Exemple CRUD avec le Framework Spring / JSF / Hibernate sur HSQL.

Auteur : Beuve Johnny et Knuchel Christophe.

Table des matières

Introduction

Dans cet article, on montrera, à l'aide d'un exemple, comment **SPRING** crée et met en relation les objets des différentes couches du framework à notre place.

L'exemple consiste en un simple **CRUD** (Create-Read-Update-Delete) d'un document **TODO**.

En plus, on verra une démonstrastion d'intégration de **JSF** et d'**Hibernate**.

Framework Spring

SPRING est un conteneur léger, simple, non intrusif, extensible qui permet un réel découplage des couches grâce à son approche basée sur les notions d'inversion de contrôle et de programmation par aspect.

Pour en savoir plus, il est vivement recommandé de lire le document écrit par Serge Tahé expliquant l'inversion de contrôle avec **SPRING** (http://tahe.developpez.com/java/springioc/).

Pour la description de tous les services offerts par le framework SPRING, veuillez consulter le site de référence :http://www.springframework.org/

Pré-requis

HSQL 1.7.3

HSQL est une base de données Java.

http://hsqldb.org/

Nous démarrons HSQL en mode serveur.

Exemple de lancement : `runServer.bat`

```
cd ..\data
@java -classpath ../lib/hsqldb.jar org.hsqldb.Server -database.0 localdb -dbname.0 jodb
```

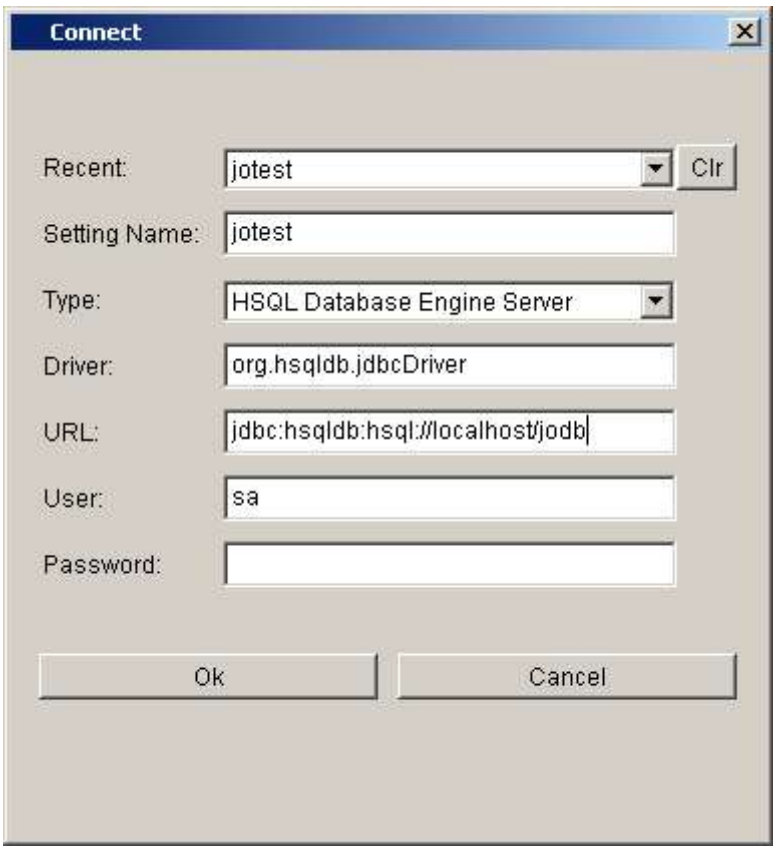
«jodb» étant le nom de la base de données.

Pour faire des opérations sur cette base de données, nous utilisons le logiciel «database manager», fourni avec HSQL

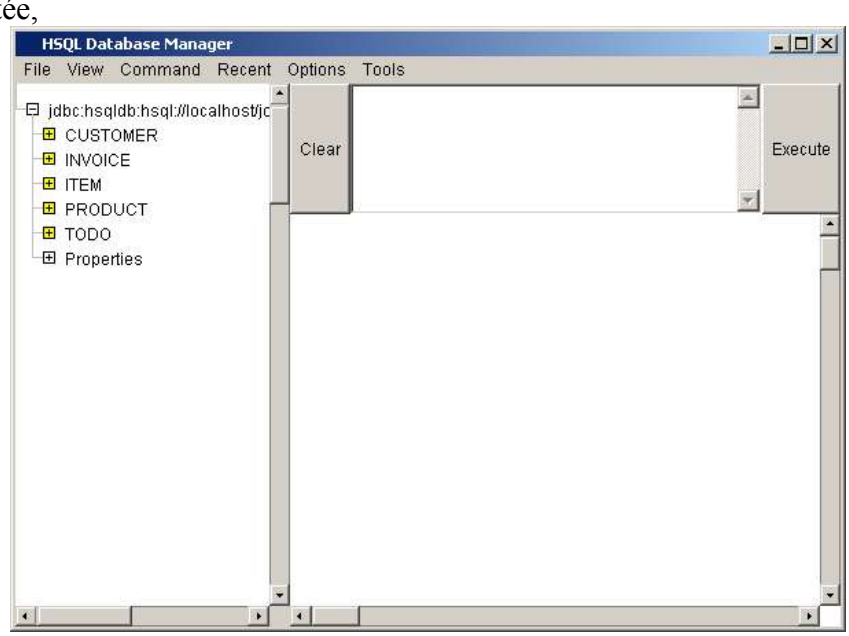
Script de lancement du logiciel «database manager» : `runManager.bat`

```
cd ..\data
@java -classpath ..\lib\hsqldb.jar org.hsqldb.util.DatabaseManager %1 %2 %3 %4 %5 %6 %7 %8 %9
```

Cela donne :



Une fois connectée,



Pour arrêter cett base, il suffit d'exécuter la commande **SHUTDOWN**.

