

springTask

一.简介

Spring Task 是 Spring 框架提供了一种任务调度和异步处理的解决方案。可以按照约定的时间自动执行某个代码逻辑它可以帮助开发者在 Spring 应用中轻松地实现定时任务、异步任务等功能，提高应用的效率和可维护性。

二.实现

在启动类中用注解`@EnableScheduling`进行标注，表明此类 存在定时任务。

不需要引入额外依赖，spring自带

一.基于注解`@Scheduled`

1.通过`@Scheduled`注释结合cron表达式实现

cron表达式:

cron表达式是一种用于设置定时任务的语法规则。它由6个字段组成，分别表示秒、分、小时、日期、月份和星期几。每个字段都可以设置一个数字、一组数字（用逗号分隔）、一段数字范围（用短横线分隔）、通配符（表示任意值）或者特定的字符（如星期几的英文缩写）。

语法规则：

<https://www.jianshu.com/p/e9ce1a7e1ed1>

示例：

0 0 0 * * ?：每天的零点整执行任务。

0 0 */2 * * ?：每隔2小时执行一次任务。

0 0 12 * * ?：每天中午12点执行任务。

0 15 10 * * ?：每天上午10点15分执行任务。

0 0 6,18 * * ?：每天的早上6点和晚上6点执行任务。

0 0/30 8-18 * * ?：每天的上午8点到下午6点之间，每隔30分钟执行一次任务。

0 0 0 1 1 ?：每年的1月1日零点整执行任务。

0 0 0 * * 2：每周的星期二零点整执行任务。

0 0 0 ? * 6#3：每月的第三个星期六零点整执行任务。

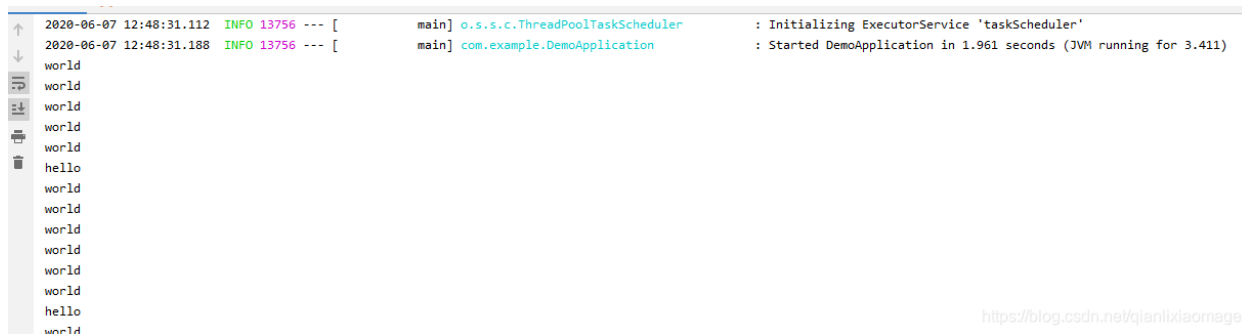
0 0 0 L * ?：每个月的最后一天零点整执行任务。

结合@Scheduled:

▼ @Scheduled注解示例

```
1
2 @Scheduled(cron = "*/6 * * * * ?")
3 public void sayHello() {
4     System.out.println("hello");
5 }
```

输出结果:



```
2020-06-07 12:48:31.112 INFO 13756 --- [main] o.s.s.c.ThreadPoolTaskScheduler : Initializing ExecutorService 'taskScheduler'
2020-06-07 12:48:31.188 INFO 13756 --- [main] com.example.DemoApplication : Started DemoApplication in 1.961 seconds (JVM running for 3.411)
world
world
world
world
world
world
hello
world
world
world
world
world
world
world
world
hello
world
```

注: 启动类需要能扫描到定时任务类, 否则定时任务启动不起来。

除了cron表达式外, 还支持 (感兴趣可以进一步了解)

1.**fixedRate**:控制方法执行的间隔时间, 是以上一次方法执行完开始算起, 如上一次方法执行阻塞住了, 那么直到上一次执行完, 并间隔给定的时间后, 执行下一次。

