Keyos Editions

# Modèle d'application Spring

## Table des matières

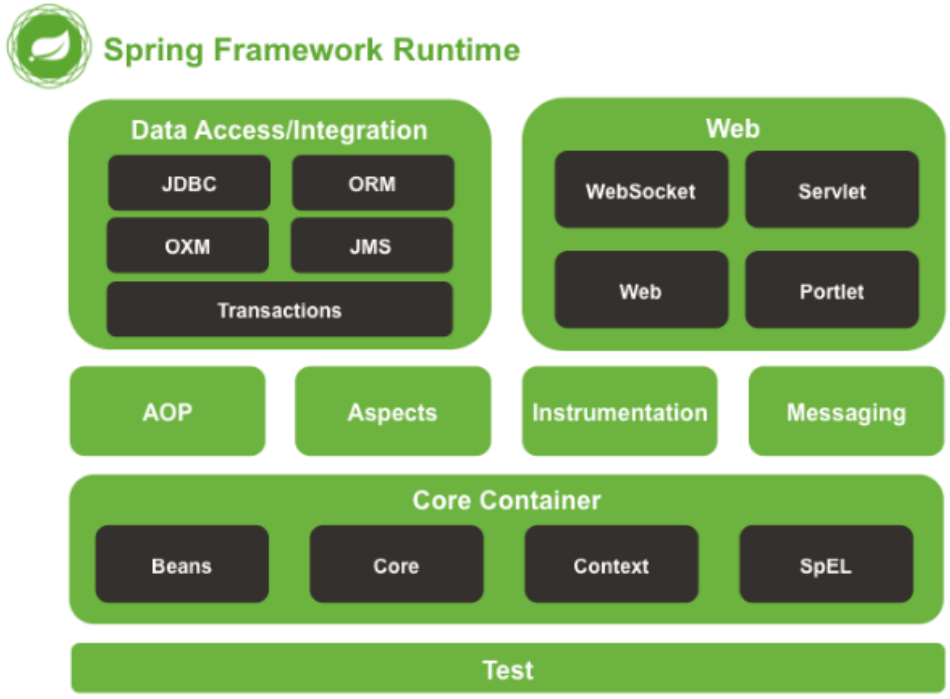
# 1. Instalation de Spring

Spring est une Framework Java qui permets de développer des applications orientées serveur d'applications.

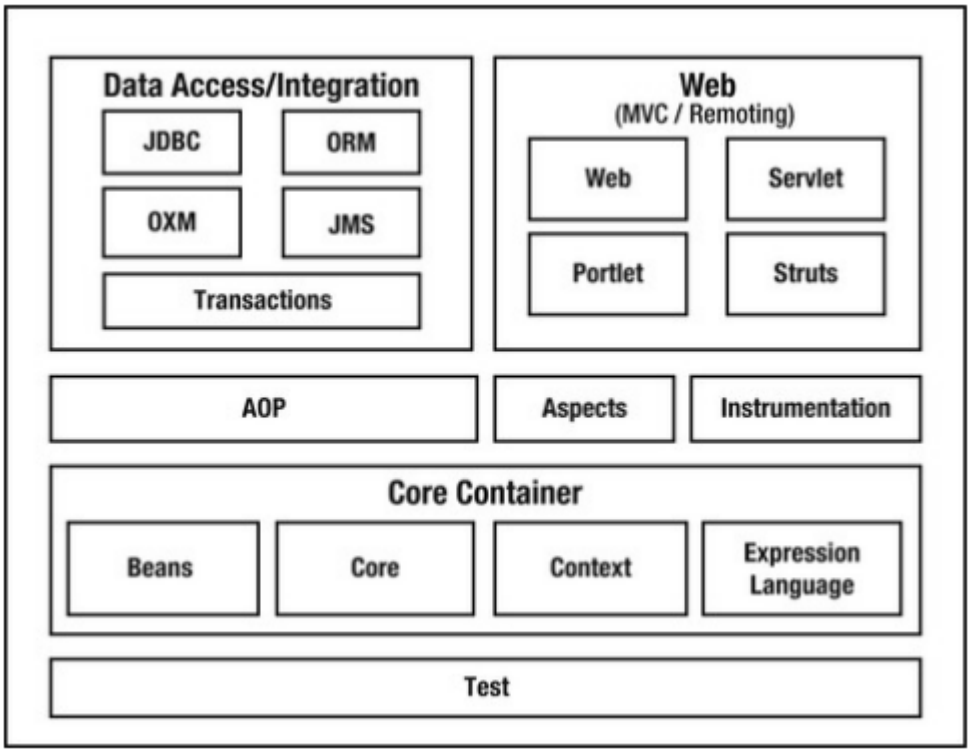
Spring est une Alternative à Java/JEE est prétend apportes les même services avec plus de simplicité ou en tout cas moins de lourdeur. Notamment s'agissant des EJB et on parle d'injection de code.

Caractéristiques :

- Dependency Injection
- Aspect-Oriented Programming avec gestion des transaction
- Spring MVC web  avec RESTful web service framework
- Support et services des technologies :  JDBC, JPA, JMS, EJB, ...




Extension de Spring




Framework Spring : plusieurs versions de Spring et plusieurs sites de téléchargement

http://maven.springframework.org/release/org/springframework/spring/
http://olex.openlogic.com/packages/spring/4.0.1

**Download(s) for this Version:**

| Platform | Name | Date Published | Filesize | Checksum | |
|---|---|---|---|---|---|
| All | Spring 4.2.5 ALL Binary | 2016-03-25 | 64 MB | MD5 | Download Now |
| All | Spring 4.2.5 (zip) ALL Source | 2016-03-25 | 16.7 MB | MD5 | Download Now |
| All | Spring 4.2.5 (tar.gz) ALL Source | 2016-03-25 | 9.85 MB | MD5 | Download Now |

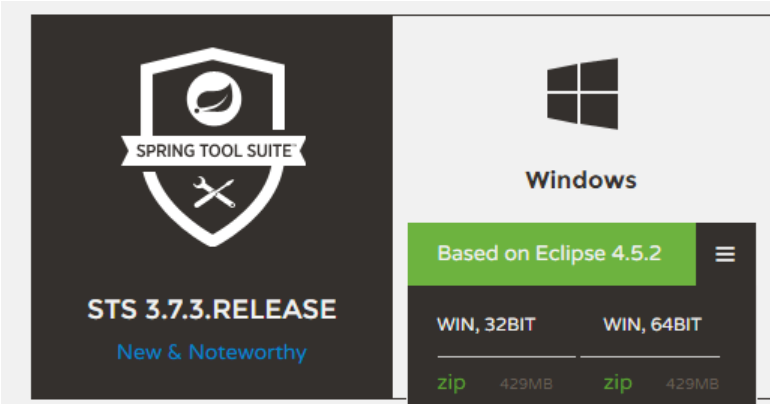
spring-framework-3.2.9.RELEASE-dist.zip
spring-framework-3.2.9.RELEASE-docs.zip
spring-framework-3.2.9.RELEASE-schema.zip

Versions Java : JDK 6+ for Spring Framework 4.x

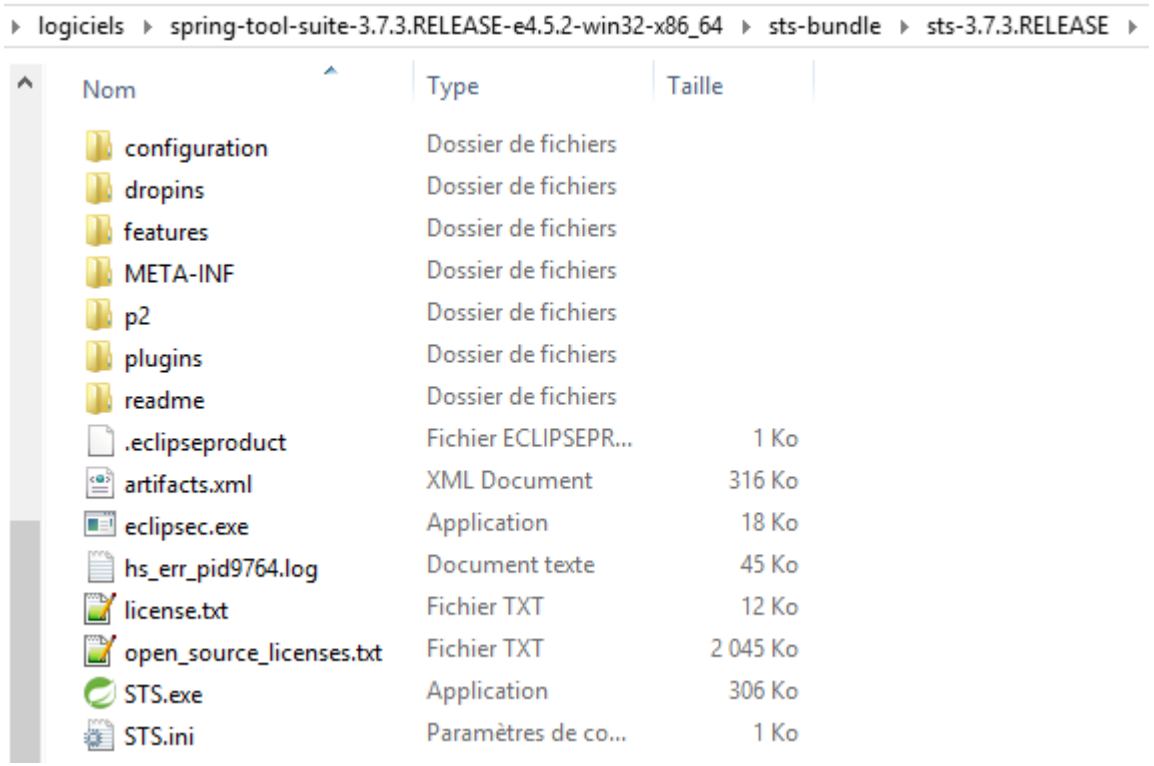
## 2.l'IDE

L'IDE peu être Eclipse avec des plugins appropriés ou directement l'environnement déjà préparé nommé STS (Spring Tool Suite) de Spring. STS est basé sur Eclipse

Download :
- Download / Installation STS (https://spring.io/tools/sts/all)
- Installation de Spring (version 4.x, 4.x)
- Librairies Spring: Hibernate,..
- Librairies générale de Java : jstl, domxml, driver,
- Tomcat pour la programmation WEB
- Base de données Mysql, Postgres,




Contenu de STS




### *Centrale Maven pour les projets Spring*

Il existe plus artefacts Maven pour Spring (.pom) pour créer des modèles d'applications
spring-aop spring-context-support spring-instrument-tomcat spring-oxm spring-web
spring-aspects spring-core spring-jdbc spring-struts spring-webmvc
spring-beans spring-expression spring-jms spring-test spring-webmvc-portlet
spring-context spring-instrument spring-orm spring-tx spring-websocket