Script de création de la table **TODO** :

```
CREATE SEQUENCE HIBERNATE_SEQUENCE AS INTEGER START WITH 1

CREATE TABLE TODO(ID INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH 0)
NOT NULL PRIMARY KEY,TITLE VARCHAR(254) NOT NULL,BODY VARCHAR(1024))
```

Eclipse 3.0.2

http://www.eclipse.org

ExadelStudio

ExadelStudio-2[1].5.2. Ce plugin gratuit permet de faire du **JSF** avec **Hibernate** et **Spring**.

http://www.exadel.com/

Hibernate 3

http://www.hibernate.org/

JSF

Version fournie avec le plugin **ExadelStudio**.

Spring

Version 1.2.2.

http://www.springframework.org

Tomcat

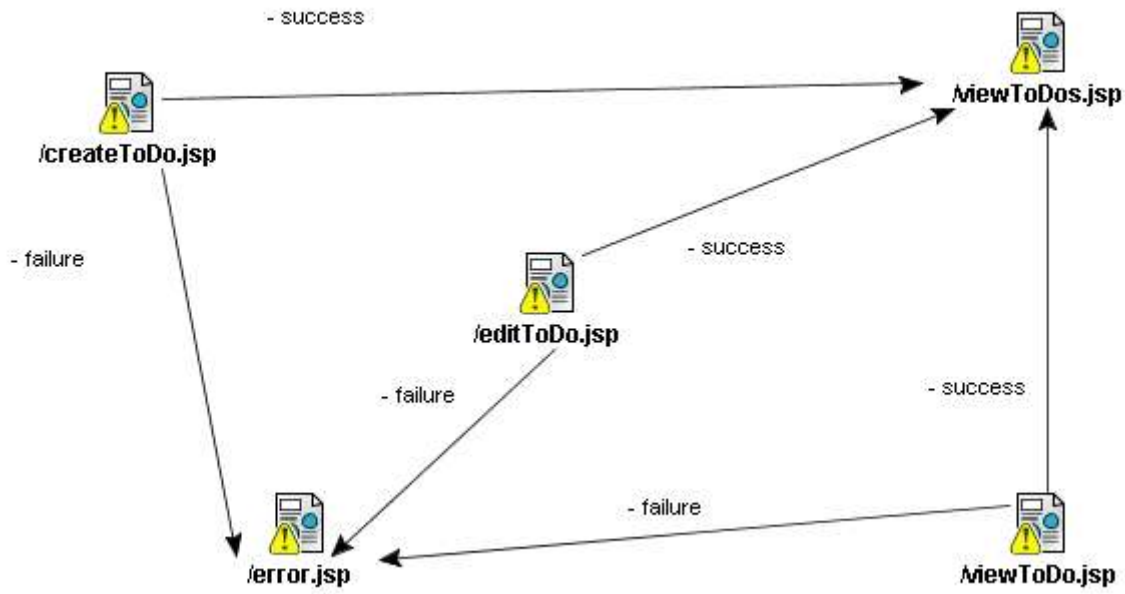
Version 5XX.

http://jakarta.apache.org/tomcat/index.html

Exemple

Ici, vous avez une page **index.jsp** qui contient un forward sur la page **jsp** qui liste tous les **todo** (**viewToDos.jsp**).

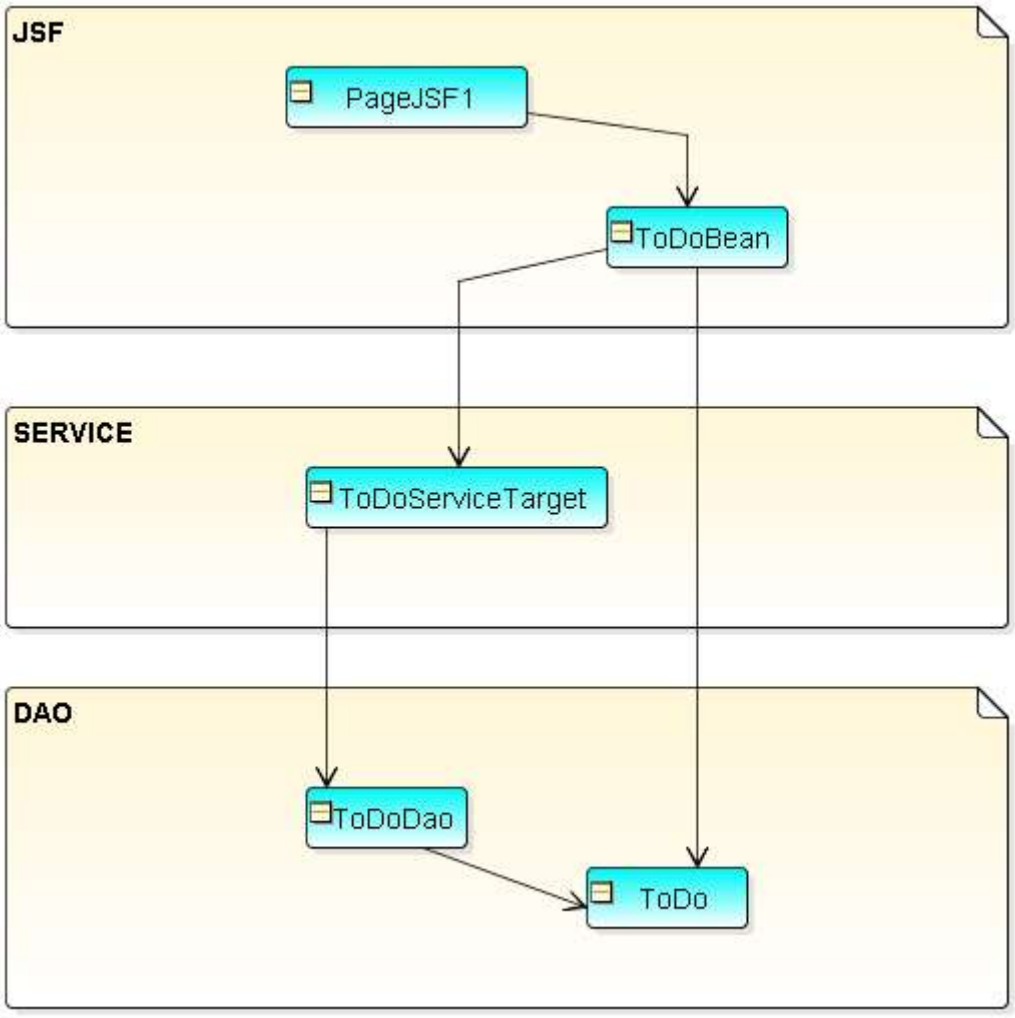
A partir de cette page il est possible de créer, supprimer, consulter et éditer un todo.



