Schedule CDI Events

Example schedule-events can be browsed at https://github.com/apache/tomee/tree/master/examples/schedule-events

This example uses a nice CDI/EJB combination to schedule CDI Events. This is useful if you want CDI Events that fire regularly or at a specific time or calendar date.

Effectively this is a simple wrapper around the `BeanManager.fireEvent(Object,Annotations…)` method that adds `ScheduleExpression` into the mix.

ScheduleExpression and @Timeout

The logic here is simple, we effecitvely expose a method identical to `BeanManager.fireEvent(Object, Annotations…)` and wrap it as `scheduleEvent(ScheduleExpression, Object, Annotation…)`

To do that we use the EJB `TimerService` (under the covers this is Quartz) and create an `@Timeout` method which will be run when the `ScheduleExpression` activates.

The `@Timeout` method, simply called `timeout`, takes the event and fires it.

```
@Singleton
@Lock(LockType.READ)
public class Scheduler {

    @Resource
    private TimerService timerService;

    @Resource
    private BeanManager beanManager;

    public void scheduleEvent(ScheduleExpression schedule, Object event, Annotation...
qualifiers) {

        timerService.createCalendarTimer(schedule, new TimerConfig(new EventConfig
(event, qualifiers), false));
    }

    @Timeout
    private void timeout(Timer timer) {
        final EventConfig config = (EventConfig) timer.getInfo();

        beanManager.fireEvent(config.getEvent(), config.getQualifiers());
    }

    // Doesn't actually need to be serializable, just has to implement it
    private final class EventConfig implements Serializable {

        private final Object event;
        private final Annotation[] qualifiers;

        private EventConfig(Object event, Annotation[] qualifiers) {
            this.event = event;
            this.qualifiers = qualifiers;
        }

        public Object getEvent() {
            return event;
        }

        public Annotation[] getQualifiers() {
            return qualifiers;
        }
    }
}
```

Then to use it, have `Scheduler` injected as an EJB and enjoy.

```
public class SomeBean {

    @EJB
    private Scheduler scheduler;

    public void doit() throws Exception {

        // every five minutes
        final ScheduleExpression schedule = new ScheduleExpression()
                .hour("*")
                .minute("*")
                .second("*/5");

        scheduler.scheduleEvent(schedule, new TestEvent("five"));
    }

    /**
     * Event will fire every five minutes
     */
    public void observe(@Observes TestEvent event) {
        // process the event
    }

}
```

Test Case

A working test case for the above would be as follows:

```
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import javax.ejb.AccessTimeout;
import javax.ejb.EJB;
import javax.ejb.ScheduleExpression;
import javax.ejb.embeddable.EJBContainer;
import javax.enterprise.event.Observes;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;

/**
 * @version $Revision$ $Date$
 */
public class SchedulerTest {

    public static final CountDownLatch events = new CountDownLatch(3);
```

```
    @EJB
    private Scheduler scheduler;

    @Test
    public void test() throws Exception {

        final ScheduleExpression schedule = new ScheduleExpression()
                .hour("*")
                .minute("*")
                .second("*/5");

        scheduler.scheduleEvent(schedule, new TestEvent("five"));

        Assert.assertTrue(events.await(1, TimeUnit.MINUTES));
    }


    @AccessTimeout(value = 1, unit = TimeUnit.MINUTES)
    public void observe(@Observes TestEvent event) {
        if ("five".equals(event.getMessage())) {
            events.countDown();
        }
    }

    public static class TestEvent {
        private final String message;

        public TestEvent(String message) {
            this.message = message;
        }

        public String getMessage() {
            return message;
        }
    }

    @Before
    public void setup() throws Exception {
        EJBContainer.createEJBContainer().getContext().bind("inject", this);
    }
}
```

You must know

- CDI Events are not multi-treaded

If there are 10 observers and each of them take 7 minutes to execute, then the total execution time for the one event is 70 minutes. It would do you absolutely no good to schedule that event to fire more frequently than 70 minutes.

What would happen if you did? Depends on the `@Singleton` `@Lock` policy

- `@Lock(WRITE)` is the default. In this mode the `timeout` method would essentially be locked until the previous invocation completes. Having it fire every 5 minutes even though you can only process one every 70 minutes would eventually cause all the pooled timer threads to be waiting on your Singleton.
- `@Lock(READ)` allows for parallel execution of the `timeout` method. Events will fire in parallel for a while. However since they actually are taking 70 minutes each, within an hour or so we'll run out of threads in the timer pool just like above.

The elegant solution is to use `@Lock(WRITE)` then specify some short timeout like `@AccessTimeout(value = 1, unit = TimeUnit.MINUTES)` on the `timeout` method. When the next 5 minute invocation is triggered, it will wait up until 1 minute to get access to the Singleton before giving up. This will keep your timer pool from filling up with backed up jobs — the "overflow" is simply discarded.