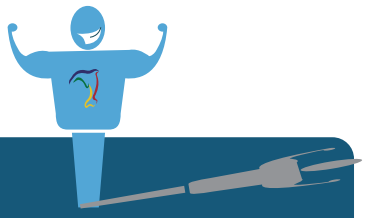
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

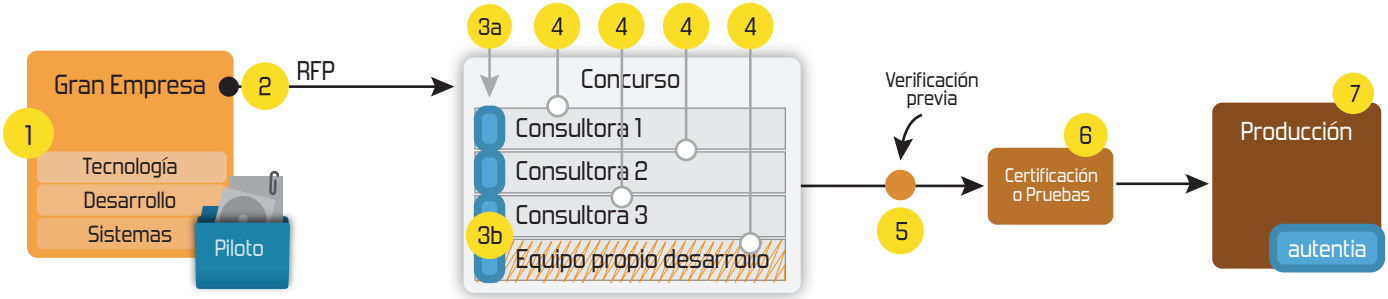
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Lanzado **TNTConcept versión 0.6** **(12/07/2007)**

Desde Autentia ponemos a vuestra disposición el software que hemos construido (100% gratuito y sin restricciones funcionales) para nuestra gestión interna, llamado TNTConcept (au**T**e**NT**ia).

Construida con las últimas tecnologías de desarrollo Java/J2EE (Spring, JSF, Acegi, Hibernate, Maven, Subversion, etc.) y disponible en licencia GPL, seguro que a muchos profesionales independientes y PYMES os ayudará a organizar mejor vuestra operativa.

Las cosas grandes empiezan siendo algo pequeño ..... Saber más en: http://tntconcept.sourceforge.net/

Fecha de creación del tutorial: 2007-08-09

EJB 3.0 y pruebas unitarias con Maven, JUnit y Embedded JBoss

Creación: 06-08-2007

Índice de contenidos

1. Introducción

Todos conocemos la importancia de los test unitarios en el proceso de desarrollo. Los tests unitarios son la garantía para poder refactorizar el código con seguridad. Nos permiten hacer cambios o introducir nuevas funcionalidades y saber al momento si algo ha dejado de funcionar como lo estaba haciendo hasta ese momento.

En Java disponemos de multitud de ayudas para realizar test:

- JUnit (http://www.junit.org/) es el indiscutible líder. Simple y sencillo nos permite realizar de forma muy sencilla tests unitarios sobre nuestras clases (http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=junit4).
- jMock (http://www.jmock.org/) permite generar mock objects al vuelo para usarlos en los tests unitarios.
- EasyMock (http://www.easymock.org/) permite generar mock objects al vuelo para usarlos en los tests unitarios.
- DbUnit (http://dbunit.sourceforge.net/) para hacer pruebas unitarias de la base de datos.
- HttpUnit (http://httpunit.sourceforge.net/) para hacer pruebas de integración, aceptación o funcionales de aplicaciones Web.
- JWebUnit (http://jwebunit.sourceforge.net/) es un wrapper sobre HttpUnit y otros motores de pruebas para Web.
- JMeter (http://jakarta.apache.org/jmeter/) para realizar pruebas funcionales, de rendimiento y de estrés de aplicaciones Web (http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=jmeter).
- ...

