


| | |
|--|---|
|  el puente hacia el conocimiento | <h2 style="text-align: center;">ORM</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| <h3 style="text-align: center;">ORM (Mapeador Objeto-Relacion)</h3> | <p>Mientras que las aplicaciones trabajan con objetos, la información de dichos objetos se guarda en las estructuras relacionales de las bases de datos. Las herramientas ORM proporcionan mecanismos de automatización de la correspondencia entre objetos y las bbdd.</p> |
| <p>Por ejemplo: Si una aplicación tiene objetos <i>Cliente</i> que además de nombre y apellidos almacenan una colección de Telefono, a la hora de actualizar esta información en la bbdd, esta se deberá traducir en dos tablas Clientes y Telefonos que mantendrán una relación de clave externa.</p> <p>Sun proporciona una especificación llamada JDO (Java Data Objects) que establece un estándar sobre herramientas ORM. Las herramientas más destacadas que implementan dicha especificación son:</p> <ul style="list-style-type: none"> • Hibernate. • Apache ObjectRelationalBridge (OBJB). <p>Uno de los aspectos más destacados de JDO es su sencillez pues emplea clases java planas (POJO) para los objetos persistentes. La vinculación entre objetos y la bbdd relacional se realiza mediante archivos xml.</p> | |

| | |
|---|--|
|  el puente hacia el conocimiento | <h2 style="text-align: center;">Hibernate</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| <h3 style="text-align: center;">Hibernate</h3> | <p>Hibernate es una herramienta ORM de código abierto. (http://www.hibernate.org)</p> |
| <p>Pasos para trabajar con Hibernate</p> <ol style="list-style-type: none"> 1.- Crear la bbdd. 2.- Crear un Javabeen (Clase Java que implementa la interface Serializable, tiene un constructor sin parámetros y tiene metodos get y set para sus atributos). 3.- Crear un archivo de mapeo que indique la correspondencia entre el bean y una tabla de la bbdd. 4.- Crea un archivo de propiedades con la configuración JDBC para acceder a la bbdd. 5.- Incluir en el CLASSPATH las bibliotecas de Hibernate. 5.- Usar el Hibernate API para leer y escribir objetos persistentes. <p>A continuación se detallan estos pasos, excepto el primero que se supone conocido por el alumno.</p> | |

| | |
|--|--|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| <h2>El Javabeen</h2> | <p>Se creará un javabeen por cada tabla de la bbdd con la que queramos intercambiar información.</p> |
| <p>Un javabeen es una clase con métodos get y set para acceder a sus atributos, con un constructor sin parámetros y que implementa la interface Serializable. Cada atributo del javabeen representa a un campo de la bbdd.</p> <pre> package clientes; public class Cliente implements Serializable { private int dni; private String nombre; private String apellidos; private double saldo; public Cliente() {super();} public String getApellidos() {return apellidos;} public void setApellidos(String apellidos) {this.apellidos = apellidos;} ... public String toString() { return "dni:" + dni + " Nombre:" + nombre + " Apellidos:" + apellidos + " Saldo:" + saldo ;} } </pre> | |

| | |
|---|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| <h2>Archivo de Mapeo</h2> | <p>Establece la correspondencia entre los beans y las tablas de la bbdd. Se le asignará el nombre de la Clase seguido de la extensión "hbm.xml"</p> |
| <p><u>Cliente.hbm.xml</u></p> <pre> <?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="clientes.Cliente" table="CLIENTES"> <id name="dni" column="DNI"> <!--<generator class="native"/>--> </id> <property name="nombre" type="string" column="NOMBRE"/> <property name="apellidos"/> <property name="saldo" type="double" column="SALDO"/> </class> </hibernate-mapping> </pre> <p>Notas: type puede ser un tipo Hibernate, un tipo primitivo java o una clase java (incluir ruta de paquetes). Generator se incluye para campos autoincrementales.</p> | |

| | |
|---|---|
|  <p>el puente hacia el conocimiento</p> | <h1 style="text-align: center;">Hibernate</h1> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| <h2 style="text-align: center;">Configuración de acceso a la bbdd</h2> | <p>Indica los parámetros de acceso a la bbdd. También indica los archivos de mapeo a emplear. Debe ubicarse en el directorio raíz de nuestro proyecto y con el nombre hibernate.cfg.xml</p> |
| <pre><?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> <property name="connection.driver_class">com.mysql.jdbc.Driver</property> <property name="connection.url">jdbc:mysql://192.168.0.21/DATOS</property> <property name="connection.username">alumno1</property> <property name="connection.password">java</property> <property name="connection.pool_size">1</property> <property name="dialect">org.hibernate.dialect.MySQLDialect</property> <property name="current_session_context_class">thread</property> <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property> <property name="show_sql">true</property> <property name="hbm2ddl.auto">create</property> <mapping resource="clientes/Cliente.hbm.xml"/> </session-factory> </hibernate-configuration></pre> | |


| | |
|--|---|
|  <p>el puente hacia el conocimiento</p> | <h1 style="text-align: center;">Hibernate</h1> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| <h2 style="text-align: center;">Bibliotecas Hibernate</h2> | <p>Se deben incluir en el classpath los jar con las clases Hibernate. Estas se pueden obtener de la carpeta lib de las distribución (excepto hibernate3.jar que está en el raíz).</p> |
| <h3 style="text-align: center;">Bibliotecas mínimas Hibernate</h3> <div style="text-align: center;"> <p>antlr.jar</p> <p>asm-attrs.jar</p> <p>asm.jar</p> <p>cglib.jar</p> <p>commons-collections.jar</p> <p>commons-logging.jar</p> <p>dom4j.jar</p> <p>log4j.jar</p> <p>hibernate3.jar</p> <p>jta.jar</p> </div> <p>Según la distribución, los nombres de estos archivos pueden incluir al final un número referente a la versión.</p> | |


| | |
|--|--|
|  <p>el puente hacia el conocimiento</p> | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| <h2>Log Hibernate</h2> | <p>Hibernate puede mostrar diferentes niveles de log, en función de la configuración expresada en el archivo log4j.properties.</p> |
| <p><u>log4j.properties para no mostrar ningún mensaje excepto errores</u></p> <pre>log4j.rootCategory=ERROR, A1 log4j.appender.A1=org.apache.log4j.ConsoleAppender log4j.appender.A1.layout=org.apache.log4j.PatternLayout log4j.appender.A1.layout.ConversionPattern=%-5p - %m%n</pre> <p><u>log4j.properties para mostrar todos los mensajes</u></p> <pre>log4j.rootCategory=INFO, A1 log4j.appender.A1=org.apache.log4j.ConsoleAppender log4j.appender.A1.layout=org.apache.log4j.PatternLayout log4j.appender.A1.layout.ConversionPattern=%-5p - %m%n</pre> <p>Notas: Para que esto funcione se ha de tener en el classpath el archivo log4j.jar Para no mostrar ningún mensaje en lugar de ERROR, poner FATAL</p> | |


| | |
|---|--|
|  <p>el puente hacia el conocimiento</p> | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| <h2>Api Hibernate (SessionFactory I)</h2> | <p>El primer paso para trabajar con Hibernate es crear un SessionFactory. Es necesario un SessionFactory por cada base de datos con la que se vaya a trabajar.</p> |
| <p>El SessionFactory nos suministra objetos Session. Cada objeto Session constituye un contexto que gestiona la persistencia de los objetos que obtengamos a través de él.</p> <p>La obtención de un SessionFactory es un proceso que consume una cantidad importante de recursos, por lo que se recomienda realizarlo una sola vez (Ver clase de utilidad de la transparencia siguiente). Por el contrario los objetos Session no consumen recursos, ya que no establecen conexión a la base de datos hasta que no se va a ejecutar una consulta.</p> <p>La obtención de un SessionFactory se realiza de la siguiente forma:</p> <pre>SessionFactory sessionFactory= new Configuration().configure().buildSessionFactory();</pre> <p>Los objetos Configuration con el método configure() lee la configuración de hibernate. Si no se indica por parámetro el archivo, por defecto se buscará el hibernate.cfg.xml.</p> | |

| | |
|--|--|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Api Hibernate (SessionFactory II) | <p>El primer paso para trabajar con Hibernate es crear un SessionFactory. Para ello crearemos una clase HibernateUtil que implemente el patrón Singleton suministrando siempre un mismo objeto SessionFactory.</p> |
| <pre> package util; import org.hibernate.*; import org.hibernate.cfg.*; public class HibernateUtil { private static SessionFactory sessionFactory = null; static { try { // Crea un SessionFactory a partir de hibernate.cfg.xml sessionFactory = new Configuration().configure().buildSessionFactory(); } catch (Throwable ex) { System.out.println("No se pudo iniciar la sesión Hibernate: " + ex); } } public static SessionFactory getSessionFactory() { return sessionFactory; } } </pre> <p>El SessionFactory nos suministra objetos Session. Cada objeto Session constituye un contexto que gestiona la persistencia de los objetos que obtengamos a través de él.</p> | |

| | |
|---|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Api Hibernate (Transaction) | <p>Las modificaciones que se realicen sobre los objetos persistentes, deben ser englobados siempre dentro de una transacción, representada por un objeto Transaction.</p> |
| <p>Un objeto Transaction se obtiene a partir de un objeto Session, iniciando así la transacción:</p> <pre>Transaction transaction = session.beginTransaction();</pre> <p>Cuando los objetos persistentes han sido modificados, estos permanecen en el contexto de persistencia, pero no se tiene garantía de que los nuevos valores se hayan enviado a la bbdd hasta que no se confirma la transacción (commit):</p> <pre>transaction.commit();</pre> <p>Finalmente, se puede cerrar la sesión. En ese momento el contexto de persistencia desaparece y los objetos persistentes se encuentran desvinculados (deattached) a los datos de la bbdd:</p> <pre>session.close();</pre> | |

| | |
|--|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Api Hibernate (Inserción) | <p>A continuación se muestra un ejemplo de inserción de un registro en la tabla de clientes. Para ello se emplea Hibernate y un objeto de la clase Cliente.</p> |
| <pre> package clientes; import org.hibernate.Session;import util.HibernateUtil; public class TestInsertarCliente { public static void main(String[] args) { Session session = HibernateUtil.getSessionFactory().getCurrentSession(); Transaction transaction = session.beginTransaction(); Cliente c = new Cliente(); c.setDni(12422); c.setNombre("Lucas"); c.setApellidos("Ruiz"); c.setSaldo(332); session.save(c); transaction.commit(); HibernateUtil.getSessionFactory().close(); } } </pre> | |

