1.Quartz是什么

大部分公司都会用到**定时任务**这个功能。 拿火车票购票来说，当你下单后，后台就会插入一条待支付的task(job)，一般是30分钟，超过30min后就会执行这个job，去判断你是否支付，未支付就会取消此次订单；当你支付完成之后，后台拿到支付回调后就会再插入一条待消费的task（job），Job触发日期为火车票上的出发日期，超过这个时间就会执行这个job，判断是否使用等。

在我们实际的项目中，当Job过多的时候，肯定不能人工去操作，这时候就需要一个任务调度框架，帮我们自动去执行这些程序。那么该如何实现这个功能呢?

- Quartz是OpenSymphony开源组织在Job scheduling领域又一个开源项目,它可以与J2EE与J2SE应用程序相结合也可以单独使用。
- 在企业级应用中，经常会制定一些“计划任务”，即在某个时间点做某件事情，核心是以时间为关注点，**即在一个特定的时间点，系统执行指定的一个操作**
- 任务调度涉及多线程并发、线程池维护、运行时间规则解析、运行现场的保护以恢复等方面
- **Quartz框架是一个开源的企业级任务调度服务，已经被作为任务调度的良好解决方案**
- Quartz对任务调度进行了高度抽象，提出了3个核心概念，并在**org.quartz**包中通过接口和类进行了描述。

2.Quartz能干什么?

- 场景 #1: 下载交易流水

  > 每天晚上12整从银行系统自动下载当日交易流水，并进行备份!

- 场景 #2: 邮件提醒和告警

  > 公司出于安全考虑，让每个员工三个月换一次邮箱密码。这种情况下，可以创建一个作业，让它每天午夜运行一次，并且向离过期时间不到三天的所有用户发邮件提醒。这里可以恰到好处的用到作业调度器。图 1.1 描绘了密码这个提醒作业。

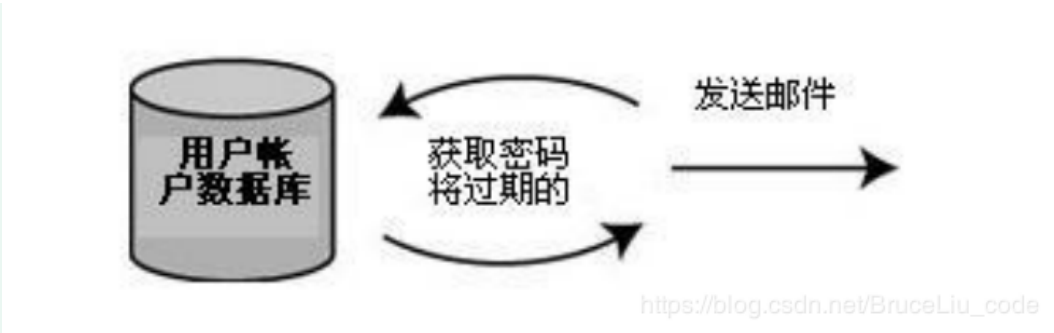


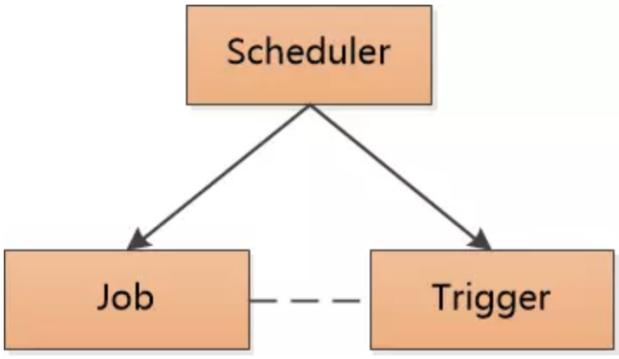
图 1.1 密码过期的作业每晚发送邮件给密码很快会过期的用户

会员续费：

> 会员到期的前一周，发送短信提示会员，进行续费！腾讯云服务器，提前一周发送短信，提示续费，如果不续费那么服务器上资源(默认保存一周)，如果还不续费直接收回服务器！

3.Quartz核心概念

- **任务**(JOB)：就是执行的工作内容。Quartz提供**Job接口**来支持任务定义
- **触发器(Trigger)**(何时触发的规则)：定义触发Job执行的时间触发规则。Quartz提供**Trigger类**及其子类支持触发器功能
- **调度器(Scheduler)**：Quartz提供了Scheduler接口，将工作任务和触发器绑定，保证任务可以在正确的时间执行



