

Initiation aux frameworks : *Spring*

Spring et les concepts de l'inversion de Contrôle

Thomas Duchatelle (duchatelle.thomas@gmail.com)

Capgemini, pour Yves Rocher

February 11, 2013

- 1 Séparation des préoccupations
- 2 Inversion de contrôle
- 3 Spring
- 4 Conclusion

Sommaire

1 Séparation des préoccupations

- Définition
- Cas concret

2 Inversion de contrôle

3 Spring

4 Conclusion

Séparation des préoccupations

SoC : Separation of Concerns

Pris isolément, chaque problème est plus facile à traiter.

Séparation des préoccupations

SoC : Separation of Concerns

Pris isolément, chaque problème est plus facile à traiter.

Découpage de l'application pour isoler les problématiques :

- persistance
- services métier
- présentation (IHM Web)
- appel webservice
- ...

Séparation des préoccupations

SoC : Separation of Concerns

Pris isolément, chaque problème est plus facile à traiter.

Découpage de l'application pour isoler les problématiques :

- persistance
- services métier
- présentation (IHM Web)
- appel webservice
- ...

Beans

Pour chaque nature de problématique : conception de "composants spécialisés", de *briques applicatives*.

Cas concret

Embauche d'un nouvel employé

Nouvelle embauche

Intégration dans le SI d'un nouvel employé : création de son matricule, email et insertion dans le système des ressources humaines.

Cas concret

Embauche d'un nouvel employé

Nouvelle embauche

Intégration dans le SI d'un nouvel employé : création de son matricule, email et insertion dans le système des ressources humaines.

Processus métier pour l'embauche d'un nouveau client :

- 1 un administrateur renseigne le nom, prénom et intitulé du poste du nouvel employé

Cas concret

Embauche d'un nouvel employé

Nouvelle embauche

Intégration dans le SI d'un nouvel employé : création de son matricule, email et insertion dans le système des ressources humaines.

Processus métier pour l'embauche d'un nouveau client :

- 1 un administrateur renseigne le nom, prénom et intitulé du poste du nouvel employé
- 2 le système génère le matricule de l'employé : identifiant unique

Cas concret

Embauche d'un nouvel employé

Nouvelle embauche

Intégration dans le SI d'un nouvel employé : création de son matricule, email et insertion dans le système des ressources humaines.

Processus métier pour l'embauche d'un nouveau client :

- 1 un administrateur renseigne le nom, prénom et intitulé du poste du nouvel employé
- 2 le système génère le matricule de l'employé : identifiant unique
- 3 le système génère l'email de l'employé : à partir de son nom et prénom, unique aussi

Cas concret

Embauche d'un nouvel employé

Nouvelle embauche

Intégration dans le SI d'un nouvel employé : création de son matricule, email et insertion dans le système des ressources humaines.

Processus métier pour l'embauche d'un nouveau client :

- 1 un administrateur renseigne le nom, prénom et intitulé du poste du nouvel employé
- 2 le système génère le matricule de l'employé : identifiant unique
- 3 le système génère l'email de l'employé : à partir de son nom et prénom, unique aussi
- 4 toutes ces données sont conservées dans le Référentiel Employés

Cas concret

Embauche d'un nouvel employé

Nouvelle embauche

Intégration dans le SI d'un nouvel employé : création de son matricule, email et insertion dans le système des ressources humaines.

Processus métier pour l'embauche d'un nouveau client :

- 1 un administrateur renseigne le nom, prénom et intitulé du poste du nouvel employé
- 2 le système génère le matricule de l'employé : identifiant unique
- 3 le système génère l'email de l'employé : à partir de son nom et prénom, unique aussi
- 4 toutes ces données sont conservées dans le Référentiel Employés
- 5 le système informe l'application des Ressources Humaines de la création de nouvel employé

Méga script !

Un script PHP suffit

- Un tel processus pourrait être écrit en un seul script PHP...

Méga script !

Un script PHP suffit

- Un tel processus pourrait être écrit en un seul script PHP...
- Mais :

Méga script !

