

24 octobre 2024

JAVA Spring

o Java 17

Maven:

- gestion de dépendances → téléchargement pour moi sans le faire manuellement
- Tomcat server "on top" de mon Spring Project
- Fichier pour magasiner pom.xml.

o GAV:

↳ Group → convention inverse com. du code
ArtifactId
Version → optionnel

o Section <dependencies>, la liste des dépendances

Hilary

First App en Spring

- Set "spring-initializer" et le zipper
- Générer package et ajouter une classe avec en-tête
 @RestController
- Écrire une nouvelle fonction avec en-tête @GetMapping("/")
- Quand annotation, automatique découverte avec "component scanning"
- "endpoint" @GetMapping("/")

Spring Projects

- Spring Cloud, Spring Web App.
- Voir en ligne

Maven File Structure

- Maven aide dans le build
- ↳ Java: Java Code
- ↳ resources: properties/config files used in app
- ↳ webapp: JSP file and other web assets
- ↳ test: unit testing
- ↳ target: destination directory for compiled code. Auto created by Maven.

Maven Key Concepts

- Pom:
- Metadata: name, version, output file type
 - Dependencies: spring, hibernate
 - Plugins: additional custom tasks to run like generate JUnit test reports.

Maven Wrapper file

- maven → pour Linux ou Mac
- maven.cmd → pour Windows
- Télécharge et corse la version Maven pour toi.

POM file :

- spring-boot-starter-web
- Maven plugin : package executable jar
Pour rouler l'app
ou tu peux rouler en : /maven package
./mvnw spring-boot:run

Application Properties :

- créée par Spring Initializr
- tu peux ajouter : server.port = 8885
- custom properties
- ressources / application.properties

Tu peut lire les valeurs dans ton code
@Value ("\${coach.name}")
private String coachName;

Static Content

• HTML, CSS, js, images, etc.

• Ne pas utiliser "/webapp" si app est jar

Templates

auto config of template engines

Unit test : Spring Boot unit test class

Spring Boot Starters

- Simplify setup of an app
- Curated list of Maven dependencies
 - plus facile commencer avec Spring

Spring MVC starters * dependances → difficile

solution: un bundle avec tout ce que tu as besoin

Aller sur Spring Initializr
Choisir et générer

Spring Boot Starter Parent

- provide "Maven defaults"
- Override default property, tu peux changer
- Avantage:
 - default configuration: Java version, UTF-encoding
 - assure consistent versions of dependencies across Spring Boot projects

Spring Boot Dev Tools

- Auto-restart your app quand code est updated
- Ajouter dans pom.xml

Note IntelliJ:

1. Preferences > Build, Execution, Deployment > Compiler
2. Click Box: Build automatically

3. "Allow auto-make to...", dans Advanced Settings
4. Edit pom file pour pointer

Spring Boot Actuator

- Monitor, status, metrics ?

- Expose endpoints to ↑

- Ajouter dans pom.xml

- utiliser /actuator prefix ⇒ /actuator/health

/health : check state of app → exposed by default

/info → . begin modif application.properties
• give info about app ↑
• par défaut empty

info.app.name = My Super Cool name

info.app.description = A crazy and fun app, yeahoo!

plusieurs autres endpoints..

How?

1. Edit "pom" avec starter-actuator

2. View actuator endpoints for: /health

3. Edit "application.properties" to customize "/info"

Securing Spring Boot Actuator

- Tu veux pas exposer tous les endpoints

- Default user → user

- password → sera dans la console

- Credential custom

- spring.security.user =

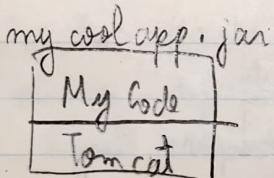
- spring.security.password =

Aussi possible d'exclure des "endpoints"
⇒ modifier le application.properties

Exercice : ajouter dans pom.xml security
→ permet de sécuriser l'actuator
en demandant des logins

Run Spring Boot Apps Using Command Line

- o Pas besoin d'un IDE
- o Spring Boot already include the server



Using a command line :

1. menu package ⇒ Va créer un snapshot
2. java -jar target\le-snapshot.jar

Run using plugin

mvn spring-boot:run

Injicing Custom Application Properties

- o Pa hand-code

src/main/resources/application.properties

coach.name = Kofi Abayei

- o Utilise l'annotation @Value

@Value("\${coach.name}")

private String coachName;

Spring Boot Properties

application.properties file

Il y en a beaucoup

Core: log levels, logfile

Web: server.port

Activator: endpoints

Security: spring.security.user = coconut

Il y en a d'autre

Inversion of Control

- o Object Factory: the approach of outsourcing the construction and management of objects
- o Spring Container → Object factory

XML Config - deserializer

Java Annotations

Java Source Code

Dependency Injection: the client delegates to another object the responsibility of providing its dependencies.