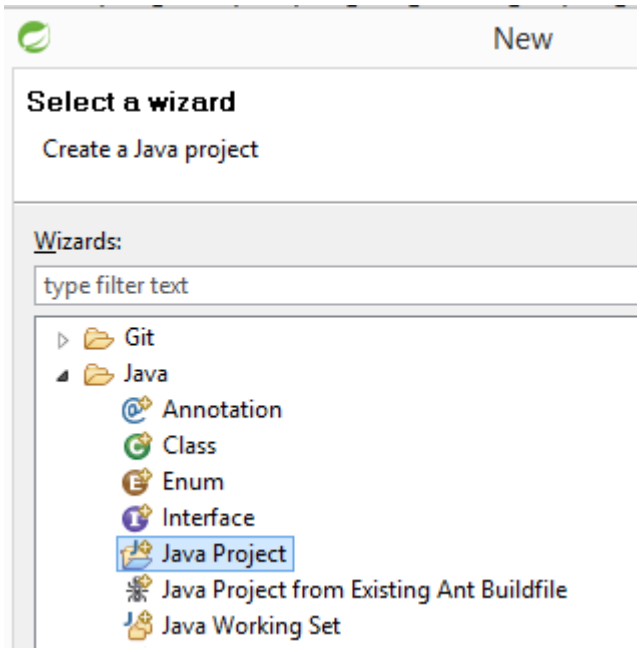
Les dépendances sont mises dans les fichiers xml

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.0.3.RELEASE</version>
</dependency>
```

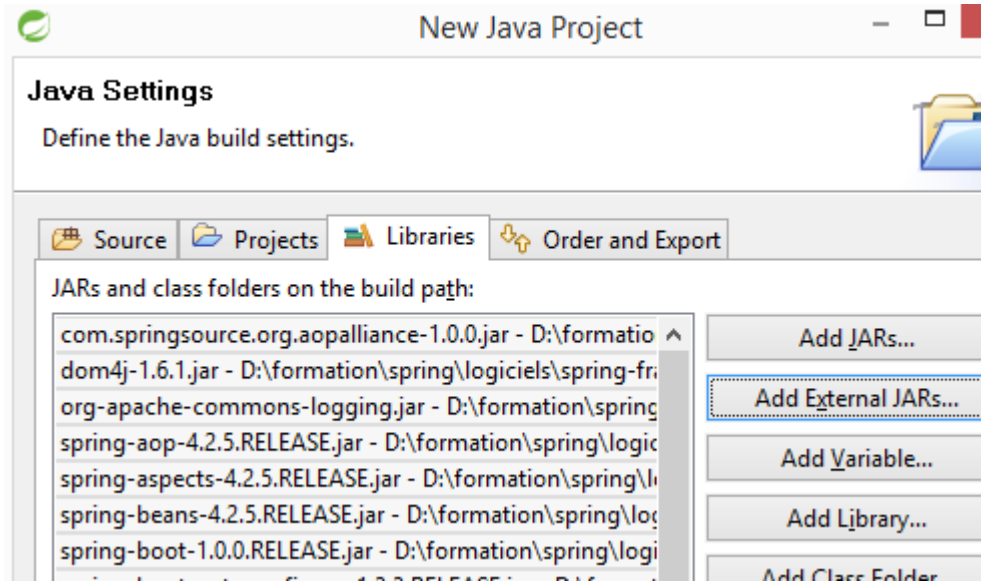
http://mvnrepository.com/artifact/org.springframework/spring-core/3.1.1.RELEASE
http://maven.springframework.org/release/org/springframework/spring/

### *Création d'un projet STS*

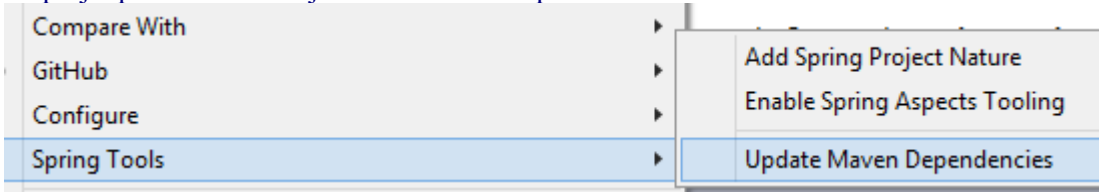
La création peut être faire en partant d'arctefact maven en partant simplement d'un projet Java.




Il faut déclarer dans le buildpath de Eclipse l'accès aux fichiers jar de la librairie Spring (version) : D:\formation\spring\logiciels\spring-framework-4.2.5.RELEASE-dist\spring-framework-4.2.5.RELEASE\libs




Le projet peut être mise à jour en terme de dépendances



## 3. Application Java selon Spring

### *Types d'applications*

Spring couvre divers types d'applications :
- Application Java classique
- Application Java classique avec EJB, JMS, Hibernate
- Application Java Web MVC
- Applications Java WebServices : SOAP, REST
- Applications Java Intégration
- Applications Java Work Flow
- Applications Java Batch

### *Application minimale*

Les applications Java font apparaître un fichier de beans dans lequel sont déclarés les Bean java qui représente un Modèle ou un service.

Le bean Bonjour.java

```
package formation;

public class Bonjour {
  private String message;

  public void setMessage(String message){
    this.message  = message;
  }

  public void getMessage(){
    System.out.println("Le Message : " + message);
  }
}
```

La déclaration des beans : beans.xml

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

  <bean id="bonjour" class="formation.Bonjour">
     <property name="message" value="Bonjour à tous !"/>
  </bean>
```

```
</beans>
```

Principal.java

```
package formation;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Principal {
  @SuppressWarnings("resource")
    public static void main(String[] args) {
        ApplicationContext context =  new ClassPathXmlApplicationContext("Beans.xml");
        Bonjour obj = (Bonjour) context.getBean("bonjour");
        obj.getMessage();
      }
}
```

### *Ajouter des phases d'initialisation et de dispose pour un bean*

Code des événements

```
package formation;

import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.beans.BeansException;

public class InitBonjour implements BeanPostProcessor {

 public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {
    System.out.println("BeforeInitialization : " + beanName);
    return bean;
  }

public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {
    System.out.println("AfterInitialization : " + beanName);
    return bean;
  }
}
```

Configuration du beans.xml

```
<bean id="bonjour" class="formation.Bonjour" >
  <property name="message" value="Bonjour à tous !" />
</bean>
```

```
<bean class="formation.InitBonjour" />
```

### *Doter le bean de méthodes init, destroy*

```
package formation;

public class Bonjour {
  private String message;

  public void setMessage(String message){
    this.message  = message;
  }

  public void getMessage(){
    System.out.println("Your Message : " + message);
  }

  public void init(){
      System.out.println("Appel de init.");
  }

  public void destroy(){
      System.out.println("Appel de destroy .");
  }
}
```

Enregistrer le hook (crocher)

```
public class Principal {
  @SuppressWarnings("resource")
    public static void main(String[] args) {
        ApplicationContext context =  new ClassPathXmlApplicationContext("Beans.xml");
        Bonjour obj = (Bonjour) context.getBean("bonjour");
        obj.getMessage();
        ((AbstractApplicationContext) context).registerShutdownHook();
    }
}
```

## 4. Spring Expression Language (SpEL)

### *Evaluation des littéraux*

Chaine

```
ExpressionParser parser = new SpelExpressionParser();
```

```
String bonjour = (String) parser.parseExpression("'Bonjour'").getValue();
System.out.println (bonjour);
```

Booléen

```
Boolean vrai = (Boolean) parser.parseExpression("true").getValue();
System.out.println (vrai);
```

Les expression de tableaux

```
List nbs = (List) parser.parseExpression("{1,2,3,4}").getValue();
for (Object i : nbs)
{
  System.out.println (i);
}
nbs = (List) parser.parseExpression("{{'A1','A2'},{'B1','B2'}}").getValue();
for (Object i : nbs)
{
  System.out.println (i);
}
```

### *Evaluation des création de tableaux*

```
// Création de tableaux
int[] nbs1 = (int[]) parser.parseExpression("new int[4]").getValue();
for (Object i : nbs1)
{
  System.out.println (i);
}
int[] nbs2 = (int[]) parser.parseExpression("new int[]{1,2,3}").getValue();
for (Object i : nbs2)
{
  System.out.println (i);
}
```

### *Evaluation de chaines*

```
String c = parser.parseExpression("'AZERTY'.substring(2, 3)").getValue(String.class);
boolean isMember = parser.parseExpression("isMember('Mihajlo Pupin')").getValue();
```

### *Evaluation d'expressions numériques et logiques*

```
boolean vrai = parser.parseExpression("2 == 2").getValue(Boolean.class);
System.out.println (vrai);
vrai = parser.parseExpression("2 < -5.0").getValue(Boolean.class);
```

```
System.out.println (vrai);
vrai = parser.parseExpression("true and false").getValue(Boolean.class);
System.out.println (vrai);
boolean vrai = parser.parseExpression("'xyz' instanceof T(int)").getValue(Boolean.class);
System.out.println (vrai);
```

### *Evaluation d'expressions arithmétiques*

```
int res = parser.parseExpression("1 + 1").getValue(Integer.class); // 2
System.out.println (res);
res = parser.parseExpression("1 - -3").getValue(Integer.class); // 4
System.out.println (res);
res = parser.parseExpression("-2 * -3").getValue(Integer.class); // 6
System.out.println (res);
res = parser.parseExpression("7 % 4").getValue(Integer.class); // 3
System.out.println (res);
```

### *Evaluation des types de données Java et réflexion*

```
// Type de données Java et réflexion
Class stringClass = parser.parseExpression("T(String)").getValue(Class.class);
for (Method m : stringClass.getMethods())
{
  System.out.println (m.toString());
}
```

## 5. Injections de dépendances

L'injection permet de maintenir un bon niveau d'indépendance dans le code. L'injection met en œuvre le concept de bean qui peut être défini par fichier de configuration XML ou par annotations

### *Déclaration de bean et construction d'instances*

Déclaration de l'instance de bean avec setter

```
<!-- Injection de propriété -->
<bean id="p1" class="formation.Personne">
      <property name="nom" value="TOTO1" />
      <property name="prenom" value="toto1" />
      <property name="age" value="1" />
</bean>
```

Code qui appelle

```
ApplicationContext context =  new ClassPathXmlApplicationContext("Spring.xml");
Personne p1  = (Personne) context.getBean("p1");
```

```
System.out.println (p1.getAll ());
```

Déclaration de l'instance de bean avec constructeur

```
<bean id="p1" class="formation.Personne">
    <constructor-arg  index="0">
            <value>TOTO2</value>
            </constructor-arg >
            <constructor-arg  index="1">
            <value>toto2</value>
            </constructor-arg >
            <constructor-arg  index="2">
            <value>2</value>
    </constructor-arg >
</bean>
```

### *Injection d'instances*

On va injecter des instances de Personne dans une instance de collection StockPersonne.

