Guía del autoestopista a Hibernate.

Fecha de creación: 20.01.2003

Revisión 1.0 (20.01.2003)

Aitor García ()

http://www.javaHispano.org

Guía del autoestopista a Hibernate.

Guía de iniciación rápida en el motor de persistencia Hibernate, escrita por Glen Smith [autor] , y traducida al castellano por Aitor García [traductor]

¿Que es Hibernate?

Hibernate ofrece la *Persistencia Relacional para Java*, que para los no iniciados, proporciona unas muy buenas maneras para la persistencia de sus objetos de Java a y desde una base de datos subyacente. Más que ensuciar con SQL tus objetos y convertir consultas a y desde los objetos de primera magnitud, Hibernate puede preocuparse de todo ese maremágnum por ti. Tú utilizas solamente a los objetos, Hibernate se preocupa del SQL y de que las cosas terminan en la tabla correcta. ¿Limpio, eh?

¿Por qué esta guía?

Hibernate viene con una sensacional y amplia documentación. Esta guía es únicamente para proporcionar un arranque rápido en el aprendizaje de los fundamentos. Una vez que hayas visto esto, tendrás los fundamentos de la carga y almacenamiento de los objetos a la base de datos de tu elección, y podrás impresionar a tus amigos en el trabajo, o por lo menos mantener una conversación inteligente en tu grupo de usuarios local de Java.

Considera esto como una guía para el impaciente, si lo deseas. Una vez que estés feliz porque has hecho algún progreso y tengas un buen "feeling" con cómo funciona Hibernate, podrás explorar los trabajos más extensos que vienen con el paquete.

Hibernate es extremadamente expresivo y flexible, pero eso significa que es difícil hacerse una idea de cómo podría ser uso #simplón# de él. Aquí es adonde este documento interviene. No vamos a tratar transacciones, ni mapeos uno-a-muchos, o aún el uso de colecciones, pero no pasa nada porque la documentación de Hibernate cubre eso. Una vez que hayas vagado a través de este documento, estarás listo para lanzarte a las profundidades de todas estas buenas cosas.

Las Primeras Cosas Primero

Necesitarás una distribución de Hibernate. Estoy utilizando Hibernate 1.2, pero puede ser que desees trabajar con la copia más reciente del sitio de hibernate [1] .

El Proceso Del Desarrollo.

Hay varias maneras de acercarse al desarrollo con Hibernate. Aquí está el que estamos utilizando nosotros hoy, porque es probablemente el más simple de entender:

1. Crea tu tabla del SQL para guardar tus objetos persistentes.
2. Crea un JavaBean que represente ese objeto en código.
3. Crea un archivo de mapeo de manera que Hibernate sepa qué características del bean se mapean a que campos del SQL.

4. Crea un archivo de propiedades de manera que Hibernate conozca la configuración JDBC para acceder a la base de datos.

5. Comienza a usar el Hibernate API.

1. Proceso de desarrollo con Hibernate

Cuando te hagas mas experto, hay herramientas para autogenerar Beans del SQL o SQL de los Beans (e incluso plug-ins que generan el fichero de mapeo por ti), pero a vamos de hacerlo de la manera larga primero, para que no te distraiga ninguna magia innecesaria.

Paso 1: Escribir el SQL

El ejemplo en el que vamos a trabajar es muy simple. Considera tener que desarrollar un subsistema básico para la gerencia de los usuarios de tu website. ¿Necesitaras una tabla de usuarios, no? Estoy pensando en algo como esto:

```
CREATE TABLE `users` (
  `LogonID` varchar(20) NOT NULL default '0',
  `Name` varchar(40) default NULL,
  `Password` varchar(20) default NULL,
  `EmailAddress` varchar(40) default NULL,
  `LastLogon` datetime default NULL,
  PRIMARY KEY  (`LogonID`)
);
```

Estoy utilizando mySql, pero eres libre para utilizar cualquier clase de base de datos para la que tengas un driver JDBC. Tenemos una tabla de usuario con una identificación de la conexión, un nombre completo, una contraseña, un email y una ultima fecha de conexión. El paso siguiente es escribir un Java Bean para manejar un usuario dado.

Paso 2: Crear el Bean

Cuando tenemos un montón de *Users* en memoria, necesitaremos una clase Java para que los contenga. Hibernate trabaja vía *reflection* en los métodos de get/set de los objetos, así que necesitaremos "bean"-ificar los objetos que deseamos persistir (lo que no es algo muy grave porque ya que íbamos a usar *beans* de todos modos, verdad?). Luego escribamos una clase parecida a esta:

```
package dbdemo;
// ...

public class User {

        private String userID;
        private String userName;
        private String password;
        private String emailAddress;
        private Date lastLogon;

        public String getID() {
               return userID;
```

```
        }

        public void setID(String newUserID) {
               userID = newUserID;
        }

        // ... a bunch of other properties
        // using getXXX() and setXXX() go in here...
        public Date getLastLogon() {
               return lastLogon;
        }

        public void setLastLogon(Date newLastLogon) {
               lastLogon = newLastLogon;
        }

}
```

En nuestro ejemplo arriba, hemos hecho los accessors (`getID()`, `getLastLogon()` ,etc#) publicos # pero este no es un requisito de Hibernate - puede utilizar los métodos públicos, protegidos o incluso privados para persistir tus datos.

