índice

Tema 6.

La estructura de un fichero XML de mapeo y las relaciones entre objetos y bases de datos. Se estudiará el objeto sesión, sus métodos y sentencias comunes, y se verán las formas de extraer, guardar y modificar datos con Hibernate.

Clases persistentes POJO

Hibernate, como un ORM, permite persistir valores de atributos de clases en Java en tablas de bases de datos. Un documento de mapeo es esencial para determinar como se mapean esos valores a los campos de la base de datos. Las clases que se almacenan en la base de datos se les llama clases persistentes.

- Todas las clases persistentes necesitan un constructor
- Deben tener un atributo ID que se mapea como clave primaria en la bbdd
- todos los atributos tienen que ser privados y tener métodos get y set publicospara acceder a ellos
- las clases persistentes no pueden ser de tipo final

Se usa el nombre de POJO (Plain Old Java Object) para indicar que estas clases son objetos ordinarios de java, sin caracteristicas especiales.

Fichero de mapeo I. Composición

Muchos desarrolladores que usan Hibernate prefieren escribir el fichero de mapeo a mano. Pero hay herramientas que generan el documento como Xdoclet o Middlegen, para desarrolladores más avanzados AndroMDA.

Partiendo de la clase Java POJO, habría una tabla correspondiente a cada objeto.

Fichero de mapeo II. Análisis de elementos

- El documento de mapeo fichero xml que tiene como elemento principal <hibernate-mapping> que almacena todas las clases definidas
- Los elementos de tipo <class> Se utilizan para definir mapeos entre las clases Java y las tablas de la base de datos. El atributo `name` especifica la clase Java y `table` vincula la tabla asociada.
- Los elementos <meta> son opcionales y pueden ser usados, por ejemplo, para definir la descripción de una clase.
- El elemento <id> Mapea el atributo ID único de la clase Java como clave primaria en la tabla. El atributo `name` se refiere al atributo en la clase, `column` a la columna en la tabla, y `type` indica la tipología del objeto para la conversión de Java a SQL.
- El elemento <generator> Se utiliza junto con `<id>` para generar automáticamente la clave primaria. El atributo `class` se establece como `native` para permitir que Hibernate use diferentes algoritmos (identify, hilo o sequence) según las capacidades de la base de datos.

- El elemento <property> Mapea los atributos de Java a columnas en la tabla. El atributo `name` se refiere al atributo en la clase, `column` a la columna en la base de datos, y `type` transforma la tipología del objeto Java a SQL.

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 bbdd. 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, por seguridad deben ser creados y destruidos cada vez que sea necesario utilizarlos.

La función principal de estos objetos es: ofrecer, crear, leer y borrar, las instancias pueden estar en uno de estos estados:

- Transient Nueva instancia de una clase persistente, no está asociada a un objeto de sesión y no tiene representación tiene una representación en la bbdd, un id y la asociación con el objeto de sesión.
- Persistent una instancia transient se puede hacer persistente asociándola con una sesión. Una instancia persistente tiene una representación en la bbdd, un id, y la asociación con el objeto de sesión
- Detached Una vez se cierra la sesión de Hiibernate, la instancia persistente se convertirá en una instancia separada.

Sesiones y objetos Hibernate II. Métodos

Métodos en la interfazSession:

- beginTransaction(): empezar una transacción
- cancelQuery(): cancela la ejecución de la consulta actual
- Clear(): elimina completamente la sesión
- Close(): finaliza la sesión liberando la conexión JDBC
- createCriteria(Class classPersistente): 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.
- getIdentifier(Object objeto): Devuelve un objeto de tipo Serializable que es el identificardor de la entidad proporcionada y asociada a esta sesión.
- createFilter(Object colección, String consulta): Crea una instancia de consulta en función a la colección pasada como parámetro y la consulta.

- createQuery(String consultaHQL): crea una instancia de consulta en función de la sentencia HQL que se le pasa como parámetro.
- delete(Object objeto): borra una instancia persistente del almacén de datos.
- GetSessionFactory(): devuelve un objeto de tipo SessionFactory
- refresh(Objeto objeto): vuelve a leer el estado de la instancia dada como objeto proviniente de la base de datos.
- IsConnected(): comprueba si la sesión está conectada actualmente
- isOpen(): comprueba si la sesión aún está abierta

Carga, almacenamiento y modificaciones de objetos

Como recuperar objetos de nuestra base de datos usando Hibernate. Comandos que ejecutaremos desde nuestro objeto Session:

- Carga de objetos: obtener datos de la base de datos
  - Session.get(): devuelve un objeto persistente de entidad e identificador y si no existe devuelve null
  - Session.load(): devuelve un objeto persistente teniendo en cuenta los parámetros que se le pasan de entidad e id. Devolverá un ObjectNotFound en caso de no encontrar la entidad
- Almacenamiento de objetos:
  - Session.persist(): Ejecuta el comando insert del lenguaje SQL almacenando filas en la base de datos. Este método es 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: actualizar los objetos de la base de datos
  - Session.update(): realizaremos un update en base de datos.
  - Session.merge(): Se ejecutará un update también en bbdd, pero en este caso, no tendremos que preocuparnos si existe ya una instancia ejecutándose,, ya que el método realizará la operación gestionando el resto.
- Otros métodos:
  - Session.delete(): Pasaremos como parámetros la entidad persistente y se realizará el borrado en bbdd.
  - Session.saveOrUpdate(): método de gran utilidad para permitir tanto la actualización de la entidad (si existe en bbdd) como el insert (si no existe en bbdd)

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 estos. Las consultas de tipo HQL son traducidas por Hibernate en consultas corrientes SQL, que más tarde son ejecutadas en la bbdd.

Se recomienda usar HQL para evitar problemas de portabilidad de la bbdd y aprovechar las estrategias de caché implementadas en Hibernate.

También podemos emplear sentencias de SQL nativas si queremos usar funcionalidades específicas de la bbdd que tenemos conectada a nuestra aplicación. Por ejemplo la ejecución de procediminetos almacenados en bbdd.

Podemos ejecutar secuencias nativas SQL con createSQLQuery()

recibirá como parametro una variable de tipo String que contendrá la consulta nativa SQL.

Devuelve un objeto de tipo SQLQuery.

Gestión de transacciones con Hibernate

Una transacción representa una unidad de trabajo. Si uno de los pasos falla, la transacción completa fallará.

(atomicidad). Se define por las características ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad)

En Hibernate, tenemos la interfaz Transaction.

Algunos métodos principales en la interfaz Transaction.

- **Void begin():** Inicia una nueva transacción.
- **Void commit():** Confirma todos los cambios realizados en la transacción actual y los guarda en la base de datos.
- **Void rollback():** Revierte todos los cambios realizados en la transacción actual, deshaciendo cualquier modificación.
- **Void setTimeout(int segundos):** Establece un tiempo de espera para la transacción. Si el tiempo se excede, la transacción puede ser cancelada.
- **Boolean isAlive():** Verifica si la transacción actual está activa y en curso.
- **Void registerSynchronization(Synchronization s):** Registra un objeto de sincronización que será notificado sobre los eventos de la transacción (como commit o rollback).
- **Boolean wasCommitted():** Indica si la transacción fue confirmada exitosamente.
- **Boolean wasRolledBack():** Indica si la transacción fue revertida.