La collection

```
import java.util.LinkedList;
import java.util.List;

public class StockPersonne {

    List<Personne> data = new LinkedList<Personne>();
    public void setItem (Personne p) {
            data.add(p);
    }
    public void setItems (Personne [] lp) {
            for (Personne p : lp) data.add(p);
    }
    public List<Personne> getAllItems ()
    {
            return data;
    }
}
```

Définition du bean

```
<bean id="stockPersonne" class="formation.StockPersonne">
    <property name="items">
    <list>
```

```
        <ref bean="p1"/>
        <ref bean="p2"/>
        <ref bean="p3"/>
       </list>
  </property>
 </bean>
```

Code qui appelle

```
public static void main(String[] args) {
        ApplicationContext context =  new ClassPathXmlApplicationContext("Spring.xml");
        StockPersonne sp  = (StockPersonne) context.getBean("stockPersonne");
        for (Personne p : sp.getAllItems())
        {
                System.out.println (p.getAll ());
        }
}
```

### *Annotation @autowired*

@Autowired permet de lier des objets entre eux par configuration XML, soit

- au niveau propriété
- au niveau setter
- au niveau constructeur

Activation des annotations et parcourir les packages à la recherche des beans
<context:component-scan base-package="com.xxx" />

Activation des annotations
<context:annotation-config />

#### Au niveau propriété

Le nom de la propriété est lié au id du bean ayant le même nom

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

public class Personne {

   private int id;
   @Autowired
   private String nom;
   @Autowired
   private String prenom;
   @Autowired
```

```
private int age;
  public String getNom() {
            return nom;
  }
  public void setNom(String nom) {
            this.nom = nom;
  }
  public String getPrenom() {
            return prenom;
  }
  public void setPrenom(String prenom) {
            this.prenom = prenom;
  }
  public int getAge() {
            return age;
  }
  public void setAge(int age) {
            this.age = age;
  }

  public int getId() {
            return id;
  }
  public void setId(int id) {
            this.id = id;
  }
  public Personne ()
  {
            /*
             nom="VIDE";
            prenom="vide";
            age = -1;*/
  }
  Personne (
                        String nom,
                        String prenom,
                        int age)
  {
            this.nom=nom;
            this.prenom=prenom;
```

```
                this.age = age;
        }
        String getAll ()
        {
                return String.format("%s %s %d", nom, prenom, age);
        }
}
```

Le fichier de configuration xml

```
<context:annotation-config />

<bean id="prenom" class="java.lang.String">
      <constructor-arg value="letoto3" />
</bean>
<bean id="age" class="java.lang.Integer">
      <constructor-arg value="3" />
</bean>
<!-- Injection de propriété -->
<bean id="personne" class="formation.Personne" >
</bean>
```

Code principal

```
public class Principal {

  @SuppressWarnings("resource")
     public static void main(String[] args) {
         ApplicationContext context =  new ClassPathXmlApplicationContext("Beans.xml");
         Personne p  = (Personne) context.getBean("personne");
         System.out.println (p.getAll ());
     }
}
```

**Au niveau setter**

```
    @Autowired
     public void setNom(String nom) {
                this.nom = nom;
       }
     @Autowired
       public void setPrenom(String prenom) {
                this.prenom = prenom;
       }
```

```
@Autowired
  public void setAge(int age) {
        this.age = age;
  }
```

**Au niveau constructeur**

Qualifier permet de matcher plus précisément le bean en évoquant son id

```
@Autowired
Personne (
        @Qualifier("lenom") String nom,
        @Qualifier("leprenom") String prenom,
        @Qualifier("lage")int age)
{
    this.nom=nom;
    this.prenom=prenom;
    this.age = age;
    }
```

### *Annotations*

Les annotations remplacent partiellement et avantageusement le directives effectuées dans les fichiers XML
Le classes déclarées dans le code sont traitées par Spring différemment par le compilateurs à l'intention du compilateur et du contenu Spring

@Component =generic stereotype for any Spring-managed component
@Repository=stereotype for persistence layer
@Service =stereotype for service layer
@Controller=stereotype for presentation layer (spring-mvc)

## 6. Spring profiles

### *Profils*

Permet de faire des enregistrements de beans de manière conditionnelle dépendant du profil choisi : dev, test, …

Activer le profil

```
public static void main(String[] args) {
  AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext();
  context.getEnvironment().setActiveProfiles("dev");
  context.register(AppConfig.class);
  context.refresh();
  ((ConfigurableApplicationContext) context).close();
}
```

Activer par variable d'environnement

```
public static void main(String[] args) {
  System.setProperty(AbstractEnvironment.ACTIVE_PROFILES_PROPERTY_NAME, "dev");
  ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
}
```

Par annotations

```
@ActiveProfiles("dev")
class CacheConfigDev {
…..
}
```

Dans les propriétés

```
spring.profiles.active=dev,hsqldb
```

Paramètre de Java

```
-Dspring.profiles.active=dev
```

Pour les applications Web : web.xml

```
<context-param>
   <param-name>spring.profiles.active</param-name>
   <param-value>profileName</param-value>
</context-param>
```

## 7.Gestion des beans

### *Bean Scopes*

L'instanciation de Bean est contrôlée par le scope :

- singleton= Un seul bean est autorisé(défaut).
- prototype=Un bean pour plusieurs instances
- request=Pour les requêtes HTTP
- session=Pour les HTTP sessions

Le bean

```
<bean id="p1" class="formation.Personne" scope="singleton"/>
```

Singleton, le bean n'est instancié qu'une seule fois

```
ApplicationContext context =
      new ClassPathXmlApplicationContext("Beans.xml");
Personne p1 = (Personne) context.getBean("p1");
p1.setNom ("TOTO1");
p1.setPrenom ("toto1");
```

```
p1.setAge (1);
System.out.println(p1.getAll());
Personne p2 = (Personne) context.getBean("p1");
System.out.println(p2.getAll());
```

Résultat

```
TOTO1 toto1 1
TOTO1 toto1 1
```

prototype

```
<bean id="p1" class="formation.Personne" scope="prototype"/>
```

Résultat

```
TOTO1 toto1 1
null null 0
```

Au moyen d'annotation

```
@Component
@Scope("prototype")
class XX
{
}
```

## 8. Modèle bean et la persistance de nature base de données

### *Création de la base*

```
mysql> create database formations;
Query OK, 1 row affected (0.05 sec)
mysql> \u formations
Database changed
mysql> create table lesformations (id int, libelle varchar(100));
Query OK, 0 rows affected (0.27 sec)
mysql> insert into lesformations values (1,'Formation 1');
Query OK, 1 row affected (0.08 sec)
mysql> insert into lesformations values (2,'Formation 2');
Query OK, 1 row affected (0.06 sec)
mysql> insert into lesformations values (3,'Formation 3');
Query OK, 1 row affected (0.05 sec)
```

### *Accès aux données avec un RowMappers et JdbcTemplate*

Auto incrémentation pour le ID

```
alter table personnes modify id int(4) unsigned auto_increment;
```

Le mapper sert de passerelle entre la base de données (resultSet) et le bean (Instance).

Afficher le contenu de la base de données : la table formation contient deux colonnes id,libelle

```
public class FormationMapper implements RowMapper<Formation> {
        public Formation mapRow(ResultSet rs, int rowNum) throws SQLException {
                Formation formation = new Formation();
                formation.setId(rs.getInt("id"));
…..
```

Soit un bean Formation

```
package formation;

public class Formation
{
	int id;
	String libelle;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getLibelle() {
		return libelle;
	}
	public void setLibelle(String libelle) {
		this.libelle = libelle;
	}
}
```

Mettre en place un mapper (bridge)

```
package formation;

import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;

public class FormationMapper implements RowMapper<Formation> {
	public Formation mapRow(ResultSet rs, int rowNum) throws SQLException {
		Formation formation = new Formation();
		formation.setId(rs.getInt("id"));
		formation.setLibelle(rs.getString("Libelle"));
	  return formation;
	}
}
```

Déclaration du datasource

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

      <bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/formations"/>
        <property name="username" value="root"/>
        <property name="password" value=""/>
      </bean>

  </beans>
```

Programme principal : lire les données de la table

```
import javax.sql.DataSource;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

public class Principal {
@SuppressWarnings("resource")
    public static void main(String[] args) {
        ApplicationContext context =  new ClassPathXmlApplicationContext("Beans.xml");
        DataSource ds = (DataSource) context.getBean("dataSource");
        String SQL = "select * from lesformations";
        JdbcTemplate jt = new JdbcTemplate(ds);
        List<Formation> formations = jt.query(SQL, new FormationMapper());
        for (Formation l : formations)
        {
            System.out.println (l.getLibelle());
        }
    }
}
```

Ajouter un record à la base : insert

```
package formation;

import javax.sql.DataSource;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
import org.springframework.jdbc.core.JdbcTemplate;

public class Principal {
  @SuppressWarnings("resource")
    public static void main(String[] args) {
        ApplicationContext context =  new ClassPathXmlApplicationContext("Beans.xml");
        DataSource ds = (DataSource) context.getBean("dataSource");
        JdbcTemplate jt = new JdbcTemplate(ds);
        String SQL = "insert into lesformations (id, libelle) values (?, ?)";
        jt.update( SQL, new Object [] {1010, "Formation " + 1010 } );
  }
}
```

### *Accès aux données avec les DAO notations*

L'utilisation de la méthode des DAO fait intervenir un niveau d'abstraction basé sur une interface

Une entitée Personne

```
public class Personne {
    String nom;
    String prenom;
    int age;
    int id;
    public int getId() {
            return id;
    }
    public void setId(int id) {
            this.id = id;
    }
    public String getNom() {
            return nom;
    }
    public void setNom(String nom) {
            this.nom = nom;
    }
    public String getPrenom() {
            return prenom;
    }
    public void setPrenom(String prenom) {
            this.prenom = prenom;
    }
```

```
        public int getAge() {
                return age;
        }
        public void setAge(int age) {
                this.age = age;
        }

        Personne ()
        {
                nom="VIDE";
                prenom="vide";
                age = -1;
        }
        String getAll ()
        {
                return String.format("%s %s %d", nom, prenom, age);
        }
    }
```

L'interface PersonneDao

```
    import java.util.List;
    import javax.sql.DataSource;

    public interface PersonneDao {
      public void setDataSource(DataSource ds);
      public void create(String nom, String prenom, Integer age);
      public void create(Personne p);
      public Personne getPersonne(Integer id);
      public List<Personne> listPersonnes();
      public void delete(Integer id);
      public void update(Integer id, Integer age);
    }
```

Le mapper : PersonneMapper

```
    import java.sql.ResultSet;
    import java.sql.SQLException;
    import org.springframework.jdbc.core.RowMapper;

    public class PersonneMapper implements RowMapper<Personne> {
      public Personne mapRow(ResultSet rs, int rowNum) throws SQLException {
```

```
        Personne Personne = new Personne();
        Personne.setId(rs.getInt("id"));
        Personne.setNom(rs.getString("nom"));
        Personne.setPrenom(rs.getString("prenom"));
        Personne.setAge(rs.getInt("age"));
        return Personne;
      }
    }
```

Template JDBC

```
    import java.util.List;
    import javax.sql.DataSource;
    import org.springframework.jdbc.core.JdbcTemplate;

    public class PersonneJDBCTemplate implements PersonneDao {
      private DataSource dataSource;
      private JdbcTemplate jdbcTemplateObject;

      public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
        this.jdbcTemplateObject = new JdbcTemplate(dataSource);
      }

      public void create(String nom, String prenom, Integer age) {
        String SQL = "insert into Personnes (nom, prenom, age) values (?, ?, ?)";

        jdbcTemplateObject.update( SQL, nom, prenom, age);
        System.out.println("Created Record Name = " + nom + " Prenom= " + prenom + " Age = " + age);
        return;
      }

      public void create(Personne p) {
            String SQL = "insert into Personnes (nom, prenom, age) values (?, ?, ?)";

            jdbcTemplateObject.update( SQL, p.getNom(), p.getPrenom(), p.getAge());
            System.out.println("Created Record Name = " + p.getNom() + " Pernom= " + p.getPrenom() + "
    Age = " + p.getAge());
            return;
      }
```

```
  public Personne getPersonne(Integer id) {
    String SQL = "select * from Personnes where id = ?";
    Personne Personne = jdbcTemplateObject.queryForObject(SQL,  new Object[]{id},  new
PersonneMapper());
    return Personne;
  }

  public List<Personne> listPersonnes() {
    String SQL = "select * from Personnes";
    List <Personne> Personnes = jdbcTemplateObject.query(SQL, new PersonneMapper());
    return Personnes;
  }

  public void delete(Integer id){
    String SQL = "delete from Personnes where id = ?";
    jdbcTemplateObject.update(SQL, id);
    System.out.println("Deleted Record with ID = " + id );
    return;
  }

  public void update(Integer id, Integer age){
    String SQL = "update Personnes set age = ? where id = ?";
    jdbcTemplateObject.update(SQL, age, id);
    System.out.println("Updated Record with ID = " + id );
    return;
  }
}
```

