Desarrollo eficiente de aplicaciones empresariales Usando el framework hibernate

Efficient enterprise application development using the Framework Hibernate

Christian Mauricio Castillo Estrada, Karina Cancino Villatoro, Luis Antonio Álvarez Oval

Abstract

This paper presents a comprehensive description of the Hibernate technology, noting its importance in the development of enterprise applications as an efficient means of access to large databases and also focuses on architectures that Hibernate provides for implementation. In addition, the powerful Hibernate Query Language, which is designed as a minimal extension of SQL focused on retrieving objects, proving to be a bridge between the relational model and the object-oriented paradigm is discussed. Hibernate is one of the most powerful ORM solutions and high performance that exists today for the implementation of Persistence. Finally, the easy integration with other frameworks like Hibernate Java Server Faces (JSF), Spring or Struts has also been discussed in the paper; allowing the rapid construction of enterprise applications based on a model of N-tier architecture.

Keywords: Databases, ORM, Persistence, Hibernate, HQL.

Resumen

El artículo presenta una descripción de la tecnología Hibernate, señalando su importancia en el desarrollo de aplicaciones empresariales, como un medio eficiente de acceso a grandes bases de datos y también se centra, en las arquitecturas que ofrece Hibernate para su implementación. Se aborda el poderoso Lenguaje de Consulta de Hibernate, el cual está diseñado como una extensión mínima de SQL enfocado a la recuperación de objetos, resultando ser un puente de comunicación entre el modelo relacional y el paradigma orientado a objetos. Hibernate es una de las soluciones ORM más poderosa y de alto desempeño que existe hoy en día para la implementación de Persistencia. La fácil integración de Hibernate con otros frameworks como Java Server Faces (JSF), Spring o Struts, también se discute en este artículo; permitiendo la rápida construcción de aplicaciones empresariales basado en un modelo de arquitectura de N-Capas.

Palabras Clave: Bases de Datos, ORM, Persistencia, Hibernate, HQL.

Recibido / Received: Abril 06 de 2014 **Aprobado / Aproved:** Marzo 13 de 2015

Tipo de artículo / Type of paper: Investigación científica y tecnológica.

Afiliación Institucional de los autores / Institutional Affiliation of authors: Universidad Autónoma de Chiapas, México. Universidad Politécnica de Tapachula, México. Universidad Autónoma de Chiapas, México.

Autor para comunicaciones / Author communications: Christian Mauricio Castillo Estrada, cmce@unach.mx

Los autores declaran que no tienen conflicto de interés.

Introduction

relevante en el desarrollo de una aplicación empresarial es la creación y mantenimiento de la información, es decir, la implementación de una capa de Persistencia que permita la interacción con la base de datos mediante el uso de objetos de forma rápida y segura.

El almacenamiento de información ordenada a gran escala juega un papel importante en la mayoría de las aplicaciones comerciales, tales como catálogos, listas de clientes, datos de contacto, texto publicado, y los diseños arquitectónicos.

Con la aparición de la World Wide Web, la demanda de bases de datos se ha incrementado; en la actualidad, los clientes de las librerías y periódicos en línea hacen uso de bases de datos; la aplicación que interactúa con ellos, se encuentra en constante comunicación con una base de datos para consultarla y dar una respuesta al cliente; en ese contexto, Hibernate simplifica la comunicación con las bases de datos relacionales. [1]

En muchas ocasiones los desarrolladores recurren al uso de prácticas costosas y problemáticas al implementar una capa de datos creada desde cero.

Framework Hibernate

Hibernate es una herramienta de mapeo objeto-relacional (ORM) para el lenguaje Java, que proporciona un framework para el mapeo de un modelo de dominio orientado a objetos a una base de datos relacional tradicional.

Hibernate ayuda a resolver problemas de tipo Objeto-Relacional respecto al desajuste de impedancia mediante el reemplazo del acceso directo a las bases de datos relacionales, e implementando la persistencia con funciones de alto nivel de manipulación de objetos.

Los ORM son herramientas para la representación y la traducción de los datos entre la base de datos y el lenguaje de programación orientado a objetos; proporciona soporte para las colecciones y las relaciones entre objetos, así como el manejo de tipos compuestos.

