Accede al documento original de Wuolah

ACCESO-A-DATOS-TEMA-4.pdf

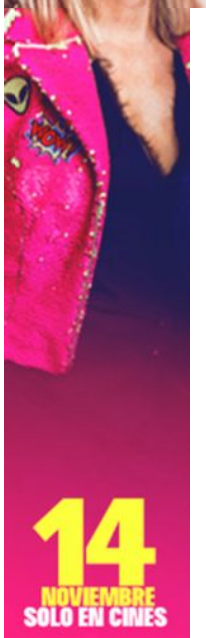
user_4383848

Acceso a datos

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.



14 NOVIEMBRE SOLO EN CINES
TODOS LOS LADOS DE LA CAMA

ACCESO A DATOS - TEMA 4
MANEJO DE CONECTORES

1. INTRODUCCIÓN AL MANEJO DE CONECTORES:

Definimos **conector** como una serie de clases y librerías que realizan la labor de unir la capa de nuestra aplicación con la capa de base de datos.

Desfase Objeto Relacional:
Desfase objeto-relacional → Las BD tienen naturaleza distinta al aplicativo que se trabaja con POO y ésto provoca discrepancias.
Aspectos importantes del desfase:
→ Hay bastante **diferencia entre los datos** que se usan en las bases de datos relacionales y en la programación orientada a objetos.
→ Implica realizar distintos diagramas, ya que vamos a tener que **realizar una traducción** desde los objetos del aplicativo Java a la base de datos relacional, por lo tanto, se crearán entidades en ambos sitios. Entidades distintas, aunque representen la misma unidad.

Protocolos de Acceso a Base de Datos:
Un conector **o driver** es una serie de clases implementadas (API) que facilitan la conexión a la base de datos asociada.
→ **JDBC** (Java Database Connectivity): Si usamos el conector JDBC no tendríamos que desarrollar un aplicativo para acceder a una base de datos Oracle y otro driver para acceder a una base de datos distinta, sino que nosotros desarrollaremos nuestra aplicación y a la hora de realizar cualquier consulta, el conector interpretaría de una forma u otra dependiendo de la base de datos asociada.
→ **ODBC** (Open Database Connectivity).
→ Una aplicación **debe tener asociado siempre un conector**. No tenemos que conocer los aspectos técnicos, ni cómo funcionan en su interior dichas bases de datos. Nos ocuparemos de cómo realizar la comunicación y de cómo funcione nuestro aplicativo.
→ **Información de una base de datos:** Simplemente utilizando dicha librería JDBC e indicando las configuraciones de acceso a cada una de ellas, tendremos acceso a las mismas

2. CONEXIONES: COMPONENTES Y TIPOS:

Componentes:

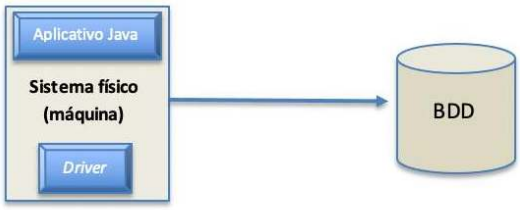
- **Api de JDBC:** Tenemos una serie de librerías y clases que nos facilitan el acceso a las BD. → Paquetes java.sql y javax.sql.
- **Paquete de pruebas JDBC:** Se encargan de validar si un *driver* pasa los requisitos previstos por JDBC.
- **Gestor JDBC:** encargado de realizar la unión entre nuestra aplicación Java con el driver apropiado JDBC → Conexión directa o a través de un pool de conexiones.
- **El puente JDBC-ODBC:** Facilita el uso de los drivers ODBC como si trabajásemos con JDBC.

WUOLAH

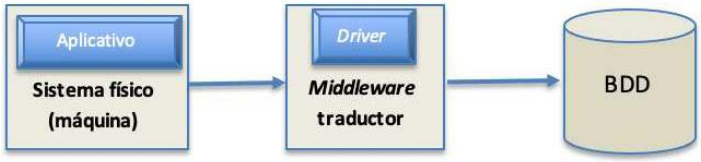
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Arquitectura:

- **En dos capas:**Nuestra aplicación se conectará a la BDD a través de un *driver*. Tanto el *driver* como la aplicación deben localizarse en el mismo sistema o máquina.



- **En tres capas:** Nuestro aplicativo enviará las instrucciones a una capa intermedia (*middleware*). Esta capa cogerá la información y la enviará a la base de datos correspondiente traduciendo los comandos que el aplicativo haya enviado.



Tipos:

- **Driver tipo 1 JDBC-ODBC:** Este *driver* usa una API nativa, traduce las llamadas realizadas de JDBC a ODBC. Los datos devueltos por la base de datos se traducirán a JDBC cuando sean devueltos.
- **Driver tipo 2 JDBC Nativo:** Estos *drivers* están escritos una parte en Java y otra parte, en código nativo. Las llamadas al API JDBC son traducidas en llamadas propias de la base de datos relacional que tengamos.
- **Driver tipo 3 JDBC net:** Es un *driver* de tres capas cuyas solicitudes JDBC están siendo traducidas en un protocolo de red en una capa intermedia o *middleware*. Esta capa intermedia recibirá dichas solicitudes y las enviará a la base de datos usando un *driver* JDBC de tipo 1 o de tipo 2. Cabe destacar que es una arquitectura muy flexible.
- **Driver tipo 4 protocolo nativo:**Este tipo de *driver* realiza las llamadas mediante el servidor, usando el protocolo nativo del mismo.

