Introduction à Spring Framework v

Présenté par Daba Diawara

Encadré par M. Mohamed Bah

Sommaire

Introduction

Le Spring Framework est un outil très important permettant aux développeurs de simplifier l'écriture et rendre plus légère l'applications J2EE. Spring est reparti en plusieurs Module

Spring Framework fournit un modèle complet de programmation et de configuration pour les applications d'entreprise modernes basées sur Java - sur tout type de plate-forme de déploiement.

Un élément clé de Spring est le support infrastructurel au niveau de l'application : Spring se concentre sur la « plomberie » des applications d'entreprise afin que les équipes puissent se concentrer sur la logique métier au niveau de l'application, sans liens inutiles avec des environnements de déploiement spécifiques.

Gestion des instances de classes (JavaBean et/ou métier).
Programmation orientée Aspect
Modèle MVC et outils pour les application WEB
Outils pour DAO (JDBC)
Outils pour ORM (Hibernate, iBatis)
Outils pour les applications J2EE (JMX, JMA, JCA, EJB)
Conclusion

- Version du Framework Spring

Le Framework Spring a été créé par Rod Johnson et Juergen Holler en 2003.

Spring a connu plusieurs versions :

- Spring 1.0 : mars 2004
- Spring 1.1 : septembre 2004
- Spring 1.2 : mai 2005
- Spring 2.0 : octobre 2006
- Spring 2.5 : novembre 2007
- Spring 3.0 : décembre 2009
- Spring 3.1 : courant 2011

Spring 1.0 implémente les fonctionnalités de base du Framework :

- Le conteneur qui implémente le motif de conception Ioc
- Le développement orienté POJO
- l'AOP par déclaration
- Le support de JDBC, ORM et Framework Web
- la configuration XML basée sur une DTD

Spring 1.2

- Support de JMX
- support JDO 2, Hibernate 3, Top Link
- Support de JCA CCI, JDBC Rowset
- Déclaration des transactions avec @Transactional

Spring 3.1 :

- Support des conversations
- Support des caches
- Ajout de la notion de profile qui permet d'avoir des configurations du context différentes pour chaque environnement
- Ajout de nouvelles annotations pour définir certaines fonctionnalités de namespaces dans la configuration
- Support des servlets 3.0

- Spring 3.2.5 en novembre 2013.
- Spring Framework 4.0 a été publié en décembre 2013.

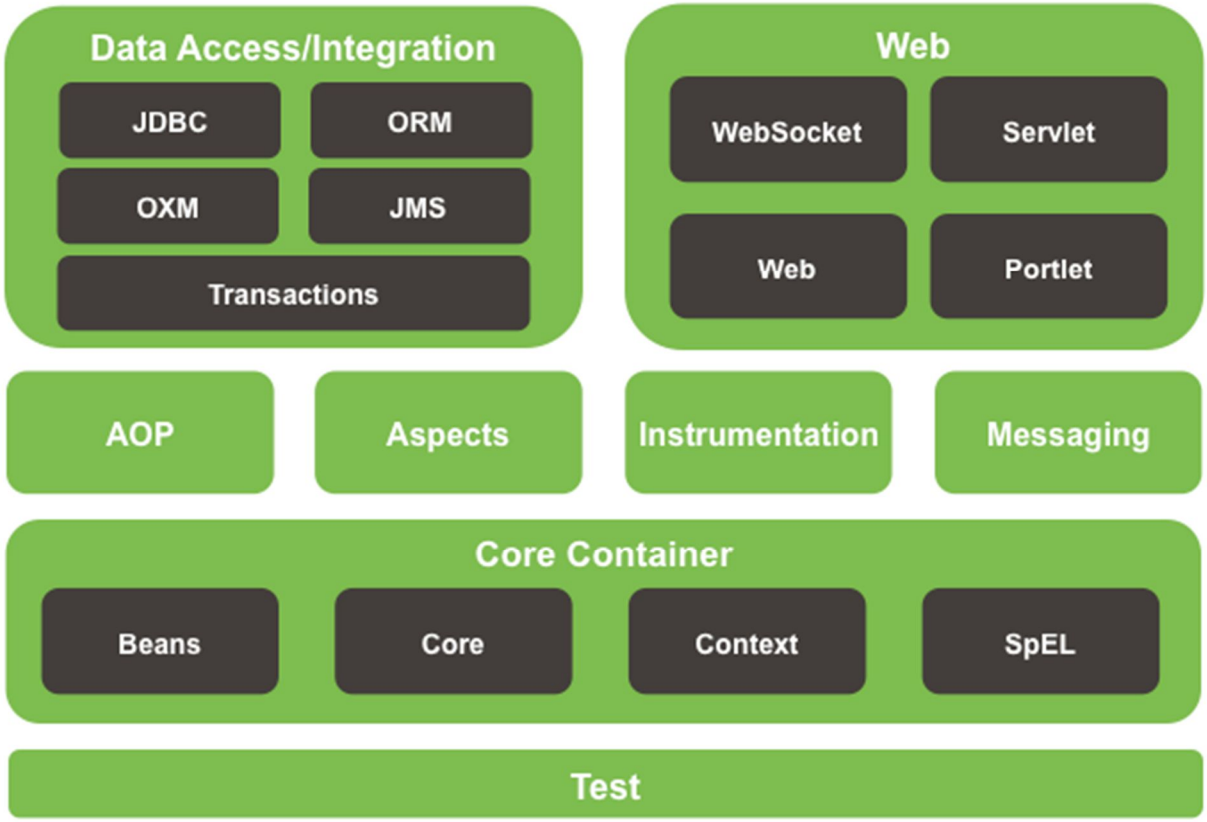
Les améliorations notables de Spring 4.0 comprenaient la prise en charge de Java SE (Standard Edition) 8, Groovy 2, certains aspects de Java EE 7 et Web Socket .