| | |
|---|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Api Hibernate (Consulta) | <p>A continuación se muestra un ejemplo de listado de datos de la tabla Clientes. El filtrado se realiza mediante HQL(Hibernate Query Language) que consulta sobre objetos no sobre tablas.</p> |
| <pre> package clientes; import java.util.List; import org.hibernate.Session;import util.HibernateUtil; public class TestListarCliente { public static void main(String[] args) { Session session = HibernateUtil.getSessionFactory().getCurrentSession(); Transaction transaction = session.beginTransaction(); List lista = session.createQuery("from Cliente").list(); transaction.commit(); HibernateUtil.getSessionFactory().close(); System.out.println("Num Clientes Encotrados: " + lista.size()); for (int i = 0; i < lista.size(); i++) { System.out.println(lista.get(i));} } } </pre> | |

| | |
|---|--|
|  el puente hacia el conocimiento | <h1 style="text-align: center;">Hibernate</h1> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| Api Hibernate: Buscar por Id con session.get | <p>El método get Realiza la consulta instantaneamente y retorna un objeto de la clase "Cliente" si se ha encontrado y null si no se ha encontrado.</p> |
| <pre> Session session = HibernateUtil.getSessionFactory().getCurrentSession(); Transaction tx = session.beginTransaction(); // Si no existe el cliente, get retorna null y load produce excepcion Cliente cliente = (Cliente) session.get(Cliente.class,2001); System.out.println("Como ya se ha hecho la consulta, hago commit"); tx.commit();// Modo thread: El commit cierra automáticamente la session. HibernateUtil.getSessionFactory().close(); if(cliente==null) { System.out.println("No existe ese objeto"); } else { System.out.println("El objeto es de la clase: " + cliente.getClass().getName()); System.out.println("Cliente obtenido: " + cliente); } </pre> | |

| | |
|--|---|
|  el puente hacia el conocimiento | <h1 style="text-align: center;">Hibernate</h1> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| Api Hibernate: Buscar por Id con session.load | <p>Si se trabaja en modo Lazy, no realiza la consulta inicialmente y retorna un objeto "proxi" de una clase heredera de Cliente que inicialmente no contiene ningún dato. Al leerse alguno de sus atributos es cuando se realiza la consulta.</p> |
| <p>Cuando se intente leer alguno de los datos del objeto mediante sus métodos get o mediante el toString, en ese momento será cuando se realice la consulta (en ese momento la session debe seguir abierta, si no dará una excepcion). Si existe un registro, el objeto quedará correctamente inicializado, si no existe, producirá una ObjectNotFoundException.</p> <pre> Session session = HibernateUtil.getSessionFactory().getCurrentSession(); Transaction tx = session.beginTransaction(); Cliente cliente = (Cliente) session.load(Cliente.class,2001); System.out.println("He obtenido un objeto hueco, todavía no puedo hacer commit"); System.out.println("El objeto es de la clase: " + cliente.getClass().getName()); System.out.println("Como ahora voy a usar los datos del cliente, " + " es ahora cuando se hace la consulta (Modo Vago, Lazy)"); try { System.out.println("Cliente obtenido: " + cliente); } catch (ObjectNotFoundException e) { System.out.println("No hay ningún objeto con ese id"); } System.out.println("Ahora, despues de la consulta es cuando puedo hacer commit"); tx.commit();// Modo thread: El commit cierra automáticamente la session. HibernateUtil.getSessionFactory().close(); </pre> | |



Hibernate

Profesor Vladimir Bataller

Api Hibernate:
Buscar por Id
resumen get/load

Resumen de Get y Load

| | lazy | No-lazy |
|------|---|--|
| Get | Realiza la consulta al ejecutar el método get. | Realiza la consulta al ejecutar el método get. |
| Load | Load retorna un objeto "proxi" inicialmente vacío. Realiza la consulta cuando se va a leer algún atributo. | Realiza la consulta al ejecutar el método load y retorna un objeto Cliente igual que el Get. |

Get: Si no existe el registro retorna null
Load: Si no existe el registro cuando se va a hacer la consulta, produce una `ObjectNotFoundException`



Hibernate

Profesor Vladimir Bataller

Api Hibernate
(Borrado)


Para borrar un registro de la tabla, primero se deberá obtener un objeto persistente asociado a dicho registro. Esto se hace con el método load, indicando la clase y el valor de la clave primaria. Posteriormente se borrará con el método delete.

```


package clientes;
import java.util.List;
import org.hibernate.Session;import util.HibernateUtil;
public class TestBorrarCliente
{
    public static void main(String[] args)
    {
        Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        Transaction transaction = session.beginTransaction();
        // Obtención de un Cliente por su Id
        Cliente cliente=(Cliente) session.load(Cliente.class, new Integer(78374));
        // Borrado
        session.delete(cliente);
        transaction.commit();
        HibernateUtil.getSessionFactory().close();
    }
}


```

El método **get** hace lo mismo que **load**, pero retorna null en lugar de Excepcion si no existe. Además get ejecuta la consulta inmediatamente, independientemente del modo lazy.


| | |
|--|---|
|  el puente hacia el conocimiento | <h1 style="text-align: center;">Hibernate</h1> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| Api Hibernate (Edición) | <p>A continuación se muestra un ejemplo de listado de datos de la tabla Clientes. El filtrado se realiza mediante HQL(Hibernate Query Language) que consulta sobre objetos no sobre tablas.</p> |
| <pre> package clientes; import org.hibernate.Session;import util.HibernateUtil; public class TestEditarCliente { public static void main(String[] args) { Session session = HibernateUtil.getSessionFactory().getCurrentSession(); Transaction transaction = session.beginTransaction(); // Obtención de un cliente por su Id. Cliente cliente=(Cliente) session.load(Cliente.class, new Integer(78374)); // Modificación de sus atributos cliente.setApellidos("Valiente Ochoa"); // Guardar los cambios session.update(cliente); transaction.commit(); HibernateUtil.getSessionFactory().close(); } } </pre> | |


| | |
|---|---|
|  el puente hacia el conocimiento | <h1 style="text-align: center;">Hibernate</h1> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| Api Hibernate (saveOrUpdate) | <p>En ocasiones no se conoce si un registro existe en la bbdd. Por lo tanto frente a nuevos datos se desconoce si hay que actualizarlos o insertarlo como un registro nuevo. Para eso dispone de un método que realiza la operación necesaria "saveOrUpdate".</p> |
| <pre> package clientes; import org.hibernate.Session;import util.HibernateUtil; public class TestEditarCliente { public static void main(String[] args) { Session session = HibernateUtil.getSessionFactory().getCurrentSession(); Transaction transaction = session.beginTransaction(); // Obtención de un cliente por su Id. Cliente cliente=(Cliente) session.load(Cliente.class, new Integer(78374)); // Cliente cliente = new Cliente(); cliente.setDni(22544565); // Modificación de sus atributos cliente.setApellidos("Valiente Ochoa");cliente.setNombre("Daniel"); // Guardar los cambios o insertar, si el objeto no existe session.saveOrUpdate(cliente); transaction.commit(); HibernateUtil.getSessionFactory().close(); } } </pre> | |

| | |
|--|--|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Caché de sesión | <p>Los objetos persistentes están asociados a un espacio de memoria de una sesión Hibernate.</p> |
| | <ol style="list-style-type: none"> 1.- En una sesión, no puede haber dos objetos de la misma clase con el mismo identificador ya que producirá una NonUniqueObjectException. 2.- Un objeto se puede desvincular de la sesión pasándolo como argumento al método evict(): <code>session.evict(Object o);</code> 3.- Para conocer si un objeto está vinculado a la sesión mediante el método contains(): <code>session.contains(Object o);</code> 4.- Para desvincular todos los objetos del caché de una sesión se empleará el método clear(): <code>session.clear();</code> 5.- Los cambios realizados en los objetos del caché de sesión se envían a la bbdd en el commit de una transacción o mediante el método flush(): <code>session.flush();</code> 6.- Para leer de nuevo un objeto de la bbdd y ver los cambios que haya sufrido, se empleará el método refresh(), con el objeto a refrescar entre parentesis. <code>session.refresh(Object o)</code> |


| | |
|--|--|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Criterios de uso de sesiones y transacciones | <p>Los objetos persistentes están asociados a un espacio de memoria de una sesión Hibernate. Una transacción, mientras está abierta tiene bloqueados recursos sobre la base de datos.</p> |
| | <p>Antipatrón "sesión por operación": Crear una sesión y una transacción por cada operación que se quiera hacer sobre los datos de la aplicación.</p> <p>Patrón "sesión por petición": En aplicaciones cliente/servidor, es una buena práctica no alargar una sesión, más tiempo que lo que dura el procesamiento de una petición de la aplicación cliente al servidor.</p> <p>En cualquier momento se puede acceder a la sesión que está actualmente abierta mediante el método getSession del Session Factory. Además esa sesión estará comprendida en la transacción actual de la bbdd.</p> <p>Antipatrones "sesión por sesión de usuario" y "sesión por aplicación". El mantenimiento de sesión mientras el usuario piensa debe ser implementado por la capa de aplicación y no por la capa de persistencia.</p> |

| | |
|---|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Demarcación de transacciones(I) | <p>Aunque solamente realicemos operaciones de lectura, se recomienda el empleo de transacciones. Se desaconseja el empleo de muchas pequeñas transacciones seguidas ya que esto perjudicará al rendimiento.</p> |
| <p>La finalización de una sesión comprende cuatro tareas:</p> <ol style="list-style-type: none"> 1.- Hacer flush de la sesión: Esto conlleva que se envíen a la bbdd las consultas pendientes de enviar para sincronizar los objetos persistentes de la sesión con la bbdd. 2.- Hacer commit de la transacción: Consolidar en la bbdd los cambios enviados. 3.- Cerrar la session: Desvincular los objetos persistentes con la bbdd mediante Hibernate. 4.- Gestionar las excepciones: Gestionar las excepciones producidas por inconsistencias de los datos de la sesión y la bbdd durante el tiempo que los objetos persistentes no han estado sincronizados. | |

| | |
|---|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Demarcación de transacciones(II) | <p>El proceso expuesto en la transparencia anterior, tendría, en un entorno de transacciones manejadas programáticamente (no por el contexto) una estructura más o menos como la que aquí se muestra.</p> |
| <pre> Session session = sessionFactory.openSession(); Transaction transaction = null; try { transaction = session.beginTransaction(); ... tx.commit(); }catch (RuntimeException e) { if (transaction != null) transaction.rollback(); throw e; <i>// o mostrar un mensaje.</i> } finally { session.close();} Otra alternativa más flexible sería: try { sessionFactory.getCurrentSession().beginTransaction(); ... sessionFactory.getCurrentSession().getTransaction().commit(); }catch (RuntimeException e) { sessionFactory.getCurrentSession().getTransaction().rollback(); throw e; <i>// o mostrar un mensaje.</i> } </pre> | |

