Utilisation de jsf et spring

Version1.0

Julis kissangou

06/03/2011

Table des matières

Introduction

Le présent tutoriel a pour but d'illustrer le couplage des Framework **JSF** et **SPRING**, nous expliciterons dans les pages qui vont suivre comment ce lien est établi. Par ailleurs nous parlerons brièvement de **SPRING** et du principe d'injection de dépendance. Pour les références à **JSF** nous vous renvoyons à notre tutoriel **JSF (introduction à JSF)**.

Et si on parlait de SPRING ?

Dans le monde **java EE** (**j2EE**) nous entendons régulièrement parler du **FRAMWORK SPRING**, autour de celui-ci est souvent évoqué le concept de **conteneur léger**, les principes **d'inversion de contrôle et d'injection de dépendance.** Ce **Framework** de plus en plus utilisé doit par ailleurs son succès par l'intégration en son sein des modules se rapportant à plusieurs problématiques du développement et de l'utilisation des applications et des systèmes d'information.

SPRING est Framework permettant de concevoir une infrastructure d'une application JAVA, il appartient par ailleurs à la famille des conteneurs légers, car les composants qu'il héberge n'ont pas besoin de d'implémenter une interface pour être pris en compte lors de sa mise en œuvre, il n y a donc pas d'impact sur ces derniers.

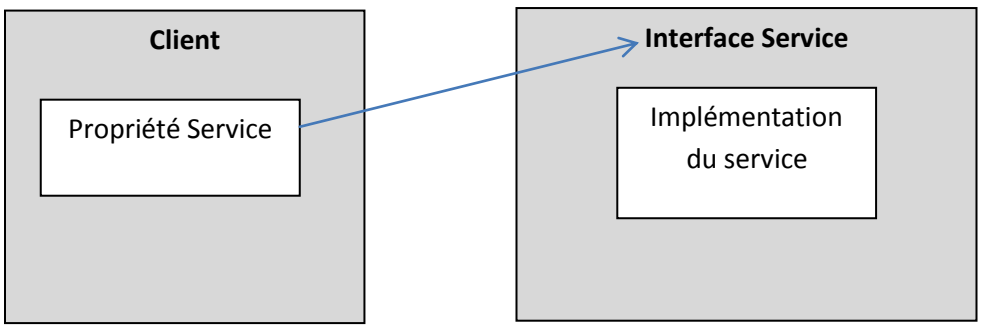
SPRING s'appuie sur trois concepts fondamentaux (**l'inversion de contrôle**, **la programmation orientée aspect** et **la couche d'abstraction**).

Dans le cadre de notre tutoriel nous parlerons uniquement du principe d'inversion de contrôle dans la gestion de composants (**injection de dépendance**), pour les autres concepts nous demandant de vous orienter vers les livres de références.

Principe d'inversion de contrôle (injection de dépendance)

L'inversion de contrôle appliquée à la gestion des composants est appelée **injection de dépendance**, ce principe est un patron de conception qui consiste à externaliser le flot d'exécution logiciel d'une application.
Supposant que nous utilisons un programme, celui-ci est client d'un service enveloppé dans une interface, quand l'instanciation d'une propriété de type service au sein du dit programme est déléguée à une application externe (un conteneur léger en l'occurrence), nous utilisons alors l'injection de dépendance.

Voici une illustration



Dans l'illustration précédente l'instanciation de la propriété sera confiée au conteneur léger Spring, nous confions donc la gestion de ce composant au conteneur évitant ainsi une dépendance avec la classe implémentant le service.

Pour renforcer la compréhension de ce principe nous allons développer une petite application qui concatène le un nom et prénom.

Nous allons pour commencer définir notre premier « Spring bean », une classe nommée **Personne** comportant deux propriétés nom et prenom, puis une autre nommée **ConcatNomPrenom** implémentant une interface permettant de rendre ce service, celle-ci sera nommée **IConcatNomPrenom**, et une dernière nommée **UseServiceConcatNomPrenom** pour l'utilisation du service**.**

Voici ci-dessous le code :

```
/** CLASSE Personne**/
Public class Personne
{
      private String nom ;
      private String prenom ;

      public Personne(String nom,String prenom)
      { this.nom=nom;
        this.prenom=prenom;
      }
      public String getNom()
      {
            return nom;
      }

      public String getPrenom() {
            return prenom;
      }

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

      public void setPrenom(String prenom) {
            this.prenom = prenom;
      }


}

/** INTERFACE IConcatNomPrenom**/
public interface IConcatNomPrenom {
      public String concat();
```

```
}
```

```
/**CLASSE ConcatNomPrenom**/
public class ConcatNomPrenom implements IConcatNomPrenom{


	public ConcatNomPrenom() {


	}

	@Override
	public String concat(String nom,String prenom) {

		return nom+prenom;
	}

}
```

```
/** CLASSE UseServiceConcatNomPrenom **/
public class UseServiceConcatNomPrenom {

	IConcatNomPrenom concat;//Cette propriété sera instanciée dans le
                                //conteneur SPRING

	Personne personne;//cette propriété sera instanciée dans le conteneur
                          //SPRING

	public UseServiceConcatNomPrenom( Personne personne,
                                          IConcatNomPrenom concat
                                        )
	 {
		this.concat=concat;
		this.personne=personne;

	}

	public void printConcatNomPrenom(){

		System.out.println(concat.concat(personne.getNom(),
                                                 personne.getPrenom()
                                   );
	}



}
```

Il n y a donc pas de dépendance entre la classe **`UseServiceConcatNomPrenom`** et la classe **`ConcatNomPrenom`** et nous remarquons par ailleurs que la gestion de des composant est confiée à SPRING.

Après définition de ces « Spring Bean » dans le conteneur léger via **un fichier de configuration XML** (nous y reviendrons plus tard), nous aurions créé, **dans un contexte hors d'une application web**, une classe (nous l'appellerons ConcatAssembler) et utilisé le code qui précède comme suit :

import org.springframwork.context.support.ClassPathXmlApplicationContext ;

public class ConcatAssembler{

public void static main(String [] args) {

ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext(''fichierConf.xml'') **;//on initialise le conteneur léger via le fichier de configuration contenant les définitions des beans**

UseServiceConcatNomPrenom concatservice = (UseServiceConcatNomPrenom)ctx.getBean("useServiceConcat");//onrécupère l'identifiant du bean de type UseServiceConcatNomPrenom défini dans le fichier de configuration

concatservice.printConcatNomPrenom() **;//on fait appel au service via la fonction printConcatNomPrenom**

}

}

Le fichier de configuration doit être placé dans un dossier source

REMARQUE :

Les implémentations respectant ce principe sont faciles à tester, à maintenir et à comprendre

Couplage JSF et SPRING

Le couplage de JSF et SPRING permet de tirer profit des apports de ces différents Framework de plus en plus utilisés dans les applications java EE. JSF intervient pour une plus grande productivité dans la conception d'interfaces graphiques et SPRING pour son infrastructure permettant de disposer des applications reproduisant les mêmes services que ceux des EJB sans faire usage des conteneurs lourds ou serveurs d'application J2EE.