Hibernate es una de las mejores soluciones ORM que existen en la actualidad, ha sido liberado bajo la Licencia Pública GNU. La versión 2.1 de Hibernate resultó ganadora del Jolt Award en el año 2005.

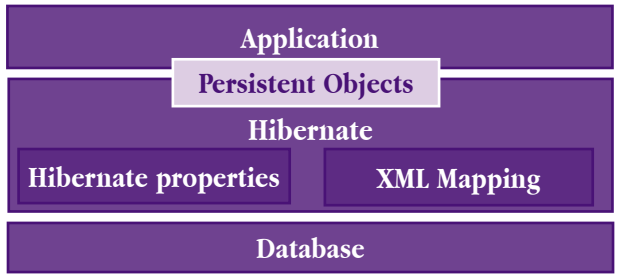
Una de sus principales características, radica en su fácil integración otros frameworks como son: Java Server Faces (JSF), Spring o Struts; los cuales ofrecen clases y anotaciones específicas que permiten establecer una rápida comunicación con Hibernate. En el caso del Spring provee una clase llamada HibernateTemplate, la cual ofrece los métodos necesarios para implementar las operaciones básicas CRUD (Create-Replace-Update-Delete) en una aplicación; los cuales resultan ser simples ocupando una sola línea de código.

Arquitectura de Hibernate

Resulta difícil decidir que arquitectura es la que mejor describe la estructura y funcionamiento de Hibernate; la flexibilidad que ofrece permite que sea implementado en múltiples maneras, provocando la generación de diferentes arquitecturas; sin embargo, en este artículo se ilustran las dos arquitecturas principales.

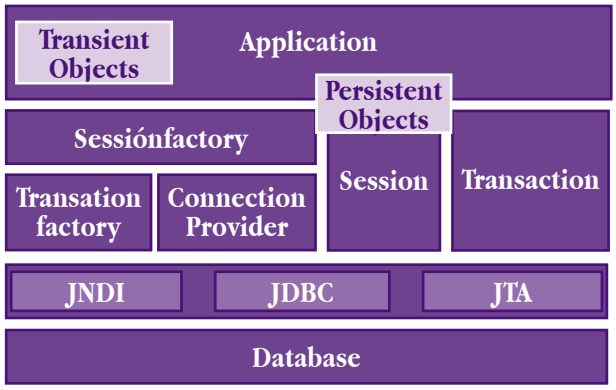
En la figura 1, se muestra una arquitectura compacta desde una perspectiva de alto nivel, en la cual se observa cómo Hibernate utiliza la base de datos y los datos de configuración para poder proporcionar los servicios de persistencia y los objetos persistentes a la aplicación que se encuentra en una capa superior.

Fig. *1. Arquitectura básica*



Una aplicación que utiliza Hibernate debe utilizar un solo archivo de configuración, el cual contendrá información relativa a la dirección URL del servidor de base de datos, el nombre de usuario / contraseña, nombre de clase del archivo del controlador, entre otras preferencias. Este archivo de configuración puede estar basado en XML (hibernate.cfg.xml) o puede ser un archivo de propiedades Java (hibernate.properties) con la combinación de clave y valor. [2]

Fig. 2. Arquitectura completa



Por otra parte, en la figura 2 se ilustra una versión completa de la arquitectura Hibernate, observándose algunos componentes importantes, tales como: SessionFactory que permite mantener un caché de datos que es reutilizables entre transacciones; ConnectionProvider para el manejo de múltiples conexiones JDBC, JTA o transacciones CORBA; Objetos persistentes, los cuales se encuentran asociados a una sesión y son de corta duración; y Objetos transitorios, las instancias de clases persistentes que no están asociados a una sesión.

Mapeo Objeto-Relacional

El Mapeo Objeto-Relacional es una técnica de representación de datos a partir de un modelo de objetos para un modelo de datos relacional y viceversa; este mecanismo se utiliza para la selección, inserción, actualización, eliminación y consulta de registros de las tablas subyacentes.

Para realizar la implementación del mapeo objeto-relacional se requiere del uso de un archivo XML, el cual debe ser guardado usando el formato <nombre_de_clase.hbm.xml>, por ejemplo: Clientes.hbm.xml. Este archivo de asignación XML define la estructura de transformación de los datos de POJO (Plain Old Java Object) a las tablas de bases de datos y viceversa.

