

Tutorial para desarrollar un proyecto con
Spring farmework 4 y hibernate 4 con el
patrón de diseño MVC.

Spring MVC 4 y Hibernate 4

Jonathan Enciso Carrillo

Contenido

| | |
|---|----|
| Introducción | 2 |
| Requisitos | 2 |
| Previo. | 2 |
| Crear el proyecto..... | 2 |
| Configuración del archivo POM | 3 |
| Configuración del web.xml..... | 5 |
| Configuración del dispatcher. | 6 |
| Estructura MVC. | 7 |
| Desarrollo | 8 |
| Configuración de Hibernate | 13 |
| Guardar usuario en la base de datos | 18 |
| vistas..... | 26 |
| Altas.jsp | 26 |
| eliminar.jsp..... | 26 |
| Listado.jsp..... | 27 |

Introducción

En este tutorial pretendo mostrarles como configurar Spring 4 y Hibernate 4. Para lograrlo, mostrare paso a paso como desarrollar un sencillo proyecto web.

Requisitos

- Maven 3
- Java 1.7
- Eclipse
- MySQL 5

Previo.

Antes de empezar verifica que tengas configurado java y maven.

Teclee Java -version en su terminal.

```
C:\Users\Jonathan>java -version
java version "1.7.0_40"
Java(TM) SE Runtime Environment (build 1.7.0_40-b43)
Java HotSpot(TM) 64-Bit Server VM (build 24.0-b56, mixed mode)
```

Teclee mvn -v para verificar maven.

```
C:\Users\Jonathan>mvn -v
Apache Maven 3.1.1 (0728685237757ffbf44136acec0402957f723d9a; 2013-09-17 10:22:22-0500)
Maven home: C:\apache-maven-3.1.1
Java version: 1.7.0_40, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.7.0_40\jre
Default locale: es_MX, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
```

Crear el proyecto.

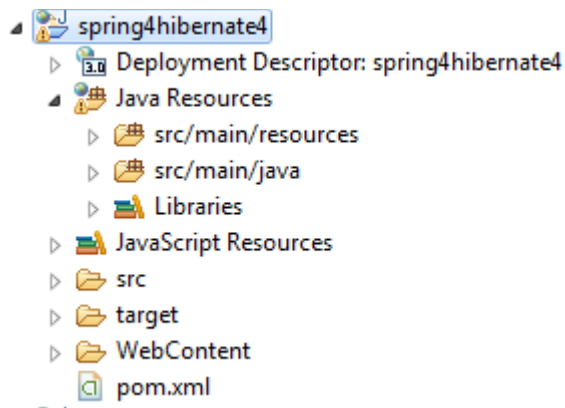
Primero vamos a crear un proyecto web con maven, desde la terminal dirijase a la ruta donde desee crear el proyecto e ingrese lo siguiente:

```
mvn archetype:generate -DgroupId=com.spring4.hibernate4 -  
DartifactId=spring4hibernate4 -DarchetypeArtifactId=maven-archetype-  
webapp -DinteractiveMode=false
```

En este punto ya debemos tener nuestra carpeta con el nombre spring4hibernate4 y dentro encontrarán otras carpetas y archivos correspondientes a un proyecto web.

Vamos a importar nuestro proyecto a eclipse. Abrimos eclipse y desde el menú file, damos clic en import. Elegimos existing Project into workspace. En select root directory, damos clic en el botón browse y buscamos la ruta donde está nuestro proyecto. Seleccionamos la carpeta, clic en aceptar y después en finish.

En este punto debemos tener esta estructura.



Debemos tener el source folder src/main/java en caso de no tenerlo, hay que crearlo, damos clic derecho en Java Resources – New – source folder. Nos aparecerá una ventana donde nos pide que pongamos el nombre del proyecto, ponemos spring4hibernate4, en el siguiente campo ponemos src/main/java y damos finish.

Configuración del archivo POM

Dentro de nuestro proyecto podemos ver un archivo llamado pom.xml, aquí es donde vamos a agregar nuestras dependencias o librerías que requiere nuestro proyecto para ser compilado. Su archivo pom debe verse como el siguiente.

