Zeitsteuerung von Abläufen in Java auf Basis von Terracotta Quartz

Ralph Löwe
Competence Center Wirtschaftsinformatik
Hochschule München

Schlüsselworte:

Ablaufsteuerung, Java, Java EE, Opensymphony Quartz, Terracotta Quartz, cron, zeitbasierte Steuerung, Jobsteuerung

Einleitung und Motivation

Abläufe (Jobs) als Folge von Aktionen bzw. Aktivitäten werden vor allem im betrieblichen Umfeld häufig ereignisgesteuert oder zyklisch angestoßen. Ereignisse wie z.B. das Erreichen eines bestimmten Zeitpunkts (z.B. genau am Sonntag, um 10:00 Uhr) oder das Eintreffen von Daten können einen vordefinierten Ablauf auslösen. Typische Beispielanwendungen, die eine Ablaufsteuerung benötigen sind die tägliche Sicherung, das Archivieren und Löschen von Bewegungsdaten und das tägliche Exportieren von operativen Systemen in ein Data Warehouse (Berichtswesen).

Zur Ablaufautomatisierung bzw. Ablaufsteuerung (auch Jobsteuerung genannt) sind spezielle Werkzeuge sinnvoll, die u.a. eine Zeitsteuerung unterstützen. Typische Werkzeuge zur Ablaufsteuerung sind cron, Torque oder UC4. Im Java-Umfeld gibt es allerdings recht wenig vordefinierte Basisfunktionalität, weshalb Eigenlösungen häufig vorzufinden sind. Java EE (Enterprise Edition) versteht sich zwar als Industriestandard für die Entwicklung von Anwendungssystemen auch im betrieblichen Umfeld, jedoch bietet der Defacto-Standard nur rudimentäre Möglichkeiten zur Unterstützung einer Ablaufsteuerung.

Diese Lücke kann man durch die Open-Source-Software *Terracotta Quartz* (kurz: *Quartz*) schließen, die oft auch in Java-EE-Servern verwendet wird. Die Lösung bietet vielfältige Schnittstellen, ermöglicht eine komfortable Zeitsteuerung und ist komplett in Java implementiert. Sie erweist sich auch als praktikable Alternative für kommerzielle Lösungen.

Im vorliegenden Beitrag wird die Anwendung von *Terracotta Quartz* zur Implementierung einer Ablaufsteuerung vorgestellt. Insbesondere wird auf die Konfiguration, auf Clustering-Ansätze, auf das Planen von Aufträgen sowie auf die Schnittstelle zu Transaktionsmanagern für die Realisierung kritischer Anwendungen eingegangen.

Ablaufsteuerung mit klassischen Java-Hilfsmitteln

Die Java Standard Edition (Java SE) umfasst eine Menge von Bibliotheken für die Entwicklung von Java-Anwendungen, allerdings keine vordefinierte API zur Unterstützung einer Ablaufsteueung. Eine Implementierung eines zeitgesteuerten Ablaufs müsste also auf Basis von Java SE individuell erfolgen.

In der Java Enterprise Edition (Java EE) gibt es seit EJB Version 2.1 die Möglichkeit, einen eigenen `TimerService` zu nutzen. Jedoch ist dieser sehr einfach gehalten und eine Trennung von

aufrufender und ausführender Logik ist nicht vorgesehen. Die Funktionalität wurde für die aktuelle Spezifikation EJB Version 3.0 komplett übernommen und hat sich nicht verbessert. Fehlersituationen werden hier nur rudimentär unterstützt. Auf die Steuerung von Abläufen in Clustern wird in der Spezifikation nicht eingegangen. Die Implementierung dieser Aspekte ist damit komplett dem Produkthersteller überlassen. Auch die Möglichkeit, zu bestimmten Zeiten alle Jobs zu sperren, wird nicht geboten und so muss das Sperren von Jobs in Wartungsfenstern individuell implementiert werden.

