Architecture JEE / Spring

Introduction

Cours n°1
ENSG - M2 TSI

Clément BOIN
clement.boin@esiee.fr

Objectifs du cours

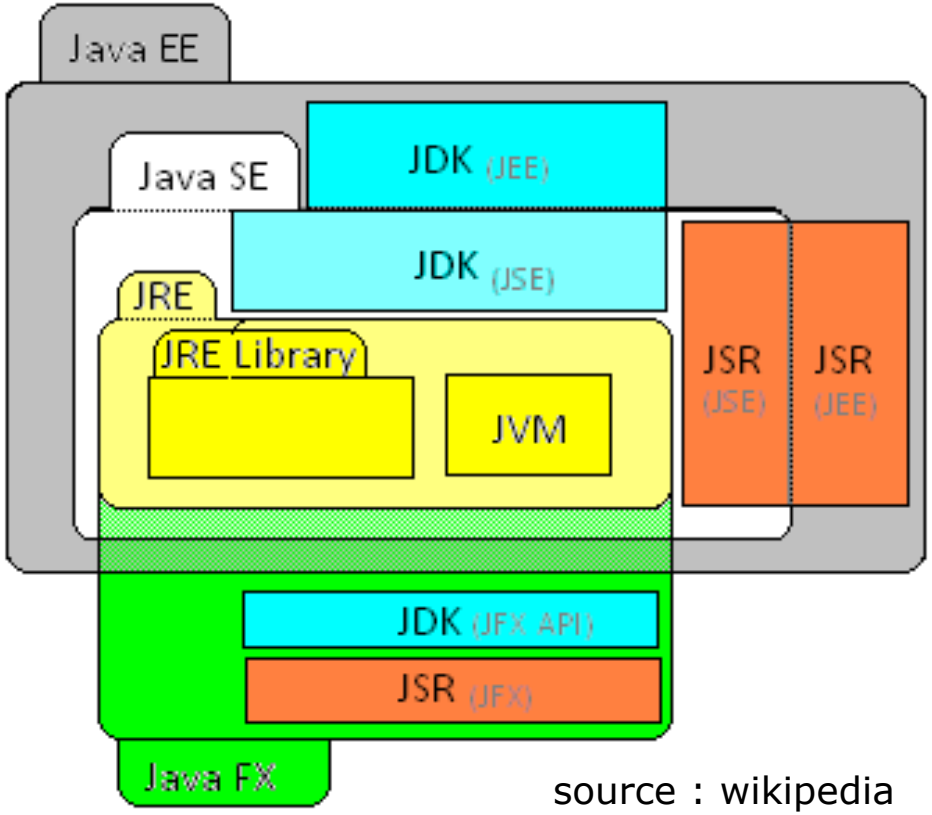
- Comprendre les concepts à la base de JEE;
- Connaître un minimum de jargon JEE ;
- Etre capable de réaliser le «backend » d'une application web de complexité modérée;
- Utilisation de plusieurs frameworks :
  - Spring
  - Hibernate/JPA pour la persistance
  - Spring Boot pour simplifier la configuration de Spring

Plan du cours

- Introduction Java EE + Spring (1,5j)
  - Injection de dépendances
- Framework Hibernate (1,5j)
  - avec JPA
  - Spring Data (peut-être !)
- SpringBoot - Spring MVC (1j)
- Evaluation :
  - Rendu individuel des TP
  - Un projet web "backend" à rendre (en binôme)
  - Un examen et/ou QCM lors de la dernière séance

Première partie

Introduction JEE/ Spring



source : wikipedia

Java -> Jakarta EE

Jakarta Enterprise Edition

Introduction Java EE

- Java EE : *Java Enterprise Edition*
  - *anciennement raccourci en « J2EE »*
  - *version courant JEE 1.10*
  - *Dernière version (fondation Eclipse) : Jakarta EE 10*

- La plate-forme Java EE (orientée serveur)
  - Java SE + un grand nombre de bibliothèques

- *Objectif :* faciliter le développement d'applications distribuées
  - pour des application de grande envergure (qui doivent évoluer à long terme)
  - code beaucoup plus compliqué

- Les applications sont déployées et exécutées sur un serveur d'applications

- Documentation : https://jakarta.ee/

Avantages Java EE

- une architecture d'applications basée sur les composants
  - découpage de l'application
  - séparation des rôles
- interfaçage possible avec de nombreuses API : JDBC, JPA, JSP...
- choix des outils de développement et des serveurs d'applications utilisés
- Découpage de l'application :
  - client/métier/données
  - MVC

