Scheduled tasks with the Timer Service

COLLABORATORS			
	TITLE : Scheduled tasks with the Timer Service		*REFERENCE* :
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 1, 2010	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Scheduled tasks with the Timer Service

1.1 Introduction

Most session beans offer services that are either invoked by clients (in the case of stateless and stateful session beans), or by the delivery of JMS messages (for message-driven beans). However, certain workflow steps may need to be performed based on scheduled, timed events (i.e. without direct user interaction).

Historically, this was accomplished by using an external timer service (such as `cron`) to invoke a program that connects to and invokes a service from an EJB. This mechanism, however, is laborious, and not contained as part of the Java EE solution.

EJB 3.0 introduced the *Timer Service*, a container-managed facility which can be scheduled to reliably invoke a *callback* service on EJBs based on any number of schedules of the following type:

- Call back after a fixed duration (i.e. after a week)
- Call back at a regular interval after a fixed duration (i.e. after five minutes, call back every minute)
- Call back at an absolutely specified future date/time (i.e. at 17:00 on 8 June 2011)
- Call back at a regular interval after an absolutely specified date/time (i.e. every hour after 19:00 tonight)

The EJB can control, inspect, and/or cancel the timer after it has been created.

1.2 What can the EJB Timer Service be used for?

The EJB timer service primarily exists to support long-running business processes. Since session beans conceptually only last for the duration of a session, the timer service extends the usefulness of EJBs in situations such as the following:

- **Time-based workflow steps** For example, imposing a fine when a rented item (library book, video, car) is not returned on time, or prompting a client to renew their support contract a month before it expires.
- **Monitoring / Polling** Periodically making sure that certain services, resources or devices are available and responding.
- **Periodical reminders** Sending a monthly reminder to users of a system to remind them to update their password for security purposes.

Scheduled timers are *reliable* and *robust* - they must survive application server crashes or machine restarts.

Timers may be used in both `Stateless Session Beans` as well as *Message-Driven Beans*.

1.3 The TimerService interface

The Timer Service is accessed via dependency injection, through the `getTimerService()` method of the `javax.ejb.EJBContext` interface, or through lookup in the JNDI namespace. The TimerService interface offers the following services:

```
package javax.ejb;

public interface TimerService
{
  /* Schedule a once-off, fixed delay timer */
  public Timer createTimer(long duration, java.io.Serializable info);

  /* Schedule a recurring, fixed initial delay timer */
  public Timer createTimer(long initialDuration, long intervalDuration,
                   java.io.Serializable info);

  /* Schedule a once-off, absolute timer */
  public Timer createTimer(java.util.Date expiration,
                   java.io.Serializable info);

  /* Schedule a recurring, absolute initial timer */
  public Timer createTimer(java.util.Date initialExpiration,
                   long intervalDuration, java.io.Serializable info);

  /* Returns all the active Timer objects associated with the bean */
  public Collection getTimers();
}
```

Note
The `info` argument present on each service is to provide the timer with a piece of information which it will reproduce when it calls back the bean - a handy mechanism to provide the call-back service with information to tell it *why* it is being called, common when multiple timers are scheduled on the same bean.

To schedule a timer as part of a service request, the following is done:

```
import javax.annotation.*;
import javax.ejb.*;

@Stateless
public class SomeBean implements SomeContract
{
  public void doSomething()
  {
    // Perform business logic
    // ...

    // Schedule a timer to call back this bean after
    // an hour (delay is expressed in milliseconds)
    Timer timer = ejbContext.getTimerService()
                    .createTimer( 1000 * 60 * 60,  null );
  }

  // Called when the timer expires
  @Timeout
  public void timeIsUp(Timer timer)
  {
      // React to time event
      // ...
  }

  @Resource
  private EJBContext ejbContext;
}
```

1.3.1 The Timer interface

When a timer is scheduled, or when one calls back, a reference is provided to a `javax.ejb.Timer`, which allows the bean to inspect or cancel the timer. Multiple timers may be active for each bean. It offers the following services:

```
package javax.ejb;

public interface Timer
{
    public void cancel();

    public long getTimeRemaining();

    public java.util.Date getNextTimeout();

    public javax.ejb.TimerHandle getHandle();

    public java.io.Serializable getInfo();
}
```

