Spring in a nutshell

Sébastien Chassande-Barrioz
sebastien.chassande-barrioz@cgi.com

CGI

http://tinyurl.com/formation-ecom

Agenda

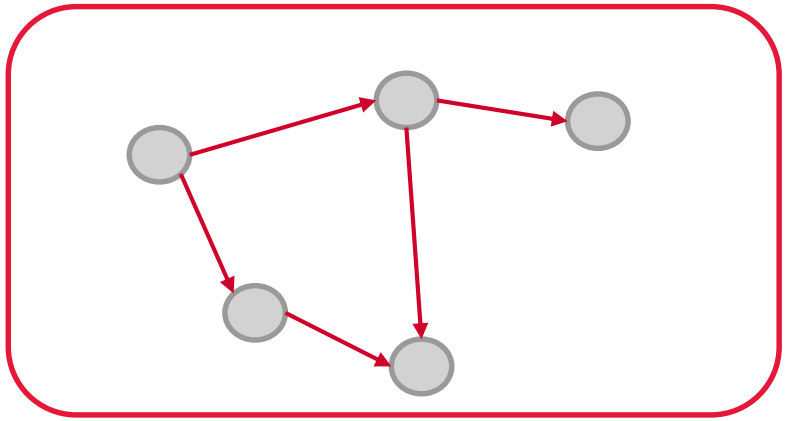
1. Introduction à Spring
2. Concepts et organisation du Framework
3. Configuration de Spring
4. Les annotations les plus répandues
5. Les projets de Spring

1. Introduction : You said Spring?

Spring est un « conteneur léger » c'est à dire un framework censé offrir les mêmes fonctionnalités qu'un serveur d'applications JEE, mais sans la lourdeur d'utilisation des API JEE

Spring gère le cycle de vie des objets (création, service, destruction)

Le noyau de Spring est basé sur une factory de beans (POJOS), et un conteneur capable de stocker ces beans



A container with 5 spring beans

2. CONCEPTS et organisation du framework : Beans spring

- Composants / Objets gérés par Spring

- Défini par :
  - Une classe dont le nom est pleinement qualifié
    Exemple : com.cgi.formation.app.FormationApp
  - Un identifiant unique dans le conteneur
  - Des paramètres de configuration optionnels
    (scope, paramètre de construction, dépendances, …)

- Cycle de vie géré par Spring

- Accessible directement ou indirectement par une Factory Spring

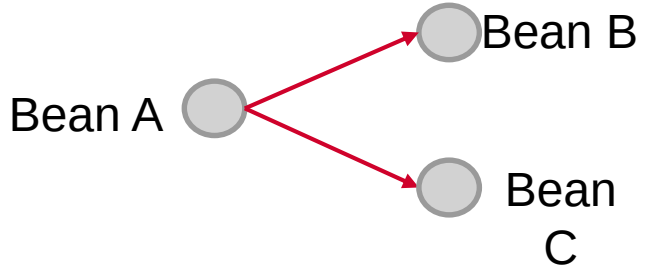
2. Concepts et organisation du framework : Les principes de Spring : Ioc ,DI, AOP

IOC = Inversion Of Control

- « Ne nous appelez pas, c'est nous qui vous appellerons »

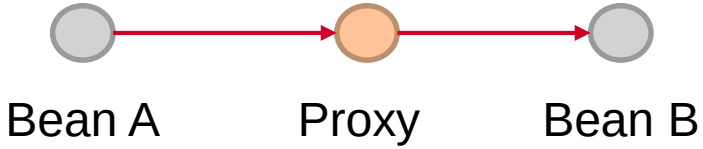
Dependancy Injection

- Découpler les dépendances



AOP = Apect Oriented Programmation

- Utilisation de proxy



Mot d'ordre : déléguer !