Fonctionnement général

L'architecture générale donne ceci :

Après une action, une page **JSF**, initialise le **beanVisuel**, remplit l'objet **value ToDo**, déclenche dans le **beanVisuel** une action. Cette action fait appel à un service, qui lui même, déclenche la persistance grâce à un **DAO**.



Documents de configuration

web.xml :

```
<?xml version="1.0"?>

<!DOCTYPE web-app PUBLIC
  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    <display-name>Jo test</display-name>
    <description>
        Jo Test String JSF Hibernate
    </description>

       <!-- JavaServer Faces -->
    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>server</param-value>
    </context-param>

    <context-param>
        <param-name>com.sun.faces.validateXml</param-name>
        <param-value>true</param-value>
    </context-param>

    <listener>
        <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
    </listener>

    <!-- Faces Servlet -->
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet>
         <servlet-name>SpringContextServlet</servlet-name>
         <servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>
         <load-on-startup>1</load-on-startup>
    </servlet>

    <!-- Faces Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>
</web-app>
```

Web.xml avec une déclaration **JSF** classique. Le contexte **Spring** est lancé comme suit :

```
    <servlet>
         <servlet-name>SpringContextServlet</servlet-name>
         <servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>
         <load-on-startup>1</load-on-startup>
    </servlet>
```

Cette servlet va, à son chargement, instancier en mémoire les objets des différentes couches, grâce à la **BeanFactory** qui lit le fichier de configuration de **Spring**.

Faces-config.xml :

Ce fichier contient principalement la navigation et les beans visuels manipulés par les pages **JSF**.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
                              "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
	<managed-bean>
		<description>
			Visual Bean with reference on Business Bean.
		</description>
		<managed-bean-name>todoBean</managed-bean-name>
		<managed-bean-class>
			jotodo.gui.bean.ToDoBean
		</managed-bean-class>
		<managed-bean-scope>request</managed-bean-scope>
		<managed-property>
			<property-name>toDoService</property-name>
			<value>#{toDoService}</value>
		</managed-property>
		<managed-property>
			<property-name>toDoId</property-name>
			<value>#{param.toDoId}</value>
		</managed-property>
	</managed-bean>
	<navigation-rule>
		<from-view-id>/createToDo.jsp</from-view-id>
		<navigation-case>
			<from-outcome>success</from-outcome>
			<to-view-id>/viewToDos.jsp</to-view-id>
		</navigation-case>
		<navigation-case>
			<from-outcome>failure</from-outcome>
			<to-view-id>/error.jsp</to-view-id>
		</navigation-case>
	</navigation-rule>
	<navigation-rule>
		<from-view-id>/editToDo.jsp</from-view-id>
		<navigation-case>
			<from-outcome>success</from-outcome>
			<to-view-id>/viewToDos.jsp</to-view-id>
		</navigation-case>
		<navigation-case>
			<from-outcome>failure</from-outcome>
			<to-view-id>/error.jsp</to-view-id>
		</navigation-case>
	</navigation-rule>
		<navigation-rule>
		<from-view-id>/viewToDo.jsp</from-view-id>
		<navigation-case>
			<from-outcome>success</from-outcome>
			<to-view-id>/viewToDos.jsp</to-view-id>
		</navigation-case>
		<navigation-case>
			<from-outcome>failure</from-outcome>
			<to-view-id>/error.jsp</to-view-id>
		</navigation-case>
	</navigation-rule>
	<!-- This JSF variable resolver lets you reference JSF managed
		beans from a Spring context, or a Spring bean from a managed bean -->
	<application>
		<variable-resolver>
			org.springframework.web.jsf.DelegatingVariableResolver
		</variable-resolver>
		<locale-config />
	</application>
	<lifecycle />
	<factory />
</faces-config>
```

Il existe trois règles de navigation concernant les pages:

- createToDo.jsp,
- editToDo.jsp,
- viewToDo.jsp.

Un bean visuel «**ToDoBean**» avec deux propriétés :

1. toDoService : Une référence au bean toDoService.
2. ToDoId : un paramètre toDoId.

Si vous cherchez le bean toDoService dans le fichier de configuration de JSF, vous ne le trouverez pas et c'est normal.

Effectivement ce bean est déclaré dans le fichier de configuration de Spring.

Mais comment demander à JSF d'aller chercher le toDoService chez Spring?

Tout simplement avec ces quelques lignes de configuration :

