[1. 今日大纲 1](#_Toc11845)

[2. Quartz 2](#_Toc10671)

[2.1. 大纲 2](#_Toc19745)

[2.2. Quartz是什么 2](#_Toc26992)

[2.3. 下载 2](#_Toc27010)

[2.4. 核心接口 3](#_Toc28542)

[2.5. Tigger 4](#_Toc31590)

[2.5.1. SimpleTrigger 5](#_Toc11466)

[2.5.2. CronTrigger 5](#_Toc19767)

[2.6. Cron Expressions 6](#_Toc29264)

[2.7. 表达式生成工具 7](#_Toc17243)

[3. 学习Quartz 8](#_Toc21737)

[3.1. 导入itcast-quartz 8](#_Toc290)

[3.2. 依赖 8](#_Toc19709)

[3.3. 示例 8](#_Toc24136)

[3.3.1. Job 8](#_Toc22660)

[3.3.2. 简单触发示例 9](#_Toc31078)

[3.3.3. 表达式触发示例 11](#_Toc23085)

[3.4. 通过Spring使用Quartz 13](#_Toc22275)

[3.4.1. 导入依赖 13](#_Toc1706)

[3.4.2. 编写Job 14](#_Toc14894)

[3.4.3. 编写配置文件文 14](#_Toc31086)

[3.4.4. 启动Spring容器（启动调度器） 15](#_Toc2743)

[3.5. 触发和job之间关系 15](#_Toc5227)

[4. 关闭超时2天未付款的订单 16](#_Toc32403)

[4.1. 分析 16](#_Toc10884)

[4.2. 实现 17](#_Toc9957)

[4.2.1. 导入依赖 17](#_Toc13013)

[4.2.2. 定义job 17](#_Toc18112)

[4.2.3. 定义触发 18](#_Toc29440)

[4.2.4. 定义调度器 18](#_Toc18600)

[4.2.5. OrderMapper中的实现 19](#_Toc16674)

# 今日大纲

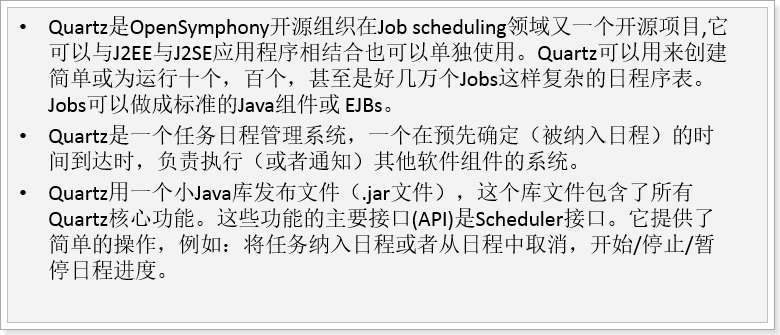
1. 学习Quartz
2. 基于Quartz实现定时关闭超时未付款的订单
3. 基于Solr实现商品的搜索
   1. 爬虫抓到的京东的商品数据、图片