- Spring Framework 4.2.0 a été publié le 31 juillet 2015 et a été immédiatement mis à niveau vers la
- Version 4.2.1, qui a été publiée le 1er septembre 2015. [7] Il est « compatible avec Java 6, 7 et 8, en mettant l'accent sur les raffinements de base. Et des capacités Web modernes ».
- Spring Framework 4.3 a été publié le 10 juin 2016 et sera pris en charge jusqu'en 2020. Il "sera la *dernière génération dans les exigences générales du système Spring 4 (Java 6+, Servlet 2.5+),*
- *Spring 5.0 2017*

  À partir de Spring Framework 5.1, Spring nécessite JDK 8+ (Java SE 8+) et fournit une prise en charge prête à l'emploi pour JDK 11 LTS. La mise à jour 60 de Java SE 8 est suggérée comme version de correctif minimale

- Spring Framework 5.3.x : JDK 8-17 (attendu)
- Spring Framework 5.2.x : JDK 8-15
- Spring Framework 5.1.x : JDK 8-12
- Spring Framework 5.0.x : JDK 8-10
- Spring Framework 4.3.x : JDK 6-8

- Architecture du Spring Framework



Conteneur de base

Le conteneur principal se compose des modules Core, Beans, Context et Expression Language dont les détails sont les suivants :

- Le module **Core** fournit les éléments fondamentaux du Framework, y compris les fonctionnalités d'IoC et d'injection de dépendances.
- Le module **Bean** fournit BeanFactory, qui est une implémentation sophistiquée du modèle d'usine.
- Le module **Context** s'appuie sur la base solide fournie par les modules Core et Beans et c'est un moyen d'accéder à tous les objets définis et configurés. L'interface ApplicationContext est le point central du module Context.
- Le module **SpEL** fournit un langage d'expression puissant pour interroger et manipuler un graphe d'objets au moment de l'exécution.

Accès aux données/Intégration

La couche d'accès/intégration de données se compose des modules JDBC, ORM, OXM, JMS et Transaction dont le détail est le suivant :

- Le module **JDBC** fournit une couche d'abstraction JDBC qui élimine le besoin de codage fastidieux lié à JDBC.
- Le module **ORM** fournit des couches d'intégration pour les API de mappage objet-relationnel populaires, notamment JPA, JDO, Hibernate et iBatis.

