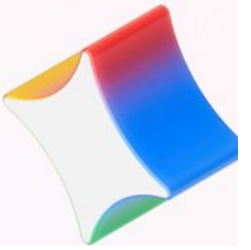
T1-Acceso-a-datos.pdf

bloodyraintatii

Acceso a datos

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España



Acceso a datos

Tema 1: Introducción al manejo de ficheros

b) Se han valorado las ventajas y los inconvenientes de las distintas formas de acceso.

Formas de acceder a un fichero

- Acceso secuencial → la información de nuestro fichero es una secuencia de caracteres o bytes, y para acceder a un byte/carácter, debemos acceder a los anteriores.
- Acceso aleatorio o directo → para acceder a un registro o posición determinada del fichero.

→ La diferencia más notable entre los dos es que los ficheros de acceso secuencial deberán ser recorridos byte a byte o carácter a carácter con el tiempo procesando la información con el uso de recursos que eso conlleva, y los ficheros de acceso aleatorio o directo establecen un puntero en bytes que indica la posición exacta para realizar la lectura y/o escritura directamente.

→ Clases de uso de acceso secuencial →
→ Clase de uso de acceso aleatorio →

Acceso basado en caracteres		Acceso basado en Bytes	
Entrada	Salida	Entrada	Salida
FileReader	FileWriter	FileInputStream	FileOutputStream
		RandomAccessFile	RandomAccessFile

Buffer
→ Realizar las operaciones con Buffer mejora el rendimiento del sistema.
→ BufferedInputStream, BufferedOutputStream, BufferedWriter y BufferedReader.
→ Es un espacio determinado y temporal, alojado en memoria, que realiza ciertas operaciones.
→ Almacena en memoria bloques de bytes completos (buffer), siendo el acceso a memoria mucho más rápido que el acceso a disco. Cuando se requiere más información, se reemplaza la que contiene el buffer volcando otro bloque de bytes a la memoria (buffer).

Tema 3: Trabajo con ficheros XML

b) Se han valorado las ventajas y los inconvenientes de las distintas formas de acceso.
g) Se han probado y documentado las aplicaciones desarrolladas.

Acceso a datos con DOM y SAX

→ DOM y SAX son estándares, herramientas para leer ficheros XML. Estas verifican si es un fichero sintácticamente válido.
→ Se les llama *parsers* o analizadores.

- DOM → carga el fichero completo en memoria, con todos los nodos y demás objetos. Es más lento y menos versátil. Ocupa mucha más memoria. Se pueden insertar o eliminar nodos.
    - Es aconsejable usarlo cuando tengamos claro el objetivo sobre el que queremos trabajar, a partir de un árbol creado en memoria.
- SAX → solo tiene en memoria la parte del nodo o evento actual. Es más rápido pero menos potente. Tenemos que introducir líneas de programación para obtener partes determinadas de los ficheros. Tiene mayor nivel de funcionalidad y versatilidad.
    - Es aconsejable usarlo para recorrer secuencialmente los elementos del fichero XML y realizar ciertas operaciones.

SAX	DOM
Basado en eventos	Carga el fichero en memoria
Va analizando nodo por nodo	Búsqueda de tags hacia delante y hacia detrás
En principio sin muchas restricciones de memoria ya que no carga la totalidad del fichero	Estructuras de árbol
Rapidez en tiempo de ejecución	Más lento en tiempo de ejecución
Es de sólo lectura	Se pueden insertar o eliminar nodos.
Es de sólo lectura	DataInputStream, DataOutputStream, PrintStream

Pruebas y documentación. JUnit.

→ Test → pieza de software que valida otra pieza de software, validando que los resultados que dé sean los esperados o ejecutando la secuencia esperada de operaciones y eventos. Ayudan al programador a verificar que un fragmento de código es correcto.
→ JUnit → framework que usa anotaciones para identificar diferentes tests.
→ Una prueba unitaria JUnit es un método que está en el interior de una clase llamada Test class.
→ Para definir que un método forma parte de un test, se le añade *@Test* sobre la cabecera.

→ Podemos poner solo esa anotación y que nuestro IDE nos pregunte si queremos importar las librerías de JUnit, o podemos añadir una dependencia maven con la librería de JUnit correspondiente y su versión.
→ Anotación @Before → es el trozo de código que se ejecutará antes de cada test. Sirve para instanciar la mayor parte de las variables que se necesitarán en los tests.
→ Anotación @After → se ejecutará después de cada test.