In Listing 1 wird beispielhaft eine zustandslose EJBean definiert, welche in der Methode `doIt(String message)` einen `Timer` erzeugt. Über die Annotation `@Timeout` wird eine Methode in einer EJBean markiert, die ausgeführt werden soll, wenn der Timer abläuft. Über die Methode `createTimer` kann ein serialisierbares Java-Objekt als Parameter übergeben werden, das bei Aufruf der Timeout-Methode über die Methode `getInfo()` gelesen werden kann.

```
// Die Annotation @Stateless ist notwendig. Klassen- und
// Schnittstellenname sind frei wählbar.
@Stateless
public class StatelessTimedBean30 implements StatelessTimedRemote30
{
    @Resource
    private SessionContext context;

    public String doIt(String message) {
        TimerService service = context.getTimerService();
        service.createTimer(1000, message);
        return message;
    }

    @Timeout
    public void timeout(Timer timer) {
        String message = (String)timer.getInfo();
        System.out.println("Timer executed: " + message);
    }
}
```

Listing 1: Beispiel einer zeitgesteuerten statuslose Enterprise Java Bean, Version 3.0.

Der `TimerService` wurde in der EJB-3.1-Spezifikation wesentlich verbessert. Über die neuen Mechanismen können Methoden von EJBeans mit den Annotationen `@Schedule`, `@Schedules` und `@Timeout` mit einem Zeitsteuerungsverhalten versehen werden. Auch die programmatische Erzeugung eines einfachen Ereignisses ist darüber möglich. Jedoch ist die Version 3.1 derzeit noch nicht stark verbreitet.

Ablaufsteuerung mit Terracotta Quartz

Wer sich bei der Entwicklung an Java orientiert, kann das Werkzeug *Terracotta Quartz* (im Folgenden abkürzend als Quartz bezeichnet) zur Ablaufsteuerung einsetzen. Quartz ist ein anwendungsübergreifendes Framework, mit der zentral auch Wartungsfenster unterstützt werden, indem anwendungsspezifische `Listener-Klassen` programmiert werden.

Ursprünglich wurde das quelloffene Framework Quartz von dem Unternehmen *Opensymphony* entwickelt. Ende 2009 wurde Quartz an die Firma *Terracotta* übergeben, welche aktuell die Weiterentwicklung vorantreibt. *Terracotta* bietet kommerzielle Erweiterungen für Quartz wie z. B. eine persistente Ablaufdatenbank. Im Folgenden beschränken wir uns aber auf die Open-Source-Bestandteile von Quartz. Quartz ist in vielen Java-EE-Application-Servern als Erweiterung eingebunden. So wird der `TimerService` in EJB-Containern oft über Quartz abgewickelt.

Anwendungsobjekte in Quartz

Quartz unterstützt eine Trennung zwischen den Ereignissen und den durch Ereignisse anzustoßenden Abläufen. In Quartz wird jedes Ereignis über ein `Trigger-Objekt` beschrieben und ein Ablauf über ein `Job-Objekt`. Zwischen beiden Objekttypen besteht eine Beziehung. Der in Quartz integrierte *Scheduler* verwendet die definierten Trigger und Jobs, um Abläufe ereignisgesteuert zu initiieren. Die wichtigsten Basisklassen und Schnittstellen, die bei Quartz verwendet werden, sind in Abbildung 1 dargestellt.