# Quartz

## 大纲

* Quartz是什么？
* 简单的一些例子
* Quartz框架学习

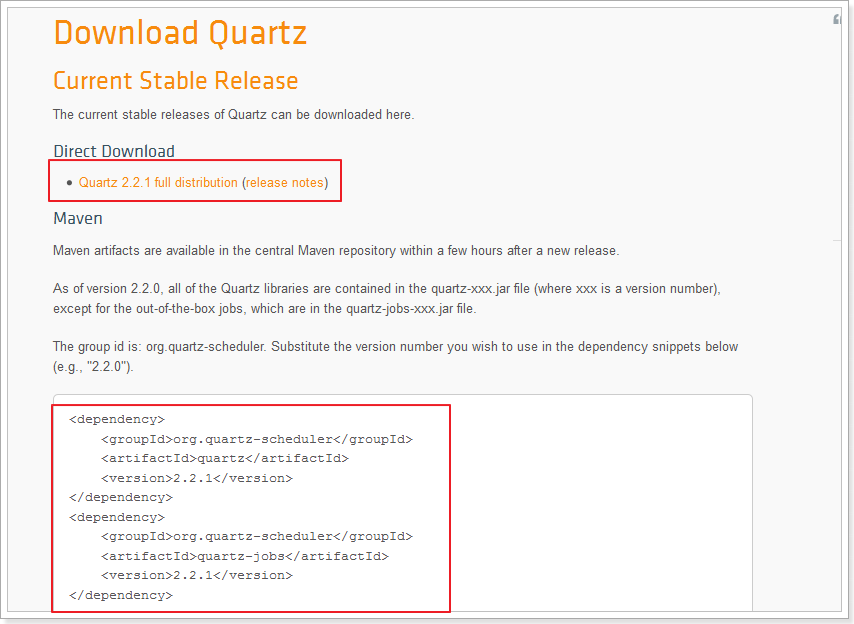
## Quartz是什么



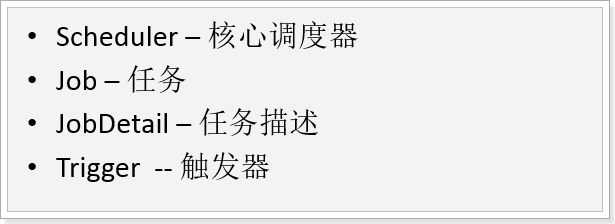
简单的认为Quartz是是一个定时器。

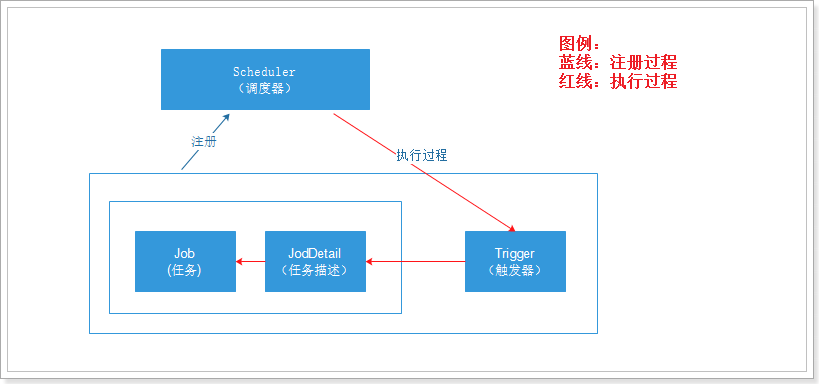
## 下载

* <http://www.quartz-scheduler.org/downloads>



## 核心接口

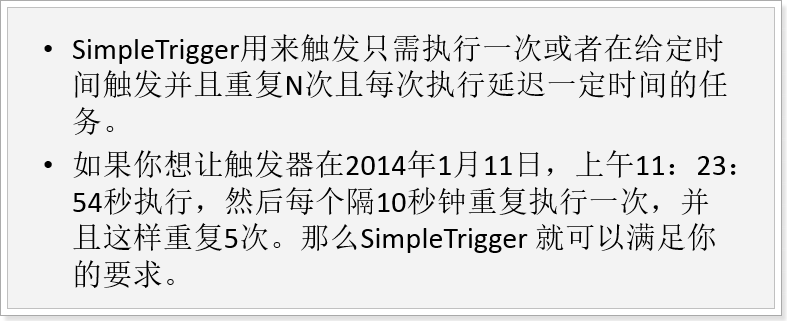




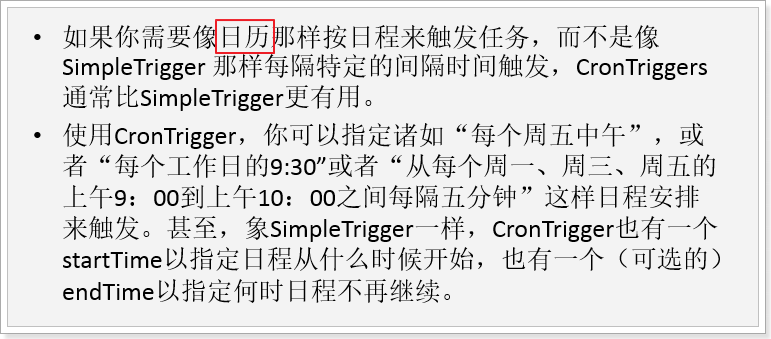
## Tigger

* SimpleTrigger
  + 简单的触发
* CronTrigger
  + 表达式触发

### SimpleTrigger



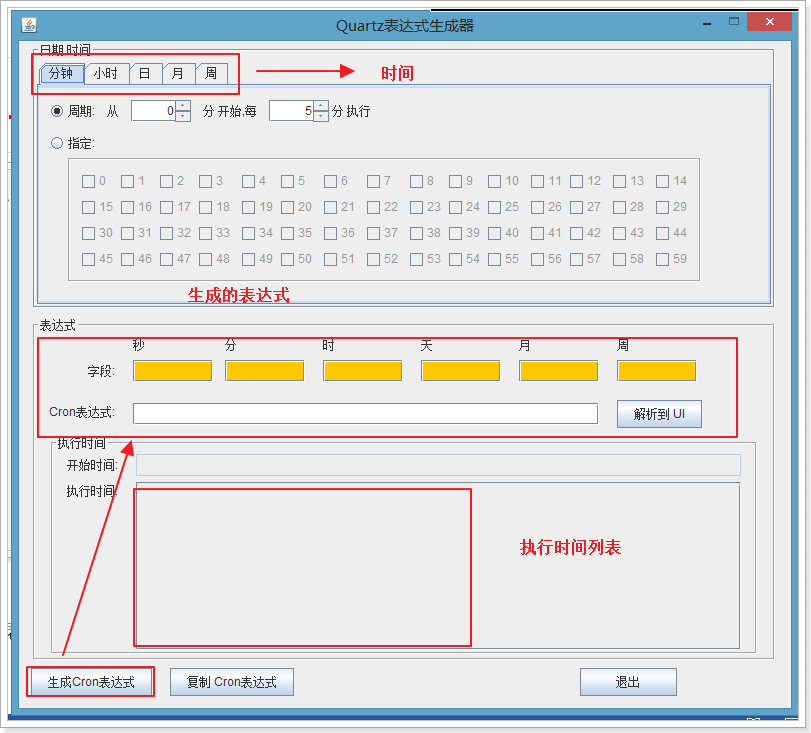
### CronTrigger



## Cron Expressions



## 表达式生成工具

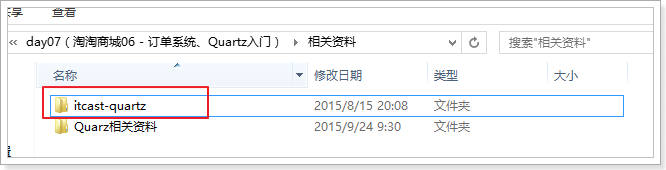


这个工具不完美：

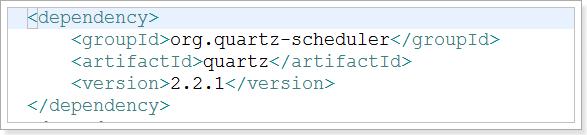
1. 缺少秒
2. 缺少L、W表达式

# 学习Quartz

## 导入itcast-quartz



## 依赖



## 示例

### Job

**public** **class** HelloJob **implements** Job {

**private** **static** Logger *\_log* = LoggerFactory.*getLogger*(HelloJob.**class**);

/\*\*

\* <p>

\* Empty constructor for job initilization

\* </p>

\* <p>

\* Quartz requires a public empty constructor so that the

\* scheduler can instantiate the class whenever it needs.

\* </p>

\*/

**public** HelloJob() {

}

/\*\*

\* <p>

\* Called by the <code>{@link org.quartz.Scheduler}</code> when a

\* <code>{@link org.quartz.Trigger}</code> fires that is associated with

\* the <code>Job</code>.

\* </p>

\*

\* **@throws** JobExecutionException