Paso 3: Escribir el archivo de Mapeo

Ahora ya hemos conseguido nuestra tabla del SQL, y el objeto de Java que va a mapearse con ella. Todavía necesitamos una manera de decir a Hibernate cómo mapear uno con otro. Esto se logra a través de un archivo de mapeo. La manera más limpia (la más mantenible) es escribir un archivo de mapeo por objeto, y si lo nombras *YourObject.Hbm.xml* y lo pones en el mismo directorio que su objeto verdadero, Hibernate te hará las cosas incluso más fácil. Aquí está un ejemplo de cómo podría verse *User.hbm.xml*:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping.dtd">
<hibernate-mapping>
   <class name="dbdemo.User" table="users">
      <id name="ID" column="LogonId" type="string">
        <generator class="assigned"/>
      </id>
      <property name="userName" column="Name" type="string"/>
      <property name="password" type="string"/>
      <property name="emailAddress" type="string"/>
      <property name="lastLogon" type="date"/>
   </class>
</hibernate-mapping>
```

Miremos algunas líneas de nuestro fichero. La primera etiqueta de interés es la etiqueta `class`. Aquí mapeamos el nombre de la clase (en el paquete del dbdemo) a la tabla de usuarios en nuestra base de datos.

Obviemos la etiqueta id por un momento y hablemos de las etiquetas `property`. Un vistazo rápido mostrará que aquí es donde se está haciendo el trabajo importante. El atributo `name` es la característica en nuestro Bean, y `column` es el nombre del campo en nuestra base de datos. El atributo `type` es opcional (Hibernate utilizará reflection para

conjeturar el tipo si tu no lo pones) - pero lo he puesto para que se viese de manera completa.

Bien. Volvamos a la etiqueta `id`. Quizás hayas sospechado que esta etiqueta tiene algo que ver con el mapeo de la clave primaria de la tabla. Tienes razón. La sintaxis de la etiqueta id es muy similar a la de las etiquetas de características que acabamos de ver. Mapeamos la característica del Bean (`name`) al campo de la base de datos(`column`).

La etiqueta embebida del generador le dice a Hibernate cómo debe producir la clave primaria (es bastante divertido que Hibernate te genere una, del tipo prefieras, pero necesitaras decirle cómo). En nuestro caso, lo fijamos a asignado, de manera que nuestro bean va a generar sus propias claves (después de todo el objeto del usuario necesitará siempre tener un UserID). Si quieres dejar que Hibernate genere las claves por ti, quizás te interesen clases como `uuid.hex` y `uuid.string` (Consulta la documentación para mas información).

Si consideras que esto del mapeo manual no es para ti, puedes hace que un plugin de tu IDE te eche una mano. Estoy utilizando uno para Eclipse llamado Hibernator [2] . O, si XDoclet [3] es más de tu preferencia, hay etiquetas de Hibernate para XDoclet que permiten que meter los mapeos directamente en el código fuente, y después autogenerar tu archivo de mapeo en tiempo de *build*. ¡Guay!

Paso 4: Crear un archivo con las características para su base de datos

Todavía no le hemos dicho a Hibernate dónde encontrar nuestra base de datos. La manera más directa es proporcionar a Hibernate un fichero de las características, con los ajustes para las secuencias de la conexión, las contraseñas, etc. Si llamas a este archivo `hibernate.properties` y lo pones en el classpath, Hibernate lo tomará automagicamente (sic). El fichero tendría un aspecto como este:

```
hibernate.connection.driver_class = org.gjt.mm.mysql.Driver
hibernate.connection.url = jdbc:mysql://localhost/testdb
hibernate.connection.username = gsmith
hibernate.connection.password = sesame
```

Si has utilizado JDBC, éste te resultara familiar. En el ejemplo de arriba un driver de mysql, se conecta con la base de datos testdb en localhost, y utiliza el usuario y contraseña proporcionados. Hay un montón más de características que puedes ajustar para determinar cómo Hibernate tiene acceso a tu base de datos. De nuevo, consulta la documentación para más información detallada.

Paso 5: Comience a hacer magia con Hibernate

Todo el trabajo duro se ha hecho. Si ha ido todo bien hasta ahora, tendrás lo siguiente:

» `User.java` - tu objeto Java a persistir

» `User.hbm.xml` - tu archivo de Mapeo Hibernate

»

`hibernate.properties` # tu archivo de propiedades con información sobre la conexión JDBC (Puedes hacer esto en el código si lo prefieres)

» Una tabla Usuario en su base de datos

2. Estado actual

Hacer uso de Hibernate en tu código fuente es bastante directo. Ahí va la versión reducida:

1. Cree un objeto *Datastore*
2. Dile al *Datastore* el tipo de objetos que deseas almacenar
3. Crea una sesión a la base de datos de tu elección
4. Carga, guarda y consulta tus objetos
5. `flush()` tu sesión de vuelta a la base de datos

3. Guía rápida de uso de Hibernate

Vamos a mirar el código fuente para verlo todo mucho mas claro.

Primero, creamos el objeto de Datastore...

El objeto Datastore tiene conocimiento de todos los mapeos que existen entre los objetos de Java y la base de datos.