Spring Container: create + manage + inject object dependencies

2 Types Injections:

◦ Constructor Injection:

- Have required dependencies

◦ Setter Injection:

- Optional dependencies

Autorouting: - Spring will look for a class that matches

↳ matches by type : class or interface

- Spring will inject it automatically

- "Is there anyone who implements the Coach interface?" CricketCoach

"Assemble tout pour moi si l'objet a de dépendance ou
helper et Donne-moi le produit" - Dependency Injection

◦ Component: pour marquer

◦ A Spring Bean is a Java Class managed by Spring

◦ L'annotation `@Component` ⇒ disponible pour dependency injection

◦ Autowired: tells Spring to Inject the dependency

Behind the scenes - Constructor Injection

Component Scanning

◦ Spring auto scan for beans like `@Component`

◦ Behind the scenes, create application context and registers all beans

◦ Starts the embedded server Tomcat etc

Start: some package of main Spring Boot Application
sub-package

- Spring only scan packages of main Spring Boot application class and sub-packages.

Solution: "scanBase Packages" pour ajouter d'autres packages @SpringBootApplication

scanBase Packages = { "le Nom des Packages à Scanner",
"com.lieu2code.util" }
? lister les packages

Setter Injection

- Inject dependencies by calling setter method(s) on your class
- Can use any method name. (Et grâce à @Autowired)

Field Injection:

- Pas recommandé par Spring → difficile à Unit test
- "Java reflection"

Qualifiers:

- * Coach implementation ? Quoi faire bean ID of class, first character is lower case

Solution 1 : Utiliser @Qualifier ("enrichCoach")

↳ Dans constructeur ou setter

@Primary Annotation:

- I don't care which coach.
- Ajoute @Primary en dessous de @Component
- Seulement 1 @Primary. Si plusieurs, problème

On ne peut pas mixer @Primary et @Qualifier

En général, utiliser @Qualifier

Lazy initialization

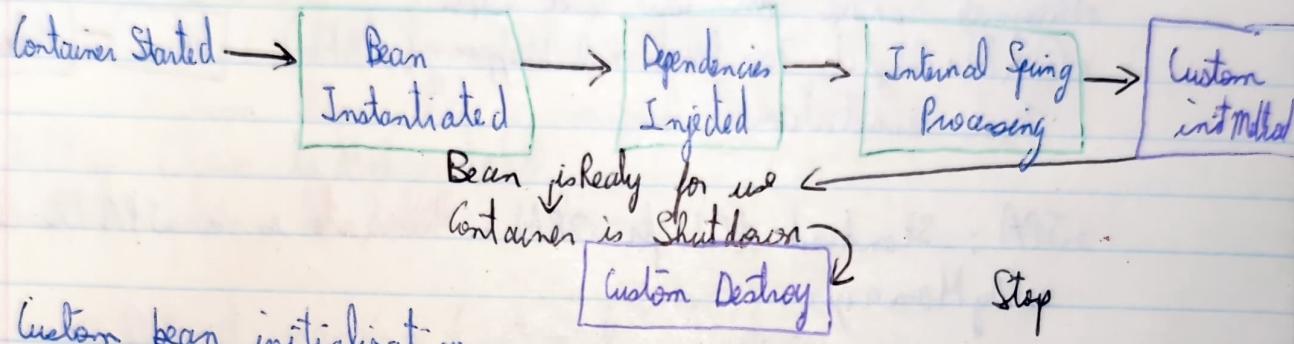
- By default * beans initialized
- Bean initialized when:
 - needed for dependency injection
 - explicitly requested
- Instead use `@Lazy` when don't want default
- Global conf `Global Lazy Config application.properties`
`spring.main.lazy-initialization = true`

Bean scopes

- Lifecycle of a Bean
- Default scope is singleton : création 1 par défaut
 @ Component
- Scope (Configurable BeanFactory, SCOPE - SINGLETON)
Autre choix :
 - Singleton
 - Prototype : new object for each injection
 - Request : scoped to HTTP web request (web app only)
 - Session : ' ' ' session (')
 - Application : ' ' ' a web app ServletContext (')
 - Websocket : ' ' ' a web socket (')

L'exemple comparé si il classe ou non. Singleton vs Prototype

Bean Lifecycle Methods



Custom bean initialization

- ② PostConstruct → initializes db, sockets, file, etc.

Custom bean destroy

- ② PreDestroy → bean cleanup db, socket, file, etc.
 - ↳ c'est exécuté quand tu arrêtes ton programme (dans l'exemple)

Java Config Bean

Dev process

1. Create a Configuration class
2. Define a Bean method to configure the bean dans la classe avec @Bean
3. Inject the bean in our controller

Manually construct Bean

② Bean

```
public Search swimSearch()  
{  
    return new SwimSearch();  
}
```

Use case for @Bean

- o third party class and make it available as a Bean !
- o Example : utiliser AWS

Note sur l'exemple :

pas besoin de Component en utilisant @Bean, SwimSearch() n'a pas de

Custom Bean id @Bean("aquarelle") pas de Component.

↳ o que tu utilises des @Qualifiers