Tema 4: Manejo de conectores I

a) Se han valorado las ventajas e inconvenientes de utilizar conectores.

Conectores

→ Conector → serie de clases y librerías que unen la capa de la aplicación con la capa de base de datos. Es necesario para conectaros a la base de datos y realizar consultas.
→ Desfase objeto-relacional → es el problema de discrepancias que surgen porque las bases de datos relacionales y los aplicativos orientados a objetos no tienen la misma naturaleza. Se tiene que realizar una traducción desde los objetos del aplicativo Java a la base de datos relacional, por lo que se crearán entidades diferentes (que representan la misma unidad) en ambos sitios.

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. Hay dos protocolos de conexión:

- JDBC (Java Database Connectivity) → desarrollado por Sun.
- ODBC (Open Database Connectivity) → desarrollado por Microsoft.

→ Una aplicación debe tener asociado siempre un conector.
→ Con el conector JDBC no tenemos que desarrollar un aplicativo para acceder a la bbdd Oracle y otro aplicativo o driver para acceder a otra bbdd distinta, sino que el conector interpretaría el aplicativo de una forma u otra dependiendo de la bbdd asociada.

Conexiones: Componentes y tipos
→ Nos centraremos en el conector JDBC.
→ Componentes:

- La API → proporciona las librerías y clases para poder acceder a las bbdd relacionales y nos brinda la posibilidad de hacer consultas a la bbdd. Las clases están en *java.sql* y *javax.sql*.
- El paquete de pruebas → validan si un driver pasa los requisitos previstos por JDBC.
- El gestor → une la aplicación Java con el driver apropiado JDBC. Puede hacerlo con una conexión directa o a través de un pool de conexiones.
- El puente JDBC-ODBC → permite que las aplicaciones escritas en Java utilicen la API JDBC con muchos controladores ODBC existentes.

→ Dos tipos de arquitecturas:

- Arquitectura en dos capas → la aplicación se conecta a la bbdd a través de un driver. El driver y la aplicación deben estar en el mismo sistema o máquina.
- Arquitectura en tres capas → el aplicativo envía las instrucciones a una capa intermedia (middleware), que cogerá la info y la enviará a la bbdd correspondiente traduciendo los comandos enviados por el aplicativo.

→ Tipos de conexiones JDBC:

- Driver tipo 1 JDBC-ODBC → traduce las llamadas realizadas de JDBC a ODBC.
- Driver tipo 2 JDBC Nativo → traduce las llamadas en llamadas propias de la bbdd.
- Driver tipo 3 JDBC net → 3 capas, envía las llamadas a la bbdd con los drivers de tipo 1 o 2.
- Driver tipo 4 protocolo nativo → llamadas mediante el servidor con su protocolo nativo.

Ventajas e inconvenientes del uso de conectores

→ Drivers tipo 1:

- Ventajas → se distribuyen con el paquete del lenguaje Java y dan acceso a gran cantidad de drivers ODBC.
- Inconvenientes → rendimiento (tienen demasiadas capas intermedias), limitación de funcionalidad y no funcionan bien con *applets* (problemas en navegadores).

→ Drivers tipo 2:

- Ventajas → mejor rendimiento que el tipo 1 (ya que son llamadas nativas).
- Inconvenientes → la librería de la bbdd se inicia en la parte del cliente sí o sí, y usa interfaz nativa Java (no movible entre plataformas).

→ Drivers tipo 3:

- Ventajas → no necesitan librería del fabricante y son los que mejor rendimiento dan en Internet, teniendo muchas opciones de portabilidad y escalabilidad.
- Inconvenientes → requieren un código específico de bbdd para la capa intermedia.

→ Drivers tipo 4:

- Ventajas → tienen buen rendimiento, no necesitan instalar un software especial (ni en el servidor ni en el cliente), y los drivers son de fácil acceso.
- Inconvenientes → se necesita un driver (software de conexión) diferente para cada bbdd.

Descansito &Coins

Gemini + 

¿Sabrías identificar en qué te puede ayudar Google Gemini para estudiar?

