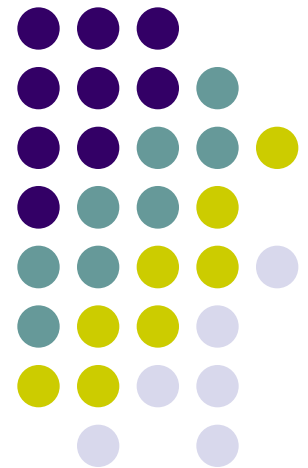


Persistencia de Objetos con Hibernate





Contexto actual

Actualmente en el mercado existen ya algunas bases de datos orientadas a objetos pero a nivel comercial no suelen ser una opción aún. Las empresas buscan el respaldo y la confianza de productos probados y consolidados a lo largo de un tiempo aceptable.

Las bases de datos relacionales aún están en su pleno apogeo, todavía librando algunas batallas respecto a su performance cuando de alto volumen transaccional se habla.



Contexto actual

De la misma manera los lenguajes orientados a objetos son la promesa y la alternativa mas elegida a la hora de nuevos desarrollos y reingenierías de sistemas. De esta forma en los escenarios actuales se suele trabajar con lenguajes orientados a objetos y se utilizan bases de datos relacionales, la combinación de estas tecnologías es un tema ampliamente discutido y de estas discusiones han surgido diferentes corrientes arquitectónicas.

Esta mezcla de tecnologías basadas en diferentes paradigmas plantea varios problemas importantes:

- 1. Se debe diseñar un sistema orientado a objetos y la base de datos debe ser creada en base a este modelo?*
- 2. Se debe crear un modelo de datos y diseñar un modelo de objetos dependiente de la estructura de datos?*



Posibles soluciones OO

- Clases de acceso.

En este tipo de estrategia y respecto al acceso a datos es comúnmente asociado con nombres como DAO (Data Access Object), DAC (Data Access Component). Para el acceso a datos esta estrategia delega las responsabilidades de acceder y manejar el código SQL (tanto en queries simples como en el acceso a Store procedures) a ciertas clases.

- ORM (Object Relation Mapper)

En este tipo de estrategia o de generación dinámica de SQL, se basa en archivos de mapeos. Es la estrategia más flexible, usa frameworks específicos diseñados para tener una mayor independencia con la DB. Consecuencias de esta estrategia: Es poco invasiva, nos permite trabajar casi sin cambios en nuestro modelo.

¿Qué es Hibernate?



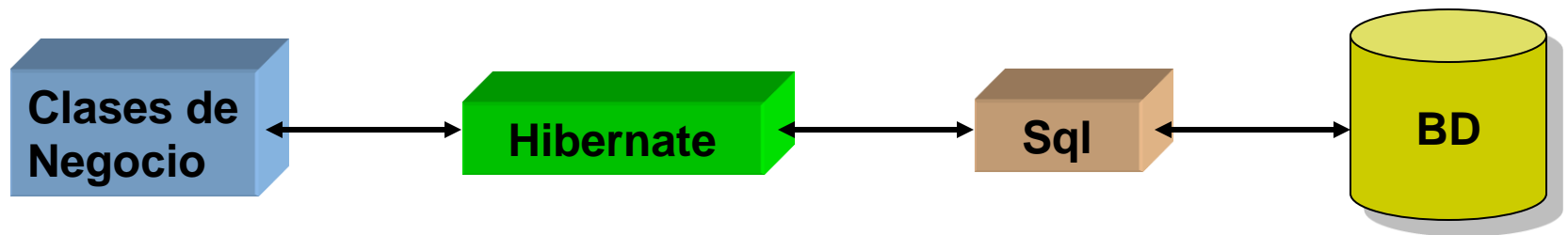
-Hibernate es un producto considerado una capa (persistence layer), framework, u object-relational mapper, tanto para JAVA (Hibernate) como para .NET (NHibernate) Este producto ya posee un nivel importante de maduración en el mercado en el orden de los 6 años. Es provisto por JBOSS, empresa que del grupo RED HAT (proveedora de una versión Linux con soporte para empresas) brindando una serie de características adicionales que le permiten mantener un liderazgo en este tipo de productos junto a los productos de otros grandes vendors como TOP Link (de Oracle), EJB (nativo de J2EE de Sun Microsystems) entre otros.

