Notion de Spring JPA (BackEND JEE)

Table of Contents

De nombreuses applications Web se composent de plusieurs «couches» également appelées «niveaux» de l'application: niveau web ou niveau de présentation pour afficher les pages de votre application, niveau commercial ou niveau intermédiaire pour l'exécution des règles logiques et commerciales de votre application et de votre niveau de données , Ou persistece pour transférer des données vers / depuis votre base de données. Ces niveaux peuvent avoir la configuration suivante:

Entity

Les classes qui contiennent des données dérivées de votre base de données et les plus utilisées de manière plausible par un framework ORM comme Hibernate;

DAO

Les classes qui seront utilisées pour accéder à la base de données et au moins effectuer des opérations CRUD sur la base de données et surtout pour vos classes d'entités de retour de partie Web pour votre niveau web.

Service

```
Les cours qui reflètent les opérations commerciales que la demande prévoit;
Bean Classes qui sauvegarderont vos vues et contiendront probablement des données, des
méthodes d'action, des transformations, etc., utilisés dans vos pages Web.
La prochaine étape est le choix du framework pour votre application Web.
```

Vous choisissez Spring pour tous les calques, ce qui signifie que vos DAO seront des @Repository cours, vos Services seront des @Servicecours et vos Beans seront des @Componentcours. Vous utiliserez probablement un cadre ORM comme Hibernate pour gérer la base de données, de sorte que vos Entités seront des @Entityclasses JPA correctement configurées dans le style Hibernate. Votre technologie de vision sera probablement Spring MVC qui a été élaborée pour fonctionner avec Spring core. Par exemple, Mkyong a de nombreux tutoriels simples sur l'utilisation de Spring.

Vous choisissez le framework JSF + EJB natif pour tous les calques, ce qui signifie que vos DAO et vos services seront des @EJBclasses, vos beans seront des @ManagedBeanclasses. Vous utiliserez probablement très probablement Hibernate en tant que solution ORM et fournisseur JPA et fera l'accès à la base de données via EntityManager. Votre technologie de visualisation sera JSF car elle était naturellement destinée à être utilisée avec les technologies susmentionnées. Par exemple, BalusC possède de nombreux tutoriels éclairés sur l'utilisation de JSF.
Les deux choix ont ses défenseurs et ses adversaires. Certains disent que pourquoi choisir quelque chose qui n'est pas natif de la solution Oracle d' Oracle, d'autres disent qu'il est trop complexe et confus et qui manque de sources à apprendre.

Comme ce n'est pas un problème sur le choix de la technologie, je ne vais pas entrer dans les détails ici, mais soulignerai que Spring est un conteneur léger qui fonctionnera sur des conteneurs de servlets simples comme Tomcat alors que les EJB ont besoin d'un serveur d'applications comme Glassfish pour s'exécuter. Je pense que c'est la principale force motrice pour combiner JSF en tant que cadre Web basé sur les composants et Spring comme une injection de dépendance légère et un cadre de niveau commercial.

Comme nous avons décidé d'intégrer les deux cadres ensemble, je vais expliquer comment l'intégration fonctionne et pourquoi les NPE se produisent.

```
Les classes d'entité seront soit des classes annotées JPA / Hibernate, soit des POJO
simples configurés par xml.
Les DAO mettront en @Repositoryplace des interfaces de base pour éviter un couplage
étanche. Ils seront gérés par le cadre de printemps.
Les services mettront @Serviceégalement en place des interfaces de base. Ils seront
également gérés par le cadre du printemps. Notez que Spring Framework offrira une
gestion de transaction hors connexion pour vous si vous marquez les méthodes de
service @Transactional.
Les haricots doivent donc être @Componentet @Scope("value")être gérés par Spring si
vous souhaitez l'utiliser comme un cadre d'injection de dépendance, permettant
d'accéder à vos services et autres beans via @Autowired.
Ainsi, le NPE provient du malentendu que vos beans, en tant que partie logique de la
vue, devraient être gérés par JSF (notez que @ManagedPropertycela ne fonctionnerait
pas aussi). Le bean est instancié par JSF, mais votre service réside dans un contexte
de printemps sur lequel JSF sait bien, ce qui rend l'injection impossible. D'autre
part, si le haricot reste dans le contexte de Spring, son cycle de vie et ses
dépendances seront injectés par Spring.
```

Spring Data

WARNING	comment configurer Spring avec HSQLDB?

HSQL DB est embarqué

C'est une base de données embarqué dans Spring. Il faut juste la configurer. C'est pratique pour une petite application en local au lieu d'utiliser Mysql, Oracle, postgres ….

Exemple: on va creer une base de données avec HSQLDB

1) dans Maven, declerer la dépendance org.hsqldb. 2) creer un fichier .sql pour la creation et insertion de données dans la table. 3) dans Spring - declarer un bean configuration de la base de données, c'est à dire, `class="org.springframework.jdbc.datasource.DriverManagerDataSource"` et les `property name=`, driverClassName, url, username, password. - declarer <jdbc:initialize-database data-source="xxx">, <jdbc:script location="">,

Hibernate

WARNING	C'est quoi Hibernate?

- Hibernate est un ORM, un fornisseur qui implémente l'interface JPA.

WARNING	Qu'est ce que JPA?

- JPA ou Java Persistance API est une interfaces pour correlier Java POJO object avec les tabldes d'une base de données. On peut utiliser JPA ave des annotations ou une configuration XML.

Configuration de Hibernate

Afin de configurer Hibernate, il faut créer dans application-context-persistence.xml un `entity manager factory`. La différence entre HSQLDB c'est qu'Hibernate doit créer unr entité manager factory et une persistence context, le bean est `<bean class= "org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"`