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.spring4.hibernate4</groupId>
  <artifactId>spring4hibernate4</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>spring4hibernate4 Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <properties>
    <spring.version>4.0.1.RELEASE</spring.version>
  </properties>
  <dependencies>
    <!-- Spring 4 dependencies -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>
    <dependency>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</artifactId>
      <version>1.1.3</version>
    </dependency>
    <!-- end of Spring 4 dependencies -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

```

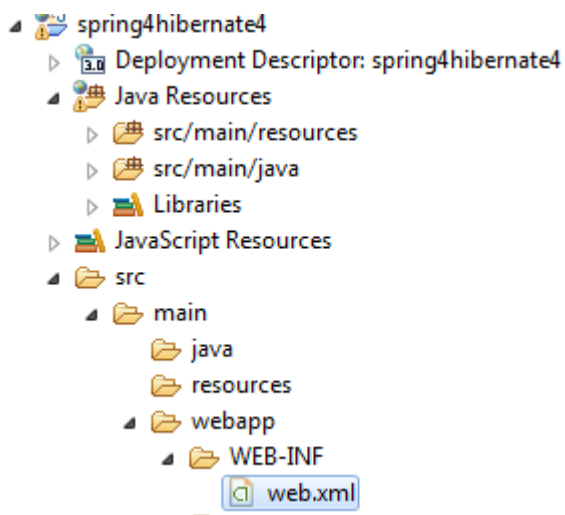
```

    <build>
      <finalName>spring4hibernate4</finalName>
    </build>
  </project>

```

Configuración del web.xml

Abrimos el archivo web.xml



Borramos el contenido y pegamos lo siguiente

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="WebApp_ID" version="2.5">
  <display-name>Spring4hibernate4</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <load-on-startup>1</load-on-startup>
  </servlet>

```

```

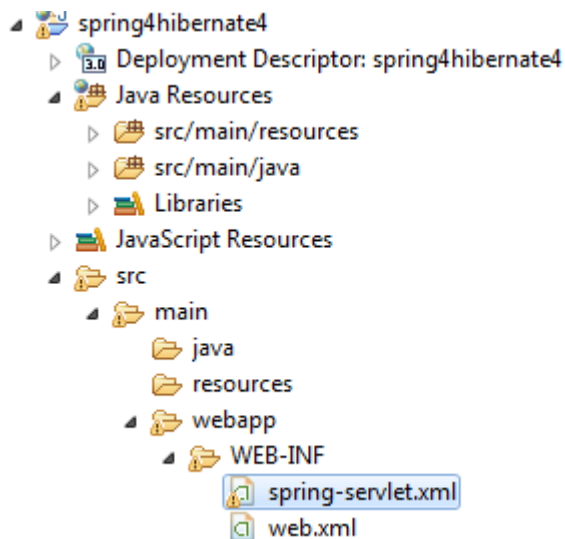
<servlet-mapping>
  <servlet-name>spring</servlet-name>
  <url-pattern>*.html</url-pattern>
</servlet-mapping>
</web-app>

```

Con las líneas anteriores estamos indicando que página se ejecutara al iniciar el proyecto, cual será nuestro servlet.

Configuración del dispatcher.

El objetivo del dispatcher será indicarle a spring donde están nuestros jsp, para eso vamos a crear el archivo spring-servlet.xml. en el siguiente directorio.