```
...
	<application>
		<variable-resolver>
			org.springframework.web.jsf.DelegatingVariableResolver
		</variable-resolver>
		<locale-config />
	</application>
...
```

Si le bean référencé n'est pas dans le fichier de configuration de JSF, alors JSF va essayer de trouver la déclaration en utilisant le «variable-resolver».

applicationContext.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-
beans.dtd">

<beans>
	<!-- ========================= Start of PERSISTENCE DEFINITIONS =========================
-->

	<!-- DataSource Definition -->
	<bean id="dataSource"
		class="org.apache.commons.dbcp.BasicDataSource"
		destroy-method="close">
		<property name="driverClassName">
			<value>org.hsqldb.jdbcDriver</value>
		</property>
		<property name="url">
			<value>jdbc:hsqldb:hsql://127.0.0.1/jodb</value>
		</property>
		<property name="username">
			<value>sa</value>
		</property>
		<property name="password">
			<value></value>
		</property>
	</bean>

	<!-- Hibernate SessionFactory Definition -->
	<bean id="sessionFactory"
		class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
		<property name="mappingResources">
			<list>
				<value>jotodo/biz/bo/todo.hbm.xml</value>
			</list>
		</property>

		<property name="hibernateProperties">
			<props>
				<prop key="hibernate.dialect">
					org.hibernate.dialect.HSQLDialect
				</prop>
				<prop key="hibernate.show_sql">true</prop>
				<prop key="hibernate.cglib.use_reflection_optimizer">
					true
				</prop>
				<prop key="hibernate.cache.provider_class">
					org.hibernate.cache.HashtableCacheProvider
				</prop>
			</props>
		</property>

		<property name="dataSource">
			<ref bean="dataSource" />
		</property>
	</bean>

	<!-- Spring Data Access Exception Translator Defintion -->
	<bean id="jdbcExceptionTranslator"
		class="org.springframework.jdbc.support.SQLErrorCodeSQLExceptionTranslator">
		<property name="dataSource">
			<ref bean="dataSource" />
		</property>
	</bean>

	<!-- Hibernate Template Defintion -->
	<bean id="hibernateTemplate"
		class="org.springframework.orm.hibernate3.HibernateTemplate">
		<property name="sessionFactory">
			<ref bean="sessionFactory" />
		</property>
		<property name="jdbcExceptionTranslator">
			<ref bean="jdbcExceptionTranslator" />
		</property>
	</bean>

	<!-- Hibernate Transaction Manager Definition -->
	<bean id="transactionManager"
		class="org.springframework.orm.hibernate3.HibernateTransactionManager">
		<property name="sessionFactory">
			<ref local="sessionFactory" />
		</property>
	</bean>
```

```
	<!-- ========================= Start of DAO DEFINITIONS ======================== -->

	<!-- TODO DAO Definition: Hibernate implementation -->
	<bean id="todoDao" class="jotodo.per.dao.ToDoDaoImpl">
		<property name="hibernateTemplate">
			<ref bean="hibernateTemplate" />
		</property>
	</bean>

	<!-- ========================= Start of SERVICE DEFINITIONS ======================== -->

	<!-- TODO Service Definition -->
	<bean id="toDoServiceTarget"
		class="jotodo.biz.bs.ToDoServiceImpl">
		<property name="toDoDao">
			<ref local="todoDao" />
		</property>
	</bean>

	<!-- Transactional proxy for the TODO Service -->
	<bean id="toDoService"
		class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
		<property name="transactionManager">
			<ref local="transactionManager" />
		</property>
		<property name="target">
			<ref local="toDoServiceTarget" />
		</property>
		<property name="transactionAttributes">
			<props>
				<prop key="get*">PROPAGATION_REQUIRED,readOnly</prop>
				<prop key="save*">PROPAGATION_REQUIRED</prop>
				<prop key="update*">PROPAGATION_REQUIRED</prop>
				<prop key="delete*">PROPAGATION_REQUIRED</prop>
			</props>
		</property>
	</bean>
</beans>
```

Ce fichier semble un peu complexe, mais après l'avoir compris, il devient facile à manipuler.

Trois parties:

1. Configurer l'ensemble de la persistance (Datasource, Hibernate sessionFactory, Data Access Exception, Hibernate Template, Hibernate Transaction.).
2. Décrire tous les DAO de l'application (TODO DAO).
3. Décrire les services de l'application avec un accès aux services via un proxy qui définit la transaction.

Notez que le proxy a une référence sur le transaction Manager.

```
		<property name="transactionManager">
			<ref local="transactionManager" />
		</property>
```