Code Principal

```
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Principal {
  @SuppressWarnings("resource")
     public static void main(String[] args) {
                ClassPathXmlApplicationContext context =  new
ClassPathXmlApplicationContext("Spring.xml");
          PersonneJDBCTemplate pjt=
                   (PersonneJDBCTemplate)context.getBean("personneJDBCTemplate");
          Personne personne = new Personne();
```

```
            personne.setNom("BELHADJkarim");
            personne.setPrenom("karim");
            personne.setAge(54);
            //pjt.create(personne);
            //System.out.println("personne : " + personne.getAll() + " ajouté");

            List<Personne> personnes = pjt.listPersonnes();
            for (Personne p : personnes)
            {
                  System.out.println(p.getAll());
            }
            context.close();
      }
}
```

### *Accès aux données avec namedParameterJdbcTemplate*

On utilise les variables binding et une collection Map<>

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.sql.DataSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

public class PersonneJDBCTemplate implements PersonneDao {
  private DataSource dataSource;
  private NamedParameterJdbcTemplate jdbcTemplateObject;

  public void setDataSource(DataSource dataSource) {
    this.dataSource = dataSource;
    this.jdbcTemplateObject = new NamedParameterJdbcTemplate(dataSource);
  }

  public void create(String nom, String prenom, Integer age) {
    String SQL = "insert into Personnes (nom, prenom, age) values (:nom, :prenom, :age)";
    Map<String, Object> argMap = new HashMap<String, Object>();
    argMap.put("nom", nom);
    argMap.put("prenom", prenom);
    argMap.put("age", age);
    jdbcTemplateObject.update(SQL, argMap);
```

```
    System.out.println("Created Record Name = " + nom + " Prenom= " + prenom + " Age = " + age);

    return;
  }

  public void create(Personne p) {
        String SQL = "insert into Personnes (nom, prenom, age) values (:nom, :prenom, :age)";
        Map<String, Object> argMap = new HashMap<String, Object>();
        argMap.put("nom", p.getNom());
        argMap.put("prenom", p.getPrenom());
        argMap.put("age", p.getAge());
        jdbcTemplateObject.update(SQL, argMap);
        System.out.println("Created Record Name = " + p.getNom() + " Pernom= " + p.getPrenom() + "
Age = " + p.getAge());
        return;
  }

  public Personne getPersonne(Integer id) {
    String SQL = "select * from Personnes where id = :id";
    Map<String, Object> argMap = new HashMap<String, Object>();
    argMap.put("id", id);
    jdbcTemplateObject.update(SQL, argMap);
    Personne Personne = jdbcTemplateObject.queryForObject(SQL,  argMap, new PersonneMapper());
    return Personne;
  }

  public List<Personne> listPersonnes() {
    String SQL = "select * from Personnes";
    List <Personne> Personnes = jdbcTemplateObject.query(SQL, new PersonneMapper());
    return Personnes;
  }

  public void delete(Integer id){
    String SQL = "delete from Personnes where id = :id";
    Map<String, Object> argMap = new HashMap<String, Object>();
    argMap.put("id", id);
    jdbcTemplateObject.update(SQL, argMap);
    System.out.println("Deleted Record with ID = " + id );
    return;
  }
```

```
    public void update(Integer id, Integer age){
      String SQL = "update Personnes set age = ? where id = :id";
      Map<String, Object> argMap = new HashMap<String, Object>();
      argMap.put("id", id);
      argMap.put("age", age);
      jdbcTemplateObject.update(SQL, argMap);
      System.out.println("Updated Record with ID = " + id );
      return;
    }

  }
```

Batch en insert avec BeanPropertySqlParameterSource

```
  public void insertBatchNamedParameter(final List<Customer> Personnes)
  {
      String sql = "INSERT INTO Personnes" + "(nom, prenom, age) VALUES (:nom, :prenom, :age)";
      List<SqlParameterSource> parameters = new ArrayList<SqlParameterSource>();
      for (Personne p : Personnes) {
            parameters.add(new BeanPropertySqlParameterSource(p));
      }
      getSimpleJdbcTemplate().batchUpdate(sql,parameters.toArray(new SqlParameterSource[0]));
  }
```

Plus simple avec createBatch

```
  public void insertBatchNamedParameter2(final List<Personnes> Personnes){
      SqlParameterSource[] params =
      SqlParameterSourceUtils.createBatch(Personnes.toArray());
      getSimpleJdbcTemplate().batchUpdate(
            "INSERT INTO Personnes (nom, prenom, age) VALUES (:nom, :prenom, :age)",params
      }
```

### *BatchSqlUpdate*

Pour le traitement par lot, il est nécessaire d'optimiser les opérations de mise à jour en faisant appel à BatchSqlUpdate

Révision de l'interface DOA

```
  import java.util.List;
  import javax.sql.DataSource;

  public interface PersonneDao {
```

```
    public void setDataSource(DataSource ds);
    public void create(String nom, String prenom, Integer age);
    public void create(Personne p);
    public Personne getPersonne(Integer id);
    public List<Personne> listPersonnes();
    public void delete(Integer id);
    public void update(Integer id, Integer age);
    public void lotInsert(List<Personne> p);
}
```

Implémentation lotInsert

```
@Override
public void LotInsert(List<Personne> lp) {

     BatchInsert batchInsert = new BatchInsert(dataSource);
    for (Personne p : lp) {
            batchInsert.update(new Object[] {p.getNom(), p.getPrenom(), p.getAge()});
    }
}
```

Code appelant

```
// remplir
List<Personne> lp = new LinkedList<Personne> ();
for (int i = 0; i < 100; i++) {
      lp.add(new Personne ("TOTO" + i, "toto" + i, i));
}
pjt.LotInsert(lp);
//Lire
List<Personne> personnes = pjt.listPersonnes();
for (Personne p : personnes)
{
 System.out.println(p.getAll());
}
```

### *Spring et la persistance JPA*

Jpa est une API de persistance standardisée dans le serveur d'application JEE. JPA va cacher toute la mécanique de traduction entre les objets en mémoire et leur stockage dans une base de données par exemple.

JPA utilise les annotation sur des classe POJOs

Spring/JPA utilise :
- La persistance Unit

- Entity Manager factory
- Transaction Manager

Spring dispose de l'API spring-datajpa qui est une surcouche de jpa et qui facilite le développement.

Download de la librairie :

### *spring-data-jpa*

Dépendance Maven

```
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-jpa</artifactId>
   <version>2.0.8</version>
</dependency>

<dependency>
        <groupId>org.springframework.data</groupId>
        <artifactId>spring-data-releasetrain</artifactId>
        <version>Gosling-RELEASE</version>
</dependency>
```

Spring dispose les providers pour la plupart des bases d données
spring-data-jdbc-core-1.0.0.RELEASE.jar
spring-data-jpa-1.3.0.RELEASE-sources.jar
spring-data-mongodb-1.0.1.RELEASE.jar
spring-data-oracle-1.0.0.RELEASE.jar
spring-data-rest-webmvc-1.0.0.RELEASE.jar
spring-data-redis-1.0.1.RELEASE.jar

### *Spring et la persistance avec EntityManager*

La persistance permet de stocker une objet à un format de donnée persistance

Pour cela le format va être donnée par le provider
Configuration pour l'entityMnager : fichier persistance.xml

Le fournisseur (provider) de la couche ORM est fournie par Hibernate

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistance
http://java.sun.com/xml/ns/persistance/persistance_2_0.xsd"
    version="2.0" xmlns="http://java.sun.com/xml/ns/persistance">
     <persistance-unit name="lestotos" transaction-type="RESOURCE_LOCAL">
      <provider>
         org.hibernate.ejb.Hibernatepersistance
```

```
        </provider>
        <properties>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
            <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
            <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/formationsdb"/>
            <property name="hibernate.connection.username" value="root"/>
            <property name="hibernate.connection.password" value=""/>
        </properties>
    </persistance-unit></persistance>
```

Avec transaction-type/RESOURCE_LOCAL, c'est nous qui prenons en charge la gestion du EntityManager
entityManagerFactory.createEntityManager()
création, @PersistenceUnit, EntityTransaction pour valider

transaction-type/JTA (JEE) : c'est le conteneur qui prend en charge la gestion du EntityManager
On n'utilise plus EntityManagerFactory
Injection au moyen de @PersistenceContext et non @PersistenceUnit

### *Bean validation*

Le besoin en terme de validation des données se pose à tous les niveaux des couches logicielles
- Présentation
- Service
- Métier
- DAO
- Dans la base de données via des contraintes d'intégrités

Pour répondre à ces différents besoins, Spring API Bean Validation (JSR 303) :
- fournit des validateurs classiques
- permet de définir ses propres validateurs

Une contrainte est composée de :
- Une annotation
- Une classe de type Validator

Dependance Maven pour l'api

```
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>1.0.0.GA</version>
</dependency>
```

Provider de validation

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
```

```
    <version>4.2.0.Final</version>
  </dependency>
```

Annotations :

- @NotNull(message="Name must be input")
- @NotEmpty
- @NotBlank
- @NotEmpty(message="At least one passenger is required")
- @Size(min=1,max=50, message="Name must not exceed 50 characters")
- @Length(max = 80)
- @Valid
- @Email

Annotation

```
import java.util.Date;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Past;
import javax.validation.constraints.Size;

public class PersonneBean {

  private String nom;
  private String Prenom;
  private Date dateNaissance;

  public PersonneBean(String nom, String prenom, Date dateNaissance) {
    super();
    this.nom = nom;
    Prenom = prenom;
    this.dateNaissance = dateNaissance;
  }

  @NotNull
  @Size(max=50)
  public String getNom() {
    return nom;
  }

  public void setNom(String nom) {
    this.nom = nom;
  }
```

```
  @NotNull
  @Size(max=50)
  public String getPrenom() {
   return Prenom;
  }

  public void setPrenom(String prenom) {
   Prenom = prenom;
  }

  @Past
  public Date getDateNaissance() {
   return dateNaissance;
  }

  public void setDateNaissance(Date dateNaissance) {
   this.dateNaissance = dateNaissance;
  }
 }
```

Api/Evenement

```
 import java.util.Calendar;
 import java.util.GregorianCalendar;
 import java.util.Set;
 import javax.validation.ConstraintViolation;
 import javax.validation.Validation;
 import javax.validation.Validator;
 import javax.validation.ValidatorFactory;

 public class TestValidation {

  public static void main(String[] args) {

   PersonneBean personne = new PersonneBean(null, null, new GregorianCalendar(
     2065, Calendar.JANUARY, 18).getTime());

   ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
   Validator validator = factory.getValidator();

