Apéndice C

Controladores JDBC

1. Introducción

Java es un lenguaje de programación que ofrece muchas ventajas a los desarrolladores de aplicaciones. Entre ellas se encuentra el acceso a bases de datos a través de JDBC [Java DataBase Connectivity].

JDBC permite escribir código independiente del DBMS que se vaya a emplear (siempre que acepte consultas SQL) de una forma rápida y eficaz. El API de JDBC se creó a partir del X/Open SQL CLI [Call Level Interface], el cual también es la base del ODBC de Microsoft.

El empleo de ODBC no es adecuado en Java ya que es una interfaz escrita en C y las llamadas desde Java a código nativo en C tienen algunas desventajas (carecen de la seguridad y de la portabilidad características de Java). Por lo tanto, los creadores del API de JDBC crearon una interfaz simple y fácil de usar que con facilidad puede implementarse sobre ODBC cuando sea necesario (v.g. el controlador de Oracle Lite).

Básicamente, un controlador JDBC se limita a pasar cadenas de caracteres (sentencias SQL que pueden no funcionar igual en todos los DBMSs) al controlador del DBMS en uso. Será responsabilidad del DBMS analizar sintáctica y semánticamente la orden SQL. Una aplicación puede, por lo tanto, utilizar características específicas de un DBMS concreto (sabiendo que se pierde la portabilidad). En realidad, las consultas realizadas a través de un controlador JDBC no tendrían por qué ser consultas SQL (podría emplearse algún lenguaje específico para determinado tipo de bases de datos).

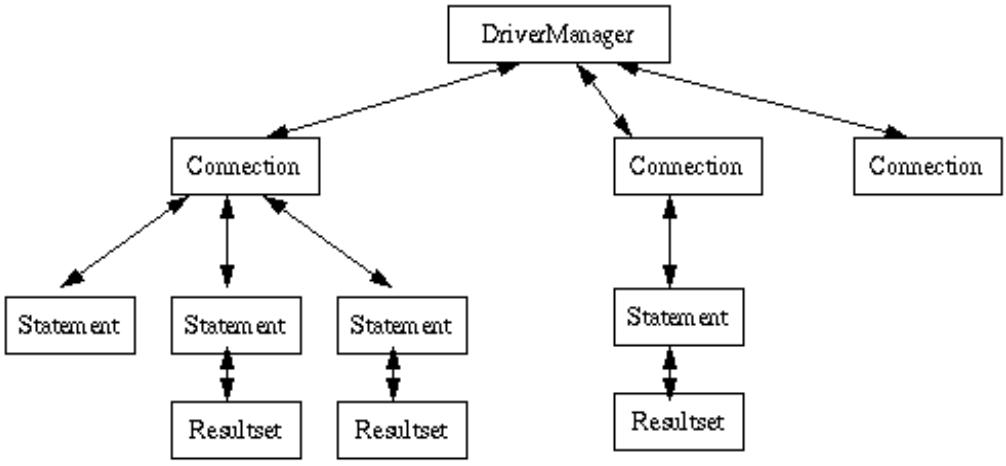
Sin duda, la característica más importante del API de JDBC es su simplicidad. Por ejemplo, para ofrecer diversas funciones se optó por definir distintos métodos en vez de utilizar sólo unos pocos métodos con muchos parámetros. Ésta es también la filosofía del resto de las clases estándar de Java. Esta decisión de diseño provoca la existencia de una gran cantidad de métodos diferentes pero permite que cada uno de ellos sea más fácil de comprender por parte del programador de aplicaciones.

2. Arquitectura de JDBC

Un controlador JDBC suele tener dos interfaces, uno para los programadores de aplicaciones en Java y otro, de más bajo nivel, para los controladores JDBC.

2.1 La interfaz de JDBC para el programador de aplicaciones

El API de JDBC para los programadores consiste en una serie de interfaces Java que permiten establecer conexiones con bases de datos particulares, ejecutar sentencias SQL y procesar los resultados.



Las interfaces más importantes para el programador de aplicaciones son los siguientes:

- ***java.sql.DriverManager*** permite cargar los controladores JDBC específicos para cada DBMS y establecer conexiones con bases de datos.

- ***java.sql.Connection*** modeliza una conexión con una base de datos.

- ***java.sql.Statement*** permite ejecutar sentencias SQL dada una conexión ya establecida. Esta interfaz tiene dos subtipos bastante importantes: ***java.sql.PreparedStatement*** para ejecutar sentencias SQL precompiladas y ***java.sql.CallableStatement*** para llamar a procedimientos almacenados en la base de datos [*database stored procedures*].