Si ahora usamos EJBs 3.0 deberíamos tener "algo" para poder hacer tests sobre estos elementos. Hacer tests sobre los EJBs no es sencillo ya que estos son objetos que necesitan vivir en un Servidor de Aplicaciones (contenedor). La aproximación "tradicional" para ejecutar los test podría ser:

1. Levantar el servidor
2. Hacer el despliegue
3. Ejecutar los test
4. Parar el servidor

Ya hemos visto como automatizar esto en otros tutoriales (http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=PruebasMaven), pero sigue siendo complicado y lento.

1.1. ¿Que es Embedded JBoss?

Embedded JBoss (http://wiki.jboss.org/wiki/Wiki.jsp?page=EmbeddedJBoss) aparece para resolver estos problemas. Embedded JBoss es un microcontenedor (es como si fuera un mini servidor de aplicaciones) que nos permite usar EJBs en tests unitarios, aplicaciones "stand alone" o incluso usarlo combinado con un Apache Tomcat para tener EJBs dentro de este.

Evidentemente Embedded JBoss no da toda la funcionalidad que da el Servidor de Aplicaciones JBoss, y no tiene sentido querer usarlo como sustituto. Pero si nos va a resultar muy conveniente para poder hacer tests automáticos sobre nuestros EJBs 3.0.

Ahora mismo Embedded JBoss está en Beta2 y no hay demasiada documentación. Esperamos que con la nueva versión de JBoss 5 esto vaya mejorando. Por ahora podemos encontrar información en:

- http://labs.jboss.com/jbossejb3/ - Página de JBoss sobre EJB 3.0.
- http://docs.jboss.org/ejb3/embedded/embedded.html - Página de JBoss sobre Embedded JBoss.
- http://wiki.jboss.org/wiki/Wiki.jsp?page=EmbeddedJBoss - La Wiki de Embedded JBoss (la mejor opción de las tres por estar más actualizada).

Otro recurso, aun más interesante, es la documentación y los tutoriales que podemos encontrar en la descarga del Embedde JBoss ¡¡¡ Muy recomendable !!!

1.2. Código fuente

Aquí podéis encontrar un .tar.gz con todo el código de este tutorial !!!

2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Asus G1 (Core 2 Duo a 2.1 GHz, 2048 MB RAM, 120 GB HD).
- Sistema Operativo: GNU / Linux, Debian (unstable), Kernel 2.6.21, KDE 3.5
- Embedded JBoss Beta2
- Maven 2.0.7
- Java 1.5.0_12

3. Consideraciones sobre la Máquina Virtual Java

¡¡¡ Ojo, muy importante !!!

Para que Embedded JBoss funcione correctamente es necesario ejecutarlo con una JVM versión 1.5.

Si tratamos de ejecutarlo con una JVM 6 nos dará un error en el arranque del propio Embedded JBoss. El error dirá algo similar a

```
...
ERROR 07-08 13:34:47,182 (AbstractController.java:incrementState:456) -Error installing to Instantiated:
name=DeploymentFilter state=Described
java.lang.IllegalStateException: Class not found: [Ljava.lang.String;
...
```

Parece que este problema ya lo tienen dado de alta en el sistema de incidencias de JBoss (http://jira.jboss.com/jira/browse/JBAS-4491;jsessionid=1DFC780F93A42E9D60A60AC47A8AB9F6?page=worklog).

De hecho el servidor de aplicaciones JBoss 4 (la actual versión estable) no está certificado para funcionar con JVM 6. Si usamos esta versión de JBoss (como es nuestro caso), debemos usar siempre una JVM 5 para no encontrarnos con problemas inesperados.

La versión 5 del servidor de aplicaciones JBoss ya estará preparada para trabajar con JVM 6 (por ahora podemos esperar ;).

4. Instalación de Embedded JBoss

Actualmente las librerías de Embedded JBoss no se encuentran en ningún repositorio de Maven, ni siquiera en el propio repositorio de JBoss (http://repository.jboss.com/maven2/). Esta situación tal vez se deba a que todavía están en Beta (a la hora de escribir este tutorial la última publicación es la Beta2 del 2007-04-17).

Pero esto no va a ser ningún problema para integrar los test unitarios con nuestros proyectos de Maven. Lo que vamos a hacer es descargarnos las librerías y añadirlas a nuestro repositorio local (o preferiblemente algún repositorio de Maven que tengamos en nuestras instalaciones).