   Set<ConstraintViolation<PersonneBean>> constraintViolations =
```

```
        validator.validate(personne);

      if (constraintViolations.size() > 0 ) {
        System.out.println("Impossible de valider les donnees du bean : ");
        for (ConstraintViolation<PersonneBean> contraintes : constraintViolations) {
          System.out.println(contraintes.getRootBeanClass().getSimpleName()+
            "." + contraintes.getPropertyPath() + " " + contraintes.getMessage());
        }
      } else {
        System.out.println("Les donnees du bean sont valides");
      }
    }
  }
```

Exécution

```
Impossible de valider les données du bean :
PersonneBean.prenom ne peut pas être nul
PersonneBean.dateNaissance doit être dans le passé
PersonneBean.nom ne peut pas être nul
```

## 9. Gestion des transactions

### *L'implémentation des transaction de Spring surpporte les API diverses*

- La gestion de transaction de Spring prend en charge les diverses API des transaction :
- JDBC
- Java Data Objects (JDO).
- Java Transaction API (JTA)
- Java Persistence API (JPA)
- Hibernate

Le dispositif de transaction peut être implémenté à n'importe quelle classe

Les règles de rollback peuvent être écrites au niveau du code et déclaration. Dans les règle on dit qu'elle exception provoquera le rollback

On peut également utiliser les instructions setRollbackOnly() de l'objet TransactionStatus pour effectuer rollback de transaction courante

La gestion des transactions est implémentée
- Par aspect-oriented programming (AOP) : <tx:advice, <tx:attributes, <tx:method
- Au moyen de d'annotation : @Transaction sur classe ou méthode
- Au moyen de déclaration xml

- Possibilité de créer ses propres annotations

On peut également utiliser l'AOP pour contrôler la transaction au moyen de TransactionInterceptor en conjonction avec l'implémentation PlatformTransactionManager

### *Au niveau  moyen d'annotations*

Classe gérée par le dispositif transactionnel

```
@Transactional
public class DefaultFooService implements FooService {
   Foo getFoo(String fooName);
   Foo getFoo(String fooName, String barName);
   void insertFoo(Foo foo);
   void updateFoo(Foo foo);
}
```

Avec des attributs détaillés

```
@Transactional(readOnly = true)
public class DefaultFooService implements FooService {

   public Foo getFoo(String fooName) {
      // do something
   }

   // these settings have precedence for this method
   @Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
   public void updateFoo(Foo foo) {
      // do something
   }
}
```

Complément de la configuration dans le fichier XML

```
<!-- from the file 'context.xml' -->
<?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:aop="http://www.springframework.org/schema/aop"
   xmlns:tx="http://www.springframework.org/schema/tx"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- this is the service object that we want to make transactional -->
    <bean id="fooService" class="x.y.service.DefaultFooService"/>

    <!-- enable the configuration of transactional behavior based on annotations -->
    <tx:annotation-driven transaction-manager="txManager"/>
     <!-- a PlatformTransactionManager is still required -->
    <bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <!-- (this dependency is defined somewhere else) -->
        <property name="dataSource" ref="dataSource"/>
    </bean>
    <!-- other <bean/> definitions here -->
</beans>
```

Appel dans le main

```
public final class Boot {

    public static void main(final String[] args) throws Exception {
        ApplicationContext ctx = new ClassPathXmlApplicationContext("context.xml", Boot.class);
        FooService fooService = (FooService) ctx.getBean("fooService");
        fooService.insertFoo (new Foo());
    }
}
```

Configuration de @Transaction

- rollbackFor : array de classes d'exception provoquant le roolback
- rollbackForClassName : Array de noms de classes d'exception provoquant le roolback
- noRollbackFor : array de classes d'exception ne devant pas provoquer le roolback
- norollbackForClassName : Array de noms de classes d'exception ne devant pas provoquer le roolback

### *Annotation avec plusieurs transactions dans la même classe*

```
public class TransactionalService {

    @Transactional("order")
    public void setSomething(String name) { ... }
```

```
    @Transactional("account")
    public void doSomething() { ... }
}
```

Dans le fichier XML

```
<bean id="transactionManager1"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    ...
    <qualifier value="order"/>
</bean>

<bean id="transactionManager2"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    ...
    <qualifier value="account"/>
</bean>
```

Annotation customisée et implémentation adapté

```
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Transactional("order")
public @interface OrderTx {
}

@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Transactional("account")
public @interface AccountTx {
}
```

Utilisation

```
public class TransactionalService {

    @OrderTx
    public void setSomething(String name) { ... }

    @AccountTx
    public void doSomething() { ... }
}
```

***Au niveau déclaration XML/AOP***

```
<tx:annotation-driven/>
```

```
<aop:config>
  <aop:pointcut id="fooServiceMethods" expression="execution(* x.y.service.*.*(..))"/>
  <aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceMethods"/>
</aop:config>
```

Contenu de context.xml

```
<!-- from the file 'context.xml' -->
<?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:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd">

  <!-- this is the service object that we want to make transactional -->
  <bean id="fooService" class="x.y.service.DefaultFooService"/>

  <!-- the transactional advice (what 'happens'; see the <aop:advisor/> bean below) -->
  <tx:advice id="txAdvice" transaction-manager="txManager">
    <!-- the transactional semantics... -->
    <tx:attributes>
        <!-- all methods starting with 'get' are read-only -->
      <tx:method name="get*" read-only="true"/>
      <!-- other methods use the default transaction settings (see below) -->
      <tx:method name="*"/>
    </tx:attributes>
  </tx:advice>

  <!-- ensure that the above transactional advice runs for any execution
    of an operation defined by the FooService interface -->
  <aop:config>
```

```
        <aop:pointcut id="fooServiceOperation" expression="execution(* x.y.service.FooService.*(..))"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceOperation"/>
    </aop:config>

    <!-- don't forget the DataSource -->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
        <property name="url" value="jdbc:oracle:thin:@rj-t42:1521:elvis"/>
        <property name="username" value="scott"/>
        <property name="password" value="tiger"/>
    </bean>

    <!-- similarly, don't forget the PlatformTransactionManager -->
    <bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>
    <!-- other <bean/> definitions here -->
</beans>
```

Dispositif du rollback

```
    <tx:advice id="txAdvice" transaction-manager="txManager">
        <tx:attributes>
        <tx:method name="get*" read-only="true" rollback-for="NoProductInStockException"/>
        <tx:method name="*"/>
        </tx:attributes>
    </tx:advice>
```

configuration de <tx:advice/> :

- Propagation setting is REQUIRED.
- Isolation level is DEFAULT.
- Transaction is read/write.
- Transaction timeout

<tx:method/>

- propagation : REQUIRED
- isolation :DEFAULT
- timeout :-1 (en seconds).
- read-only :false
- rollback-for : liste des Exception qui provoquent le roolback (separateur = ,)
- no-r ollback-for : liste des Exception ne devant pas provoquer le roolback (separateur = ,)

# Application Java/Spring et le MVC

## 10. Introduction

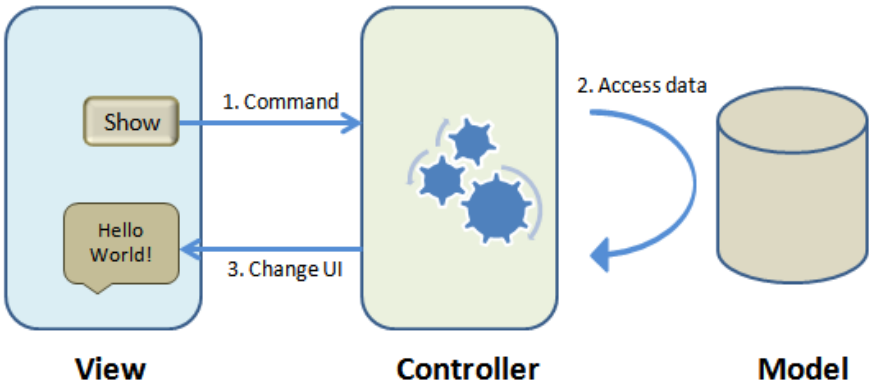
La programmation WEB se décline en partie client léger et en partie Serveur pour les échanges HTTP et contrôles et traitement ou stockage des données.

La technologie client est dominée par les normes :

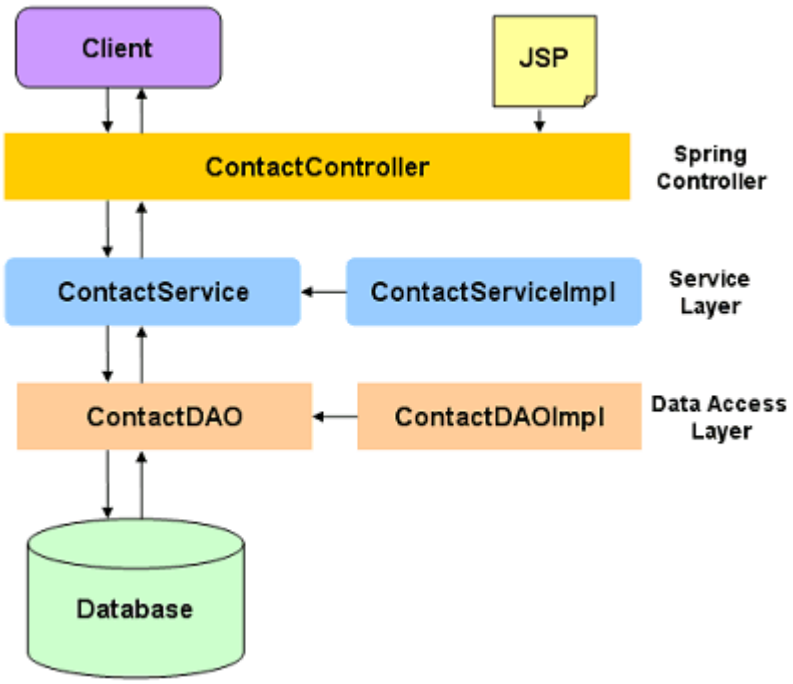
- HTML 5
- CSS
- Javascript

La technologies côté serveur est constituée par un serveur d'applications tel que TOMCAT ou JETTY.

Les technologies Java de base utilisée sont : JSP, JSTL, taglib, Servlet. Mais ces technologies ne permettent pas d'organiser le code complexe des applications. Le découpage MVC est proposé dans le monde du WEB au-delà de Java et apporte beaucoup de rigueur au programmeurs, on parle de framework MVC




Le contrôleur intercepte les interactions du client, il produite les données au travers du modèle avant de passer la main à la vue en direction du client.




### *Application simple avec un JSP seul et sans le contrôleur*

Une page JSP sert de Vue : hello.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@page import="java.util.*" %>
<%
String laliste [] = { "Article 1", "Article 2", "Article 3", "Article 4" };
pageContext.setAttribute("laliste", laliste);
%>
<html>
  <head>
  <title>Application minimale Spring/Web jsp = MVC</title>
  </head>
  <body>
  <h3>Liste de choix : ${ 4 * 1 } </h3>
    <c:forEach var="v" items="${laliste }">
            <li>${v}</li>
    </c:forEach>
  </body>
</html>
```

