



Module 15: Spring Scheduling

CS544: Enterprise Architecture



Wholeness

- Many enterprise applications need to periodically run certain processes.
- For example:
 - Daily / weekly / monthly reports
 - Daily / weekly backups
 - Generate monthly payroll
- Many processes have to wait for their appropriate time. Rest and activity are the steps or progress.



Job scheduling Technologies

- JDK's `java.util.Timer` & `java.util.concurrent.Scheduler`
 - Basic scheduling support
 - Execute at a given time
 - Execute at some fixed frequency
- Quartz scheduling
 - Well established open source framework
 - Powerful job scheduling engine
 - Cron-based scheduling
- Spring's support for Job Scheduling



Scheduling

SCHEDULING TERMINOLOGY



Scheduling basics

- Job / Task
 - Unit of work that needs to execute at a specific time or interval
- Trigger
 - The condition (specific time or interval) that causes a job to run
- Schedule
 - A collection of triggers



CRON Expressions

- CRON Expressions will match one or more instances in time
- String with 6 or 7 space separated sub-expressions with the following meaning:

Expression that matches
the time for our lunch break

seconds	minutes	hours	dayOfMonth	month	dayOfWeek	year(optional)
0	30	12	*	*	MON-FRI	



CRON Special Characters

- Asterisk (*) The asterisk indicates that the cron expression matches for all values of the field. E.g., using an asterisk in the 4th field (month) indicates every month.
- Slash (/) Slashes describe increments of ranges. For example 3-59/15 in the 1st field (minutes) indicate the third minute of the hour and every 15 minutes thereafter. The form "*/..." is equivalent to the form "first-last/...", that is, an increment over the largest possible range of the field.
- Comma (,) Commas are used to separate items of a list. For example, using "MON,WED,FRI" in the 5th field (day of week) means Mondays, Wednesdays and Fridays.
- Hyphen (-) Hyphens define ranges. For example, 2000-2010 indicates every year between 2000 and 2010 AD, inclusive.
- Hash (#) is allowed for the day-of-week field, and must be followed by a number between one and five. It allows you to specify constructs such as "the second Friday" of a given month.
- Question mark (?) In some implementations, used instead of '*' for leaving either day-of-month or day-of-week blank. Other cron implementations substitute "?" with the start-up time of the cron daemon, so that ? * * * * would be updated to 25 8 * * * * if cron started-up on 8:25am, and would run at time every day until restarted again.

http://en.wikipedia.org/wiki/CRON_expression#CRON_expression



Examples

■ Examples:

■ "0 0 12 ? * WED"

■ every Wednesday at 12:00 pm

■ "0 0/5 * * * ?"

■ every 5 minutes

?=no specific value (only for
dayOfMonth and dayOfWeek)

■ "10 0/5 * * * ?"

■ every 5 minutes, at 10 seconds after the minute (i.e. 10:00:10 am, 10:05:10 am, etc.).

■ "0 30 10-13 ? * WED,FRI"

■ 10:30, 11:30, 12:30, and 13:30, on every Wednesday and Friday.

■ "0 0/30 8-9 5,20 * ?"

■ every half hour between the hours of 8 am and 10 am on the 5th and 20th of every month.



Main Point 1

- A schedule has many jobs, each of which has its own trigger. Triggers are commonly based on CRON expressions; which are industry standard patterns for expressing time intervals.
- Seek the highest first, once you've learned how to read and write CRON expressions you can use them in many scheduling environments



Scheduling

SCHEDULING WITHOUT SPRING



JDK Timer example

```
public class HelloWorldTask extends TimerTask{
```

Not a POJO

```
    public void run() {
        Date date = Calendar.getInstance().getTime();
        DateFormat timeFormatter = DateFormat.getTimeInstance(DateFormat.DEFAULT);
        String currenttime = timeFormatter.format(date);

        System.out.println("This task runs at "+currenttime);
    }
}
```

```
import java.util.Timer;
```

```
public class Application {
```

```
    public static void main(String[] args) {
        Timer timer = new Timer();
        timer.scheduleAtFixedRate(new HelloWorldTask(), 5000, 5000);
    }
}
```