4.Quartz下载

下载： http://www.quartz-scheduler.org/download/index.html

- docs/wikidocs Quartz 的帮助文件
- docs/api Quartz 框架的JavaDoc Api 说明文档
- Examples Quartz 的例子
- Lib Quartz 使用到的第三方包
- src 源码

5.HelloWorld程序

- 创建maven工程，导入依赖

```
<dependency>
    <groupId>org.quartz-scheduler</groupId>
    <artifactId>quartz</artifactId>
    <version>2.2.1</version>
</dependency>
<dependency>
    <groupId>org.quartz-scheduler</groupId>
    <artifactId>quartz-jobs</artifactId>
    <version>2.2.1</version>
</dependency>
```

一个最简单的应用包含两个类:

- HelloJob.java
  任务类,需要实现 job 接口,里面实现了你想要 定时执行的方法
- SimpleExample.java 主线程，用于任务的调度,执行等操作.

上代码:

- HelloJob.java

```
/**
 * Description: 打印任意内容
 */
public class HelloJob implements Job{

    @Override
    public void execute(JobExecutionContext jobExecutionContext)
throws JobExecutionException {
        String printTime = new SimpleDateFormat("yy-MM-dd HH-mm-
ss").format(new Date());
        System.out.println("HelloJob start at:" + printTime + ",
prints: Hello Job-" + new Random().nextInt(100));

    }
}
```

任务很简单，HelloJob 实现了Quartz 的Job接口中的execute方法，任务的内容就是打印一行字。具体这个任务应该何时去执行，都在SimpleExample.java里面去定义实现。

- SimpleExample.java

```
public class SimpleExample{
```

```
2
3       public static void main(String[] args) throws
    SchedulerException, InterruptedException {
4           // 1、创建调度器Scheduler
5           SchedulerFactory schedulerFactory = new
    StdSchedulerFactory();
6           Scheduler scheduler = schedulerFactory.getScheduler();
7           // 2、创建JobDetail实例，并与PrintWordsJob类绑定(Job执行内容)
8           JobDetail jobDetail = JobBuilder.newJob(HelloJob.class)
9                                           .withIdentity("job1",
    "group1").build();
10          // 3、构建Trigger实例,每隔1s执行一次
11          Trigger trigger =
    TriggerBuilder.newTrigger().withIdentity("trigger1",
    "triggerGroup1")
12                  .startNow()//立即生效
13
    .withSchedule(SimpleScheduleBuilder.simpleSchedule()
14                  .withIntervalInSeconds(1)//每隔1s执行一次
15                  .repeatForever()).build();//一直执行
16
17          //4、执行
18          scheduler.scheduleJob(jobDetail, trigger);
19          System.out.println("--------scheduler start ! ----------
    -");
20          scheduler.start();
21
22          //睡眠
23          TimeUnit.MINUTES.sleep(1);
24          scheduler.shutdown();
25          System.out.println("--------scheduler shutdown ! --------
    ----");
26      }
27  }
28
29
```

