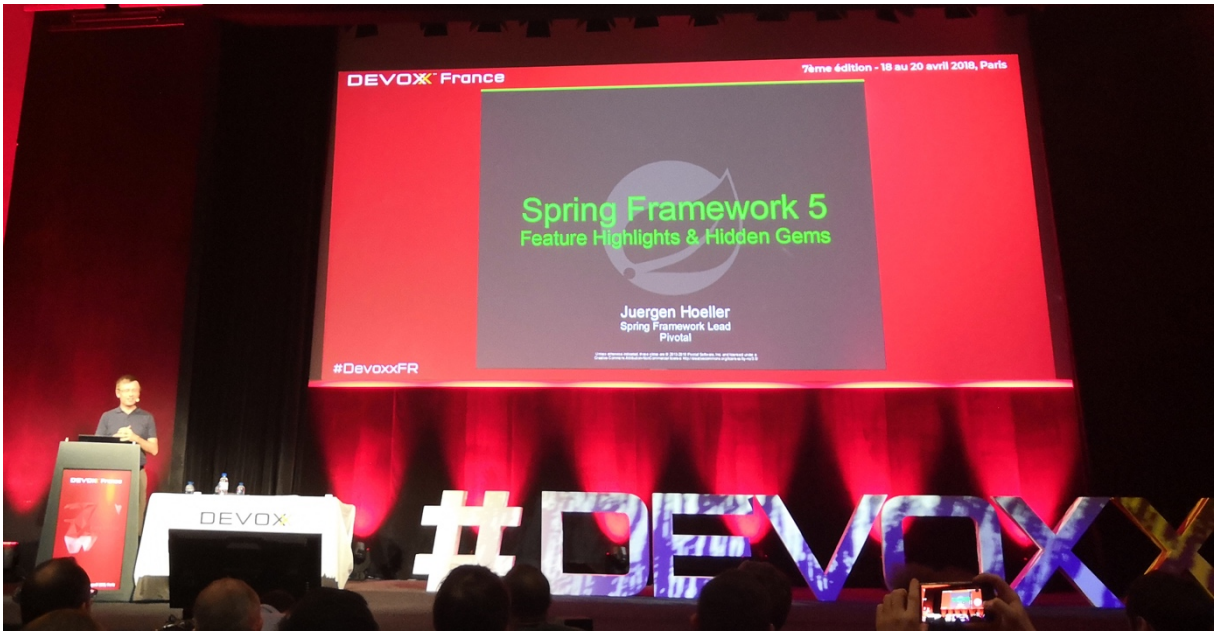
Spring Framework 5 : Feature Highlights & Hidden Gems

Speakers : Juergen Hoeller (Pivotal)
Format : Conférence
Date : 19 avril 2018
Support : spring5-highlights-gems.pdf

Juergen est le lead developpeur de Spring Framework et en est le co-fondateur.



Spring Framework 5.0

- La release GA est sortie en septembre 2017, une semaine après Java 9
- Supporte les JDK 9 et 10, le langage Kotlin et le projet Reactor
    - Java 9 : n'utilise que des API non dépréciés en Java 9, supporte les Java Modules
- Driven by functional API design and reactives architecture
- Prérequis de Spring 5 : Java SE 8 et +, Java EE 7 et +
    - Servlet 3.1, JDK 8, Bean Validation 1.1, JPA 2.1, JMS 2.0
- Support de JUnit 5
- Intégration complète avec l'API Java EE 8. Spring s'adapte pour les exploiter.
    - Servlet 4.0, Bean Validation 2.0, JPA 2.2, JSON Binding API 1.0
    - Tomcat 9.0, Hibernate Validator 6.0, Apache Johnzon 1.1

Component Style

Etat de l'art d'une application Spring

- Component Classes : @Service, @Lazy, @Autowired, @Transactional
- Composable annotation : création d'annotation @MyService annotée avec @Service, @Scope(« session »), @Primary, @Transactional(rollbackFor=Exception.class)

- Configuration Classes : @Configuration, @Profile, @Bean, @EnableTransactionManagement
- Annotated MVC Controllers : @RestController, @CrossOrigin, @GetMapping, @PostMapping, @ParhVariable, @Valid

Functional Style vs Annotation Style

Spring 5 apporte un style de programmation fonctionnelle alternative à l'usage des annotations.
Avantages de l'approche par annotation : composant faiblement couplé, auto-documentés
Avantages de l'approche fonctionnelle :

- Enregistrement des beans par programmation
- Plus besoin d'annotations ou de scanning, pas de réflexion
- Non-null API design avec l'annotations @Nullable

Spring 5 supporte Kotlin out of the box : les Spring 5's Kotlin Extensions font gagner en concision du code

Enregistrement de beans par programmation en Java 8

```
// 1ière version
GenericApplicationContext ctx = new GenericApplicationContext();
ctx.registerBean(Foo.class);
ctx.registerBean(Bar.class, () -> new Bar(ctx.getBean(Foo.class)));

// 2nde version alternative
// Possibilité de récupérer le BeanDefinition pour setter par exemple la propriété lazyInit.
// Possibilité de passer un new Bar() => n'utilise plus la réflexion.
GenericApplicationContext ctx = new GenericApplicationContext();
ctx.registerBean(Foo.class, Foo::new);
ctx.registerBean(Bar.class, () -> new Bar(ctx.getBean(Foo.class)), bd -> bd.setLazyInit(true));
```