Por otra parte, en las últimas versiones de Hibernate, se implementa el uso de Anotaciones representando una nueva forma de definir las asignaciones sin utilizar archivos XML; siendo la forma más eficaz de proporcionar los metadatos para el objeto y la estructura de la tabla relacional. Todos los metadatos son incluidos en un archivo de tipo clase java usando el formato <nombre_de_clase.java> junto con el código que ayuda a los desarrolladores a entender la estructura de la tabla relacional y los objetos Java de tipo POJO simultáneamente. [3]

Otro aspecto relevante en los ORM, es el mapeo de las asociaciones entre las clases de entidad y las relaciones entre las tablas. Hibernate define cuatro modos en que la cardinalidad de la relación entre los objetos se puede expresar, siendo estos: *Many-to-One, One-to-One, One-to-Many y Many-to-Many*. [4] Una asociación de mapas puede ser unidireccional y bidireccional.

Estados de objetos

Hibernate soporta los siguientes estados de objetos:

Transient: un objeto es transitorio, sólo si ha sido creado mediante una instancia usando el operador new, y no está asociado con una sesión de Hibernate. No tiene ninguna representación persistente en la base de datos y no tiene valor identificador se le ha asignado. Las instancias transitorias serán destruidas por el recolector si la aplicación no tiene una referencia más.

Persistente: una instancia persistente tiene una representación en la base de datos y un valor de identificador. Es posible, que el objeto pudo haber sido guardado o recuperado, en el ámbito de una sesión. Hibernate detectará los cambios realizados en un objeto en estado persistente y sincronizará su estado con la base de datos cuando la unidad de trabajo sea completada.

Independiente: es un objeto que ha sido persistente, pero su período de sesión se ha finalizado; es decir, la referencia al objeto sigue siendo válida, en ese sentido, la instancia independiente podría incluso ser modificada en este estado. Una instancia de este tipo puede añadirse a una sesión, en un tiempo posterior, lo cual provocaría que se encuentre activa nuevamente y sus modificaciones realizadas se conviertan en persistentes.

Lenguaje de Consulta Hibernate - HQL

Hibernate Query Language (HQL), es un lenguaje de consulta orientado a objetos que provee la plataforma Hibernate, similar al lenguaje SQL; sin embargo, a diferencia de trabajar con tablas y columnas como en el

caso del lenguaje SQL, este lenguaje trabaja con objetos persistentes y propiedades. Las consultas creadas con HQL son traducidas en sentencias SQL de manera automática, para mejorar el desempeño de la manipulación de tablas de una base de datos relacional.

Consultas en HQL

Para escribir una consulta usando HQL, resulta importante conocer que las palabras claves como SELECT, FROM y WHERE no son sensibles al uso de mayúsculas y minúsculas; sin embargo, el uso de algunas propiedades como nombres de tablas o columnas, sí son sensibles en HQL.

La cláusula SELECT ofrece un mayor control sobre los resultados que se desean obtener; si deseas obtener ciertas propiedades de los objetos en vez de obtener el objeto completo es necesario el uso de esta cláusula; por ejemplo, si deseamos obtener únicamente los valores de la propiedad nombre de la clase Clientes:

```
Query q = session.createQuery ("SELECT C.nombre FROM Clientes C");

Query q = session.createQuery ("SELECT C.nombre FROM Clientes C");

List resultado = q.list ();
```

En ocasiones resulta necesario seleccionar objetos específicos basándose en ciertas condiciones; para este tipo de situaciones, podemos usar la cláusula WHERE; la cual nos permitirá filtrar los objetos que deseamos obtener, veamos el siguiente ejemplo:

```
Query q = session.createQuery ("FROM Clientes C WHERE C.edad > 30");

List resultado = q.list ();
```

Consultadas predefinidas o nombradas

Las Consultas SQL o creadas con HQL, pueden ser definidas en el archivo de mapeo de una Clase, especificando un alias para poder ser identificada.

```
<sql-query name="personas">
    <return alias="persona" class="eg.Persona"/>
    SELECT persona.NOMBRE AS {persona.nombre},
        persona.EDAD AS {persona.edad},
        persona.TEL AS {persona.telefono}
    FROM PERSONA persona
    WHERE persona.NOMBRE LIKE :nombreVAR
</sql-query>

List resultado = session.getNamedQuery("personas")
    .setString("christian", nombreVAR)
    .list();
```