- ***java.sql.ResultSet*** permite acceder al resultado de una consulta SQL. En las siguientes tablas se resume el manejo de tipos realizado por los controladores JDBC:

		SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
getByte	**X**	x	x	x	x	x	x	x	x	x	x	x	x						
getShort	x	**X**	x	x	x	x	x	x	x	x	x	x	x						
getInt	x	x	**X**	x	x	x	x	x	x	x	x	x	x						
getLong	x	x	x	**X**	x	x	x	x	x	x	x	x	x						
getFloat	x	x	x	x	**X**	x	x	x	x	x	x	x	x						
getDouble	x	x	x	x	x	**X**	**X**	x	x	x	x	x	x						
getBigDecimal	x	x	x	x	x	x	x	**X**	**X**	x	x	x	x						
getBoolean	x	x	x	x	x	x	x	x	x	**X**	x	x	x						
getString	x	x	x	x	x	x	x	x	x	x	**X**	**X**	x	x	x	x	x	x	x
getBytes														**X**	**X**	x			
getDate											x	x	x				**X**		x
getTime											x	x	x					**X**	x
getTimestamp											x	x	x				x		**X**
getAsciiStream											x	x	**X**	x	x	x			
getUnicodeStream											x	x	**X**	x	x	x			
getBinaryStream														x	x	**X**			
getObject	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Table 1: Use of ResultSet.getXXX methods to retrieve common SQL data types
An "x" means that the given getXXX method can be used to retrieve the given SQL type.
An "**X**" means that the given getXXX method is recommended for retrieving the given SQL type

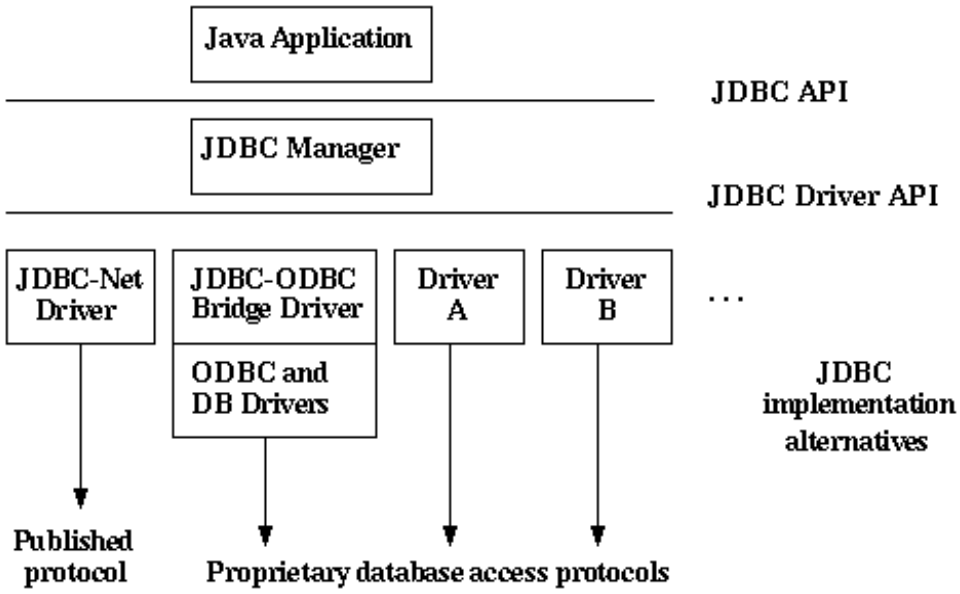
SQL	***Java***	***SQL***	***Java***
CHAR	String	REAL	float
VARCHAR	String	FLOAT	double
LONGVARCHAR	String	DOUBLE	double
NUMERIC	java.math.BigDecimal	BINARY	byte[]
DECIMAL	java.math.BigDecimal	VARBINARY	byte[]
BIT	boolean	LONGVARBINARY	byte[]
TINYINT	byte	DATE	java.sql.Date
SMALLINT	short	TIME	java.sql.Time
INTEGER	int	TIMESTAMP	java.sql.Timestamp
BIGINT	long		

Table 2: Standard mapping from SQL types to Java types

2.2 La interfaz JDBC para los controladores JDBC

Los controladores JDBC simplemente han de implementar las clases que necesitan los programadores de aplicaciones (ver sección anterior): *java.sql.Connection*, *java.sql.Statement*, *java.sql.PreparedStatement*, *java.sql.CallableStatement* y *java.sql.ResultSet*.