En la wiki (http://wiki.jboss.org/wiki/Wiki.jsp?page=EmbeddedJBoss) podemos encontrar un enlace a la zona de descargas de sourceforege. Aquí también os dejo un enlace directo: https://sourceforge.net/project/showfiles.php?group_id=22866&package_id=228977

Descargamos el archivo **embedded-jboss-beta2.zip** y lo descomprimimos, por ejemplo en /opt:

```
$ cd /opt
$ unzip <ruta_a_la_descarga>/empedded-jboss-beta2.zip
```

Ahora vamos al directorio de librerías **/opt/embedded-jboss-beta2/lib** y nos encontramos con:

- hibernate-all.jar
- jboss-embedded-all.jar
- jboss-embedded-tomcat-bootstrap.jar
- thirdparty-all.jar

Estas 4 librerías son las que vamos a meter en nuestro repositorio local de maven con los siguientes comandos:

```
$ cd /opt/embedded-jboss-beta2/lib

$ mvn install:install-file -Dfile=hibernate-all.jar -DgroupId=org.jboss.embedded -DartifactId=hibernate-
all -Dversion=beta2 -Dpackaging=jar -DgeneratePom=true

$ mvn install:install-file -Dfile=jboss-embedded-all.jar -DgroupId=org.jboss.embedded -
DartifactId=jboss-embedded-all -Dversion=beta2 -Dpackaging=jar -DgeneratePom=true

$ mvn install:install-file -Dfile=jboss-embedded-tomcat-bootstrap.jar -DgroupId=org.jboss.embedded -
DartifactId=jboss-embedded-tomcat-bootstrap -Dversion=beta2 -Dpackaging=jar -DgeneratePom=true

$ mvn install:install-file -Dfile=thirdparty-all.jar -DgroupId=org.jboss.embedded -
DartifactId=thirdparty-all -Dversion=beta2 -Dpackaging=jar -DgeneratePom=true
```

Si en vez de en nuestro repositorio local quisiéramos añadirlos al repositorio de nuestra organización tendríamos que poner algo como (acordaros que esto lo vimos en http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=mavenRelease):

```
$ mvn deploy:deploy-file -Dfile=hibernate-all.jar -DgroupId=org.jboss.embedded -DartifactId=hibernate-all
-Dversion=beta2 -Dpackaging=jar -DrepositoryId=autentia-repository -
Durl=scp://servidorDeAutentia/maven/repository

$ mvn deploy:deploy-file -Dfile=jboss-embedded-all.jar -DgroupId=org.jboss.embedded -DartifactId=jboss-
```

```
embedded-all -Dversion=beta2 -Dpackaging=jar -DrepositoryId=autentia-repository -
Durl=scp://servidorDeAutentia/maven/repository


$ mvn deploy:deploy-file -Dfile=jboss-embedded-tomcat-bootstrap.jar -DgroupId=org.jboss.embedded -
DartifactId=jboss-embedded-tomcat-bootstrap -Dversion=beta2 -Dpackaging=jar -DrepositoryId=autentia-
repository -Durl=scp://servidorDeAutentia/maven/repository


$ mvn deploy:deploy-file -Dfile=thirdparty-all.jar -DgroupId=org.jboss.embedded -DartifactId=thirdparty-
all -Dversion=beta2 -Dpackaging=jar -DrepositoryId=autentia-repository -
Durl=scp://servidorDeAutentia/maven/repository
```

Nótese como en cualquier caso todas las librerías las hemos metido con **groupId=org.jboss.embedded** y **version=beta2**. Así las tenemos bien identificadas y sabemos que son las dependencias que necesitamos para nuestro contenedor embebido.