3. CONFIGURACIÓN DE UNA CONEXIÓN EN CÓDIGO:

```
private static final String DRIVER = "org.mysql.jdbc.Driver";
private static final String URL_CONEXION = "jdbc:mysql://localhost:3306/Pruebas";
Fuente http://decodigo.com/java-conexion-a-base-de-datos-con-jdbc
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga carpetas completas de una vez con el Plan PRO y PRO+

```
public static void main(String args[]) throws SQLException {

    final String usuario = "user_db";
    final String password = "password_db";
    Connection dbConnection = null;
    Statement statement = null;
Fuente: http://decodigo.com/java-conexion-a-base-de-datos-con-jdbc
```

- Variables de tipo ***String*** que nos van a servir para realizar la conexión con la base de datos más tarde.
- Instanciamos el **usuario y la contraseña de nuestra conexión** y también una variable de tipo Connection y otra Statement.
- **Connection** es una interfaz que representa una conexión directa con una base de datos. El motivo de que sea una interfaz es porque tendrá distintas implementaciones posibles.
- Estableceremos la conexión con **"java.sql.DriverManager"**, recomendada para aquellos aplicativos que se hayan desarrollado en lenguaje Java.

Establecer conexión:

```
try {
    Class.forName(DRIVER);
    dbConnection = DriverManager.getConnection(URL_CONEXION, usuario, password);
    String selectTableSQL = "SELECT ID,USERNAME,PASSWORD,NOMBRE FROM Usuarios";
    statement = dbConnection.createStatement();
    ResultSet rs = statement.executeQuery(selectTableSQL);
    while (rs.next()) {
        String id = rs.getString("ID");
        String usr = rs.getString("USERNAME");
        String psw = rs.getString("PASSWORD");
        String nombre = rs.getString("NOMBRE");
        System.out.println("userid : " + id);
        System.out.println("usr : " + usr);
        System.out.println("psw : " + psw);
        System.out.println("nombre : " + nombre);
    }
```

- **DriverManager:** Gestiona los *drivers* que poseemos en nuestra aplicación y permite en una misma capa el acceso a todos y cada uno de ellos.
- **getConnection:** Establece conexiones. (método)
- **Class.forName():** de esta forma registramos el *driver* que anteriormente hemos indicado en la variable estática "DRIVER"
- **dbConnection:** Todo esto nos devolverá un objeto de tipo *Connection*, en nuestro caso lo hemos llamado *dbConnection*.

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga carpetas completas de una vez con el Plan PRO y PRO+

Operaciones con variables y excepciones:

- **DriverManager:** una vez que *DriverManager* nos ha devuelto la conexión a base de datos, realizaremos un ejemplo sencillo de consulta simple y la almacenaremos en una variable de tipo *String* para más tarde ser ejecutada.
- **createStatement:** Con la variable *Connection*, ejecutamos el método "*createStatement*" y lo asignamos a la variable definida al principio del ejercicio de tipo *Statement*. Más tarde, simplemente, tendremos que realizar la consulta con el método "*executeQuery*" pasándole como parámetro la *query* previamente definida en la variable de tipo *String*.
- **Resultset:** El resultado de la *query* se asignará a una variable de tipo *Resulset*. Como podemos comprobar en el código, dicho *Resulset* está envuelto en un bucle "*while*", ya que por cada fila que nos devuelva esta tabla, podremos ir dando una vuelta más al bucle y seguir mostrando los resultados.

```
} catch (SQLException e) {
    System.out.println(e.getMessage());
} catch (ClassNotFoundException e) {
    System.out.println(e.getMessage());
} finally {
    if (statement != null) {
        statement.close();
    }
    if (dbConnection != null) {
        dbConnection.close();
    }
}
```

- **SQLException:** es capturada si a la hora de ejecutar el método "*executeQuery*" algo va mal en base de datos, ya sea gramaticalmente, sintácticamente, etc.
- **ClassNotFoundException:** es lanzada y capturada en este punto si en nuestra línea: "*Class.forName(DRIVER)*", el fichero del *driver* que le estamos indicando no encontrara la librería.
- **finally:** la sentencia *finally* se ejecutará siempre, hayamos capturado excepción o no. En esta, simplemente, se realizan los cierres de la clase *Statement* y del objeto *Connection*.

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

4. VENTAJAS E INCONVENIENTES DEL USO DE CONECTORES:

- **TIPO 1**

VENTAJAS	INCONVENIENTES
Se encuentran fácilmente. (paquete Java)	Rendimiento: demasiadas capas intermedias.
Acceso a gran cantidad de drivers.	Limitación funcionalidad
	No funcionan bien con applets.

- **TIPO 2**

VENTAJAS	INCONVENIENTES
Mejor rendimiento al tipo 1	La librería de la BD se inicia en la parte del cliente.
	Interfaz nativa Java.

- **TIPO 3**

VENTAJAS	INCONVENIENTES
No necesita librería del fabricante.	Requieren de un código específico de BDD para la capa intermedia
Son los que mejor rendimiento dan en internet,	

- **TIPO 4**

VENTAJAS	INCONVENIENTES
Buen rendimiento.	El usuario necesitará distinto software de conexión (driver) para cada BD.
No necesitan instalar un software especial. Drivers de fácil acceso.	

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga carpetas completas de una vez con el Plan PRO y PRO+