| | |
|---|--|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Concurrencia Optimista con control versión gestionado por la aplicación | <p>Si a lo largo de la conversación, se emplean diferentes sesiones, se puede pretender actualizar un atributo de un objeto de una sesión anterior. En este caso se debe comprobar si la versión del objeto es la misma que la que hay en la bbdd. Para ello se emplea un campo en la bbdd que se incrementa cada vez que se hace un cambio.</p> |
| <pre>// obj es una instancia cargada en una sesión anterior session = sessionFactory.openSession(); Transaction t = session.beginTransaction(); int versionAntigua = obj.getVersion(); session.load(obj, obj.getKey()); // cargar la versión actual del objeto (resincronizo) if (versionAntigua!=obj.getVersion) throw new StaleObjectStateException(); obj.setXxxx("nuevo valor"); t.commit(); session.close();</pre> <p>El atributo versión se mapea en Hibernate mediante la etiqueta <version>, e hibernate incrementará automáticamente su valor en cada actualización que se haga del objeto modificado sobre la base de datos (como muy tarde hacer flush).</p> | |

| | |
|---|--|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Concurrencia Optimista con control versión automático y sesión extendida | <p>Si se opta por mantener abierta la sesión durante toda una conversación, hibernate en los momentos de flush, comprueba las versiones, emitiendo una excepción si la antigua y la que hay en la bbdd no coinciden.</p> |
| <pre>//obj es un objeto cargado anteriormente en la larga sesión Transaction t = session.beginTransaction();// Nueva conexion jdbc y transacción obj.setXxxxxx("nuevo valor"); session.flush(); // Aquí s e puede producir una Excepción por inconsistencia de datos t.commit(); session.close();</pre> <p>En este caso Hibernate puede hacer automáticamente el control del valor de la versión porque la sesión ha permanecido abierta. Por lo que tiene constancia de los valores iniciales del objeto y los puede comparar con los que hay en la bbdd en el instante de hacer cambios desde la aplicación.</p> <p>Para poder trabajar de esta forma es necesario:</p> <ol style="list-style-type: none"> 1.- Poner la propiedad current_session_context_class de configuración del hibernate.cfg.xml con el valor managed en lugar de thread. 2.- Abrir la sesión con: <code>sessionFactory.openSession()</code> en lugar de con <code>sessionFactory.getCurrentSession()</code>. 3.- Cerra la sesión con <code>session.close()</code> después del último commit. | |



Hibernate

Profesor Vladimir Bataller

Asociaciones n a 1

Hibernate permite trabajar con las relaciones entre las tabla y trasladarla a los objetos persistentes. En este caso se obtiene el objeto Provincia asociado a cada Cliente.

TABLAS

CLIENTES
dni
nombre
apellidos
saldo
id_provincia

PROVINCIAS
id_provincia
provincia


Diagram showing a 1 to n relationship between CLIENTES and PROVINCIAS. CLIENTES has attributes: dni, nombre, apellidos, saldo, id_provincia. PROVINCIAS has attributes: id_provincia, provincia.

CLASES

class Cliente
dni:int
nombre:String
apellidos:String
saldo: double
provincia:Provincia

class Provincia
idProvincia:int
provincia:String

Diagram showing the mapping between the tables and the classes. The Cliente class has attributes: dni:int, nombre:String, apellidos:String, saldo: double, provincia:Provincia. The Provincia class has attributes: idProvincia:int, provincia:String.



Hibernate

Profesor Vladimir Bataller

Asociaciones n a 1


Hibernate permite trabajar con las relaciones entre las tabla y trasladarla a los objetos persistentes. En este caso se obtien el objeto Provincia asociado a cada Cliente.


- Los atributos de la clase Provincia (con sus getters y setters) son:
int idProvincia;
String **provincia**;
- El elemento class del archivo de mapeo de provincia (Provincia.hbm.xml) será:

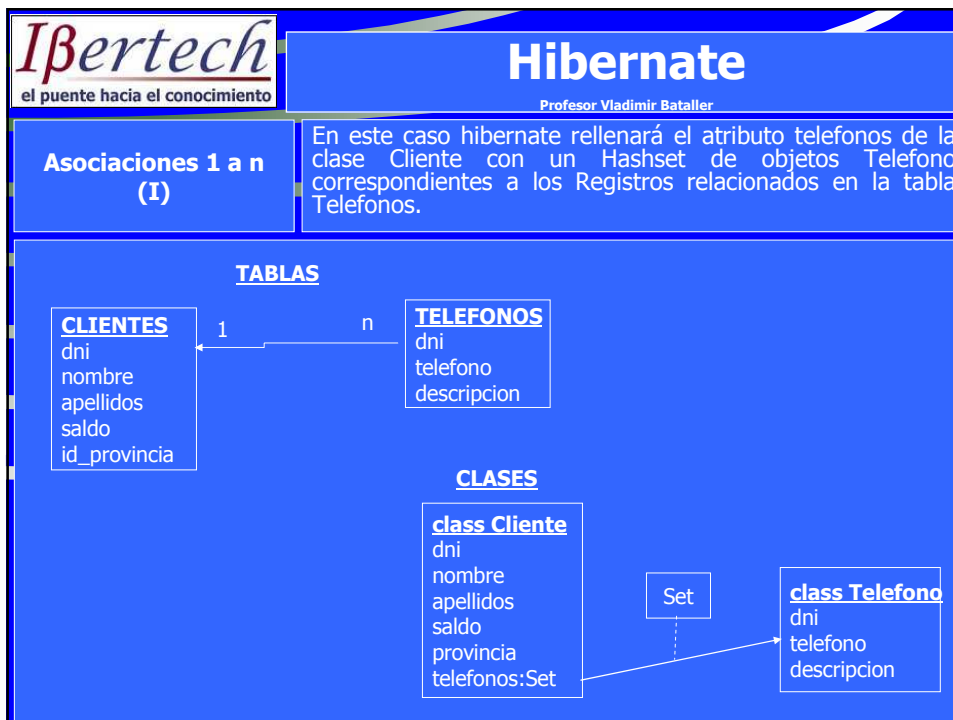
```
<class name="clientes.Provincia" table="PROVINCIAS">
  <id name="idProvincia" column="ID_PROVINCIA"/>
  <property name="provincia"/>
</class>
```
- Al archivo hibernate.cfg.xml se le agregará la referencia al mapeo de Provincia:


```
<mapping resource="clientes/Provincia.hbm.xml"/>
```
- A la clase Cliente se le agregará un atributo (con sus correspondientes getters y setters):
Provincia **provincia**;
- Agregar a Cliente.hbm.xml dentro del elemento class:

```
<many-to-one name="provincia" column="ID_PROVINCIA" class="clientes.Provincia"/>
```
- Modificar el toString de Cliente para que muestre además su atributo provincia.

| | |
|--|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Asociaciones n a 1 (atributo lazy)(I) | <p>Hibernate maneja por defecto las relaciones en modo "lazy". Es decir, no hace la consulta sobre la tabla Provincias hasta que no pedimos la provincia de un cliente.</p> |
| <p>El modo lazy o no-lazy se especifica en el atributo lazy="false proxy no-proxy" en la etiqueta many-to-one del mapeo de clientes.</p> <p>Cientes.hbm.xml</p> <pre> <many-to-one name="provincia" column="ID_PROVINCIA" class="clientes.Provincia" lazy="false"/> ó <many-to-one name="provincia" column="ID_PROVINCIA" class="clientes.Provincia" lazy="proxy"/> ó <many-to-one name="provincia" column="ID_PROVINCIA" class="clientes.Provincia" lazy="no-proxy"/> </pre> <p>Modo no-lazy (lazy=false): Una vez obtenidos los clientes con los que vamos a trabajar, podemos cerrar la sesión, porque ya se han obtenido los datos de provincia de esos clientes. Los objetos cliente tienen todos los valores de Provincia desde el principio.</p> <p>Modo lazy (proxy o no-proxy): No se puede cerrar la session hasta que que no se termine de trabajar con el último cliente si es que la aplicación va pedir datos de sus provincias relacionadas, ya que esos datos se piden a la bbdd en el momento justo.</p> | |

| | |
|---|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Asociaciones n a 1 (atributo lazy)(II) | <p>Hibernate maneja por defecto las relaciones en modo "lazy". Es decir, no hace la consulta sobre la tabla Provincias hasta que no pedimos la provincia de un cliente.</p> |
| <p>Nota: En modo lazy hay que recordar que las consultas sql lazy son disparadas por los métodos get y de la clase del extremo de la relación. Si se accede a los atributos directamente se obtendrá null ya que de esta forma no se dispara la consulta SQL. Conclusión: Se recomienda poner los atributos private para evitar accesos directos que no disparen las consultas SQL en modo lazy.</p> | |





Ibervtech
el puente hacia el conocimiento

Hibernate

Profesor Vladimir Bataller

**Asociaciones 1 a n
(II)**

En este caso hibernate rellenará el atributo telefonos de la clase Cliente con un Hashset de objetos Telefono correspondientes a los Registros relacionados en la tabla Telefonos.