Dans cette section il sera donc question de concevoir une interface web via **JSF** et de faire appel aux beans définis dans un **fichier de configuration Spring** (fichier XML) via les expressions langages(**EL**) de **JSF** (voir le tutoriel de JSF pour plus d'infos sur les **EL**).

Installation de l'environnement de travail

Nous ne nous étalerons pas sur l'installation se rapportant à JSF, vous pouvez vous référer au tutoriel sur JSF (introduction à JSF).

L'archive de base nécessaire à l'utilisation de SPRING est téléchargeable à l'adresse suivante :

http://www.springsource.com/products/spring-community-download

Nous avons pour notre tutoriel utiliser l'archive suivante **spring-framework-3.0.5.RELEASE-with-docs.**

Une fois téléchargée, décompressez la et allez dans répertoire **dist,** vous y trouverez les fichiers **jar** nécessaire à notre tutoriel :

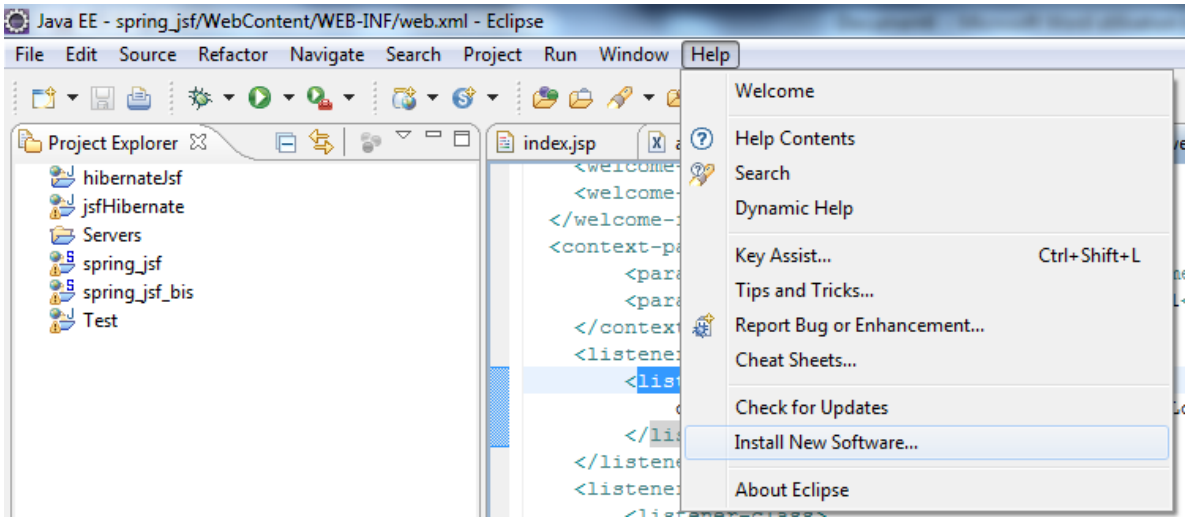
1. **org.springframework.asm-3.0.1.RELEASE-A**
2. **org.springframework.beans-3.0.1.RELEASE-A**
3. **org.springframework.context-3.0.1.RELEASE-A**
4. **org.springframework.core-3.0.1.RELEASE-A**
5. **org.springframework.expression-3.0.1.RELEASE-A**
6. **org.springframework.web-3.0.1.RELEASE-A**

En plus de ces fichiers nous aurons aussi besoin des fichiers suivants, en effet Spring s'appuie sur d'autres librairies issues du projet **Jakarta Commons** :

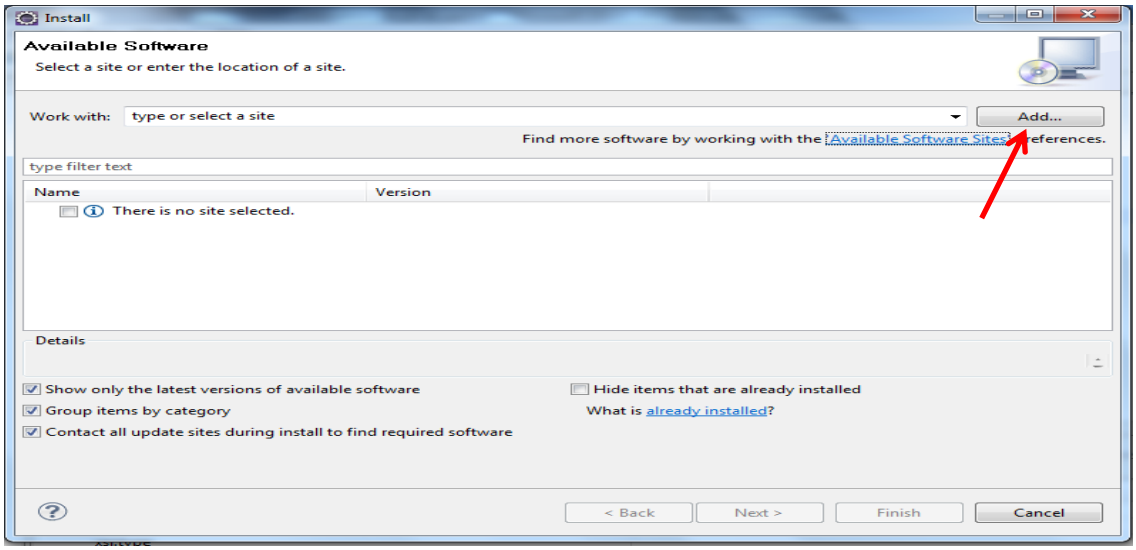
commons-lang-2.4, commons-logging-1.1.1.