\* if there is an exception while executing the job.

\*/

**public** **void** execute(JobExecutionContext context)

**throws** JobExecutionException {

// Say Hello to the World and display the date/time

*\_log*.info("Hello World! - " + **new** Date());

}

}

### 简单触发示例

**public** **class** SimpleExample {

**public** **void** run() **throws** Exception {

Logger log = LoggerFactory.*getLogger*(SimpleExample.**class**);

log.info("------- Initializing ----------------------");

// 定义调度器

SchedulerFactory sf = **new** StdSchedulerFactory();

Scheduler sched = sf.getScheduler();

log.info("------- Initialization Complete -----------");

// 获取当前时间的下一分钟

Date runTime = *evenMinuteDate*(**new** Date());

log.info("------- Scheduling Job -------------------");

// 定义job

// 在quartz中，有组的概念，组+job名称 唯一的

JobDetail job = *newJob*(HelloJob.**class**).withIdentity("job1", "group1").build();

// 定义触发器，在下一分钟启动

Trigger trigger = *newTrigger*().withIdentity("trigger1", "group1").startAt(runTime).build();

// 将job注册到调度器

sched.scheduleJob(job, trigger);

log.info(job.getKey() + " will run at: " + runTime);

// 启动调度器

sched.start();

log.info("------- Started Scheduler -----------------");

// 等待65秒

log.info("------- Waiting 65 seconds... -------------");

**try** {

// wait 65 seconds to show job

Thread.*sleep*(65L \* 1000L);

// executing...