Así también, una consulta SQL llamada puede devolver un valor escalar; por tal motivo, se debe declarar el alias de columna y el tipo de dato usado en Hibernate:

```
<sql-query name="mySqlQuery">
    <return-scalar column="nombre" type="string"/>
    <return-scalar column="edad" type="long"/>
    SELECT p.NOMBRE AS nombre,
        p.EDAD AS edad,
FROM PERSONA p WHERE p.NOMBRE LIKE 'K%'
</sql-query>
```

Consultas Nativas

Existe la posibilidad de utilizar SQL nativo para expresar las consultas. A partir de la versión 3 de Hibernate, se permite el uso de SQL, incluyendo procedimientos almacenados, para crear, actualizar, eliminar, y las operaciones de carga; por lo anterior, es necesario hacer uso del método *createSQLQuery* () de la interfaz Session.

Después de pasar la cadena que contiene la consulta SQL al método createSQLQuery (), es posible asociar el resultado con una entidad existente, una combinación, o un resultado escalar utilizando los métodos addEntity (), addJoin (), y addScalar () respectivamente.

La consulta SQL más básica, es obtener una lista de valores escalares de una o más tablas:

```
String sqlnativo = "SELECT idc, nombre, edad FROM clientes";
```

SQLQuery q = session.createSQLQuery (sqlnativo);

q.setResultTransformer (Criteria.ALIAS_TO_ENTITY_MAP);

List resultados = q.list ();

La siguiente sintaxis permite obtener objetos de entidad en su conjunto de una consulta SQL nativa especificando el nombre de la clase mediante el método addEntity ().

*String sql = "SELECT * FROM clientes";*

SQLQuery q = session.createSQLQuery (sql);

q.addEntity (Clientes.class);

List resultados = q.list ();

Si la entidad posee una cardinalidad de muchos-a-uno con otra entidad diferente, entonces se requiere también devolver dicha entidad al momento de realizar la consulta nativa, de lo contrario se producirá un error específico indicándose que "una columna no encontrada". Las columnas adicionales serán automáticamente devueltos cuando se utiliza la notación *.

Uso de Criterios

Hibernate ofrece varias alternativas para la manipulación de objetos y en los datos disponibles a su vez en las tablas RDBMS. Una de estas alternativas, es el API de Criterios, la cual permite crear un objeto de consulta basado en criterios de programación donde se pueden aplicar reglas de filtrado y condiciones lógicas para generar una consulta.

La interfaz Session de Hibernate ofrece el método createCriteria () que devuelve instancias de la clase de persistencia especificada, cuando la aplicación ejecuta la consulta basada en criterios. A continuación, se muestra un ejemplo de este método para obtener una lista de todos los objetos de la clase Clientes.

Criteria r = session.createCriteria (Clientes.class);

List resultados = r.list();

Así también, podemos añadir Restricciones para obtener información específica, por ejemplo, si necesitamos obtener todos los Clientes que poseen una edad igual a 30 años, podríamos agregar la siguiente restricción:

Criteria r = session.createCriteria (Clientes.class);

r.add (Restrictions.eq ("edad", 30));

List resultados = r.list();

Dependiendo de la situación, se pueden implementar diferentes tipos de restricciones que ofrece Hibernate, tales como: validar propiedades nulas, comparar el valor de una propiedad entre rango de valores, validar propiedades que inician con un carácter en específico, entre otras.

La clase org.hibernate.criterion.Projections es una fábrica de instancias de Proyección, que puede ser aplicada a una consulta con el fin de recuperar algunos valores especiales como: valores máximos y mínimos, valor promedio de una propiedad o para contar el número de objetos que se obtiene en una consulta.

Por ejemplo, si necesitamos conocer el promedio de salarios que poseen los Clientes de cierta compañía, podríamos implementar una proyección para el cálculo de promedios AVG respecto a la propiedad salario de la clase Clientes.