5. Añadiendo las dependencias a nuestro pom.xml

Ahora lo que vamos a hacer es añadir las dependencias a los jars de Embedded JBoss en nuestro pom.xml. Este tutorial parte del tutorial anterior (http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=ejb3WithJBoss), así que no vamos a mostrar todo el código del pom.xml, sólo lo que tenemos que añadir al fichero **autentia-parent/autentia-ejb/pom.xml** (en cualquier caso en la introducción tenéis un .tar.gz con el código completo):

```
...
<dependency>
   <groupId>org.jboss.embedded</groupId>
   <artifactId>hibernate-all</artifactId>
   <version>beta2</version>
   <scope>test</scope>
</dependency>

<dependency>
   <groupId>org.jboss.embedded</groupId>
   <artifactId>jboss-embedded-all</artifactId>
   <version>beta2</version>
   <scope>test</scope>
</dependency>

<dependency>
   <groupId>org.jboss.embedded</groupId>
   <artifactId>jboss-embedded-tomcat-bootstrap</artifactId>
   <version>beta2</version>
   <scope>test</scope>
</dependency>

<dependency>
   <groupId>org.jboss.embedded</groupId>
   <artifactId>thirdparty-all</artifactId>
   <version>beta2</version>
   <scope>test</scope>
</dependency>
...
```

6. Configurando nuestros test

Embedded JBoss no deja de ser un contenedor de EJBs (un microcontenedor), y por lo tanto requiere de unos cuentos ficheros de configuración (casi los mismos que encontraríamos en servidor de aplicaciones JBoss).

Todos los ficheros de configuración los podemos encontrar en el directorio **/opt/embedded-jboss-beta2/bootstrap**. Tendremos que asegurarnos de que estos ficheros acaban en el CLASSPATH de nuestros test unitarios. Para ello, y usando Maven, lo mejor es copiarlos al directorio **src/test/resources** de nuestro proyecto (antes de hacer la copia asegurar que el directorio destino existe):

```
$ cd /opt/embedded-jboss-beta2/bootstrap
$ cp -R * <ruta_al_proyecto>/autentia-parent/autentia-ejb/src/test/resources/
```

7. Creando nuestro TestCase

Con JUnit 4 (JUnit con anotaciones) vamos a realizar nuestro caso de test. En el .tar.gz que hay en la introducción podéis encontrar el código completo del test (EJBsTest.java). Aquí vamos a ir viendo las partes interesantes referentes a Embedded JBoss.

```
private static AssembledDirectory jar;
```

Lo primeo que hacemos declararnos un `AssembledDirectory`. Esta clase permite representar los elementos que vamos a desplegar en el contenedor. Es como si simulara un directorio del disco duro o un jar (aunque sólo existe en memoria).

```
private static void deploy() throws DeploymentException {
   jar = AssembledContextFactory.getInstance().create("ejbTestCase.jar");
   jar.addClass(Greeting.class);
   jar.addClass(Greeter.class);
   jar.addClass(DummyGreeterBean.class);
   jar.addClass(SmartGreeterBean.class);
   jar.addClass(GreetingModerator.class);
   jar.addClass(GreetingModeratorBean.class);
   jar.addClass(PoliteSmartGreeterBean.class);
   jar.mkdir("META-INF").addResource("test-persistence.xml", "persistence.xml");
   Bootstrap.getInstance().deploy(jar);
    }
```

Tenemos un método `deploy` que se encarga de añadir a nuestro `AssmbledDirectory` todos los elementos que queremos desplegar en el contenedor. Estos elementos se buscaran en el CLASSPATH del test, pero de cara al contenedor es como si todos estuvieran dentro del jar `ejbTestCase.jar`.

Aquí es muy importante no olvidarnos de ninguna clase necesaria.

También hay que destacar la línea `jar.mkdir...` en esta línea estamos "montando" el directorio META-INF del jar, donde JBoss irá a buscar el fichero `persistence.xml` (donde se define el Data Source que hay que usar, el proveedor de JTA, ...).

Fijaros como al método `addResource` le estamos pasando dos argumentos: el primero es el nombre real del fichero que tiene que estar en el CLASSPATH del test. El segundo es el nombre "virtual" con el que el contenedor verá el fichero.