- Le module **OXM** fournit une couche d'abstraction qui prend en charge les implémentations de mappage Objet/XML pour JAXB, Castor, XMLBeans, JiBX et XStream.
- Le module **JMS** Java Messaging Service contient des fonctionnalités permettant de produire et de consommer des messages.
- Le module **Transaction** prend en charge la gestion des transactions programmatique et déclarative pour les classes qui implémentent des interfaces spéciales et pour tous vos POJO.

la toile

La couche Web se compose des modules Web, Web-MVC, Web-Socket et Web-Portlet dont les détails sont les suivants :

- Le module **Web** fournit des fonctionnalités d'intégration Web de base telles que la fonctionnalité de téléchargement de fichiers en plusieurs parties et l'initialisation du conteneur IoC à l'aide d'écouteurs de servlet et d'un contexte d'application orienté Web.
- Le module **Web-MVC** contient l'implémentation Model-View-Controller (MVC) de Spring pour les applications Web.
- Le module **Web-Socket prend en** charge la communication bidirectionnelle basée sur WebSocket entre le client et le serveur dans les applications Web.
- Le module **Web-Portlet** fournit l'implémentation MVC à utiliser dans un environnement de portlet et reflète la fonctionnalité du module Web-Servlet.

Divers

Il existe peu d'autres modules importants tels que les modules AOP, Aspects, Instrumentation, Web et Test dont les détails sont les suivants :

- Le module **AOP** fournit une implémentation de programmation orientée aspect vous permettant de définir des intercepteurs de méthode et des points de coupure pour découpler proprement le code qui implémente des fonctionnalités qui doivent être séparées.
- Le module **Aspects** permet l'intégration avec AspectJ, qui est à nouveau un Framework AOP puissant et mature.
- Le module **Instrumentation** fournit une prise en charge de l'instrumentation de classe et des implémentations de chargeur de classe à utiliser dans certains serveurs d'applications.
- Le module de **messagerie prend en** charge STOMP en tant que sous-protocole WebSocket à utiliser dans les applications. Il prend également en charge un modèle de programmation d'annotations pour le routage et le traitement des messages STOMP des clients WebSocket.
- Le module **Test** prend en charge le test des composants Spring avec les frameworks JUnit

- L’inversion de contrôle ou injection de dépendance

L'inversion de contrôle est un principe du génie logiciel qui transfère le contrôle d'objets ou de parties d'un programme à un conteneur ou à un Framework. Nous l'utilisons le plus souvent dans le cadre de la programmation orientée objet.

Contrairement à la programmation traditionnelle, dans laquelle notre code personnalisé appelle une bibliothèque, IoC permet à un Framework de prendre le contrôle du flux d'un programme et d'appeler notre code personnalisé. Pour permettre cela, les frameworks utilisent des abstractions avec un comportement supplémentaire intégré. Si nous voulons ajouter notre propre comportement, nous devons étendre les classes du Framework ou plugin nos propres classes.

Les avantages de cette architecture sont :

Découpler l'exécution d'une tâche de sa mise en œuvre

Facilitant le basculement entre les différentes implémentations

Une plus grande modularité d'un programme

Une plus grande facilité à tester un programme en isolant un composant ou en se moquant de ses dépendances, et en permettant aux composants de communiquer via des contrats

Nous pouvons réaliser l'inversion de contrôle grâce à divers mécanismes tels que : modèle de conception de stratégie, modèle de localisateur de service, modèle d'usine et injection de dépendance (DI).

Nous allons examiner DI ensuite.

Qu'est-ce que l'injection de dépendance ?

L'injection de dépendances est un modèle que nous pouvons utiliser pour implémenter IoC, où le contrôle inversé définit les dépendances d'un objet.

Connecter des objets avec d'autres objets, ou « injecter » des objets dans d'autres objets, est effectué par un assembleur plutôt que par les objets eux-mêmes.

Voici comment créer une dépendance d'objet en programmation traditionnelle :

```
Public class Store {

   Private  Item item ;

Public Store() {

Item = new ItemImpl1();

}

}
```

- La Programmation Orienté Aspects(POA)
  - *La programmation orienté Aspects (POA)*

Simplement mis ensemble, le Framework Spring AOP détourne l'exécution du programme et injecte des fonctionnalités supplémentaires généralement avant, après ou autour de l'exécution de la méthode.

Prenons un exemple de l'aspect journalisation. L'exigence est de :

Consignez un message d'entrée avant d'exécuter le corps de la méthode

Consignez un message de sortie après avoir exécuté le corps du message

Enregistrez le temps nécessaire à la méthode pour terminer l'exécution.