Fichier de configuration des beans : Hello-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  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="formation" />
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/" />
    <property name="suffix" value=".jsp" />
  </bean>
</beans>
```

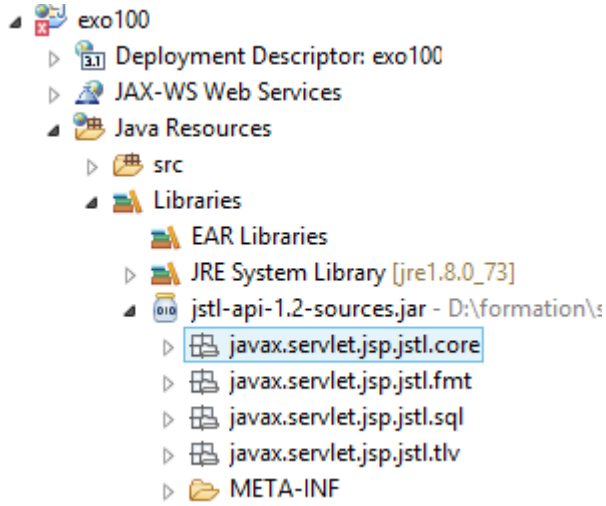
web.xml

```
<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>Spring MVC Application</display-name>
  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

Dans ce cas simple les données proviennent localement du JSP et leur affichage passe par des expressions JSTL.

```
<%
String laliste [] = { "Article 1", "Article 2", "Article 3", "Article 4" };
```

```
pageContext.setAttribute("laliste", laliste);
%>
```




### *Application simple avec un JSP avec le contrôleur*

Le contrôleur réagit à l'url /page, il constitue des données dynamiquement et les fournit à la vue page.jsp

web.xml : le servlet DispatcherServlet sert de point d'entrée

```
<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>Spring MVC Application</display-name>

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

  <servlet-mapping>
    <servlet-name>Page</servlet-name>
    <url-pattern>/page</url-pattern>
  </servlet-mapping>
```

```
</web-app>
```

Le contrôleur est sollicité par la framework qui renvoie le résultat à vue

```
package formation;

import java.io.IOException;
import java.util.ArrayList;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlêtrequest;
import javax.servlet.http.HttpServlêtresponse;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.portlet.ModelAndView;

@Controller
public class PageController{

@RequestMapping(value="/page")
 public String getData(Model m) {
        System.out.println ("Le Controleur ");
        m.addAttribute("message",  "Choisir les éléments");
        ArrayList<String> l = new ArrayList<String>();
        for (int i=0; i<10; i++)
        {
            l.add("Elément de liste " + i);
        }
        m.addAttribute("laliste",  l);
        return "views/page";
   }
}
```

La configuration du Servlet nommé Page est contenue dans Page-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  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="formation" />

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/" />
      <property name="suffix" value=".jsp" />
    </bean>

  </beans>
```

Vue : page.jsp récupérer les données arrivant depuis le contrôleur pour les formater et les afficher.

```
  <%@ page session="true"%>

  <%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
  <%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
  <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
  <%@page import="java.util.*" %>
  <html>
    <head>
    <title>Page 1</title>
    </head>
    <body>
    <h3>Titre  = <c:out value="${message}" /></h3>
    <c:forEach items="${ laliste}" var="l">
       <li>${l}</li>
    </c:forEach>
    </body>
  </html>
```

### *Annotation et le traitement des requêtes Http entrantes*

Le chemin de l'annotation @RequestMapping () est associé à une méthode qui traitent les requêtes entrantes.

Le modèle MVC de Spring utilise en arguement entrant et un valeur de retour un objet Model ou ModelAndView pour permettre au contrôleur de communiquer avec la vue devant s'occuper du rendu.

Le Model représente alors les données que l'on veut communiquer à la Vue pour effectuer un rendu selon MVC.

Le Model contient des données sous forme de clés valeurs donc sous forme de Map

- m.asMap()
- m.addAttribute(arg0, arg1)
- m.addAllAttributes(arg0)

```
@RequestMapping(value="/page/1")
public String getData(Model m) {
        System.out.println ("Le Controlleur page 1");
        m.addAttribute("message",  "Choisir les éléments");
        ArrayList<String> l = new ArrayList<String>();
        for (int i=0; i<10; i++)
        {
                l.add("Elément de liste " + i);
        }
        m.addAttribute("laliste",  l);
        return "views/page1";
```

Côté vue, on peut accéder et afficher les données de l'objet Model grâce à l'opérateur ${} que l'on combine avec JSTL pour créer des rendu dynamique.

Vue accèdant à l'attribut  "laliste" de Model

```
<%@ page session="true"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@page import="java.util.*" %>
<html>
  <head>
  <title>Page 1</title>
  </head>
  <body>
  <h1>Page 1</h1>
  <h3>Titre  = <c:out value="${message}" /></h3>
  <c:forEach items="${ laliste}" var="l">
     <li>${l}</li>
  </c:forEach>
  </body>
</html>
```

Autres requêtes

```
@RequestMapping(value="/page2")
```

```
public String getData(Model m) {
                System.out.println ("Le Controlleur page 2");
                  return "views/page2";
}

@RequestMapping(value = "/page2/1",      )
public void getData(HttpServlêtresponse httpServlêtresponse) {
   httpServlêtresponse.setHeader("Location", "http://www.oracle.com/index.html");
}

@RequestMapping(value = "/page2/2", method = RequestMethod.GET)
public ModelAndView getData() {
      return new ModelAndView("redirect:http://www.ibm.com/en-us/homepage-a.html");

}

@RequestMapping(value="/google")
public String getData2(Model m) {
        System.out.println ("Le Controlleur page 2");
        return "redirect:http://www.google.fr";
}
```

### *ModelAndView, Model*

Spring propose également ModelAndView qui manipule la vue et les données à transmettre pour le rendu

new ModelAndView(view, name, value);

Map model = ...
model.put(name, value);
new ModelAndView(view, model);

Le model permet de transmettre des beans complets à la vue

```
public class Personne {
    private String nom;
    String prenom;
    int age;
    int id;
    public int getId() {
            return id;
    }
    public void setId(int id) {
```

```
		this.id = id;
	}
	public String getNom() {
		return nom;
	}
	public void setNom(String nom) {
		this.nom = nom;
	}
	public String getPrenom() {
		return prenom;
	}
	public void setPrenom(String prenom) {
		this.prenom = prenom;
	}
	public int getAge() {
		return age;
	}
	public void setAge(int age) {
		this.age = age;
	}

	Personne ()
	{
		nom="VIDE";
		prenom="vide";
		age = -1;
	}
	public String getAll ()
	{
		return String.format("%s %s %d", nom, prenom, age);
	}
}
```

Côté Model

```
Personne p= new Personne ();
m.addAttribute("lapersonne",  p);
return "views/page1";
```

Pour le rendu

```
<br/>${lapersonne.all}
```

### *paramètres de GET et POST*

Le modèle de code de Servlet n'est pas loin ; on peut manipuler les objets tels que
- HttpServlêtrequest
- HttpServlêtresponse

Le client demande

http://localhost:8080/Exo9_3/page/2?id=1

Le contrôleur recoit et conduit à la view page1.jsp ou page2.jsp

```
@RequestMapping(value="/page/2")
public String getData(@RequestParam String id) {
        System.out.println ("Le Controlleur page 1, /page/2");
        return "views/page"+id;
}
```

Accès aux paramètres de l'URL

```
public String getData2(Model m, HttpServlêtrequest request) {
        System.out.println ("Le Controlleur page 1, /page/4");
        Enumeration<String> p = request.getParameterNames();
        while (p.hasMoreElements()) {
                        String s = (String)p.nextElement();
                        System.out.println (s+":::"+request.getParameter(s));
        }
        return "views/page1";
}
```

Paramétrage détaillé avec valeur par défaut

```
@RequestMapping("/hello")
public ModelAndView showMessage(
                @RequestParam(value = "name", required = false, defaultValue = "World") String name) {
```

## 11. Programmation des aspects

### *Présentation*

L'AOP permet de facilement mettre en place des fonctionnalités dans différents points d'une application.

On définition une expression pointcuts qui va provoquer l'appel à un traitement appelé Advise. La relation entre les deux se faisant par un jointpoint : before, after, around ...

La mise en place de ces liens se nomme le tissage. Il y a plusieurs façons de faire le tissage
- A la compilation
- Au runtime

- Par configuration XML
- En utilisation des annotations / AspectJ

Spring AOP propose 5 types d'advices :
- before : le code de l'advice est exécuté avant l'exécution de la méthode. Il n'est pas possible d'inhiber l'invocation de la méthode sauf si une exception est levée dans l'advice
- after returning : le code de l'aspect est exécuté après l'exécution de la méthode qui renvoie une valeur de retour (aucune exception n'est levée)
- after throwing : le code de l'aspect est exécuté lorsqu'une exception est levée suite à l'invocation de la méthode
- after : le code de l'aspect est exécuté après l'exécution de la méthode, même si une exception
- est levée.
- around : le code de l'aspect permet de lancer l'exécution de la méthode et ainsi de réaliser
- des traitements avant, pour par exemple conditionner l'invocation de la méthode et des traitements après

### *Gestion des aspects dans une classe @Aspect avec @Before, @After et @Around*

Définition d'un bean, sur lequel

```
public class Bonjour {
  private String message;

  public void setMessage(String message){
    this.message  = message;
  }

  public String getMessage(){
    return "le Message : " + message;
  }
}
```

Définition des aspects

```
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class BonjourAspect {


    @Before("execution(public String getMessage())")
    public void avant(){
            System.out.println("before BonjourAspect  : Executing Advice on getMessage()");
```

```
	}
	@After("execution(public String getMessage())")
	public void apres(){
		System.out.println("After BonjourAspect  : Executing Advice on getMessage()");
	}
	@Around("execution(public String getMessage())")
	public Object pendant(ProceedingJoinPoint proceedingJoinPoint) throws Throwable{
		Bonjour b = (Bonjour) proceedingJoinPoint.getThis();
		b.setMessage("AAAAAAAAAAAAAAAAAAAA");
		System.out.println(proceedingJoinPoint.getThis().getClass());
		Object value = proceedingJoinPoint.proceed();
		System.out.println("Around BonjourAspect  : Executing Advice on getMessage()");
		return value;
	}
}
```

Il est également possible de mettre en place plus d'un traitement dans le @before par exemple

```
@Before("execution(public String getMessage())")
public void avant(){
	System.out.println("before BonjourAspect  : Executing Advice on getMessage()");
}
@Before("execution(public String getMessage())")
public void avant2(){
	System.out.println("before 2 BonjourAspect  : Executing Advice on getMessage()");
}
```

Fichier de configuration des beans

```
<aop:aspectj-autoproxy />

<!-- Configure Bonjour Bean and initialize it -->
<bean name="bonjour" class="formation.Bonjour">
   <property name="message" value="The message"></property>
</bean>

<!-- Configure Aspect Beans, without this Aspects advices wont execute -->
<bean name="bonjourAspect" class="formation.BonjourAspect" />
```

Main

```
public static void main(String[] args) {
  ClassPathXmlApplicationContext context =  new ClassPathXmlApplicationContext("Spring.xml");
  Bonjour bonjour = context.getBean("bonjour", Bonjour.class);
```

```
    System.out.println(bonjour.getMessage());
    context.close();
  }
```

### *Gestion des aspects avec une classe et une configuration XML*

Un bean de configuration est contient les méthodes After,Before et around