Un script PHP suffit

- Un tel processus pourrait être écrit en un seul script PHP...
- Mais :
 - difficulté d'écrire le script

Méga script !

Un script PHP suffit

- Un tel processus pourrait être écrit en un seul script PHP...
- Mais :
 - difficulté d'écrire le script
 - longueur et lisibilité du script ?

Méga script !

Un script PHP suffit

- Un tel processus pourrait être écrit en un seul script PHP...
- Mais :
 - difficulté d'écrire le script
 - longueur et lisibilité du script ?
 - tests de tous les cas

Méga script !

Un script PHP suffit

- Un tel processus pourrait être écrit en un seul script PHP...
- Mais :
 - difficulté d'écrire le script
 - longueur et lisibilité du script ?
 - tests de tous les cas
 - gestion des cas d'erreurs

Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

- *IHM* (couche de présentation) : propose une interface intuitive à l'administrateur afin de récolter les données

Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

- *IHM* (couche de présentation) : propose une interface intuitive à l'administrateur afin de récolter les données
- *Gestionnaire des Employés* (objet métier) : détient les règles et le processus de création d'un employé

Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

- *IHM* (couche de présentation) : propose une interface intuitive à l'administrateur afin de récolter les données
- *Gestionnaire des Employés* (objet métier) : détient les règles et le processus de création d'un employé
- *Générateur de matricules* (objet métier) : détient les règles de génération d'un identifiant unique

Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

- *IHM* (couche de présentation) : propose une interface intuitive à l'administrateur afin de récolter les données
- *Gestionnaire des Employés* (objet métier) : détient les règles et le processus de création d'un employé
- *Générateur de matricules* (objet métier) : détient les règles de génération d'un identifiant unique
- *Générateur d'email* (objet métier) : génère un email à partir du nom et prénom.

Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

- *IHM* (couche de présentation) : propose une interface intuitive à l'administrateur afin de récolter les données
- *Gestionnaire des Employés* (objet métier) : détient les règles et le processus de création d'un employé
- *Générateur de matricules* (objet métier) : détient les règles de génération d'un identifiant unique
- *Générateur d'email* (objet métier) : génère un email à partir du nom et prénom.
- *DAO Employés* (objet d'accès aux données) : persiste l'employé et détermine si un email est disponible

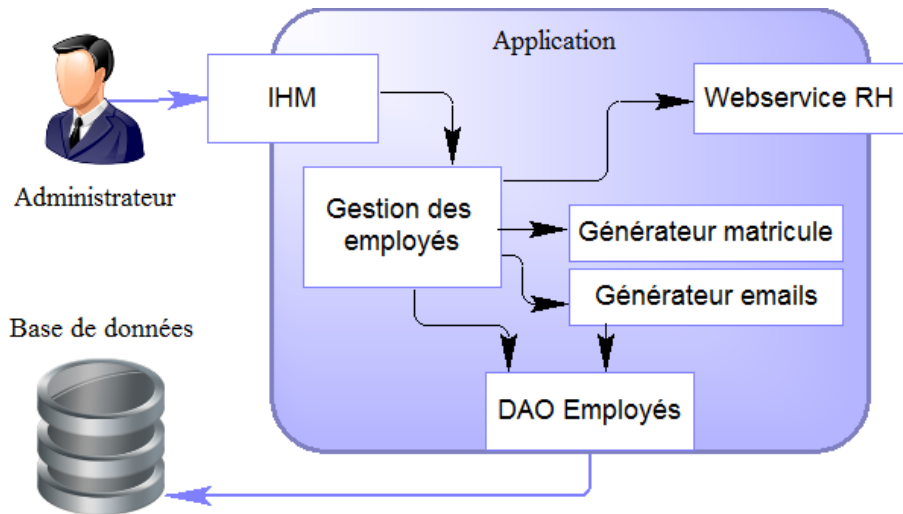
Séparation des préoccupations

Division de la problématique en petites sous problématique

Proposition de découpage :

- *IHM* (couche de présentation) : propose une interface intuitive à l'administrateur afin de récolter les données
- *Gestionnaire des Employés* (objet métier) : détient les règles et le processus de création d'un employé
- *Générateur de matricules* (objet métier) : détient les règles de génération d'un identifiant unique
- *Générateur d'email* (objet métier) : génère un email à partir du nom et prénom.
- *DAO Employés* (objet d'accès aux données) : persiste l'employé et détermine si un email est disponible
- *Connecteur Webservice Ressources Humaines* (objet métier) : gère la connexion avec le webservice de l'application des RH.