SPRING

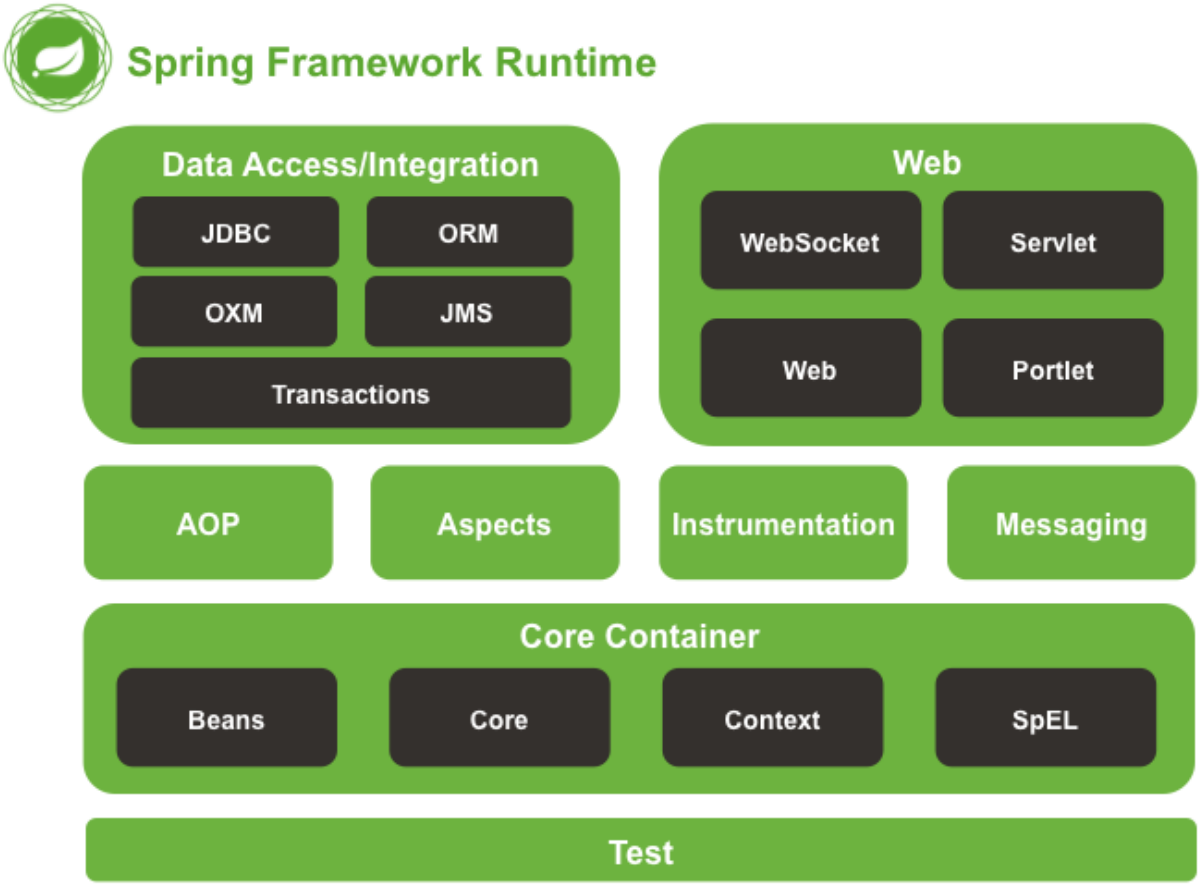
Spring

- Framework Java libre
  - Rod Johnson (2002)

- Définir l'infrastructure d'une application Java
  - Version actuelle : 6.1.2
    - En réaction à Java EE (et les EJB2)
    - *Remarque : Désormais les EJB3 sont plus « simple » à utiliser*

- Conteneur "léger"
  - implémente les spécifications Java EE + d'autres modules
  - pas d'interface à implémenter (comme pour les EJB)
    - POJO : Plain Old Java Object (bon vieil objet Java)

Schéma Architecture Spring



Les concepts de Spring
(fruit des design pattern)

- IoC : Inversion de contrôle (voir dans la suite)

- Programmation orientée aspect
  - paradigme de programmation
  - éléments transversaux : sécurité, journalisation...
  - non traité dans ce cours (sauf si on a le temps !)

- Un couche d'abstraction
  - intégration d'autres frameworks et bibliothèques
    - **Hibernate**, Junit, AspectJ…
  - framework multi-couches, peut s'intégrer au niveau de toutes les couches (d'un MVC par exemple)