```
public class BonjourXMLConfigAspect {

   public Object bonjourAroundAdvice(ProceedingJoinPoint proceedingJoinPoint){
      System.out.println("BonjourXMLConfigAspect: Before l'invocation de getMessage()");
      Object value = null;
      try {
         value = proceedingJoinPoint.proceed();
      } catch (Throwable e) {
         e.printStackTrace();
      }
      System.out.println("BonjourXMLConfigAspect: After l'invocation de getMessage()");
      System.out.println("BonjourXMLConfigAspect: Retour de getMessage() = "+value);
      return value;
   }
}
```

Dans le fichier de configuration XML on déclare les connexion de l'AOP

```
<bean name="bonjourXMLConfigAspect" class="formation.BonjourXMLConfigAspect" />
<aop:config>
   <aop:aspect ref="bonjourXMLConfigAspect" id="bonjourXMLConfigAspectID" order="1">
      <aop:pointcut expression="execution(* formation.Bonjour.getMessage())" id="lepointcut"/>
      <aop:around method="bonjourAroundAdvice" pointcut-ref="lepointcut" arg-
names="proceedingJoinPoint"/>
   </aop:aspect>
</aop:config>
```

## 12. Securité Spring

### *Présentation*

La sécurité sous Java est faite de Java 2 security et le module JAAS pour les plateforme de type JEE
La sécurité parle de :

- Authentification
- Intégrité
- Confidentialité / chiffrement

- SSO
- realm
- LDAP

Dans les applications web, l'authentification peut se faire :
- Au niveau du serveur Http en rejoignant un d'authentification
- Au niveau de l'application

Pour ce concerne Spring, le module spring-security-web-3.2.5 constitue une aide

### *Mise en place*

http://mvnrepository.com/artifact/org.springframework.security/spring-security-web/3.2.5.RELEASE
spring-security-web-3.2.5.RELEASE.jar

avec Maven

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>${spring.security.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>${spring.security.version}</version>
</dependency>
```

fichier : spring-security.xml

```
<beans:beans xmlns="http://www.springframework.org/schema/security"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security-3.2.xsd">

    // admin doit être autorisé
    <http auto-config="true">
            <intercept-url pattern="/admin**" access="ROLE_USER" />
    </http>

    <authentication-manager>
      <authentication-provider>
```

```
        <user-service>
            <user name="karim" password="123456" authorities="ROLE_ADMIN" />
            <user name="julien" password="123456" authorities="ROLE_USER" />
        </user-service>
      </authentication-provider>
    </authentication-manager>
</beans:beans>
```

Interception d'URL pour conduire à l'authentification
web.xml

Interception d'URL

```
...
<!-- Spring MVC -->
<servlet>
        <servlet-name>mvc-dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>mvc-dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
</servlet-mapping>

<listener>
        // Iterception d'URL
        <listener-class>org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>

 <!-- Loads Spring Security config file -->
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
                /WEB-INF/spring-security.xml
        </param-value>
</context-param>
….
```

Formulaire d'authentifucation

Authentification avec formulaire

```
spring-security.xml
....
<http auto-config="true">
        <intercept-url pattern="/admin**" access="ROLE_USER" />
        <form-login
          login-page="/login"
          default-target-url="/welcome"
            authentication-failure-url="/login?error"
            username-parameter="username"
            password-parameter="password" />
        <logout logout-success-url="/login?logout" />
        <!-- enable csrf protection -->
        <csrf/>
    </http>

    <authentication-manager>
        <authentication-provider>
          <user-service>
            <user name="mkyong" password="123456" authorities="ROLE_USER" />
          </user-service>
        </authentication-provider>
    </authentication-manager>
....
```

authentification basic

```
<http auto-config='true'>
  <intercept-url pattern="/**" access="ROLE_USER" />
  <http-basic />
</http>
```

Authentification avec formulaire

```
<http>
    <intercept-url pattern='/login.htm*' filters='none'/>
    <intercept-url pattern='/**' access='ROLE_USER' />
    <form-login login-page='/login.htm' default-target-url='/home.htm'
            always-use-default-target='true' />
</http>
```

Authentification avec datasource

```
<authentication-manager>
<authentication-provider>
```

```
    <jdbc-user-service data-source-ref="securityDataSource"/>
  </authentication-provider>
  </authentication-manager>
```

L'authentification par JAAS (Authentification et Autorisation ) permet de rejoindre le modèle standard de la JEE.

JASS dispose d'une librairie de beans pour répondre à la plupart de demandes en authentification. De plus les étapes d'authentifications sont empilables à volonté .

applicationContext

```
    <sec:http auto-config="true" use-expressions="true">
      <sec:intercept-url pattern="/private/admin/**" access="hasRole('ADMIN')" />
      <sec:intercept-url pattern="/private/customer/**" access="hasRole('CUSTOMER')" />
      <sec:form-login login-page="/login.jsp" authentication-failure-url="/login.jsp?error=1" />
      <sec:logout logout-success-url="/home.jsp" logout-url="/j_spring_security_logout" />
    </sec:http>

    <sec:authentication-manager>
      <sec:authentication-provider ref="jaasAuthProvider" />
    </sec:authentication-manager>

    <bean id="jaasAuthProvider"
      class="org.springframework.security.authentication.jaas.DefaultJaasAuthenticationProvider">
      <property name="configuration">
        <bean
          class="org.springframework.security.authentication.jaas.memory.InMemoryConfiguration">
          <constructor-arg>
            <map>
              <entry key="SPRINGSECURITY">
                <array>
                  <bean class="javax.security.auth.login.AppConfigurationEntry">
                    <constructor-arg value="it.springwebjaas.Login" />
                    <constructor-arg>
                      <util:constant
                        static-
field="javax.security.auth.login.AppConfigurationEntry$LoginModuleControlFlag.REQUIRED" />
                    </constructor-arg>
                    <constructor-arg>
                      <map></map>
                    </constructor-arg>
                  </bean>
```

```
                    </array>
                </entry>
            </map>
        </constructor-arg>
    </bean>
</property>
<property name="authorityGranters">
    <list>
        <bean class="it.springwebjaas.RoleGranter" />
    </list>
</property>
</bean>
```

Les annotations

```
<global-method-security secured-annotations="enabled" />
```

```
public interface BankService {

    @Secured("IS_AUTHENTICATED_ANONYMOUSLY")
    public Account readAccount(Long id);

    @Secured("IS_AUTHENTICATED_ANONYMOUSLY")
    public Account[] findAccounts();

    @Secured("ROLE_TELLER")
    public Account post(Account account, double amount);
}
```

# WebService

## Middleware SOAP vs REST

Le WS permet de créer des composant métiers exposés sur le WEB. Pour communiquer avec les WS il existe des protocoles utilisant des modèles persistances propres :

- SOAP  transporté sur http (entre autre) avec une persistance XML
- REST transporté sur http (seulement) avec une persistance JSON

Le protocole SOAP est une norme défendue par la W3C alors que REST ne l'est pas.

Dans SOAP le transport se fait sur HTTP ou SMTP au formation texte ou plutôt XML (enveloppes).

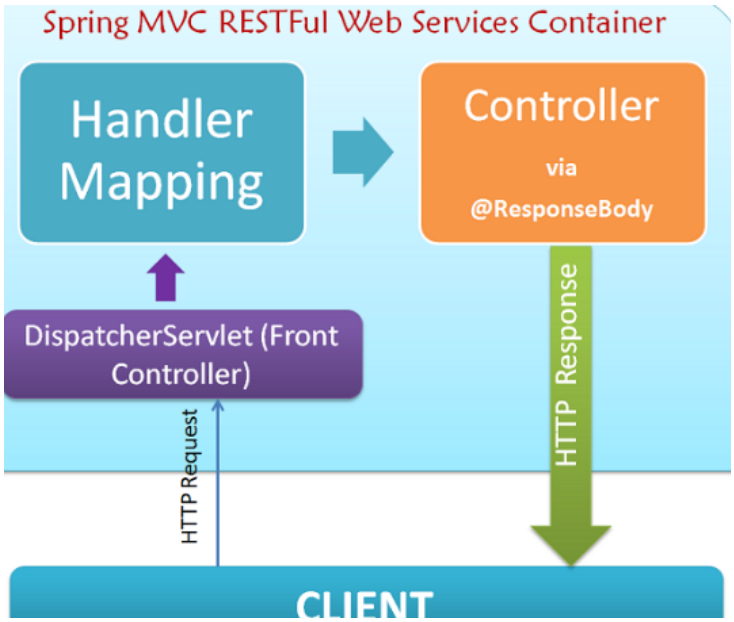
REST utilise spécialement HTTP et les échanges sont donc texte structuré, JSON est préféré mais pas XML.

Les deux implémentations sont proposée également dans la spécification JEE : JAX-WS, JAX-RS

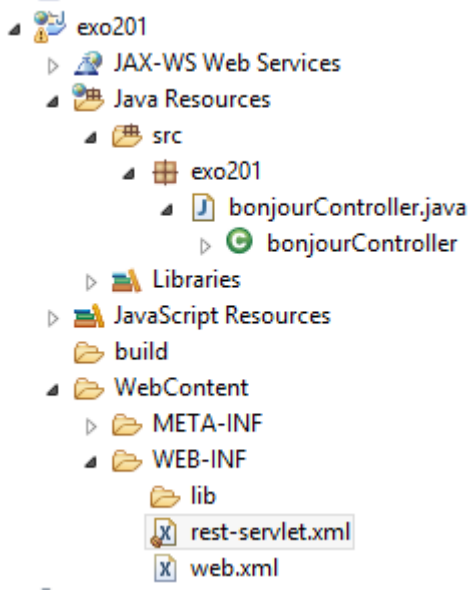
### *WS avec REST*

REST met en valeur les opérations directement implémentées par les verbes de HTTP : GET, PUT, ..

| | |
|---|---|
| Get | Permet de lire la ressource |
| Post | Créer une ressource |
| Put | Mettre à jour la ressource |
| Delete | Supprimer une ressource |



Modèle de projet Spring / Rest




Code du service REST : qui dit bonjour

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class bonjourController {

   private static final String message = "Bonjour à %s!";

   @RequestMapping("/bonjour")
   public String ditbonjour(@RequestParam(value="name", defaultValue="Karim") String name) {
      return String.format(message, name));
   }
}
```

Code du main

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

   public static void main(String[] args) {
```

```
        SpringApplication.run(Application.class, args);
    }
}
```

web.xml

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

<servlet-mapping>
    <servlet-name>rest</servlet-name>
    <url-pattern>/*</url-pattern>
</servlet-mapping>
</web-app>
```

rest-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    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
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">
    <mvc:annotation-driven/>
    <context:component-scan base-package="org.arpit.java2blog.controller" />
</beans>
```

### *Rest et les verbes HTTTP*

```
Controleur
@Controller
publicclass EmployeeController {
@RequestMapping(method=RequestMethod.GET, value="/employee/{id}")
public ModelAndView getEmployee(@PathVariable String id) {
```

```
        Employee e = employeeDS.get(Long.parseLong(id));
        returnnew ModelAndView(XML_VIEW_NAME, "object", e);
    }
    }

    @RequestMapping(method=RequestMethod.POST, value="/employee")
    public ModelAndView addEmployee(@RequestBody String body) {
        Source source = new StreamSource(new StringReader(body));
        Employee e = (Employee) jaxb2Mashaller.unmarshal(source);
        employeeDS.add(e);
        returnnew ModelAndView(XML_VIEW_NAME, "object", e);
    }

    @RequestMapping(method=RequestMethod.PUT, value="/employee/{id}")
    public ModelAndView updateEmployee(@RequestBody String body) {
        Source source = new StreamSource(new StringReader(body));
        Employee e = (Employee) jaxb2Mashaller.unmarshal(source);
        employeeDS.update(e);
        returnnew ModelAndView(XML_VIEW_NAME, "object", e);
    }

    @RequestMapping(method=RequestMethod.DELETE, value="/employee/{id}")
    public ModelAndView removeEmployee(@PathVariable String id) {
        employeeDS.remove(Long.parseLong(id));
        List<Employee> employees = employeeDS.getAll();
        EmployeeList list = new EmployeeList(employees);
        returnnew ModelAndView(XML_VIEW_NAME, "employees", list);
    }
```

Installation d'un

# Spring JMS

## 13. Introduction

Communiquer entre applications avec des messages, constitués, déposés ,livrés, acquittés, routés… Ceci est la communication par message ou MOM (MiddleWare Orienté Message). C'est une technologies qui offre un grand niveau d'intégration et de souplesse dans les échanges avec l'idée de l'asynchrone mais les échanges sont moins rapides qu'un protocole RMI. MOM reste toutefois plus rapide qu'un intégration de  type Web Service.