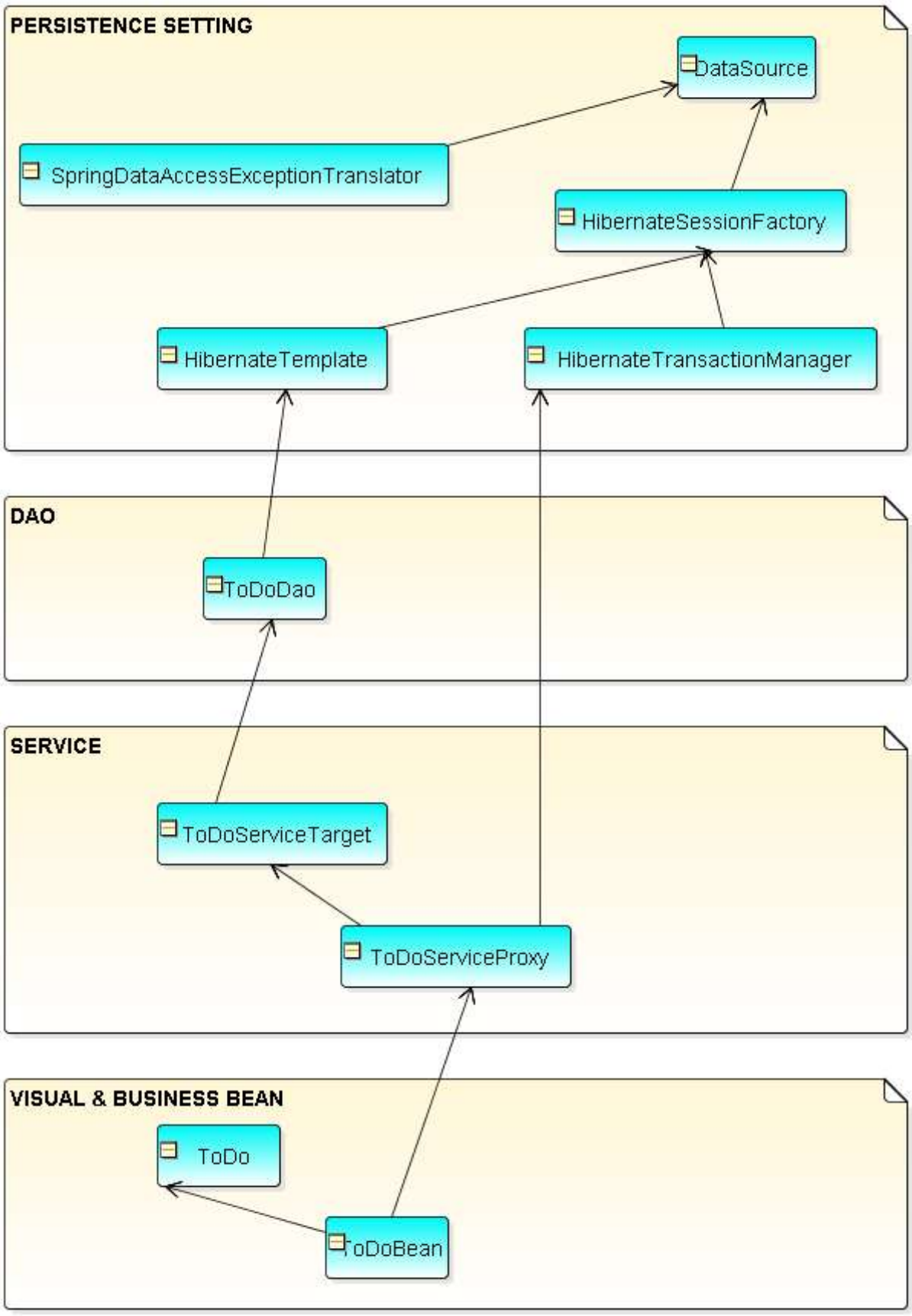
Notez que le service a une référence sur le DAO.

```
		<property name="transactionManager">
			<ref local="transactionManager" />
		</property>
```

Notez que le DAO a une référence sur le Template Hibernate.

```
		<property name="transactionManager">
			<ref local="transactionManager" />
		</property>
```

Vue d'ensemble :



On verra plus loin le détail de chaque couche.

Log4j:

log4j.properties

```
### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}\:%M\:%L -
%m%n

### set log levels - for more verbose logging change 'info' to 'debug' ###
log4j.rootLogger=info, stdout
```

Classe utilisée pour instancier les logs

```
package util;

import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

/**
 * Class Which give you log object.
 */
public class Log {

	public static final String DELIMITER = " £ ";
	/**
	 * @param o Object to log.
	 * @param level
	 * @return Logger
	 */
	public static Logger getLog(Object o, Level level) {
		//PropertyConfigurator.configure("log4j.properties");
		Logger log = Logger.getLogger(o.getClass());
		log.setLevel(level);
		return log;
	}

	/**
	 * Set WARN level by default.
	 * @param o
	 * @return Logger
	 */
	public static Logger getLog(Object o) {
		return getLog(o, Level.DEBUG);
	}
}
```

Bean visuel et bean métier à persister

Todo.java :

```
package jotodo.biz.bo;

import org.apache.log4j.Logger;
import util.Log;

/**
 * @author Jumper
 */
public class ToDo {
    private Logger log = Log.getLog(this);
    private Integer id;
    private String title;
    private String body;

    /**
     * @return id.
     */
    public Integer getId() {
        return id;
    }

    /**
     * @param id
     *        The id to set.
     */
    public void setId(Integer id) {
        this.id = id;
        log.debug("#DDD###### todo id : " + id );
    }

    /**
     * @return body.
     */
    public String getBody() {
        return body;
    }

    /**
     * @param body
     *        The body to set.
     */
    public void setBody(String body) {
        this.body = body;
    }

    /**
     * @return title.
     */
    public String getTitle() {
        return title;
    }

    /**
     * @param title
     *        The title to set.
     */
    public void setTitle(String title) {
        this.title = title;
    }

    /**
     * constructor
     */
    public ToDo() {
    }

    public String toString() {
        return "Title : " + title + "\nBody : " + body + "\nid : " + id;
    }
}
```

Un simple Object Value.