2.**initialDelay**: initialDelay = 10000 表示在容器启动后, 延迟10秒后再执行一次定时器。

优缺点: 添加注解即可, 使用方便。

但是: 1.@Scheduled作用在方法上, 方法不能有参数

2.@Scheduled注解只能在开始就写好, 无法动态定义

3.spring支持的springtask的cron语句无法识别年份, 也就是定时任务以固定频率执行, 无法做到只执行一次。

二.基于接口方式SchedulingConfigurer:

为了实现动态定义定时任务

一.创建数据库表和相应字段存放cron语句

▼ 初始化数据库表字段

```

1 drop table if exists scheduled;
2 create table scheduled (
3 cron_id varchar(30) NOT NULL primary key,
4 cron_name varchar(30) NULL,
5 cron varchar(30) NOT NULL
6 );
7 insert into scheduled values ('1','定时器任务一','0/6 * * * * ?');

```

二.新增mapper类获取数据库存放的cron表达式

```

1 @Select("select cron from cron_demo where cron_id=#{id}")
2 public String getCronById(int id);

```

三.新建task类执行定时任务

```

1 public class TaskDemo implements SchedulingConfigurer {
2     @Override
3     public void configureTasks(ScheduledTaskRegistrar taskRegistrar)
4     {
5         taskRegistrar.addTriggerTask();//or: addCronTask...
6     }
7     private void process(){
8         System.out.println("cron执行");
9     }//要执行的逻辑
10 }

```

注意实现[SchedulingConfigurer](#)接口

用于添加定时任务的方法在这里很多很灵活，如addTriggerTask,addCronTask，并且方法重载也较多，建议查看源码学习

这里介绍常用api:addTriggerTask，和addCronTask

addTriggerTask：


```

public void addTriggerTask(Runnable task, Trigger trigger) {
    this.addTriggerTask(new TriggerTask(task, trigger));
}

no usages
public void addTriggerTask(TriggerTask task) {
    if (this.triggerTasks == null) {
        this.triggerTasks = new ArrayList();
    }

    this.triggerTasks.add(task);
}

```



第一个方法实际是调用第二个方法

Runnable task为要执行的逻辑（想要定时实现的方法），Trigger trigger为使用某种方式封装的cron语句，介绍一个简单易懂好用的实现类---*CronTrigger*

```

public class CronTrigger implements Trigger {
    no usages
    private final CronExpression expression;
    no usages
    private final ZoneId zoneId;

    1 usage
    public CronTrigger(String expression) {
        this(expression, ZoneId.systemDefault());
    }

    no usages
    public CronTrigger(String expression, TimeZone timeZone) { this(expression, timeZone.toZoneId()); }

    2 usages
    public CronTrigger(String expression, ZoneId zoneId) {
        Assert.hasLength(expression, message: "Expression must not be empty");
        Assert.notNull(zoneId, message: "ZoneId must not be null");
        this.expression = CronExpression.parse(expression);
        this.zoneId = zoneId;
    }
}

```

expression为cron表达式，zonid为代表时区（不用管，会调用系统默认时区）

默认使用第一个构造方法即可

示例：

▼ cronTrigger示例

```

1 public class TaskDemo implements SchedulingConfigurer {
2     @Override
3     public void configureTasks(ScheduledTaskRegistrar taskRegistrar)
4         taskRegistrar.addTriggerTask(this::process,
5                                     new CronTrigger(cronMapper.getCronByic
6     } //要执行的逻辑
7     private void process(){

```

```

8         System.out.println("cron执行");
9     }

```

addCronTask:

```

1 usage
2
3 public void addCronTask(Runnable task, String expression) {
4     if (!"-".equals(expression)) {
5         this.addCronTask(new CronTask(task, expression));
6     }
7 }

```

```

1 no usages
2
3 public void addCronTask(CronTask task) {
4     if (this.cronTasks == null) {
5         this.cronTasks = new ArrayList();
6     }
7
8     this.cronTasks.add(task);
9 }

```

```

1 1 usage
2
3 public class CronTask extends TriggerTask {
4     no usages
5     private final String expression;
6
7     1 usage
8     public CronTask(Runnable runnable, String expression) { this(runnable, new CronTrigger(expression)); }
9
10    1 usage
11    public CronTask(Runnable runnable, CronTrigger cronTrigger) {
12        super(runnable, cronTrigger);
13        this.expression = cronTrigger.getExpression();
14    }
15
16    public String getExpression() { return this.expression; }
17 }