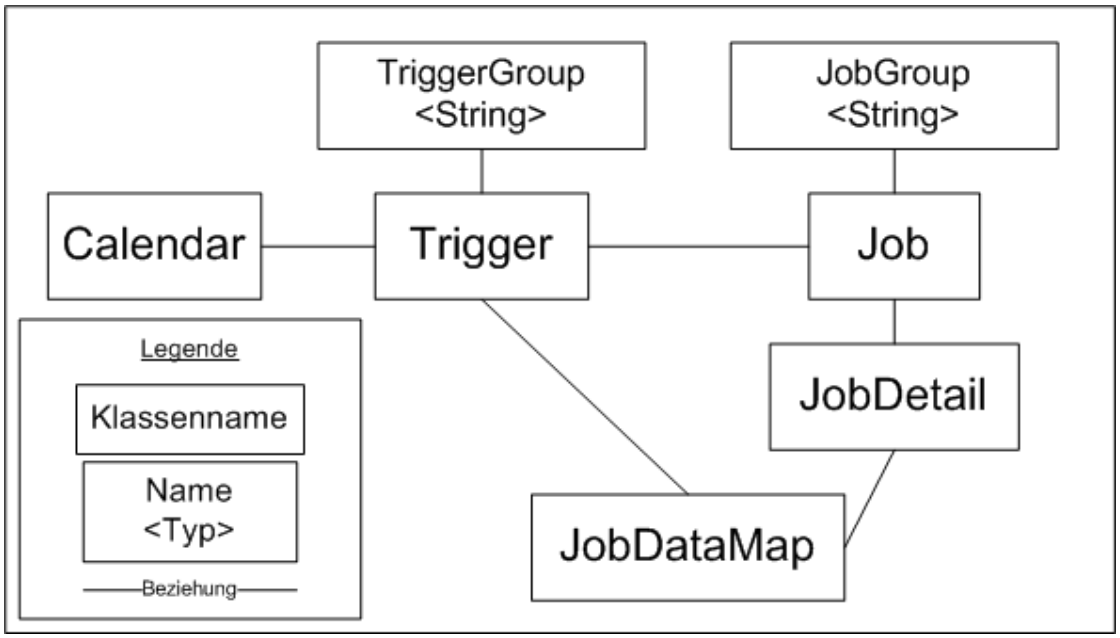

Abbildung 1: Basisklassen und Schnittstellen bei Terracotta Quartz.

Ein Job muss eine der vorgegebenen Schnittstelle `Job`, `StatefulJob` oder `InteruptableJob` implementieren. Je nach Variante verhält sich der Ablauf dann unterschiedlich. Eine Implementierung von `StatefulJob` kann ihren Status – in Form einer `JobDataMap` – über mehrere Aufrufe hinweg erhalten. Ein Job, der die Schnittstelle `InteruptableJob` implementiert, kann während der Ausführung unterbrochen und später fortgesetzt werden. Eine Gruppierung von Jobs ist über die Angabe eines Gruppennamens möglich.

Die Parameter für Abläufe und Ereignisse werden über ein `JobDataMap`-Objekt übergeben, wobei Java-Basistypen und beliebige Java-Objekte zulässig sind. Es wird davon abgeraten, Objekte als Parameter zu verwenden, da es zu Problemen bei Laden der Klassen führen kann. Die Zuordnung von Abläufen zu ihren Parametern erfolgt über ein `JobDetail`-Objekt. Hier ist auch konfiguriert, ob im

Fehlerfall eine Wiederherstellung unterstützt werden soll oder nicht. Dem `JobDetail`-Objekt wird die zugehörige `Job`-Klasse übergeben. Ein Erzeugen des Objektes erfolgt erst beim Eintreten des zugeordneten Ereignisses.

Für die Beschreibung von Ereignissen existiert eine abstrakte Klasse mit der Bezeichnung `Trigger`. Es gibt ein paar Implementierungen und im Gegensatz zum Ablauf muss normalerweise keine eigene Klasse programmiert werden. Die vordefinierten Ereignisklassen mit den Bezeichnungen `SimpleTrigger` und `CronTrigger` decken bereits einen sehr großen Bereich von Ereignissen ab. Kennern von *Cron* wird auffallen, dass in Quartz auch eine sekundengenaue und jahresbezogene Steuerung möglich ist. Zusätzlich vereinfacht die Klasse `TriggerUtils` – ähnlich wie das Fabrik-Entwurfsmuster – die Instanziierung von Ereignisobjekten. Die Parameter können über eine Instanz der Klasse `JobDataMap` erzeugt werden. Beim Aufruf des Ablaufs werden dann seine Parameter mit denen des Ereignisses überschrieben und stehen über die Methode `getMergedJobDataMap()` zur Verfügung. Wie Abläufe können auch Ereignisse zu Gruppen zusammengefasst werden.