Architecture de l'exemple



Comment gérer efficacement toutes ces briques applicatives ?

Sommaire

1 Séparation des préoccupations

2 Inversion de contrôle

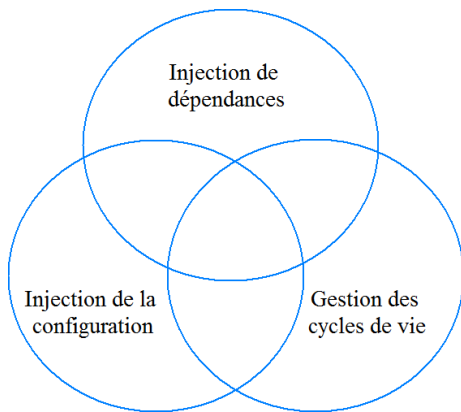
- Définitions
- Cycle de vie
- Injection des dépendances
- Gestion de la configuration
- Environnements multiples

3 Spring

4 Conclusion

Inversion de Contrôle

Association de 3 grands patterns



Les 3 grands patterns :

- **Injection de dépendances**
- **Injection de la configuration**
- **Gestion des cycles de vie**

Conteneur

Différents types de conteneurs

Conteneur

Infrastructure fournissant l'inversion de contrôle : gestion du cycle de vie, injection des dépendances, injection de la configuration.

Conteneur

Différents types de conteneurs

Conteneur

Infrastructure fournissant l'inversion de contrôle : gestion du cycle de vie, injection des dépendances, injection de la configuration.

Il en existe de 2 types :

- **Conteneur lourd** : intégré à une application à part entière, un serveur d'applications (Websphère, Tomcat, JBoss, ...)
- **Conteneur léger** : le conteneur n'est qu'une librairie embarquée dans l'application, moins intrusive et plus flexible (*Spring*, Guice)

Gestion du cycle de vie

Vie d'un objet

Un objet est instancié (créé) à l'aide du mot clef `new` et est détruit par le *garbage collector* lorsqu'il n'est plus référencé.

```
1 // Intanciation d'un nouvel employe  
2 Employee employee = new Employee();
```


Gestion du cycle de vie

Vie d'un objet

Un objet est instancié (créé) à l'aide du mot clef `new` et est détruit par le *garbage collector* lorsqu'il n'est plus référencé.

```
1 // Intanciation d'un nouvel employe  
2 Employee employee = new Employee();
```

Cycle de vie

Gérer le cycle de vie d'un objet consiste à le créer lorsqu'il est utile, et le déréférencer lorsqu'on en a plus besoin.

Portée

La portée définit la validité d'une instance en fonction du contexte. Autrement dit, savoir quand une instance doit être réutilisée, ou quand une nouvelle instance doit être créée.

Portée

La portée définit la validité d'une instance en fonction du contexte. Autrement dit, savoir quand une instance doit être réutilisée, ou quand une nouvelle instance doit être créée.

Portées possibles d'un bean :

- **singleton** : une seule instance est créée pour l'ensemble de l'application, en général au lancement de l'application. Elle n'est détruite qu'à l'arrêt de l'application

Portée

La portée définit la validité d'une instance en fonction du contexte. Autrement dit, savoir quand une instance doit être réutilisée, ou quand une nouvelle instance doit être créée.