1.4 Timout callbacks

When a timer that has been scheduled on a bean 'times out', it notifies the bean by calling a *call-back* service. Though a bean can have multiple active timers scheduled for it, it can only have one call-back service. The call-back service can be specified, either by the bean implementing the `javax.ejb.TimedObject` interface, which looks as follows:

```
package javax.ejb;

public interface TimedObject
{
    public void ejbTimeout(Timer timer);
}
```

With an example EJB implementing it as follows:

```
@javax.ejb.Stateless
public class MyBean implements MyBusinessInterface, javax.ejb.TimedObject
{
    public void myBusinessService()
    {
        // Business logic, including scheduling of a new timer
        // ...
        ejbContext.getTimerService().createTimer( /* ... */ );
    }

    public void ejbTimeout(Timer timer)
    {
        // React to time-out event
        // ...
    }

    @Resource
    private EJBContext ejbContext;
}
```

Or, alternatively, by annotating any service (public, protected, or private) which returns `void` and accepts a single argument of type `javax.ejb.Timer` with the `@javax.ejb.Timeout` annotation. For example:

```
@javax.ejb.Stateless
public class MyBean implements MyBusinessInterface
{
    public void myBusinessService()
    {
        // Business logic, including scheduling of a new timer
        // ...
        ejbContext.getTimerService().createTimer( /* ... */ );
    }

    @javax.ejb.Timeout
    public void timeIsUp(Timer timer)
    {
        // React to time-out event
        // ...
    }

    @Resource
    private EJBContext ejbContext;
}
```

1.4.1 Controlling the timer

As the call-back service receives the timer, it may (say, in the case of a recurring timer) inspect it to see when it will next call again, or it may choose to cancel it in order to no longer receive call-backs. For example:

```
@javax.ejb.Stateless
public class MyBean implements MyBusinessInterface
{
    ...

    @javax.ejb.Timeout
    public void timeIsUp(Timer timer)
    {
        // React to time event
        doSomething();

        // Cancel further call-backs under some circumstance
        if ( /* ... */ )
        {
            timer.cancel();
        }
    }
}
```

1.4.2 Passing information through a timer

Often it is convenient, when scheduling a timer, to provide it with some information which it must reproduce when it calls back. This information is passed as an argument during one of the `createTimer(...)` service invocations. For example, a registration service might wish to schedule a ‘welcome’ e-mail to be sent to newly registered users one day after they register: The timer may be provided with an e-mail address, which it will provide to the call-back service.

```
@javax.ejb.Stateless
public class MyBean implements MyBusinessInterface
{
    /* Registers a new client */
    public void registerNewClient( Registration r )
    {
        // Perform business logic
        // ...

        // Schedule a timer to send a welcome e-mail after 24 hours
        // The client's e-mail address is passed to the timer
        ejbContext.getTimerService().createTimer(
            1000 * 60 * 60 * 24, r.getEmailAddress() );
    }

    @javax.ejb.Timeout
    public void sendWelcomeEmail(Timer timer)
    {
        String emailAddress = (String)timer.getInfo();

        // Use JavaMail to send e-mail message
        // ...
    }

    @Resource
    private EJBContext ejbContext;
}
```

Note
The information passed to a timer must implement `java.io.Serializable`, as the timer must be able to store it over a potentially long period of time (across app server restarts, etc).

1.4.3 Understanding the call-back invocation

Timer services can only be used to call-back stateless session beans, as well as message-driven beans. This allows the incoming call to be treated like any other call to a stateless bean: The application server will direct the call to any available bean from the method-ready pool, which results in very high performance, as well as minimal interference of timer call-backs with normal business services.

The call-back service may, of course, use any of the facilities at its disposal (such as injecting a `javax.persistence.EntityManager`) in order to look up or query information needed to complete the call-back workflow. It usually uses information provided to it by the timer itself to perform such lookups.