-Básicamente Hibernate permite desarrollar clases persistentes basándose en Java y en el paradigma OO (incluyendo asociaciones, herencias, polimorfismo, composición y las colecciones de framework Java) sin necesidad de trabajar directamente con la base de datos y con sql.

¿Qué es Hibernate?



Hibernate es el puente entre nuestra aplicación y la BBDD, sus funciones van desde la ejecución de sentencias SQL a través de JDBC hasta la creación, modificación y eliminación de objetos persistentes.



Características



Relaciones 1-1 / Relaciones 1-n / Relaciones n-m.

Inherente a las colecciones de objetos, permite mantener esta convención de diseño manteniendo independiente su implementación en la base de datos ya sea teniendo referencias distintas tablas o a tablas de relación en el caso de relaciones n-m.

Uni-direccionalidad, Bi-Direccionalidad en 1-n, n-m en todas las asociaciones. Incluye la posibilidad de configurar y administrar las referencias entre los objetos asociados.



Características

Lazy Load en relaciones 1-n, n-m.

Permite configurar de que manera de van a ir instanciando y accediendo los objetos miembros de colecciones en aquellos objetos que las contengan.

Updates y Deletes en Cascada.

Puede administrar el mantenimiento de algunas de los requerimientos de integridad referenciar en la base desde el modelo y viceversa.

Mapeo de herencias.

Con tres tipos de estrategias:

- 1) tabla única para toda la generalización
- 2) Mapeo uno a uno clase – tabla para toda la generalización con “foreign key” para vinculación de extensión de clases concretas
- 3) solo clases concretas a tablas.



Características

Estrategias de generación de IDs

Dado que es necesario mantener e identificar la identidad de los objetos aún persistidos con su relación de las tuplas que componen el estado interno en la base. Provee algoritmos de generación de Id: HiLo, Identidad, Secuencia. Soporte de Id compuestos, etc.

Soporte para múltiples bases.

Permite y facilita el intercambio de principales vendedores como DB2, SQL Server, Oracle entre otras sin necesidad de cambiar nada de nuestro moleo.

Lenguaje de consulta de objetos HQL propietario

Para casos de aplicaciones de mediano rango que necesiten mayor accesibilidad al código de consultas, es fácil de asimilar para el desarrollador.

¿Qué ventaja tiene Hibernate?



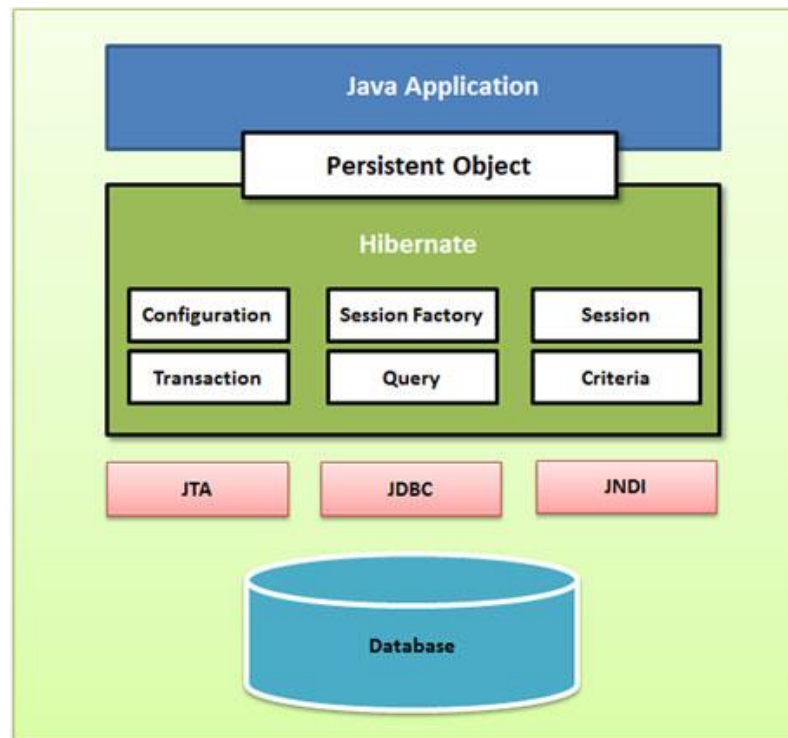
**Reducir los tiempos de desarrollo y
mantenimiento de nuestras
aplicaciones**

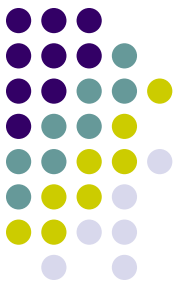




Arquitectura de Hibernate

La siguiente Figura muestra los roles de las interfaces Hibernate más importantes en las capas de persistencia y de negocio de una aplicación JEE. La capa de negocio está situada sobre la capa de persistencia, ya que la capa de negocio actúa como un cliente de la capa de persistencia.