} **catch** (Exception e) {

//

}

// 关闭调度器

log.info("------- Shutting Down ---------------------");

sched.shutdown(**true**);

log.info("------- Shutdown Complete -----------------");

}

**public** **static** **void** main(String[] args) **throws** Exception {

SimpleExample example = **new** SimpleExample();

example.run();

}

}

### 表达式触发示例

**public** **class** SimpleCronExample {

**public** **void** run() **throws** Exception {

Logger log = LoggerFactory.*getLogger*(SimpleCronExample.**class**);

log.info("------- Initializing ----------------------");

// 定义调度器

SchedulerFactory sf = **new** StdSchedulerFactory();

Scheduler sched = sf.getScheduler();

log.info("------- Initialization Complete -----------");

// 获取当前时间的下一分钟

Date runTime = *evenMinuteDate*(**new** Date());

log.info("------- Scheduling Job -------------------");

// 定义job

JobDetail job = *newJob*(HelloJob.**class**).withIdentity("job1", "group1").build();

// 定义触发器，每2秒执行一次

Trigger trigger = *newTrigger*().withIdentity("trigger1", "group1")

.withSchedule(*cronSchedule*("0/5 \* \* \* \* ?")).build();

// 将job注册到调度器

sched.scheduleJob(job, trigger);

log.info(job.getKey() + " will run at: " + runTime);

// 启动调度器

sched.start();

log.info("------- Started Scheduler -----------------");

// 等待1分钟

log.info("------- Waiting 60 seconds... -------------");

**try** {

Thread.*sleep*(60L \* 1000L);

} **catch** (Exception e) {

//

}

// 关闭调度器

log.info("------- Shutting Down ---------------------");

sched.shutdown(**true**);

log.info("------- Shutdown Complete -----------------");

}

**public** **static** **void** main(String[] args) **throws** Exception {

SimpleCronExample example = **new** SimpleCronExample();

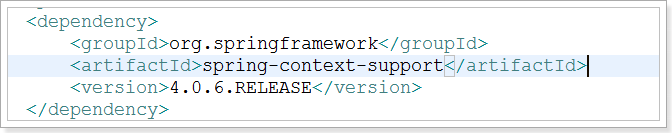
example.run();

}

}

## 通过Spring使用Quartz

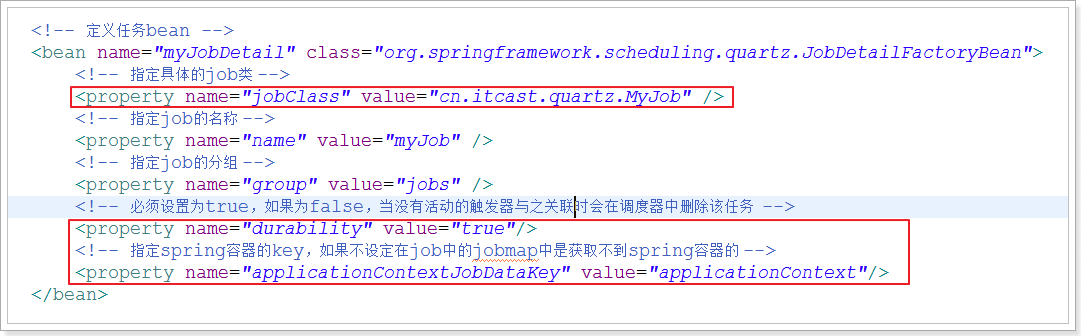
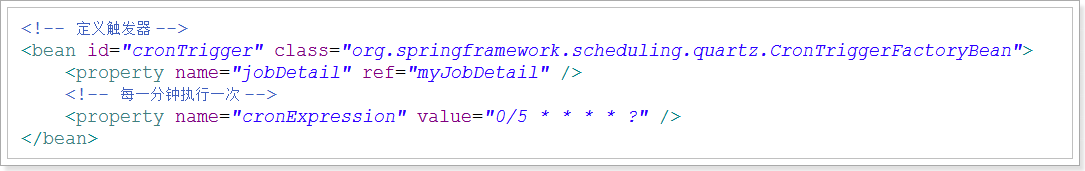
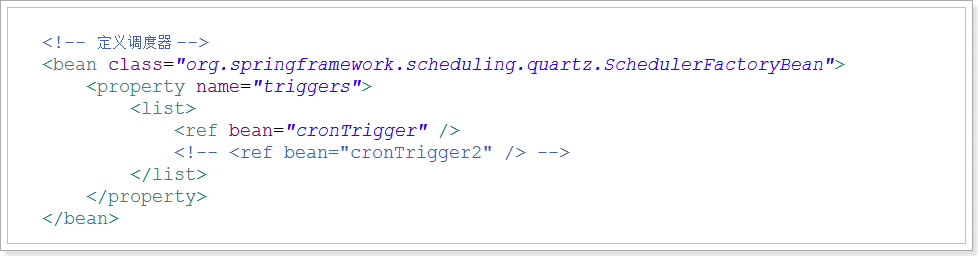
### 导入依赖