1.5 Persistent Timer References

Often, different services offered by a stateless EJB need to interact with the same timer, in order to, for example, cancel it. Imagine a traffic fine system, where, upon notification of an offence, a session bean schedules a timer which will ultimately result in a warrant of arrest being served upon the offender within a month of not paying.

However, it offers a service of paying for a fine, which should cancel the timer that will dispatch the friendly police officer to your residence. This service may even reside in a different EJB. For example:

```
@Stateless
public class TrafficLawEnforcementBean implements TrafficLawEnforcement
{
    public void registerOffence( Offence o )
    {
        // Dispatch fine to user's postal address
        // ...

        // Schedule a warrant-of-arrest timer
        Timer t = ejbContext.getTimerService().createTimer(
                           /* 1 month */ , o.getDriverDetails() );
    }

    public void payForOffence( PaymentRequest p )
    {
        // Process payment
        // ...

        // Cancel Timer
        // ???
    }

    @Timeout
    public void dispatchOfficer(Timer timer)
    {
        DriverDetails d = (DriverDetails)timer.getInfo();
        dispatch.dispatchVehicle( /* ... */ );
    }

    @Resource
    private EJBContext ejbContext;

    @EJB
    private VehicleDispatch dispatch;
}
```

The problem lies with the fact that a stateless session bean cannot maintain object state between service requests (i.e. we cannot assign the created timer to an instance variable). Furthermore, multiple timers may be scheduled for a single bean. One solution to this problem, if the different services are within the same bean, is to ask the `TimerService` for all the timers for the current bean, and manually inspect them to find the one to cancel. For example:

```
@Stateless
public class TrafficLawEnforcementBean implements TrafficLawEnforcement
{
    ...

    public void payForOffence( PaymentRequest p )
    {
        // Process payment
        // ...

        // Look for applicable timer, and cancel it
        for (Timer t : ejbCOntext.getTimerService().getTimers())
        {
            if (t.getInfo().equals( /* ... */ )
            {
                t.cancel();
            }
        }
    }
    ...

    @Resource
    private EJBContext ejbContext;

}
```

This is however not an elegant solution, and we may wish to use the `TimerHandle` interface instead.

1.5.1 TimerHandle

A `javax.ejb.TimerHandle` is a serializable, persistent reference to an active timer. It is created by requesting a service from a timer:

```
                Timer t = ... ;
                TimerHandle th = t.getHandle();
```

Because it is serializable, the handle may now be persisted, such as storing it in the database. A common technique is to use a JPA Entity which has, as one of its fields, a TimerHandle. At any time when a handle to the original, running timer is required, it is simply requested from the timer handle:

```
                TimerHandle th = /* Get from database */;
                Timer t = th.getTimer();
```

The traffic law-enforcement bean may now be implemented in a manner where the timer is stored in the database in the one service, and retrieved from the database in the other: A simple entity bean called `WarrantOfArrestTimer` is used in this example:

```
@Stateless
public class TrafficLawEnforcementBean implements TrafficLawEnforcement
{
    public void registerOffence( Offence o )
    {
        // Dispatch fine to user's postal address
        // ...

        // Schedule a warrant-of-arrest timer
        Timer t = ejbContext.getTimerService().createTimer(
                            /* 1 month */ , o.getDriverDetails() );

        // Store timer info in the database
        em.persist( new WarrantOfArrestTimer( t, o.getId() ) );
    }

    public void payForOffence( PaymentRequest p )
    {
        // Process payment
        // ...

        // Cancel Timer
        WarrantOfArrestTimer woa = em.find( WarrantOfArrestTimer.class, p.getOffence(). ↩
            getId() );
        TimerHandle handle = woa.getTimerHandle();
        handle.getTimer().cancel();
    }

    @Timeout
    public void dispatchOfficer(Timer timer)
    {
        DriverDetails d = (DriverDetails)timer.getInfo();
        dispatch.dispatchVehicle( /* ... */ );
    }

    @Resource
    private EJBContext ejbContext;

    @EJB
    private VehicleDispatch dispatch;

    @PersistenceContext
    private EntityManager em;
}
```

1.5.1.1 Referencing an expired timer

If a `TimerHandle` is serialized, say, to a database, the information will exist until it is explicitly removed, which may be for a very long time.