Toutes les librairies seront installées dans le répertoire **WebContent/WEB-INF/lib** du projet qui sera créé

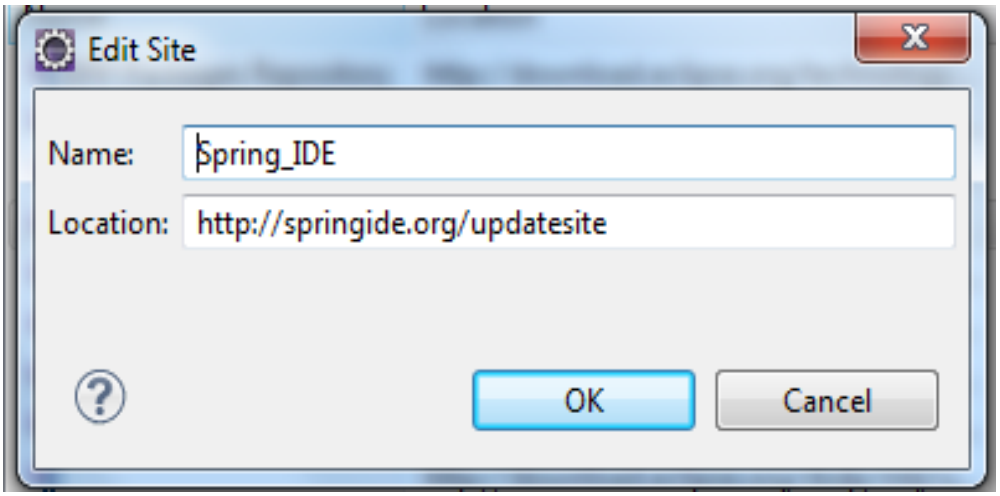
Pour le confort du travail dans éclipse il sera nécessaire aussi d'utiliser les extensions correspondant au projet **SPRING IDE,** pour le faire exécutez éclipse et allez dans l'onglet **help**, puis choisissez **Install New Software**.



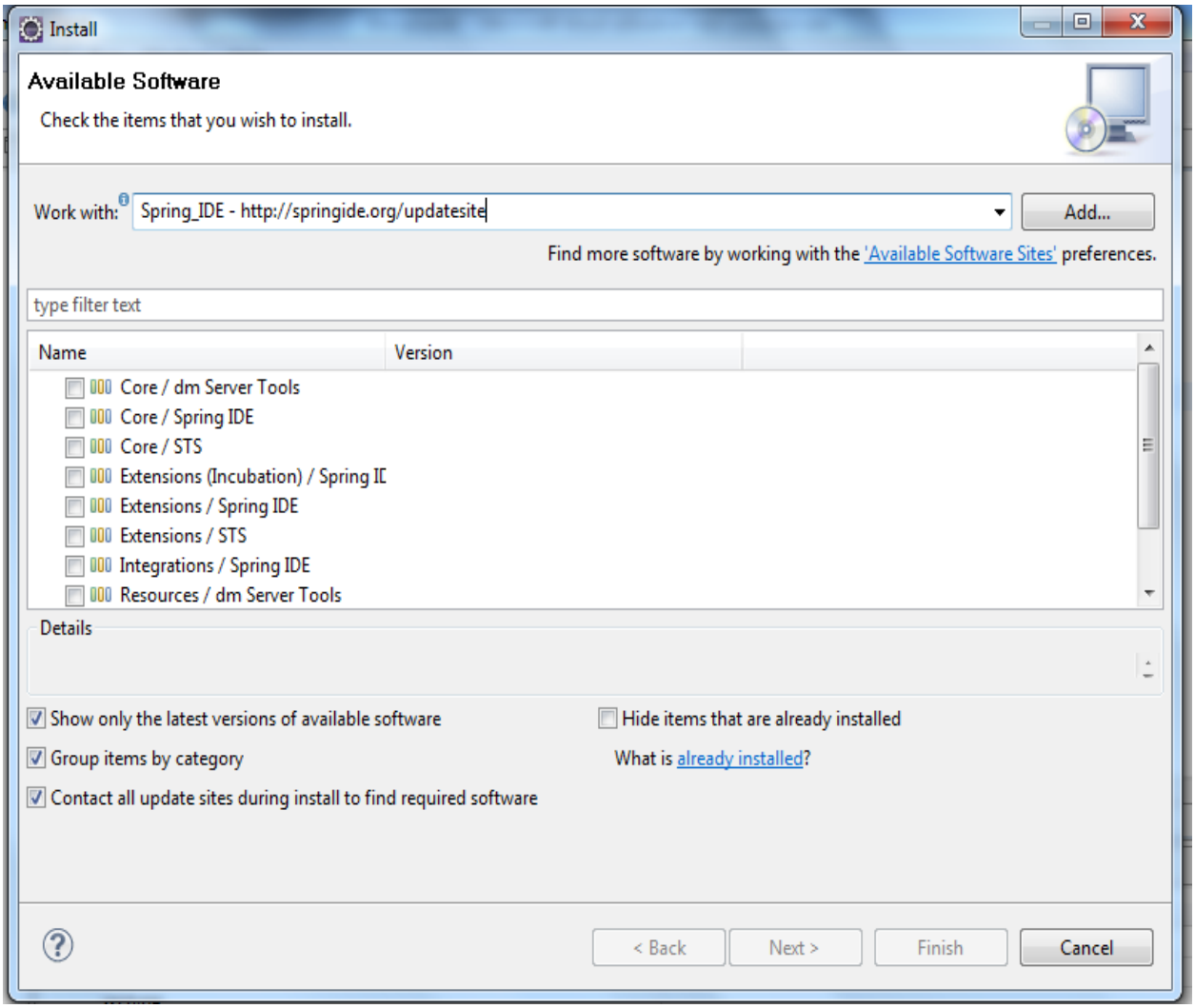
Dans la nouvelle fenêtre cliquez sur **add.**



Dans la nouvelle fenêtre entrez dans le champ **Name** une description que vous désirez **(**par exemple **Extension Spring IDE**) et dans le champ **Location** entrez l'URL suivante **http://springide.org/updatesite** puis cliquez sur **ok**.



L'**IDE** éclipse se mettra à rechercher les modules d'extensions, ceux-ci apparaitront dans la fenêtre principale de mise à jour**.**

