Modern Entreprise Java Architecture with Spring 4.1

Date : 9 avril 2015
Format : Conférence
Speakers : Juergen Hoeler, Spring Framework lead chez Pivotal

Juergen est l'un des papas du framework Spring. Il travaille dessus depuis 12 ans.
Dans cette session, Juergen aborde l'état de l'art de l'architecture des applications d'entreprise développées en Java. Il énumère ensuite les fonctionnalités apportées par les versions 4.0 et 4.1 de Spring.

Le framework Spring 4.1 est adapté aux architectures modernes :

- Embedded web servers
- Non-traditionnal datastores
- Lightweight messaging et Websocket

Pour l'utiliser, le minimum requis est : Java 6, Servlet 2.5 et JPA 2.0
Juergen rappelle que Google App Engine fonctionne toujours avec Servlet 2.5.
Bien que compatible avec des versions relativement anciennes, Spring s'adapte en fonction de l'architecture et de l'infrastructure applicative afin d'utiliser les dernières fonctionnalités : Servlet 3, Java 8, Websocket …

Spring 4.1 supporte de nombreuses spécifications de JEE 7 : Servlet 3.1, JSR-356 WebSocket , JSR-107 (cache), JSR 330 (Bean Validation 1.1) …

Etat de l'Art : Component Classes

Une classe annotée avec l'annotation @Component (ou l'une de ses dérivées). Juergen fait référence aux stéréotypes UML. Sur une classe, on appose des annotations : @Transactional, @Lazy, @Autowired …
Spring essaie d'être le plus expressif possible. Les annotations ajoute des caractéristiques au code source. Les annotations ont 2 rôles :

1. apportent une fonctionnalité
2. contribuent à la lisibilité du code source.

Etat de l'Art : Configuration Classes

Juergen montre une classe de configuration Spring (@Configuration). Avec l'annotation @Bean, une méthode devient une frabrique de beans Spring.
Associée au component scanning, la configuration en Java permet d'être très flexible. Elle permet notamment de mutualiser de la conf.

Composable Annotations

Juergen montre le code d'une annotation @MyService elle même annotée avec de nombreuses annotations Spring :

- @Service
- @Scope
- @Primary
- @Transactional(rollbackFor=Exception.class)

Cette méta-annotation est bien plus qu'un shortcut.
La difficulté réside dans le choix du nom de l'annotation qui doit permettre de comprendre son objectif.

Composable Annotations with Overridable Attributes

Juergen prend pour exemple l'annotation @MySessionScoped qui fixe la propriété proxyMode() de l'annotation @Scope.

L'annotation @MyTransactional est un second exemple qui redéclare l'attribut readOnly() de l'annotation @Transactional sans valeur par défaut pour forcer l'utilisateur à déclarer explicitement s'il s'agit d'une transaction en lecture seule ou en lecture/écriture.

Generic-bases Injection Matching

Lors de l'injection de dépendances, Spring est désormais capable d'utiliser le type paramétré d'un bean. Plus besoin de nommer les beans et d'utiliser les @Qualifier.
Exemple : @Autowired sur MyRepository<Account>

Recommandation de Juergen : ne pas utiliser de Qualifier si pas besoin. Le code n'en est que plus lisible.

Ordered Collection Injection

Un exemple illustre la possibilité d'ordonner les beans dans une liste à l'aide de l'annotation @Order :

```
@Bean
@Order(2)
public MyRepistory xxx;

@Bean
@Order(1)
public MyRepistory yyy;

@Autowired
List<MyRepository>
```

Lazy Injection Point

L'annotation *@Lazy* permet une initialisation paresseuse d'un bean, lors de sa première utilisation.
Utilisation possible dans le constructeur d'un bean : permet d'obtenir une référence lazy sur un bean.

Component Declarations with JSR-250 et JSR-330

Depuis de nombreuses années, Spring sait interpréter les annotations *@ManagedBean*, *@Bean* et *@PreDestroy*. Désormais, il n'y a plus besoin de configuration particulière pour que Spring les prenne en compte.

Optional Injection Points on Java 8

La classe *Optional* de Java 8 peut être utilisée pour injecter un composant optionnel (équivalent du paramètre required=false de *Autowired*)

Déclarative Formatting with Java 8 Date-Time

Spring supporte les classes *LocalDate* et *LocalDateTime* de java.time. Pour la classe *LocalDate*, Spring sait qu'il s'agit d'une date sans heure. Spring utilise un formateur par défaut. L'utilisation de l'annotation *@DateTimeFormat* n'est nécessaire que si l'on veut un formatage différent de celui proposé par défaut.

Declarative Scheduling (with a java 8 twist)

Mise en œuvre des annotations *@Async* et *@Scheduled*. Le pool de threads à utiliser doit être déclaré. En Java 8, grâce aux annotations répétables, on peut mettre plusieurs *@Scheduled* sur la même méthode : permet d'avoir du code concis.

Java 8 Lambdas with Spring APIs

L'API *JdbcTemplate* profite pleinement des Lambdas, par exemple pour passer les paramètres à un *PreparedStatement* ou mapper un result set dans un bean. Et ceci, Pivotal n'a pas eu besoin de changer la signature des méthodes.

Annotated MVC Controllers

Exemple d'utilisation des annotations @Controller, @RequestMapping, @PathVariable, @Valid.
Avec Java 8, Spring est capable de résoudre le nom des paramètres d'une méthode. Plus besoin de les passer dans @PathVariable. Spring les lit directement dans le byte code.

Annotated JMS Endpoints

Exemple d'utilisations de l'annotations *@JmsListener*.

Messaging Abstraction with JMS, AMQ, STOMP

Le package messaging et la classe *Message* permettent de s'abstraire des technologies de messaging sous-jacentes.

Joergen conclue la présentation par la promotion de Spring Boot et une overview sur ce que va introduire Sping 4.2 (GA in July)

Vient ensuite une démo présentée par Stéphane Nicoll :

Le site web start.spring.io permet de démarrer rapidement un projet.
Dans IntelliJ, un wizard permet de choisir la version de Spring Boot et des technologies à utiliser. Toujours dans IntelliJ, il y'a de la complétion dans le fichier application.properties.

Exemple : @RepositoryEventListener et @HandleAfterCreate pour poster dans RabitMQ un messsage lorsqu'un speaker est créé.