Hibernate como ORM para desarrollo de Aplicaciones.

Mauricio J. Mercado A.
Facultad de Ciencias y Tecnología, Universidad Católica de Asunción.
Asunción, Paraguay
mauriciojose@gmail.com

Resumen

El siguiente es un documento que sirve de introducción de la tecnología y la idea de un ORM, asi mismo, sus problemas y ventajas.

Introducción

Hoy día, mucho desarrolladores trabajan en Sistemas de Información Empresariales (SIE). Esta clase de aplicación crea, maneja y almacena información; y comparte esta información entre muchos usuarios en múltiples localidades.

El almacenamiento de datos de los SIE involucra utilización masiva de bases de datos relacionales utilizando SQL, lo que la mayoría de las veces lleva a una dependencia de la tecnología de la base de datos utilizada.

En los últimos tiempos, la adopción del lenguaje de programación Java ha traído la ascendencia del paradigma orientado a objetos para el desarrollo de aplicaciones. Desarrolladores ahora son comprados con los beneficios de la orientación a objetos. Sin embargo, la mayoría de los negocios están atados a inversiones a largo plazo en base de datos relaciones muy caras.

Sin embargo, la representación tabular de los datos en los sistemas relacionales es fundamentalmente diferente que la red de objetos utilizada en aplicaciones orientadas a objetos. Esta diferencia ha llevado a lo que se llama el paradigma del objeto/relación. Tradicionalmente, la importancia y costo de este desacuerdo ha sido dejado de lado, y herramientas para

resolver este problema han sido insuficientes. Entre tanto, los desarrolladores culpan a la tecnología relacional, y los profesionales en datos culpan a la orientación a objetos.

El *Object/relational mapping* (ORM) es el nombre que se le ha dado para automatizar soluciones para el problema anteriormente citado. Para los desarrolladores preocupados por un tedioso código de acceso a los datos, el ORM puede ser la solución. Aplicaciones construidas con ORM como mediador se puede esperar que sean mas baratos, con mas perfomance, menos dependiente, y mas capaz de estar a la par con los cambios internos de un esquema SQL.

Desarrollo

Object/Relational Mapping:

Antes de empezar el desarrollo definimos la idea central del trabajo que es object/relational mapping, y se define como sigue: Es el proceso de transformación entre el modelo de objetos y relacional y entre el sistema que soporta estos enfoques.

Modelado de Objetos y el Modelado Relacional.

A continuación introducimos los conceptos basicos de los modelados de Objetos y Relacionales.

Modelado de Objetos.

Los modelos de objetos son distintos de otros modelos porque ha fusionado los conceptos de variables y tipos abstractos de datos en una variable de tipo abstracto: un objeto.

Los objetos tienen identidades, estados, y comportamiento; modelos de objetos son hechos a partir de sistemas con estos objetos.

Existen conceptos de tipo, herencia, asociación, encapsulación y otros.

El modelado de objetos es solo un paso de la programación orientada a objetos.

El modelado de objetos se concentra en la identidad y el comportamiento, es completamente diferente del modelo relacional se concentre en la información.

Modelado Relacional.

El modelo relacional trabaja en terminos de predicados y axiomas de verdad.

Las realciones definen las sentencias que sean posiblemente verdaderas.

Las tuplas describen el conocimiento actual de la verdad.

El modelo relacional es muy distinto del modelo de objetos.

Que es la Persistencia?

Casi todas las aplicaciones requieren persistencia de datos. Persistencia es uno de los conceptos fundamentales en el desarrollo de aplicaciones. Si un sistema de información no preserva la información introducida por un usuario cuando su computadora se apaga el sistema es de muy poca utilidad. Cuando hablamos de persistencia, hablamos normalmente de almacenar datos en una base de datos relacional utilizando SQL.

Persistencia en Aplicaciones Orientadas a Objetos.

En aplicaciones orientadas a objetos, la persistencia permite a un objeto sobrevivir el proceso que lo creo. El estado del objeto puede ser almacenado en el disco y un objeto con el mismo estado puede ser recreado en algún punto en el futuro. Esta aplicación no esta limitada simplemente a objetos simples, grafos completos interconectados de objetos pueden ser persistentes y después recreados en un nuevo proceso. La mayoría de los objetos no son persistentes, un objeto trascendente tiene un tiempo de vida limitado que esta asociado al proceso que lo creo.

Bases de datos relacionales modernas proveen una representación estructurada de la persistencia de datos, posibilitando el ordenamiento, la búsqueda, y la conjunción de datos. Manejadores de bases de datos son responsables de manejar la concurrencia y la integridad de los datos, son los responsables de compartir datos entre múltiples usuarios y múltiples aplicaciones.

Capas de persistencia.