通过代码可以看到几个重要的类：

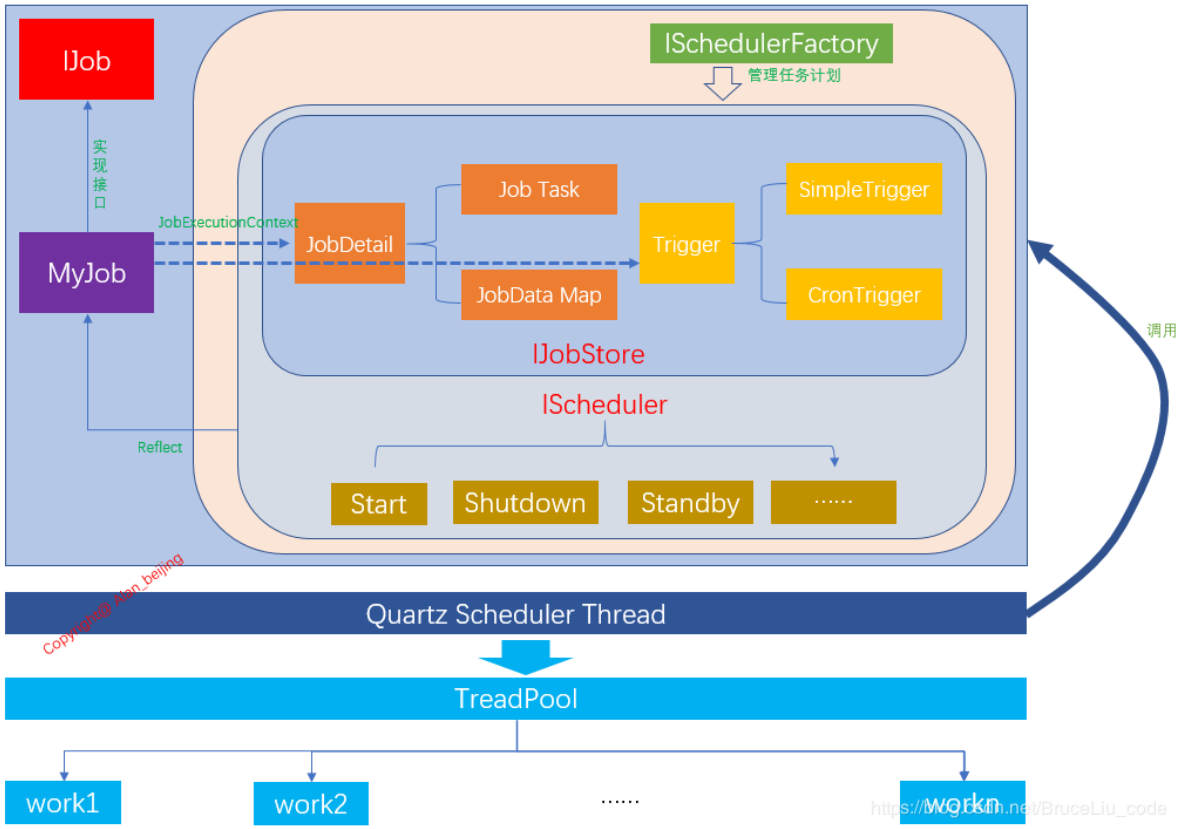
- JobDetail： 真正的任务内容，任务本身是集成Job接口的，但是真正的任务是JobBuilder通过反射的方式实例化的，
- Trigger： 触发器，定义任务应当开始的时间，主要分为两类**SimpleTrigger**,**CronTrigger**，当前例子的就是简单触发器，CronTrigger主要用于处理quartz表达式定义的任务，比如每个月20号，每个星期一之类的。
- Scheduler： 计划执行者，现在我们有了要做的内容(HelloJob)，有了要做的时间(下一分钟)，接下来，就把这两个内容填充到计划任务Scheduler对象里面，到了

时间它就可以自动运行了.

6.Quart架构

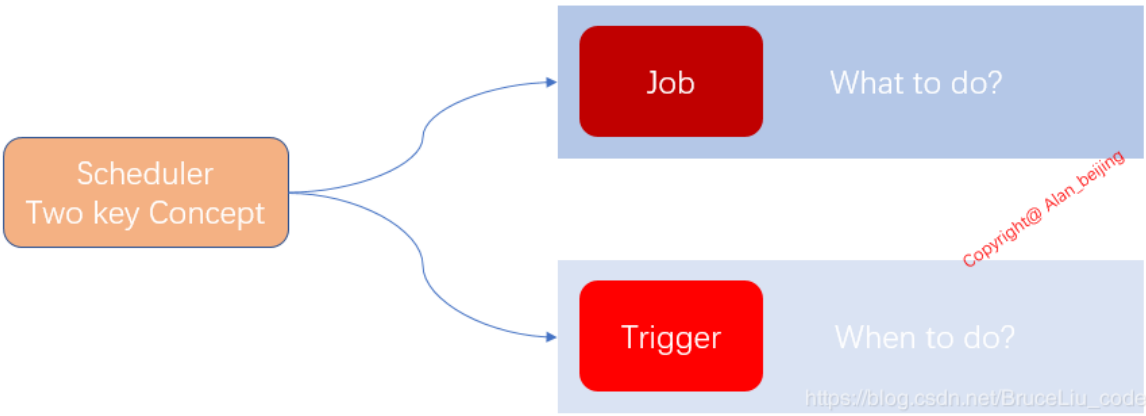
当要深入研究一个技术时，研究它的体系结构和内部运行原理，不失为一种较好的方式。同理，我们在研究Quartz时，也采用类似的方法，