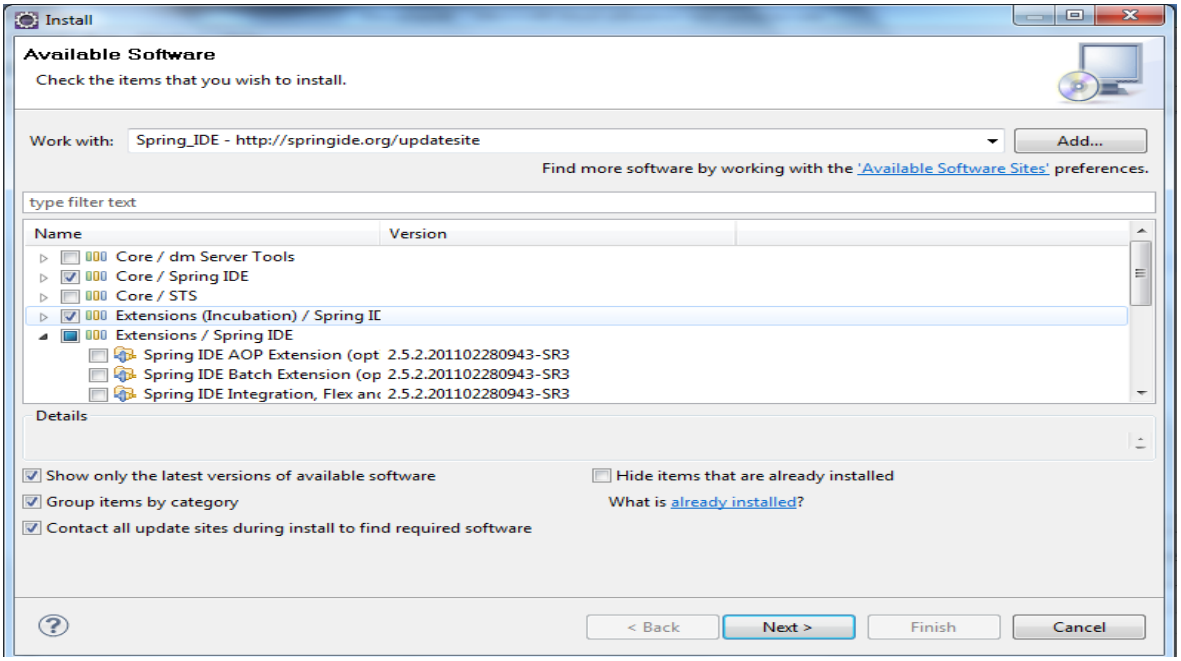


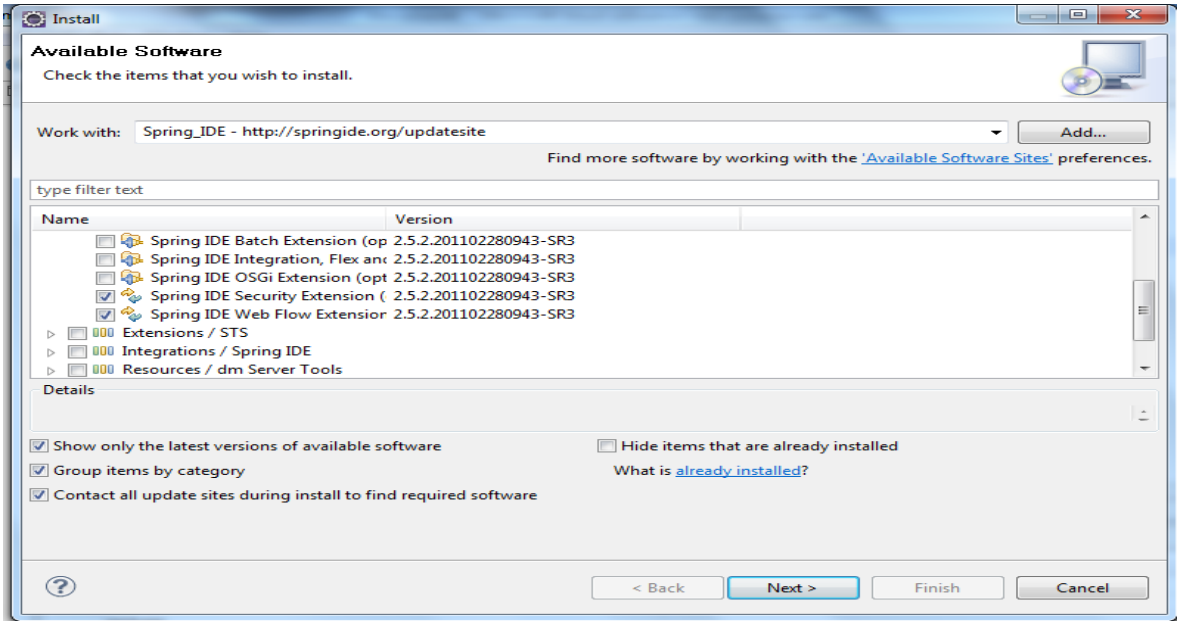
Il sera à vous de cocher les modules qui vous intéressent puis de cliquer sur **Next** et de laisser éclipse vous guider pour leur installation.

Pour notre tutoriel nous avons cochez les modules suivants :

1. Core/ Spring IDE
2. Extension (incubation)/Spring II
3. Spring IDE Security Extension (dans la case à cocher **Extension Spring IDE**)
4. Spring IDE Web Flow Extension (dans la case à cocher **Extension Spring IDE**)

