TEMA-8-ACCESO-A-DATOS.pdf

user_4383848

Acceso a datos

1° Desarrollo de Aplicaciones Multiplataforma

Estudios España

TEMA 8 - ACCESO A DATOS
EXPLORACIÓN DEL MAPEO OBJETO - RELACIONAL

1. CLASES PERSISTENTES:

A las clases cuyos objetos serán almacenados en base de datos son llamadas **clases persistentes** en Hibernate.
Reglas principales:
• Todas las clases que van a ser persistidas necesitarán constructor por defecto.
• Todas las clases deben contener un atributo ID para facilitar la identificación de los objetos tanto en Hibernate como en base de datos. Será mapeado como primary key en base de datos.
• Todos los atributos de la clase deberán ser definidos como privados y tener métodos get() y set().
• Una característica de las clases persistentes de Hibernate suele ser que dichas clases no sean de tipo “final”.

2. FICHEROS DE MAPEO I. COMPOSICIÓN:

Las relaciones de mapeo objeto-relacional están definidas en un documento con extensión XML. Este documento guía a Hibernate, pero, existen diversas herramientas que generan dicho documento. Entre otras, **Xdoclet, Middlegen** y, para desarrolladores algo más avanzados, **AndroMDA**.

3. FICHERO DE MAPEO II. ANÁLISIS DE ELEMENTOS:

• El documento de mapeo es un fichero XML que tiene como elemento principal **<hibernate-mapping>**, el cual, en su interior, almacenará todas las clases definidas.
• Los elementos de tipo **<class>** son usados para definir mapeos especiales que van desde nuestras clases Java definidas a las tablas de base de datos. El nombre de la clase Java es **especificado usando el atributo name**, y la tabla asociada en base de datos es vinculada con el **atributo table**.
• Los elementos **<meta>** son opcionales y pueden ser usados para definir la descripción de una clase.
• El elemento **<id>** mapea el atributo ID único en la parte de la clase Java y lo transforma en Primary Key en la tabla de la base de datos. El atributo name hace referencia al atributo de la clase Java y column se refiere a la columna real que hay en la tabla de base de datos. El atributo type proporciona a Hibernate la tipología del objeto y será convertido desde tipología Java a SQL.
• El elemento **<generator>**, junto con el elemento id, es usado para generar la clave primaria automáticamente. El atributo class del elemento generator se establece con el valor native para permitir a Hibernate crear la Primary Key con diferentes algoritmos: identity, hilo o sequence, dependiendo de las capacidades de la base de datos
• El elemento **<property>** se usa para mapear los atributos o propiedades de Java y transformarlos en columnas de la tabla asociada en la base de datos. El atributo **type** proporciona y transforma la tipología del objeto Java a objeto de tipo SQL.

4. SESIONES Y OBJETOS HIBERNATE I. ESTADOS:

Un **objeto de sesión** es usado para establecer una conexión física con una base de datos. Este objeto es ligero y diseñado para ser instanciado cada vez que se necesite una interacción con la base de datos. Los objetos persistentes son almacenados y devueltos a través del objeto de sesión **(Session object).**
El objeto de sesión no debe mantenerse abierto durante mucho tiempo, ya que podría ser peligroso por temas de seguridad y, por lo tanto, deben ser creados y destruidos cada vez que sea necesario utilizarlos. La función principal de estos objetos es ofrecer, crear, leer y borrar operaciones para las instancias de las clases mapeadas. Dichas instancias pueden estar en uno de los siguientes estados:
• **Transient:** Nueva instancia de una clase persistente, no está aún asociada a un objeto de sesión y no tiene representación en la base de datos ni identificador según Hibernate.
• **Persistent:** Una instancia transient se puede hacer persistente asociándola con una sesión. Una instancia persistente tiene una representación en la base de datos, un identificador, y la asociación con el objeto de sesión.
• **Detached:** Una vez se cierra la sesión de Hibernate, la instancia persistente se convertirá en una instancia separada.

5. SESIONES Y OBJETOS HIBERNATE II. MÉTODOS:

• **beginTransaction():** Permite comenzar una unidad de trabajo y devolver el objeto asociado de la transacción.
• **cancelQuery():** Cancela la ejecución de la consulta actual.
• **Clear():** Elimina completamente la sesión.
• **Close()**: Finaliza la sesión liberando la conexión JDBC y limpiándola. Devuelve un objeto de tipo Connection.
• **createCriteria(Class clasePersistente):** Crea una instancia nueva de Criteria para la clase persistente proporcionada como parámetro. Devuelve un objeto de tipo Criteria.
• **createCriteria(String entityName):** Crea una instancia de tipo Criteria para la entidad que se le pasa como parámetro. Devuelve un objeto de tipo Criteria.
• **getIdentifier(Object objeto)**: Devuelve un objeto de tipo Serializable que es el identificador de la entidad proporcionada y asociada a esta sesión.
• **createFilter(Object colección, String consulta)**: Crea una nueva instancia de consulta en función a la colección pasada como parámetro y la consulta. Devuelve un objeto de tipo Query.
• **createQuery(String consultaHQL):** Crea una instancia de consulta en función a la sentencia HQL que se le pasa como parámetro. Devuelve un objeto de tipo SQLQuery.
• **delete(Object objeto):** Borra una instancia persistente del almacén de datos.
• **getSessionFactory():** Devuelve un objeto de tipo SessionFactory, el cual creó la sesión actual.
• **refresh(Object objeto):** Vuelve a leer el estado de la instancia dada como objeto proveniente de la base de datos.
• **isConnected():** Comprueba si la sesión está conectada actualmente.
• **isOpen():** Comprueba si la sesión aún está abierta.