Arquitectura de Hibernate

INTERFACES

- **Session** interfaz primaria en toda aplicación Hibernate. Instancia "poco pesada, su creación y destrucción es muy "barata". (Creación y destrucción de varias sesiones)
- **SessionFactory** permite obtener instancias *Session*. Típicamente hay una única *SessionFactory* para toda la aplicación. Una por cada base de datos.
- **Configuration** se utiliza para configurar y "arrancar" Hibernate.
- **Query** permite realizar peticiones a la base de datos y controla cómo se ejecuta. Se escriben en **HQL** o en el dialecto SQL nativo
- **Type** hace corresponder un tipo Java con un tipo de una columna de la base de datos.

Todas las propiedades persistentes de las clases persistentes, incluyendo las asociaciones, tienen un tipo Hibernate correspondiente. Esto lo hace altamente flexible y extendible. Incluso se permiten tipos definidos por el usuario (interfaz *UserType* y *CompositeUserType*).

Archivo de configuración inicial de Hibernate



Hibernate trabaja muy bien en ambientes de producción con la mayoría de las aplicaciones de J2EE y con servidores de aplicación.

Hibernate facilita la utilización de objetos tipo POJO (Plain Old Java Object) para aprovechar la capa de persistencia de las clases (propiedades accesibles en las clases a través de los métodos setter y setter).

Para comenzar a utilizar hibernate lo primero que tenemos que realizar es configurar con que base de datos queremos trabajar y cuales son nuestros archivos de mapeo (opcionalmente podemos utilizar anotaciones).

Archivo de configuración inicial de Hibernate



```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
```

```
<hibernate-configuration>
  <session-factory>
```

```
    <!-- Database connection settings -->
```

```
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
```

```
    <property name="connection.url">jdbc:mysql://localhost/testdb</property>
```

```
    <property name="connection.username">root</property>
```

```
    <property name="connection.password">root</property>
```

```
    <!-- SQL dialect -->
```

```
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

```
    <!-- Echo all executed SQL to stdout -->
```

```
    <property name="show_sql">>true</property>
```

```
    <!-- Drop and re-create the database schema on startup -->
```

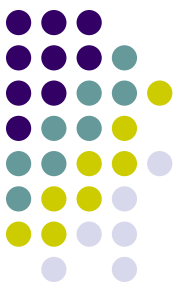
```
    <property name="hbm2ddl.auto">update</property>
```

```
    <mapping resource="edu/hibernate/bo/Cliente.hbm.xml"/>
```

```
    .....
```

```
  </session-factory>
```

```
</hibernate-configuration>
```



Mapeo de Objetos a la BD

- Clase Modelo

```
package de.gloegl.road2hibernate;

import java.util.Date;

public class Event {
    private String title;
    private Date date;
    private Long id;

    public Long getId() {
        return id;
    }

    private void setId(Long id) {
        this.id = id;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

- Archivo XML de Mapeo

Cada clase persistente tiene que tener un ID de atributo, esta propiedad es usada para distinguir los objetos persistentes, este concepto es llamado Database Identity

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

    <class name="de.gloegl.road2hibernate.Event" table="EVENTS">
        <id name="id" column="uid" type="long">
            <generator class="increment"/>
        </id>
        <property name="date" type="timestamp"/>
        <property name="title" column="eventtitle"/>
    </class>