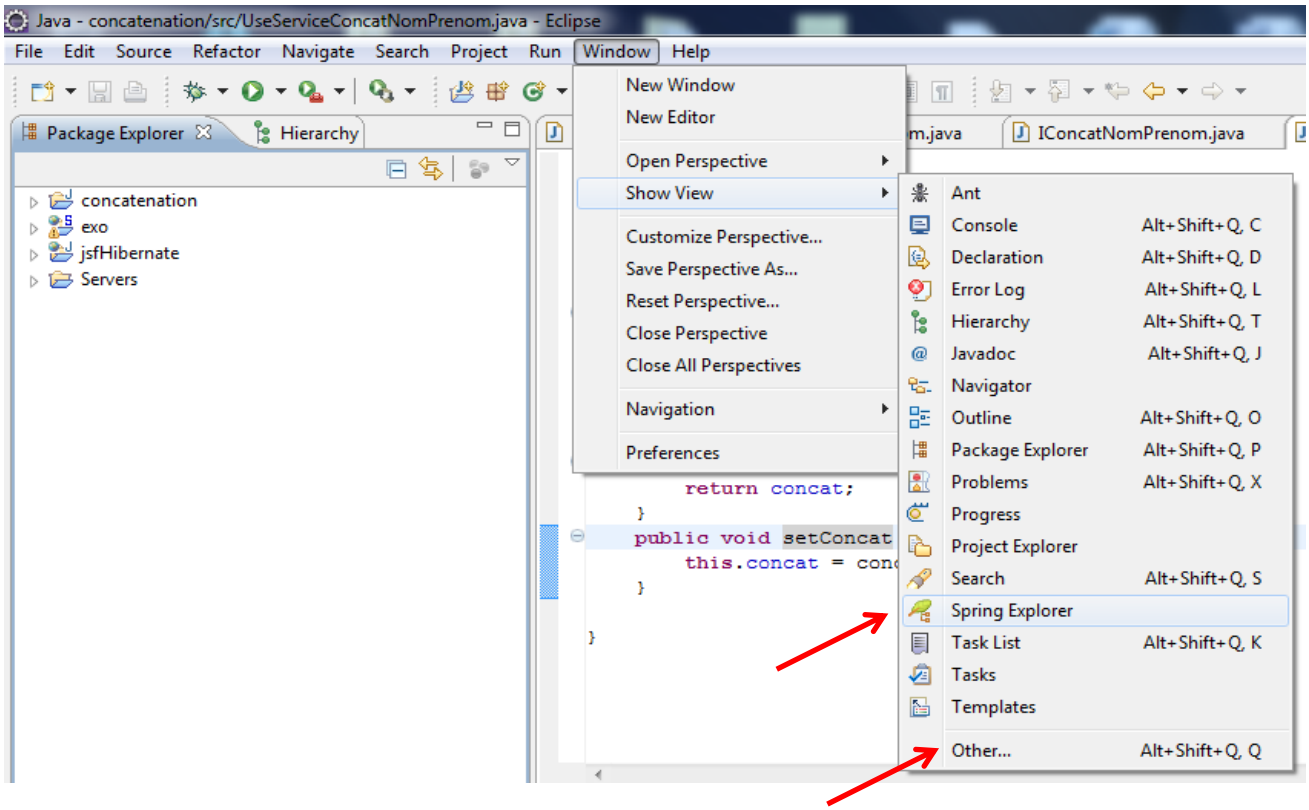
Ci-dessous une illustration





Apres avoir effectué l'installation, pour vous rendre compte qu'elle a été prise en compte allez dans l'onglet **Windows**, puis sélectionnez le champ **Show view** , vous découvrirez qu'un nouveau champ a été ajouté **Spring Explorer,** à défaut vous pourriez cliquez dans **other** et voir dans le répertoire **Spring** ce nouveau champ.

Ci-dessous une illustration

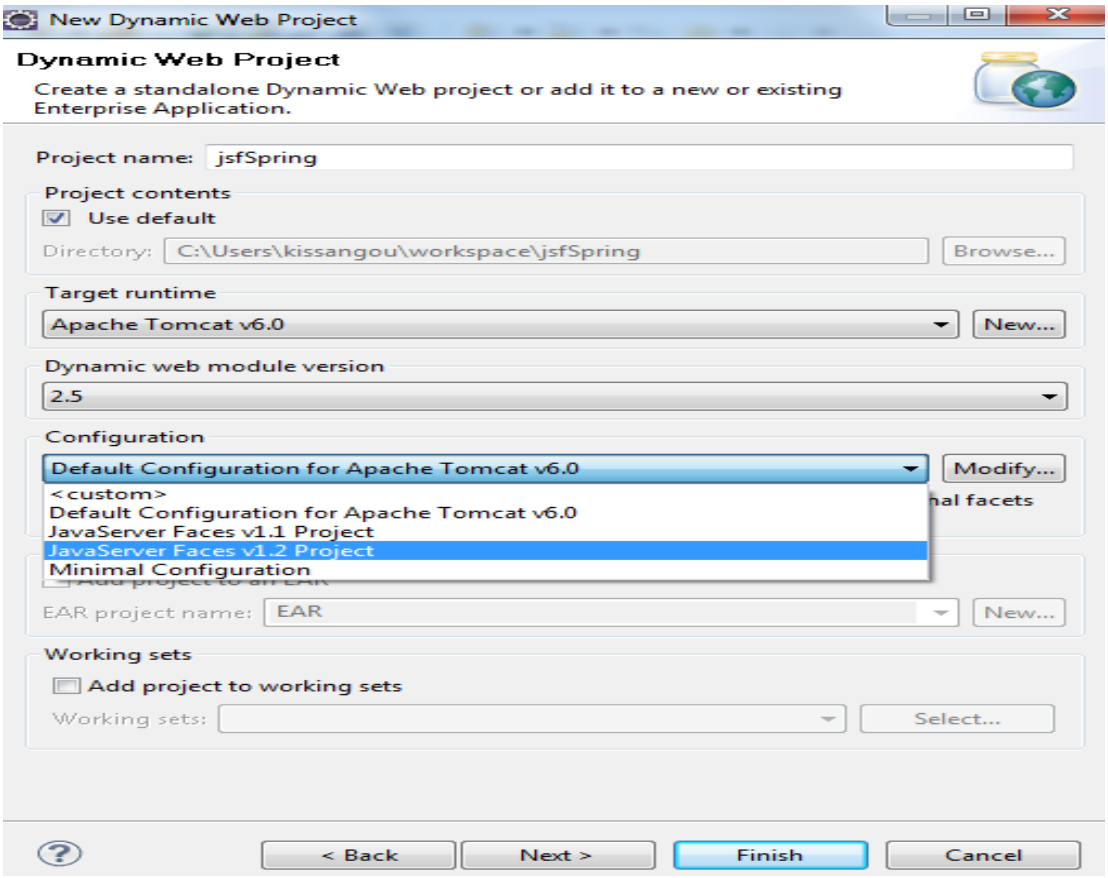


Configuration du couplage JSF et SPRING

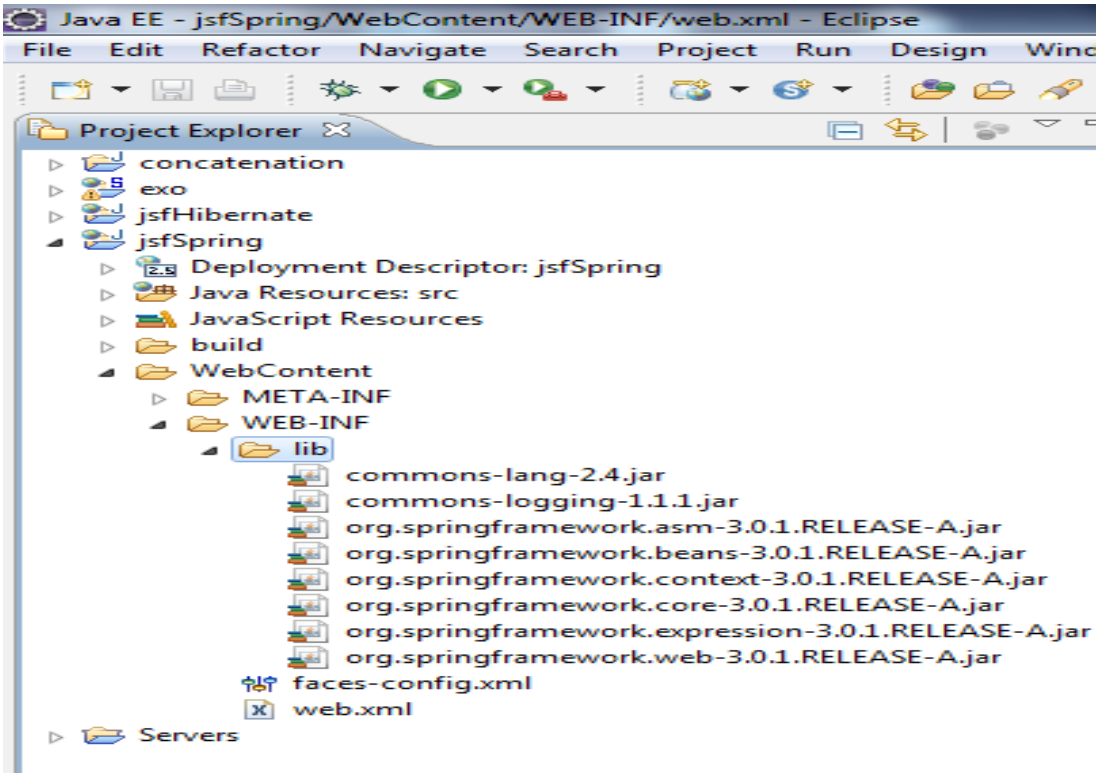
Ce couplage est rendu effectif grâce à la possibilité de SPRING d'intégrer d'autres Framework, cela est matérialisé par configuration du conteneur Spring(le contexte d'application web) dans le descripteur de déploiement (fichier **web.xml**), la création d'un fichier de configuration de Spring(**applicationContext.xml** en l'occurrence) contenant les beans et de l'ajout ou la définition d'un résolveur(classe java définie dans Spring permettant de rendre possible l'accès aux beans Spring via les **EL** de **JSF**) dans le fichier de configuration de JSF(**faces-config.xml**).