```

CronTask是TriggerTask的子类,其成员可谓非常人性化, expression即为cron表达式, 构造方法再传入runnable执行内容即可

示例:

▼ addCronTask

```

1 public class TaskDemo implements SchedulingConfigurer {
2     @Override
3     public void configureTasks(ScheduledTaskRegistrar taskRegistrar)
4     {

```

```

4      taskRegistrar.addCronTask(this::process,cronMapper.getCronById(1))
5      }//要执行的逻辑
6      private void process(){
7          System.out.println("cron执行");
8      }
9  }

```

从上示例看出，当用addTriggetTask时，如果用Crontask，用法和addCronTask差不多，这两个的底层都是调用了一个叫add的方法

扩展：Runnable runnable的写法

注：Runnable是线程的知识点，由于本人目前没有学习java线程部分，无法讲解其底层原理，只介绍在实现定时任务时的用法

Runnable只是一个接口，内部只有一个void的run方法

```

@FunctionalInterface
public interface Runnable {

    When an object implementing interface Runnable is used to create a thread, starting the thread
    causes the object's run method to be called in that separately executing thread.

    The general contract of the method run is that it may take any action whatsoever.

    See Also: Thread.run()

    public abstract void run();
}

```

需要定义实现类，如下在纳新大作业中的实现：

```

1 private class TaskRunnable implements Runnable{
2     private final Cron cron;
3
4     public TaskRunnable(Cron cron) {
5         this.cron = cron;
6     }
7
8     @Override
9     public void run() {
10         //定义任务要做的事，即把visibility字段设为0表示可见，
11         // 同时把时间设为设定的发送时间
12         // （如果设为当前时间由于定时任务管理器CronManageTask扫面时间间隔问
            题会导致实际执行时间与预期发送时间不一致）
13         mailboxService.lambdaUpdate()
14             .set(Email::getVisibility,(short)0)
15             .set(Email::getSendTime,cron.getExecuteTime())
16             .eq(Email::getId,cron.getEmailId())
17             .update();
18     }
19 }

```

```
20 }
```

这里的cron为从外传入的参数,可以通过构造方法将cron传入对象中,这就解决了@Scheduled无法传递参数的问题

示例:

```
1 @RequiredArgsConstructor
2 public class TaskDemo implements SchedulingConfigurer {
3     private final CronMapper cronMapper;
4     @Override
5     public void configureTasks(ScheduledTaskRegistrar taskRegistrar)
6     {
7         Runnable task = this::process;
8         CronTask cronTask = new
9         CronTask(task, cronMapper.getCronById(1));
10        taskRegistrar.addTriggerTask(cronTask);
11    }
12    private void process(){
13        System.out.println("cron执行");
14    } //要执行的逻辑
15
16 }
```

还有一个在查找博客时看到的示例,方法基本上一样只不过使用了lambda表达式,但是匿名内部类我只了解一点点,还请大佬help:

```
1 @Autowired
2 protected CronMapper cronMapper;
3
4 @Override
5 public void configureTasks(ScheduledTaskRegistrar
6     scheduledTaskRegistrar) {
7     scheduledTaskRegistrar.addTriggerTask(() -> process(),
8         triggerContext -> {
9             String cron = cronMapper.getCron(1);
10            if (cron.isEmpty()) {
11                System.out.println("cron is null");
12            }
13        }
14    }
15 }
```

```

12         return new
    CronTrigger(cron).nextExecutionTime(triggerContext);
13     });
14 }
15
16 private void process() {
17     System.out.println("基于接口定时任务");
18 }

```

三.基于ThreadPoolTaskScheduler轻量级多线程定时任务框架

上述基于接口的方法解决了基于注解无法实现的动态定义cron表达式和方法传入参数的问题，但示例无法实现根据传入的年份指定在某一年特定日期执行定时任务，下面介绍一种实现方式