REGLAS

1. Observa las opciones disponibles
2. Responde como Gemini te ayuda a estudiar.
3. Gana Wuolah coins para descargas sin publi.

Fácil | 10

A. Sintetiza horas de investigación en minutos.

B. Convierte tus apuntes en un esquema visual.

C. Prepara un examen para autoevaluarte.

D. Convierte tus apuntes en podcasts.

E. Sube hasta 1.500 páginas y analiza textos largos.

F. Todas las anteriores.

Gemini

Tema 7: El mapeo objeto relacional

a) Se ha instalado la herramienta ORM.

Herramientas ORM

→ ORM → framework de mapeo de objetos.

→ HIBERNATE → es el ORM más extendido y usado, disponible para Java y .Net. Facilita el mapeo relacional de los objetos entre una bbdd relacional y el modelo de objetos de la aplicación. Para ello, usa un fichero XML o anotaciones con las relaciones.

- Simplicidad → sencillo e intuitivo.
- Robustez → tiene muchas características adaptadas a Java (colecciones, herencia, etc.) Ofrece una propia capa de consultas SQL llamada HQL para facilitar la sintaxis y mejorar la eficiencia.

Instalación de Hibernate

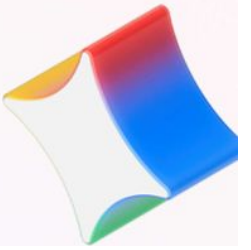
→ Se puede instalar de diferentes formas:

- Añadiendo el .jar al proyecto con la versión de Hibernate descargada.
- Añadiendo la dependencia de la versión Hibernate al proyecto web con gestor de dependencias maven. Al compilar, se descargará la librería y tendremos las clases del framework disponibles.

→ En nuestro caso usaremos la instalación agregando la dependencia usando Spring Boot si estamos haciendo una aplicación desde cero. Si no, tendríamos que quedarnos solo con la parte de agregar la dependencia.

→ Vamos a la web inicializadora de arquetipos de Spring Boot y añadimos el framework que estamos buscando con *Add dependencies*.

→ Si queremos agregar Hibernate sin usar Spring Boot de mediador, iríamos a la web oficial de Hibernate para encontrar la dependencia adecuada.



Tema 8: Exploración del mapeo objeto-relacional

c) Se han definido configuraciones de mapeo.

Composición
→ Normalmente las relaciones de mapeo objeto-relacional están definidas en un XML, que guía a Hibernate.
→ Hay diversas herramientas para generar el documento (aunque también se puede hacer manualmente) → Xdoclet, Middlegen y AndroMDA.

Elementos
→ El documento de mapeo es un fichero XML cuyo elemento principal es <hibernate-mapping>, que almacenará las clases definidas dentro.
→ Elementos de tipo <class> → definen mapeos esenciales que van desde las clases Java (atributo *name*) a las tablas de bbdd (atributo *table*).
→ Elementos <meta> → definen la descripción de una clase (opcionales).
→ Elementos <id> → mapea el atributo ID en Java y lo transforma en PK (Primary Key) en la bbdd. El atributo *name* es el atributo de la clase Java y *column* es la columna de la tabla de la bbdd. El atributo *type* proporciona la tipología del objeto a Hibernate, desde Java a SQL.
→ Elemento <generator> → genera automáticamente la PK junto con el elemento *id*. El atributo *class* de este elemento se establece con el valor *native* para permitir a Hibernate crear la PK con diferentes algoritmos (*identity*, *hilo* o *sequence*).
→ Elemento <property> → mapea los atributos o propiedades de Java y los transforma en columnas de la tabla asociada en la bbdd. El atributo *name* es el nombre del atributo en la clase Java y *column* es la columna de la bbdd. El atributo *type* proporciona y transforma la tipología del objeto Java a objeto de tipo SQL.

acceso a datos
---1-1b
b) Se han valorado las ventajas y los inconvenientes de las distintas formas de acceso.
---3-1b, 1g
b) Se han valorado las ventajas y los inconvenientes de las distintas formas de acceso.
g) Se han probado y documentado las aplicaciones desarrolladas.
---4-2a
a) Se han valorado las ventajas e inconvenientes de utilizar conectores.
---7-3a
a) Se ha instalado la herramienta ORM.
---8-3c
c) Se han definido configuraciones de mapeo.