- POJOS

Spring framework

Configuration de spring

3. Configuration de spring

Le paramétrage de Spring permet :

- Déclaration des beans
- Paramétrage des beans
- Définition des packages à scanner
- Import d'autres fichiers de configuration
- Définition d'autres contextes
- Utilisation de profils

Spring est configuré au démarrage de l'application

- Dans une appli web : web.xml ➔ fichier de config xml Spring

3. Configuration de spring : 3 approches de configuration possibles

Dans le fichier de config spring context.xml

```
<bean id="radiateur"
    class="com.cgi.formation.appareil.AppareilThermostate" >
      <property name="nom" value="radiateur" />
      <property name="marque" value="ChauffeFort" />
      <property name="modele" value="King" />
      <property name="puissance" value="12" />
      <property name="thermostat" value="0" />
</bean>
```

Par des annotations sur la classe

<context:annotation-config> (dans le ficher spring)

```
@Component
public class AppareilThermostate
        extends AppareilElectrique implements Variator {
    …
}
```

Une classe de configuration

```
@Configuration
public class VariatorConfiguration {
    @Bean(name = "otherradian")
    public Variator radiateur() {
        return new AppareilThermostate("Radiateur", "Radian", 1000, 0);
    }
}
```

3. Configuration de spring : web.xml

Sélectionner le profile

```
<context-param>
     <param-name>spring.profiles.active</param-name>
     <param-value>profile1,profile2</param-value>
</context-param>
```

Initialiser Spring

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

Déclaration des fichiers de configuration

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath:/spring/context.xml,classpath:/spring/context2.xml
    <param-value>
</context-param>
```

3. Configuration de spring : Injection de dépendances

ProductDao.java

```
public class ProductDao {
    @Getter
    @Setter
    private List<Product> allProducts;

    public ProductDao() {
        // connect to DB and set allProducts var
    }
}
```

ProductService.java

```
public class ProductService {
    @Getter
    @Setter
    private ProductDao productDao;

    public ProductService(final ProductDao dao) {
        this.productDao = dao;
    }
}
```

Context.xml

```
<bean id="productDao" class="com.application.dao.ProductDao"/>

<bean id="productService" class="com.application.service.productService">
    <property name="productDao" ref="productDao"/>
</bean>
```

Note: @Getter et @Setter sont des annotations Lombok pour éviter d'écrire les getter et setter

3. Configuration de spring : Séquence au démarrage

1. Création des beans avec le constructeur vide

2. Injection des propriétés (utilisation des setters en XML, ou injection directe sur attribut si annotation)

3. Injection des autres beans (setters ou direct)

4. Déclanchement de la méthode initialisation (si paramétrée)
   ➔ le bean est alors en mode « service »

5. Déclanchement de la méthode de nettoyage si Spring est averti explicitement par :
   `context.close()`

3. Configuration de spring : Modèle de conteneur Spring

Le modèle le plus utilisé (celui de Spring par défaut) est le modèle **Singleton**

- Dure le temps de l'application
- Multithread non synchronisé
- Il faut donc coder avec le pattern stateless

D'autres modèles dédiés à des contextes particuliers existent :

- Prototype : une nouvelle instance chaque fois que nécessaire
- Request : dure le temps d'une requête web
- Session : dure le temps d'une session
- Job : dure le temps de l'exécution d'un job Spring Batch
- Step : dure le temps de l'exécution d'une step d'un job Spring Batch

Spring framework

4. Les annotations les plus répandues

4. Les annotations les plus répandues

- Spring s'utilise beaucoup via des annotations dans le code Java
- Présentation des principales annotations :
  - **@Component** : déclare un bean (générique)
  - **@Repository** : déclare un bean d'accès aux données
  - **@Service** : déclare un service métier
  - **@Autowired /@Inject** : Permet d'injecter des dépendances vers d'autres bean
  - **@Qualifier /@Named** : Permet de qualifier la dépendance
  - **@PostConstruct / @PreDestroy** : permet de tagger une méthode pour être appelée au moment de la construction/destruction du bean
  - **@Transactional** : indique que la méthode ou toutes les méthodes de la classe doivent être transactionnelles

4. Les annotations les plus répandues : @Repository (niveau classe)

- Annotation qui permet de définir le DAO (couche de persistance)
- Possibilité de définir la portée du bean (@Scope)
  - Rappel : Singleton, prototype, request, session…
- Un seul paramètre possible : le nom
- Utilisation de l'annotation @Repository
  - Interface définie sans annotation
  - Implémentation portant l'annotation @Repository

```
public interface ProductDao extends GenericDao<Product> {
   …
}
@Repository
public final class ProductDaoImpl extends GenericDaoImpl<Product> implements ProductDao {
   …
}
```

4. Les annotations les plus répandues : @service (niveau classe)

- Annotation qui permet de définir un Service (couche Métier)
- Possibilité de définir la portée du bean (@Scope)
  - Rappel : Singleton, prototype, request, session…
- Un seul paramètre possible : le nom
- Utilisation de l'annotation @Service
  - Interface définie sans annotation
  - Implémentation portant l'annotation @Service
- Appelle le DAO (Repository)

