# L'utilisation de frameworks pour le développement avec Java EE

## Module 4 – Spring Core

# Objectifs

- Comprendre la nécessité d'un couplage faible vs un couplage fort
- Mettre en place Spring sur les différentes couches d'une application n-tiers
- Savoir configurer Spring à travers le fichier XML
- Comprendre les annotations spécifiques

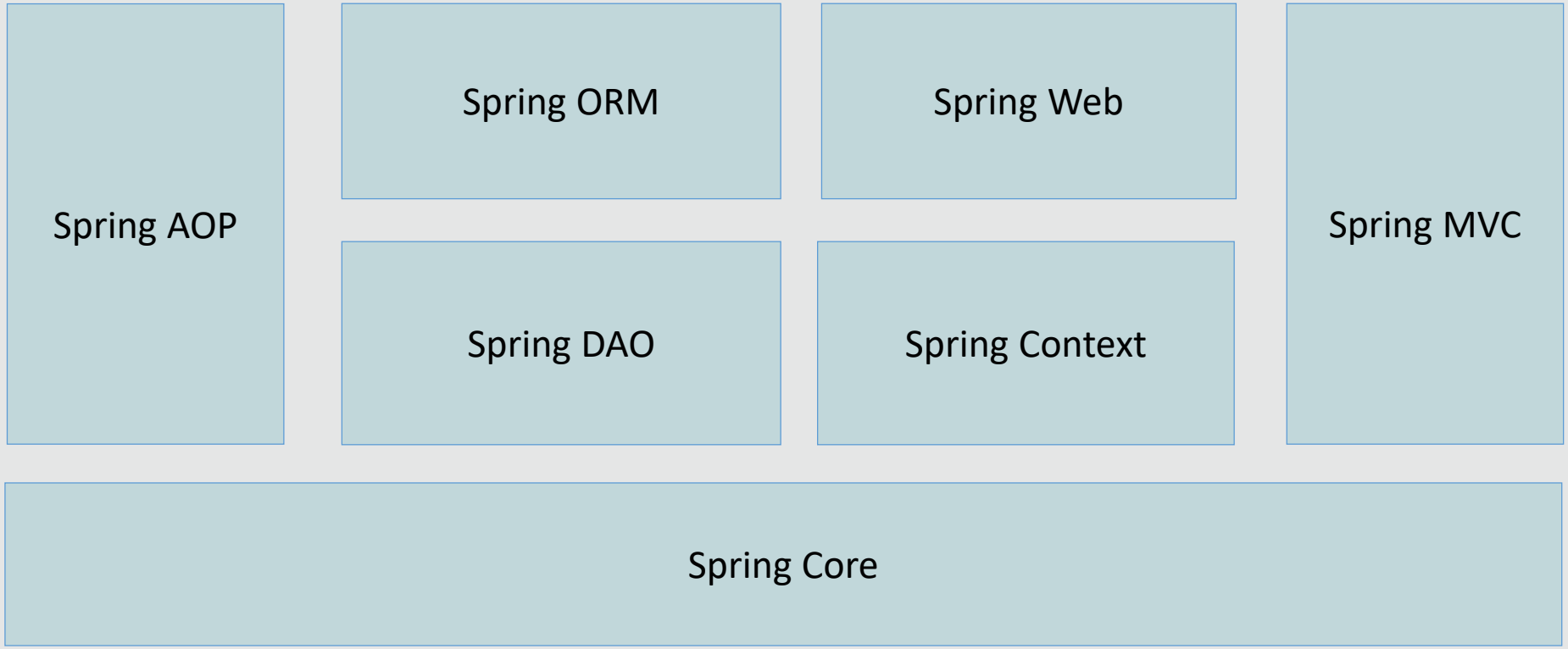
Spring Core

# Origines

- Java EE (à l'époque J2EE)
  - Très puissant
  - Mais très complexe…
    - Les EJB 2
- Alternative
  - Principe de conteneur léger
    - Gestion du cycle de vie des composants métier et techniques
    - Initialisation des composants
    - Gestion des dépendances entre composants
  - Programmation Orientée Aspect (AOP)
    - Intégration des couches transversales (transactions…)

# Les principaux modules de Spring

+ de 20 modules springs !



