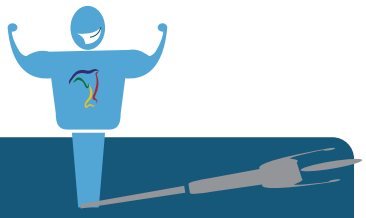
autentia

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...

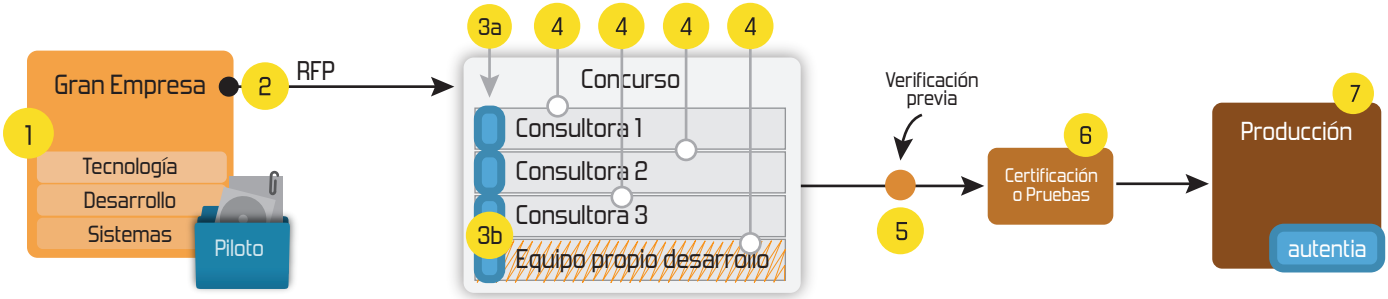
## 1. Desarrollo de componentes y proyectos a medida



## 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**

Jose Manuel Sánchez Suárez

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2012-09-28 Tutorial visitado 5 veces Descargar en PDF

# Creación de una base de datos embebida en memoria con el soporte de Spring.

## 0. Índice de contenidos.



## 1. Introducción

A partir de la versión 3.0, Spring proporciona un espacio de nombres específico en el esquema de configuración xml, para crear una base de datos embebida y en memoria en el arranque de nuestra aplicación java.

Hasta ahora podíamos crearla fácilmente publicando un bean de tipo DataSource y configurando los parámetros de la conexión, ahora es mucho más simple y, además, nos permite procesar ficheros sql después de su creación para crear la estructura de las tablas o realizar una carga de datos inicial.

Por defecto, Spring proporciona soporte para:

- HSQL (por defecto), http://hsqldb.org/
- H2, http://www.h2database.com/html/main.html
- Derby, http://db.apache.org/derby/

Usamos una base de datos embebida y en memoria para ejecutar nuestros tests de integración contra una base de datos real, aunque no física; para que la ejecución de los mismos no dependan de su estado; de tal modo que se crea y se destruye en el ámbito de ejecución de los tests.

## 2. Entorno.

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook Pro 15' (2.4 GHz Intel Core i7, 8GB DDR3 SDRAM).
- Sistema Operativo: Mac OS X Lion 10.7.4
- Spring 3.1.

## 3. Configuración.

Lo primero, como siempre, configurar las dependencias de las librerias necesarias, con el soporte de Maven:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>3.1.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
```

```
    <version>3.1.0.RELEASE</version>
</dependency>
    <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>3.1.0.RELEASE</version>
</dependency>
</dependency>
    <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.3.154</version>
    <scope>test</scope>
</dependency>
```

### 4. Creación de la base de datos embebida.

Sin el soporte del espacio de nombres, la creación de la base de datos pasaba por configurar un bean como el siguiente en nuestro applicationContext.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schem

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource
    <property name="driverClassName" value="org.hr.Driver" />
    <property name="url" value="jdbc:h2:mem:test" />
    <property name="username" value="sa" />
    <property name="password" value="" />
</bean>

</beans>
```

Ahora con el espacio de nombres, la configuración se reduce sensiblemente puesto que solo hay que incluir lo siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:jdbc="http://www.springf
xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema
        http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/

<jdbc:embedded-database id="dataSource" type="H2" />

</beans>
```

Muy simple y, además, nos permite configurar referencias a ficheros sql que se procesarán después de crear la base de datos, para crear la estructura de tablas o poblarlas con datos iniciales.

```
<jdbc:embedded-database id="dataSource" type="H2">
    <jdbc:script location="classpath:ddl.sql"/>
    <jdbc:script location="classpath:dml.sql"/>
</jdbc:embedded-database>
```

### 5. Creación de la estructura de tablas con el soporte de Hibernate y carga de datos iniciales.

En el entorno de test, haciendo uso de un ORM como Hibernate o cualquiera de las implementaciones de JPA (el propio EntityManager de Hibernate) podemos crear la estructura de las tablas de base de datos (de esta que hemos configurado embebida y en memoria) en función de la configuración de nuestras clases de entidad.

Para ello, no tenemos más que incluir una propiedad específica para configurarlo en la definición de la factoría de sesiones:

```
<jdbc:embedded-database id="dataSource" type="H2" />

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.annotation.Annotatic
    <property name="dataSource" ref="dataSource" />
    <property name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguratior
    <property name="configLocation">
        <value>classpath:hibernate.cfg.xml</value>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.hbm2ddl.auto">true</prop>
        </props>
    </property>
    <property name="packagesToScan" value="com.autentia.training.**.*" />
</bean>
```

Si, además de lo anterior, tuviéramos la necesidad de poblar la base de datos con una carga inicial, podríamos añadir la siguiente configuración:

```
<jdbc:initialize-database data-source="dataSource">
    <jdbc:script location="data.sql" />
</jdbc:initialize-database>
```

Con ello, despúes de crear la base de datos, se ejecuta el contenido del script sql y tendremos los datos listos para nuestro entorno de tests.

No debemos olvidar que los test, aunque sean de integración, deben ser atómicos y deben dejar el estado de la base de datos consistente, de modo que la base de datos, después de la ejecución del test se mantenga en su estado inicial.

Para ello, también son el soporte de Spring y haciendo uso de la gestión declarativa de transacciones, debemos marcar todos los métodos de los test como transaccionales y, con la estrategia por defecto de defaultRollBack, Spring se encargará de no confirmar los cambios tras la salida de cada método.

```
@RunWith(SpringJUnit4ClassRunner.class)
```

```
@ContextConfiguration(locations =
{ "classpath:applicationContext-test.xml" })
@Transactional
public class CatalogoDaoTest {

    @Test
    public void should...

}
```

Con la anotación en la cabecera de la clase de tests, después de la ejecución de cada método se producirá un rollback.

### 6. Conclusiones.

Si no tenemos el soporte de una herramienta como liquibase, la configuración que hemos visto en este tutorial puede ser una buena alternativa.

Un saludo.

Jose

jmsanchez@autentia.com