下图为Quartz的大致结构图。

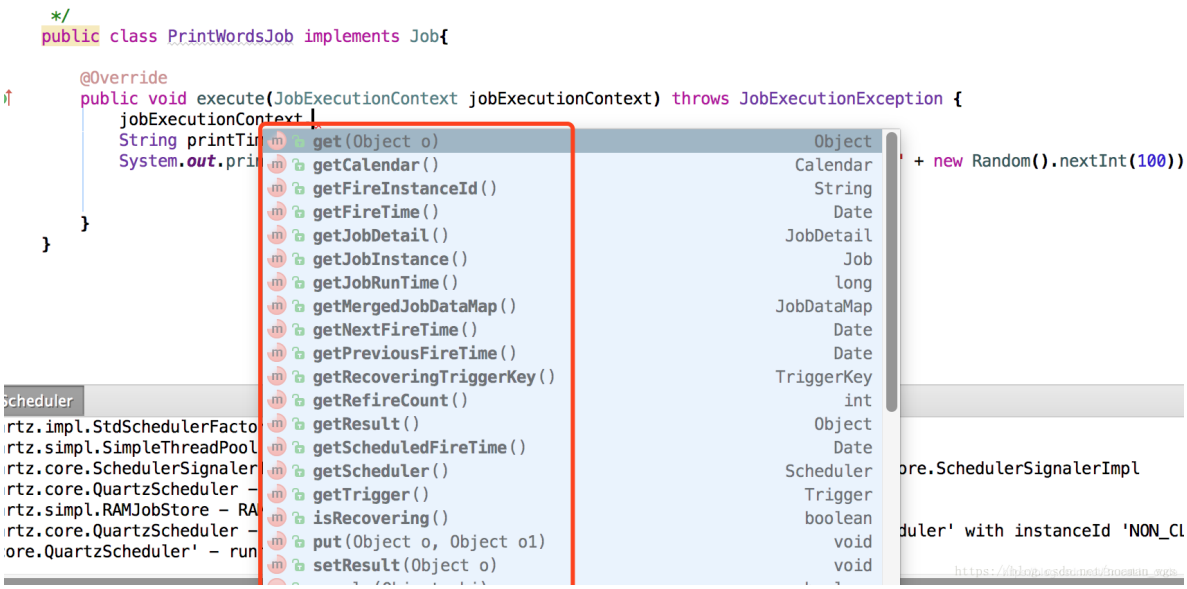


6.1.Quartz关键组件

Quartz比较关键的两个核心组件分别为Job和Trigger job--表示任务是什么 trigger--表示何时触发任务



JobExecutionContext JobExecutionContext中包含了Quartz运行时的环境以及Job本身的详细数据信息。 当Schedule调度执行一个Job的时候，就会将JobExecutionContext传递给该Job的execute()中，Job就可以通过JobExecutionContext对象获取信息。 主要信息有



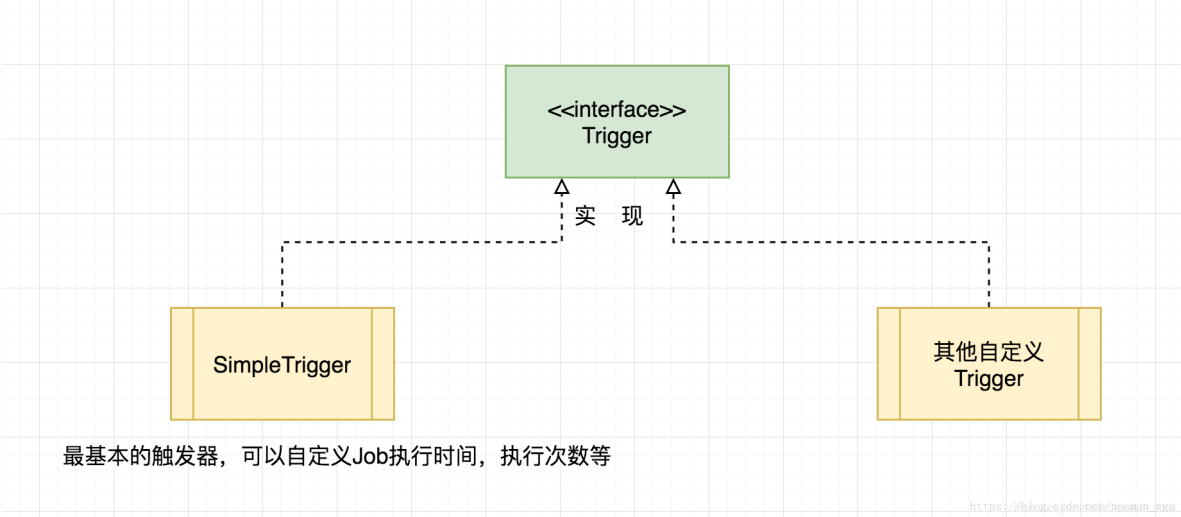
JobDataMap JobDataMap实现了JDK的Map接口，可以以Key-Value的形式存储数据。JobDetail、Trigger都可以使用JobDataMap来设置一些参数或信息， Job执行execute()方法的时候，JobExecutionContext可以获取到JobExecutionContext中的信息： 如：

```
 // 2、创建JobDetail实例，并与PrintWordsJob类绑定(Job执行内容)
 JobDetail jobDetail = JobBuilder.newJob(HelloJob.class).usingJobData("jobDetail1", "这个Job用来测试的")
         .withIdentity("job1", "group1").build();

 Trigger trigger = TriggerBuilder.newTrigger().withIdentity("trigger1", "triggerGroup1")
         .usingJobData("trigger1", "这是jobDetail1的trigger")
         .startNow()//立即生效
         .withSchedule(SimpleScheduleBuilder.simpleSchedule()
                 .withIntervalInSeconds(1)//每隔1s执行一次
                 .repeatForever()).build();//一直执行
```