Enregistrement de beans par programmation en Kotlin

Exemple en Kotlin plus concis qu'en Java avec 2 styles possibles

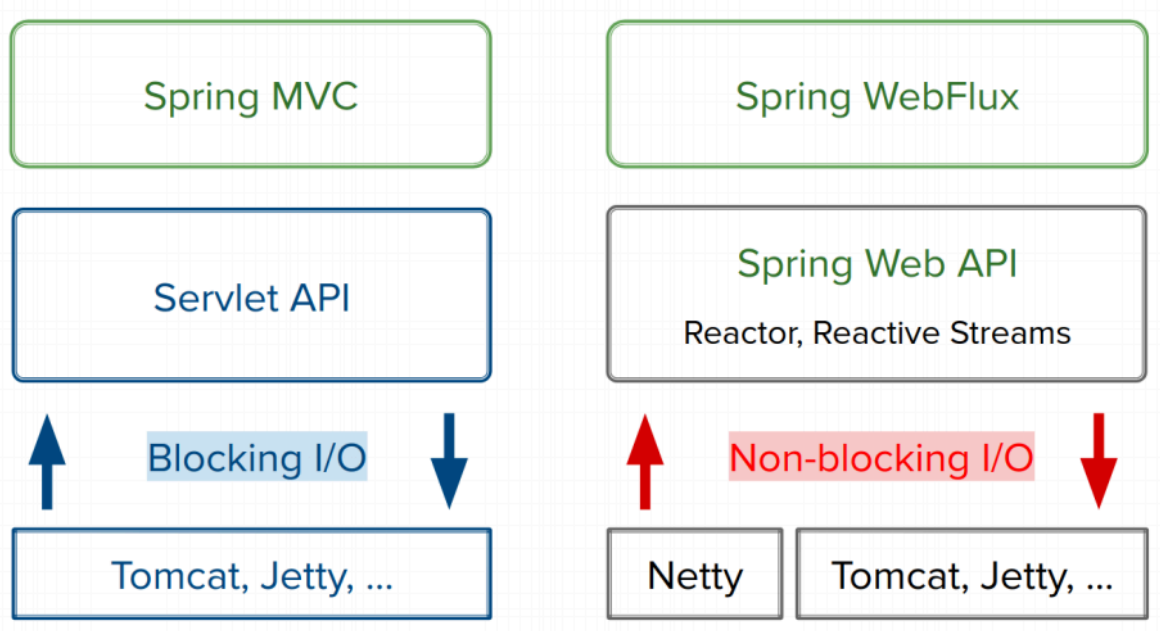
```
// Java-style
val ctx = GenericApplicationContext()
ctx.registerBean(Foo::class)
ctx.registerBean { Bar(it.getBean(Foo::class)) }

// Gradle-style avec Spring's Kotlin extensions
val ctx = GenericApplicationContext { registerBean() registerBean { Bar(it.getBean()) } }
```

Reactive Architectures

Reactor 3 : Reactive Streams with backpressure

- Streaming non bloquant. Pas de threads bloqués.
- Le Publisher n'envoie des données que lorsque le Subscriber est disponible pour les traiter



Spring 5 propose 2 choix d'architecture :

1. Blocking IO : Spring MVC + Servlet API + Tomcat/Jetty
2. Non-blocking IO : Spring WebFlux + Spring Web API (Reactor, Reactive Streams) + Netty (ou Tomcat/Jetty utilisé dans un mode particulier, sans Servlet)

Les Non-blocking IO sont à privilégier sur des applications avec beaucoup de charge.
Une event loop est initialisée au démarrage.

Spring MVC et Spring WebFlux partage un modèle commun : @Controller, Reactive Clients, Tomcat/Jetty/Undertow.
A partir du moment où l'une des API utilisée par l'application est bloquante (ex : JDBC ou JPA), il est préférable de rester sur la stack IO Bloquante. Avantage : elle est plus simple à debugger et à utiliser.

Avec l'approche réactive, la signature des méthodes des contrôleurs change : elles retournent un Mono ou un Flux. Ces 2 classes implémentent l'API Reactive Streams.
Exemple :

```
@GetMapping("/users/{id}")
public Mono<User> getUser(@PathVariable Long id) { return this.repository.findById(id); }
```

Hidden Gems

Nullability

- Spring 5 introduit l'annotation @Nullable permettant d'indiquer qu'un paramètre, un champ ou une valeur de retour peut-être null. Par défaut, ces champs ne peuvent être null (comme en Kotlin).
- Support dans IntelliJ
- Permet de s'abstraire de vérifications qui ne servent à rien car la valeur ne peut jamais être null

Data Class Binding

Spring 5 sait désormais gérer des classes immuables. Il sait désormais passer par le constructeur pour initialiser les instances.
Cela va lui permettre de travailler avec des data classes de Kotlin et les @Data de Lombok.

Common Logging Bridge

Spring 5 vient avec le module spring-jcl qui n'est autre qu'un bridge pour Jakarta Commons Logging.