> **Nota:** ¿Porque hacemos esto? Esto es necesario porque el contenedor siempre busca el fichero `persistence.xml`, y nosotros tenemos este fichero duplicado en el CLASSPATH uno en **src/main/resources** para la aplicación "de verdad" y otro en **src/test/resoureces** para usarlo en los test. Como el fichero está duplicado lo que hemos hecho es cambiarle el nombre al que se encuentra en src/test/resources, y con `addResource` le hacemos el cambio de nombre "en caliente" para "engañar" al contenedor y que lo encuentre correctamente.

```
private static void undeploy() throws DeploymentException {
   Bootstrap.getInstance().undeploy(jar);
   AssembledContextFactory.getInstance().remove(jar);
    }
```

Método para replegar lo que habíamos desplegado con el método deploy.

```
@BeforeClass
public static void setUpBeforeAllTest() throws Exception {
   if (!Bootstrap.getInstance().isStarted()) {

            org.jboss.net.protocol.URLStreamHandlerFactory factory =
          new org.jboss.net.protocol.URLStreamHandlerFactory();
            URL.setURLStreamHandlerFactory(factory);

            Bootstrap.getInstance().bootstrap();
        }
        deploy();
    }
```

Este método se encarga de hacer la inicialización del contenedor una única vez antes de ejecutar los test. Esto es muy interesante ya que esta inicialización puede tardar unos 5 sg. (es mucho menos de lo que tardaríamos en levantar un JBoss y hacer el despliegue real de la aplicación, pero puede resultar demasiado para test unitarios), así que hacerla antes de cada test puede no resultar útil.

Son muy importantes las líneas dentro del if que salen en negrita. Estas dos líneas son necesarias para que el test funcione desde Maven. No serían necesarias si, por ejemplo, lanzáramos el test desde Eclipse o desde ant; pero tampoco afecta negativamente en estos casos, así que podemos dejarlas siempre.

Esto es un bug (http://jira.codehaus.org/browse/SUREFIRE-104) documentado de Surefire (el plugin de Maven que se encarga de lanzar los test).

```
@AfterClass
public static void tearDownAfterAllTest() throws Exception {
        undeploy();
   if (System.getProperty("shutdown.embedded.jboss") != null) {
            Bootstrap.getInstance().shutdown();
        }
    }
```

El método recíproco a `setUpBeforeAllTest`. Se encarga de para el contenedor.

```
private String sayHi(String ejbName) throws Exception {
   final InitialContext ctx = new InitialContext();
   final Greeter greeter = (Greeter)ctx.lookup(ejbName);
   return greeter.sayHi();
```

```
    }

  private void addGreeting(String ejbName, String message) throws Exception {
    final InitialContext ctx = new InitialContext();
    final Greeter greeter = (Greeter)ctx.lookup(ejbName);
       greeter.addGreeting(message);
    }
```

Nos hacemos un par de método genéricos para probar los EJBs. No son métodos de test de JUnit (no están anotados con `@Test`), simplemente es para reutilizar código, los llamaremos desde los test.

Los métodos son muy sencillos, simplemente se le pasa el nombre del EJB, este se busca por JNDI y se llama a algún método de negocio.

```
  @Test
  public void smartGreeterBean_sayHi() throws Exception {
    final String greeting = sayHi("SmartGreeterBean/local");
    assertEquals(Greeting.DEFAULT_MESSAGE, greeting);
    }

  @Test(expected = InvalidGreetingMessageException.class)
  public void politeSmartGreeterBean_addUnpoliteGreting() throws Exception {
    addGreeting("PoliteSmartGreeterBean/local", GREETING_MESSAGES.get(3));
    }
```

Estos son ejemplos de métodos de test de JUnit que usan los métodos anteriores, aunque en el código podéis encontrar algunos más.

```
  @Test
  public void checkGreetingsInDataBase() throws Exception {
    final EntityManager em = (EntityManager)new InitialContext().lookup("java:/EntityManagers/test-autentia");

    final List<Greeting> greetings = em.createQuery("from Greeting").getResultList();

    int greetingsInDataBase = 0;
    for (Greeting greeting : greetings) {
      assertTrue(GREETING_MESSAGES.contains(greeting.getMessage()));
         greetingsInDataBase++;
      }

    // Tiene que haber tantos como el tamaño de la lista - 1 porque uno no es educado
    // y el PoliteSmartGreeterBean se ha tenido que negar a añadirlo.
    assertEquals(GREETING_MESSAGES.size() - 1, greetingsInDataBase);
    }
```

Otro ejemplo de como hacer un test que accede directamente al `EntityManager`. Fijaros como este se localiza por JNDI, y luego se utiliza con normalidad.