Los atributos de la clase Telefono (con sus getters y setters) son:
int dni; String telefono; String descripcion;


La clase Telefono debe implementar la interface Serializable y además como su clave primaria es compuesta por dni y telefono, debemos sobrescribir el método hashCode() heredado de Object:

```
public int hashCode()
{
    int hash = 31 * 7 + dni;
    return 31 * hash + telefono.hashCode();
}
```

El elemento class del archivo de mapeo de telefonos (Telefono.hbm.xml) será:

```
<class name="clientes.Telefono" table="TELEFONOS">
  <composite-id>
    <key-property name="dni" column="DNI"/>
    <key-property name="telefono" column="TELEFONO"/>
  </composite-id>
  <property name="descripcion"/>
</class>
```

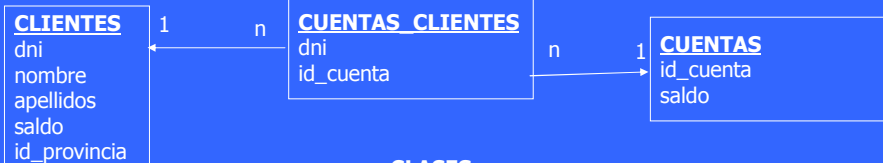
| | |
|---|---|
|  el puente hacia el conocimiento | <h2 style="text-align: center;">Hibernate</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| Asociaciones 1 a n (III) | <p>En este caso hibernate rellenará el atributo telefonos de la clase Cliente con un Hashset de objetos Telefono correspondientes a los Registros relacionados en la tabla Telefonos.</p> |
| <p>Al archivo hibernate.cfg.xml se le agregará la referencia al mapeo de Telefono:</p> <pre><mapping resource="clientes/Telefono.hbm.xml"/></pre> <p>A la clase Cliente se le agregará un atributo (con sus correspondientes getters y setters):</p> <pre>private Set telefonos = new HashSet();</pre> <p>Agregar a Clientes.hbm.xml dentro del elemento class:</p> <pre><set name="telefonos" table="TELEFONOS" fetch="select"> <key column="DNI"/> <one-to-many class="clientes.Telefono"/> </set></pre> <p>Nota: Si el atributo fetch toma el valor select (valor por defecto), hará una consulta para obtener los datos del Cliente y una consulta para obtener los teléfonos relacionados. Si por el contrario fetch toma el valor join, hibernate obtiene con una sola consulta (left outer join) los datos del cliente y sus teléfonos. Esto también se puede forzar en tiempo de ejecución mediante HQL:</p> <pre>List lista = session.createQuery("from Cliente as c left join fetch c.telefonos").list();</pre> | |

| | |
|--|--|
|  el puente hacia el conocimiento | <h2 style="text-align: center;">Hibernate</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| Asociaciones 1 a n (IV) | <p>Ejemplo de inserción de un telefono a un cliente.</p> |
| <pre>// Obtenemos un cliente Cliente cliente = (Cliente) session.get(Cliente.class, new Integer(1781)); // Creamos un Telefono Telefono tel =new Telefono(); tel.setTelefono("6" + (int)(1000000*Math.random())); tel.setDescripcion("Movil"); tel.setDni(1781); System.out.println("Se va a guardar: " + tel.getTelefono()); session.save(tel); // Enviamos la consulta SQL a la bbdd. session.flush(); // Leemos el cliente de la bbdd para que tenga el nuevo teléfono en su set. session.refresh(cliente); tx.commit(); HibernateUtil.getSessionFactory().close(); System.out.println("Insertado: " + cliente); System.out.println(cliente.imprimeTelefonos());</pre> | |

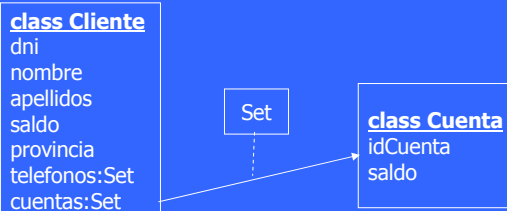
Asociaciones n a n (I)

Un cliente puede tener varias cuentas y una cuenta puede tener asociados varios clientes. Esto se expresa en la tabla CUENTAS_CLIENTES

TABLAS



CLASES



Asociaciones n a n (II)

Un cliente puede tener varias cuentas y una cuenta puede tener asociados varios clientes. Esto se expresa en la tabla CUENTAS_CLIENTES

Los atributos de la clase Cuenta (con sus getters y setters) son:

```
int idCuenta;
double saldo;
```

El elemento class del archivo de mapeo de Cuenta (Cuenta.hbm.xml) será:

```
<class name="clientes.Cuenta" table="CUENTAS">
  <id name="idCuenta" column="ID_CUENTA"/>
  <property name="saldo"/>
</class>
```

Al archivo hibernate.cfg.xml se le agregará la referencia al mapeo de Cuenta:

```
<mapping resource="clientes/Cuenta.hbm.xml"/>
```

Campos de la tabla intermedia

Agregar a Clientes.hbm.xml dentro del elemento class:

```
<set name="cuentas" table="CUENTAS_CLIENTES">
  <key column="DNI"/>
  <many-to-many column="ID_CUENTA" class="clientes.Cuenta"/>
</set>
```

A la clase Cliente se le agregará un atributo (con sus correspondientes getters y setters):

```
private Set cuentas = new HashSet();
```

Asociaciones n a n asignar nuevos elementos

En el siguiente ejemplo se asigna una nueva cuenta a un cliente, si ya estaba asignada no da error.

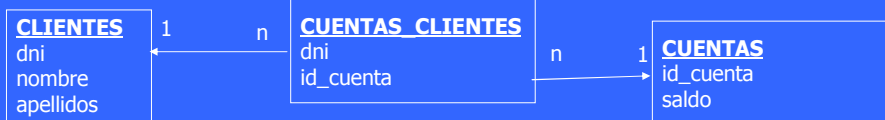
Hibernate detecta cuando las colecciones que representan asociaciones cambian y actualiza la bbdd de acuerdo a estos cambios.

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
Cliente cliente=(Cliente) session.load(Cliente.class, new Integer(1));
Cuenta cuenta=(Cuenta) session.load(Cuenta.class, new Long(19));
cliente.getCuentas().add(cuenta);
System.out.println("Vinculo Agregado");
session.getTransaction().commit();
HibernateUtil.getSessionFactory().close();
```

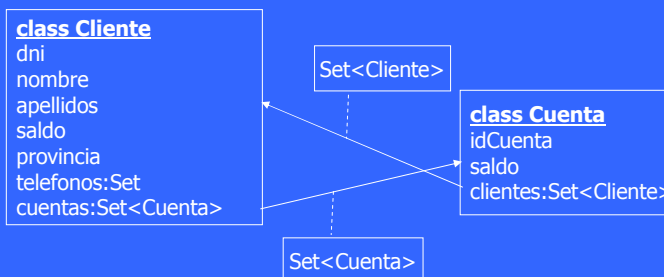
Asociaciones n a n bidireccionales (I)

En las asociaciones n a n, ambas clases pueden tener una colección con los elementos relacionados en la otra tabla.

TABLAS



CLASES



Asociaciones n a n bidireccionales (II)

Si queremos además de obtener las cuentas de un cliente, los clientes de una cuenta deberemos trabajar con una asociación bidireccional.

Agregar a la clase Cuenta un atributo (con sus getters y setters) para guardar la colección de Clientes asociados:

```
private Set clientes=new HashSet();
```

El archivo Cliente.hbm.xml queda como en la relacion unidireccional:

```
<set name="cuentas" table="CUENTAS_CLIENTES">
    <key column="DNI"/>
    <many-to-many column="ID_CUENTA" class="clientes.Cuenta"/>
</set>
```

Agregar al archivo Cuenta.hbm.xml el bloque set que representa la colección de Clientes asociados y el atributo inverse = true;

```
<set name="clientes" table="CUENTAS_CLIENTES" inverse="true">
    <key column="ID_CUENTA"/>
    <many-to-many column="DNI" class="clientes.Cliente"/>
</set>
```

Observese que en una y solo una de las dos set se indica inverse="true". Esto debe ser así y además implica que cuando se agregue un objeto a la colección inverse, no se traducirá a la bbdd, sino que será la otra colección la que indique los registros de la tabla intermedia.

Asociaciones n a n bidireccionales (III)

Si queremos además de obtener las cuentas de un cliente, los clientes de una cuenta deberemos trabajar con una asociación bidireccional.

Se recomienda hacer protected los métodos de get y set acceso a las colecciones.

```
protected Set getClientes() { return clientes;}
```

```
protected void setClientes(Set clientes) { this.clientes = clientes;}
```

Para añadir un cliente a la colección clientes de una cuenta se hará de la siguiente forma:


```
public void addCliente(Cliente cliente)
{
    this.getClientes().add(cliente);
    cliente.getCuentas().add(this);
}
```