Jedem Ereignis kann optional über eine eindeutige Zeichenkette ein Kalender zugeordnet werden. Ein Objekt, das die Schnittstelle `Calendar` implementiert, beschreibt, wann das Ausführen von Abläufen erlaubt ist. Somit können zentral mehrere Kalender verwaltet werden, die für Gruppen von Ereignissen gelten. Hier kann man beispielsweise Urlaubs- und Wartungszeiten definierten und in der Ablaufsteuerung zentralisieren.

Anwendungsbeispiel mit Quartz

Nachfolgend soll ein Beispiel eines einfachen Ablaufs, der zyklisch über ein tägliches Ereignis um 22 Uhr angestoßen wird, skizziert werden. Als Einschränkung wird festgelegt, dass der Beispielablauf nicht an Sonn- und Feiertagen ausgeführt werden darf. Zur Vereinfachung werden im Beispiel die Ausnahmebehandlung, Paketnamen, Importanweisungen und Klassen- sowie Methodenrahmen weggelassen. Muss eine Codezeile aus Platzgründen in der folgenden Zeile fortgesetzt werden, wird dies mit dem Symbol „→“ angezeigt.

Bevor man Ereignisse und Abläufe in Quartz registrieren kann, muss man einen Scheduler-Thread erzeugen. Das Starten und Herunterfahren eines Scheduler-Threads kann wie folgt programmiert werden (siehe Listung 2):

```
Scheduler sched = StdSchedulerFactory.getDefaultScheduler();

sched.start();

// Hier können Ihre Ereignisse und Abläufe registriert werden.

sched.shutdown();
```

Listing 2: Starten eines Schedulers in Quartz.

Die Konfiguration erfolgt im einfachsten Fall über eine Datei mit dem vordefinierten Namen `quartz.properties`, die als Ressource im Klassenpfad verfügbar sein muss. Nachfolgend wird eine mögliche Konfiguration des Scheduler gezeigt (Listing 3):

```
// Der Name der Scheduler-Instanz.
org.quartz.scheduler.instanceName = MyFirstScheduler
// Der Thread des Schedulers ist ein User-Thread.
```

```
org.quartz.scheduler.makeSchedulerThreadDaemon = false

// Einfaches Thread-Pooling mit 5 Threads und Thread-Priorität 5.
org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
org.quartz.threadPool.threadCount = 5
org.quartz.threadPool.threadPriority = 5

// Die Jobs werden im flüchtigen Speicher gehalten.
// Alternativ kann man hier eine Datenbank konfigurieren.
org.quartz.jobStore.class = org.quartz.simpl.RAMJobStore

// Plugin für das saubere Herunterfahren des Scheduler.
org.quartz.plugin.shutdownhook.class =
  →  org.quartz.plugins.management.ShutdownHookPlugin

org.quartz.plugin.shutdownhook.cleanShutdown = true
```

Listing 3: Konfiguration eines Schedulers in Quartz

Die Konfigurationsoption `org.quartz.jobStore.class` definiert dabei, wie Abläufe und Ereignisse dauerhaft gespeichert werden. Dadurch sind die Aufträge nach einem Neustart des Scheduler wieder verfügbar. Folgende Optionen stehen zur Verfügung:

1. `org.quartz.simpl.RAMJobStore` – Die Abläufe und Ereignisse werden transient im flüchtigen Speicher gehalten; ein persistentes Abspeichern findet nicht statt. Für Demonstrationszwecke oder für Anwendungen, welche ihre Abläufe nicht erhalten müssen, ist dieser Modus ausreichend.
2. `org.quartz.impl.jdbcjobstore.JobStoreCMT` – Bei dieser Option wird die Transaktionsbehandlung (Festlegung der Transaktionsgrenzen) implizit über den Container ausgeführt. Üblicherweise wird diese Konfiguration verwendet, falls Quartz in einen Application-Server eingebettet ist.
3. `org.quartz.impl.jdbcjobstore.JobStoreTX` – Damit werden die Transaktionsgrenzen direkt von Quartz gesetzt. Sie eignet sich für die Java-SE-Umgebung.