Portées possibles d'un bean :

- **singleton** : une seule instance est créée pour l'ensemble de l'application, en général au lancement de l'application. Elle n'est détruite qu'à l'arrêt de l'application
- **prototype** : une instance est créée à chaque demande, elle est déréférencée dès que possible

Portée

La portée définit la validité d'une instance en fonction du contexte. Autrement dit, savoir quand une instance doit être réutilisée, ou quand une nouvelle instance doit être créée.

Portées possibles d'un bean :

- **singleton** : une seule instance est créée pour l'ensemble de l'application, en général au lancement de l'application. Elle n'est détruite qu'à l'arrêt de l'application
- **prototype** : une instance est créée à chaque demande, elle est déréférencée dès que possible
- "pool" : un nombre fini d'instances sont créées. Elles sont fournies aux objets qui en ont besoin, ces derniers les libèrent lorsqu'ils sont détruits, ou qu'ils n'en ont plus besoin.

Portée

La portée définit la validité d'une instance en fonction du contexte. Autrement dit, savoir quand une instance doit être réutilisée, ou quand une nouvelle instance doit être créée.

Portées possibles d'un bean :

- **singleton** : une seule instance est créée pour l'ensemble de l'application, en général au lancement de l'application. Elle n'est détruite qu'à l'arrêt de l'application
- **prototype** : une instance est créée à chaque demande, elle est déréférencée dès que possible
- "pool" : un nombre fini d'instances sont créées. Elles sont fournies aux objets qui en ont besoin, ces derniers les libèrent lorsqu'ils sont détruits, ou qu'ils n'en ont plus besoin.
- *session / request* : portées spécifiques à un contexte WEB

Exemples de code

Sans gestion du cycle de vie :

```
1  // Instance de type singleton (2 solutions) :  
2  EmployeeNumberGenerator generator1 = EmployeeNumberGenerator.getInstance();  
3  EmployeeNumberGenerator generator2 = Factory.getEmployeeNumberGenerator();  
4  
5  // Instance de type prototype :  
6  PayReport report = new PayReport();
```

Exemples de code

Sans gestion du cycle de vie :

```
1 // Instance de type singleton (2 solutions) :  
2 EmployeeNumberGenerator generator1 = EmployeeNumberGenerator.getInstance();  
3 EmployeeNumberGenerator generator2 = Factory.getEmployeeNumberGenerator();  
4  
5 // Instance de type prototype :  
6 PayReport report = new PayReport();
```

Avec gestion du cycle de vie

```
1 // Demande au context du conteneur une instance du type voulu  
2 EmployeeNumberGenerator employeeNumberGenerator = applicationContext.getBean(  
    EmployeeNumberGenerator.class);
```


Exemples de code

Sans gestion du cycle de vie :

```
1 // Instance de type singleton (2 solutions) :  
2 EmployeeNumberGenerator generator1 = EmployeeNumberGenerator.getInstance();  
3 EmployeeNumberGenerator generator2 = Factory.getEmployeeNumberGenerator();  
4  
5 // Instance de type prototype :  
6 PayReport report = new PayReport();
```

Avec gestion du cycle de vie

```
1 // Demande au context du conteneur une instance du type voulu  
2 EmployeeNumberGenerator employeeNumberGenerator = applicationContext.getBean(  
    EmployeeNumberGenerator.class);
```

Conteneur

Ce n'est plus le code qui détermine si l'objet est un singleton ou un prototype, mais la **configuration du conteneur**.

Injection de dépendances

Dépendance

Certaines briques applicatives utilisent d'autres briques pour fonctionner. Ces briques nécessaires au fonctionnement sont appelées **dépendances**.

Injection de dépendances

Dépendance

Certaines briques applicatives utilisent d'autres briques pour fonctionner. Ces briques nécessaires au fonctionnement sont appelées **dépendances**.

Exemples

Le bean "*générateur d'emails*" a 1 dépendance : *DAO Employés*.

Injection de dépendances

Dépendance

Certaines briques applicatives utilisent d'autres briques pour fonctionner. Ces briques nécessaires au fonctionnement sont appelées **dépendances**.