En una aplicación mediana o grande, usualmente tiene sentido organizar las clases por consentimiento. Persistencia es un consentimiento. Otros consentimientos son presentación, flujo de trabajo y lógica de negocios.

Típicamente una arquitectura orientada a objetos comprende de capas que representan estos consentimientos.

Una arquitectura con capas define interfaces entre códigos que implementan varios consentimientos, permitiendo una cambio en un consentimiento sin afectar a otro.

Las reglas son las siguientes:

- Las capas se comunican de arriba hacia abajo. Una capa depende solo de la capa que esta directamente debajo de el.
- Cada capa no conoce la existencia de cualquier otra copa excepto de la capa debajo de el.

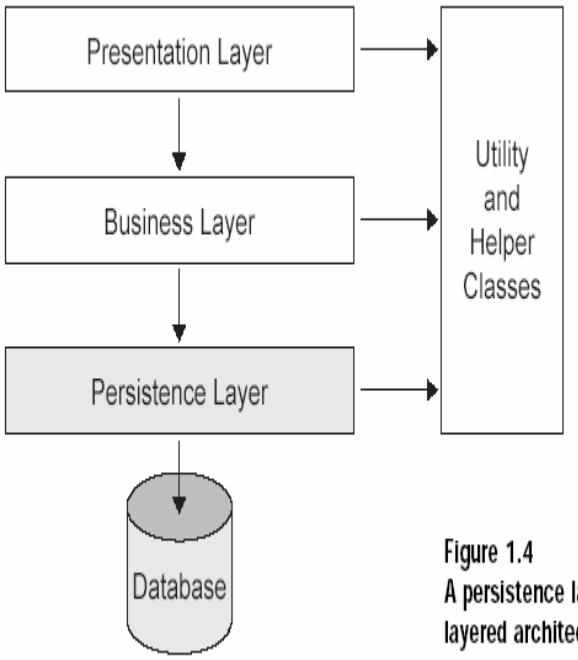

Figure 1.4
A persistence layer is the basis in a layered architecture.

Las capas comúnmente definidas son las siguientes:

- Capa de presentación: la lógica de la interfase de usuario esta siempre mas arriba. Código responsable de la presentación y control de las paginas y la navegación.
- Capa de negocios: la forma exacta que la siguiente capa varia ampliamente entre aplicaciones. Sin embargo es de común acuerdo que la capa de negocios es responsable de implementar cualquier regla de negocio o requerimiento de sistema que puede ser entendido por los usuarios como parte del dominio del problema.

- Capa de Persistencia: la capa de persistencia es una grupo de clases y componentes responsables del almacenamiento de datos y el acceso a los mismos por uno o mas usuarios. Esta capa incluye necesariamente un modelo de las entidades del dominio del negocio.
- Base de Datos: es la representación actual de la persistencia del estado del sistema.

Ahora que vimos la introducción necesaria de los conceptos podemos entender un poco mas donde el hibernate actuá y para que sirve. Hibernate es un de los ORM mas populares que existen en la actualidad.

En resumidas cuentas, un ORM es la persistencia automatizada de objetos de una aplicación en las tablas de una base de datos relacional. ORM en esencia, es transformar los datos de una forma a otra.

Esto implica por supuesto, algunas perdidas en perfomance. Sin embargo si el ORM es implementado como mediador hay muchas oportunidades de optimización que no existirían en otras circunstancias.

Una solución ORM consiste de las siguientes piezas:

- Un API para realizar las operaciones básicas con los objetos.
- Un lenguaje para especificar las consultas que refieren a las clases y las propiedades de las mismas.
- Una facilidad para especificar el mapeado de los meta-datos.
- Una técnica para la implementación del ORM para interactuar con objetos transaccionales para realizar ciertas funciones.

Utilizamos el termino ORM para incluir cualquier capa de persistencia donde el SQL es generado de una descripción de meta-datos.

Existen varios niveles de calidad de los ORM. Ellos son:

- Relacional Puro: la aplicación entera, incluyendo la interfase de usuario, esta diseñada alrededor del modelo relacional y de las operaciones SQL. Este esquema, sin tener en cuenta las deficiencias para sistemas grandes, puede ser una excelente solución para aplicaciones simples donde un bajo nivel de reutilización de código es aceptable. SQL directo puede ser fácilmente optimizable en varios aspectos, pero las malas noticias son la falta de portabilidad y manutención. Aplicaciones en esta categoría utilizan mucho procedimientos almacenados.
- Mapeado ligero de Objetos: entidades son representadas como clases que son mapeadas manualmente a las tablas relacionales. Esta forma es extremadamente utilizada y es muy exitosa para aplicaciones con pocas entidades.
- Mapeado mediano de Objetos: la aplicación esta diseñada alrededor de un modelo de objetos. El SQL es generado en tiempo de construcción utilizando un generador de código. Asociaciones entre objetos son soportadas por el mecanismo de persistencia. Los objetos son cacheados por la capa de persistencia.
- Mapeado completo de Objetos: soporta modelado sofisticado de objetos: composición, herencia, polimorfismo y persistencia por alcance. La capa de persistencia implementa persistencia transparente; clases persistentes no heredan de ninguna clase especial o tienen que implementar alguna interfase.