Exkurs: Ein Clustering von mehreren Quartz-Instanzen ist möglich. Dabei müssen die Konfigurationen grundlegend gleich sein und jede Instanz muss ihren eigenen `JobStore` besitzen. Wichtig ist dabei, dass die Zeit auf allen Rechnern synchronisiert ist. So ist zwischen den Uhren der beteiligten Server höchstens Differenz von einer Sekunde zulässig. Wenn eine Serverinstanz nicht verfügbar ist, kann Quartz eine Lastverteilung und Übernahme von fehlgeschlagenen Abläufen durchführen, ohne die Transaktionssicherheit zu verletzen. Außerhalb eines Java-EE-Containers kann man Quartz über Remote Method Invocation (RMI) für andere Anwendungen verfügbar machen.

Ist ein Quartz-Scheduler-Thread gestartet, können Anwendungen ihre Abläufe und Ereignisse hinzufügen. Hier das Beispiel für die Job-Implementierung (Listing 4):

```
public class BackupJob implements Job {
    // Der JobExecutionContext beinhaltet JobDataMap und JobDetail.
    public void execute(JobExecutionContext context) {
        Calendar fireTime = Calendar.getInstance();
        fireTime.setTime(context.getFireTime());
        System.out.println("executing backup: " +
```

```
            fireTime.get(Calendar.SECOND));
        // Hier kann die Logik der Sicherung implementiert sein.
    }
}
```

Listing 4: Job-Implementierung innerhalb von Quartz.

Im vereinfachten Beispiel wird nur die Startzeit des Ablaufes ausgegeben. Das `JobExecutionContext`-Object beinhaltet die Steuerinformation des aktuellen Ereignisses und des Ablaufs.

Listing 5 zeigt einen passenden Kalender, welcher das Wochenende sowie die gesetzlichen Feiertage im Oktober für Deutschland abdeckt.

```
public class GermanyHoliday extends BaseCalendar
        implements org.quartz.Calendar {

    public boolean isTimeIncluded(long l) {
        Calendar cal = Calendar.getInstance();
        cal.setTimeInMillis(l);

        // Heute ist Sonntag
        if (cal.get(Calendar.DAY_OF_WEEK) == Calendar.SUNDAY)
            return false;
        // Heute ist Samstag.
        else if (cal.get(Calendar.DAY_OF_WEEK) == Calendar.SATURDAY)
            return false;
        // Tag der Deutschen Einheit (jedes Jahr am 3. Oktober).
        else if (cal.get(Calendar.DAY_OF_MONTH) == 3 &&
                cal.get(Calendar.MONTH) == Calendar.OCTOBER)
            return false;
        else
            return true;
    }
}
```

Listing 5: Kalender für den Monat Oktober, welche Sams-, Sonn- und Feiertage ausschließt.

Die Zeit wird im Beispiel über die Methode `isTimeIncluded(long l)` als Zeitstempel übergeben. Die Schnittstelle `Calendar` benötigt mehrere Methoden. Damit nur die notwendigen implementiert werden müssen, kann man von der abstrakten Klasse `BaseCalendar` ableiten. Dies ist ein hilfreiches Muster bei Schnittstellen mit vielen Methoden.

Eine mögliche Planung von Ablauf und Ereignis im Quartz-Scheduler ist in Listing 6 dargestellt.

```
GermanyHoliday holiday = new GermanyHoliday();
sched.addCalendar("GermanyHoliday", holiday, true, true);

JobDetail detail = new JobDetail("BackupJob",null,BackupJob.class);

Trigger trigger = TriggerUtils.makeDailyTrigger(22, 0);
trigger.setCalendarName("GermanyHoliday");
```

```
sched.scheduleJob(detail, trigger);
```

Listing 6: Planung von Ablauf, Kalender und des Ereignis mit Quartz.