Job执行的时候，可以获取到这些参数信息：

```
    @Override
    public void execute(JobExecutionContext jobExecutionContext) throws JobExecutionException {


     System.out.println(jobExecutionContext.getJobDetail().getJobDataMap().get("jobDetail1"));

     System.out.println(jobExecutionContext.getTrigger().getJobDataMap().get("trigger1"));
            String printTime = new SimpleDateFormat("yy-MM-dd HH-mm-ss").format(new Date());
            System.out.println("HelloJobstart at:" + printTime + ", prints: Hello Job-" + new Random().nextInt(100));

    }
```

7.Trigger、SimpleTrigger、CronTrigger



Trigger Trigger是Quartz的触发器，会去通知Scheduler何时去执行对应Job。

```
new Trigger().startAt():表示触发器首次被触发的时间;
new Trigger().endAt():表示触发器结束触发的时间;
```

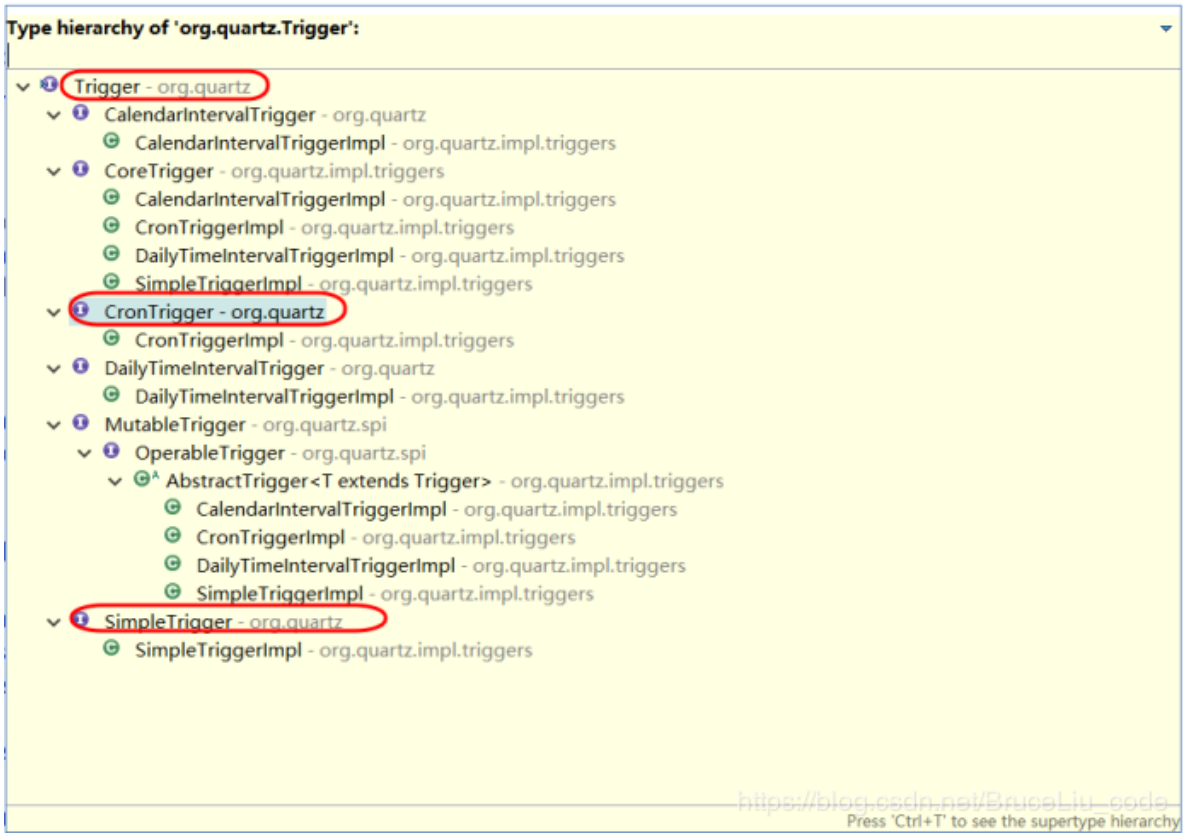
SimpleTrigger SimpleTrigger可以实现在一个指定时间段内执行一次作业任务或一个时间段内多次执行作业任务。 下面的程序就实现了程序运行5s后开始执行Job，执行Job 5s后结束执行：

```
 Date startDate = new Date();
 startDate.setTime(startDate.getTime() + 5000);

  Date endDate = new Date();
  endDate.setTime(startDate.getTime() + 5000);

