Cours Spring

1 Présentation Générale

- **Historique**

  Framework créé en 2002 par Rod Johnson, afin de proposer une réelle alternative technique aux EJB's, alors gouffre de performance en terme d'applications serveur.

  Les mots d'ordre sont :

  - Design Orienté Objet
    Utiliser les standards en terme de conception objets, et notamment le respect du tryptique : interface, classe abstraite, classe(s) concrête(s).
  - Simplicité d'utilisation
    Ne pas complexifier le cadre de développement applicatif, comme ça a été le cas avec les EJB
  - Faible dépendance
    L'intégration de Spring doit être la plus transparente possible dans les applications qui l'utilisent.
  - Facilité d'intégration avec des outils externes
    Permettre aux applications Spring de fonctionner en se câblant facilement avec des outils et librairies additionnelles
  - Respect de l'existant
    Spring ne cherche pas à ré-implémenter à sa sauce des frameworks fonctionnels déjà existant (comme l'ORM Hibernate par ex)
  - Gestion de problématiques transverses
    Comme le traitement des exceptions, ou bien la mise en place d'Aspects, permettant d'épurer une partie du code.

- Spring utilise certaines technologies comme :

  - XML Schema (cf fichier XML de configuration du TP),
  - les @nnotations (apparues avec Java SE 5.0, très pratiques pour enrichir simplement l'environnement et les méta informations de classes).
  - Un « Expression Language », caractérisé par « #{*expression*} », qui permet :
    - De manipuler des références d'objets
    - D'effectuer des appels de méthodes/propriété d'une classe/objets.
    - D'appeler des fonctions (regexp, gestion de tableaux, ...) sur des valeurs littérales.

- **Concepts**

  - Contexte

  A la différence des containers J2EE d'EJB qualifiés de « *lourds* » (de par les contraintes serveurs imposées), l'utilisation de Spring permet la mis en place d'un environnement appelé « *Context ».*

  -> Principe de « *Containers légers* », déployables dans un simple container de Servlet (Tomcat par ex), régissant les processus fondamentaux du chargement/démarrage d'une population d'objets au sein d'une application.

  Interfaces principales :
  - *BeanFactory*
    Ex : *XMLBeanFactory* : définition des beans dans un fichier XML
  - *ApplicationContext* (Hérite de *BeanFactory*)
    Ex : *ClassPathApplicationContext* : application standalone
    WebApplicationContext : application web

http://docs.spring.io/spring/docs/3.0.x/api/org/springframework/beans/factory/BeanFactory.html
http://docs.spring.io/spring/docs/3.0.x/api/org/springframework/context/ApplicationContext.html

  - Archétypage des classes

  Le démarrage du contexte Spring commence par l'introspection des structures et comportement des classes + interfaces pris en charge, ainsi que la définition des règles ultérieures afférentes à leur bon fonctionnement.

  Ces modèles « idéaux » sont utilisés ensuite, lorsque pour chaque demande émanant de l'application, le contexte Spring doit créer et assembler les objets, afin de les rendre disponibles pour la partie appelante.

  - Cycle de Vie d'objets

  Prise en charge de la politique [d'instanciation, de construction] des objets (Design Pattern *Factory*, famille « *Création* »),
  par appels réflexifs
  (cf package d'abstraction *java.lang.reflect*)
  vers :

  - Un *Constructor* (avec ses éventuels paramètres)
    ➔ gestion des dépendances (obligatoires ou optionnelles) lors de la construction d'un objet !
  - La méthode de classe standardisée statique *newInstance()*, qui permet de construire un objet à partir de son constructeur « vierge ».

- Scope

Spring offre un ensemble de « *portées* » dans lesquelles seront situés/logés les objets de classes nouvellement créés.

Ces scopes regroupent à la fois des stratégies d'instanciation relatives :

- Aux Design Pattern associées (famille « *Création* ») :
  - *Singleton* : une seule instance disponible par application.
  - *Prototype* : création par clonage, pour chaque appel.
- A l'environnement Web
  - *Request :* à la requête
  - *Session*: par utilisateur
  - *Global session* : « only makes sense in the context of portlet-based web applications »

http://docs.spring.io/spring/docs/3.0.0.M3/reference/html/ch04s04.html

- Composition

En visualisation « macro » (dénuée des types réels - concrets ou abstraits – s'y rapportant), on considère que tout objet issu d'une classe est « *composé* » de propriétés, (Design Pattern *Composite*, famille « *Structuration* »)

(Rappel : en Java, les « *Membres* » d'une classe = propriétés + méthodes).