Exemples

Le bean "*générateur d'emails*" a 1 dépendance : *DAO Employés*.

Le bean "*gestion des employés*" a 4 dépendances :


- *générateur de matricules*
- *générateur d'emails*
- *DAO Employés*
- *Connecteur Webservice RH*

Injecter une dépendance

À la création d'un bean, ses dépendances sont instanciées et lui sont affectées (injectées).

Injecter une dépendance

À la création d'un bean, ses dépendances sont instanciées et lui sont affectées (injectées).

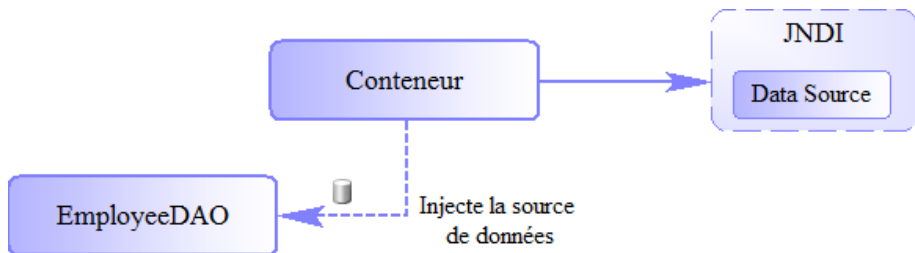
```
1 public class EmailGenerator {  
2  
3     private EmployeeDAO employeeDAO;  Injecte la dépendance  
4  
5     public void uneMethode() {  
6         // employeeDAO n'est PAS nul : il a ete injecte    la creation de l'  
7             EmailGenerator.  
8         employeeDAO.emailExists("un@email.com");  
9     }  
}
```

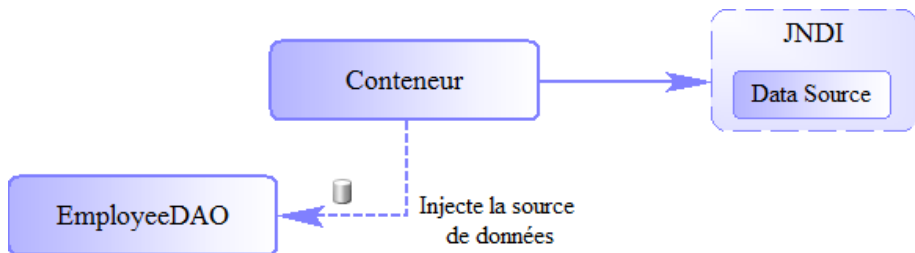
Gestion de la configuration

... par injection !

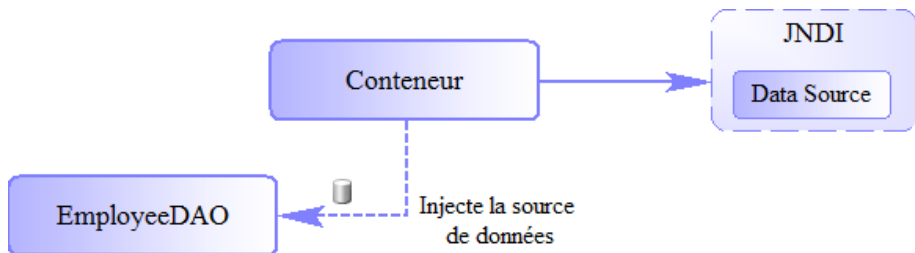
Injection de la configuration

La configuration est paramétrée de façon globale (fichiers properties, dictionnaire jndi, ...), et elle est distribuée à tous les beans qui en ont besoin.