</hibernate-mapping>
```

Hola Mundo con Hibernate



```
package edu.hibernate.bo;

public class Producto {

    private long id;
    private String descripcion;
    private double precio;

    public Producto() {}

    public Producto(String descripcion, double precio) {
        this.descripcion = descripcion;
        this.precio = precio;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public long getId() {
        return id;
    }

    public double getPrecio() {
        return precio;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public void setId(long id) {
        ..... etc
    }
}
```




Hola Mundo con Hibernate

Archivo Producto.hbm.xml en el mismo directorio de la clase

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="edu.hibernate.bo.Producto">

        <id name="id" column="id" >
            <generator class="native" />
        </id>

        <property name="descripcion" column="descripcion"/>
        <property name="precio" column="precio"/>

    </class>
</hibernate-mapping>
```



Hola Mundo con Hibernate

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
```

```
<hibernate-configuration>
  <session-factory>
```

```
    <!-- Database connection settings -->
```

```
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
```

```
    <property name="connection.url">jdbc:mysql://localhost/testdb</property>
```

```
    <property name="connection.username">root</property>
```

```
    <property name="connection.password">admin</property>
```

```
    <property name="connection.pool_size">5</property>
```

```
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

```
    <property name="show_sql">true</property>
```

```
    <!-- Drop and re-create the database schema on startup -->
```

```
    <property name="hbm2ddl.auto">update</property>
```

```
    <mapping resource="edu/hibernate/bo/Producto.hbm.xml"/>
```

```
    .....
```

```
  </session-factory>
```

```
</hibernate-configuration>
```



Trabajando con Hibernate



Guardar un nuevo objeto

```
// Crea una factoría de Sesiones a partir del archivo hibernate.cfg.xml
// configurar y crear una única instancia del SessionFactory
Configuration config = new Configuration();
SessionFactory sessionFactory = config.configure().buildSessionFactory();

Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

User user = new User();
user.setNombre("Juan");
user.setApellido("Perez");
session.save(user);

tx.commit();
session.close();
```



Recuperar un Objeto por OLD

```
Session session = sessionFactory.openSession();  
Transaction tx = session.beginTransaction();
```

```
int userID = 1234;
```

```
User user = (User) session.get(User.class, userID);
```

```
tx.commit();  
session.close();
```



Actualizar un Objeto

```
Session session = sessionFactory.openSession();  
Transaction tx = session.beginTransaction();
```

```
int userID = 1234;  
User user = (User) session.get(User.class, userID);
```

```
user.setNombre("Maria");  
session.update(user);
```

```
tx.commit();  
session.close();
```



Borrar un objeto

```
Session session = sessionFactory.openSession();  
Transaction tx = session.beginTransaction();  
  
int userID = 1234;  
User user = (User) session.get(User.class, userID);  
session.delete(user);  
  
tx.commit();  
session.close();
```



Consulta de objetos: HQL

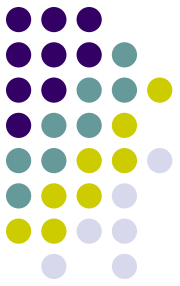
```
Session session = sessionFactory.openSession();
```

```
Query query = session.createQuery("from Producto as producto where  
    producto.precio > 10");
```

```
List<Producto> productos = query.list();
```

```
for (Producto p : productos) {  
    System.out.println(p.getId());  
    System.out.println(p.getDescripcion());  
    System.out.println(p.getPrecio());  
}
```

```
session.close();
```

Session.load vs Session.get

Session.load

- Retorna un proxy liviano y rápido (solo se carga la información on demand)
- `ObjectNotFoundException` si el objeto no existe en la BD

Session.get

- Siempre retorna el objeto completo
- Null si el objeto no existe en la BD



Pessimistic locking

```
Session session = sessionFactory.openSession();  
Transaction tx = session.beginTransaction();
```

```
int userID = 1234;
```

```
User user = (User) session.get(User.class, userID, LockMode.UPGRADE);
```

```
user.setNombre("Maria");  
session.update(user);
```

```
tx.commit();  
session.close();
```

LockMode.WRITE

LockMode.UPGRADE

LockMode.UPGRADE_NOWAIT

LockMode.READ



Optimistic locking

```
@Entity
public class Persona {
    ...
    @Version
    public int getVersion() { ... }
}
```

StaleObjectStateException!!!