L'« *injection de dépendances* » prône que ces dernières soient définies / assemblées à l'extérieur de la classe concernée, depuis le Contexte.

➔ Evite d'instancier les propriétés/dépendances, directement dans le corps de classe, ce qui le complique/l'alourdit, car c'est le rôle du Contexte !

C'est l'« *IoC - Inversion of Control* »,
ou « *Hollywood Principle - don't call us, we'll call you !* »)

➔ Le Contexte peut être défini par fichier(s) XML de configuration, ou bien par *@nnotation @Configuration* dans le une/des classe(s) Java dédiée(s).

- Spring se fonde sur le développement et la définition de services selon la programmation par « *Contrat d'Interface* » :

- Faible couplage

Permet d'avoir des dépendances entres références typées/signées selon une interface, plutôt qu'un type concret, plus lourd à remanier lors de refactoring de code.

(Pour rappel, le faible couplage est un standard de développement en Java, cf implémentation du JDK :
ex: Interface *Map*, Classe Abstraite *AbstractMap*, classe concrête *TreeMap*).

- Spring à eu un impact notable sur la normalisation des architectures J2EE, de par sa mécanique découpage et d'assemblage de composants.

  La politique de programmation par interface s'inscrit dans la lignée de la structuration et la standardisation des modèles macro d'architecture d'application J2EE, qui divise sous forme de couche/layer les strates suivantes:

  - Présentation

    La *View* (cf MVC2) : composants web (JSF & facelets)

    Complétée par des packages divers associés aux traitements web :

    - Controler : Coordination des appels propriétés des pages
    - Validator : Validation des données échangées lors d'appels.
    - Converter : Conversion entre données brutes et entités DB.
    - ...

  - Business:

    Façade (cf Design Pattern « *Facade* », famille de « *Structuration* ») fédérant l'ensemble des « *Services* » métiers.

    Elle se charge de :

    - Masquer la complexité des accès aux services, qui peuvent être distribués sur une topologie réseau.
    - Valider les appels entrants par des règles métiers qui sont logés dans les classes de cette couche.
    - Convertir au besoin des Exceptions *Techniques* en Exception *Métiers,* afin que la partie cliente ne remonte pas dans la View des problèmes techniques.

      (on souhaite éviter par ex, d'avoir des *SQLException* avec le code SQL associé remonté au client, on préfère avoir une *BOException* plus générique « un problème est survenu, code XXX, contactez votre administrateur).

    - Services

      Ensemble des briques de services accédé par la partie cliente Business :

      - DAO (*Data Access Object*) : Accès aux données
      - Mail
      - Reporting
      - ...

Remarque : Cette structuration par couche Présentation/Métier/Intégration, est celle donnée par les recommandations de Sun/Oracle pour l'implémentation d'application J2EE.

http://www.oracle.com/technetwork/java/index-138725.html

- Spring, comme JSF ou Hibernate favorise le modèle POJO (« *Plain Old Java Objects »)* : un package de modèle comportant des java beans simples, sans aucune référence à des interfaces/Framework externes.

  Cela permet, en effet, de conserver un design de classe « peu intrusif » en terme de dépendances (le code applicatif ne requiert pas l'API Spring pour être défini).

  Remarque : On verra que dans le cadre d'une approche d'implémentation « programmative » par @nnotations, cette règle initialement reconnue et partagée unanimement, a quelque peu subit des transgressions dans les méthodologies d'implémentation, et que l'on se trouve dorénavant avec un modèle hybride, ou l'on a des dépendances existantes vers des @nnotations (par exemple EJB Entity JPA), mais aucune dans le code réel.

- Spring est très bien intégré dans les architectures contemporaines, J2SE + J2EE (énormément pour cette dernière…), ainsi que vis à vis de très nombreux frameworks (Hibernate, JPA, par exemple, pour tout ce qui a trait aux ressources et méthodes propres à la persistance de données – via JDBC).