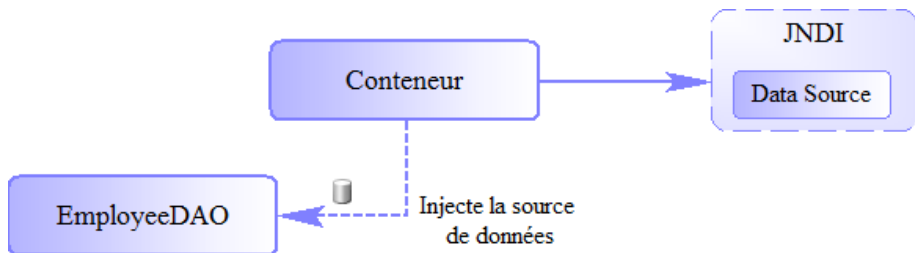




- **Couplage lâche** : EmployeeDAO n'a pas connaissance de la façon dont sont configurées les sources de données.



- **Couplage lâche** : EmployeeDAO n'a pas connaissance de la façon dont sont configurées les sources de données.
- *Gestion des portée* : la source de données injectée peut dépendre d'un contexte (pays).



- **Couplage lâche** : EmployeeDAO n'a pas connaissance de la façon dont sont configurées les sources de données.
- *Gestion des portée* : la source de données injectée peut dépendre d'un contexte (pays).
- *Cohérence* : toute l'application, voire même les applications, sont configurées de la même façon.

Environnements multiples

Différences d'environnements

Comment gérer les différences entre un environnement *WEB*, et *Batch* ?

Environnements multiples

Différences d'environnements

Comment gérer les différences entre un environnement *WEB*, et *Batch* ?

Alternatives

En remplaçant, ou ajoutant, un fichier de configuration à l'initiation du conteneur, il est possible de charger l'implémentation d'un bean.

Configuration dans un contexte WEB et Batch

Dans un environnement WEB, les sources de données sont configurées dans un dictionnaire JNDI. Mais dans l'environnement Batch, elles sont configurées dans un fichier.

Configuration dans un contexte WEB et Batch

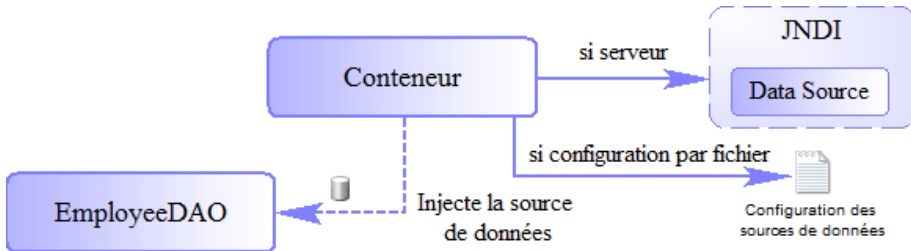
Dans un environnement WEB, les sources de données sont configurées dans un dictionnaire JNDI. Mais dans l'environnement Batch, elles sont configurées dans un fichier.

- Ajout d'une alternative : chargement de sources de données via un fichier.

Configuration dans un contexte WEB et Batch

Dans un environnement WEB, les sources de données sont configurées dans un dictionnaire JNDI. Mais dans l'environnement Batch, elles sont configurées dans un fichier.

- Ajout d'une alternative : chargement de sources de données via un fichier.



Sommaire

1 Séparation des préoccupations

2 Inversion de contrôle

3 Spring

- Définition
- Configuration du conteneur
- Annotations Spring
- Interfacier les beans

4 Conclusion

Spring = Conteneur léger !

Définitions ...

Spring

Spring est un conteneur léger.

Il se présente comme un ensemble de librairies à embarquer dans l'application. Il fournit l'*inversion de contrôle* : gestion du cycle de vie, injection des dépendances, injection de la configuration.

Spring = Conteneur léger !

Définitions ...