```
public void removeCliente(Cliente cliente) {
    this.getClientes().remove(cliente);
    cliente.getCuentas().remove(this);
}
```

En la Clase Cliente habrá que hacer lo mismo recíprocamente.


| | |
|--|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| HQL (Hibernate Query Language) | <p>Es una de las opciones que Hibernate nos proporciona para realizar consultas. Estas se realizan sobre objetos no sobre Tablas.</p> |
| <p>Pueden declararse alias de los objetos como en SQL. A las propiedades de los objetos se accede con la sintaxis java. Puede o no anteponerse a una clase su ruta de paquetes ya que por defecto está activado el "auto-import".</p> <pre>from clientes.Telefono t where t.descripcion like '%Mov%'</pre> <p>Pueden retornarse varios tipos de objetos simultaneamente (producto cartesiano), para ello se retornará por cada fila una matriz de Object.</p> <pre>select c, t from Cliente c, Telefono t where t.dni = c.dni</pre> <p>Pueden emplearse Funciones de agregado:</p> <pre>select max(c.saldo) as max, min(c.saldo) as min, count(*) as n from Clientes c</pre> <p>Consultas sobre Tablas cuya clave primaria es compuesta:</p> <pre>from clientes.Telefono t where t.id.dni = 1125 and t.id.telefono = '91456'</pre> | |

| | |
|--|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Hibernate Criteria(I) | <p>Además de HQL, Hibernate dispone de otra modalidad de expresar las consultas: Criteria. Consiste en una serie de Clases con las cuales se pueden expresar las condiciones.</p> |
| <p>Los objetos Criteria representan a una consulta sobre una determinada clase. Con el metodo add, se pueden agregar Criterios de consulta, obtenidos mediante objetos Restrictions.</p> <pre>// Creación de un objeto Criteria preparado opara trabajar con objetos Cliente. Criteria crit = session.createCriteria(Cliente.class); // Agregamos una condición de igualdad para el campo nombre. crit.add(Restrictions.eq("nombre", "Pepe")); // Limitamos el numero maximo de objetos que se pueden obtener. crit.setMaxResults(10); // Ejecutamos obtenemos una lista con los objetos que cumplen los criterios. List lista = crit.list();</pre> | |

| | |
|---|--|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| <h2>Hibernate Criteria(II)</h2> | <p>Además de HQL, Hibernate dispone de otra modalidad de expresar las consultas: Criteria. Consiste en una serie de Clases en las cuales se pueden expresar las condiciones.</p> |
| <p>El método add de los objetos criteria retornan el mismo objeto criteria por lo que se pueden agregar consecutivamente diferentes criterios.</p> <pre> List clis = sess.createCriteria(Cliente.class) .add(Restrictions.like("apellidos", "%gar%")) .add(Restrictions.or(Restrictions.eq("saldo", 0), Restrictions.isNull("idProvincia"))) .list(); </pre> | |

| | |
|--|--|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| <h2>Hibernate Criteria(Restrictions) (I)</h2> | <p>Además de HQL, Hibernate dispone de otra modalidad de expresar las consultas: Criteria. Consiste en una serie de Clases en las cuales se pueden expresar las condiciones.</p> |
| <p>métodos</p> <p>between(String nombrePropiedad, Object min, Object max) Retorna el valor de la propiedad indicada debe encontrarse entre los valores</p> <p>eq (String nombrePropiedad, Object valor) Compara la igualdad "equal" entre la propiedad y el valor suministrado</p> <p>ge(String nombrePropiedad, Object valor) Iguale que eq pero comprueba si es mayor o igual (>=).</p> <p>gt(String nombrePropiedad, Object valor) Iguale que eq pero comprueba si es mayor (>).</p> <p>idEq(Object valor) Retorna todos los objetos cuyo identificador coincida con el suministrado.</p> <p>ilike(String nombrePropiedad, Object valor) Like case-insensitive (no distingue entre mayúsculas y minúsculas)</p> <p>in(String nombrePropiedad, Object[] valores) Retorna los objetos cuya propiedad indicada contenga un valor de los suministrados en la matriz o Collection valores</p> | |

| | |
|---|--|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Hibernate Criterias(Restrictions) (II) | Además de HQL, Hibernate dispone de otra modalidad de expresar las consultas: Criterias. Consiste en una serie de Clases en las cuales se pueden expresar las condiciones. |
| <p>métodos</p> <p>isNotNull(String nombrePropiedad), isNull(String nombrePropiedad) Retorna los objetos cuya propiedad indicada no es nula o es nula respectivamente</p> <p>le(String nombrePropiedad, Object valor) Retorna los objetos cuya propiedad indicada es menor o igual que el valor indicado.</p> <p>lt(String nombrePropiedad, Object valor) Retorna los objetos cuya propiedad indicada es menor o igual que el valor indicado.</p> <p>ne(String nombrePropiedad, Object valor) Retorna los objetos cuya propiedad indicada es distinta al valor indicado.</p> <p>métodos que conjugan varios criterios</p> <p>not(Criterion c) Retorna el criterio contrario (negado)</p> <p>or(Criterion c1, Criterion c2) Retorna la OR de los dos criterios suministrados.</p> <p>and(Criterion c1, Criterion c2) Retorna el AND de los dos criterios suministrados.</p> | |

| | |
|--|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Hibernate Criterias en asociaciones (I) | Criterias permite crear criterios de búsqueda para un Objeto y hacer referencia a los objetos relacionados. |
| <p>Podemos obtener todos los clientes tengan un telefono con descripcion que empiece por Mov. (Los clientes aparecen repetidos tantas veces como moviles tengan)</p> <pre>List lstClientes = sess.createCriteria(Clientes.class) .createCriteria("telefonos") .add(Restrictions.like("descripcion", "Mov%")) .list();</pre> <p>Si queremos que no repita los clientes que tienen varios moviles, lo indicaremos con:</p> <pre>List lstClientes = sess.createCriteria(Clientes.class) .createCriteria("telefonos") .add(Restrictions.like("descripcion", "Mov%")) .setResultTransformer(CriteriaSpecification.DISTINCT_ROOT_ENTITY) .list();</pre> | |



Hibernate
Profesor Vladimir Bataller

Hibernate Criteria en asociaciones(II)

Criteria permite crear criterios de búsqueda para un Objeto y hacer referencia a los objetos relacionados.


Podemos agregar cliterios a la los objetos obtenidos además de a los objetos relacionados:

```
List lstClientes = sess.createCriteria(Clientes.class)
.add( Restrictions.like("apellidos", "M%") )
.createCriteria("telefonos")
.add( Restrictions.like("descripcion", "Mov%") )
.list();
```

También podemos crear un alias de un elemento relacionado para usarlo en una Restricción posterior:

```
Criteria crit = session.createCriteria(Cliente.class)
.add(Restrictions.like("apellidos", "%a%"))
.createAlias("telefonos", "t")
.add( Restrictions.like("t.descripcion", "Cas%"))
.createAlias("cuentas", "cuen")
.add( Restrictions.eq("cuen.saldo", new Double(0)))

.setResultTransformer(CriteriaSpecification.DISTINCT_ROOT_ENTITY) ;
```



Hibernate
Profesor Vladimir Bataller

Herencia de Objetos persistentes Tipo 1 (I)

Hibernate dispone de mecanismos para asociar diferentes objetos de una jerarquía de clases con elementos relacionales de la bbdd.

Tipo 1: Jerarquía de clases almadenada en una sola tabla

Hibernate asociará al objeto persistente una clase correspondiente de la jerarquía de clases en función de el valor que tome un campo discriminador de la tabla.

Clases

```


classDiagram
    class Estudios {
        idEstudios
        estudios
        String toString()
    }
    class EstudiosSecundaria {
        String toString()
    }
    class EstudiosUniversitarios {
        especialidad
        String toString()
    }
    Estudios <|-- EstudiosSecundaria
    Estudios <|-- EstudiosUniversitarios
        
```


HIBERNATE


Tabla de la bbdd

| |
|---------------|
| ESTUDIOS |
| ID_ESTUDIOS |
| ESTUDIOS |
| ESPECIALIDAD |
| TIPO_ESTUDIOS |

| | |
|---|--|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Herencia de Objetos persistentes Tipo 1 (II) - mapeo | El el archivo de mapeo se debe indicar el campo discriminador y los valores de este para cada clase. |
| <p>Estudios.hbm.xml</p> <pre> <class name="estudios.Estudios" table="ESTUDIOS" discriminator-value=""> <id name="idEstudios" column="ID_ESTUDIOS"> <generator class="native"/> </id> <discriminator column="TIPO_ESTUDIOS" type="string"/> <property name="estudios" type="string" column="ESTUDIOS"/> <subclass name="estudios.EstudiosSecundaria" discriminator-value="SECUNDARIA"> </subclass> <subclass name="estudios.EstudiosUniversitarios" discriminator-value="UNIVERSITARIOS"> <property name="especialidad" type="string" column="ESPECIALIDAD"/> </subclass> </class> </pre> | |

| | |
|---|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Herencia de Objetos persistentes Tipo 1 (III) - clases | Las clases son javabeans como cualquier otra clase persistente de Hibernate que guardan relaciones de herencia entre ellas. |
| <pre> public class Estudios { private int idEstudios; private String estudios; public String toString() { return idEstudios + " - " + estudios; } //... getters y setters } public class EstudiosSecundaria extends Estudios { public String toString() { return super.toString() + " -- Nivel Secundaria --"; } } public class EstudiosUniversitarios extends Estudios { private String especialidad; public String toString() { return super.toString() + "(Esp: " + especialidad + ")"; } //... getters y setters } </pre> | |

| | |
|--|---|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Herencia de Objetos persistentes Tipo 1 (IV) - inserción | La inserción de objetos pertenecientes a una jerarquía no presenta diferencias. |
| <p><u>Inserción de jerarquía de objetos</u></p> <pre> Estudios e = new Estudios(); e.setEstudios("Primaria"); session.save(e); EstudiosSecundaria s = new EstudiosSecundaria(); s.setEstudios("E.S.O."); session.save(s); EstudiosUniversitarios u = new EstudiosUniversitarios(); u.setEstudios("Ingeniería Informática"); u.setEspecialidad("Informática de gestión"); session.save(u); transaction.commit(); </pre> | |

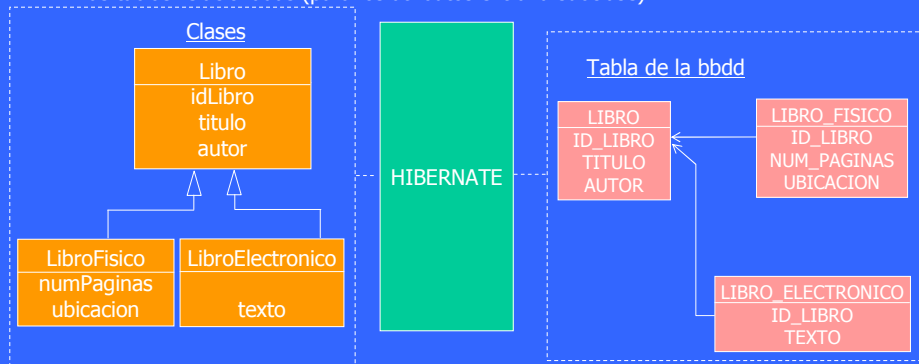
| | |
|---|--|
|  el puente hacia el conocimiento | <h1>Hibernate</h1> <p>Profesor Vladimir Bataller</p> |
| Herencia de Objetos persistentes Tipo 1 (V) - consulta | La consulta de objetos pertenecientes a una jerarquía no presenta diferencias. |
| <p><u>Consulta de jerarquía de objetos</u></p> <pre> List lista = session.createQuery("from Estudios").list(); System.out.println("Num Estudios Encotrados: " + lista.size()); System.out.println("----- LISTA DE TODOS LOS ESTUDIOS -----"); for (int i = 0; i < lista.size(); i++) { System.out.println(lista.get(i)); } List listaUniversitarios = session.createQuery("from EstudiosUniversitarios").list(); System.out.println("Num Estudios Encotrados: " + listaUniversitarios.size()); System.out.println("----- LISTA DE ESTUDIOS UNIVERSITARIOS -----"); for (int i = 0; i < listaUniversitarios.size(); i++) { System.out.println(listaUniversitarios.get(i)); } transaction.commit(); </pre> | |

Herencia de Objetos persistentes Tipo 2 (I)

Hibernate dispone de mecanismos para asociar diferentes objetos de una jerarquía de clases con elementos relacionales de la bbdd.

Tipo 2: Jerarquía de clases almacenada en una tabla por cada clase


Los objetos de una subclase de la Jerarquía, tomarán sus valores de la tabla base y de las tablas relacionadas (para los atributos extra la subclase).



Herencia de Objetos persistentes Tipo 2 (II)

Hibernate dispone de mecanismos para asociar diferentes objetos de una jerarquía de clases con elementos relacionales de la bbdd.