Además, cada controlador debe suministrar una implementación de ***java.sql.Driver***, un interfaz Java utilizado por la clase *java.sql.DriverManager* cuando necesita localizar un controlador adecuado para una URL particular.



JavaSoft proporciona una implementación de JDBC sobre ODBC (el puente JDBC-ODBC mostrado en la figura), útil cuando no se dispone de un controlador JDBC pero sí de uno ODBC. Otro controlador interesante es el que permite el empleo de otros protocolos independientes del DBMS (el controlador JDBC-Net de la figura). Finalmente, cada empresa propietaria de un DBMS particular suele ofrecer controladores JDBC específicos para sus productos (controladores A y B de la figura).

3. Implementación de controladores JDBC

El único secreto de la implementación de un controlador JDBC reside en crear un objeto de la clase *Driver* estáticamente y registrar el controlador (llamando al método *registerDriver* de la clase *java.sql.DriverManager*). El controlador debe especificar el prefijo de las URLs de las que se hará cargo (el controlador será responsable de las peticiones dirigidas a dichas URLs). A continuación se ofrece una implementación posible de la clase *Driver* de un controlador JDBC que se hará cargo de las URLs de la forma *"**jdbc:db3**..."*:

```
import java.sql.*;

public class DB3Driver implements java.sql.Driver
{
 // Constructor: Registro del driver

 public DB3Driver ()
        throws SQLException
 {
   DriverManager.registerDriver(this);
 }

 // Inicialización: Creación de un objeto de la clase DB3Driver

 static
 {
   DB3Driver driver = new DB3Driver();
 }

 // Establecimiento de una conexión (método llamado por DriverManager)

 public Connection connect (String url, java.util.Properties info)
                   throws SQLException
 {
   if (acceptsURL(url)) {
      String ruta = url.substring(9);
      return new DB3Connection(ruta);
   } else {
      return null;
   }
 }

 // ¿Dirección a cargo del driver? (método llamado por DriverManager)

 public boolean acceptsURL(String url)
                throws SQLException
 {
   return url.startsWith("jdbc:db3");
 }

 // Otros métodos recogidos en java.sql.Driver
 // ...
}
```

Una vez que ya tenemos registrado nuestro propio controlador JDBC para responder a las solicitudes de conexión dirigidas a las direcciones *"**jdbc:db3**..."*, hemos de implementar todas y cada una de las clases que definen la interfaz JDBC para el programador de aplicaciones, incluyendo los cientos de métodos recogidos en el estándar. Las siete clases que han de implementarse son las siguientes:

```
public class DB3CallableStatement implements java.sql.CallableStatement
{
 // ...
}

public class DB3Connection implements java.sql.Connection
{
 // ...
}

public class DB3DatabaseMetaData implements java.sql.DatabaseMetaData
{
 // ...
}

public class DB3PreparedStatement implements java.sql.PreparedStatement
{
 // ...
}

public class DB3ResultSet implements java.sql.ResultSet
{
 // ...
}

public class DB3ResultSetMetaData implements java.sql.ResultSetMetaData
{
 // ...
}

public class DB3Statement implements java.sql.Statement
{
 // ...
}
```

Lo normal es que estas clases realicen llamadas a alguna DLL del DBMS o establezcan una conexión TCP/IP con algún servidor. No obstante, también es posible que implementen el propio DBMS en Java (tal como se ha realizado en el controlador JDBC para DB3, un minisistema gestor de bases de datos semi-relacionales) y se encarguen por completo del análisis y ejecución de las sentencias SQL.

De hecho, el controlador implementado para DB3 realiza un simple análisis sintáctico predictivo de una orden MiniSQL (el controlador no acepta cualquier sentencia SQL) y accede a las tablas almacenadas en el directorio especificado en la URL de la conexión tras el prefijo *"jdbc:db3:"*.

También se ha desarrollado completamente en Java un pequeño servidor al que se puede acceder a través de un controlador JDBC. Dicho servidor, al que se le ha denominado NanoServer por su pequeño tamaño, accede en el servidor a ficheros almacenados en formato ASCII (tipo C4.5) y permite el mismo tipo de consultas que DB3 (el cual utiliza ficheros con un formato bastante más complejo y potente).

En el caso de DB3, el controlador JDBC accede directamente a los ficheros almacenados en la máquina local, por lo que no puede emplearse en la distribución de applets. Sin embargo, NanoServer se instala en una máquina y, dado que la comunicación entre el cliente JDBC y el servidor se realiza completamente a través de sockets, el controlador JDBC suministrado puede utilizarse en cualquier máquina remota. Además, al estar completamente escrito en Java y no acceder a ningún tipo de DLL, puede emplearse en el desarrollo de applets.