Criteria c = session.createCriteria (Clientes.class).

setProjection (Projections.avg ("salario"));

Integer prom = (Integer) criteria.uniqueResult ();

Fácil integración con otras plataformas

Una de las grandes virtudes de Hibernate es la integración con otros frameworks, como: Struts, Spring o JavaServer Faces (JSF), facilitando el desarrollo de aplicaciones basadas en un entorno web. Para poder realizar este tipo de integración, se requiere del uso de un modelo de arquitectura de N-Capas, tal como se ilustra en la figura 3; en la cual se observa diferentes niveles o capas: [5]

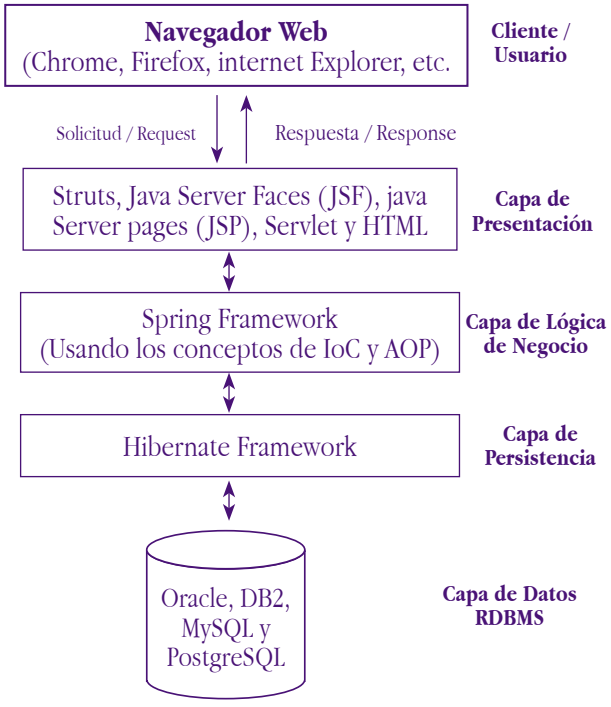
Capa de presentación: Tiene la responsabilidad de atender las solicitudes de los usuarios y de remitir las respuestas a través del navegador web. Proporciona a los usuarios diferentes vistas, para ello, es posible el uso de los frameworks: Struts, Spring o Java Server Faces (JSF); así también, el uso de componentes: Java Server Pages (JSP) o Servlet; y documentos HTML.

Capa de negocio: Es la responsable de manejar la lógica de negocio y realizar diversas operaciones como

la gestión de transacciones; regularmente, en esta capa se definen clases y procedimientos; en ese sentido, está capa es manejada por el framework Spring, el cual posee características como la programación orientada a aspectos y la inyección de dependencia. Así también, esta capa hace referencia a la interacción con la capa de datos o de persistencia, para ello, es posible usar el componente Spring DAO, que ofrece una compatibilidad con Hibernate.

Capa de persistencia: Se utiliza generalmente con el propósito de establecer la conectividad con la base de datos; ocupándose del mapeo entre la capa de negocio y la capa de base de datos. Aquí por lo general se implementan las herramientas ORM como iBATIS ó JPA; sin embargo, para el caso de estudio que se presenta, se utiliza Hibernate, el cual permite una conectividad más eficiente, mediante la eliminación de los tediosos mecanismos relacionados con el manejo de errores de conexión que provoca el uso de JDBC.

Fig. *3. Modelo de arquitectura de N-Capas*



Conclusiones

Como resultado del estudio realizado se logró corroborar que el framework Hibernate es uno de los principales ORM de código abierto que ofrece un alto desempeño comparable a muchos otros de tipo comercial; es notable, que los desarrollo de aplicaciones en los cuales se haga uso de Hibernate pueden reducir en gran medida la cantidad de tiempo y esfuerzo necesarios para codificar, probar e implementar la persistencia.

El framework Hibernate se ha logrado posicionar como una de las mejores soluciones ORM de alto desempeño que existen hoy en día; su fácil integración con otros framework como: Struts, JSF y Spring, la convierten en una herramienta poderosa para el desarrollo de aplicaciones empresariales, favoreciendo el acoplamiento y separación de las tecnologías en capas. Así también, con la tecnología de almacenamiento en caché que posee, para una rápida recuperación de los datos mejora el rendimiento del sistema; de tal forma, que si usamos los patrones de diseño y la integración de frameworks adecuados, podremos garantizar la calidad del software.

