<http://www.cnblogs.com/slimer/p/6222485.html>

# 1. 定时任务实现方式

**定时任务实现方式：**

* Java自带的java.util.Timer类，这个类允许你调度一个java.util.TimerTask任务。使用这种方式可以让你的程序按照某一个频度执行，但不能在指定时间运行。一般用的较少，这篇文章将不做详细介绍。
* 使用Quartz，这是一个功能比较强大的的调度器，可以让你的程序在指定时间执行，也可以按照某一个频度执行，配置起来稍显复杂，有空介绍。
* SpringBoot自带的Scheduled，可以将它看成一个轻量级的Quartz，而且使用起来比Quartz简单许多，本文主要介绍。

**定时任务执行方式：**

* 单线程（串行）
* 多线程（并行）

# 2. 创建定时任务

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | package com.autonavi.task.test;    import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.scheduling.annotation.Scheduled;  import org.springframework.stereotype.Component;    import com.autonavi.task.ScheduledTasks;    @Component  public class ScheduledTest {        private static final Logger logger = LoggerFactory.getLogger(ScheduledTasks.class);        @Scheduled(cron="0 0/2 \* \* \* ?")      public void executeFileDownLoadTask() {            // 间隔2分钟,执行任务          Thread current = Thread.currentThread();          System.out.println("定时任务1:"+current.getId());          logger.info("ScheduledTest.executeFileDownLoadTask 定时任务1:"+current.getId()+ ",name:"+current.getName());      }  } |

@Scheduled 注解用于标注这个方法是一个定时任务的方法，有多种配置可选。cron支持cron表达式，指定任务在特定时间执行；fixedRate以特定频率执行任务；fixedRateString以string的形式配置执行频率。

# 3. 启动定时任务

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | @SpringBootApplication  @EnableScheduling  public class App {        private static final Logger logger = LoggerFactory.getLogger(App.class);        public static void main(String[] args) {            SpringApplication.run(App.class, args);          logger.info("start");      }  } |

其中 @EnableScheduling 注解的作用是发现注解@Scheduled的任务并后台执行。

Springboot本身默认的执行方式是串行执行，也就是说无论有多少task，都是一个线程串行执行，并行需手动配置

# 4. 并行任务

继承SchedulingConfigurer类并重写其方法即可，如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | @Configuration  @EnableScheduling  public class ScheduleConfig implements SchedulingConfigurer {        @Override      public void configureTasks(ScheduledTaskRegistrar taskRegistrar) {          taskRegistrar.setScheduler(taskExecutor());      }        @Bean(destroyMethod="shutdown")      public Executor taskExecutor() {          return Executors.newScheduledThreadPool(100);      }  } |

再次执行之前的那个Demo，会欣然发现已经是并行执行了！

# 4. 异步并行任务

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | import org.springframework.scheduling.TaskScheduler;  import org.springframework.scheduling.annotation.AsyncConfigurer;  import org.springframework.scheduling.annotation.EnableAsync;  import org.springframework.scheduling.annotation.EnableScheduling;  import org.springframework.scheduling.annotation.SchedulingConfigurer;  import org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler;  import org.springframework.scheduling.config.ScheduledTaskRegistrar;    @Configuration  @EnableScheduling  @EnableAsync(  mode = AdviceMode.PROXY, proxyTargetClass = false,  order = Ordered.HIGHEST\_PRECEDENCE  )  @ComponentScan(  basePackages = "hello"  )  public class RootContextConfiguration implements  AsyncConfigurer, SchedulingConfigurer {  @Bean  public ThreadPoolTaskScheduler taskScheduler()  {  ThreadPoolTaskScheduler scheduler = new ThreadPoolTaskScheduler();  scheduler.setPoolSize(20);  scheduler.setThreadNamePrefix("task-");  scheduler.setAwaitTerminationSeconds(60);  scheduler.setWaitForTasksToCompleteOnShutdown(true);  return scheduler;  }    @Override  public Executor getAsyncExecutor()  {  Executor executor = this.taskScheduler();  return executor;  }    @Override  public void configureTasks(ScheduledTaskRegistrar registrar)  {  TaskScheduler scheduler = this.taskScheduler();  registrar.setTaskScheduler(scheduler);  }  } |

在启动的main方法加入额外配置：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | @SpringBootApplication  public class Application {        public static void main(String[] args) throws Exception {           AnnotationConfigApplicationContext rootContext =       new AnnotationConfigApplicationContext();        rootContext.register(RootContextConfiguration.class);      rootContext.refresh();      }  } |

标签: [springboot](http://www.cnblogs.com/slimer/tag/springboot/)