一.简介：

- springboot中有一个bean，ThreadPoolTaskScheduler，可以很方便的对重复执行的任务进行调度管理；相比于通过java自带的周期性任务线程池
- ScheduleThreadPoolExecutor，此bean对象支持根据cron表达式创建周期性任务。
- 当然，ThreadPoolTaskScheduler其实底层使用也是java自带的线程池。

二.常用api介绍：

ThreadPoolTaskScheduler 内部方法非常丰富，本文实现的是一种cron表达式，周期执行

- schedule(Runnable task, Trigger trigger) cron表达式，周期执行
- schedule(Runnable task, Date startTime) 定时执行
- scheduleAtFixedRate(Runnable task, Date startTime, long period) 定时周期间隔时间执行。间隔时间单位 TimeUnit.MILLISECONDS
- scheduleAtFixedRate(Runnable task, long period) 间隔时间执行。单位毫秒

三.上实战

1.新建实现类cron

```

▼ cron.class
1  @Data
2  @AllArgsConstructor
3  @NoArgsConstructor
4  @TableName("thread_cron")
5  public class cron {
6      private String title;
7      private LocalDate startTime;//起始时间

```



```

8     private LocalDate deadTime;//结束时间
9     private LocalDateTime executeTime;//运行时间
10 }

```

解释：startTime为任务启动年份第一天，deadTime为任务启动年份最后一天(指定年份执行，也可以根据需求调整)，executeTime为任务执行时间

2.创建对应的service接口和实现类

```

▼
1 public interface ThreadService extends IService<Cron> {
2     void startCron(Cron cron);//启动定时任务
3     void stopCron(Cron cron);//停止定时任务
4     void changeCron(Cron cron);//更新定时任务
5 }

```

接下来是实现类的具体逻辑：

1.每个任务有一个执行期限，就是cron类中的startTime和deadTime，这里一般存储年份信息，将任务限定在某年执行，在启动定时任务也就是调用startCron方法时，需要判断当前时间是否在期限内

2.同一任务可能被多次启动，这显然是多余的，因此需要将已经启动过的定时任务放入一个集合中，在调用startCron时检查当前任务是否在集合中。执行定时任务的方法是ThreadPoolTaskScheduler中的public ScheduledFuture<?> schedule(Runnable task, Trigger trigger)这个方法，可以看到，方法参数在上面基于接口处讲过，方法返回值ScheduledFuture<?>包含执行的任务的详细信息，停止任务也需要调用其中的boolean cancel(boolean mayInterruptIfRunning)方法，因此，可以用此类型的集合来存放执行中的定时任务

示例：

▼ 准备

```

1 private final ThreadPoolTaskScheduler threadPoolTaskScheduler;
2 private final Map<Integer, ScheduledFuture<?>> futureMap = new
    HashMap<>();
3 @Bean
4 public ThreadPoolTaskScheduler threadPoolTaskScheduler() {
5     return new ThreadPoolTaskScheduler();
6 }

```

▼ startCron

```

1 public void startCron(Cron cron) {
2     //1.判断cron是否被执行过

```

```

3     if(futureMap.containsKey(cron.getId())){log.info("定时任务存在 id=
    {} ",cron.getId());return;}
4     //2.判断是否还没过执行时间
5     //在springTask中，cron表达式无法对年进行定时，故使用startTime和deadTime
    来限制定时任务要执行的年份
6     if(LocalDate.now().isEqual(cron.getStartTime()) ||
    LocalDate.now().isEqual(cron.getDeadTime()) ||
7         (LocalDate.now().isAfter(cron.getStartTime()) &&
    LocalDate.now().isBefore(cron.getDeadTime()))){
8         //提取执行时间
9         LocalDateTime executeTime = cron.getExecuteTime();
10        //组装cron表达式
11        DateTimeFormatter cronFormatter =
    DateTimeFormatter.ofPattern("s m H d M");
12        String cronExp = cronFormatter.format(executeTime)+" ?";
13        //执行scheduled任务
14        ScheduledFuture<?> future =
    threadPoolTaskScheduler.schedule(new TaskRunnable(cron), new
    CronTrigger(cronExp));
15        //将future传入futureMap集合表示任务启动，避免任务重复启动
16        futureMap.put(cron.getId(),future);
17        //输出日志
18        log.info("任务启动, id:{},executeTime:
    {} ",cron.getId(),cron.getExecuteTime());
19    }
20 }