```
public interface ProductService {
   …
}
@Service
public final class ProductServiceImpl implements ProductService {
   …
}
```

4. Les annotations les plus répandues : @Component (niveau classe)

Permet de définir un bean sans contrat d’interface

Peut être utilisé par exemple pour une classe de type utils

```
@Component
public class CountryCodeUtil {
    …
}
```

4. Les annotations les plus répandues : @AUTOWIRED (ou @INJECT)

- Permet l’injection de Beans
- Peut être utilisé avec l’annotation @Qualifier / @Named
- @Autowired peut se placer
  - sur un attribut
  - sur le setter d’un attribut
  - sur le constructeur d’une classe
- Attribut « required=false » permet au bean de fonctionner même sans l’attribut injecté
- Ordre de recherche d’un bean :
  - Par type
  - Par qualifier
  - Par nom

4. Les annotations les plus répandues : @Autowired (Ou @Inject) exemples

Sur une propriété

```
@Service
public class ProductServiceImpl
       implements ProductService{

    @Autowired
    private ProductDao productDao;
}
```

Sur un constructeur

```
@Service
public class ProductServiceImpl
       implements ProductService{

    private ProductDao productDao;

    @Autowired
    public ProductServiceImpl(ProductDao productDao) {
        this.productDao = productDao;
    }
}
```

Sur un setter

```
@Service
public class ProductServiceImpl
       implements ProductService{

    private ProductDao productDao;

    @Autowired
    public void setProductDao(ProductDao productDao) {
        this.productDao = productDao;
    }
    public ProductDao getProductDao() {
        return productDao;
    }
}
```

4. Les annotations les plus répandues: @Qualifier (ou @NAMED)

- Permet de qualifier un bean
  - Donner un nom différent à des bean qui implémentent la même interface

- Annotation à définir à la déclaration d'un bean mais également lors de son injection

  Déclaration :

```
@Service
@Qualifier ("productService1")
public final class ProductServiceImpl implements ProductService {
   …
}
```

  Injection :

```
@Autowired
@Qualifier("productService1")
 private ProductService productService;
```

4. Les annotations les plus répandues @Postcontruct/@predestroy (niveau Méthode)

- Méthode pour interagir pendant le cycle de vie d'un bean
- @PostConstruct : Avant que le bean passe en mode « service » (Juste après que Spring ait créé le bean et injecté les dépendances)
- @PreDestroy : Juste après que le bean ait terminé son mode « service » (context close)