```

```
7           Trigger trigger =
    TriggerBuilder.newTrigger().withIdentity("trigger1",
    "triggerGroup1")
8                   .usingJobData("trigger1", "这是jobDetail1的
    trigger")
9                   .startNow()//立即生效
10                  .startAt(startDate)
11                  .endAt(endDate)
12
    .withSchedule(SimpleScheduleBuilder.simpleSchedule()
13                  .withIntervalInSeconds(1)//每隔1s执行一次
14                  .repeatForever()).build();//一直执行
15
```

7.1 CronTrigger介绍

CronTrigger也是Trigger的子类



CronTrigger功能非常强大，是基于**日历**的作业调度，而SimpleTrigger是精准指定间隔，所以相比SimpleTrigger，CroTrigger更加常用。CroTrigger是基于Cron表达式的，先了解下Cron表达式： 由7个子表达式组成字符串的。

CronTrigger和SimpleTrigger的对比

触发器	应用场景	使用方式
SimpleTrigger	固定时间间隔的调度任务（例如：每隔2小时执行1次）	通过设置触发器的属性：开始时间、结束时间、重复次数、重复间隔等
CronTrigger	指定时间点的调度任务（例如：每天凌晨1:00执行1次）	通过定义Cron表达式

CronTrigger允许用户更精准地控制任务的运行日期和时间，而不仅仅是定义工作的频度 CronTrigger通过Cron表达式定义准确的运行时间点。创建CronTrigger的语法如下： 创建CronTrigger的语法很简单，最关键的是Cron表达式的编写

要使用CronTrigger，必须掌握Cron表达式

```
CronTrigger cronTrig = new CronTrigger("触发器名", "组名", "Cron表达式");
```

Cron表达式由6~7个由空格分隔的时间元素组成。第7个元素可选

字段	允许值	允许的特殊字符
秒（Seconds）	0~59的整数	, - * /　四个字符
分（*Minutes*）	0~59的整数	, - * /　四个字符
小时（*Hours*）	0~23的整数	, - * /　四个字符
日期（*DayofMonth*）	1~31的整数（但是你需要考虑你月的天数）	,- * ? / L W C　八个字符
月份（*Month*）	1~12的整数或者 JAN-DEC	, - * /　四个字符
星期（*DayofWeek*）	1~7的整数或者 SUN-SAT （1=SUN）	, - * ? / L C #　八个字符
年(可选，留空)（*Year*）	1970~2099	, - * /　四个字符

位置	字段含义	范围	允许的特殊字符
1	秒	0~59	* /
2	分钟	0~59	* /
3	小时	0~23	* /
4	月份中的哪一天	1~31	* / ? L
5	月份	1~12 或 JAN~DEC	* /
6	星期几	1~7 或 SUN~SAT	* / ? L #
7	年份	1970~2099	* /

