@Asynchronous Methods

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

The @Asynchronous annotation was introduced in EJB 3.1 as a simple way of creating asynchronous processing.

Every time a method annotated `@Asynchronous` is invoked by anyone it will immediately return regardless of how long the method actually takes. Each invocation returns a [Future][1] object that essentially starts out **empty** and will later have its value filled in by the container when the related method call actually completes. Returning a `Future` object is not required and `@Asynchronous` methods can of course return `void`.

Example

Here, in `JobProcessorTest`,

`final Future<String> red = processor.addJob("red");` proceeds to the next statement,

```
final Future<String> orange = processor.addJob("orange");
```

without waiting for the addJob() method to complete. And later we could ask for the result using the `Future<?>.get()` method like

```
assertEquals("blue", blue.get());
```

It waits for the processing to complete (if its not completed already) and gets the result. If you did not care about the result, you could simply have your asynchronous method as a void method.

[Future][1] Object from docs,

NOTE

> A Future represents the result of an asynchronous computation. Methods are provided to check if the computation is complete, to wait for its completion, and to retrieve the result of the computation. The result can only be retrieved using method get when the computation has completed, blocking if necessary until it is ready. Cancellation is performed by the cancel method. Additional methods are provided to determine if the task completed normally or was cancelled. Once a computation has completed, the computation cannot be cancelled. If you would like to use a Future for the sake of cancellability but not provide a usable result, you can declare types of the form Future<?> and return null as a result of the underlying task

The code

```
@Singleton
public class JobProcessor {
@Asynchronous
@Lock(READ)
@AccessTimeout(-1)
public Future<String> addJob(String jobName) {

    // Pretend this job takes a while
    doSomeHeavyLifting();

    // Return our result
    return new AsyncResult<String>(jobName);
}
```

```
    private void doSomeHeavyLifting() {
        try {
            Thread.sleep(SECONDS.toMillis(10));
        } catch (InterruptedException e) {
            Thread.interrupted();
            throw new IllegalStateException(e);
        }
      }
    }
=  Test
```

```
public class JobProcessorTest extends TestCase {

public void test() throws Exception {

    final Context context = EJBContainer.createEJBContainer().getContext();

    final JobProcessor processor = (JobProcessor) context.lookup("java:global/async-
methods/JobProcessor");

    final long start = System.nanoTime();

    // Queue up a bunch of work
    final Future<String> red = processor.addJob("red");
    final Future<String> orange = processor.addJob("orange");
    final Future<String> yellow = processor.addJob("yellow");
    final Future<String> green = processor.addJob("green");
    final Future<String> blue = processor.addJob("blue");
    final Future<String> violet = processor.addJob("violet");

    // Wait for the result -- 1 minute worth of work
    assertEquals("blue", blue.get());
    assertEquals("orange", orange.get());
    assertEquals("green", green.get());
    assertEquals("red", red.get());
    assertEquals("yellow", yellow.get());
    assertEquals("violet", violet.get());

    // How long did it take?
    final long total = TimeUnit.NANOSECONDS.toSeconds(System.nanoTime() - start);

    // Execution should be around 9 - 21 seconds
        // The execution time depends on the number of threads available for
asynchronous execution.
        // In the best case it is 10s plus some minimal processing time.
    assertTrue("Expected > 9 but was: " + total, total > 9);
    assertTrue("Expected < 21 but was: " + total, total < 21);

  }
}
```

Running

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running org.superbiz.async.JobProcessorTest
Apache OpenEJB 7.0.0-SNAPSHOT    build: 20110801-04:02
```

