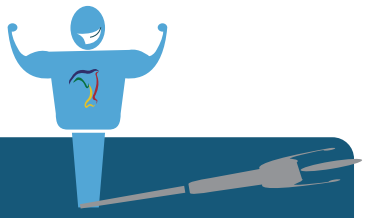
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

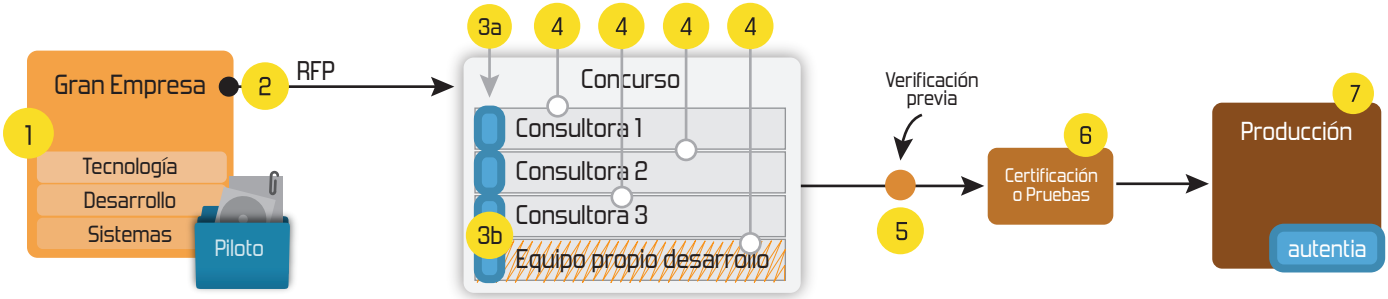
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Lanzamiento TNTConcept

Autentia da un paso más en su evolución: Lanzamiento de software propio. Ponemos a vuestra disposición el software que hemos construido para nuestra gestión interna, llamado TNTConcept (au**T**e**NT**ia).

Construida con las últimas tecnologías de desarrollo Java/J2EE (Spring, JSF, Hibernate, Maven, Subversion, etc.) y disponible en licencia GPL, seguro que a muchos profesionales independientes y PYMES os ayudará a organizar mejor vuestra operativa.

Las cosas grandes empiezan siendo algo pequeño ..... Saber más en: http://tntconcept.sourceforge.net/

Tutorial desarrollado por: Javier Antoniucci

Puedes encontrarme en Autentia
Somos expertos en Java/J2EE
Contacta en javier.antoniucci@autentia.com

Descargar este documento en formato PDF PersistenciaJava.pdf

Persistencia Básica en Java

Cuando abordamos el desarrollo de una aplicación en Java, uno de los primeros requerimientos que debemos resolver es la integración con una base de datos para guardar, actualizar y recuperar la información que utiliza nuestra aplicación.

Se llama "persistencia" de los objetos a su capacidad para guardarse y recuperarse desde un medio de almacenamiento. La persistencia en Base de Datos relacionales se suele implementar mediante el desarrollo de funcionalidad específica utilizando la tecnología JDBC o mediante frameworks que automatizan el proceso a partir de mapeos (conocidos como Object Relational Mapping, ORM) como es el caso de Hibernate.

Si bien existen más alternativas de persistencia, en este tutorial aboraremos estas dos alternativas ya que son las más frecuentemente utilizadas.

Los código fuente que se muestran a continuación son fragmentos del Zip que se puede encontrar en PersistenciaBasicaEnJava.zip

JDBC

Java DataBase Connectivity es el API de Java que define cómo una aplicación cliente accederá a una base de datos, independientemente del motor de base de datos al que accedamos.

A continuación mostramos un ejemplo del código necesario para insertar un elemento y hacer una consulta:

```
// Cargar el driver

Class.forName(jdbcDriver);

// Crear una conexión

Connection con = DriverManager.getConnection(jdbcUrl, userName,

password);

// Crear un Statement para obtener el máximo id

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT MAX(id) FROM " + schema
```

```
+ "DEMO_QUERIES");

int id = 0;

if (rs.next()) {

id = rs.getInt(1) + 1;

}

rs.close();

stmt.close();

// Crear un PreparedStatement para insert

PreparedStatement prepareInsert = con.prepareStatement(

"INSERT INTO " + schema + "DEMO_QUERIES (id, nombre, fecha)”

+ “VALUES (?, ?, ?)");

// Completar sus atributos

int i = 1;

prepareInsert.setInt(i++, id);

prepareInsert.setString(i++, "Ejemplo JDBC");

prepareInsert.setDate(i++, new Date((new Date()).getTime()));

// Ejecutar el insert

prepareInsert.executeUpdate();

prepareInsert.close();

// Crear un PreparedStatement para select

PreparedStatement prepareSelect = con

.prepareStatement("SELECT id, nombre, fecha FROM " + schema

+ "DEMO_QUERIES WHERE fecha < ?");

// Completar sus atributos

prepareSelect.setDate(1, new Date((new java.util.Date()).getTime()

+ DAY_IN_MILLIS));

// Ejecutar la select

rs = prepareSelect.executeQuery();

while (rs.next()) {

System.out.println("Id = " + rs.getInt(1) + " - nombre = "

+ rs.getString(2) + " - fecha = " + rs.getDate(3) + "");

}

rs.close();

prepareSelect.close();
```

Como vemos el código es esencialmente simple aunque si tenemos un número amplio de clases y problemáticas de maestro/detalle (como pueden ser facturas con items) o soporte multi-base de datos la implementación puede volverse muy difícil de mantener.