Ici, la journalisation est connue sous le nom d'aspect. L'exécution de la méthode est connue sous le nom de *point de jointure.* Le morceau de code qui enregistre le message d'entrée, le message de sortie et le temps nécessaire pour exécuter la méthode sont appelés les trois *conseils.* La liste des méthodes pour lesquelles ce comportement est requis est connue sous le nom de *coupures de points.* Et enfin, les objets java sur lesquels cet aspect est appliqué sont connus sous le nom de *cibles.*

Il existe un certain nombre de conseils disponibles dans le cadre AOP. Ceux-ci sont :

1. *Avant conseil* - Exécuter avant le point de jointure (exécution de la méthode)
2. *Après avoir renvoyé un avis* - Exécutez si le point de jointure s'exécute normalement.
3. *Après avoir lancé un conseil* - Exécutez si le point de jointure lève une exception.
4. *Autour des conseils* - Exécuter autour du point de jointure (exécution de la méthode)

Ainsi, la journalisation du message d'entrée peut être implémentée par le conseil Avant, la journalisation du message de sortie peut être implémentée par le conseil Après et la journalisation du temps nécessaire à l'exécution de la méthode peut être implémentée à l'aide du conseil Around.

- **BILAN DES SOLUTIONS APPORTEES PAR SPRING**

Le but de Spring est de faciliter et de rendre productif le développement d'applications, particulièrement les applications d'entreprises.

Spring propose de nombreuses fonctionnalités de base pour le développement d'applications :

Un conteneur léger implémentant le design pattern IoC pour la gestion des objets et de leurs dépendances en offrant des fonctionnalités avancées concernant la configuration et l'injection automatique. Un de ses points forts est d'être non intrusif dans le code de l'application tout en permettant l'assemblage d'objets faiblement couplés.

Une gestion des transactions par déclaration offrant une abstraction du gestionnaire de transactions sous-jacent

Faciliter le développement des DAO de la couche de persistance en utilisant JDBC, JPA, JDO ou une solution open source comme Hibernate, iBatis, ... et une hiérarchie d'exceptions

un support pour un usage interne à Spring (notamment dans les transactions) ou personnalisé de l'AOP qui peut être mis en œuvre avec Spring AOP pour les objets gérés par le conteneur et/ou avec AspectJ

Faciliter la testabilité de l'application

Spring favorise l'intégration avec de nombreux autres Framework notamment ceux de type ORM ou web.

Une application typique utilisant Spring est généralement structurée en trois couches :

La couche présentation : interface homme machine

La couche service : interface métier avec mise en œuvre de certaines fonctionnalités (transactions, sécurité, ...)

La couche accès aux données : recherche et persistance des objets du domaine

- Evolution de Spring

- Mise à niveau vers ASM 9.0 et Kotlin 1.4.
- Prise en charge de RxJava 3 in `ReactiveAdapterRegistry`tandis que la prise en charge de RxJava 1.x est obsolète.
- Améliorez la prise en charge native de GraalVM en supprimant les fonctionnalités non prises en charge des images natives.
- Une `spring.spel.ignore`propriété pour supprimer la prise en charge de SpEL pour les applications qui ne l'utilisent pas.

Conteneur de base

- Prise en charge de la liaison pour les classes d'enregistrement Java 14/15 et les constructeurs/accesseurs de style similaire.
- `ObjectProvider.ifAvailable/ifUnique` ignore explicitement les beans des portées actuellement inactives.

- `ApplicationListener.forPayload(Consumer)`méthode pour une `PayloadApplicationEvent`gestion programmatique pratique .
- Prise en charge des extensions Quartz dans `CronExpression`:
    - le champ jour du mois peut être utilisé `L`pour exprimer le dernier jour du mois, `nL`pour exprimer l'avant-dernier jour du mois ou `nW`pour exprimer le jour de la semaine le plus proche du jour du mois n.
    - le champ jour de la semaine peut être utilisé `DDDL`pour exprimer le dernier jour de la semaine DDD du mois, ou `DDD#n`pour exprimer le nième jour de la semaine DDD.