```
http://tomee.apache.org/
INFO - openejb.home = G:\Workspace\fullproject\openejb3\examples\async-methods
INFO - openejb.base = G:\Workspace\fullproject\openejb3\examples\async-methods
INFO - Using 'javax.ejb.embeddable.EJBContainer=true'
INFO - Configuring Service(id=Default Security Service, type=SecurityService,
provider-id=Default Security Service)
INFO - Configuring Service(id=Default Transaction Manager, type=TransactionManager,
provider-id=Default Transaction Manager)
INFO - Found EjbModule in classpath: g:\Workspace\fullproject\openejb3\examples\async-
methods\target\classes
INFO - Beginning load: g:\Workspace\fullproject\openejb3\examples\async-
methods\target\classes
INFO - Configuring enterprise application:
g:\Workspace\fullproject\openejb3\examples\async-methods
INFO - Configuring Service(id=Default Singleton Container, type=Container, provider-
id=Default Singleton Container)
INFO - Auto-creating a container for bean JobProcessor: Container(type=SINGLETON,
id=Default Singleton Container)
INFO - Configuring Service(id=Default Managed Container, type=Container, provider-
id=Default Managed Container)
INFO - Auto-creating a container for bean org.superbiz.async.JobProcessorTest:
Container(type=MANAGED, id=Default Managed Container)
INFO - Enterprise application "g:\Workspace\fullproject\openejb3\examples\async-
methods" loaded.
INFO - Assembling app: g:\Workspace\fullproject\openejb3\examples\async-methods
INFO - Jndi(name="java:global/async-
methods/JobProcessor!org.superbiz.async.JobProcessor")
INFO - Jndi(name="java:global/async-methods/JobProcessor")
INFO -
Jndi(name="java:global/EjbModule100568296/org.superbiz.async.JobProcessorTest!org.supe
rbiz.async.JobProcessorTest")
INFO - Jndi(name="java:global/EjbModule100568296/org.superbiz.async.JobProcessorTest")
INFO - Created Ejb(deployment-id=org.superbiz.async.JobProcessorTest, ejb-
name=org.superbiz.async.JobProcessorTest, container=Default Managed Container)
INFO - Created Ejb(deployment-id=JobProcessor, ejb-name=JobProcessor,
container=Default Singleton Container)
INFO - Started Ejb(deployment-id=org.superbiz.async.JobProcessorTest, ejb-
name=org.superbiz.async.JobProcessorTest, container=Default Managed Container)
INFO - Started Ejb(deployment-id=JobProcessor, ejb-name=JobProcessor,
container=Default Singleton Container)
INFO - Deployed Application(path=g:\Workspace\fullproject\openejb3\examples\async-
methods)
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 13.305 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 21.097s
[INFO] Finished at: Wed Aug 03 22:48:26 IST 2011
[INFO] Final Memory: 13M/145M
[INFO] ------------------------------------------------------------------------
```

How it works <small>under the covers</small>

Under the covers what makes this work is:

- The `JobProcessor` the caller sees is not actually an instance of `JobProcessor`. Rather it's a subclass or proxy that has all the methods overridden. Methods that are supposed to be asynchronous are handled differently.
- Calls to an asynchronous method simply result in a `Runnable` being created that wraps the method and parameters you gave. This runnable is given to an [Executor][3] which is simply a work queue attached to a thread pool.
- After adding the work to the queue, the proxied version of the method returns an implementation of `Future` that is linked to the `Runnable` which is now waiting on the queue.
- When the `Runnable` finally executes the method on the **real** `JobProcessor` instance, it will take the return value and set it into the `Future` making it available to the caller.

Important to note that the `AsyncResult` object the `JobProcessor` returns is not the same `Future` object the caller is holding. It would have been neat if the real `JobProcessor` could just return `String` and the caller's version of `JobProcessor` could return `Future<String>`, but we didn't see any way to do that without adding more complexity. So the `AsyncResult` is a simple wrapper object. The container will pull the `String` out, throw the `AsyncResult` away, then put the `String` in the **real** `Future` that the caller is holding.

To get progress along the way, simply pass a thread-safe object like [AtomicInteger][4] to the `@Asynchronous` method and have the bean code periodically update it with the percent complete.

Related Examples

For complex asynchronous processing, JavaEE's answer is `@MessageDrivenBean`. Have a look at the simple-mdb example

[1]: http://download.oracle.com/javase/6/docs/api/java/util/concurrent/Future.html [3]: http://download.oracle.com/javase/6/docs/api/java/util/concurrent/Executor.html [4]: http://download.oracle.com/javase/6/docs/api/java/util/concurrent/atomic/AtomicInteger.html