Nous allons commencer par créer un projet web dynamique dans l'atelier éclipse prenant en compte **JSF** dans sa **version 1.2** (**nous supposons que les librairies de JSF sont dans le répertoire lib de Tomcat**), ce projet sera nommé **jsfSpring**.

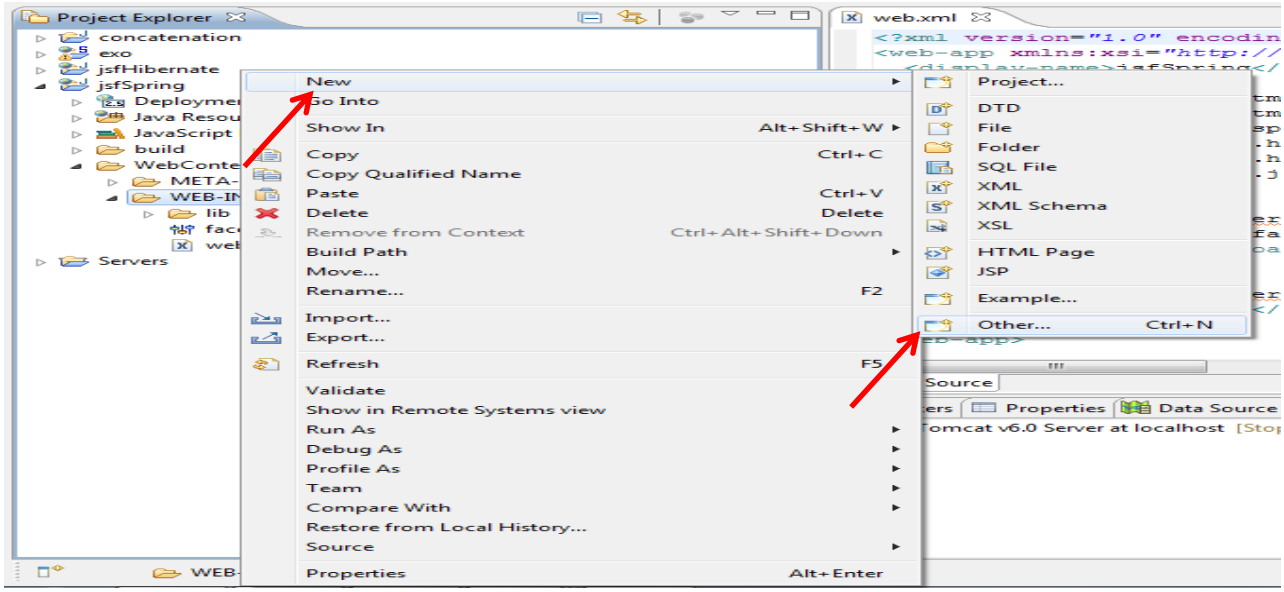
Voir illustration



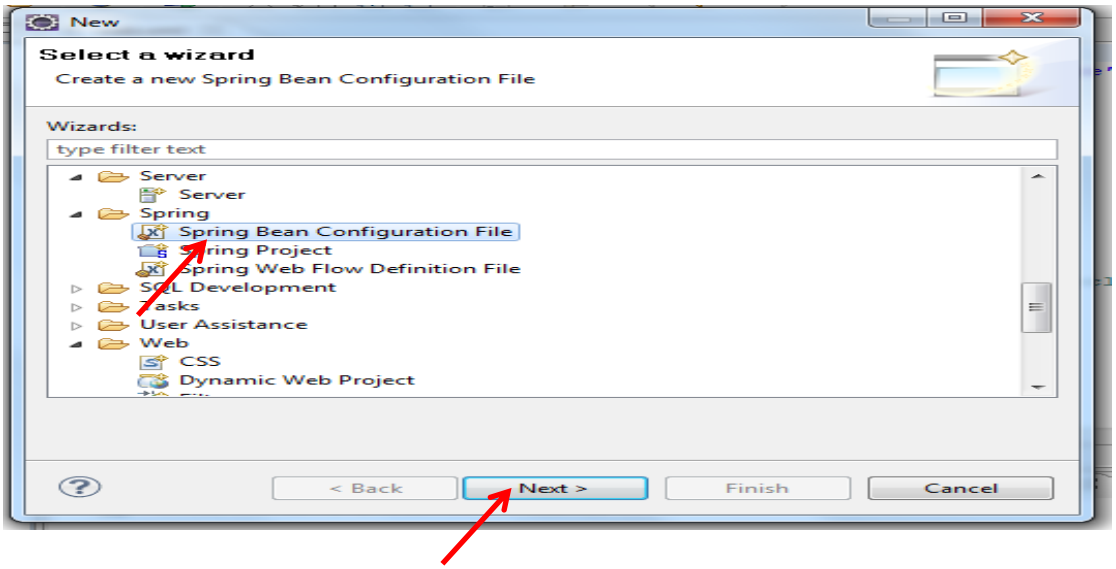
Par la suite importons nos librairies Spring dans le répertoire **/WEB-INF/lib**