The job to run

Start the job
after 5 seconds

Run the job every
5 seconds

```
This task runs at 10:45:52
This task runs at 10:45:57
This task runs at 10:46:02
This task runs at 10:46:07
```



Java 5

- Java 5 provides an updated way to schedule with the **ScheduledThreadPoolExecutor**
- No longer requires subclassing TimerTask

Better, but still not a POJO

```
public class HelloWorldTask implements Runnable {  
  
    @Override  
    public void run() {  
        Date date = Calendar.getInstance().getTime();  
        DateFormat timeFormatter = DateFormat.getTimeInstance(DateFormat.DEFAULT);  
        String currenttime = timeFormatter.format(date);  
        System.out.println("This task runs at " + currenttime);  
    }  
}
```



Java 5 Main

```
public class App {  
    public static void main(String[] args) throws InterruptedException {  
        ScheduledExecutorService tpool = Executors.newScheduledThreadPool(5);  
        HelloWorldTask task = new HelloWorldTask();  
  
        // executes 5 seconds after being called (does not repeat)  
        tpool.schedule(task, 5, TimeUnit.SECONDS);  
  
        // executes 1 second after being called  
        // repeats exactly 5 seconds after first call  
        tpool.scheduleAtFixedRate(task, 1, 5, TimeUnit.SECONDS);  
  
        // executes 1 second after being called  
        // repeats 5 after the last one finished  
        tpool.scheduleWithFixedDelay(task, 1, 5, TimeUnit.SECONDS);  
  
        // close the schedule thread pool (stops all scheduled tasks)  
        Thread.sleep(60000);  
        tpool.shutdown();  
    }  
}
```



Quartz simple scheduling example

```
public class HelloWorldJob implements Job{

    public void execute(JobExecutionContext arg0) throws JobExecutionException {
        Date date = Calendar.getInstance().getTime();
        DateFormat timeFormatter = DateFormat.getTimeInstance(DateFormat.DEFAULT);
        String currenttime = timeFormatter.format(date);

        System.out.println("This task runs at "+currenttime);
    }
}
```

```
public class Application {
    public static void main(String[] args) throws SchedulerException {
        Scheduler scheduler =new StdSchedulerFactory().getScheduler();
        scheduler.start();

        JobDetail jobDetail=new
            JobDetail("HelloWorldJob",scheduler.DEFAULT_GROUP,HelloWorldJob.class);
        Trigger simpleTrigger=new SimpleTrigger("mytrigger",scheduler.DEFAULT_GROUP,new
            Date(),null,SimpleTrigger.REPEAT_INDEFINITELY,5000);
        scheduler.scheduleJob(jobDetail, simpleTrigger);
    }
}
```

The job

The
schedule

The trigger: every
5 seconds

```
This task runs at 12:00:15
This task runs at 12:00:20
This task runs at 12:00:25
```



Quartz cron scheduling example

```
public class HelloWorldJob implements Job{

    public void execute(JobExecutionContext arg0) throws JobExecutionException {
        Date date = Calendar.getInstance().getTime();
        DateFormat timeFormatter = DateFormat.getTimeInstance(DateFormat.DEFAULT);
        String currenttime = timeFormatter.format(date);

        System.out.println("This task runs at "+currenttime);
    }
}
```

```
public class CronApplication {
    public static void main(String[] args) throws SchedulerException, ParseException {
        Scheduler scheduler =new StdSchedulerFactory().getScheduler();
        scheduler.start();

        JobDetail jobDetail=new
            JobDetail("HelloWorldJob",scheduler.DEFAULT_GROUP,HelloWorldJob.class);

        String cronExpression="0/5 * * * * ?";
        Trigger crontrigger=new
            CronTrigger("crontrigger",scheduler.DEFAULT_GROUP,cronExpression);
        scheduler.scheduleJob(jobDetail, crontrigger);
    }
}
```

The trigger is expressed with a cron expression

```
This task runs at 12:04:25
This task runs at 12:04:30
This task runs at 12:04:35
```



Main Point 2

- The Java SE JDK includes a couple of scheduling options; in addition to which Quartz offers a more powerful open source solution.
- Unfortunately our beans must always extend a super class or implement an interface (not a POJO), to ensure your class has the required method name.
- We will see how we can solve this problem by introducing Spring. Similar to the principle of the second element, the problem is not able to be solved on the level of the problem.