# Problématique

- Couplage fort / couplage faible

  - Le couplage est une métrique indiquant le niveau d'interaction entre deux ou plusieurs composants logiciels
  - Deux composants sont dits couplés s'ils échangent de l'information
  - Le couplage fort implique une dépendance forte entre deux composants
    - Difficilement réutilisable
    - Difficilement testable
  - Le couplage faible favorise :
    - La faible dépendance entre les classes
    - La réduction de l'impact des changements dans une classe
    - La réutilisation des classes ou modules

# En pratique – couplage fort

- Gestion d'un ensemble de musiciens dans un orchestre
  - Notion d'instrument
    - Création de classes Piano, Violon
  - Notion de musicien
    - Création de classes Pianiste, Violoniste
      - Association avec leur instrument respectif
  - Notion d'orchestre
    - Création d'une classe Orchestre
      - Liste d'objets

# En pratique – couplage fort

```java
public class Piano {
    public void afficher(){
        System.out.println("Je suis un piano ...");
    }
    public void jouer(){
        System.out.println("LA LA LA");
    }
}
```

```java
public class Violon {
    public void afficher(){
        System.out.println("Je suis un violon ...");
    }
    public void jouer(){
        System.out.println("ZIN ZIN ZIN");
    }
}
```

```java
public class Pianiste {
    private Piano piano;
    private String morceau;

    public Pianiste(String morceau) {
        this.morceau = morceau;
        piano = new Piano();
    }
    public void jouerMorceau(){
        piano.afficher();
        System.out.println("et je joue le morceau " + morceau);
        piano.jouer();
    }
}
```

```java
public class Violoniste {
    private Violon violon;
    private String morceau;

    public Violoniste(String morceau) {
        this.morceau = morceau;
        violon = new Violon();
    }
    public void jouerMorceau(){
        violon.afficher();
        System.out.println("et je joue le morceau " + morceau);
        violon.jouer();
    }
}
```

VOIR DEMO

# En pratique – couplage fort

```
public class Orchestre {
    private List<Object> listeMusiciens;

    public Orchestre() {
        listeMusiciens = new ArrayList<>();
    }
    public void ajout(Object musicien) {
        listeMusiciens.add(musicien);
    }

    public void jouer(){
        for (Object object : listeMusiciens) {
            if (object instanceof Pianiste){
                ((Pianiste) object).jouerMorceau();
            }
            if (object instanceof Violoniste){
                ((Violoniste) object).jouerMorceau();
            }
        }
    }
}
```

```
Pianiste pianiste = new Pianiste("La 9eme de Beethoven");
pianiste.jouerMorceau();

Violoniste violoniste = new Violoniste("La 9eme de Beethoven");
violoniste.jouerMorceau();

Orchestre orchestre = new Orchestre();
orchestre.ajout(pianiste);
orchestre.ajout(violoniste);

orchestre.jouer();
```

VOIR DEMO

# En pratique – couplage fort

## Démonstration

# En pratique – couplage faible

- Gestion d'un ensemble de musiciens dans un orchestre
  - Notion d'instrument
    - Création d'une interface Instrument
    - Création de classes Piano, Violon implémentant Instrument
  - Notion de musicien
    - Création d'une classe Musicien
      - Association avec un instrument
  - Notion d'orchestre
    - Création d'une classe Orchestre
      - Liste de Musiciens

# En pratique – couplage faible

```java
public interface Instrument {
    public void afficher();
    public void jouer();
}
```

```java
public class Violon implements Instrument {
    @Override
    public void afficher(){
        System.out.println("Je suis un violon ...");
    }
    @Override
    public void jouer(){
        System.out.println("ZIN ZIN ZIN");
    }
}
```

```java
public class Piano implements Instrument{
    @Override
    public void afficher(){
        System.out.println("Je suis un piano ...");
    }
    @Override
    public void jouer(){
        System.out.println("LA LA LA");
    }
}
```

```java
public class Musicien {
    private String morceau;
    private Instrument instrument;

    public Musicien(String morceau, Instrument instrument) {
        this.morceau = morceau;
        this.instrument = instrument;
    }
    public String getMorceau() {
    public void setMorceau(String morceau) {
    public Instrument getInstrument() {
    public void setInstrument(Instrument instrument) {

    public void jouerMorceau() {
        instrument.afficher();
        System.out.println("et je joue le morceau " + morceau);
        instrument.jouer();
    }
}
```

VOIR DEMO

# En pratique – couplage faible

```java
public class Orchestre {
    private List<Musicien> listeMusiciens;

    public Orchestre() {
        listeMusiciens = new ArrayList<>();
    }
    public void ajout(Musicien musicien) {
        listeMusiciens.add(musicien);
    }

    public void jouer(){
        for (Musicien musicien : listeMusiciens) {
            musicien.jouerMorceau();
        }
    }
}
```

```java
Musicien violoniste = new Musicien("Sonate en UT mineur", new Violon());
violoniste.jouerMorceau();

Musicien pianiste = new Musicien("Sonate en UT mineur", new Piano());
pianiste.jouerMorceau();


Orchestre orchestre = new Orchestre();
orchestre.ajout(pianiste);
orchestre.ajout(violoniste);

orchestre.jouer();
```