Idée principale

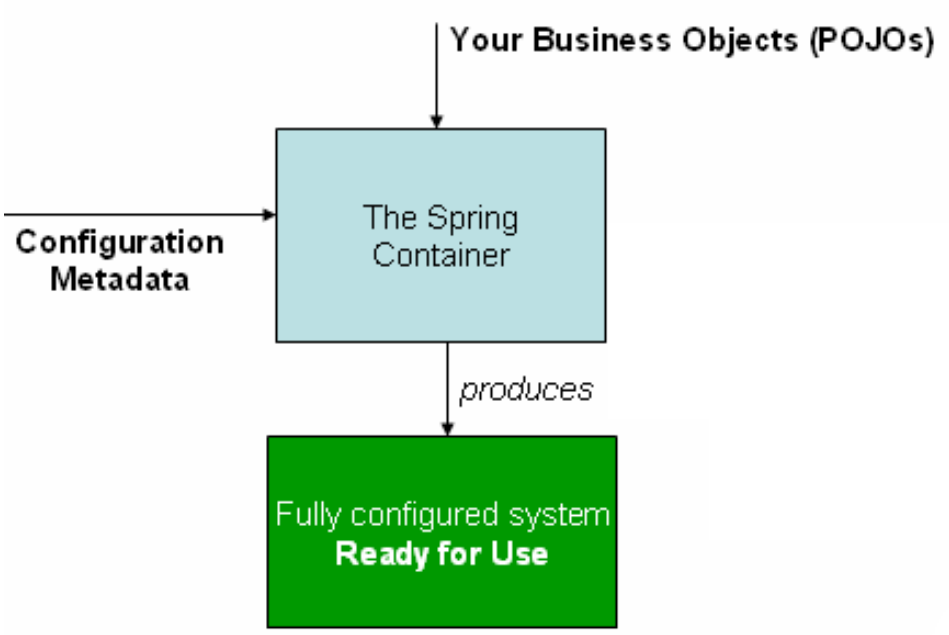
- L'idée est de faire évoluer l'application sans toucher à une ligne de code déjà écrite.

Exemples :

- rajouter un nouveau code promotionnel,
- changer la méthode de validation des mots de passe,
- rallonger les délais de livraisons indicatifs

Inversion de contrôle dans Spring

- Deux façons de faire :
  - la recherche de dépendance
    - interroger le conteneur afin de trouver ses dépendances avec les autres objets
  - l'injection de dépendances (voir suite)
    - découpler les liens de dépendances entre objets



Injection de dépendance

- Permet d'implémenter le principe d'inversion de contrôle (IoC)

- Créer dynamiquement les dépendances entres les objets
  - en s'appuyant sur une description
    - fichier de configuration
    - métadonnées

Inversion de contrôle
Exemple / Constatation

```
public interface PasswordChecker {
        public boolean check(String pwd);
}

public RegularPasswordChecker {
        public boolean check(String pwd){
                return pwd.length>=8;
        }
}

public SecurePasswordChecker {
        public boolean check(String pwd){
                return pwd.length>=12;
        }
}
```

```
public class Application {
   ...
   String password = ...
   PasswordChecker pc = new
RegularPasswordChecker();
if(pc.check(password))
{ .... }
}
```

- Si on veut changer de méthode de vérification de mot de passe, il faut modifier le code de la classe Application

Inversion de contrôle
Exemple / Solution

Application en Spring

```
public class Application {
  ApplicationContext applicationContext=new ClassPathXmlApplicationContext("Maconfig.xml");

  PasswordChecker pc = (PasswordChecker) applicationContext.getBean("typedeverif",tsi.ensg.RegularChecker);
  String password = ...
  if(pc.check(password))
  { .... }
}
```

- La méthode getBean prend en paramètre l'id du bean
  - Peut aussi prendre le nom de la classe, voire les deux

Fichier XML

```
<?xml version="1.0" encoding="UTF-8"?>
   <beans>
   <bean id="typedeverif" class="tsi.ensg.RegularChecker"/>
   </beans>
</xml>
```

- Délégation de la création d'objet à un conteneur qui lit les informations d'un fichier XML.

Le composant logiciel
bean

- Un bean est un objet
  - instancié, assemblé ou géré par le conteneur IoC

- Un composant logiciel est :
  - une interface
  - et les différents beans qui lui sont associés

```
<bean id="typedeverif" class="tsi.ensg.RegularChecker"/>
```

- **id** est l'identifiant utilisé (`typedeverif`) pour injecter la méthode
- **class** est le nom de la classe à injecter (`tsi.ensg.RegularChecker`)

Deuxième partie

Spring : Injection de dépendances

Injection de dépendances

- Le conteneur IoC peut injecter dans les beans qu'il construit:
  - des valeurs
  - d'autres beans

- Injections :
  - par constructeur
  - par setter (par un champs)

Injection par constructeur (1/2)

```
public class Adresse {
        private final int numero;
        private final String rue;

        public Adresse(int numero, String rue) {
                this.numero = numero;
                this.rue = rue;
        }

        public String toString() {...};
}
```

```
<bean id="adresseToto" class="tsi.ensg.Adresse"/>
          <constructor-arg value=11/>
          <constructor-arg value="rue des Roses"/>
</beans>
```

- Par défaut les beans sont créés en appelant leur constructeur sans paramètres.
- On peut aussi fournir des paramètres au constructeur.

