

Scheduling und Thread-Ausführer



Scheduling

- Ein **Scheduler** arbeitet Programmstücke
 - ▶ nach einer festen Zeitspanne oder zu einer fixen Zeitpunkt
 - ▶ wiederholt oder einmalab.
- Notwendigkeiten für Scheduling sind zum Beispiel:
 - ▶ In regelmäßigen Abständen Reports erstellen.
 - ▶ Aufräumarbeiten durchführen, wie temporäre Daten löschen.
- Spring unterstützt direkt zwei Scheduling Technologien:
 - ▶ Die eingebauten `java.util.Timer`.
 - ▶ Das Scheduling-Framework Quartz (<http://www.opensymphony.com/quartz>).

Vorteile Quartz gegenüber Timer

- Die Timer in Java sind relativ einfach, während Quartz eine Menge von Möglichkeiten bietet:
 - ▶ **Persistenz der Timer.** Die Informationen über den Zeitverbleib lassen sich zum Beispiel in einer Datenbank sichern.
 - ▶ **Gruppen bilden.** Quartz kann Gruppen von Jobs bilden und unter einem Namen verwalten.
 - ▶ **Cron-Jobs.** Die unter Unix bekannten Cron-Jobs definieren über Strings Wiederholungen, Zeitpunkte, Intervalle.
 - ▶ **Thread-Pools.** Quartz geht performanter mit Timern um, da zur Abarbeitung neuer Jobs ein Thread aus einem Pool genommen wird.

Was kann Spring?

- Das Spring-Framework hilft bei Scheduling auf mehreren Ebenen:
 - ▶ Der Container ruft automatisch entweder über `java.util.Timer` oder Quartz beliebige Methoden auf.
 - ▶ Die aufgerufenen Methoden haben keinen Bezug zu einer Scheduling-Realisierung wie `java.util.Timer` oder Quartz, teilen also eine besondere Schnittstelle.
 - ▶ Die Konfiguration ist ähnlich und extern, so dass später beim Austausch nur in der Konfig-Datei eine Änderung nötig ist, aber nicht im Programm.

Scheduling mit `java.util.Timer`



Die mitspielenden Klassen

- Beim Scheduling ist der einfachste Fall, dass Spring von einem Objekt eine bestimmte Methode aufruft.
- Die `MethodInvokingTimerTaskFactoryBean` konfiguriert über
 - ▶ `targetObject` das Ziel-Objekt und über
 - ▶ `targetMethod` die aufzurufende Methode.
- Der `ScheduledTimerTask` bestimmt für eine konfigurierte Bean die Ausführungsparameter:
 - ▶ Einen Verweis auf die Aufgabe (`timerTask`).
 - ▶ Nach welcher Wartezeit der Task beginnt (`delay`).
 - ▶ Welchen Intervall gibt es (`period`).
- Zum Schluss bestimmt eine `TimerFactoryBean` einer Liste (`scheduledTimerTasks`) der Tasks.

Ein Dauerlacher

- Nehmen wir eine Klasse `Dauerlacher` an, die eine Methode `lache()` definiert.

```
package com.tutego.spring.scheduling;
```

```
public class Dauerlacher
```

```
{
```

```
    public void lache()
```

```
    {
```

```
        System.out.println( "Hi hi " );
```

```
    }
```

```
}
```

MethodInvokingTimerTaskFactoryBean

```
<bean id="MyTask" class=
    "org.springframework.scheduling.timer.MethodInvokingTimerTaskFactoryBean" >
    <property name="targetObject">
        <bean class="com.tutego.spring.scheduling.Dauerlacher" />
    </property>
    <property name="targetMethod" value="lache" />
</bean>
```

Task-Beschreibung ScheduledTimerTask

```
<bean id="MyScheduledTask"  
    class="org.springframework.scheduling.timer.ScheduledTimerTask">  
    <property name="delay" value="1000" />  
    <property name="period" value="1000" />  
    <property name="timerTask" ref="MyTask" />  
</bean>
```

Wann die Abarbeitung beginnt (nach erster Sekunde)

In welchem Intervall Wiederholungen (jede Sekunde)

TimerFactoryBean listet Tasks auf

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

Thread-Pooling



TaskExecutor und Java 5 Executor

- Seit Java 5 gibt es mit dem `java.util.concurrent.Executor` (JSR 166) eine Schnittstelle für „Ausführer“ von `Runnable`-Objekten.
- Spring bietet mit dem `TaskExecutor` die gleiche Schnittstelle.
 - ▶ So steht die Funktionalität auch für Java 1.4 und Java 1.3 und auch für Java EE Umgebungen zur Verfügung.



TaskExecutor-Klassen (1/2)

- SimpleThreadPoolTaskExecutor
 - ▶ Thread-Pool durch Quartz's SimpleThreadPool.
- SimpleAsyncTaskExecutor
 - ▶ Erzeugt pro Anfrage einen neuer Thread.
 - ▶ Es gibt eine optionale maximale Größe.
- SyncTaskExecutor
 - ▶ Führt Runnable im aktuellen Thread aus.
 - ▶ Erzeugt also keinen neuen Thread zur Nebenläufigkeit.
- ThreadPoolTaskExecutor
 - ▶ Geht auf den `java.util.concurrent.ThreadPoolExecutor`.
 - ▶ Ermöglicht Anpassung der Properties "corePoolSize", "maxPoolSize", "keepAliveSeconds", "queueCapacity".

TaskExecutor-Klassen (2/2)

- **TimerTaskExecutor**
 - ▶ Nutzt einen `java.util.Timer` zur Ausführung.
 - ▶ Der `Runnable` wird als `TimerTask` hintereinander durch den Timer-Thread ausgeführt.
- **WorkManagerTaskExecutor**
 - ▶ Nutzt CommonJ WorkManager als Implementierung. Ermöglicht Ausführung auf Applikationsservern von BEA und IBM.
- **ConcurrentTaskExecutor**
 - ▶ Ein Wrapper zum `java.util.concurrent.Executor` von Java 5.

Beispiel für TaskExecutor

```
public class DoItExecutor {
    private TaskExecutor executor;
    public DoItExecutor( TaskExecutor executor ) {
        this.executor = executor;
    }
    public void doIt() {
        Runnable run = new Runnable() {
            public void run() {
                System.out.println( Thread.currentThread() );
            }
        };
        executor.execute( run ); executor.execute( run );
    }
}
```

Injizierung vom TaskExecutor

```
<bean id="taskExecutor" class="
    "org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">
    <property name="corePoolSize" value="5" />
    <property name="maxPoolSize" value="10" />
    <property name="queueCapacity" value="25" />
</bean>
```

```
<bean id="doItExecutor" class="DoItExecutor">
    <constructor-arg ref="taskExecutor" />
</bean>
```

„Fehlende“ Typen aus Java 5

- Spring bietet unterschiedliche Implementierungen für den `TaskExecutor`, aber Java 5 bietet außer `execute()` im `Executor` noch mehr.
- Der `java.util.concurrent.ThreadPoolExecutor` implementiert mehr als nur `Executor`, nämlich `ExecutorService`. Das bietet
 - ▶ `awaitTermination()`, `invokeAll()`, `invokeAny()`, `isShutdown()`, `isTerminated()`, `shutdown()`, `shutdownNow()`, `submit()`
 - ▶ mit weiteren Typen wie `Future`, `Callable`.
- Spring bietet dafür bisher keine Abstraktion!