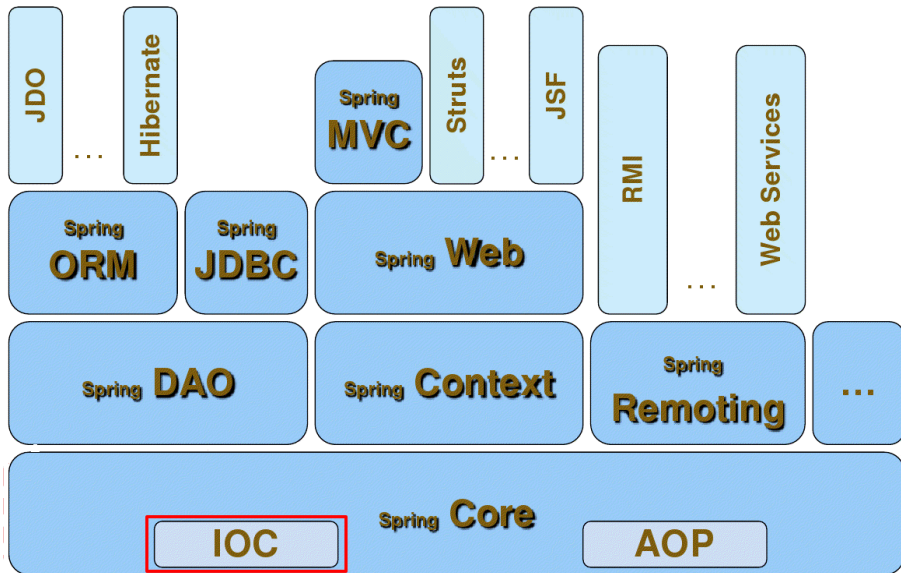
Spring

Spring est un conteneur léger.

Il se présente comme un ensemble de librairies à embarquer dans l'application. Il fournit l'*inversion de contrôle* : gestion du cycle de vie, injection des dépendances, injection de la configuration.

Mais pas que ...

Spring, la boîte à outils du développeur JAVA



Configuration du conteneur

Instanciation de l'*application context*

ApplicationContext

Le conteneur de *Spring* s'appelle `ApplicationContext`. Il est créé à partir d'un fichier de configuration XML.

Configuration du conteneur

Instanciation de l'*application context*

ApplicationContext

Le conteneur de *Spring* s'appelle `ApplicationContext`. Il est créé à partir d'un fichier de configuration XML.

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

Configuration du conteneur

Contenu du fichier XML

Fichier context.xml :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.
4      springframework.org/schema/context"
5      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
6      springframework.org/schema/beans/spring-beans.xsd
7      http://www.springframework.org/schema/context http://www.springframework.org/
8      schema/context/spring-context-3.1.xsd">
9
10     <!-- Active les annotations pour le package donne -->
11     <context:annotation-config />
12     <context:component-scan base-package="net.yvesrocher.tutorial.employees" />
13
14     <!-- Lecture des fichiers de proprietes (tous les fichiers dans config) -->
15     <context:property-placeholder location="classpath:config/*.properties"/>
16
17 </beans>
```

Déclarer un bean

@Named

Les beans sont déclarés par l'annotation `@Named` placé au dessus de la classe.

Déclarer un bean

@Named

Les beans sont déclarés par l'annotation `@Named` placé au dessus de la classe.

```
1  @Named
2  public class EmployeeManager {
3  }
```

Limiter la portée d'un bean

@Scope

L'annotation @Scope définit la portée du bean. La portée par défaut avec *Spring* est singleton

Limiter la portée d'un bean

@Scope

L'annotation @Scope définit la portée du bean. La portée par défaut avec *Spring* est singleton

```
1  @Named
2  @Scope("singleton")
3  public class EmployeeManager {
4  }
```

Limiter la portée d'un bean

@Scope

L'annotation @Scope définit la portée du bean. La portée par défaut avec *Spring* est singleton

```
1  @Named
2  @Scope("singleton")
3  public class EmployeeManager {
4  }
```

Rappels :

Singleton : une seule instance est créée pour toute l'application.

Prototype : une instance est créée à chaque demande

Injecter une dépendance

@Inject

L'annotation `@Inject` déclare une dépendance à *Spring*. Le conteneur cherche un bean du même type.

Injecter une dépendance

@Inject

L'annotation `@Inject` déclare une dépendance à *Spring*. Le conteneur cherche un bean du même type.