Les logiciels supportant l'API JMS sont nombreux :

- hornetQ (open source)
- ActiveQ (open source Apache)
- mqseries (IBM)
- msmQ (Micro Soft)
- sonicQ (open source)

Dans le JMS les échanges entre programmes ou applications se font sur des files d'attente nommées et connues des correspondants

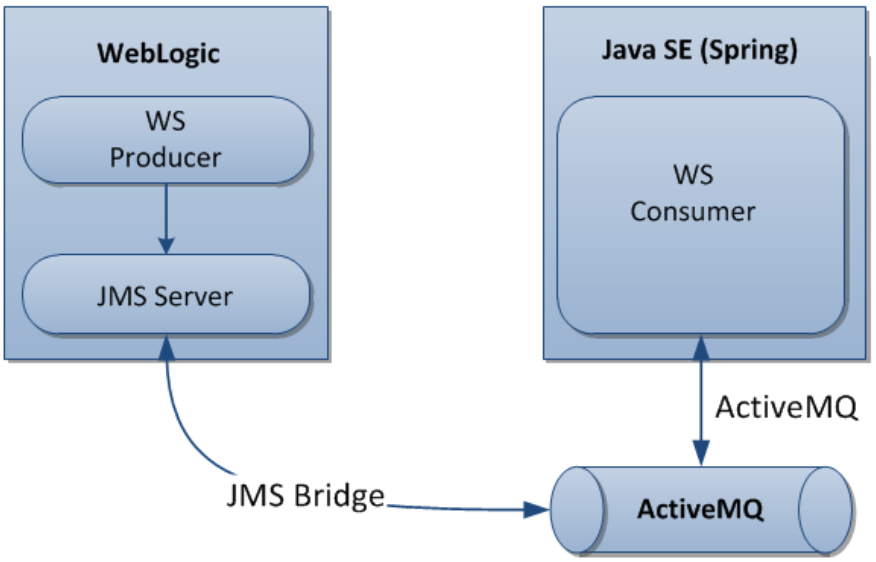
Spring propose JmsTemplate pour englober l'API, apportant un niveau de portabilité supplémentaire.

Les paquetages jar :

- org.springframework.jms.annotation
- org.springframework.jms.config
- org.springframework.jms.connection
- PlatformTransactionManager

API callback, classes et interfaces :

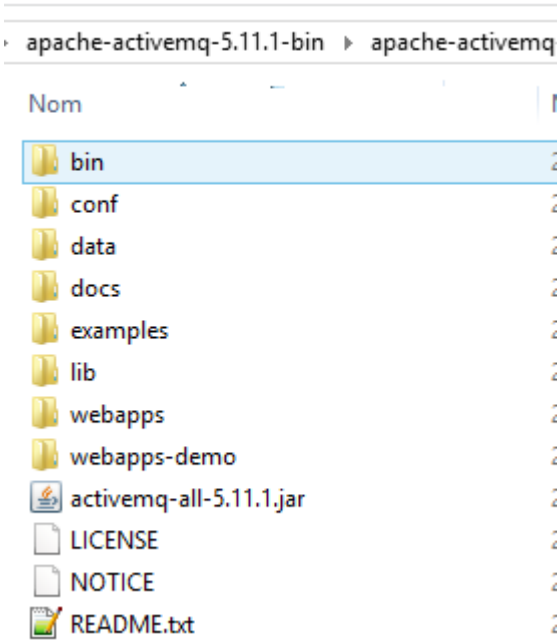
- JmsTemplate
- ProducerCallback
- MessageCreator
- MessageProducer
- Session




### *Installation d'un serveur JMS : ActiveQM*

Téléchargement du logiciel
http://activemq.apache.org/activemq-5130-release.html

Installation et lancement



Les queues sont bidirectionnelles, elles peuvent être créée à un processus ou par configuration.

Le clients se connectent à une file d'attente et font leurs échanges

Le producteur : peut envoyer du texte ou une personne serialisée

```
package formation;

import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.Session;

import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;

public class SpringJmsProducteur {
	private JmsTemplate jmsTemplate;
	private Destination destination;

	public JmsTemplate getJmsTemplate() {
		return jmsTemplate;
	}

	public void setJmsTemplate(JmsTemplate jmsTemplate) {
		this.jmsTemplate = jmsTemplate;
	}

	public Destination getDestination() {
		return destination;
	}

	public void setDestination(Destination destination) {
		this.destination = destination;
	}

	public void sendMessage(final String msg) {
		System.out.println("Le producteur envoie " + msg);
		jmsTemplate.send(destination, new MessageCreator() {
```

```
                        public Message createMessage(Session session) throws JMSException {
                                return session.createTextMessage(msg);
                        }});
        }
        public void sendObjet(final Personne p) {
                System.out.println("Le producteur envoie ");
                jmsTemplate.send(destination, new MessageCreator() {
                        public Message createMessage(Session session) throws JMSException {
                                return session.createObjectMessage(p);
                        }});
        }
}
```

Le consommateur

```
package formation;

import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.TextMessage;
import javax.jms.Message;

import org.springframework.jms.core.JmsTemplate;

public class SpringJmsConsommateur {
        private JmsTemplate jmsTemplate;
        private Destination destination;

        public JmsTemplate getJmsTemplate() {
                return jmsTemplate;
        }

        public void setJmsTemplate(JmsTemplate jmsTemplate) {
                this.jmsTemplate = jmsTemplate;
        }

        public Destination getDestination() {
                return destination;
        }

        public void setDestination(Destination destination) {
```

```
                this.destination = destination;
        }

        public String receiveMessage() throws JMSException {
                TextMessage textMessage = (TextMessage) jmsTemplate.receive(destination);
                return textMessage.getText();
        }

        public Personne receiveObjet() throws JMSException {
                Personne p = (Personne) jmsTemplate.receiveAndConvert(destination);
                return p;
        }
}
```

Programme principal

```
package formation;

import java.net.URI;
import java.net.URISyntaxException;

import org.apache.activemq.broker.BrokerFactory;
import org.apache.activemq.broker.BrokerService;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
        public static void main(String[] args) throws URISyntaxException, Exception {
                BrokerService broker = BrokerFactory.createBroker(new URI(
                                "broker:(tcp://localhost:61616)"));
                broker.start();
                ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(
                                "applicationContext.xml");

                try {
                SpringJmsProducteur p = (SpringJmsProducteur) context.getBean("springJmsProducteur");
                p.sendMessage("<<Message echangé>>");
                p.sendObjet(new Personne ());

                SpringJmsConsommateur c = (SpringJmsConsommateur)
context.getBean("springJmsConsommateur");
                System.out.println("Le consommateur recoit " + c.receiveMessage());
```

```
                //System.out.println("Le consommateur recoit " + c.receiveObjet().getAll());
                System.out.println(c.receiveObjet().getAll());
                } finally {
                broker.stop();
                context.close();
                }
        }
}
```

Objet Personne échangé

```
package formation;

import java.io.Serializable;

@SuppressWarnings("serial")
public class Personne implements Serializable {

   private int id;
   private String nom;
   private String prenom;
   private int age;
     public String getNom() {
                return nom;
     }
     public void setNom(String nom) {
                this.nom = nom;
     }
     public String getPrenom() {
                return prenom;
     }
     public void setPrenom(String prenom) {
                this.prenom = prenom;
     }
     public int getAge() {
                return age;
     }
     public void setAge(int age) {
                this.age = age;
     }
```

```
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public Personne ()
	{
		nom="VIDE";
		prenom="vide";
		age = -1;
	}
	String getAll ()
	{
		return String.format("%s %s %d", nom, prenom, age);
	}
}
```

Configuration de JMS

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

	<bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
		<property name="brokerURL" value="tcp://localhost:61616" />
	</bean>
	<bean id="messageDestination" class="org.apache.activemq.command.ActiveMQQueue">
		<constructor-arg value="messageQueue1" />
	</bean>
	<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
		<property name="connectionFactory" ref="connectionFactory" />
		<property name="receiveTimeout" value="10000" />
	</bean>

	<bean id="springJmsProducteur" class="formation.SpringJmsProducteur">
		<property name="destination" ref="messageDestination" />
		<property name="jmsTemplate" ref="jmsTemplate" />
	</bean>
```

```
    <bean id="springJmsConsommateur" class="formation.SpringJmsConsommateur">
        <property name="destination" ref="messageDestination" />
        <property name="jmsTemplate" ref="jmsTemplate" />
    </bean>
</beans>
```

Démarrage du serveur d'application ActiveMQ

```
INFO | JMX consoles can connect to
service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi
INFO | PListStore:[D:\formation\spring\logiciels\ateliers4\exo302_1\activemq-
data\localhost\tmp_storage] started
INFO | Using Persistence Adapter:
KahaDBPersistenceAdapter[D:\formation\spring\logiciels\ateliers4\exo302_1\act
ivemq-data\localhost\KahaDB]
INFO | KahaDB is version 6
INFO | Recovering from the journal @1:66407
INFO | Recovery replayed 1 operations from the journal in 0.022 seconds.
INFO | Apache ActiveMQ 5.12.0 (localhost, ID:keyos1-16033-1464729634871-0:1)
is starting
INFO | Listening for connections at: tcp://127.0.0.1:61616
INFO | Connector tcp://127.0.0.1:61616 started
INFO | Apache ActiveMQ 5.12.0 (localhost, ID:keyos1-16033-1464729634871-0:1)
started
INFO | For help or more information please see: http://activemq.apache.org
Le producteur envoie <<Message echangé>>
Le producteur envoie
Le consommateur recoit <<Message echangé>>
VIDE vide -1
INFO | Apache ActiveMQ 5.12.0 (localhost, ID:keyos1-16033-1464729634871-0:1)
is shutting down
INFO | Connector tcp://127.0.0.1:61616 stopped
INFO | PListStore:[D:\formation\spring\logiciels\ateliers4\exo302_1\activemq-
data\localhost\tmp_storage] stopped
INFO | Stopping async queue tasks
INFO | Stopping async topic tasks
INFO | Stopped KahaDB
INFO | Apache ActiveMQ 5.12.0 (localhost, ID:keyos1-16033-1464729634871-0:1)
uptime 1.895 seconds
INFO | Apache ActiveMQ 5.12.0 (localhost, ID:keyos1-16033-1464729634871-0:1)
is shutdown
```