Cron表达式的每个字段，都可以显式地规定一个值（如49）、一个范围（如1-6）、一个列表（如1,3,5）或者一个通配符（如*）

Cron表达式有几个特殊的字符，说明如下“ - ”：中划线，表示一个范围 1-10“ , ”：使用逗号间隔的数据，表示一个列表 1,8,10“ * ”：表示每一个值，它可以用于所有字段。例如：在小时字段表示每小时 * “ ? ”：该字符仅用于“月份中的哪一天”字段和“星期几”字段，表示不指定值“ / ”：通常表示为x/y，x为起始值，y表示值的增

量。“ L ”：表示最后一天，仅在日期和星期字段中使用“ # ”：只能用于“星期几”字段，表示这个月的第几个周几。例如：“6#3”指这个月第三个周五

7.2 Cron表达式示例

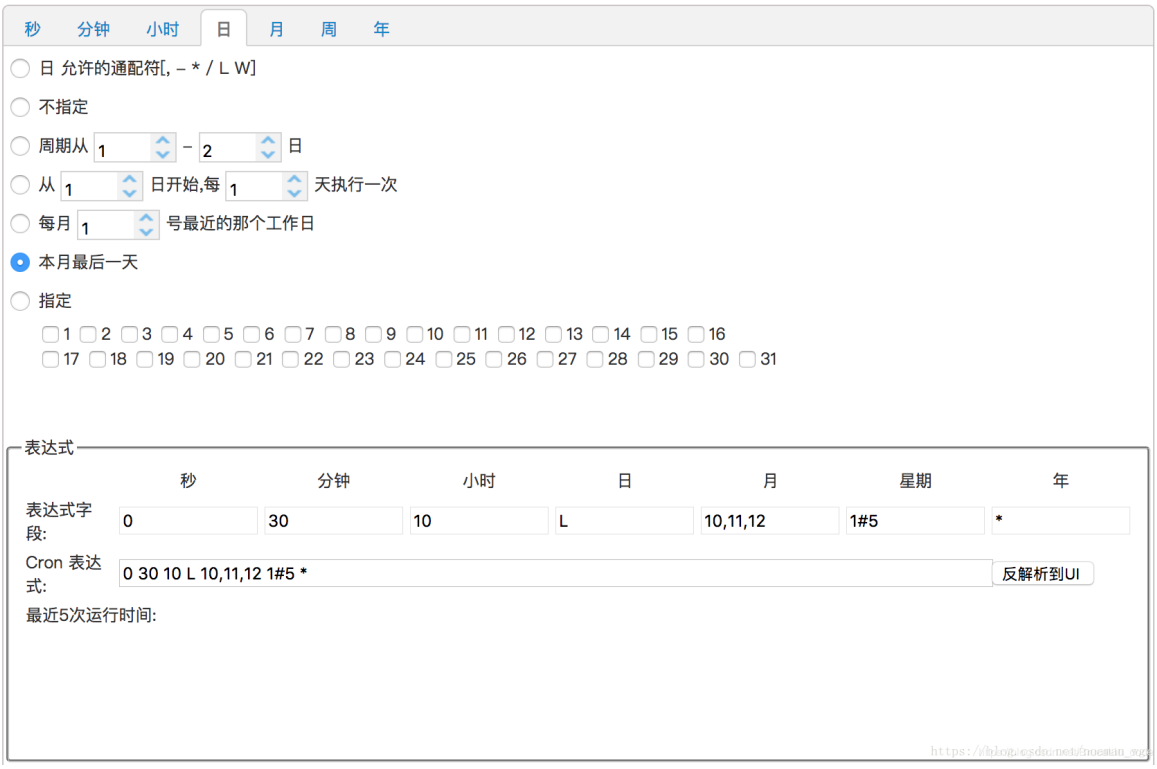
语法格式：

> 秒 分 时 天 月 星期 年

常用表达式例子 （1）0 0 2 1 * ? * 表示在每月的1日的凌晨2点调整任务 （2）0 15 10 ? * MON-FRI 表示周一到周五每天上午10:15执行作业 （3）0 15 10 ? 6L 2002-2006 表示2002-2006年的每个月的最后一个星期五上午10:15执行作 （4）0 0 10,14,16 * * ? 每天上午10点，下午2点，4点 （5）0 0/30 9-17 * * ? 朝九晚五工作时间内每半小时 （6）0 0 12 ? * WED 表示每个星期三中午12点 （7）0 0 12 * * ? 每天中午12点触发 （8）0 15 10 ? * * 每天上午10:15触发 （9）0 15 10 * * ? 每天上午10:15触发 （10）0 15 10 * * ? * 每天上午10:15触发 （11）0 15 10 * * ? 2005 2005年的每天上午10:15触发 （12）0 * 14 * * ? 在每天下午2点到下午2:59期间的每1分钟触发 （13）0 0/5 14 * * ? 在每天下午2点到下午2:55期间的每5分钟触发 （14）0 0/5 14,18 * * ? 在每天下午2点到2:55期间和下午6点到6:55期间的每5分钟触发 （15）0 0-5 14 * * ? 在每天下午2点到下午2:05期间的每1分钟触发 （16）0 10,44 14 ? 3 WED 每年三月的星期三的下午2:10和2:44触发 （17）0 15 10 ? * MON-FRI 周一至周五的上午10:15触发 （18）0 15 10 15 * ? 每月15日上午10:15触发 （19）0 15 10 L * ? 每月最后一日的上午10:15触发 （20）0 15 10 ? * 6L 每月的最后一个星期五上午10:15触发 （21）0 15 10 ? * 6L 2002-2005 2002年至2005年的每月的最后一个星期五上午10:15触发 （22）0 15 10 ? * 6#3 每月的第三个星期五上午10:15触发

表示式	说明
0 0 12 * * ?	每天12点运行
0 15 10 ? * *	每天10:15运行
0 15 10 * * ?	每天10:15运行
0 15 10 * * ? *	每天10:15运行
0 15 10 * * ? 2008	在2008年的每天10：15运行
0 * 14 * * ?	每天14点到15点之间每分钟运行一次，开始于14:00，结束于14:59。
0 0/5 14 * * ?	每天14点到15点每5分钟运行一次，开始于14:00，结束于14:55。
0 0/5 14,18 * * ?	每天14点到15点每5分钟运行一次，此外每天18点到19点每5钟也运行一次。
0 0-5 14 * * ?	每天14:00点到14:05，每分钟运行一次。
0 10,44 14 ? 3 WED	3月每周三的14:10分到14:44，每分钟运行一次。
0 15 10 ? * MON-FRI	每周一，二，三，四，五的10:15分运行。