ToDoBean.java :

```
package jotodo.gui.bean;

import java.util.Collection;

import org.apache.log4j.Logger;
import util.Log;
import jotodo.biz.bo.ToDo;
import jotodo.biz.bs.ToDoServiceAble;
import jotodo.biz.exception.JoTestException;

/**
 * @author Jumper
 */
public class ToDoBean {
    private Logger log = Log.getLog(this);
    private String toDoId;
    private Collection toDos = null;
    private ToDo toDo;
    private ToDoServiceAble toDoService;

    /**
     * Constructor
     */
    public ToDoBean() {
        super();
        log.debug("#DDD############ Constructor");
        toDo = new ToDo();
    }

    /**
     * @return Return the toDo.
     */
    public ToDo getToDo() {
        return this.toDo;
    }

    public Collection getToDoS() {
        if (toDos == null) {
            try {
                log.debug("#DDD############ toDos null --> service.getToDos");
                toDos = toDoService.getToDoS();
            } catch (Exception e) {
                log.error("#DDD############ Error when searching the todo list");
            }
        }
        return toDos;
    }

    public String createToDoAction() {
        log.debug("#DDD############ createToDoAction()");
        try {
            this.toDoService.saveToDo(this.toDo);
            log.debug("#DDD############ createToDoAction->success");
            return "success";
        } catch (JoTestException e) {
            e.printStackTrace();
            return "failure";
        }
    }

    public String updateToDoAction() {
        log.debug("#DDD############ updateToDoAction()");
        try {
            this.toDoService.updateToDo(this.toDo);
            log.debug("#DDD############ updateToDoAction->success");
            return "success";
        } catch (JoTestException e) {
            e.printStackTrace();
            return "failure";
        }
    }
```

```
   public String deleteToDoAction() {
        log.debug("#DDD############ deleteToDoAction()");
        try {
            this.toDoService.deleteToDo(this.toDo);
            log.debug("#DDD############ deleteToDoAction->success");
            return "success";
        } catch (JoTestException e) {
            e.printStackTrace();
            return "failure";
        }
    }

    /**
     * @param toDoService
     *        The toDoService to set.
     */
    public void setToDoService(ToDoServiceAble toDoService) {
        log
                .debug("#DDD############ setToDoService(ToDoServiceAble toDoService)");
        this.toDoService = toDoService;
    }

    /**
     * @return Returns the toDoId.
     */
    public String getToDoId() {
        log.debug("#DDD############ getToDoId");
        return toDoId;
    }

    /**
     * @param toDoId
     *        The toDoId to set.
     */
    public void setToDoId(String toDoId) {
        this.toDoId = toDoId;
        log.debug("#DDD############ setToDoId(-" + toDoId + "-)");
        if (toDoId != null && !toDoId.equals("")) {
            Integer id = new Integer(toDoId);
            try {
                if (toDoService != null) {
                    toDo = toDoService.getToDo(id);
                }
            } catch (JoTestException e) {
                log.error("setToDoId, error :" + e);
            }
        }
    }
}
```

Cette classe correspond au bean visuel déclaré dans JSF.

Elle contient :

- Un objet Todo.qui correspond à un object value contenant les informations d'un Todo.
- Une référence sur un singleton ToDoService qui correspond à la classe métier qui contient les règles applicatives.
- Un objet Log permettant de loguer avec log4j.
- Une série d'actions qui seront appelées directement par JSF.
- Un toDoID qui sera rempli automatiquement par JSF (`#{param.toDoId}`).

Rappelons que cet objet est instancié lors de chaque requête HTTP puisqu' il est de type requête. Lors de son instanciation, la mécanique JSF – Spring :

- lie automatiquement le bean visuel au singleton ToDoService,
- cherche à remplir le toDoID avec un paramètre toDoID.

Persistance avec Hibernate: Le DAO

Le but du DAO est d'assurer la persistance d'un ou plusieurs objets.

Losque l'application est lancée, la beanFactory de Spring instancie les DAOs automatiquement sous forme de singleton.

Ici, nous avons utilisé Hibernate pour persister Todo.class. Il faut donc un fichier de configuration qui décrit le mapping entre la classe et la table.

Le fichier de configuration : todo.hbm.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
                                   "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping package="jotodo.biz.bo">
 <class name="ToDo" table="TODO" lazy="false">
  <id name="id" column="id" type="integer">
   <generator class="native"/>
  </id>
  <property name="title" column="TITLE"/>
  <property name="body" column="BODY"/>
 </class>
</hibernate-mapping>
```

Attention la propriété **`lazy="false"`** semble absolument nécessaire.

L'interface : ToDoDaoAble.java

```
package jotodo.per.dao;

import java.util.Collection;
import jotodo.biz.bo.ToDo;

/**
 * @author Jumper
 */
public interface ToDoDaoAble {
	public ToDo getToDo(Integer id);
	public void saveToDo(ToDo toDo);
	public void updateToDo(ToDo toDo);
	public Collection getToDoS();
	public void deleteToDo(ToDo todo);
}
```

L'implémentation : DaoToDoImpl :

```
package jotodo.per.dao;

import java.util.Collection;
import org.apache.log4j.Logger;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import util.Log;
import jotodo.biz.bo.ToDo;

/**
 * @author Jumper
 */
public class ToDoDaoImpl extends HibernateDaoSupport implements ToDoDaoAble {

    private Logger log = Log.getLog(this);

        /**
         * Construtor
         */
        public ToDoDaoImpl() {
                super();
                log.debug("#DDD############ Constructor DAO TODO");
        }

        /*
         * @see jotest.per.dao.ToDoDaoAble#getToDo(java.lang.Long)
         */
        public ToDo getToDo(Integer id) {
                log.debug("#DDD######### getTodo ID in todoDAO ");
                return (ToDo) this.getHibernateTemplate().load(ToDo.class, id);
        }

        /*
         * @see jotest.per.dao.ToDoDaoAble#saveToDo(jotest.biz.bo.ToDo)
         */
        public void saveToDo(ToDo toDo) {
                log.debug("#DDD######### Savetodo in todoDAO : "
                                + toDo.toString());
                this.getHibernateTemplate().save(toDo);
        }

        /*
         * @see jotest.per.dao.ToDoDaoAble#updateToDo(jotest.biz.bo.ToDo)
         */
        public void updateToDo(ToDo toDo) {
                log.debug("#DDD######### Updatetodo in todoDAO : "
                                + toDo.toString());
                this.getHibernateTemplate().update(toDo);
        }