# En pratique – couplage faible

## Démonstration

# L’inversion de contrôle

- Inversion of Control (IoC)
  - Réduire les dépendances (couplage) entre des objets dont l'implémentation peut varier
  - Diminuer la complexité de gestion du cycle de vie de ces objets (patterns Singleton et Factory)
  - Le contrôle du flot d'exécution d'une application n'est plus géré par l'application elle-même mais par une structure externe (conteneur)
- Mise en place
  - Utilisation du patron de conception (design pattern) Factory
  - Utilisation de l'injection de dépendances.

# L'injection de dépendances

- Dependency Injection (DI)
- Mécanisme permettant d'implémenter le principe de l'inversion de contrôle
  - Permet d'éviter une dépendance directe entre deux classes et définissant dynamiquement la dépendance plutôt que statiquement
  - Permet à une application de déléguer la gestion du cycle de vie de ses dépendances et leurs injections à une autre entité
  - L'application ne crée pas directement les instances des objets dont elle a besoin : les dépendances d'un objet ne sont pas gérées par l'objet lui-même mais sont gérées et injectées par une entité externe à l'objet

# Ce qu'apporte Spring

- Spring apporte le "conteneur léger"
  - Permet la prise en charge du cycle de vie des objets et leur mise en relation par l'intermédiaire d'un fichier de configuration
- Le noyau de Spring est basé sur :
  - Une fabrique générique de composants informatiques, composants nommés beans
  - Un conteneur capable de stocker ces beans

# Mise en place

- Création d'un fichier de configuration
  - Doit être situé dans le classpath

```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-4.0.xsd">


</beans>
```

# Injection par setter

```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-4.0.xsd">

    <bean id="piano" class="fr.eni.spring.Piano" />
    <bean id="violon" class="fr.eni.spring.Violon" />


    <bean id="musicien" class="fr.eni.spring.Musicien">
        <property name="morceau" value="Pierre et le loup"/>
        <property name="instrument" ref="violon"/>
    </bean>
</beans>
```

```java
ApplicationContext context =
        new ClassPathXmlApplicationContext("ApplicationContext.xml");

Musicien musicien1 = context.getBean("musicien", Musicien.class);
musicien1.jouerMorceau();
```

```
Je suis un violon ...
et je joue le morceau Pierre et le loup
ZIN ZIN ZIN
```

VOIR DEMO

# Injection par setter

```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-4.0.xsd">

    <bean id="piano" class="fr.eni.spring.Piano" />
    <bean id="violon" class="fr.eni.spring.Violon" />


    <bean id="musicien" class="fr.eni.spring.Musicien">
        <property name="morceau" value="Le Bolero de Ravel"/>
        <property name="instrument" ref="piano"/>
    </bean>
</beans>
```

```java
ApplicationContext context =
        new ClassPathXmlApplicationContext("ApplicationContext.xml");

Musicien musicien1 = context.getBean("musicien", Musicien.class);
musicien1.jouerMorceau();
```

```
Je suis un piano ...
et je joue le morceau Le Bolero de Ravel
LA LA LA
```

VOIR DEMO

# Injection par constructeur

- Ordre des arguments par défaut
- Utilisation des index
- Utilisation des noms des arguments

```
public Musicien(String nom, String morceau, Instrument instrument) {
    this.nom = nom;
    this.morceau = morceau;
    this.instrument = instrument;
}
```

```
<bean id="p1" class="fr.eni.spring.Musicien">
    <constructor-arg value="Lemoyen"/>
    <constructor-arg value="La Traviata"/>
    <constructor-arg ref="piano"/>
</bean>

<bean id="p2" class="fr.eni.spring.Musicien">
    <constructor-arg index="2" ref="piano"/>
    <constructor-arg index="1" value="La Traviata"/
    <constructor-arg index="0" value="Legrand"/>
</bean>

<bean id="v" class="fr.eni.spring.Musicien">
    <constructor-arg name="instrument" ref="violon"
    <constructor-arg name="morceau"  value="La Trav      "/>
    <constructor-arg name="nom"  value="Lepetit"/>
</bean>
```