Ein Kalender wird angelegt und dem Scheduler mit einer Zeichenkette übergeben. Für den Ablauf wird eine `JobDetail`–Instanz und als Ereignis ein `Trigger`-Objekt über die `TriggerUtils`-Klasse erzeugt. Der Kalender für das Ereignis wird über eine Zeichenkette referenziert. Abschließend werden Ereignis und Ablauf im Scheduler geplant.

Zusammenfassung und kritische Betrachtung

In diesem Artikel wurden die Möglichkeiten der Ablaufsteuerung mit dem Werkzeug Terracotta Quartz näher betrachtet. Es kann festgehalten werden, dass Terracotta Quartz deutlich bessere Mechanismen für diese Aufgabe bereitstellt, als dies bisher im Java-Standard vorgesehen ist.

Terracotta Quartz hat für die Jobsteuerung deutlich mehr Möglichkeiten als EJB Version 3.0. Parameter lassen sich einfacher an Jobs übergeben und eine bessere Trennung von Ereignissen, die Abläufe anstoßen und den Abläufen selbst ist möglich. Anwendungsübergreifende Kalender können ebenfalls in mehreren Jobs berücksichtigt und Jobs können auch für definierte Zeiten, wie etwa Wartungsfenstern, kontrolliert deaktiviert werden.

Einige Probleme sind während der Erprobung von Quartz aufgefallen:

- So kann die Verknüpfung von Objekten mittels Zeichenketten fehleranfällig sein. Bei einer einzelstehenden Anwendung kann dies mit der strikten Definition der Zeichenketten als Konstanten vermieden werden. Bei mehreren Anwendungen kann es jedoch zu Konflikten kommen. Es bietet sich deshalb an, Zeichenketten über einen Mechanismus zu vergeben, welcher die Eindeutigkeit sicherstellt.
- Die Fehlerdiagnose einer zeitgesteuerten Anwendung kann sich als schwieriger erweisen als bei einer manuell gesteuerten Anwendung. So müssen die aktuelle Umgebung und insbesondere zeitgleiche Abläufe beachtet werden. Eine Verbesserung kann durch Einsatz mehrerer `Listener`-Objekte erreicht werden. Die Erstellung und die Ausführung von Abläufen und das Eintreffen und Bearbeiten von Ereignissen werden dadurch transparenter.

Durch die Verbesserungen in EJB Version 3.1 wurden einige Schwächen behoben. Die erwähnten Annotationen erlauben mehr als in der Version 3.0 und die Klasse `TimerService` besitzt einen größeren Funktionsumfang. Es kann vermutet werden, dass sich hier noch weitere Verbesserungen ergeben.

Literaturverzeichnis

Cavaness, C. (2006) *Quartz Job Scheduling Framework: Building Open Source Enterprise Applications*, Prentice Hall. Verfügbar unter: http://www.amazon.com/Quartz-Job-Scheduling-Framework-Applications/dp/0131886703

Gamma, E. et al., 2004. *Entwurfsmuster* Übersetzer: D. Riehle, Addison-Wesley

Java Community Process (JCP), http://jcp.org

JCP, 2006. Java™ Platform, Enterprise Edition 5 (Java EE 5) Specification, JSR 244

JCP, 2009. Java™ Platform, Enterprise Edition 6 (Java EE 6) Specification, JSR 316

JCP, 2003. Enterprise JavaBeans™ 2.1, JSR 153

JCP, 2006. Enterprise JavaBeans™ 3.0, JSR 220

JCP, 2009. Enterprise JavaBeans™ 3.1, JSR 318

Opensymphony, http://www.opensymphony.com

Quartz, http://www.quartz-scheduler.org

Terracotta Inc., http://www.terracotta.org

Torque Resource Manager, http://www.clusterresources.com/products/torque-resource-manager.php

UC4 Software GmbH, http://www.uc4.com

Kontaktadresse:

Ralph Löwe
Hochschule München
Lothstraße, 34
D-80335 München

Telefon:	+49 (0) 89 1265 3776
Fax:	+49 (0) 89 1265 3780
E-Mail	rloewe@hm.edu
Internet:	http://www.cs.hm.edu