Hibernate

Ahora introduciremos básicamente al ORM posiblemente mas popular, el hibernate, vamos a ver un funcionamiento básico del mismo, como cargar datos y como actualizar. También veremos como se configuran las clases para que el hibernate pueda realzar su trabajo.

Utilizamos la siguiente clase de ejemplo:

```
public class Message {
private Long id;
private String text;
private Message nextMessage;
private Message() {}
public Message(String text) {
this.text = text;
}
public Long getId() {
        return id;
}
private void setId(Long id) {
        this.id = id;
}
            public String getText() {
        return text;
}
public void setText(String text) {
        this.text = text;
}
```

```
public Message getNextMessage() {
        return nextMessage;
}
public void setNextMessage(Message nextMessage) {
        this.nextMessage = nextMessage;
}
}
```

La clase Message tiene tres atributos: el atributo identificador, el texto del mensaje y una referencia a otro Message. El atributo identificador permite a la aplicación acceder a la base de datos identidad –primary key- de un objeto persistente. Si dos instancias de Message tienen el mismo valor en el identificador, se refieren a la misma fila en la base de datos.

Insertando datos a la BD

```
Session session = getSessionFactory().openSession();
Transaction tx = session.beginTransaction();
Message message = new Message("Hello World");
session.save(message);
tx.commit();
session.close();
```

El código anterior hace llamadas a las interfaces de Session y Factory. El resultado de la ejecución del código anterior seria algo como:

insert into MESSAGES (MESSAGE_ID, MESSAGE_TEXT, NEXT_MESSAGE_ID) values (1, 'Hello World', null)

De esta manera podemos ver básicamente como podemos insertar datos a nuestra base de datos y como podemos acceder a los mismos. Aun nos queda un aspecto muy importante por aclarar que es como el hibernate sabe como crear las tablas para las clases. Esto lo hace leyendo archivos de

configuración xml para cada donde se especifican las informaciones de cada clase, cabe destacar que se tiene un archivo de configuración de estos para cada clase. Para el ejemplo anterior tenemos que el archivo de configuración xml seria el siguiente:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
        <class
                name="hello.Message"
                Table="MESSAGES">
        <id
                name="id"
                column="MESSAGE_ID">
                <generator class="increment"/>
        </id>
        <property
                name="text"
                column="MESSAGE_TEXT"/>
        <many-to-one
                name="nextMessage"
                cascade="all"
                column="NEXT_MESSAGE_ID"/>
        </class>
</hibernate-mapping>
```

Como acceder a los objetos.