8. El fichero de configuración de la persistencia test-persistence.xml

Este fichero lo hemos creado nosotros y lo hemos situado en **src/test/resources** (junto al resto de ficheros de configuración que habíamos copiado de Embedded JBoss).

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence version="1.0"
      xmlns="http://java.sun.com/xml/ns/persistence"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">

   <persistence-unit name="test-autentia">
      <!-- Defines the JPA provider. For JBoss you should use Hibernate. -->
      <provider>org.hibernate.ejb.HibernatePersistence</provider>

      <jta-data-source>java:/DefaultDS</jta-data-source>

      <!-- As Hibernte is your JPA provider, you can set some Hibernate properties -->
      <properties>
         <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
         <property name="jboss.entity.manager.jndi.name" value="java:/EntityManagers/test-autentia"/>
      </properties>
   </persistence-unit>
</persistence>
```

Cosas interesantes de este fichero:

- El DataSource que estamos usando **java:/DefaultDs**, es un DataSource que ya viene configurado en Embedded JBoss. Este DataSource usa como base de datos HSQLDB (Hypersonic SQL DataBase http://hsqldb.sourceforge.net/). Esta es una base de datos relacional SQL escrita en Java, muy ligera y muy rápida (prácticamente todo lo hace en memoria).

  Con este DataSource no tenemos porque instalar una base de datos adicional para ejecutar los test. Y además tiene la ventaja de que como sólo vamos a trabajar en memoria, cuando acaben los test se borrará todo y tendremos un entorno limpio para la siguiente ejecución de los test. Si necesitamos datos tendremos que tener algún test que se ejecute al principio para poblar la base

de datos; con la ventaja de que podemos tener distintos casos de test que pueblen la base de datos de distintas maneras.

- La propiedad **hibernate.hbm2ddl.auto=create-drop**. Con esto le estamos diciendo a Hibernate que, cuando arranque el contenedor, se encargue de crear las tablas necesarias en la base de datos para poder soportar nuestros EJBs de entidad (cuando se pare; el contenedor también se encarga de borrar las tablas). Así que no nos tendremos que preocupar de la creación de las tablas para pasar nuestros test !!!

  Esta funcionalidad puede resultar muy útil en nuestro entorno de pruebas, pero no se debería usar en un entorno de producción. En producción deberíamos llegar a un acuerdo con el DBA y que sea él quien nos diga donde como y cuando se deben crear las tablas para sacar el mejor rendimiento al sistema.

- La propiedad **jboss.entity.manager.jndi.name**. Por defecto JBoss no exporta el `EntityManager` por JNDI. Con esta propiedad le estamos diciendo que lo exporte con el nombre que le damos en el atributo `value`. Esto lo hacemos para poder localizar el `EntityManager` directamente en nuestros test (hemos visto un ejemplo justo al final del apartado anterior).

9. Resumiendo

Parece que hemos hecho muchas cosas, pero la verdad es que casi todo lo ha hecho Embedded JBoss. Eso es lo bueno del asunto !!!

¿Qué hemos hecho nosotros realmente?

- Simular un jar con la clase `AssembledDirectory`, donde hemos metido nuestras clases y fichero de configuración de la persistencia.
- Hacer el despliegue del jar en la clase `Bootstrap`. ¡¡¡ Ojo que no se nos olvide poner las dos líneas para "solucionar" el problema del Surefire !!!
- Hacer nuestros test de EJBs :)

10. Conclusiones

JPA está pensado para poder ser usado fuera de servidores de aplicaciones, y los EJBs 3.0 no dejan de ser POJOs, así que podríamos haber optado por otras estrategias para probar estas clases. Lo bueno de usar JBoss Embedded es que nos facilita mucho el trabajo y estamos probando algo muy parecido a lo que será cuando hagamos el despliegue real de la aplicación.

Si bien habría que decir que este tipo de test no se pueden considerar realmente test unitarios. Una definición que podemos encontrar (de gurús en estos temas como Michael Feathers y Alberto Savoia) de test unitario es:

- Si accede a la base de datos, no es un test unitario.
- Si invoca el sistema de ficheros o trabaja con el sistema de ficheros, no es un test unitario.
- Si necesita alguna configuración especial de tu entorno, no es un test unitario.
- Y, si no lo puedes ejecutar mientras se ejecutan otros test, no es un test unitario.

Nuestros tests de EJBs incumplen todos los puntos, así que no se pueden considerar estrictamente tests unitarios. Tendría más sentido decir que son tests de integración. Lo que está claro, lo llamemos como lo llamemos, es que son tests, y que son más que convenientes.

Lo que si podríamos hacer, teniendo en cuenta que no son estrictamente tests unitarios, es sacar este tipo de tests a un proyecto de Maven diferente, de forma que, dentro de un proyecto tengamos tests unitarios, estrictamente hablando, de ese proyecto; y en un proyecto diferente los tests de integración, que además suelen consumir más tiempo. De esta forma ejecutaremos muchas veces los unitarios (casi continuamente), y no habrá problema porque serán muy rápidos. Y los de integración lo haremos menos veces (3, 4 veces al día).

Espero que os haya sido de ayuda, y recordar: tests, tests, tests, tests !!!

11. Sobre el autor

Alejandro Pérez García, Ingeniero en Informática (especialidad de Ingeniería del Software)

Socio fundador de Autentia (Formación, Consultoría, Desarrollo de sistemas transaccionales)

mailto:alejandropg@autentia.com

Autentia Real Business Solutions S.L. - "Soporte a Desarrollo"

http://www.autentia.com

Puedes opinar sobre este tutorial aquí

Recuerda

que el personal de Autentia te regala la mayoría del conocimiento aquí compartido (Ver todos los tutoriales)

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?

¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

info@autentia.com

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos .....
Autentia = Soporte a Desarrollo & Formación

Formación en nuevas tecnologías

Autentia S.L. Somos expertos en:
J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ..
y muchas otras cosas

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
EJB 3.0: Resurrection	Este turorial nos va a presentar las nuevas funcionalidades que nos aportan los EJB 3.0.
Guía rápida de instalación de JBOSS Application Server 4.	En este manual veremos paso a paso la forma de instalar en tu equipo JBoss Application Server 4.
Maven, nunca antes resultó tan fácil compilar, empaquetar, ...	En este tutorial aprenderemos el uso de esta herramienta que nos permite compilar, empaquetar, generar documentación, pasar los test, preparar las builds de nuestros proyectos
EJB´s y Orion	Recreación de la guía paso a paso de como crear una aplicación Web con EJB´s y Servlets y su despliegue con ANT sobre Orion
Interceptando un EJB en JBoss	En este tutorial os vamos a enseñar la arquitectura de EJBs en JBoss y a como modificarla, insertando un interceptor propio dentro de la cadena de interceptores del Proxy Cliente.
Anotaciones en EJB 3.0	Este tuotiral nos va a enseñar algunas características del API de EJB 3.0 y las mejoras introducidas en la nueva version 3.0
Instalar JBoss	Os mostramos como instalar en servidor gratuito de aplicaciones JBOSS así como a automatizar su arranque y parada.
Hibernate y las anotaciones de EJB 3.0	En este tutorial Alejandro Pérez nos muestra las ventajas que nos aporta Hibernate y las anotaciones de EJB 3.0
Test con JUnit	Cuando se hacen desarrollo profesionales, no basta con hacer los programas, hay que asegurarse de que van a funcionar. Una de las técnicas más seguras es crear aplicaciones que incluyan el código para autoprobarse. Os mostramos como usar JUnit
Planificar tareas en JBoss	En este tutorial os enseñaremos a planificar tareas periódicas con JBoss

solicitarlo.

Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador rcanales@adictosaltrabajo.com para su resolución.