Servidores Web

Sesión 4: Introducción a Jboss. Servicios en JavaEE

Puntos a tratar

- Introducción a JBoss
  - Configuración
  - Despliegue de aplicaciones y servicios
- Servicios en JavaEE
  - JNDI y fuentes de datos
  - ... en Tomcat
  - ... en JBoss

JBoss

- Es un **servidor de aplicaciones** (*a diferencia de Tomcat*)
  - Además de aplicaciones web, incluye todos los servicios requeridos por el estándar JavaEE
  - Tomcat por ejemplo, no tiene EJBs
- La versión “community”es libre (jboss.org)
  - Con la versión de pago, se ofrece soporte y herramientas adicionales
- Instalación
  - Descomprimir
  - Definir la var.entorno JBOSS_HOME con el dir. de instalación
  - Ejecutar *run.bat* Probar **http://localhost:8080**

Las consolas de JBoss

- Consola JMX
  - JMX es un estándar Java que permite monitorizar/manipular componentes remotos
  - Cada elemento de JBoss es un *bean* JMX
- Consola web
  - Applet para monitorizar aplicaciones
  - Estadísticas de uso, alertas (p.ej. si la memoria baja de un cierto nivel, etc)

Configuración de JBoss

- Estructura física (directorios)
  - **conf**: servicios fijos durante toda la vida del servidor.
  - **data**: para servicios que quieran almacenar datos de manera permanente.
  - **deploy**: para desplegar aplicaciones y servicios.
  - **lib**: librerías comunes al servidor y a las aplicaciones. Por ejemplo, drivers JDBC.
  - **log**
  - **tmp** y **work**: ídem a Tomcat
- La versión *community* no tiene herramientas gráficas de configuración
  - Editar archivos XML. Por ejemplo **conf/jboss-service.xml**. Afortunadamente, están ampliamente comentados y hay mucha documentación libre

Despliegue

- De aplicaciones web
  - JBoss “lleva un Tomcat dentro”
  - El dir. que “hace de webapps” aquí se llama “**deploy**”. Basta con dejar caer un .war en él. Por defecto JBoss no lo descomprime aquí
  - En Eclipse, basta definirnos el servidor como Jboss 4.2
- De servicios
  - Se coloca un XML con la config. en “deploy”.
  - Servicios como: mail, scheduler, ...

Servicios JavaEE

- Accesibles a nuestras aplicaciones y que debe ofrecer un servidor “compatible JavaEE”
- **DataSource**
  - Conexión con B.D. con facilidades adicionales, como **pooling**
  - La conexión la abre el servidor, no nosotros directamente. Debemos dejarle accesible el *driver*
- **JNDI** *(Java Naming & Directory Interface)*
  - Posibilidad de **localizar recursos** (*beans*, conexiones con BD, ...) **mediante un nombre lógico**. Si cambia la localización física del recurso, nuestro código no se ve afectado.

Pooling de conexiones

- Abrir una conexión con la B.D. es costoso en tiempo

  Connection = DriverManager.getConnection(...);

  - Si cada operación de B.D. Implica abrir una nueva conexión (que luego hay que cerrar) y hay muchas peticiones simultáneas, el coste se multiplica
- Solución: **pool** de conexiones
  - Conjunto de conexiones que el servidor abre durante el arranque y siempre mantiene abiertas
  - Al solicitar una conexión, el servidor nos da una libre
  - Cuando la devolvemos al servidor, no se cierra, simplemente se marca como libre

DataSource

- El servidor nos da las conexiones con la B.D a través de la clase DataSource

```
DataSource ds;
...
Connection con = ds.getConnection();
```

- Suele ofrecer *pooling* automático y otras ventajas con respecto a usar directamente el *driver*
  - Como el *driver* lo gestiona el servidor, hay que dejárselo accesible. En Tomcat y JBoss basta con dejarlo en la carpeta **lib** y rearrancar el servidor

Configurar el DataSource: info necesaria

- En general se configura en un fichero XML
- Información:
  - Nombre lógico para el DataSource (luego será parte del nombre JNDI)
  - URL de la BD
  - Clase que implementa el driver
  - Usuario y pasword para acceder a la BD
  - Tamaño inicial y máximo del pool

Configurar el DataSource en JBoss

- Dejar el fich. con la configuración en "**deploy**". El nombre debe acabar en "**-ds.xml**"

(Fichero deploy/mysql-prueba-ds.xml)

```
<datasources>
  <local-tx-datasource>
    <jndi-name>PruebaDS</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/prueba</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>prueba</user-name>
    <password>prueba</password>
    <exception-sorter-class-name>
      org.jboss.resource.adapter.jdbc.
          vendor.MySQLExceptionSorter
    </exception-sorter-class-name>
  </local-tx-datasource>
</datasources>
```

Acceder al DataSource en nuestro código

- El nombre JNDI suele llevar un “prefijo”
  - Por ejemplo, al desplegar el fichero anterior, en la terminal aparece:

```
Bound ConnectionManager
'jboss.jca:service=DataSourceBinding,name=PruebaDS'
 to JNDI name 'java:PruebaDS'
```

- API para obtener la conexión : **JNDI** + **JDBC**

```
//Obtener el contexto JNDI
Context initCtx = new InitialContext();
//Obtener el recurso con su nombre lógico (JNDI)
DataSource ds = (DataSource) initCtx.lookup("java:PruebaDS");
//A través del DataSource podemos obtener una conexión con la BD
Connection conn = ds.getConnection();
//A partir de aquí trabajaríamos como es habitual en JDBC
...
```

Configurar el DataSource en Tomcat

- Se coloca en **META-INF/context.xml**
  - Por supuesto, cambia el formato XML.

```
<Context>
   <Resource
      name="PruebaDS"
      type="javax.sql.DataSource"
      auth="Container"
      username="prueba"
      password="prueba"
      driverClassName="com.mysql.jdbc.Driver"
      url="jdbc:mysql://localhost:3306/prueba"
      maxActive="20"          <!-- tamaño máximo del pool -->
      maxIdle="5"             <!-- si hay más de 5 libres, se cierran de verdad -->
      maxWait="10000"/>       <!-- tiempo máximo espera a la BD. Si más, excepción -->
</Context>
```

Acceder al DataSource en nuestro código

- En principio el código es el mismo, la única diferencia es el “prefijo” JNDI, que en tomcat es **java:comp/env** en lugar de **java:**

```
//Obtener el contexto JNDI
Context initCtx = new InitialContext();
//Obtener el recurso con su nombre lógico (JNDI)
DataSource ds = (DataSource) initCtx.lookup("java:comp/env/PruebaDS");
//A través del DataSource podemos obtener una conexión con la BD
Connection conn = ds.getConnection();
//A partir de aquí trabajaríamos como es habitual en JDBC
...
```