```
@Service
public final class ProductServiceImpl implements ProductService{

    @PostConstruct
    public void init() {
        ...
    }
}
```

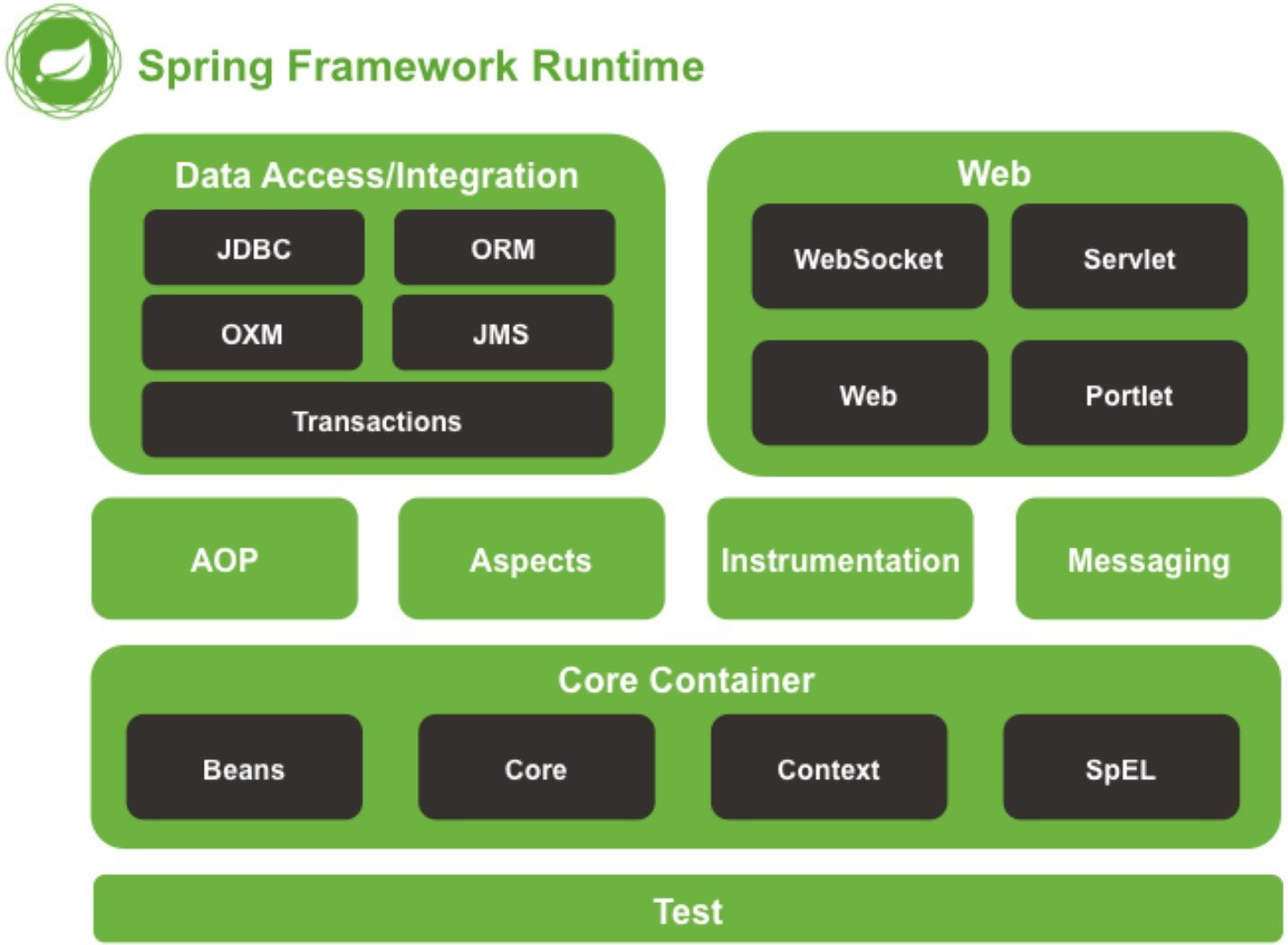
4. Les annotations les plus répandues
@Transactional (niveau Classe ou méthode)

- Permet de délimiter une transaction (Début & fin)
  - Suite d'opération faisant passer un objet / une donnée d'un état A à un état B
- Plusieurs transaction peuvent être imbriquées
- Propagation de la transaction (attribut « propagation »)
  - **MANDATORY** : Erreur si aucune transaction en cours
  - **NESTED** : S'exécute dans la transaction si possible
  - **NEVER** : Erreur si une transaction est en cours
  - **NOT_SUPPORTED** : Suspend la transaction
  - **REQUIRED** (défaut) : Utilise la transaction courante ou en créé une
  - **REQUIRES_NEW** : Créé une nouvelle transaction et suspend l'actuelle
  - **SUPPORTS** : S'exécute dans la transaction actuelle si possible
- Définir la lecture seule (readOnly) dans une méthode comme findAll

Spring framework

5. Les autres projets spring

Organisation de spring framework



SPRING DATA

Accès simplifié aux données dans une base relationnelle, non-relationelle ou encore map-reduce

SPRING INTEGRATION

Support du pattern EIP (Entreprise Integration Patterns) via des « messages » et des « adapteurs » (équivalent Apache Camel)

SPRING BATCH

Création facilitée de batch (traitement en lot), et de tâches planifiées

SPRING HATEOAS

Création simplifiée d’une API REST

SPRING WEB SERVICES

Création simplifiée de Web Service SOAP

SPRING SECURITY

Authentification et autorisation des utilisateurs

SPRING BOOT

Création simplifiée et rapide d’application Spring runnable

SPRING IO

Gestion des dépendances simplifiée (pas de gestion manuelle des versions)

Le TP

Le TP

- Télécharger le zip du TP : tp-spring-core.zip et le readme.md depuis htts://tinyurl.com/formationEcom, dans 1-Spring

- Dézipper le zip ..

- Ouvrir dans votre IDE : Eclipse, IntelliJ, …

- Lire le readme.md pour savoir ce qu'il faut faire

Bon TP !