A Timer handle can, however, only be used to reference a timer which exists in the application server. Calling `getTimer()` to obtain a timer which has been cancelled, or a single-event timer which has already timed out, will cause a `javax.ejb.NoSuchObjectLocalException` to be thrown. This is because, once a timer's work is complete, the application server will remove it from the runtime environment.

Any code which thus uses a TimerHandle to reference a timer, *must* be prepared to catch this exception. When a timer is cancelled, it would also make sense to remove any persistent remnants of the time handle from, for example, the database, as the handle can no longer be used to obtain information about the timer. For example, to cancel a timer which was stored in a database, the following approach could be used:

```
...

/* An internal service which may be used by other services to cancel the
   reminder timer for a given account */
private void cancelTimerForAccountReminder( Account a )
{
    // Use information form account to get timer handle from database
    SomeEntity entity = entityManager.find( /* ... */ );
    TimerHandle handle = entity.getTimerHandle();

    try
    {
        // Use handle to get timer and cancel it
        Timer t = handle.getTimer();
        t.cancel();
    }
    catch (NoSuchObjectLocalException e)
    {
        // Timer no longer active, so there is nothing to cancel
    }
    finally
    {
        // Remove from database
        entityManager.remove( entity );
    }
}

...
```

1.6 Examples of using the EJB Timer Service

1.6.1 Library

Consider a public library: when a member checks out an item (such as a book), a timer is set for two weeks in the future. When the timer expires, it means that the member has not yet checked the borrowed item(s) back in - a fine should be levied, and a clerk should perhaps be notified to give the member a call.

The service contract for the library service (purposefully simplified) looks as follows:

```
package za.co.solms.example;

/** Defines the services offered by a library: Used by the
 * books checkout counter, or a self-service kiosk. */
public interface Library
{
  /** Checks out an item to a library member.
   * @param checkout Information regarding the
   * member and the book to be checked out
   * @return A receipt which indicates when the book
   * should be returned
   * @throws InvalidMemberException If an invalid member
   * tries to checkout items
   * @throws CheckoutLimitExceededException If the mamximum
   * number of items allowable are already checked out.
   * @throws OutstandingPaymentException If fees or fines are
   * outstanding, no further items may be borrowed. */
  public Receipt checkOut( CheckoutRequest checkout )
  throws InvalidMemberException, CheckoutLimitExceededException,
  OutstandingPaymentException;

  /** Returns an item back to the library */
  public void checkIn( CheckInRequest checkin );
}
```

The exchanged value objects (with partially omitted implementations) are as follows:

```
package za.co.solms.example;

/** A request to check out an item by a member. For simplicity,
 * only a single item is supported for now. */
public class CheckoutRequest implements java.io.Serializable
{
  public MemberInfo getMember()
  {
    return member;
  }

  public void setMember(MemberInfo member)
  {
    this.member = member;
  }

  public ItemInfo getItem()
  {
    return item;
  }

  public void setItem(ItemInfo item)
  {
    this.item = item;
  }

  private MemberInfo member;
  private ItemInfo item;
}
```

```
package za.co.solms.example;

/** A receipt issued after successful checkout of item(s)
 * from the library. */
public class Receipt implements java.io.Serializable
{
  public java.util.Date getReturnDate()
  {
    return returnDate;
  }

  public void setReturnDate( java.util.Date returnDate )
  {
    this.returnDate = returnDate;
  }

  // TODO: Extra info
  // ...

  private java.util.Date returnDate;
}
```

```
package za.co.solms.example;

/** A request that checks in (returns) an item to the library.
 * For simplicity, only a single item is supported.
 */
public class CheckInRequest implements java.io.Serializable
{
  public ItemInfo getItem()
  {
    return item;
  }

  public void setItem(ItemInfo item)
  {
    this.item = item;
  }

  private ItemInfo item;
}
```