Y dentro pegamos lo siguiente

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://www.springframework.org/schema/p"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

  <context:component-scan base-package="com.spring4.hibernate4.controller" />

  <bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">
      <value>/WEB-INF/pages/</value>
    </property>
  </bean>

```

```

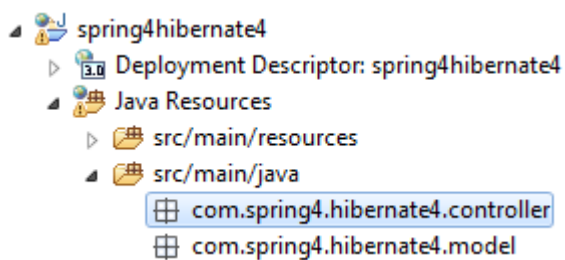
</property>
<property name="suffix">
  <value>.jsp</value>
</property>
</bean>
</beans>

```

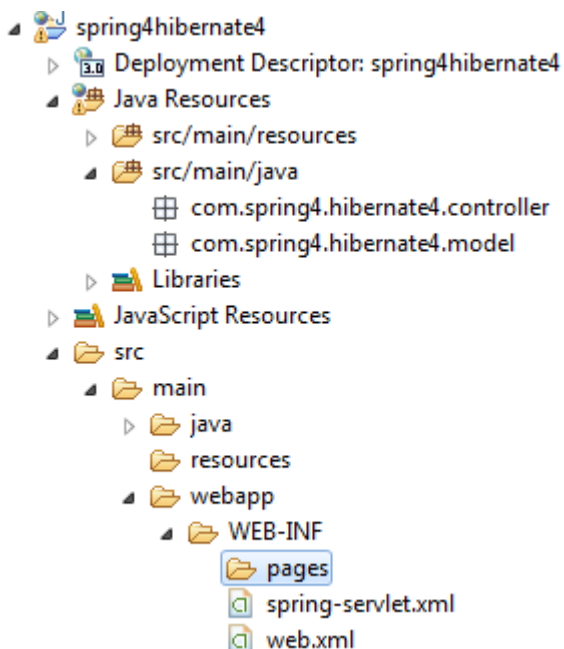
Hasta este punto ya tenemos configurado spring. Ahora vamos a proceder a organizar nuestro proyecto en base a MVC.

Estructura MVC.

Empecemos creando los siguientes paquetes:



Tenemos el paquete com.spring4.hibernte4.model donde van a ir pojo. Y tenemos el paquete com.spring4.hibernate4.controller donde van a ir nuestros controladores. Para la vista, vamos a crear una carpeta que se llame pages en el siguiente directorio.



En la carpeta pages van a ir todos nuestros archivos jsp que ocupemos en nuestro proyecto.

Desarrollo

Ya tenemos spring configurado y la estructura del proyecto, ahora podemos proceder a crear nuestros archivos. Para este ejemplo vamos a crear una aplicación web que nos permita dar de alta un usuario.

Vamos a crear nuestro modelo, nos vamos al paquete `com.spring4.hibernate4.model` y creamos la clase usuario y pegamos el siguiente código.

```
package com.spring4.hibernate4.model;

import java.io.Serializable;

public class Usuario implements Serializable{

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private int id;
    private String nombre;
    private String apellidos;
    private int edad;
    public Usuario(){

    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellidos() {
        return apellidos;
    }
    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
    public static long getSerialversionuid() {
```

```

        return serialVersionUID;
    }

}

}

```

Ahora que tenemos nuestro modelo, podemos empezar con nuestro controlador.

Creemos la clase UsuariosController en el paquete com.spring4.hibernate4.controller y pegamos lo siguiente:

```

package com.spring4.hibernate4.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import com.spring4.hibernate4.model.Usuario;

@Controller
public class UsuariosController {
    @RequestMapping("/altas")
    public ModelAndView darAltas(){
        Usuario usuario = new Usuario();
        ModelAndView mv = new ModelAndView("altas");
        usuario.setId(1);
        usuario.setNombre("Rob");
        usuario.setApellidos("RR RR");
        usuario.setEdad(30);
        mv.addObject("usuario", usuario);
        return mv;
    }
}

```

Van a notar que eclipse nos marca varios errores ya que no cuenta con las librerías y si tratan de importarlas no las van a encontrar. Para esto vamos a descargar las librerías que necesitamos con ayuda de maven. en la terminal nos vamos hasta donde esta nuestro archivo pom.xml y tecelamos lo siguiente: `mvn dependency:copy-dependencies`

Esto nos descargará las librerías dentro de nuestro proyecto en la carpeta target - dependency. Debemos indicarle a eclipse donde están las librerías.

Con esto ya no debemos tener ningún error en nuestro controller. Vamos a proceder con la vista, primero modifiquemos la pagina index.jsp que se creo automáticamente y pongamos lo siguiente:

```
<html>
<body>
<a href="altas.html">click aqui</a>
</body>
</html>
```

Ahora dentro de la carpeta pages, creemos un jsp con el nombre altas.jsp y ponemos lo siguiente:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

    <p>Nombre: ${usuario.nombre}</p>

    <p>Apellidos: ${usuario.apellidos }</p>

    <p>Edad: ${usuario.edad}</p>

</body>

</html>
```

Y como resultado de vemos ver lo siguiente:

Nombre: Rob

Apellidos: RR RR

Edad: 30

Pero bueno, lo que nosotros queremos es poder dar de alta un usuario, entonces ocupamos un formulario que nos mande los datos desde el navegador al servidor. Vamos a modificar un poco nuestro controlador UsuariosController.

Antes de mostrarles el código del modelo, quiero explicar algunas cosas.

Podemos notar que arriba de nuestro método tenemos una anotación que dice `@RequestMapping("/altas")`. Esta anotación le indica a spring que nuestra pagina altas.jsp esta relacionada con nuestro método darAltas.

Para el método onSubmit vemos que hay una anotación similar solo con un parámetro de más `@RequestMapping(value="/altas", method = RequestMethod.POST)`. con esto estamos indicando que debe entrar a ese método desde la pagina altas.jsp pero solo cuando hacemos un submit por POST, si en nuestro formulario del html indicamos que por GET, nunca entrara a dicha función.

El return de nuestro método onSubmit es importante ponerle el nombre de la pagina jsp ya que de otro modo no sabra a donde regresar.

El código de nuestro controlador quedará de la siguiente manera:

```
package com.spring4.hibernate4.controller;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import com.spring4.hibernate4.model.Usuario;

@Controller
public class UsuariosController {
    @RequestMapping("/altas")
    public ModelAndView darAltas(){
        Usuario usuario = new Usuario();
        ModelAndView mv = new ModelAndView("altas");
        mv.addObject("usuario", usuario);
    }
}
```

```

        return mv;
    }

    @RequestMapping(value="/altas", method = RequestMethod.POST)
    public String onSubmit(@ModelAttribute("usuario") Usuario usuario,
        BindingResult result){

        System.out.println("submit sucess!");
        return "altas";
    }
}

```

Y también debemos modificar el jsp de altas para que quede del siguiente modo

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <form:form method="POST" modelAttribute="usuario" >
        <p>Nombre: <form:input path="nombre"/></p>
        <p>Apellidos: <form:input path="apellidos"/></p>
        <p>Edad: <form:input path="edad"/></p>
        <input type="submit" value="enviar"/>
    </form:form>
    <br/>
    <br/>
    <c:if test="${!empty usuario.nombre}">
        <h3>Usuario dado de alta: </h3>
        <p>nombre ${usuario.nombre }</p>
    </c:if>
</body>
</html>

```

Lo que hice en esta pagina fue importar la librería de spring para manejar formularios y la librería JSTL para las etiquetas como la de <c:if>

Como resultado de vemos ver lo siguiente:

Nombre:

Apellidos:

Edad:

Usuario dado de alta:

nombre Robert

Hasta este punto ya tenemos lo básico de spring donde ya sabemos cómo mostrar datos desde el servidor y como mandar datos desde el navegador al servidor.

Para terminar el proyecto nos falta poder guardar los usuarios dados de alta, hacer un listado y poder borrarlos. Para esto vamos a configurar Hibernate para poder guardar y acceder a los datos desde nuestra base de datos.

Configuración de Hibernate

Antes de empezar, hay que crear la tabla donde se guardará nuestro usuario.

```
CREATE TABLE `usuario` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `nombre` varchar(100) DEFAULT NULL,  
  `apellidos` varchar(100) DEFAULT NULL,  
  `edad` int(11) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Ahora vamos a configurar Hibernate. Primero necesitamos configurar el pom.xml para agregar las librerías que necesitamos.