Scheduling

SCHEDULING WITH SPRING



Spring Annotations

- FixedRate, FixedDelay, or Cron
- Annotate any method on any POJO
 - That does not take any params and returns void

```
public class MyClass {  
  
    @Scheduled(fixedRate = 5000, initialDelay = 1000)  
    public void fixedRateMethod() {  
        System.out.println("Fixed rate");  
    }  
  
    @Scheduled(fixedDelay = 5000, initialDelay = 2000)  
    public void fixedDelayMethod() {  
        System.out.println("Fixed delay");  
    }  
  
    @Scheduled(cron="0/5 * * * * *")  
    public void cronMethod() {  
        System.out.println("Cron expression");  
    }  
}
```



Optional Parameter



Spring Config

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:task="http://www.springframework.org/schema/task"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
    http://www.springframework.org/schema/task
    http://www.springframework.org/schema/task/spring-task-4.0.xsd">

    <bean id="myClass" class="cs544.scheduling.spring.MyClass" />

    <task:annotation-driven scheduler="myScheduler" />
    <task:scheduler id="myScheduler" pool-size="10" />
</beans>
```

Task namespace

Look for @Scheduled

Scheduling Thread Pool



Spring scheduling with Quartz

```
public class PojoApplication {  
    public static void main(String[] args) {  
        ApplicationContext context = new ClassPathXmlApplicationContext("springconfigpojo.xml");  
    }  
}
```

```
public class HelloWorldPojo {  
    private String message;  
  
    public void setMessage(String message) {  
        this.message = message;  
    }  
  
    public void welcome() {  
        Date date = Calendar.getInstance().getTime();  
        DateFormat timeFormatter = DateFormat.getTimeInstance(DateFormat.DEFAULT);  
        String currenttime = timeFormatter.format(date);  
  
        System.out.println("This task runs at "+currenttime+" message= "+message);  
    }  
}
```

message is injected

```
This task runs at 12:07:50 message= Hello World  
This task runs at 12:07:55 message= Hello World  
This task runs at 12:08:00 message= Hello World
```



Spring scheduling with Quartz

```
<bean name="welcomeBean" class="timer.HelloWorldPojo" >  
  <property name="message" value="Hello World" />  
</bean>
```

The Pojo

```
<bean name="job" class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">  
  <property name="targetObject" ref="welcomeBean" />  
  <property name="targetMethod" value="welcome" />  
</bean>
```

The job: call the welcome()
method of the
timer.HelloWorldPojo class

```
<bean id="trigger" class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">  
  <property name="jobDetail" ref="job" />  
  <property name="cronExpression" value="0/5 * * * * ?" />  
</bean>
```

The trigger: start after 0 seconds
and repeat every 5 seconds

```
<bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">  
  <property name="triggers">  
    <list>  
      <ref bean="trigger" />  
    </list>  
  </property>  
</bean>
```

The schedule contains only 1
trigger



Spring XML JDK Timer (1/2)

```
public class Application {  
    public static void main(String[] args) {  
        ApplicationContext context = new ClassPathXmlApplicationContext("springconfig.xml");  
    }  
}
```

```
public class HelloWorldPojo {  
    private String message;  
  
    public void setMessage(String message) {  
        this.message = message;  
    }  
  
    public void welcome() {  
        Date date = Calendar.getInstance().getTime();  
        DateFormat timeFormatter = DateFormat.getTimeInstance(DateFormat.DEFAULT);  
        String currenttime = timeFormatter.format(date);  
  