VOIR DEMO

# Autowiring by Name

- Le nom du bean à injecter doit correspondre au nom de l'attribut

```java
public class Musicien {
    private String morceau;
    private Instrument instrument;

    public Musicien() {

    }

    public void setInstrument(Instrument instrument) {
        this.instrument = instrument;
    }
```

```xml
<bean id="instrument" class="fr.eni.spring.Piano" />
<!--    <bean id="instrument" class="com.formation.bean.Violon" /> -->


<bean id="musicien" class="fr.eni.spring.Musicien" autowire="byName">
    <property name="morceau" value="Pierre et le loup"/>
</bean>
```

VOIR DEMO

# Autowiring by Type

- Le conteneur recherche un bean dont le type correspond

```java
public class Musicien {
    private String morceau;
    private Instrument instrument;

    public Musicien() {

    }

    public void setInstrument(Instrument instrument) {
        this.instrument = instrument;
    }
```

```xml
<!--
    <bean id="piano" class="fr.eni.spring.Piano" />
    <bean id="violon" class="fr.eni.spring.Violon" />
 -->

<bean class="fr.eni.spring.Piano" />

<bean id="musicien" class="fr.eni.spring.Musicien" autowire="byTy
    <property name="morceau" value="La chevauchée des Walkyries"/
</bean>
```

VOIR DEMO

# Scope des beans

- Spring définit 5 scopes
  - singleton
    - Scope par défaut
  - prototype
    - Une nouvelle instance à chaque injection
  - request (uniquement en environnement web)
    - Une nouvelle instance pour chaque requête HTTP
  - session (uniquement en environnement web)
    - Une nouvelle instance pour chaque nouvelle session
  - Mais encore : Application, Websocket...

  - Ressource : http://www.baeldung.com/spring-bean-scopes

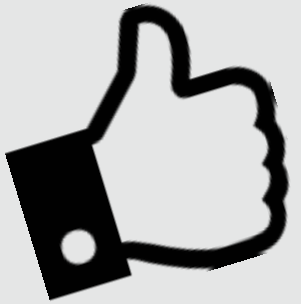
```
<bean id="musicien" class="fr.eni.spring.Musicien" scope="prototype">
    <property name="morceau" value="Le Bolero de Ravel" />
    <property name="instrument" ref="piano" />
</bean>

<bean id="musicien2" class="fr.eni.spring.Musicien" scope="singleton">
    <property name="morceau" value="Le Bolero de Ravel" />
    <property name="instrument" ref="violon" />
</bean>
```

Spring Core

# Configuration par fichier

## Démonstration

# Les annotations



- Ajout de la balise <context:annotation-config /> dans le fichier de configuration indiquant au conteneur l'utilisation des annotations

- Ajout de la balise <context:component-scan base-package=" ... " /> indiquant les packages que le conteneur devra scanner afin de rechercher les classes annotées

# Les annotations


- Annotations SPRING :
  - @Component
    - Annotation de base
  - @Repository
    - Annotation sur une classe DAO
  - @Service
    - Annotation sur un service métier
  - @Controller
    - Annotation sur un contrôleur Spring MVC
  - @Autowired
    - Injection by Type
    - Attribut required (injection optionnelle)

# Les annotations


- Annotation JAVAX (Spécification Java)
  - @Resource
    - Injection by Name
    - Standard Java SE 6 (JSR 250 : commons annotations)
  - @Inject
    - Standard Java EE 6 (JSR 330 : Java D.I.)
    - Pas d'attribut required

# En pratique

```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:context="http://www.springframework.org/schema/context"

    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.0.xsd
    ">

    <!-- Possibilité d'avoir des injections via des annotations -->
    <context:annotation-config />

    <!--  liste de tous les packages qui peuvent contenir des classes "à injecter"
        (ceux annotés avec @Component, @Controller, @Service ou @Repository -->

    <context:component-scan  base-package="fr.eni.spring" />

</beans>
```

```java
@Component
public class Violon implements Instrument {
    @Override
    public void afficher(){
        System.out.println("Je suis un violon ...");
    }
    @Override
    public void jouer(){
        System.out.println("ZIN ZIN ZIN");
    }
}
```

```java
@Component
public class Piano implements Instrument{
    @Override
    public void afficher(){
        System.out.println("Je suis un piano ...");
    }
    @Override
    public void jouer(){
        System.out.println("LA LA LA");
    }
}
```