Optimistic locking

```
Session session = sessionFactory.openSession();  
Transaction tx = session.beginTransaction();
```

```
int oldVersion = persona.getVersion();  
session.load(persona, persona.getId());
```

```
if (oldVersion != persona.getVersion()) throw new StaleObjectStateException();  
persona.setNombre("Juan");
```

```
tx.commit();  
session.close();
```



Batch Processing

```
Session session = SessionFactory.openSession();  
Transaction tx = session.beginTransaction();  
for ( int i=0; i<100000; i++ ) {  
    Empleado e = new Empleado(.....);  
    session.save(e);  
}  
tx.commit();  
session.close();
```

OutOfMemoryException!!!!!!!

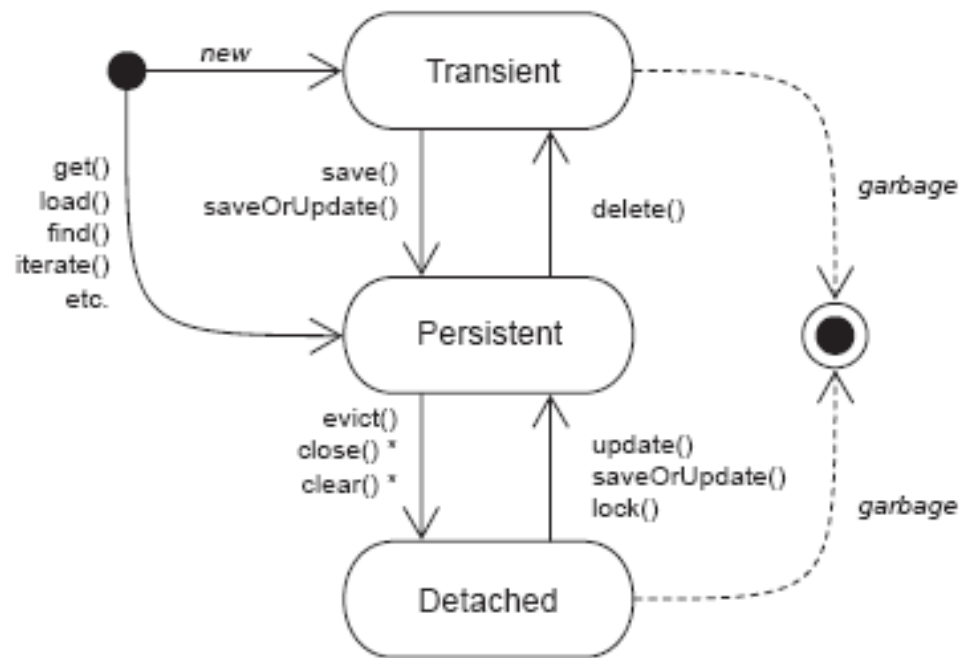


Batch Processing

```
Session session = SessionFactory.openSession();
Transaction tx = session.beginTransaction();
for ( int i=0; i<100000; i++ ) {
    Empleado e = new Emepleado(.....);
    session.save(e);
    if( i % 50 == 0 ) {
        session.flush();
        session.clear();
    }
}
tx.commit();
session.close();
```

hibernate.jdbc.batch_size=50

Ciclo de vida de un Objeto Persistente



* affects all instances in a Session



Conclusiones

- Consideramos que esta herramienta es una excelente elección para aplicaciones de hasta mediana complejidad y volumen transaccional
- Su implementación permite conservar y preservar el paradigma de objetos (POJOS tanto para Java como para .NET)
- Su curva de aprendizaje no es tan empinada con respecto al aprendizaje necesario que los desarrolladores deben aprender para su manejo.
- Posee mecanismos para manejo transaccional adecuados y su overhead de trabajo para incorporarlos es aceptable. Entre 10% y 30% mas lento que el sql directo
- Implica invertir un tiempo en el mapeo de las clases del modelo de dominio a las tablas.

Conclusiones



- Si bien el trabajo de mapeo consiste en tiempo, se aportan estrategias adecuadas para manejarlo
- Permite independencia de la base de datos con la que se opera.
- Permite alguna independencia de los IDE a utilizar interactuando con los más importantes.
- Es open source, permitiendo reducir los costos de desarrollo.
- Posee ya algunos años de evolución que avalan su aceptación en el mercado.
- Permite variar su configuración interna dando alternativas a cuestiones de performance, posibilita el HQL (lenguaje SQL propietario) aunque ninguna de estas dos opciones son recomendables, en términos de mantener el sistema puro o lo mas cercano posible al paradigma de objetos.