        System.out.println("This task runs at "+currenttime+" message= "+message);  
    }  
}
```

Message is injected

```
This task runs at 11:09:18 message= Hello World  
This task runs at 11:09:23 message= Hello World  
This task runs at 11:09:28 message= Hello World
```



Spring XML JDK timer (2/2)

```
<bean name="welcomeBean" class="timer.HelloWorldPojo">
  <property name="message" value="Hello World" />
</bean>
```

```
<bean id="theJob"
      class="org.springframework.scheduling.timer.MethodInvokingTimerTaskFactoryBean">
  <property name="targetObject" ref="welcomeBean" />
  <property name="targetMethod" value="welcome" />
</bean>
```

The job: call the welcome() method of the timer.HelloWorldPojo class

```
<bean id="timerTrigger" class="org.springframework.scheduling.timer.ScheduledTimerTask">
  <property name="delay" value="5000" />
  <property name="period" value="5000" />
  <property name="timerTask" ref="theJob" />
</bean>
```

The trigger: start after 5 seconds and repeat every 5 seconds

```
<bean id="timerFactory" class="org.springframework.scheduling.timer.TimerFactoryBean">
  <property name="scheduledTimerTasks">
    <list>
      <ref local="timerTrigger" />
    </list>
  </property>
</bean>
```

The schedule contains only 1 trigger



Main Point 3

- Spring provides `@Scheduled` annotation support that lets us schedule POJO methods and adds CRON support. If desired it can also integrate with Quartz.
- Once again, spring lets us do less and accomplish more.



Scheduling

ASYNC METHOD CALLS



Asynchronous Methods

- Calling an asynchronous method will always return right away (not wait for a return)
- The method is started in a separate thread so that the application does not have to wait

```
package cs544.scheduling.async;
import org.springframework.scheduling.annotation.Async;

public class SomeClass {

    @Async
    public void longRunningMethod() {
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) { e.printStackTrace(); }
        System.out.println("Finished LongRunningMethod");
    }
}
```

Any method that takes no params, and returns void



Running an @Async Method

```
public class App {  
    public static void main(String[] args) {  
        ApplicationContext context = new ClassPathXmlApplicationContext("springasync.xml");  
        SomeClass sc = context.getBean("someClass", SomeClass.class);  
        sc.longRunningMethod();  
        System.out.println("Main method continues on!");  
    }  
}
```

Main method continues on!
Finished longRunningMethod

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xmlns:task="http://www.springframework.org/schema/task"  
    xsi:schemaLocation="  
        http://www.springframework.org/schema/beans  
        http://www.springframework.org/schema/beans/spring-beans-4.0.xsd  
        http://www.springframework.org/schema/task  
        http://www.springframework.org/schema/task/spring-task-4.0.xsd">  
  
    <bean id="someClass" class="cs544.scheduling.async.SomeClass" />  
  
    <task:annotation-driven executor="myExecutor" />  
    <task:executor id="myExecutor" pool-size="5" />  
  
</beans>
```

Look for @Async

Executor Thread Pool



Main Point 4

- Asynchronous method calls allows our application to invoke longer running processes at any time, without having to sit and wait.
- It does this by providing a thread pool, where a separate thread provides for the needs of the called method.
- Harmony exists in diversity, our application achieves more by having multiple threads.



Active Learning

- Write a Cron Trigger that has a delay of 30 seconds, and then fires every 10 seconds, Friday afternoon at 2:45pm
- What if we want it to start at 2:45pm and continue firing every 10 seconds until 2:50pm?



Summary

- Job scheduling
 - JDK timer-based scheduling
 - Quartz scheduling
 - Cron-based scheduling
- Spring-based scheduling
 - Only the job is coded
 - Scheduler, triggers and job details are configured
 - Any method of any POJO can be scheduled
 - All scheduling details are in one place
 - Easier to maintain