Accès aux données et transactions

- Nouveau `spring-r2dbc`module de support, déplaçant le support R2DBC de base et le réactif `R2dbcTransactionManager`dans le parapluie Spring Framework.
- Nouvelle `JdbcTransactionManager`sous - classe de `DataSourceTransactionManager`, ajoutant la traduction des exceptions d'accès aux données lors de la validation.
- Nouveau `DataClassRowMapper`pour la prise en charge de la liaison basée sur le constructeur, y compris les classes de données Kotlin/Lombok et les classes d'enregistrement Java 14/15.
- Prise en charge de `queryForStream`on `JdbcTemplate`, permettant une itération paresseuse sur un fermable `java.util.stream.Stream`.
- Initialiseurs EntityManager/Session configurables sur `Jpa/HibernateTransactionManager`et `Local(Container)EntityManagerFactoryBean`.
- `HibernateJpaVendorAdapter`expose les conventions Hibernate ORM 5.2+ par défaut (par exemple en `SessionFactory`tant qu'interface fournisseur EMF).
- Les définitions de transaction peuvent déclarer des étiquettes personnalisées maintenant (à utiliser dans les gestionnaires de transactions personnalisées).
- Prise en charge des valeurs de délai d'expiration avec des `${...}`espaces réservés dans les définitions de transaction.
- `TransactionalApplicationListener`interface avec `forPayload`les méthodes d'usine, la prise en charge des rappels et les classes d'adaptateur pour l'enregistrement par programmation (comme alternative aux `@TransactionalEventListener`méthodes annotées).
- Prise en charge de la `@Transactional`suspension de fonctions (Kotlin Coroutines)

Messagerie de printemps

- `RSocketRequester`prise en charge du nouveau `RSocketClient`grâce à quoi un `RSocketRequester`peut être obtenu en tant qu'instance, c'est-à-dire sans `Mono`wrapper ni besoin de se connecter d'abord. Une connexion est obtenue de manière transparente au fur et à mesure des demandes, y compris la prise en charge de la reconnexion.
- `RSocketRequester`soutien au nouveau `LoadbalanceRSocketClient`.
- `RSocketRequester` prise en charge des interactions metadataPush.

- L' `preservePublishOrder`option pour les applications STOMP/WebSocket fonctionne désormais en combinaison avec la taille du tampon d'envoi et les limites de temps.
- Prise en charge de la sérialisation multiplateforme Kotlin (JSON uniquement pour l'instant)

Révision générale du Web

- La configuration CORS expose une nouvelle `allowedOriginPatterns`propriété pour déclarer une plage dynamique de domaines via des modèles de caractères génériques.
- `RequestEntity` prend en charge les modèles d'URI avec des variables.
- `Jackson2ObjectMapperBuilder`expose l' `Consumer<ObjectMapper>`option pour les personnalisations avancées.
- `DataBinder`permet de basculer entre l'accès direct aux propriétés du champ et du bean pendant l'initialisation. Un exemple de scénario est la `@ControllerAdvice`configuration d'un accès direct au champ par défaut globalement avec certains contrôleurs le remplaçant localement, via la `@InitBinder`méthode, pour accéder à la propriété du bean.
- Une `spring.xml.ignore`propriété pour supprimer le support XML pour les applications ne pas l' utiliser, y compris les convertisseurs et les codecs connexes.

Printemps MVC

Correspondance d'URL efficace avec les éléments analysés `PathPattern`dans Spring MVC ; voir "URI Patterns" dans la section "Web Servlet" de la documentation et de l'article de blog "URL Matching with PathPattern in Spring MVC" .

`UrlPathHelper`vérifie le `HttpServletMapping`(Servlet 4.0) pour une détermination plus efficace du chemin de l'application, voir #25100 .

`@ControllerAdvice`peut gérer les exceptions de n'importe quel type de gestionnaire (c'est-à-dire pas seulement `@Controller`mais d'autres comme `HttpRequestHandler`, `HandlerFunction`, etc.) tant qu'il correspond aux mappages de gestionnaire définis sur `ExceptionHandlerExceptionResolver`.

`@ExceptionHandler` peut cibler les causes d'exception à n'importe quel niveau d'imbrication.

`ForwardedHeaderFilter` met à jour l'adresse/le port distant à partir des en-têtes « Transféré pour ».