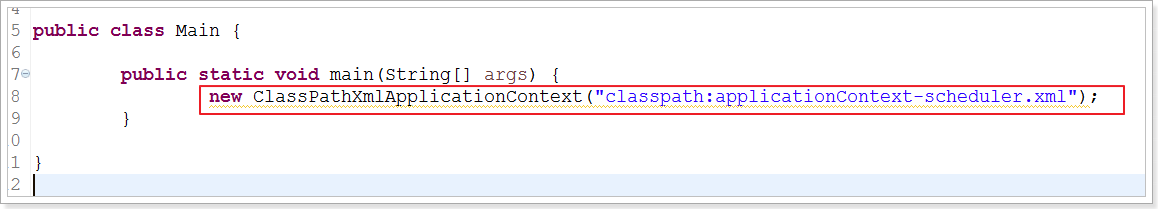
### 编写Job



### 编写配置文件文

1. 定义job  
   
2. 定义触发  
   
3. 定义调度器，并且注册触发  
   

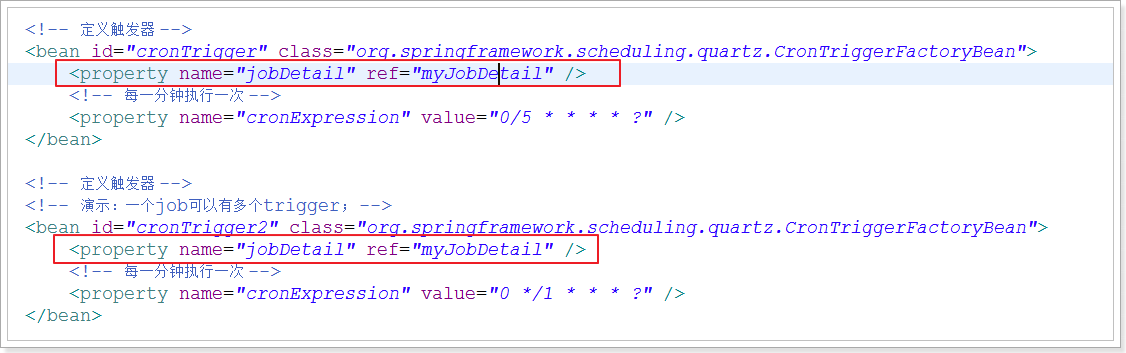
### 启动Spring容器（启动调度器）



## 触发和job之间关系

一个触发能都多个job吗？ 不能！

一个job能有多个触发吗？ 是的！



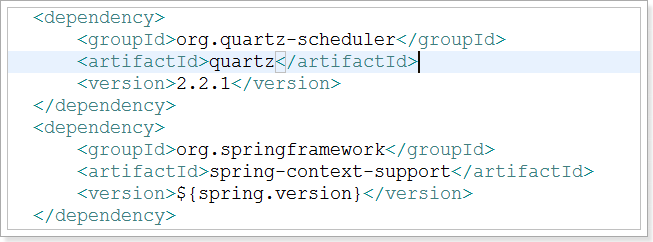
# 关闭超时2天未付款的订单

## 分析

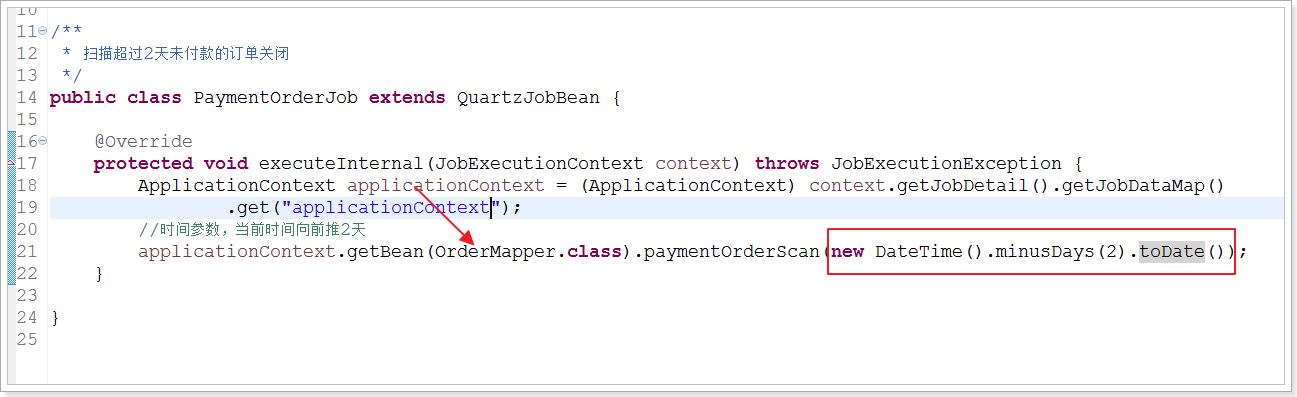