VOIR DEMO

# En pratique


```java
@Component(value="mus")
public class Musicien {
    private String morceau;

    @Resource(name = "violon")
    private Instrument instrument;

    public Musicien() {
        this.morceau = "Bolero de Ravel";
    }
```

```java
ApplicationContext context =
        new ClassPathXmlApplicationContext("ApplicationContext.xml");

Musicien musicien = context.getBean("mus", Musicien.class);
musicien.jouerMorceau();

((ClassPathXmlApplicationContext) context).close();
```

```
Je suis un violon ...
et je joue le morceau Bolero de Ravel
ZIN ZIN ZIN
```

VOIR DEMO

# En pratique - @Autowired

```java
@Component
public class Violon implements Instrument {
    @Override
    public void afficher(){
        System.out.println("Je suis un violon ...");
    }
    @Override
    public void jouer(){
        System.out.println("ZIN ZIN ZIN");
    }
}
```

```java
@Component
public class Piano implements Instrument{
    @Override
    public void afficher(){
        System.out.println("Je suis un piano ...");
    }
    @Override
    public void jouer(){
        System.out.println("LA LA LA");
    }
}
```

```java
@Component(value="mus")
public class Musicien {
    private String morceau;

    @Autowired(required=false)
    private Instrument instrument;

    public Musicien() {
        this.morceau = "Bolero de Rave
    }
```

VOIR DEMO

```
Exception in thread "main" org.springframework.beans.factory.BeanCreationException:
Error creating bean with name 'mus': Injection of autowired dependencies failed;
Could not autowire field: private fr.eni.spring.Instrument fr.eni.spring.Musicien.instrument;
No qualifying bean of type [fr.eni.spring.Instrument] is defined:

expected single matching bean but found 2: piano,violon
```

# En pratique - @Autowired


```java
@Component
public class Violon implements Instrument {
    @Override
    public void afficher(){
        System.out.println("Je suis un violon ...");
    }
    @Override
    public void jouer(){
        System.out.println("ZIN ZIN ZIN");
    }
}
```

```java
@Component
public class Piano implements Instrument{
    @Override
    public void afficher(){
        System.out.println("Je suis un piano ...");
    }
    @Override
    public void jouer(){
        System.out.println("LA LA LA");
    }
}
```

```java
@Component(value="mus")
public class Musicien {
    private String morceau;

    @Autowired(required=false)
    @Qualifier("piano")
    private Instrument instrument;

    public Musicien() {
        this.morceau = "Bolero de Rave
    }
```

VOIR DEMO

```
Je suis un piano ...
et je joue le morceau Bolero de Ravel
LA LA LA
```

Spring Core

# Configuration par annotations

## Démonstration

# En pratique - @Inject et @Named


```java
import javax.inject.Named;

@Named(value = "violon")
public class Violon implements Instrument {
    @Override
    public void afficher() {
        System.out.println("Je suis un violon ...");
    }

    @Override
    public void jouer() {
        System.out.println("ZIN ZIN ZIN");
    }
}
```

```java
import javax.inject.Named;

@Named(value="piano")
public class Piano implements Instrument{
    @Override
    public void afficher(){
        System.out.println("Je suis un piano ...");
    }

    @Override
    public void jouer(){
        System.out.println("LA LA LA");
    }
}
```

```java
@Component(value="mus")
public class Musicien {
    private String morceau;

    @Inject
    @Named(value="violon")
    private Instrument instrument;

    public Musicien() {
        this.morceau = "Bolero de Ravel";
    }
```

Spring Core

# Configuration par annotations

## Démonstration

# Intégration dans une application web

- Importation des bibliothèques suivantes :
  - org.springframework : spring-context
  - org.springframework : spring-web
  - javax.servlet : javax.servlet-api

# Intégration dans une application web

- Modification du fichier web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0">

  <display-name>10-Web</display-name>

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

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

</web-app>
```

VOIR DEMO

# Intégration dans une application web

- Création du fichier de configuration Spring

```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-4.0.xsd">

    <bean class="fr.eni.spring.Message">
        <property name="msg" value="Bienvenue chez Spring !"/>
    </bean>

</beans>
```

# Intégration dans une application web

- Modification de la servlet

```
@WebServlet("/mess")
public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Autowired
    private Message message;

    @Override
    public void init(ServletConfig config) throws ServletException {
        SpringBeanAutowiringSupport
            .processInjectionBasedOnServletContext(this, config.getServletContext());
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        PrintWriter pw = response.getWriter();
        pw.println("<html><body><h1>" + message.getMsg() + "</h1></body></html>");
    }
}
```

VOIR DEMO

Spring Core

# Spring dans une application web

## Démonstration

# TP-04-Mediatheque