Una mejora a la mantenibilidad es utilizar el patrón de diseño Data Access Object (DAO, Objeto de Acceso a Datos ver http://es.wikipedia.org/wiki/Data_Access_Object) donde básicamente definimos una interfaz y una clase que concentren toda la funcionalidad de creación, lectura, actualización y borrado en la base de datos.

Hibernate

En Hibernate esta misma funcionalidad se simplifica a:

```
Entidad entidad = new Entidad();

entidad.setId(id);

entidad.setNombre("Ejemplo Hibernate\n");


Session session = sessionFactory.getCurrentSession();

session.beginTransaction();

session.save(entidad);

session.getTransaction().commit();
```

O para obtener el listado:

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

session.beginTransaction();

Query q = session

.createQuery("from e in class com.persistencia.hibernate.Entidad where e.fecha < :fecha");

// se inyecta el valor del parámetro utilizando el nombre

q.setParameter("fecha", new java.sql.Date(fecha.getTime()));

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

¿¡¿Pero cómo funciona la magia?!? Bueno, Entidad es un POJO (Plain Old Java Object), es decir una simple clase Java que tiene métodos get y set para cada uno de los atributos; así que ahí no está la magia.

Es en el fichero de mapeo donde ponemos la magia, que tampoco es tan complicada:

```
<hibernate-mapping>

<class name="com.hemasoft.demos.queries.hibernate.Entidad"

table="DEMO_QUERIES">

<id name="id" type="java.lang.Integer" />

<property name="nombre" />

<property name="fecha" type="java.sql.Date" />

</class>

</hibernate-mapping>
```

Este fichero de mapeo lo declaramos en hibernate.cfg.xml junto a la información de acceso a la base de datos de la siguiente forma:

```
<hibernate-configuration>


<session-factory>


<!-- Database connection settings -->

<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

<property name="connection.url">jdbc:oracle:thin:@192.168.1.20:1521:qemu</property>

<property name="connection.username">test</property>

<property name="connection.password">test</property>


<!-- JDBC connection pool (use the built-in) -->
```

```
<property name="connection.pool_size">1</property>

<!-- SQL dialect -->

<property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>

<mapping resource="com/hemasoft/demos/queries/hibernate/Entidad.hbm.xml"/>

</session-factory>

</hibernate-configuration>
```

Creo que el fichero es lo suficientemente intuitivo como para no necesitar mayores explicaciones.

Los costes de aprendizaje de este framework en términos de tiempo dedicado se van recompensando desde el primer día con el ahorro en tiempo de desarrollo y depuración de código JDBC. Incluso, a partir de la versión 3 de Hibernate su configuración se simplifica notablemente mediante Anotaciones (recomiendo ver http://www.hibernate.org/hib_docs/annotations/reference/en/html_single/#entity-mapping).

Además, Hibernate nos proporciona grandes beneficios como:

- soporte a múltiples motores de base de datos
- bajo acoplamiento entre negocio y persistencia, ya que su diseño está orientado a objetos así como el soporte a consultas y operaciones (HQL).
- desarrollo robusto, ya que el framework ha madurado tras años de uso en decenas de miles de proyectos
- optimizado, ya que el SQL generado contiene optimizaciones específicas para cada motor de base de datos mediante componentes especializados llamados dialectos.
- rápido y completo, ya que con la funcionalidad estándar de Hibernate podremos cubrir el 80 - 90% de la persistencia de nuestra aplicación

Todo esto nos permite centrar nuestros esfuerzos en desarrollar la funcionalidad de la aplicación.

Conclusiones

El acceso mediante JDBC conviene utilizarlo cuando tenemos pocas clases, escasos conocimientos, escaso tiempo para formación, modelo de datos muy desnormalizado o alguna otra razón de peso.

Para todo lo demás, Hibernate es una solución madura y dinámica que nos permite simplificar a la vez que robustecer nuestro acceso a bases de datos.

Desde Autentia (http://www.autentia.com) os animamos a que utilicéis este tipo de tecnologías. Basta ya de reinventar la rueda en cada desarrollo. Debemos construir en función de patrones y estándares, y reutilizando al máximo. Esto nos permite centrarnos en la lógica de negocio, reducir los tiempos de desarrollo, y aumentar la calidad del resultado.

Puedes opinar sobre este tutorial aquí

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
CachedRowSet: JDBC y Java 5.	En este tutorial aprenderemos a el uso y funcionamiento del rowSet de tipo CachedRowSet.
Algunas características menos conocidas del api JDBC 2.0	En este tutorial veremos algunas de las caraterísticas que menos se conocen del api JDBC.
Introducción a Hibernate	Cesar Crespo nos enseña como utilizar unos de los sistemas más extendidos de mapeo de objetos a estructuras relacionales (tablas de base de datos)
Introducción a JDBC	En este tutorial os explicamos los fundamentos teóricos de JDBC
Creación automática de recursos Hibernate con Middlegen	En este tutorial aprendereis como utilizar la herramienta middlegen para generar distintas capas de persistencia (CMP 2.0, JDO, Hibernate, Torque), a partir de un modelo físico de datos, de un modo automático, mediante el uso de la herramienta middlegen
Manejar dos bases de datos distintas con Hibernate	Alejandro Pérez nos enseña como manejar dos bases de datos distintas con Hibernate
Hibernate 3.1, Colecciones, Fetch y Lazy	En este tutorial vamos a ver cómo se comportan ciertas relaciones, y cómo podemos optimizar las consultas a la base de datos con Hibernate
JDBC y MySql	En el tutorial anterior vimos como instalar MySQL en Windows, ahora vamos a ver como acceder desde una aplicación Java.
Generación automática de código JDBC	En este tutorial os enseñamos como, sin conocimiento de JDBC, crear vuestro programas en Java, gracias a JDBCTest.