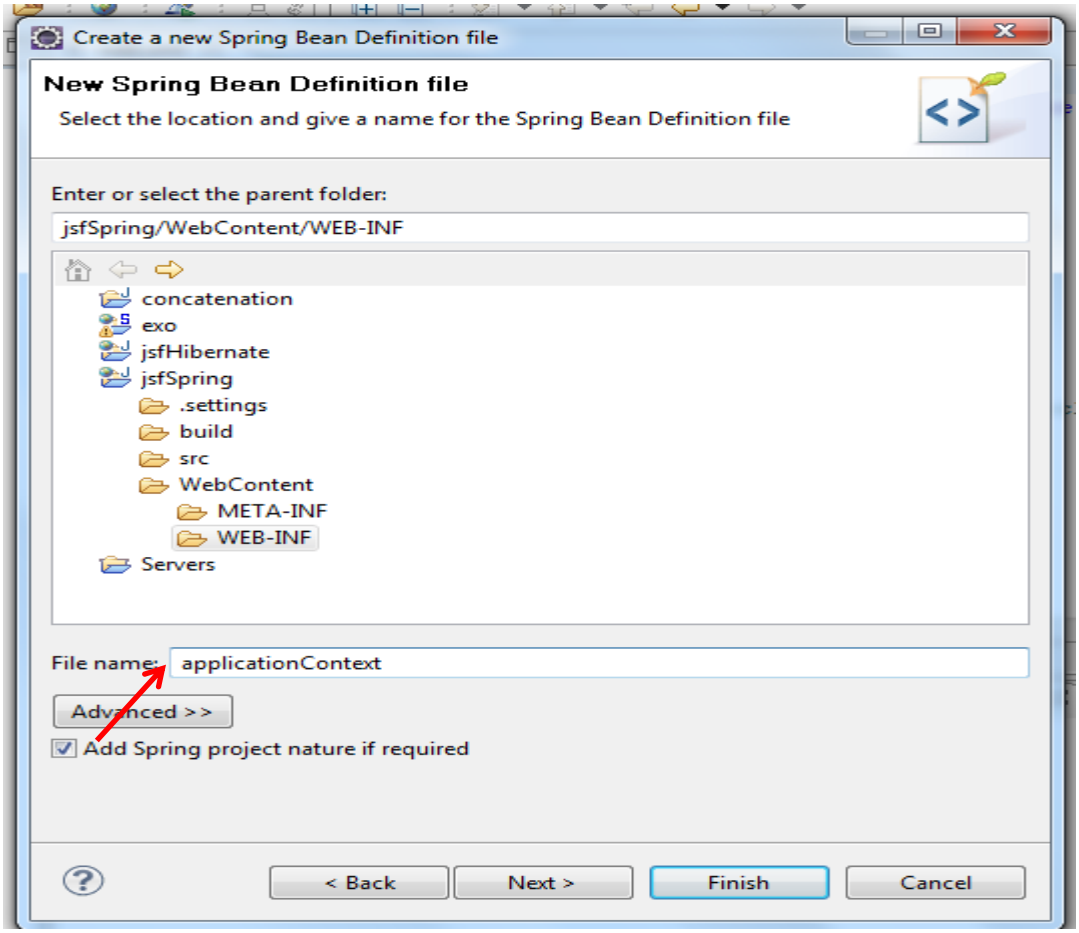
Créons un fichier de configuration Spring **applicationContext.xml** dans le répertoire **/WEB-INF** pour le faire faites un clic droit de souris sur le répertoire **WEB-INF**, allez sur le champ **New**, cliquez sur **other**



Dans la nouvelle fenêtre allez dans le répertoire **Spring**, sélectionnez **Spring bean Configuration File** et cliquez **sur Next**

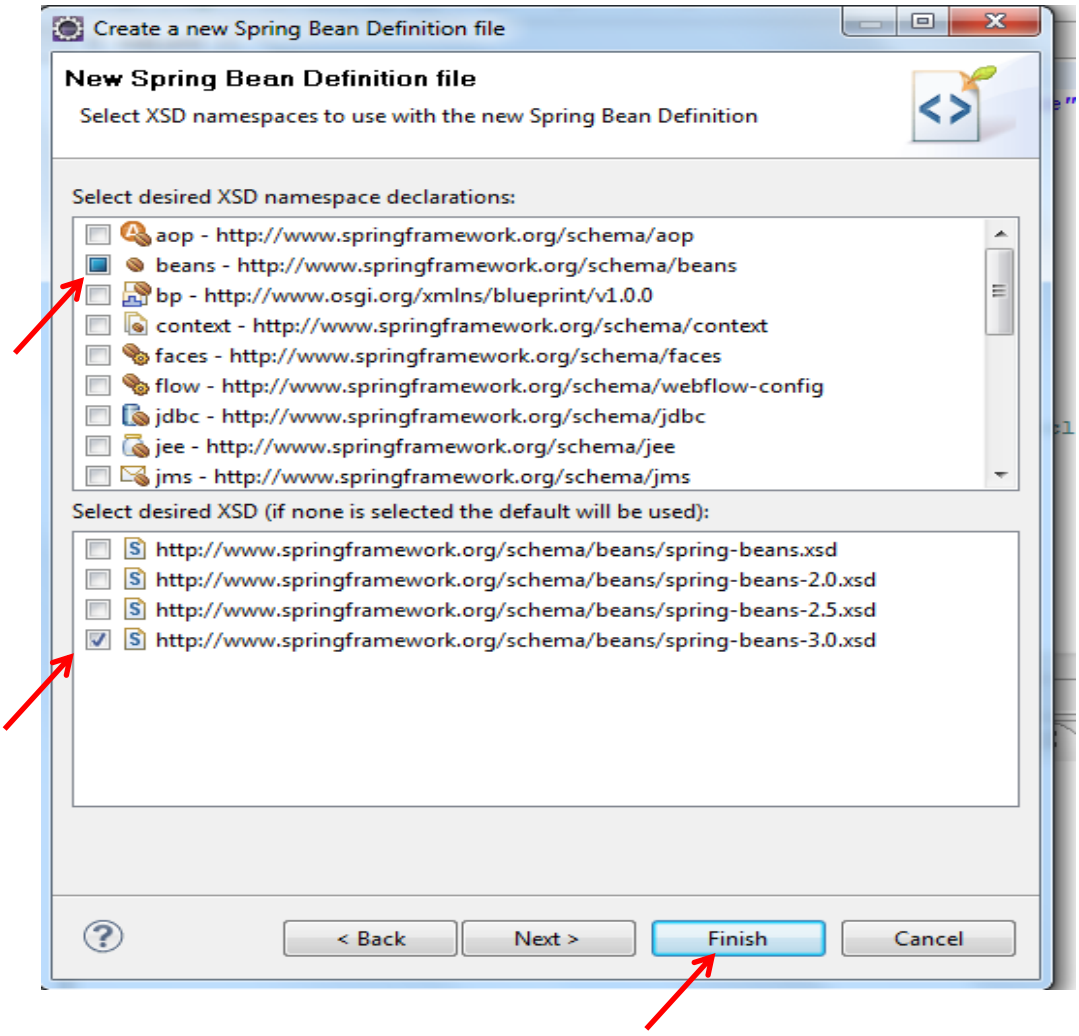


Nommons ce fichier applicationContext



Et cliquons sur **Next,** puis dans la nouvelle fenêtre cliquez sur **beans (dans la première fenêtre)** et **cochez** (**dans la deuxième fenêtre**) la case correspondant au modèle de fichier de configuration se rapportant à la version de Spring utilisée (**la version 3 dans notre tutoriel**), puis cliquez sur **finish**

Voir illustration



Nous venons de créer le fichier de configuration de Spring, celui-ci contiendra les beans Spring qui seront accessibles via les EL de JSF.

