Conectando con Postgresql usando JNDI y Spring en Tomcat.

(Parte 1)

1 Introducción

En este ejemplo veremos como conecta a Postgresql usando JPA + Hibernate y Spring JDBC. Para ello uso como servidor de aplicaciones Tomcat. La configuración esta realizada con anotaciones y XML y utilizo el pool de conexiones de Tomcat recogido a través de JNDI, para que la aplicación no deba saber realmente ni donde se conecta.

El código fuente lo teneis en: https://github.com/chuchip/jdbc_jpa_eclipse_tomcat

El ejemplo usa Maven y explicare como deberá estar configurado Tomcat para que la aplicación funcione correctamente.

2 Configuración de Postgresql:

Esta sera la única tabla a la que accederemos a través de Postgresql:

```
create table usuario
(
	login varchar(15) not null,
	nombre varchar(100) not null,
	constraint ix_usuario primary  key (login)
);
```

3 Configuración de Tomcat:

La configuración de Tomcat deberá tener las siguientes características:

En server.xml (estara en $TOMCAT_HOME/conf) deberemos añadir dentro de **<GlobalNamingResources>** las siguientes lineas:

```
 <GlobalNamingResources>
…..
<Resource name="jdbc/anjelica" auth="Container"
        type="javax.sql.DataSource" driverClassName="org.postgresql.Driver"
        url="jdbc:postgresql://localhost:5432/MIBASEDATOS"
        username="USUARIO" password="CONTRASEÑA" maxTotal="20" maxIdle="10" maxWaitMillis="-1"/>
…..
 </GlobalNamingResources>
```

Esto se utilizara para configurar nuestra fuente JNDI que permitirá conectarnos con la base de datos. Para probar esta aplicación cambiar los valores de *url* para que apunten a vuestra base de datos, asi como el *username* y la *password.*

También sera necesario añadir las siguientes lineas al fichero *context.xml dentro de* ***<context>*** de tomcat, que también estará en $TOMCAT_HOME/conf

```
<Context>
….
        <ResourceLink name="jdbc/anjelica"
                global="jdbc/anjelica"
                type="javax.sql.DataSource"/>
 ….
</Context>
```

Con estos dos ficheros ya tendremos nuestro tomcat configurado para que use su propio pool de conexiones al que hemos llamado “jdbc/anjelica”.

Ahora debemos añadir a Tomcat la librería para conectarnos a postgresql, en este caso usamos la versión 42.2.2

- postgresql-42.2.2.jar

4 Configuración de la aplicación.

En esta aplicación usaremos tanto ficheros xml como configuración en Java.

Lo primero es configurar nuestro ficheros xml para que Tomcat use Spring, para ello en el directorio **src\main\webapp\WEB-INF** de nuestra aplicación tenderemos estos ficheros.

- applicationContext.xml
- dispatcher-servlet.xml
- web.xml

4.1 web.xml

El único fichero que usa Tomcat es web.xml, y este, a su vez, usa los dos anteriores, de tal manera que básicamente en web.xml , lo primero que hacemos es especificar que use el servlet de Spring y le decimos donde tendrá la configuración para ese servlet. Esto se hace con estas lineas:

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>
       org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
       <param-name>contextConfigLocation</param-name>
       <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

A continuación configuramos el contexto con las siguientes lineas:

```
<context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/applicationContext.xml</param-value>
  </context-param>
  <listener>
      <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
```

4.2 dispatcher-servlet.xml

Para configurar la parte Web de nuestra aplicación (el servlet realmente) pondremos las siguientes lineas en el fichero dispatcher-servlet.xml

```
<annotation-driven />

 <context:component-scan base-package="chu.jdbc" />

 <beans:bean
  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <beans:property name="prefix" value="/WEB-INF/views/" />
  <beans:property name="suffix" value=".jsp" />
 </beans:bean>
```

Con **<annotation-driven />** permitiremos que en nuestra aplicación haya anotaciones @Controller y *@RequestMapping*
Con ***<context:component-scan base-package="chu.jdbc" />*** especificaremos que paquete deberá escanear Spring para buscar anotaciones en los ficheros java. En este caso especificamos que busque en el paquete “chu.jdbc” y sus hijos.