1. 在订单系统中导入相关的依赖
2. 定义job
   1. 扫描订单表，修改订单的状态为关闭状态
   2. 扫描条件：创建时间 <= 当前之间 – 2天 并且，付款方式为在线支付的订单
3. 定义触发
   1. 理论上需要实时触发（性能问题）
   2. 1分钟触发一次

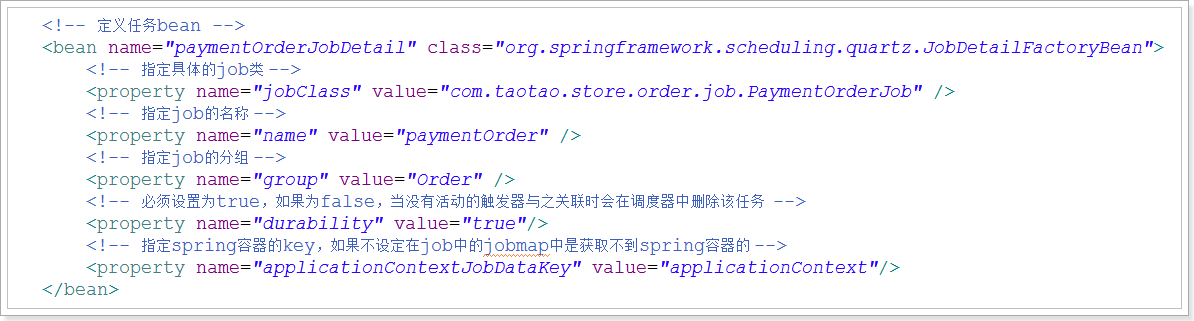
## 实现

### 导入依赖

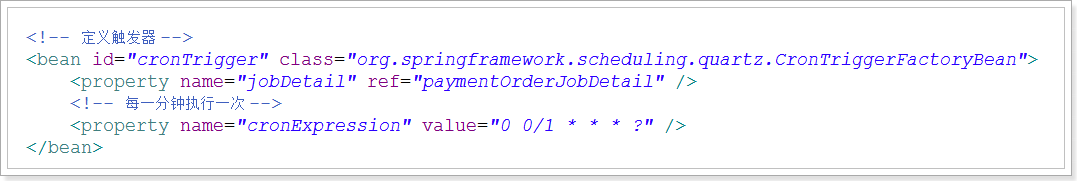


### 定义job

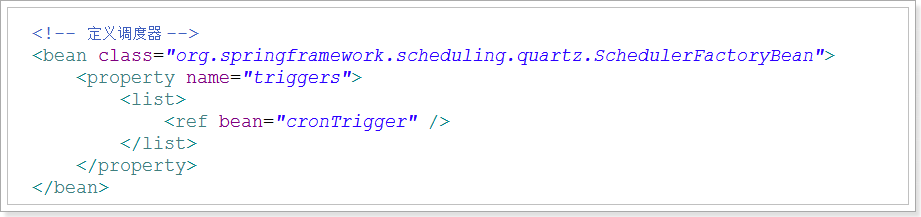




### 定义触发



### 定义调度器



### OrderMapper中的实现