Configurons maintenant le conteneur **Spring (contexte d'application web)** dans le fichier web.xml comme suit :

……

……

```
<listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>

<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
```

……

……

La classe **org.springframework.web.context.ContextLoaderListener** permet de spécifier quel est le fichier de configuration de SPRING à charger dans le contexte d'application web

Nous allons enfin configurer le fichier **faces-config.xml** en ajoutant le résolveur comme suit

```
<application>
      <el-resolver>
      org.springframework.web.jsf.el.SpringBeanFacesELResolver
   </el-resolver>
</application>
```

Remarque :
Les résolveurs varient selon les versions de JSF, nous aurions fait usage d'un autre si nous utilisions une version inférieure à 1.2.Vous pourrez consulter la documentation de référence de Spring pour plus d'infos.

Développement de l'application

Nous nous appuierons sur le travail déjà effectué plus haut, nous utiliserons le code faisant référence au service de concaténation du nom et du prénom.

Créez un package (nous l'avons nommé concatenation) et copier ce code dans celui-ci .

Nous modifierons légèrement modifier la fonction `printConcatNomPrenom() de la classe` **`UseServiceConcatNomPrenom`** `comme suit :`

```
public String printConcatNomPrenom(){

        concat.concat(personne.getNom(),personne.getPrenom());

        return "resultatConcat";//allias(ou outcome, voir dans le fichier
                                //faces-config.xml) de la page web
                               // donnant le résultat  à venir

      }
```

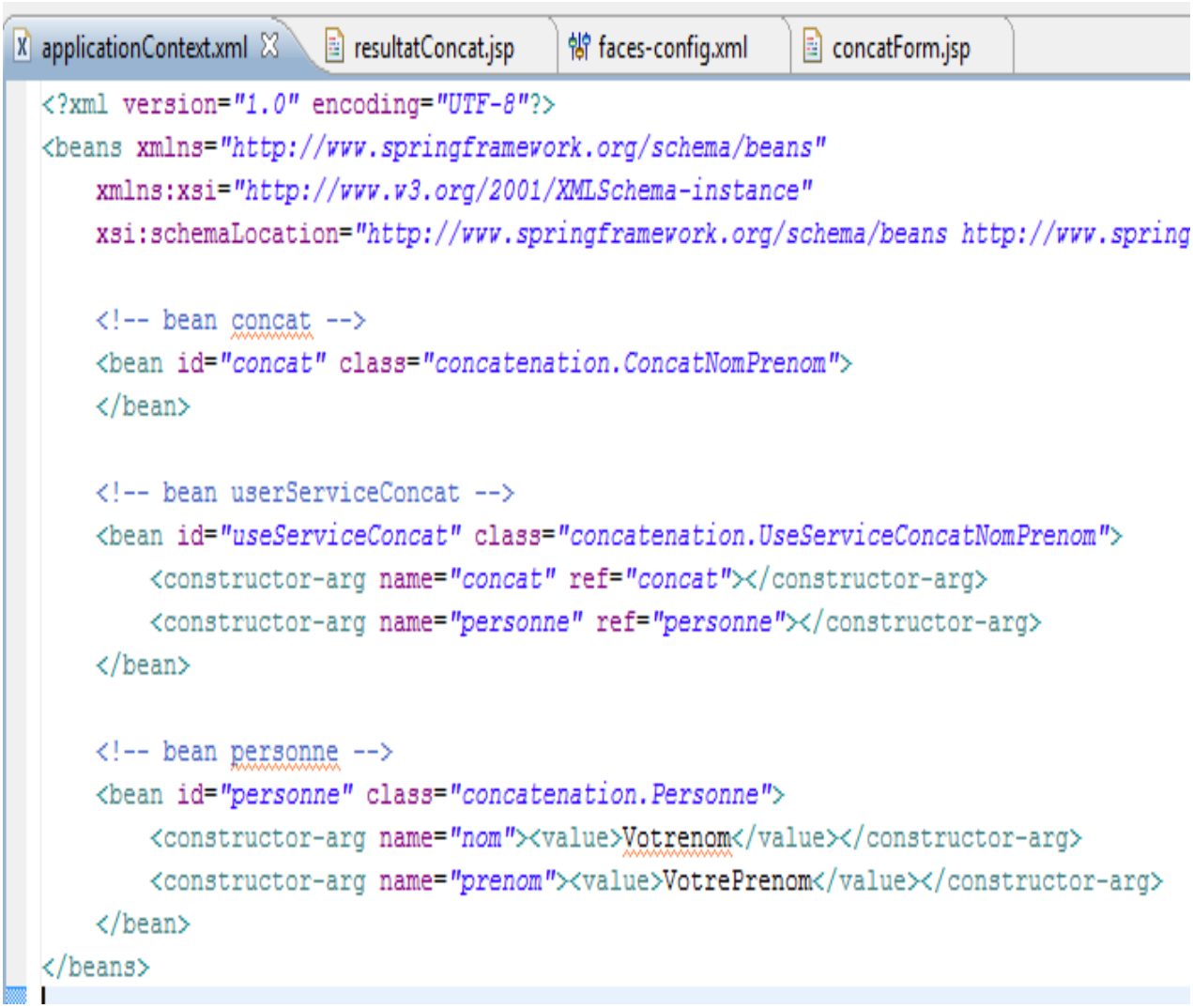
Et nous ajouterons une propriété privée nommée resultat dans la classe ConcatNomPrenom, cette propriété contiendra le résultat de la concaténation.

Par ailleurs nous créerons deux pages web en faisant usage de la techno JSF(concatForm.jsp et resultatConcat.jsp), l'une nous permettons d'entrer le nom et le prénom et l'autre affichant le résultat de la concaténation.

Pour plus d'infos sur JSF se référer à mon tutoriel (introduction sur JSF), par ailleurs le code de ce tutoriel est téléchargeable sur mon site.

Ci-dessous l'illustration du fichier de configuration.

Fichier de configuration



Petite explication :

Nous créons premièrement un bean nommé(ou ayant un identifiant) **concat** de type **ConcatNomPrenom** se trouvant dans le **package concatenation**

Le deuxième bean (**useServiceConcat** de type **UserServiceCncatNomPrenom**) comporte un constructeur avec deux argument, l'un de type **ConcatNomPrenom** et l'autre de **type Personne**. Ces arguments font référence aux beans **concat** et **personne**.

Le dernier bean (**personne**) est de type **Personne**, il comporte aussi un constructeur, les paramètres de ce constructeur auront des valeurs par défaut(**VotreNom**).