        /*
         * @see jotest.per.dao.ToDoDaoAble#getToDoS()
         */
        public Collection getToDoS() {
                log.debug("#DDD######### getTodoS ID in todoDAO ");
                return (Collection) this.getHibernateTemplate().loadAll(ToDo.class);
        }

        /*
         * @see jotodo.per.dao.ToDoDaoAble#deleteToDo(jotodo.biz.bo.ToDo)
         */
        public void deleteToDo(ToDo toDo) {
                log.debug("#DDD######### deleteTodo in todoDAO : " + toDo.toString());
                this.getHibernateTemplate().delete(toDo);
        }
}
```

La persistance avec Hibernate est très simple. Il suffit d'une seule page de code pour persister un objet.

Service: TodoService

Le but de la couche métier est d'appliquer les règles métiers et d'appeler les daos pour persister les objets selon les besoins.

Spring instancie automatiquement sous forme de singleton les beans «service» déclarés dans le document de configuration.

Spring référence automatiquement les daos associés au service.

Interface : TodoServiceAble.java

```
package jotodo.biz.bs;

import java.util.Collection;
import jotodo.biz.bo.ToDo;
import jotodo.biz.exception.JoTestException;

/**
 * @author Jumber
 */
public interface ToDoServiceAble {
       public ToDo getToDo(Integer id) throws JoTestException;
       public void saveToDo(ToDo todo) throws JoTestException;
       public void updateToDo(ToDo todo) throws JoTestException;
       public Collection getToDoS()throws JoTestException;
       public void deleteToDo(ToDo todo)throws JoTestException;
}
```

L'implémentation : ToDoServiceImpl.java

```
package jotodo.biz.bs;

import java.util.Collection;
import org.apache.log4j.Logger;
import util.Log;
import jotodo.biz.bo.ToDo;
import jotodo.biz.exception.JoTestException;
import jotodo.per.dao.ToDoDaoAble;

/**
 * @author Jumper
 */
public class ToDoServiceImpl implements ToDoServiceAble {
    private Logger log = Log.getLog(this);
    private ToDoDaoAble toDoDao;

    /**
     * @param todoDao
     *        The todoDao to set.
     */
    public void setToDoDao(ToDoDaoAble toDoDao) {
        log.debug("#DDD############ setToDoDao in Service TODO");
        this.toDoDao = toDoDao;
    }

    /**
     * Constructor.
     */
    public ToDoServiceImpl() {
    }

    /*
     * (non-Javadoc)
     * @see jotest.biz.bs.ToDoServiceAble#getToDo(java.lang.Long)
     */
    public ToDo getToDo(Integer id) throws JoTestException {
        log.debug("#DDD############ getToDo(" + id + ") in Service TODO");
        return toDoDao.getToDo(id);
    }
```

```
    /*
     * (non-Javadoc)
     * @see jotest.biz.bs.ToDoServiceAble#saveToDo(jotest.biz.bo.ToDo)
     */
    public void saveToDo(ToDo todo) throws JoTestException {
        log.debug("#DDD############ saveToDo(toDo) in Service TODO");
        toDoDao.saveToDo(todo);
    }
    /*
     * (non-Javadoc)
     * @see jotest.biz.bs.ToDoServiceAble#updateToDo(jotest.biz.bo.ToDo)
     */
    public void updateToDo(ToDo todo) throws JoTestException {
        log.debug("#DDD############ updateToDo(toDo) in Service TODO");
        toDoDao.updateToDo(todo);
    }

    /*
     * (non-Javadoc)
     * @see jotest.biz.bs.ToDoServiceAble#getToDoS()
     */
    public Collection getToDoS() throws JoTestException {
        log.debug("#DDD############ getToDoS() in Service TODO");
        return this.toDoDao.getToDoS();
    }

    /*
     * (non-Javadoc)
     * @see jotodo.biz.bs.ToDoServiceAble#deleteToDo(jotodo.biz.bo.ToDo)
     */
    public void deleteToDo(ToDo todo) {
        log.debug("#DDD############ deleteToDo(toDo) in Service TODO");
        toDoDao.deleteToDo(todo);
    }
}
```

Le service est à peine plus grand que le DAO, ce qui est normal, puisqu'il n'y a pas de réelle logique applicative dans cet exemple CRUD.

Au chargement de l'application, nous avons les instanciations suivantes qui se font automatiquement avec SPRING : log

```
21:07:36,313  INFO XmlBeanDefinitionReader:loadBeanDefinitions:150 - Loading XML bean definitions
from ServletContext resource [/WEB-INF/applicationContext.xml]
21:07:36,634  INFO XmlWebApplicationContext:refreshBeanFactory:90 - Bean factory for application
context [Root WebApplicationContext]:
org.springframework.beans.factory.support.DefaultListableBeanFactory defining beans
[dataSource,sessionFactory,jdbcExceptionTranslator,hibernateTemplate,transactionManager,todoDao,toD
oServiceTarget,toDoService]; root of BeanFactory hierarchy
...
21:07:36,724  INFO DefaultListableBeanFactory:getBean:222 - Creating shared instance of singleton
bean 'dataSource'
21:07:36,774  INFO DefaultListableBeanFactory:getBean:222 - Creating shared instance of singleton
bean 'sessionFactory'
...
10:38:02,983  INFO DefaultListableBeanFactory:getBean:222 - Creating shared instance of singleton
bean 'jdbcExceptionTranslator'
...
21:07:39,858  INFO DefaultListableBeanFactory:getBean:222 - Creating shared instance of singleton
bean 'hibernateTemplate'
21:07:39,878  INFO DefaultListableBeanFactory:getBean:222 - Creating shared instance of singleton
bean 'transactionManager'
...
21:07:39,918  INFO DefaultListableBeanFactory:getBean:222 - Creating shared instance of singleton
bean 'todoDao'
21:07:39,918 DEBUG ToDoDaoImpl:<init>:24 - #DDD############ Constructor DAO TODO
21:07:39,928  INFO DefaultListableBeanFactory:getBean:222 - Creating shared instance of singleton
bean 'toDoServiceTarget'
21:07:39,928 DEBUG ToDoServiceImpl:setToDoDao:22 - #DDD############ setToDoDao in Service TODO
21:07:39,928  INFO DefaultListableBeanFactory:getBean:222 - Creating shared instance of singleton
bean 'toDoService'
```

Création et liste des Todo.

createTodo.jsp :

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<html>
	<head>
		<title>Create TODO : 111</title>
	</head>
	<body>
		<f:view>
		<h:form id="createToDoForm">
			<h:panelGrid columns="2">
				<h:outputText value="Title : "/>
				<h:inputText  value="#{todoBean.toDo.title}"/>
				<h:outputText value="Body : "/>
				<h:inputText  value="#{todoBean.toDo.body}"/>

		 <h:commandButton value="Submit" action="#{todoBean.createToDoAction}"/>
			</h:panelGrid>
			</h:form>
		</f:view>
	</body>
</html>
```