Nuestro pom debe quedar así:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.spring4.hibernate4</groupId>
  <artifactId>spring4hibernate4</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>spring4hibernate4 Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <properties>
    <spring.version>4.0.1.RELEASE</spring.version>
  </properties>
  <repositories>
    <repository>
      <id>JBoss repository</id>

      <url>http://repository.jboss.org/nexus/content/groups/public/</url>
    </repository>
  </repositories>
  <dependencies>
    <!-- Spring 4 dependencies -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
```

```

        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>1.1.3</version>
    </dependency>
    <!-- end of Spring 4 dependencies -->
    <!-- Hibernate 4 dependencies -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>3.6.3.Final</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.28</version>
    </dependency>
    <dependency>
        <groupId>commons-collections</groupId>
        <artifactId>commons-collections</artifactId>
        <version>3.2.1</version>
    </dependency>

    <dependency>
        <groupId>cglib</groupId>
        <artifactId>cglib</artifactId>
        <version>2.2</version>
    </dependency>
    <dependency>
        <groupId>dom4j</groupId>
        <artifactId>dom4j</artifactId>
        <version>1.6.1</version>
    </dependency>
    <dependency>
        <groupId>javax.transaction</groupId>
        <artifactId>jta</artifactId>
        <version>1.1</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.5</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate.javax.persistence</groupId>
        <artifactId>hibernate-jpa-2.0-api</artifactId>
        <version>1.0.1.Final</version>
    </dependency>
    <dependency>
        <groupId>javassist</groupId>
        <artifactId>javassist</artifactId>
        <version>3.12.1.GA</version>
    </dependency>

```



```

        <!-- end of hibernate 4 dependencies -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>

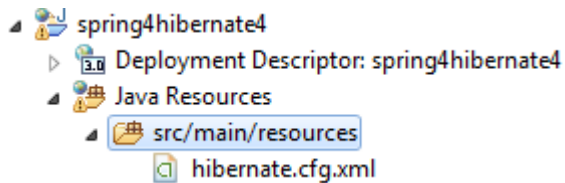
    </dependencies>
    <build>
        <finalName>spring4hibernate4</finalName>
    </build>
</project>

```

Descargamos las librerías con el comando `mvn dependency:copy-dependencies` y le indicamos a eclipse de las nuevas librerías.

Vamos a hacer el archivo que se encargará de hacer la conexión con la base de datos.

Creamos el archivo `hibernate.cfg.xml` en la ruta `src/main/resources`



Y pegamos lo siguiente:

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.bytecode.use_reflection_optimizer">false</property>
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.password">tu password aqui
</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/el
nombre de tu schema aqui</property>
        <property name="hibernate.connection.username">tu usuario
aqui</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="show_sql">true</property>
        <mapping
resource="com/spring4/hibernate4/model/Usuario.hbm.xml"></mapping>
    </session-factory>
</hibernate-configuration>

```

```

    </session-factory>
</hibernate-configuration>

```

Dentro del mismo source folder vamos a crear el paquete `com.spring4.hibernate4.model` y la clase `Usuario.hbm.xml` y pegamos lo siguiente:

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="com.spring4.hibernate4.model.Usuario" table="usuario"
catalog="hibernate">
        <id name="id" type="java.lang.Integer">
            <column name="id" />
            <generator class="identity" />
        </id>
        <property name="nombre" type="string">
            <column name="nombre" length="100" />
        </property>
        <property name="apellidos" type="string">
            <column name="apellidos" length="100" />
        </property>
        <property name="edad" type="java.lang.Integer">
            <column name="edad" length="100" />
        </property>
    </class>
</hibernate-mapping>

```

Hasta aquí ya tenemos configurada la conexión y el mapeo de nuestro modelo con la tabla de la base de datos. Lo que sigue es crear un paquete dentro de `src/main/java` con el siguiente nombre `com.spring4.hibernate4.persistence` y creamos la clase `HibernateUtil` que será quien se encargue de abrir nuestra conexión y cerrarla. Pegamos lo siguiente:

```
package com.mkyong.persistence;
```

```
import org.hibernate.SessionFactory;
```

```
import org.hibernate.cfg.Configuration;
```

```
public class HibernateUtil {
```

```
    private static final SessionFactory sessionFactory = buildSessionFactory();
```

```

private static SessionFactory buildSessionFactory() {
    try {
        // Create the SessionFactory from hibernate.cfg.xml
        return new Configuration().configure().buildSessionFactory();
    }
    catch (Throwable ex) {
        // Make sure you log the exception, as it might be swallowed
        System.err.println("Initial SessionFactory creation failed." + ex);
        throw new ExceptionInInitializerError(ex);
    }
}

public static SessionFactory getSessionFactory() {
    return sessionFactory;
}

public static void shutdown() {
    // Close caches and connection pools
    getSessionFactory().close();
}
}

```

Guardar usuario en la base de datos

Ahora ya podemos guardar nuestro usuario en nuestra base de datos, para esto hay que dejar el controller de la siguiente manera:

```

package com.spring4.hibernate4.controller;

import org.hibernate.Session;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import com.spring4.hibernate4.model.Usuario;
import com.spring4.hibernate4.persistence.HibernateUtil;

@Controller

public class UsuariosController {

    @RequestMapping("/altas")

    public ModelAndView darAltas(){

        Usuario usuario = new Usuario();

        ModelAndView mv = new ModelAndView("altas");

        mv.addObject("usuario", usuario);

        return mv;

    }

    @RequestMapping(value="/altas", method = RequestMethod.POST)

    public String onSubmit(@ModelAttribute("usuario") Usuario usuario,
BindingResult result){

```

```

        Session session = HibernateUtil.getSessionFactory().openSession();

        session.beginTransaction();

        session.save(usuario);

        session.getTransaction().commit();

        session.close();

        HibernateUtil.shutdown();

        System.out.println("submit success!");

        return "altas";

    }
}

```

Antes de continuar haciendo consultas con hibernate, vamos a organizar y definir qué es lo que queremos hacer. Creamos un paquete en src/main/java con el nombre de com.spring4.hibernate4.dao y creamos una interfaz con el nombre UsuariosDao y pegamos lo siguiente:

```

package com.spring4.hibernate4.dao;

import java.util.List;

import com.spring4.hibernate4.model.Usuario;

public interface UsuariosDao {

    public void guardar(Usuario usuario);
}

```

```

        public List listado();

        public void borrar(int id);
    }

```

Despues vamos a crear nuestra implementación en el paquete com.spring4.framework4.hibernate creamos la clase UsuariosHibernate y pegamos lo siguiente:

```

package com.spring4.hibernate4.hibernate;

import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.criterion.Restrictions;

import com.spring4.hibernate4.persistence.HibernateUtil;
import com.spring4.hibernate4.dao.UsuariosDao;
import com.spring4.hibernate4.model.Usuario;

public class UsuariosHibernate implements UsuariosDao {

    Session session = HibernateUtil.getSessionFactory().openSession();

    @Override
    public void guardar(Usuario usuario){

        // TODO Auto-generated method stub
    }

```

```

        try{

            session.beginTransaction();

            session.save(usuario);

            session.getTransaction().commit();

        }finally{

            session.close();

        }

    }

    @Override

    public List listado() {

        // TODO Auto-generated method stub

        Criteria cr = session.createCriteria(Usuario.class);

        List usuarios = cr.list();

        return usuarios;

    }

    @Override

    public void borrar(int id) {

        // TODO Auto-generated method stub

        try{

            Criteria cr =

session.createCriteria(Usuario.class).add(Restrictions.eq("id", id));

            Usuario usuario =(Usuario) cr.list().get(0);

            System.out.println("Usuario a ser borrado: " + usuario.getNombre());

            session.beginTransaction();

```

```

        session.delete(usuario);

        session.getTransaction().commit();

    }finally{

        session.close();