```
1  @Named
2  public class EmployeeManager {
3
4      @Inject
5      private EmployeeNumberGenerator employeeNumberGenerator;
6  }
```

Injecter la configuration

À partir d'un fichier de propriétés

@Value

L'annotation `@Value`, couplée à l'annotation `@Inject`, recherche dans les fichiers de propriété la valeur.

Injecter la configuration

À partir d'un fichier de propriétés

@Value

L'annotation `@Value`, couplée à l'annotation `@Inject`, recherche dans les fichiers de propriété la valeur.

```
1  @Named
2  public class EmailGenerator {
3
4      /** Suffixe a utiliser pour les externes */
5      @Inject
6      @Value("generators.email.externalSufix")
7      private String sufix;
8  }
```


Utilisation d'interfaces

Limiter le couplage au maximum

Interface Une interface *est un contrat* décrivant les signatures méthodes (nom, paramètres d'entrée et de retour).

Utilisation d'interfaces

Limiter le couplage au maximum

Interface Une interface *est un contrat* décrivant les signatures méthodes (nom, paramètres d'entrée et de retour).

Implementation Classes réalisant les méthodes définies dans une ou plusieurs interfaces.

Utilisation d'interfaces

Limiter le couplage au maximum

Interface Une interface *est un contrat* décrivant les signatures méthodes (nom, paramètres d'entrée et de retour).

Implementation Classes réalisant les méthodes définies dans une ou plusieurs interfaces.

Utilisation d'interfaces

Limiter le couplage au maximum

Interface Une interface *est un contrat* décrivant les signatures méthodes (nom, paramètres d'entrée et de retour).

Implementation Classes réalisant les méthodes définies dans une ou plusieurs interfaces.

Pourquoi interfacer tous les beans ?

L'objectif de l'inversion de contrôle est de limiter le couplage (connaissance) entre un bean et ses dépendances. En utilisant une interface, le couplage est à son minimum : l'implémentation utilisée, sa configuration et sa portée sont totalement indépendants.

Conventions de nommage

Pour s'y retrouver plus facilement

Par convention, il est d'usage de :

- Préfixer les interfaces par un "i" majuscule : `IEmployeeDAO`.
- Nommer les implémentations de la même façon que l'interface (sans le préfixe), et d'y suffixer "*Impl*".

Conventions de nommage

Pour s'y retrouver plus facilement

Par convention, il est d'usage de :

- Préfixer les interfaces par un "i" majuscule : `IEmployeeDAO`.
- Nommer les implémentations de la même façon que l'interface (sans le préfixe), et d'y suffixer "*Impl*".

```
1  public interface IEmployeeDAO {
2
3      /** Sauvegarde l'employe */
4      void saveEmployee(Employee employee);
5
6      /** Verifie si l'email est deja utilise */
7      boolean emailExists(String email);
8  }
9
10 @Named
11 public class EmployeeDAOImpl implements IEmployeeDAO {
12     /* les methodes de l'interface sont retrouvees ici. */
13 }
```

Sommaire

- 1 Séparation des préoccupations
- 2 Inversion de contrôle
- 3 Spring
- 4 Conclusion**

Conclusion

Ce qu'il faut retenir...

Inversion de contrôle (IoC)

Sans inversion de contrôle, chaque bean a la charge de créer ses dépendances, de les configurer et de lire sa propre configuration.

Avec, il ne fait que déclarer ce dont il a besoin. Le *conteneur* (*Spring*) lui injectera.

Conclusion

Ce qu'il faut retenir...

Inversion de contrôle (IoC)

Sans inversion de contrôle, chaque bean a la charge de créer ses dépendances, de les configurer et de lire sa propre configuration.

Avec, il ne fait que déclarer ce dont il a besoin. Le *conteneur* (*Spring*) lui injectera.

Annotations à retenir :

@Named Déclare la classe comme étant un bean géré par *Spring*

@Scope Définit la portée d'un bean (défaut : singleton)

@Inject Déclare une dépendance à injecter

@Value Recherche une valeur dans un fichier de propriété

Merci, des questions ?