On remplit le todo avec : `todoBean.toDo.title`

On appelle via un «command button» l'action : `todoBean.createToDoAction`

Cette action fait appel au service, qui lui-même, fait appel au DAO qui sauve le Todo.
Si tout se passe bien, on retourne une String «success» qui permet à JSF de naviguer sur la page viewToDos.jsp

Petit rappel: le document de configuration de JSF spécifie la navigation.

```
	<navigation-rule>
		<from-view-id>/createToDo.jsp</from-view-id>
		<navigation-case>
			<from-outcome>success</from-outcome>
			<to-view-id>/viewToDos.jsp</to-view-id>
		</navigation-case>
		<navigation-case>
			<from-outcome>failure</from-outcome>
			<to-view-id>/error.jsp</to-view-id>
	</navigation-case>
```

viewToDos.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>

<html>
<head>
<title>viewToDos</title>
</head>
<body>
<f:view>
                CREATE :
        <h:outputLink value="createToDo.jsf">
                <h:outputText value="Create" />
        </h:outputLink>
        <br>
        <h:form>
                <h:dataTable value="#{todoBean.toDoS}" var="todotodo">
                        <h:column>
                                <f:facet name="header">
                                        <h:outputText value="Title" />
                                </f:facet>
                                <h:outputText value="#{todotodo.title}" />
                        </h:column>
                        <h:column>
                                <f:facet name="header">
                                        <h:outputText value="Body" />
                                </f:facet>
                                <h:outputText value="#{todotodo.body}" />
                        </h:column>
                        <h:column>
                                <f:facet name="header">
                                        <h:outputText value="Update" />
                                </f:facet>
                                <h:outputLink value="editToDo.jsf">
                                        <h:outputText value="Edit" />
                                        <f:param value="#{todotodo.id}" name="toDoId" />
                                </h:outputLink>
                        </h:column>
                        <h:column>
                                <f:facet name="header">
                                        <h:outputText value="Delete" />
                                </f:facet>
                                <h:outputLink value="viewToDo.jsf">
                                        <h:outputText value="Delete" />
                                        <f:param value="#{todotodo.id}" name="toDoId" />
                                </h:outputLink>
                        </h:column>
                </h:dataTable>
        </h:form>
</f:view>
</body>
</html>
```

Sur cette page on remarquera :

- le lien sur createToDo.jsf
- Une «datatable» qui liste la collection toDo.
- Pour chaque item un lien sur editToDo.jsf avec le paramètre toDoId.
- Pour chaque item un lien sur viewToDo.jsf avec le paramètre toDoId.

Update

editToDo.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<html>
	<head>
		<title>Edit ToDO ONE</title>
	</head>
	<body>
		<f:view>
		<h:form id="editToDoForm">
			<h:panelGrid columns="2">
				<h:outputText value="Id : "/>
				<h:inputText  value="#{todoBean.toDo.id}"/>
				<h:outputText value="Title : "/>
				<h:inputText  value="#{todoBean.toDo.title}"/>
				<h:outputText value="Body : "/>
				<h:inputText  value="#{todoBean.toDo.body}"/>
		 <h:commandButton value="Submit" action="#{todoBean.updateToDoAction}"/>
			</h:panelGrid>
		</h:form>
		</f:view>
	</body>
</html>
```

Delete

viewToDo.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>

<html>
<head>
<title>ViewToDO ONE</title>
</head>
<body>
<f:view>
	<h:form id="viewToDoForm">
		<h:inputHidden value="#{todoBean.toDoId}" />
		<h:panelGrid columns="2">
			<h:outputText value="Id : " />
			<h:outputText value="#{todoBean.toDo.id}" />
			<h:outputText value="Title : " />
			<h:outputText value="#{todoBean.toDo.title}" />
			<h:outputText value="Body : " />
			<h:outputText value="#{todoBean.toDo.body}" />
			<h:commandButton value="Delete" action="#{todoBean.deleteToDoAction}" />
				<f:param value="" />
			</h:commandButton>
		</h:panelGrid>
	</h:form>
</f:view>
</body>
</html>
```

Conclusion

Il est difficile de faire plus simple: un objet par couche avec l'objet métier à persister.

Spring est réellement efficace et gère à votre place les problématiques de communication inter-couche, les problématiques de transaction, etc.

Pour finir de vous convaincre je vous invite à lire l'article suivant: «Java with Spring just as productive as a 4GL RAD tool » by Ervin Bolwidth and Vencent Partingto.
http://www.xebia.com/oth_publications_java_with_spring_just_as_productive_as_4gl.html