Podemos concluir que Hibernate persigue un objetivo de diseño bastante interésate, el cual es liberar al desarrollador del 95% de las tareas de programación relacionadas con la persistencia de datos comunes, eliminando la necesidad del procesamiento de los datos de forma manual haciendo uso de SQL y JDBC. Sin embargo, a diferencia de muchas otras soluciones de persistencia, Hibernate no oculta el poder que aporta SQL; al contrario, trata de garantizar que los conocimientos enfocados al modelo relacional son tan válidos y útiles como siempre.

Referencias

[1] Joseph B. Ottinger, Dave Minter, Jeff Linwood (2013) *"Beginning Hibernate"*. Editorial Apress, 3ra. Edición.

[2] Madhusudhan Konda (2014), *"Just Hibernate - A lightweight introduction to the Hibernate framework"*. Editorial O'Reilly, 1ra. Edición.

[3] *"Hibernate Annotations"*. Consultado en el mes de Abril de 2015 en el sitio web http://www.tutorialspoint.com/hibernate/hibernate_annotations.htm

[4] *"Association Mappings"*. Consultado en el mes de Abril de 2015 en el sitio web http://www.tutorialspoint.com/hibernate/hibernate_or_mappings.htm

[5] Ankur Bawiskar, Prashant Sawant, Vinayak Kankate.*"Integration of Struts, Spring and Hibernate for an University Management System"*. ISSN 2250-2459, Volume 2, Issue 6, June 2012.

[6] Red Hat, Inc. (2015). *"Hibernate Developer Guide"*. Consultado en marzo de 2015 en http://docs.jboss.org/hibernate/orm/4.2/devguide/en-US/html/

[7] *"Using Hibernate in a Web Application"*. Consultado en el marzo de 2015 en https://netbeans.org/kb/docs/web/hibernate-webapp.html

[8] Steve Perkins (2013). *"Hibernate Search by Example"*, Editorial Packt Publishing, 1ra. Edición..

[9] Oracle. "*Hibernate Tutorial: Developing a Hibernate Application"*. Consultado en el mes de marzo de 2015 en http://docs.oracle.com/cd/E13224_01/wlw/docs101/guide/index.html

[10] Elizabeth Thomas (2014). *"Building Java Web Application Using Hibernate With Spring"*. Consultado en marzo de 2015 en http://www.javacodegeeks.com/2014/03/building-java-web-application-using-hibernate-with-spring.html

Los Autores

M.C.E. Christian Mauricio Castillo Estrada

Licenciado en Sistemas Computacionales por la Universidad Autónoma de Chiapas, México y posee el grado de Maestro en Comercio Electrónico con la Especialidad en Tecnologías de Información por el Instituto Tecnológico y de Estudios Superiores de Monterrey, México. Actualmente, se desempeña como Profesor de Tiempo Completo en la Universidad Autónoma de Chiapas; así mismo, pertenece al Sistema Estatal de Investigadores (SEI) del Consejo de Ciencia y Tecnología del Estado de Chiapas (COCYTECH), México; se especializa en el desarrollo de aplicaciones empresariales basadas en web, aplicaciones móviles y realidad aumentada.

M.C.C. Karina Cancino Villatoro

Licenciado en Sistemas Computacionales por la Universidad Autónoma de Chiapas y posee el grado de Maestría en Ciencias de la Computación con Especialidad en Bases de Datos por la Universidad Valle de Grijalva, México. Actualmente, colabora en la Universidad Politécnica de Tapachula, como Profesor de Tiempo Completo. Así también, pertenece al Sistema Estatal de Investigadores (SEI) del Consejo de Ciencia y Tecnología del Estado de Chiapas (COCYETCH), México; se especializa en el área de minería de datos, bigdata, base de datos distribuidas.

M.C. Luis Antonio Álvarez Oval

Ingeniero en Computación por la Universidad Autónoma de Guadalajara, México; posee el grado de Maestro en Ciencias de la Computación por la Universidad de Houston Campus Clear Lake, USA. Actualmente, colabora en la Universidad Autónoma de Chiapas, como Profesor de Tiempo Completo; y se especializa en las áreas de Diseño y Administración de Base de Datos.