- Spring dispose de nombreux modules qui composent le Framework :

  - Core (noyau de l'*IoC :* Factory & injection),
  - Context (i18N, LifeCycle, JNDI, intégration EJB, ...)
  - AOP (développement d'aspects, support AspectJ),
  - DAO (accès JDBC pure),
  - ORM (solutions de mapping : Hibernate, IBatis, JPA, JDO, ...),
  - JMX (Monitoring de l'exécution d'une application, ex : Metrics)
  - JCA (Connecteur d'intégration vers solutions tiers: ERP, CRM, ...)
  - JMS (Gestion de Messages asynchrones, pub/sub, topics)
  - Remote (Accès de Bean distant : RMI, JAX-RPC, JAX-WS)

  Ainsi que différents projets qui utilisent le noyau Spring et ses modules :

  - Spring WebFlow (flow HTTP type Wizard),
  - Spring MVC (alternative à JSF, ou Struts, pour du),
  - WebServices (bridge d'invocation)
  - Security (gestion de la sécurité),
  - Batch (gestion de processus, avec pagination),
  - Transaction (cycle de vie transactionnel, par @nnotation)
  - LDAP (accès d'annuaires),
  - IDE (plugin d'intégration dans Eclipse),
  - Test
  - …

2 Contenu du TP

- Migration du TP 8 JSF + Reporting afin d'intégrer un contexte Spring :
    - Changements appliqués dans web.xml & faces-config.xml.
- Exemple de données « brutes » manipulables par un Context Spring (*contextDataExemple.xml*)
- Evolution de la définition du contexte Spring :
    - *contextWeb.xml*
      Transfert de la définition des beans depuis le *faces-config.xml*
    - *contextWebAuto.xml*
      Définition de l'injection par *autowiring* (auto-cablage) à travers les règles spécifiées dans le fichier de *context XML*.
    - *contextWebAutoWithAnnotations.xml*
      Définition de l'injection par *autowiring* à travers des @*nnotations* sur des propriétés de classes (modifications dans le *context XML*, ainsi que dans les champs de classes Java)
    - *contextWebAutoWithAnnotationsAndDetection.xml*
      Déclaration d'un *component-scan* dans le *context* XML, en charge de découvrir les annotations sur les classes déclarantes.

      Automatisation de la création et de l'injection des beans à travers un deuxième niveau d'@*nnotations,* au niveau de la classe + Scope (*Session, Singleton, Prototype*).

Remarques :

- Les configurations par classe, utilisant @Configuration, ne sont pas présentées ici, le médium de configuration XML leur est préféré et est bien plus rependu, par ailleurs, il correspond à la « convention » de configuration par fichier.

- Mise en place de Stéréotypes d'@nnotations par famille :

@*Component* type générique pour tout type de composant ;
@*Controller* (Web MVC), @*Service* (Business / back end services), @*Repository* (DAO), sont des spécialisations d'@*Component.*

Il est conseillé de spécialisé ses @nnotations de classes afin d'avoir une ganularité d'information plus fine sur les registres d'instances manipulés par un context ; peut aussi servir dans le ciblage de pointcuts par AOP (cf cours AOP...).

@Repository prend en charge les Exceptions de la couche de persistance.

- CDI (« *Context Data Injection* », plateforme J2EE) :

Selon la doc Spring, les @nnotations *@Inject/@Named* de CDI peuvent être utilisées en remplacement de celles Spring *@Autowired/@Qualifier*. Un exemple est donné dans la classe *ProductEditAnnotatedAutomation*.

pour plus d'informations sur CDI :
http://docs.oracle.com/javaee/6/tutorial/doc/giwhl.html

- Les extensions/particularités apportées par Spring 4 ne sont pas vues ici :
support de Java8 : lambda expression, support de java time (JSR 310), debug, ...
http://docs.spring.io/spring/docs/current/spring-framework-reference/html/new-in-4.0.html

- Liens et Références Spring :

Documentation officielle, et complète, de Spring sur toutes les fonctionnalités et comportements mis en place dans un Container d'IoC:
http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/beans.html