```
<hibernate-mapping>
<class name="libros.Libro" table="LIBRO">
  <id name="idLibro" column="ID_LIBRO">
    <generator class="native"/>
  </id>
  <property name="titulo"/>
  <property name="autor"/>
  <joined-subclass name="libros.LibroFisico" table="LIBRO_FISICO">
    <key column="ID_LIBRO"/>
    <property name="numPaginas" column="NUM_PAGINAS"/>
    <property name="ubicacion" column="UBICACION"/>
  </joined-subclass>
  <joined-subclass name="libros.LibroElectronico" table="LIBRO_ELECTRONICO">
    <key column="ID_LIBRO"/>
    <property name="texto" column="TEXTO"/>
  </joined-subclass>
</class></hibernate-mapping>
```



Spring


Profesor Vladimir Bataller

Spring Framework: Patrones

Spring es un framework que basado en reconocidos patrones de diseño para aplicacione java. La página oficial de Spring es: <http://www.springsource.org>

Principales patrones empleados en Spring

- Inversión del control (IoC) e Inyección de dependencias.
- Singleton (Obtención de objetos únicos para toda la aplicación).
- DAO (Objetos de acceso a datos).
- MVC (Modelo vista controlador) en aplicaciones web.
- Programación orientada a aspectos (AOP).

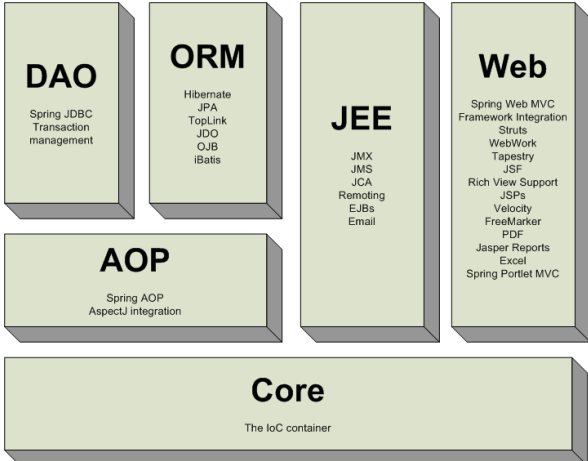


Spring

Profesor Vladimir Bataller

Spring Framework: Modulos

Spring se compone de siete módulos, siendo el núcleo la inversión del control.



The diagram shows the following modules and their components:

- DAO**: Spring JDBC, Transaction management
- ORM**: Hibernate, JPA, TopLink, JDO, OJB, iBatis
- JEE**: JMX, JMS, JCA, Remoting, EJBs, Email
- Web**: Spring Web MVC, Framework Integration, Struts, WebWork, Tapestry, JSF, Rich View Support, JSPs, Velocity, FreeMarker, PDF, Jasper Reports, Excel, Spring Portlet MVC
- AOP**: Spring AOP, AspectJ integration
- Core**: The IoC container

- 1.- Core (Nucleo): Inversión del control (IoC). Context, BeanFactorys.
- 2.- AOP: Aspectos
- 3.- DAO: Acceso a Datos
- 4.- ORM: Mapeo de datos.
- 5.- JEE: Integración con servicios J2EE.
- 6.- Aplicaciones Web.



Ibervtech
el puente hacia el conocimiento

Spring


Profesor Vladimir Bataller

Spring: Tipos de aplicaciones

Spring permite la creación de todo tipo de aplicaciones, applets, aplicaciones web, aplicaciones multicapa, clientes ricos.

Facilidades para la creación de aplicaciones empleando Spring:

- No es invasivo: Permite su uso conjunto a otros frameworks (p.e.: Struts o Hibernate).
- No es todo o nada: No obliga a la implementación de toda la aplicación empleando Spring.
- Aunque Spring es adecuado para aplicaciones empresariales "lijeras" (no requieren de un servidor de aplicaciones, solamente una máquina virtual) pero también se pueden integrar con servidores JEE para obtener las ventajas que estos ofrecen (Seguridad frente a fallos y escalabilidad).
- Provee un sistema de manejo de transacciones de forma declarativa, pero solamente sobre un único recurso transaccional (p.e.: un DataSource). Si se desean realizar transacciones globales (que agrupen varios recursos transaccionales) se deberá usar una transacción de un servidor JEE mediante JTA.



Ibervtech
el puente hacia el conocimiento

Spring

Profesor Vladimir Bataller

El problema de las dependencias

Se dice que una clase depende de otra cuando la primera tiene variables cuyo tipo es de la segunda y cuando la primera crea objetos de la segunda.

ClaseA

```
ClaseB b = new ClaseB();
b.metodo1();
```

ClaseB

```
public void metodo1()
{....}
```

<<depends>>

Inconvenientes de las dependencias (sistemas con muchas de pendencias o muy acoplados)

- 1.- Dificil mantenimiento y rediseño: En tiempo de diseño, cualquier cambio realizado en una clase, puede afectar a las clases que dependen de ella y por lo tanto que sea necesario modificarlas también.
- 2.- Menor estabilidad: En tiempo de ejecución, si una parte de la aplicación tiene un error o deja de funcionar, si el sistema tiene muchas dependencias, se verá mayormente afectado.



Ib Bertech
el puente hacia el conocimiento

Spring

Profesor Vladimir Bataller

Soluciones al problema de las dependencias (I)

La primera solución que se aporta para reducir el problema de las dependencias es dividir la aplicación en capas.




```

graph LR
    A[Presentacion] --> B[Lógica de negocio]
    B --> C[Persistencia (clases DAO)]
    C --> D[(BBDD)]
  
```

De esta forma se reducen las dependencias porque cada capa solamente se comunica con las capas contiguas y por lo tanto solamente depende de ellas.

Así, por ejemplo, si hacemos un cambio en el acceso a la BBDD o en la tecnología de esta, solamente habrá que hacer cambios en la capa de persistencia, no viéndose afectadas el resto de capas.



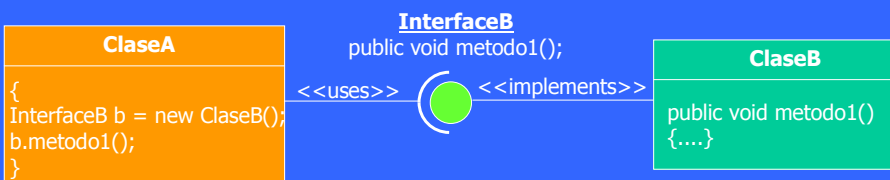
Ib Bertech
el puente hacia el conocimiento

Spring

Profesor Vladimir Bataller

Soluciones al problema de las dependencias(II)

La segunda solución para reducir las dependencias es emplear interfaces para que las variables empleadas en una parte de la aplicación no queden ligada a una clase sino a una interface, pudiendo hacer referencia a diferentes clases que implementen la interface.



```

classDiagram
    class ClaseA {
        InterfaceB b = new ClaseB()
        b.metodo1()
    }
    class InterfaceB {
        public void metodo1()
    }
    class ClaseB {
        public void metodo1()
    }
    ClaseA --> InterfaceB : <<uses>>
    InterfaceB <|-- ClaseB : <<implements>>
  
```

El unico problema que queda ahora por resolver es la creación de objetos. Spring mediante la inversión del control hace desaparecer los operadores new mediante dos formas:

- 1.- Obtención de objetos mediante una clave a traves de un contexto de Spring.
- 2.- Asignación del objeto por el contexto mediante un metodo "set" de la claseA

De esta forma desaparecen de la claseA todas las referencias a la claseB.

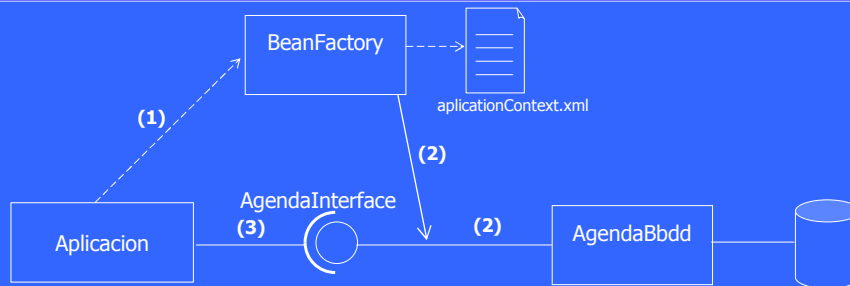
ClaseA