Injection par constructeur (2/2)

Injection d'un bean

```
public class Personne {
        private final String nom;
        private final Adresse adresse;

        public Personne(String nom, Adresse adresse) {
                this.nom = nom;
                this.adresse = adresse;
        }

        public String toString() {...};
}
```

```
<bean id="Toto" class="tsi.ensg.Personne"/>
          <constructor-arg value="Toto"/>
          <constructor-arg ref="adresseToto"/>
</beans>
```

- On peut aussi injecter un bean existant (`adresseToto` ici) par constructeur

Injection par setter

```
public class Personne {
        private final String nom;
        private final Adresse adresse;

        public Personne(String nom) {
                this.nom = nom;
        }

        public void setAdresse(Adresse adresse){
                this.adresse = adresse;
        }
}
```

- On peut aussi injecter une valeur en passant par le setter

```
<bean id="adresseTiti" class="tsi.ensg.Adresse"/>
          <constructor-arg value="4"/>
          <constructor-arg value="boulevard Copernic"/>
</beans>

<bean id="titi" class="tsi.ensg.Personne">
         <constructor-arg value="Titi"/>
         <property name="adresse" ref="adresseTiti"/>
</bean>
```

Injections de collections

List, Set, Map

- Il est également possible d'injecter des collections Java
  - collections de valeurs ou de beans

```
public class Personne {
        private final String nom;
        private final Adresse adresse;
        private List<String> emails;

        public Personne(String nom) {
                this.nom = nom;
        }

        public void setEmails(List<String> emails){
                this.emails = List.copyOf(emails);
        }
}
```

Injections de collections

List, Set, Map

```
<bean id="titi" class="tsi.ensg.Personne">
        <constructor-arg value="Titi"/>
        <property name="adresse" ref="adresseTiti"/>
        <property name="emails">
                <list>
                        <value>titi@hotmail.com</value>
                        <value>titi@gmail.com</value>
                </list>
        </property>
</bean>
```

Injections de collections

en utilisant des beans

```
public class Book {
    private String title;
    private long ref;

    public Book(String title,long ref){...};
}

public class Library {
    private Set<Book> books;

    public Library(Set<Book> books) {...};

     ...
}
```

```
<bean id="book1" class="Book">
    <constructor-arg  value="Le livre de la jungle"/>
    <constructor-arg  value="55453"/>
</bean>
<bean id="book2" class="Book">
    <constructor-arg  value="Le lion"/>
    <constructor-arg  value="554545"/>
</bean>
<bean id="library" class="Library">
    <constructor-arg>
        <set>
            <ref bean="book1"/>
            <ref bean="book2"/>
        </set>
    </constructor-arg>
</bean>
```

Autowire

- Nous dépendons des id des beans pour déterminer les beans à injecter.

- Avec autowire, on peut simplifier les beans à injecter.

  - autowire="byName"
    - Pour chaque champs avec un setter, le conteneur cherche un bean ayant pour id le nom du champs et il l'injecte.

  - autowire="byType"
    - Pour chaque champs avec un setter, le conteneur cherche un bean ayant le même type que le champs.
    - *Attention si plusieurs beans ont le même type, on a une exception.*

  - autowire="constructor"
    - Comme pour autowire="byType" mais avec le constructeur.

Autowire byName

Sans autowire

```
<bean id="adresseTiti" class="tsi.ensg.Adresse"/>
          <constructor-arg value="4"/>
          <constructor-arg value="boulevard Copernic"/>
</beans>

<bean id="titi" class="tsi.ensg.Personne">
         <constructor-arg value="Titi"/>
         <property name="adresse" ref="adresseTiti"/>
</bean>
```

Avec autowire

```
<bean id="adresse" class="tsi.ensg.Adresse"/>
          <constructor-arg value="4"/>
          <constructor-arg value="boulevard Copernic"/>
</beans>

<bean id="titi" class="tsi.ensg.Personne" autowire="ByName"/>
```

- Attention : l'id du bean est égal au champs adresse de la Personne

Autowire byType

Sans autowire

```
<bean id="adresseTiti" class="tsi.ensg.Adresse"/>
          <constructor-arg value="4"/>
          <constructor-arg value="boulevard Copernic"/>
</beans>

<bean id="titi" class="tsi.ensg.Personne">
         <constructor-arg value="Titi"/>
         <property name="adresse" ref="adresseTiti"/>
</bean>
```

Avec autowire byType

```
<bean id="adresseTiti" class="tsi.ensg.Adresse"/>
          <constructor-arg value="4"/>
          <constructor-arg value="boulevard Copernic"/>
</beans>

<bean id="titi" class="tsi.ensg.Personne" autowire="ByType"/>
```

- Attention : fonctionne uniquement si adresseTiti est le seul bean de ce type