Seam Cron Module

Reference Guide

Peter Royle

Introduction

Seam Cron is a CDI portable extension for scheduled and asynchronous method invocation. It wraps common scheduling and backgrounding tasks in an intuitive, type-safe, event driven API.

1.1. Mission statement

Seam Cron aims to simplify configuration and deployment of scheduled tasks across multiple CDI containers and for multiple scheduling engine implementations.

1.2. Introductory Examples

1.2.1. Scheduling

Seam Cron uses standard CDI `@Observer` methods for both configuration and execution of scheduled methods.

```
public void generateDailyReport(@Observes @Scheduled("01:00") Trigger trigger) { ... }
```

The method above will be executed at 1:00 AM daily. For more detailed schedule definitions use a cron-style expression. In the following example, the method will execute at 1:00 AM on the 3rd Wednesday of every month.

```
public void deliverNewsletters(@Observes @Scheduled("0 0 1 ? * 4") Trigger trigger) { .. }
```

Alternatively, use any valid property name instead of a schedule definition, and then specify the definition for that name in the `cron.properties` file at the root of your classpath. Here is an example.

```
# cron.properties
newsletter.delivery=0 0 1 ? * 4
```

```
// NewsletterBean.java
public void deliverNewsletters(@Observes @Scheduled("newsletter.delivery") Trigger trigger) { ... }
```

To replace the `@Scheduled` annotation with something more type-safe create a custom qualifier with the `@Scheduled` annotation applied to it like so:

```
@Scheduled("newsletter.delivery")
@Qualifier
@Retention( RUNTIME )
@Target( { PARAMETER })
public @interface NewsletterDelivery {}
```

Now your scheduled observer methods are fully type-safe and higly readable:

```
public void deliverNewsletters(@Observes @NewsletterDelivery Trigger trigger) { ... }
```

If your requirements are fairly simple, for example running a task repeatedly at a specific `Interval`, then you can use the `@Every` qualifier as in the example below. Valid interval values are `SECOND`, `MINUTE` and `HOUR`. `@Every` also takes an optional "`nth`" parameter which defaults to 1.

```
public void clockChimes(@Observes @Every(HOUR) Trigger t) {
   int chimes = t.getValue() % 12;
   if (chimes == 0) { chimes = 12; }
   for (int i=0; i<chimes; i++) {
      bellTower.getRope().pull();
   }
}
public void workBreak(@Observes @Every(nth=30, value=MINUTE) Trigger t) {
   ...
}
```

1.2.2. Asynchronous Method Invocation

Seam Cron provides the `@Asynchronous` annotation which when applied to a method causes that method to be invoked asynchronously. When applied to a type, all methods in that type will be invoked asynchronously.

```
@Inject DocumentIndex index;

@Asynchronous
public DocumentIndexStats reindexDocuments() {
   return index.reindex();
}
```

To obtain a reference to the result once the invocation is finished simply observe the return type of the method as follows:

```
public void reportIndexStats(@Observes DocumentIndexStats stats) {
   log.info("Index Updated: " + stats.toString());
}
```

The rules concerning return types of `@Asynchronous` methods are as follows:

- If method return type is void, no event will be fired
- If the method invocation returns a value of null, no event will be fired. This is definately something to be aware of!

You would typically want one dedicated return type per asynchronous method invocation for a one-to-one mapping between methods and their observers, but you may also have multiple asynchronous methods all reporting their results to a single observer. Alternatively you might wish to introduce some additional qualifiers. Below is an example covering these concepts.

```
@Asynchronous @Credit
public Balance addCredit(int dollars) {
   ...
   return new Ballance();
```

```
}

@Asynchronous @Debit
public Balance addDebit(int dollars) {
    ...
    return new Ballance();
}

public void reportNewBalance(@Observes Balance balance) {
    log.report(balance.amount());
}

public void trackSpending(@Observes @Debit Balance balance) {
    db.saveSpendingRecord(balance);
}
```

Finally, if you prefer a more traditional, EJB-esque approach then you can specify a return type of `Future` and use the `AsyncResult` helper to return the result of your method call. Seam Cron will automatically wrap this in a legit `Future` which the calling code can use as expected immediately.

```
@Asynchronous
public Future<Box> doSomeHeavyLiftingInTheBackground() {
    ...
    return new AsyncResult(new Box());
}
```

The calling code would look like this:

```
@Inject LiftingBean liftingBean;

public void someMethod() {
    Future<Box> future = liftingBean.doSomeHeavyLiftingInTheBackground();
    // blocks until asynch method returns or gives up
    Box result = future.get(10, SECONDS);
}
```

Configuration

2.1. Building Seam Cron From Source

2.1.1. Prerequisites:

- JDK 5 or above
- Maven 3 build tool
- Git version control client

2.1.2. Checkout and Build

```
git clone git://github.com/seam/cron.git
cd cron
mvn clean install
```

The above commands will build and install Cron into your local Maven repository. To run an example swing app use the following mvn command:

```
mvn install -Drun -Dswing-example
```

2.2. Application Configuration

2.2.1. Maven Dependencies

To use Seam Cron in your Maven project, include the following dependencies in your pom:

```
<dependency>
   <groupId>org.jboss.seam.cron</groupId>
   <artifactId>seam-cron-api</artifactId>
   <version>3.0.0.Alpha1</version>
   <scope>compile</scope>
</dependency>
<dependency>
   <groupId>org.jboss.seam.cron</groupId>
   <artifactId>seam-cron-scheduling-quartz</artifactId>
   <version>3.0.0.Alpha1</version>
```

```
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.jboss.seam.cron</groupId>
        <artifactId>seam-cron-asynchronous-quartz</artifactId>
        <version>3.0.0.Alpha1</version>
        <scope>runtime</scope>
    </dependency>
```

2.2.2. Scheduling

No additional configuration is necessary to use scheduling functionality

2.2.3. Asynchronous Method Invocation

No additional configuration is necessary to use asynchronous method invocation functionality