```

{
    InterfaceB b = ctx.getBean("claveObjetoB");
    b.metodo1();
}
public void setB(InterfaceB b){this.b=b;}
  
```

Spring: Inversión del control

El patrón más característico de Spring es la inversión del control. Que elimina las dependencias entre los objetos proveedores de servicios de la aplicación. Estas dependencias se centralizan en un archivo de configuración.

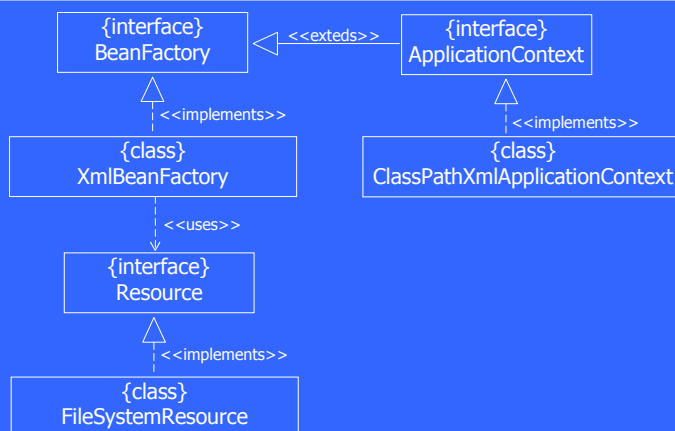


La aplicación no crea los Objetos directamente (new + Clase del objeto). Sino que la aplicación solicita (1) a la factoría de Beans una referencia (2) al objeto y es la factoría quien lo crea. La aplicación guarda la referencia en una interface (3) implementada por la clase del objeto obtenido.

La factoría de bean decide que objeto crear de acuerdo a un archivo xml de configuración, de esta forma si se decide cambiar la clase del objeto a instanciar, el cambio habrá que hacerlo únicamente en el archivo de configuración, que es donde se centralizan las dependencias.

Spring: Factorías de beans y contextos

Mientras que los BeanFactories únicamente contienen la funcionalidad para la obtención de beans. Los Context, además, permiten configurar otros servicios del framework. (internacionalización, recursos web, AOP entre otros).





| | |
|---|---|
|  <p>el puente hacia el conocimiento</p> | <h2 style="text-align: center;">Spring</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| <h3>Spring: Obtención de una Factoría de Beans</h3> | <p>Una factoría de beans se puede obtener a partir de uno o varios documentos xml, a partir de un archivo properties o por una clase java programada adhoc. La forma más versatil suele ser la primera (a partir de xml).</p> |
| <pre>// Creación de un BeanFactory a partir de un recurso asociado a un xml. Resource res = new FileSystemResource("A01applicationContext.xml"); BeanFactory ctx = new XmlBeanFactory(res); // Creación de un ApplicationContext a partir de un xml. ApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");</pre> | |


| | | | |
|---|---|---|---|
|  <p>el puente hacia el conocimiento</p> | <h2 style="text-align: center;">Spring</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> | | |
| <h3>Spring: Jars de Spring</h3> | <p>Dependencias de Spring según el tipo de aplicacion.</p> | | |
| <table border="0" style="width: 100%;"> <tr> <td style="vertical-align: top; width: 50%;"> <p>Jar minimos para ejecutar Spring (IoC): dist/spring.jar lib/log4j/log4j.jar lib/jakarta-commons/commons-loggin.jar</p> <p>Jar para usar DaoJdbc: lib/jakarta-commons/commons-dbcj.jar lib/jakarta-commons/commons-pool.jar (además el driver de la bbdd)</p> <p>Jar para usar aspectos (AOP): lib/aspectj/aspectjweaver.jar dist/weaving/spring-aspects.jar lib/cglib/cglib-nodep.jar</p> <p>Jar para aplicaciones Web (carpeta lib del proyecto): dist/spring.jar dist/modules/spring-webmvc.jar lib/j2ee/jstl.jar lib/jakarta-taglibs/standard.jar lib/log4j/log4j.jar</p> </td> <td style="vertical-align: top; width: 50%;"> <p>Jar para hibernate: lib/cglib/cglib-nodep.jar lib/asm/asm-commons.jar lib/asm/asm-util.jar lib/asm/asm.jar lib/j2ee/jta.jar lib/dom4j/dom4j.jar lib/antlr/antlr.jar lib/jakarta-commons/commons-collection.jar lib/hibernate/hibernate3.jar lib/slf4j/slf4j-api.jar lib/slf4j-log4j.jar lib/javassist/javassist.jar</p> </td> </tr> </table> | | <p>Jar minimos para ejecutar Spring (IoC): dist/spring.jar lib/log4j/log4j.jar lib/jakarta-commons/commons-loggin.jar</p> <p>Jar para usar DaoJdbc: lib/jakarta-commons/commons-dbcj.jar lib/jakarta-commons/commons-pool.jar (además el driver de la bbdd)</p> <p>Jar para usar aspectos (AOP): lib/aspectj/aspectjweaver.jar dist/weaving/spring-aspects.jar lib/cglib/cglib-nodep.jar</p> <p>Jar para aplicaciones Web (carpeta lib del proyecto): dist/spring.jar dist/modules/spring-webmvc.jar lib/j2ee/jstl.jar lib/jakarta-taglibs/standard.jar lib/log4j/log4j.jar</p> | <p>Jar para hibernate: lib/cglib/cglib-nodep.jar lib/asm/asm-commons.jar lib/asm/asm-util.jar lib/asm/asm.jar lib/j2ee/jta.jar lib/dom4j/dom4j.jar lib/antlr/antlr.jar lib/jakarta-commons/commons-collection.jar lib/hibernate/hibernate3.jar lib/slf4j/slf4j-api.jar lib/slf4j-log4j.jar lib/javassist/javassist.jar</p> |
| <p>Jar minimos para ejecutar Spring (IoC): dist/spring.jar lib/log4j/log4j.jar lib/jakarta-commons/commons-loggin.jar</p> <p>Jar para usar DaoJdbc: lib/jakarta-commons/commons-dbcj.jar lib/jakarta-commons/commons-pool.jar (además el driver de la bbdd)</p> <p>Jar para usar aspectos (AOP): lib/aspectj/aspectjweaver.jar dist/weaving/spring-aspects.jar lib/cglib/cglib-nodep.jar</p> <p>Jar para aplicaciones Web (carpeta lib del proyecto): dist/spring.jar dist/modules/spring-webmvc.jar lib/j2ee/jstl.jar lib/jakarta-taglibs/standard.jar lib/log4j/log4j.jar</p> | <p>Jar para hibernate: lib/cglib/cglib-nodep.jar lib/asm/asm-commons.jar lib/asm/asm-util.jar lib/asm/asm.jar lib/j2ee/jta.jar lib/dom4j/dom4j.jar lib/antlr/antlr.jar lib/jakarta-commons/commons-collection.jar lib/hibernate/hibernate3.jar lib/slf4j/slf4j-api.jar lib/slf4j-log4j.jar lib/javassist/javassist.jar</p> | | |


| | |
|---|--|
|  el puente hacia el conocimiento | <h2 style="text-align: center;">Spring</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| <h3>Spring: Obtención de un bean del contexto(II)</h3> | <p>Al crear contexto, podemos indicar el archivo xml de configuración como un string o varios archivos de configuración como una matriz de string.</p> |
| <pre>// Creación del contexto de aplicación ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml"); // Petición de un bean al contexto (No hay dependencia de la implementación la clase de // implementación se indica en el xml.) PersonaInterface persona = (PersonaInterface) ctx.getBean("unaPersona"); persona.setId(1); persona.setNombre("Daniel"); persona.setTelefono("612345678"); System.out.println("Persona: " + persona); // Si se le pide otra vez una persona al contexto se observa que // retorna el mismo objeto, es decir, es un Singleton. persona = (PersonaInterface) ctx.getBean("unaPersona"); System.out.println("Persona: " + persona);</pre> | |


| | |
|--|---|
|  el puente hacia el conocimiento | <h2 style="text-align: center;">Spring</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| <h3>Spring JDBC (I)</h3> | <p>Spring proporciona un acceso declarativo a bases de datos. El elemento para realizar consultas será un JdbcTemplate.</p> |
| <ol style="list-style-type: none"> 1.- En el classpath, además del driver específico de la bbdd, se deben incluir los jars siguientes del jakarta commons project: commons-dbcp.jar commons-pool.jar 2.- Configurar el DataSource en el applicationContext.xml con los datos de conexión a la bbdd. 3.- Declarar, en el applicationContext.xml, un JdbcTemplate asociado al DataSource. 4.- Asignarle, en el applicationContext.xml, al objeto Dao el JdbcTemplate. 5.- Ejecutar en el Dao el método <i>queryForList</i> pasándole la cadena sql a ejecutar para realizar una búsqueda. Este método retorna un List de Maps. Cada elemento de la lista es un objeto Map que representa un registro. Los elementos del map contienen los valores de los campos asociados a una clave que contiene el nombre del campo. | |

| | |
|--|--|
|  el puente hacia el conocimiento | <h2 style="text-align: center;">Spring</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| <h3 style="text-align: center;">Spring JDBC: Declaración de un DataSource</h3> | <p>Spring permite obtener un DataSource local o un DataSource de un servidor JEE obtenido mediante JNDI.</p> |
| <p><u>DataSource local (Se ejecuta en la misma maquina Virtual)</u></p> <pre><bean id="dataSource" destroy-method="close" class="org.apache.commons.dbcp.BasicDataSource"> <property name="driverClassName" value="com.mysql.jdbc.Driver"/> <property name="url" value="jdbc:mysql://localhost/DATOS"/> <property name="username" value="alumno17"/> <property name="password" value="java"/> </bean></pre> <p><u>DataSource mediante JNDI</u></p> <pre><bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean"> <property name="jndiName"> <value>jdbc/DATOS</value> </property> </bean></pre> | |

| | |
|---|---|
|  el puente hacia el conocimiento | <h2 style="text-align: center;">Spring</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| <h3 style="text-align: center;">Spring JDBC (III)</h3> | <p>Spring proporciona un acceso declarativo a bases de datos. El elemento para realizar consultas será un JdbcTemplate.</p> |
| <p><u>applicationContext.xml</u></p> <pre><bean id="dataSource" destroy-method="close" class="org.apache.commons.dbcp.BasicDataSource"> <property name="driverClassName" value="com.mysql.jdbc.Driver"/> <property name="url" value="jdbc:mysql://localhost/DATOS"/> <property name="username" value="alumno17"/> <property name="password" value="java"/> </bean> <bean id="jdbcTemplate" class="org.springframework.jdbc.core.simple.SimpleJdbcTemplate"> <constructor-arg> <ref local="dataSource" /> </constructor-arg> </bean> <bean id="agendaDao" class="agenda.AgendaBbddDaoJdbcTemplate"> <property name="jdbcTemplate" ref="jdbcTemplate" /> </bean></pre> | |


| | |
|---|---|
|  el puente hacia el conocimiento | <h2>Spring</h2> <p>Profesor Vladimir Bataller</p> |
| Spring JDBC: Consultas de Selección | <p>A continuación se muestra la implementación un método de la clase <i>AgendaBbdd</i> que emplea un atributo <i>JdbcTemplate</i> de tipo <i>SimpleJdbcTemplate</i> sobre el cual se ejecuta <i>queryForList</i> que recibe la cadena sql y un parámetro.</p> |
