ScheduledThreadPoolExecutor，代码如下：

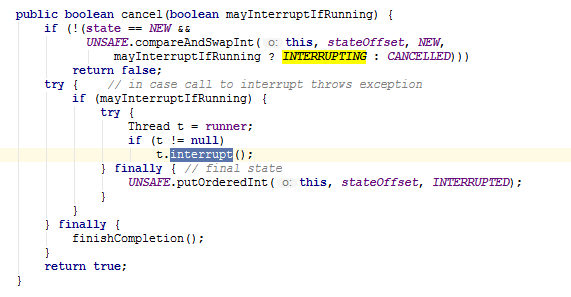


最后通过ScheduledThreadPoolExecutor调度我们的ScheduledMethodRunnable

在容器销毁时ThreadPoolTaskScheduler会进调用destroy方法进行线程池销毁，默认是调用shutdownnow方法，如果线程池中的线程还没有执行过则手动去调用cancel方法取消未执行的任务，也是就是调用FutureTask.cancel()方法，具体就是先判断线程状态是否是NEW也就是没有被执行，如果是，根据参数mayInterruptIfRunning，更新线程状态为INTERRUPTING或者为CANCELLED，如果mayInterruptIfRunning为true，则更新为INTERRUPTING否则更新为CANCELLED

如果线程状态不是NEW,则调用Thread.interrupt()方法尝试中断线程





补充:线程池的**shutDown()** ， **shutdownNow()** 方法区别

**shutDown()**   
  
    当线程池调用该方法时,线程池的状态则立刻变成SHUTDOWN状态。此时，则不能再往线程池中添加任何任务，否则将会抛出RejectedExecutionException异常。但是，此时线程池不会立刻退出，直到添加到线程池中的任务都已经处理完成，才会退出。   
  
**shutdownNow()**   
  
     根据JDK文档描述，大致意思是：执行该方法，线程池的状态立刻变成STOP状态，并试图停止所有正在执行的线程，不再处理还在池队列中等待的任务，当然，它会返回那些未执行的任务。   
     它试图终止线程的方法是通过调用Thread.interrupt()方法来实现的，但是大家知道，这种方法的作用有限，如果线程中没有sleep 、wait、Condition、定时锁等应用, interrupt()方法是无法中断当前的线程的。所以，ShutdownNow()并不代表线程池就一定立即就能退出，它可能必须要等待所有正在执行的任务都执行完成了才能退出。