```

▼ stopCron

```

1
2 public void stopCron(Cron cron) {
3     ScheduledFuture<?> future = futureMap.get(cron.getId());
4     if (future != null) {
5         future.cancel(true);
6         futureMap.remove(cron.getId());
7         log.info("任务停止, id:{},",cron.getId());
8     }
9 }

```

▼ changeCron

```

1 public void changeCron(Cron cron) {
2     startCron(cron);
3     stopCron(cron);
4 }

```

▼ TaskRunnable类

```

1 private class TaskRunnable implements Runnable{
2     private final Cron cron;
3     public TaskRunnable(Cron cron) {
4         this.cron = cron;
5     }
6     @Override
7     public void run() {
8         //定义任务要做的事
9         System.out.println("定时任务执行, id:"+cron.getId());
10    }
11 }

```

3.创建cronTaskManager类

注： cronTaskManager类上加注解@Component

上述解决了基于注解的三个问题，但是还存在一个问题，定时任务制定后被启用需要保持服务器或应用程序一直被启动，如果关闭应用程序，定时任务也将失效，因此需要一个类来管理定时任务，基本思路是：在应用启动时每隔一段时间扫描一边数据库存放的定时任务，将其启动或停止。

```

1 public class cronTaskManager {
2     @Lazy
3     private final ThreadService threadService;
4     //定义每半个小时扫描一次
5     @Scheduled(cron = "0 0/30 * * * ?")
6     public void cronManage() {
7         log.info("定时任务启动");
8         List<Cron> list = threadService.list();
9         list.forEach(cron -> {
10             if (LocalDate.now().isAfter(cron.getDeadTime())) {
11                 threadService.stopCron(cron);
12                 threadService.removeById(cron.getId());
13                 log.info("任务过期删除, id:{},executeTime:{},cron.getId()",cron.getId());
14             } else {
15                 log.info("尝试启动任务, id:{},executeTime:{},cron.getId()",cron.getId());
16                 threadService.startCron(cron);
17             }
18         });
19     }
20 }

```

启动应用定时启动ronManager方法扫描数据库存在的定时任务，如果任务过期则删除，否则尝试启动。

成功示例：

```
2024-04-22 17:44:00.008 INFO 20776 --- [ scheduling-1] o.e.s.demos.web.task.cronTaskManager : 定时任务启动
2024-04-22 17:44:00.100 INFO 20776 --- [ scheduling-1] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2024-04-22 17:44:00.411 INFO 20776 --- [ scheduling-1] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2024-04-22 17:44:00.489 INFO 20776 --- [ scheduling-1] o.e.s.demos.web.task.cronTaskManager : 尝试启动任务, id:1,executeTime:2024-04-22T17:45
2024-04-22 17:44:00.495 INFO 20776 --- [ scheduling-1] o.e.s.d.web.service.ThreadServiceImpl : 任务启动, id:1,executeTime:2024-04-22T17:45
2024-04-22 17:45:00.005 INFO 20776 --- [ scheduling-1] o.e.s.demos.web.task.cronTaskManager : 定时任务启动
2024-04-22 17:45:00.015 INFO 20776 --- [ scheduling-1] o.e.s.demos.web.task.cronTaskManager : 尝试启动任务, id:1,executeTime:2024-04-22T17:45
2024-04-22 17:45:00.016 INFO 20776 --- [ scheduling-1] o.e.s.d.web.service.ThreadServiceImpl : 定时任务存在 id=1
定时任务执行, id:1
```

参考博客：

[https://blog.csdn.net/qianlixiaomage/article/details/106599951?
spm=1001.2014.3001.5506](https://blog.csdn.net/qianlixiaomage/article/details/106599951?spm=1001.2014.3001.5506)

https://blog.csdn.net/qq_18948359/article/details/125499389?spm=1001.2014.3001.5506

https://blog.csdn.net/qq_44709990/article/details/123471552?spm=1001.2014.3001.5506

<https://developer.aliyun.com/article/1079232>

https://blog.csdn.net/qq_41144667/article/details/103937193