```
Datastore ds = Hibernate.createDatastore();
ds.storeClass(User.class);
```

Esto asume que tienes un User.hbm.xml en el mismo directorio que la clase user. Si decides llamar a tu archivo de otra manera, utiliza:

```
ds.storeFile("MyMappingFile.hbm.xml")
```

Entonces, crea un objeto de sesión...

El objeto de sesión representa una conexión a su base de datos. Puedes proporcionar a Hibernate una sesión de JDBC que hayas creado a mano, pero soy perezoso, así que solo voy a proporcionarle a Hibernate el archivo de propiedades, y dejar que haga el trabajo por mí.

```
// Then build a session to the database
SessionFactory sf = ds.buildSessionFactory();
```

```
        // or supply a Properties arg
        Session session = sf.openSession();
```

He llamado a mi archivo de propiedades `hibernate.properties` y lo he puesto en mi classpath, pero si has llamado al tuyo de manera diferente (o quieres crear las propiedades en el código), entonces necesitaras proporcionar tu propio objeto de propiedades a `buildSessionFactory()`.

¡Después carga, guarda y consulta tus objetos!...

Ahora solo tienes que usar los objetos de una manera Java. ¿Quieres almacenar un nuevo usuario en la base de datos? Seria algo así:

```
        // Create new User and store them the database
        User newUser = new User();
        newUser.setID("joe_cool");
        newUser.setUserName("Joseph Cool");
        newUser.setPassword("abc123");
        newUser.setEmailAddress("joe@cool.com");
        newUser.setLastLogon(new Date());

        // And the Hibernate call which stores it
        session.save(newUser);
```

Como puedes ver, la gran ventaja de Hibernate es el poco código adicional a emplear. La mayor parte del tiempo solo te preocupas de tus objetos de negocio, y haces una sola llamada a Hibernate cuando todo esta preparado.

Pongamos por caso que deseas recuperar un objeto y sabes el *UserID* del usuario (ej. durante un proceso de la conexión a su sitio). Con una unica línea, pasando la clave ya lo tienes :

```
    newUser = (User) session.load(User.class, "joe_cool");
```

El objeto usuario que has recuperado esta vivo! Cambia sus propiedades y se guardara en la base de datos en el próximo `flush()`.

Incluso mejor , puedes consultar tu tabla y obtener una `java.util.List` de objetos vivos. Prueba con algo como esto:

```
List myUsers = session.find("from user in class dbdemo.User");

if (myUsers.size() > 0) {
   for (Iterator i = myUsers.iterator(); i.hasNext() ; ) {
      User nextUser = (User) i.next();
      System.out.println("Resetting pass: "+ nextUser.getUserName());
      nextUser.setPassword("secret");
   }
}else {
   System.out.println("Didn't find any matching users..");
```

```
}
```

Esta consulta devuelve toda la tabla . Normalmente querrás mucho mas control (ej. From user in class `dbdemo.User` where userID tenga algún valor especifico),de manera que harás algo como esto:

```
        List myUsers = session.find("from user in class dbdemo.User
where user.ID = ?","joe_cool", Hibernate.STRING);
```

Hay una gran abundancia de opción de consulta en la documentación, pero esto te da una idea de lo que Hibernate puede ofrecerte.

...Y finalmente flush() tu session

De vez en cuando tu sesión de Hibernate se hará `flush()` a si misma para mantener una correcta sincronización entre la BD y tus objetos en Memoria. Puedes querer liberar la conexión JDBC que Hibernate esta usando para lo cual normalmente harás:

```
        // close our session and release resources
        session.flush();
        session.close();
```

Y estoy gastando...

Y eso es todo . ¡Bien hecho! Has creado objetos, los has guardado, y recuperado.¡Te sentirás feliz con esto!

Ahora que ya tiene cierto conocimiento de Hibernate, puede echar un buen vistazo a la extensa documentación que viene con el paquete. Hay un montón de funcionalidades que explorar como mapeo uno-a-varios, persistencia ordenada, *nested collections*, optimización y mucho más.

Disfruta! Y Buena Hibernación!

Conclusión

Y eso es todo lo que se me ha ocurrido (herencia, polimorfismo y diseño), por ahora me he quedado satisfecho, en la próxima entrevista ya tendré claro que contestar, pero espero que todo aquel que tenga nuevas ideas para el uso de interfaces me las haga llegar.

Recursos y referencias

[autor] Web personal del autor, *http://glen.blog-city.com/*
[traductor] Web personal del traductor, *http://freeroller.net/page/aitor*
[1] Sitio web de Hibernate, *http://hibernate.sf.net*
[2] Sitio web de Hibernator, *http://hibernator.sourceforge.net*

[3] Sitio web de XDoclet, *http://xdoclet.sourceforge.net*

Acerca del autor