Las ultimas lineas indican que usaremos JSP y especifica donde tendremos nuestros ficheros jsp.

Tenéis un excelente documento explicando como hacer esta misma configuración usando anotaciones java en la siguiente página: https://www.baeldung.com/bootstraping-a-web-application-with-spring-and-java-based-configuration

4.3 applicationContext.xml

En este fichero crearemos nuestro DataSource que utilizaremos para conectarnos con el pool de conexiones anteriormente configurado en Tomcat.
Esto se hace añadiendo la siguiente linea:

```
<jee:jndi-lookup id="dataSource" jndi-name="jdbc/anjelica"   resource-ref="true"/>
```

Tenéis más documentación de lo que hace esta linea en http://www.jtech.ua.es/j2ee/publico/spring-2012-13/sesion01-apuntes.html

De todos modos básicamente lo que hace es injectar en nuestra aplicación una clase DataSource que luego podremos usar con sentencias como esta:

```
 @Autowired
   DataSource ds;
```

5 La aplicación

5.1 Configuracion JPA y JDBC

En la clase JpaConfig es donde se hace toda la configuración que necesitamos para conectarnos a la base de datos.

Para ello, lo primero especificamos las siguientes configuraciones java

```
@Configuration
@EnableLoadTimeWeaving
@EnableJpaRepositories("chu.jdbc")
```

Explico las directivas, una a una.

- @Configuration Especifica que es una clase de configuración, con ello haremos que Spring la cargue y ejecute.
- @EnableLoadTimeWeaving Con esto permitimos a tomcat usar AOP. Si no la ponemos simplemente Tomcat no podrá hacer uso de Programación Orienta a a Aspectos (AOP) y la aplicación no funcionara. Mas documentación en: https://docs.spring.io/spring/docs/5.0.8.RELEASE/spring-framework-reference/core.html#aop-aj-ltw-environment-tomcat
- @EnableJpaRepositories("chu.jdbc") especificamos que busque repositorios (clases marcadas con la directiva @Repository) en el paquete *chu.jdbc*

Ahora inyectamos el Datasource que ya tendremos disponible por la configuración aplicada anteriormente en nuestro fichero *applicationContext.xml*

```
@Autowired
DataSource ds;
```

Lo siguiente es crear nuestro fabrica de controladores de entidades (bueno, en ingles, nuestro EntityManagerFactoryBean), para ello y como usamos una versión de Hibernate superior a la 4 (con versiones anteriores utilizaríamos otra clase), ponemos el siguiente código:

```
@Bean
  public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
    LocalContainerEntityManagerFactoryBean em  = new LocalContainerEntityManagerFactoryBean();
    em.setDataSource(ds);
    em.setPackagesToScan(new String[] { "chu.jdbc" });

    JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
    em.setJpaVendorAdapter(vendorAdapter);

    return em;
  }
```

Como se puede ver simplemente creamos nuestro *LocalContainerEntityManagerFactoryBean* , le añadimos nuestro DataSource y el adaptador de Hibernate. También le decimos los paquetes que deberá escanear para buscar “Entity Classes” (clases que sean entidades, vamos).

Por ultimo creamos nuestro controlador de transaciones con el siguiente código:

```
@Bean
  public PlatformTransactionManager transactionManager( EntityManagerFactory emf){
     JpaTransactionManager transactionManager = new JpaTransactionManager();
     transactionManager.setEntityManagerFactory(emf);
     return transactionManager;
  }
```

Ahora ya solo falta crear nuestro ficherito hibernate.cfg.xml en nuestro directorio de *resources*, pero en el solamente tenemos lo siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
                        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
                        "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
   </session-factory>
</hibernate-configuration>
```

Es decir, simplemente definimos que vamos a usar una base de datos Postgresql. Este fichero **debe** existir. Cosas de Hibernate que hay que complacer.

Y *voila* nuestro entorno para JPA + Hibernate ya esta configurado y listo para funcionar.