Ajoutez des beans manquants `WebMvcConfigurationSupport`afin de rendre `DispatcherServlet.properties`(maintenant analysés paresseusement) pas nécessaire pour la plupart des cas d'utilisation.

Prise en charge de la sérialisation multiplateforme Kotlin (JSON uniquement pour l'instant)

1.1.1 WebFlux de printemps

- New `DefaultPartHttpMessageReader`fournit un lecteur de messages entièrement réactif qui convertit un flux tampon en un`Flux<Part>`
- Nouveau `PartHttpMessageWriter`pour écrire le `Flux<Part>`reçu d'un client à un service distant.
- Nouveau `WebClient`connecteur pour Apache Http Components .
- `WebClient`et `ClientRequest`donner accès au `ClientHttpRequest`et à la demande native. Ceci est utile pour personnaliser les options par requête spécifiques à la bibliothèque HTTP.
- `Encoder`et `Decoder`implémentations pour Netty `ByteBuf`.
- `ForwardedHeaderTransformer` met à jour l'adresse/le port distant à partir des en-têtes « Transféré pour ».
- `@EnableWebFlux`permet la prise en charge des gestionnaires de type `WebSocketHandler`.
- `WebSocketSession`donne accès au `CloseStatus`.
- `WebHttpHandlerBuilder`possibilité de décorer toute la `WebFilter`chaîne au niveau du `HttpHandler`.
- Recherches de chemin direct plus efficaces pour les `@RequestMapping`méthodes qui n'ont pas de modèles ou de variables URI.
- `ClientResponse`optimisations de performances et `mutate()`méthode pour des changements efficaces via un filtre client ou un `onStatus`gestionnaire, voir #24680 .
- Prise en charge de la sérialisation multiplateforme Kotlin (JSON uniquement pour l'instant)

1.1.2 Essai

- Le *framework Spring TestContext* est maintenant construit et testé avec JUnit Jupiter 5.7, JUnit 4.13.1 et TestNG 7.3.0.
- Les annotations liées aux tests sur les classes englobantes sont désormais *héritées* par défaut pour `@Nested`les classes de test JUnit Jupiter .
    - Il s'agit d'un changement potentiellement décisif, mais le comportement peut être inversé pour *remplacer la* configuration des classes englobantes via l' `@NestedTestConfiguration`annotation, une propriété système JVM ou une entrée dans un `spring.properties`fichier à la racine du chemin de classe.
    - Consultez la Javadoc`@NestedTestConfiguration` et le manuel de référence pour plus de détails.
- La `spring.test.constructor.autowire.mode`propriété peut désormais être définie via un paramètre de configuration JUnit Platform pour modifier le `@TestConstructor`mode de câblage automatique par défaut , par exemple via le `junit-platform.properties`fichier.
- A `PlatformTransactionManager`configuré via l' `TransactionManagementConfigurer`API a désormais la priorité sur tout gestionnaire de transactions configuré en tant que bean dans le à `ApplicationContext`moins qu'il ne `@Transactional`soit configuré avec un qualificatif pour le gestionnaire de transactions explicite à utiliser dans les tests.

- Les transactions gérées par les tests peuvent désormais être désactivées via `@Transactional(propagation = NEVER)`en plus de la prise en charge existante pour `propagation = NOT_SUPPORTED`- par exemple, pour remplacer une `@Transactional`déclaration à partir d'une annotation composée, sur une superclasse, etc.
- `WebTestClient`prise en charge de l'exécution des requêtes contre `MockMvc`. Cela permet la possibilité d'utiliser la même API pour les `MockMvc`tests et pour les tests HTTP complets. Voir la section mise à jour sur les tests dans la documentation de référence.
- `WebTestClient` a amélioré la prise en charge de l'affirmation de toutes les valeurs d'un en-tête.
- Matchers de données en plusieurs parties dans la prise en charge du test REST côté client pour le `RestTemplate`.
- L'intégration HtmlUnit pour Spring MVC Test prend en charge les paramètres de téléchargement de fichiers.
- Améliorations mineures `MockHttpServletResponse`concernant l'encodage des caractères et les `Content-Language`valeurs d'en-tête multiples .
- Révision majeure de MockMVC Kotlin DSL pour prendre en charge plusieurs matchers