| <pre> public List<PersonaInterface> buscarPorNombre(String nom) { Vector<PersonaInterface> resul = new Vector<PersonaInterface>(); Object[] param = new Object[]{"%" + nom + "%"}; // Matriz con los parámetros String sql="SELECT id, nombre, telefono FROM AGENDA WHERE nombre like ?"; try { List<Map<String, Object>> lista = jdbcTemplate.queryForList(sql,param); for (int i = 0; i < lista.size(); i++) { Persona p = new Persona(); p.setId((Integer) lista.get(i).get("ID")); p.setNombre((String) lista.get(i).get("NOMBRE")); p.setTelefono((String) lista.get(i).get("TELEFONO")); resul.add(p); } } catch (DataAccessException ex) { return null;} return resul; } </pre> | |

| | |
|--|--|
|  el puente hacia el conocimiento | <h2>Spring</h2> <p>Profesor Vladimir Bataller</p> |
| Spring JDBC: Consultas de Selección por Id | <p>En este caso empleamos <i>queryForMap</i> ya que solamente retorna un registro. Si no lo encuentra, produce una <i>EmptyResultDataAccessException</i></p> |
| <pre> public PersonaInterface buscarPorId(int id) { String sql="SELECT id,nombre,telefono FROM agenda WHERE id = ?"; Object[] param = new Object[]{ new Integer(id)}; Map<String,Object> registro =jdbcTemplate.queryForMap(sql,param); if(registro!=null) { PersonaInterface p = new Persona(); p.setId((Integer)registro.get("id")); p.setNombre((String)registro.get("nombre")); p.setTelefono((String)registro.get("telefono")); return p; } return null; } </pre> | |

| | |
|---|---|
|  el puente hacia el conocimiento | <h2 style="text-align: center;">Spring</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| <h3>Spring JDBC: Consultas de Actualización</h3> | <p>Para ejecutar consultas de actualización (DELETE, UPDATE, INSERT), los objetos SimpleJdbcTemplate disponen de el método update que recibe dos parámetros, la consulta sql y una matriz con los parámetros.</p> |
| <pre> public boolean borrar(int id) { String sql="DELETE FROM AGENDA WHERE id = ?"; Object[] param = new Object[]{new Integer(id)}; int n=jdbcTemplate.update(sql,param); return n==1; } public boolean insertar(PersonaInterface nueva) { String sql = "INSERT INTO AGENDA(NOMBRE,TELEFONO) VALUES(?,?)"; Object [] params=new Object[]{nueva.getNombre(),nueva.getTelefono()}; int n = jdbcTemplate.update(sql, params); return n==1; } public boolean editar(PersonaInterface persona) { String sql="UPDATE AGENDA SET nombre=?,telefono=? WHERE id = ?"; Object[] param = new Object[]{ persona.getNombre(), persona.getTelefono(),persona.getId()}; int n = jdbcTemplate.update(sql,param); return n==1; } </pre> | |

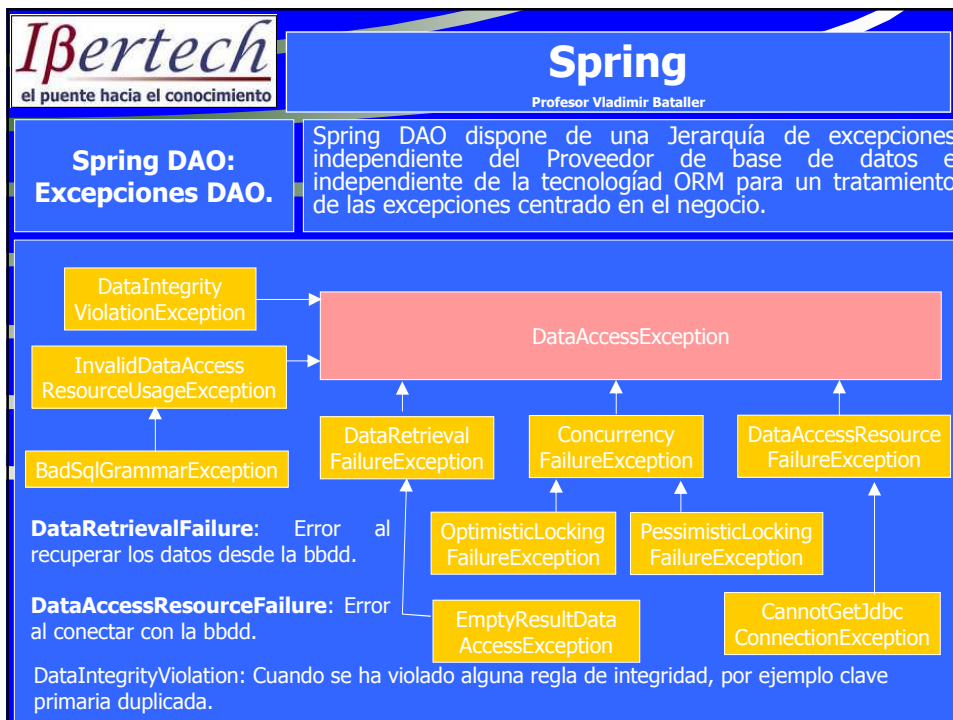
| | |
|---|---|
|  el puente hacia el conocimiento | <h2 style="text-align: center;">Spring</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| <h3>Spring DAO JDBC SUPPORT</h3> | <p>Para la creación de datos como los anteriormente vistos, se puede heredar de la clase SimpleJdbcDaoSupport que contiene dos atributos simpleJdbcTemplate y dataSurce que no será necesario declarar.</p> |
| <p>En el contexto es suficiente con asignarle al dao el DataSource y el ya crea internamente el SimpleJdbcTemplate. Ahora para ejecutar cualquier operación sobre la bbdd, obtendremos el SimpleJdbcTemplate con el método getSimpleJdbcTemplate().</p> <p>applicationContext.xml</p> <pre> <bean id="ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close"> </bean> <bean id="agendaDao" class="agenda.AgendaDaoJdbcRowMapper"> <property name="dataSource" ref="ds"/> </bean> </pre> <p>Clase AgendaDao</p> <p>Para consultas de actualización: getSimpleJdbcTemplate().update(sql,param); Para consultas de un registro: getSimpleJdbcTemplate().queryForMap(sql,param); para consultas de un conjunto de registros:</p> <p style="text-align: center;">List<Map<String,Object>> registros =getSimpleJdbcTemplate().queryForList(sql,param);</p> | |


| | |
|--|--|
|  el puente hacia el conocimiento | <h2>Spring</h2> <p>Profesor Vladimir Bataller</p> |
| Spring JDBC: Consultas con RowMapper | <p>Los RowMapper son objetos que se pueden pasar en una consulta para indicar en que objeto convertir cada registro obtenido de la bbdd y la identificación de cada campo con los atributos de dicho objeto.</p> |
| <pre> public class PersonaRowMapper implements ParameterizedRowMapper<PersonaInterface> { public PersonaInterface mapRow(ResultSet rs, int pos)throws SQLException { Persona p = new Persona(); p.setId(rs.getInt("id")); p.setNombre(rs.getString("nombre")); p.setTelefono(rs.getString("telefono")); return p; } } String sql = "SELECT ID, NOMBRE, TELEFONO FROM AGENDA WHERE NOMBRE LIKE ?"; PersonaRowMapper rm = new PersonaRowMapper(); Object [] params = new Object[] {"%" + nom + "%"}; List<PersonaInterface> personas = getSimpleJdbcTemplate().query(sql, rm, params); return personas; </pre> <p>Nota: En este ejemplo solamente se indica la parte interior al try del método buscarPorNombre, no se muestra el tratamiento de Excepciones.</p> | |

| | |
|---|---|
|  el puente hacia el conocimiento | <h2>Spring</h2> <p>Profesor Vladimir Bataller</p> |
| Spring JDBC: Consulta de un objeto | <p>El método queryForObject retorna un objeto especificado bien por un objeto Class o por un ParametrizedRowMapper.</p> |
| <pre> public class PersonaRowMapper implements ParameterizedRowMapper<PersonaInterface> { public PersonaInterface mapRow(ResultSet rs, int pos)throws SQLException { Persona p = new Persona(); p.setId(rs.getInt(1)); p.setNombre(rs.getString(2)); p.setTelefono(rs.getString(3)); return p; } } String sql = "SELECT ID, NOMBRE, TELEFONO FROM AGENDA WHERE ID = ?"; PersonaRowMapper rm = new PersonaRowMapper(); Object [] params = new Object[] {new Integer(3)}; Persona persona = (Persona) jdbc.queryForObject(sql,rm,params); return persona; String nombre = jdbc.queryForObject("SELECT NOMBRE FROM AGENDA WHERE ID=?", String.class, new Object[] {1}); </pre> | |

| | |
|--|---|
|  el puente hacia el conocimiento | <h2 style="text-align: center;">Spring</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| Spring JDBC: Consulta de un entero | <p>Las consultas de agregado se pueden ejecutar con queryForInt.</p> |
| <pre>int cuantos = jdbc.queryForInt("SELECT count(ID) from AGENDA"); long cuantos = jdbc.queryForLong("SELECT count(ID) from AGENDA");</pre> <p>Nota: para Obtener un SimpleJdbcTemplate por código, se debe pasar un Datasource en su constructor. SimpleJdbcTemplate jdbc = new SimpleJdbcTemplate(dataSource);</p> | |

| | |
|--|---|
|  el puente hacia el conocimiento | <h2 style="text-align: center;">Spring</h2> <p style="text-align: center;">Profesor Vladimir Bataller</p> |
| Spring DAO: Superclases DAO. | <p>Spring DAO permite la implementación del Patrón DAO a partir de Clases específicas que facilitan la inversión del control. Existen clases para acceso a la información mediante JDBC, Hibernate, JDO o iBatis.</p> |
| <p>Las clases que implementen el patrón DAO, pueden extender, según la tecnología con la que accedan a los datos de una de las siguientes Clases:</p> <ul style="list-style-type: none"> -JdbcDaoSupport : Posee un atributo dataSource que debe ser inicializado por el contexto. A cambio, ofrece un método getJdbcTemplate() que proporciona el objeto para realizar las consultas Spring JDBC. -SimpleJdbcDaoSupport : Posee un atributo dataSource que debe ser inicializado por el contexto. A cambio, ofrece un método getSimpleJdbcTemplate() que proporciona el objeto para realizar las consultas Spring JDBC con genericos java. - HibernateDaoSupport: Requiere la asignación de un atributo sessionFactory y mediante el método getHibernateTemplate proporciona un objeto sobre el que realizar las consultas. - JdoDaoSupport: atributo persistenceManagerFactory, proporciona un método getJdoTemplate(). - JpaDaoSupport: atributo entityManagerFactory, método getJpaTemplate(). | |





Ibervtech
el puente hacia el conocimiento

Spring

Profesor Vladimir Bataller

**Spring DAO:
Excepciones DAO.**

Spring DAO dispone de una Jerarquía de excepciones independiente del Proveedor de base de datos e independiente de la tecnología ORM para un tratamiento de las excepciones centrado en el negocio.

EmptyResultDataAccessException: Cuando se realiza una búsqueda por clave primaria (queryForMap) y no se encuentra el registro.

CannotGetJdbcConnectionException: No se puede conectar con el servidor.

Spring: Acceso a datos con Hibernate

Los DAO para Hibernate tienen un método sessionFactory en el cual se debe crear un HibernateTemplate.

La clase HibernateTemplate, que se inicializa con un SessionFactory en su constructor, permite realizar consultas con HQL y operaciones de mantenimiento de los Objetos persistentes:

List **find**(String hql, Object[] args)-> Retorna un listado de objetos con el resultado de ejecutar una consulta HQL con los parámetros suministrados en la matriz de Object.

Object **get**(Class clase, int id) -> Retorna un objeto de la clase especificada con el id suministrado, si no existe retorna null.

void **delete**(Object entidad) -> Borra de la bbdd el registro asociado al objeto suministrado.

void **update**(Object entidad) -> Actualiza en la bbdd el registro asociado al objeto suministrado.

void **save**(Object entidad)-> Inserta en la bbdd un registro con los valores del objeto suministrado. Si no se sabe si el objeto existe ya en la bbdd se puede emplear saveOrUpdate.