0 15 10 15 * ?	每月15日10:15分运行。
0 15 10 L * ?	每月最后一天10:15分运行。
0 15 10 ? * 6L	每月最后一个星期五10:15分运行。
0 15 10 ? * 6L 2007-2009	在2007,2008,2009年每个月的最后一个星期五的10:15分运行。
0 15 10 ? * 6#3	每月第三个星期五的10:15分运行。

Cron表达式在线插件： http://cron.qqe2.com/



7.3 Cron表达式案例

下面的代码就实现了每周一到周五上午10:30执行定时任务

```
1  public class Test3 {
2
3      public static void main(String[] args) throws
   SchedulerException, InterruptedException {
4          // 1、创建调度器Scheduler
5          SchedulerFactory schedulerFactory = new
   StdSchedulerFactory();
6          Scheduler scheduler = schedulerFactory.getScheduler();
7          // 2、创建JobDetail实例，并与PrintWordsJob类绑定(Job执行内容)
8          JobDetail jobDetail = JobBuilder.newJob(HelloJob.class)
9                  .usingJobData("jobDetail1", "这个Job用来测试的")
10                 .withIdentity("job1", "group1").build();
11         // 3、构建Trigger实例,每隔1s执行一次
12         Date startDate = new Date();
13         startDate.setTime(startDate.getTime() + 5000);
14
15         Date endDate = new Date();
16         endDate.setTime(startDate.getTime() + 5000);
17
18         CronTrigger cronTrigger =
   TriggerBuilder.newTrigger().withIdentity("trigger1",
   "triggerGroup1")
19                 .usingJobData("trigger1", "这是jobDetail1的
   trigger")
```

```
                .startNow()//立即生效
                .startAt(startDate)
                .endAt(endDate)
                .withSchedule(CronScheduleBuilder.cronSchedule("*
30 10 ? * 1/5 2018"))
                .build();

        //4、执行
        scheduler.scheduleJob(jobDetail, cronTrigger);
        System.out.println("--------scheduler start ! ----------
-");
        scheduler.start();
        System.out.println("--------scheduler shutdown ! --------
----");

    }

}
```