    }

}

}

}

```

Hasta este punto ya tenemos definidos los métodos para guardar, borrar y traer datos desde la base de datos. Nuestro UsuariosController quedará así:

```

package com.spring4.hibernate4.controller;

import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

```



```
import com.spring4.hibernate4.dao.UsuariosDao;

import com.spring4.hibernate4.hibernate.UsuariosHibernate;

import com.spring4.hibernate4.model.Usuario;
```

@Controller

```
public class UsuariosController {
```

```
    @RequestMapping("/altas")
```

```
    public ModelAndView darAltas(){
```

```
        Usuario usuario = new Usuario();
```

```
        ModelAndView mv = new ModelAndView("altas");
```

```
        mv.addObject("usuario", usuario);
```

```
        return mv;
```

```
    }
```

```
    @RequestMapping(value="/altas", method = RequestMethod.POST)
```

```
    public String onSubmit(@ModelAttribute("usuario") Usuario usuario,
        BindingResult result){
```

```
        UsuariosDao trans = new UsuariosHibernate();
```

```
        trans.guardar(usuario);
```

```
        System.out.println("submit sucess!");
```

```
        return "altas";
```

```
    }
```

```

@RequestMapping("/listado")

    public ModelAndView hacerListado(){

        UsuariosDao trans = new UsuariosHibernate();

        List ls = trans.listado();


        ModelAndView mv = new ModelAndView("listado");

        mv.addObject("usuarios", ls);

        return mv;

    }

```

```

@RequestMapping("/eliminar")

    public ModelAndView listadoEliminar(){

        UsuariosDao trans = new UsuariosHibernate();

        List ls = trans.listado();


        ModelAndView mv = new ModelAndView("eliminar");

        mv.addObject("lista", ls);

        return mv;

    }

```

```

@RequestMapping(value="/eliminarUsuario", method = RequestMethod.GET)

    public String eliminarFila(ModelMap m, @RequestParam(value="id") int id){

```

```

        UsuariosDao trans = new UsuariosHibernate();

        trans.borrar(id);

        System.out.println("delete sucess!");

        return "redirect:/eliminar.html";

    }

}

```

Ahora solo nos queda hacer las vistas

vistas

Altas.jsp

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <form:form method="POST" modelAttribute="usuario" >
        <p>Nombre: <form:input path="nombre"/></p>
        <p>Apellidos: <form:input path="apellidos"/></p>
        <p>Edad: <form:input path="edad"/></p>
        <input type="submit" value="enviar"/>
    </form:form>
    <br/>
    <br/>
    <c:if test="${!empty usuario.nombre}">
        <h3>Usuario dado de alta: </h3>
        <p>nombre ${usuario.nombre }</p>
    </c:if>
</body>
</html>

```

eliminar.jsp

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="c"

```

```

        uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Eliminar</title>
</head>
<body>

        <table>
            <thead>
                <tr>
                    <td>Nombre</td>
                    <td>Apellidos</td>
                    <td>Edad</td>
                </tr>
            </thead>
            <tbody>
                <c:forEach items="${lista}" var="ls"
varStatus="status">
                    <tr>
                        <td>${ls.nombre}</td>
                        <td>${ls.apellidos}</td>
                        <td>${ls.edad }</td>
                        <td><a href="eliminarUsuario.html?id=${ls.id}">Eliminar</a>
                    </tr>
                </c:forEach>
            </tbody>
        </table>

</body>
</html>

```

Listado.jsp

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Listado</title>
</head>
<body>
    <table>
        <thead>
            <tr>
                <td>Nombre</td>
                <td>Apellidos</td>
                <td>Edad</td>
            </tr>
        </thead>
    </table>

```

```
        </tr>
    </thead>
    <tbody>
        <c:forEach items="${usuarios}" var="usuario">
            <tr>
                <td>${usuario.nombre}</td>
                <td>${usuario.apellidos}</td>
                <td>${usuario.edad}</td>
            </tr>
        </c:forEach>
    </tbody>
</table>

</body>
</html>
```