6. CARGA, ALMACENAMIENTO Y MODIFICACIONES DE OBJETOS:

Para recuperar objetos de nuestra base de datos usando Hibernate, y diferentes formas de actualizarlos, guardarlos, etc.

Comandos que ejecutaremos desde nuestro objeto Session:

• Carga de objetos: Obtendremos datos de nuestra base de datos.

- **Session.get():** Devuelve un objeto persistente según los parámetros de objeto de entidad e identificador, y si no existe objeto en base de datos, devolverá **null**.
- **Session.load():** También devuelve un objeto persistente teniendo en cuenta los parámetros que se le pasan de entidad e identificador. Devolverá un **ObjectNotFoundException** en caso de no encontrar dicha entidad.

• Almacenamiento de objetos: Para guardar la información en base de datos desde Hibernate usaremos:

- **Session.persist():** Ejecuta el comando insert del lenguaje SQL almacenando filas en la base de datos. Este método es de tipo void(), no devuelve nada.
- **Session.save():** Igual que el método anterior, ejecuta internamente un método insert de SQL con la diferencia de que devuelve un objeto de tipo Serializable.

• Modificación de objetos: Con los siguientes comandos actualizaremos los diferentes objetos en base de datos.

- **Session.update():** Realizaremos un update en base de datos. Es el método primitivo para actualizar filas y necesita que haya otra instancia de session ejecutándose, y si no, lanzará una excepción.
- **Session.merge():** Se ejecutará un update también en base de datos, pero, en este caso, no tendremos que preocuparnos si existe ya una instancia ejecutándose, ya que este método realizará la operación gestionando el resto.

• Otros métodos:

- **Session.delete():** Pasaremos como parámetros la entidad persistente y se realizará el borrado en base de datos.
- **Session.saveOrUpdate():** Método de gran utilidad para permitir tanto la actualización de la entidad (si existe en base de datos) como el insert (si no existe en base de datos).

7. CONSULTAS SQL Y HQL:

El lenguaje **Hibernate Query (HQL)** es un lenguaje orientado a objetos muy parecido a SQL, pero en lugar de operar con tablas y columnas, HQL trabaja con objetos persistentes y con las propiedades de los mismos. En Hibernate, se recomienda usar HQL para evitar problemas de portabilidad de la base de datos y aprovechar también las estrategias de caché implementadas en Hibernate.

```
String hql = "FROM Customer";
Query consulta = session.createQuery(hql);
consulta.setFirstResult(1);
consulta.setMaxResults(40);
List results = consulta.list();
```

Aquí consultamos la **tabla "Customer**", con la que obtenemos todos los registros, y más tarde, con "**setFirstResult" y "setMaxResults**", aplicamos un filtro de cuantas filas queramos obtener. En este caso, las 40 primeras.
También, podemos emplear sentencias de SQL nativas si queremos usar funcionalidades específicas de la base de datos que tenemos conectada a nuestra aplicación.
Se ejecutan dentro del motor de la base de datos relacional.
Continuando con el tema de las sentencias nativas SQL, podremos ejecutarlas desde nuestro código con el comando: **createSQLQuery()**
Este método recibirá como parámetro una variable de tipo String que contendrá la consulta nativa SQL. Devuelve un **objeto de tipo SQLQuery** y una vez obtenido dicho objeto, se puede asociar perfectamente a una entidad existente de Hibernate.

8. GESTIÓN DE TRANSACCIONES CON HIBERNATE:

Una transacción representa una unidad de trabajo. Si uno de los pasos falla, la transacción completa fallará (relacionado con el concepto de atomicidad). Una transacción se define por las características **ACID (Atomicidad, Consistencia, aislamiento (Isolation) y Durabilidad)**.
En Hibernate, tenemos la interfaz Transaction que define la unidad de trabajo. Mantiene la abstracción desde su propia implementación.
Algunos de los **métodos principales en la interfaz** Transaction son:
• **Void begin**: Empieza una transacción nueva.
• **Void commit():** Finaliza la unidad de trabajo a menos que estemos en el modo "FlushMode.NEVER".
• **Void rollback():** Fuerza a cancelar totalmente la transacción.
• **Void setTimeout(int segundos):** Establece un time out determinado por parámetro en segundos.
• **Boolean isAlive():** Comprueba si la transacción está aún activa.
• **Void registerSynchronization(Synchronizations):** Registra la respuesta sincronizada de un usuario para esa transacción.
• **Boolean wasCommited():** Comprueba si se ha cerrado la transacción satisfactoriamente.
• **Boolean wasRolledBack():** Comprueba si la transacción ha sido cancelada satisfactoriamente.