```
Session session = getSessionFactory().openSession();
Transaction tx = session.beginTransaction();
Message message =
(Message) session.load( Message.class, new Long(id) );
message.setText(“Hola");
Message nextMessage = new Message(“Hola");
message.setNextMessage( nextMessage );
tx.commit();
session.close();
```

Con este bloque de condigo podemos ver como retirar los objetos que se encuentran en nuestra base, si miramos detenidamente podemos ver que se utilizan de vuelta las interfaces de Session y Transaction.

Podemos ver además que para cargar un objeto utilizamos el método *load* de la interface Session y le pasamos como parámetro el *id* del objeto que queremos acceder.

Si bien existe mucho mas acerca de hibernate no esta al alcance de este documento desarrollar en profundidad todas sus bondades, tan solo una breve introducción de que como funciona para que podamos ver con un poco mas de claridad los conceptos de un ORM.

Porque ORM?.

Una de las ventajas de la utilización de un ORM es que sirve como “escudo” ante la cantidad de SQL que podemos tener en frente. Algunos de los beneficios del hibernate como ORM son:

- **Productividad:** código relacionado con persistencia puede ser a lo mejor el código mas tedioso en cualquier lenguaje. Hibernate elimina mucho de este trabajo y permite que uno se concentre mas en el problema del negocio. No importa que estrategia de desarrollo uno adopte, hibernate siempre puede reducir el tiempo de desarrollo.
- **Manutención:** menos líneas de código hace que el sistema sea mas entendible ya que enfatiza mas la lógica del negocio. Mas importante, un sistema con menos código es mas fácil de mantener. Sin embargo hay otras razones porque hibernate es mas fácil de mantener. En sistema donde la persistencia esta hecha a mano, existe una inevitable tensión entre la representación relacional y el modelo objeto que implementa el dominio. Cambios en alguno de los dos siempre implica cambios en el otro.
- **Perfomance:** se suele decir que la persistencia hecha a mano siempre puede ser al menos tan rápido como la automatizada. Es cierto sentido es cierto, pero que sucede cuando tenemos en cuenta el tiempo y el presupuesto, obviamente no tenemos todo el tiempo del mundo ni el presupuesto para poder hacer las cosas como queremos, siempre se hacen ciertos sacrificios. En proyectos donde el tiempo es escaso un ORM siempre ayuda a ganar tiempo. Hibernate permite muchas optimizaciones durante todo el tiempo. Además, la persistencia automática mejora la productividad del desarrollador que puede utilizar su tiempo optimizando otras cosas.

- **Independencia:** un ORM nos permite abstraernos de la base de datos SQL y el dialecto SQL. Si la herramienta soporta un numero variado de base de datos, esto confiere un cierto nivel de portabilidad a la aplicación. En ciertas situación esto es muy difícil de lograr, ya que en algunos casos se requiere de ciertas características propias de algunas base de datos.

Problemas de los ORM.

La siguiente es una lista de las preguntas mas frecuentes que se realizan los usuarios de los ORM.

1. A que se parece la persistencia de datos?.
2. Como se define el mapeado de los meta-datos?.
3. Como se debería mapear la jerarquía de herencia de los objetos?.
4. Como la lógica de la persistencia interactuá en tiempo de ejecución con los objetos?.
5. Cual es el ciclo de vida de los objetos persistentes?.
6. Cuales son las facilidades para el ordenamiento, búsqueda y agregación?
7. Como obtenemos efectivamente los datos con asociaciones?.

Además de estas preguntas que reflejan las inquietudes mas frecuentes, existen dos aspectos de suma importancia que siempre imponen restricciones en el diseño y en la arquitectura de un ORM:

- Transacciones y Concurrencia.
- Manejador de Caches.

Caso de Estudio.

Para el trabajo practico de la materia de Ingeniería de Software quisimos utilizar el hibernate para poder conseguir un aumento del desempeño de los integrantes del grupo de tal manera que podamos alcanzar todos los objetivos propuestos para este trabajo.

El problema fue que muy pocos tenían experiencia en el desarrollo de aplicaciones de Base de Datos, y esto llevo a que los conceptos y las ideas del hibernate no sean del todo explotadas, de hecho, se convirtió en un problema mismo, ya que costo de una manera considerable poder entender el funcionamiento del mismo mas aun cuando se utilizaba un grafo muy complicado de dependencia de los objetos.

Por lo tanto es importante tener en cuenta sobre todo la experiencia del grupo que quiere utilizar la herramienta y sobre todo de un tiempo prudencial de aprendizaje de la herramienta ya que esto es clave para poder utilizar con eficiencia. Esto también consistió en un problema ya que no disponíamos de mucho tiempo para la finalización del trabajo por lo que en vez de ayudarnos a ganar tiempo se convirtió en un perdida de tiempo que no teníamos.

De cualquier manera, la experiencia fue muy valida, y logramos comprender la potencia que un framework como hibernate posee, pero consideramos su utilización para sistemas medianos a grandes, ya que posee una complejidad propia que debe ser tenida en cuenta y no se justifica para sistemas de poca envergadura.

Conclusión.

La persistencia de objetos significa que objetos individuales pueden sobrevivir el proceso de una aplicación, pueden ser guardados y vueltos a recrear en un punto mas adelante del proceso. El problema del objeto/relación entra en juego cuando el lugar de almacenamiento es base de datos relacional SQL, donde un grafo de objetos no puede ser guardado simplemente en una tabla de la base datos, tiene que desarmarse e insertarse en columnas. Una buena solución a esto son los ORM.

En la arquitectura de un modelo con capas, el dominio del modelo es utilizado para ejecutar la lógica del negocio en la capa de negocio. Esta capa de negocios se comunica con la capa de persistencia para poder cargar y almacenar la persistencia de objetos del dominio del modelo. El ORM es el mediador en la capa de persistencia que maneja la persistencia.

Los ORM no son la solución absoluta para todas las tareas de persistencia; su trabajo es el de aliviar al desarrollador en un 95 % del trabajo de la persistencia de los objetos.

Es posible que una mejor solución exista con el tiempo, pero se debería de pensar en muchos factores como el SQL, los APIs de persistencia y la integración de las aplicaciones. Los ORM son la mejor solución actualmente disponible y ahorra mucho tiempo para los desarrolladores que se enfrentan con el problema de objeto/relación cada día.

Bibliografía

- Hibernate in Action, Christian Bauer; Gavin King.
- http://www.hibernate.org