Para establecer una conexión a *NanoServer* ha de especificarse una URL de la forma ***"jdbc:fbg@***<*host*>***:***<*port*>***:***<*db*>***"***, donde <*host*> es el nombre de la máquina en la que se ejecuta el servidor (localizada a través de DNS), <*port*> es el puerto TCP a través del cual se establecerá la conexión y <*db*> es el nombre de la base de datos. En la siguiente figura se muestra cómo habría de establecerse una conexión con un nanoservidor:

4. Guía rápida de acceso a bases de datos a través de JDBC

A continuación se ofrece una tabla resumen que recopila la información necesaria para establecer una conexión JDBC con distintos tipos de servidores:

Oracle Call Interface	`Driver  oracle.jdbc.driver.OracleDriver`
	`URL  jdbc:oracle:oci(7|8):@<host>`
	
	☞ Requiere encontrar CLASSES102.ZIP o CLASSES111.ZIP en el CLASSPATH (según se utilice JDK 1.0 o superior).
	
	☞ Además, hace uso de OCI803JDBC.DLL (bajo Windows), que debe encontrarse en el PATH del sistema.

Oracle Thin	`Driver  oracle.jdbc.driver.OracleDriver`
	`URL  jdbc:oracle:thin:@<host>:<port>:<db>`
	
	☞ Requiere encontrar CLASSES102.ZIP o CLASSES111.ZIP en el CLASSPATH (según se utilice JDK 1.0 o superior).
	
	☞ Se puede emplear en applets, al estar completamente implementado en Java.

Personal Oracle Lite	`Driver  oracle.pol.poljdbc.POLJDBCDriver`
	`URL  jdbc:POLite:<db>`
	
	☞ POLJDBC.JAR ha de colocarse en el CLASSPATH.
	
	☞ No permite ser utilizado en applets.

IBM DB2	`Driver  COM.ibm.db2.jdbc.app.DB2Driver`
	`URL  jdbc:db2:<db>`
	
	☞ El path del sistema debe incluir el directorio SQLLIB/BIN de la instalación de DB2 Universal Database.
	
	☞ No puede utilizarse en applets.

IBM DB2 for applets	`Driver  COM.ibm.db2.jdbc.net.DB2Driver`
	`URL  jdbc:db2://<servidor>:<puerto>/<db>`
	
	☞ Permite acceder remotamente a cualquier servidor DB2, siempre que éste admita conexiones a través del puerto indicado, lo que se puede conseguir ejecutando `db2jstrt <puerto>`.
	
	☞ Puede emplearse en aplicaciones y applets.

ODBC Bridge	`Driver  sun.jdbc.odbc.JdbcOdbcDriver`
	`URL  jdbc:odbc:<dsn>`
	
	☞ Este controlador distribuido por Sun permite el acceso a bases de datos a través de cualquier controlador ODBC.
	
	☞ Obviamente, no se puede emplear en applets.

FBG Nano Server	`Driver  fbg.jdbc.client.ClientDriver`
	`URL  jdbc:fbg@<host>:<port>:<db>`
	
	☞ Permite el acceso remoto a cualquier servidor utilizando un protocolo muy simple a través de sockets TCP/IP.
	
	☞ Puede ser empleado en aplicaciones y applets.

FBG DB3	`Driver  fbg.jdbc.DB3Driver`
	`URL  jdbc:db3@<dir>`
	
	☞ Permite acceder a ficheros locales almacenados en un formato propio que permite el almacenamiento de textos completos. Dichos ficheros han de encontrarse en el subdirectorio `<dir>`.
	
	☞ Al acceder directamente a ficheros, no puede utilizarse en applets (aunque sí en aplicaciones).

5. Referencias

Graham Hamilton & Rick Cattell
"JDBC™: A Java SQL"
Version 1.20, February 9, 1998

Especificación de la versión 1.2 del estándar JDBC [Java DataBase Connectivity], el interfaz de Java con bases de datos SQL [Structured Query Language].

"Oracle JDBC Drivers"
Oracle Corporation, 1997

Documentación ofrecida con los controladores JDBC desarrollados por Oracle. En ella se especifican qué características estándar se han implementado y cuáles no. Así mismo, se describen algunas peculiaridades añadidas en los controladores Oracle al estándar JDBC y su uso en aplicaciones y applets en Java.