The stateless session bean which implements the functionality to, upon checkout, schedule a once-off timer, and to cancel the timer when the item is checked back in, is as follows:

```
package za.co.solms.example.impl;

import java.util.*;
import javax.annotation.*;
import javax.ejb.*;
import javax.ejb.Timer;
import javax.jws.*;
import javax.persistence.*;
import za.co.solms.example.*;

@Stateless
@Remote({Library.class})
public class LibraryBean implements Library
{
  public Receipt checkOut(CheckoutRequest checkout)
  throws InvalidMemberException, CheckoutLimitExceededException,
  OutstandingPaymentException
  {
    // Check preconditions
    // ...

    // Perform business logic of checking out an item
    //...

    // Item must be returned 14 days from now
    Calendar returnDate = Calendar.getInstance();
    returnDate.add( Calendar.DATE, 14);

    // Schedule a timer to call back this on the return date
    createReturnTimer(checkout.getItem(), returnDate.getTime());

    // Create receipt
    Receipt receipt = new Receipt();
    receipt.setReturnDate( returnDate.getTime() );
    return receipt;
  }


  public void checkIn(CheckInRequest checkin)
  {
    // Perform business logic
    // ...

    // Cancel timer
    cancelReturnTimer( checkin.getItem() );
  }


  /* Creates a timer that is set to call back on the given
   * returnDate, with the given item as info. */
  private void createReturnTimer( ItemInfo item, Date returnDate )
  {
    // Schedule timer for return date. As info, provide the item
    Timer timer = ejbContext.getTimerService().createTimer(
        returnDate, item);

    // Persistently store a handle to the timer in database, with
    // borrowed item serial number as key
    ItemTimer timerInfo = new ItemTimer(
        item.getSerialNumber(), timer.getHandle() );
    entityManager.merge( timerInfo );
  }


  /* Cancels the timer for the given item */
  private void cancelReturnTimer( ItemInfo item )
  {
    // Lookup the timer handle for the given item
    ItemTimer timerInfo = entityManager.find(
        ItemTimer.class, item.getSerialNumber() );

    if (timerInfo != null)
    {
      try
      {
        // Use the timer handle to get the timer, and cancel it
        Timer timer = timerInfo.getTimerHandle().getTimer();
        timer.cancel();
      }
      catch (NoSuchObjectLocalException ignored)
      {
        // Timer no longer active (i.e. timeOut callback already occurred)
      }
      catch (NullPointerException ignored)
      {
        // Timer does not exist in the first place
      }
      finally
      {
        // Remove timer handle from database
        entityManager.remove( timerInfo );
      }
    }
  }


  /* Called by a timer, we assume to indicate that an item
   * has not yet been returned. */
  @Timeout
  private void timeOut( Timer timer )
  {
    ItemInfo item = (ItemInfo)timer.getInfo();
    System.out.println("** ALERT: Item not yet returned: " + item);

    // Perform business logic (e.g. allocate fine)
    //...
  }


  @Resource
  private EJBContext ejbContext;

  @PersistenceContext
  private EntityManager entityManager;
}
```

The entity which stores the timer handle in the database (an internal artifact, never seen by the member which uses the library service) is as follows:

```
package za.co.solms.example.impl;

import java.io.Serializable;
import javax.ejb.TimerHandle;
import javax.persistence.*;

/** An entity which maps a persistent timer handle to
the identifier for a library item */

@Entity
public class ItemTimer implements Serializable
{
  public ItemTimer(){}

  public ItemTimer( int item, TimerHandle timerHandle )
  {
    setItem(item);
    setTimerHandle(timerHandle);
  }

  public int getItem()
  {
    return item;
  }

  public void setItem(int item)
  {
    this.item = item;
  }

  public TimerHandle getTimerHandle()
  {
    return timerHandle;
  }

  public void setTimerHandle(TimerHandle timerHandle)
  {
    this.timerHandle = timerHandle;
  }

  @Id
  private int item;
  private TimerHandle timerHandle;
}
```

Chapter 2

Index