Les annotations Spring les plus utilisées

Philémon Globléhi
Développeur Back-end Java

Scan des composants & Injection de dépendances

- @Component : Marque une classe comme un composant géré par Spring. Utilisée pour les composants génériques qui ne rentrent pas dans les catégories @Service, @Repository ou @Controller.

- @Controller : Version spécialisée de @Component, utilisée dans Spring MVC pour définir un contrôleur web.

- @RestController : Combine @Controller et @ResponseBody pour simplifier le développement des APIs REST.

- @Service : Version spécialisée de @Component, utilisée pour la logique métier et les composants de la couche service.

- @Repository : Version spécialisée de @Component, utilisée pour les objets d'accès aux données (DAO) afin d'indiquer les opérations liées au stockage.

- @Qualifier("nomDuBean") : Spécifie quel bean injecter lorsqu'il y a plusieurs choix possibles.

- @Primary : Marque un bean comme le choix par défaut lorsqu'il existe plusieurs options possibles.

- @Value("${cle.propriete}") : Injecte des valeurs provenant de fichiers de configuration dans les beans Spring.

Configuration et la gestion des beans

- @Configuration : Marque une classe comme une source de définitions de beans pour le conteneur Spring.
- @Bean : Déclare une méthode qui produit un bean géré par Spring.
- @Import({ConfigClass.class}) : Importe une autre classe de configuration.
- @ImportResource("classpath:config.xml") : Charge un fichier de configuration XML.
- @DependsOn("nomDuBean") : Assure qu'un bean est initialisé seulement après que des dépendances spécifiques soient disponibles.
- @Lazy : Retarde l'initialisation du bean jusqu'à ce qu'il soit d'abord utilisé.
- @Scope("prototype") : Définit le scope (portée) du bean (singleton, prototype, etc.).

- @PropertySource("classpath:app.properties") : Charge des fichiers de propriétés externes.

- @EnableAspectJAutoProxy : Active la prise en charge de la programmation orientée aspect (AOP) dans Spring.

Planification et exécution asynchrone

- @Scheduled(cron="0 0 * * * ?") : Planifie une méthode pour qu’elle s’exécute à un intervalle fixe (par exemple, toutes les heures).

- @Async : Marque une méthode pour une exécution asynchrone, ce qui signifie qu'elle s'exécutera de manière parallèle sans bloquer l'exécution du reste du programme.

Programmation orientée aspect (AOP)

- @Aspect : Définit un aspect, c'est-à-dire une modularisation des préoccupations transversales (comme le logging).

- @Before("execution(package.Class.method(..))")* : Exécute une logique avant l'exécution de la méthode correspondante.

- @After("execution(package.Class.method(..))")* : Exécute une logique après l'exécution de la méthode (quel que soit le résultat).

- @AfterReturning("execution(package.Class.method(..))")* : S'exécute uniquement lorsque la méthode retourne un résultat avec succès.

- @AfterThrowing("execution(package.Class.method(..))")* : S'exécute si la méthode lance une exception.

- @Around("execution(package.Class.method(..))")* : Enveloppe l'exécution de la méthode pour effectuer des traitements avant et après l'exécution.

- @Pointcut("execution(package..(..))")** : Définit des expressions AOP réutilisables.

Annotations Spring Boot

- @SpringBootApplication : Combine @Configuration, @EnableAutoConfiguration et @ComponentScan. Sert de point d’entrée principal d’une application Spring Boot.

- @EnableAutoConfiguration : Active la configuration automatique de Spring Boot en fonction des dépendances présentes.

- @ComponentScan("com.example") : Scanne les packages spécifiés pour détecter automatiquement les composants Spring (@Component, @Service, etc.).

Spring MVC & REST

Gestion des requêtes

- @RequestMapping("/endpoint") : Associe des requêtes HTTP à des méthodes de traitement.
- @GetMapping("/endpoint") : Gère les requêtes HTTP GET.
- @PostMapping("/endpoint") : Gère les requêtes HTTP POST.
- @PutMapping("/endpoint") : Gère les requêtes HTTP PUT.
- @DeleteMapping("/endpoint") : Gère les requêtes HTTP DELETE.
- @PatchMapping("/endpoint") : Gère les requêtes HTTP PATCH.
- @RequestParam("param") : Lie les paramètres de requête (query) à des variables.

- @PathVariable("id") : Extrait des valeurs depuis l'URL (ex: /users/{id}).
- @RequestBody : Convertit le corps de la requête en objet Java.
- @ResponseBody : Convertit les objets Java en réponse JSON/XML.
- @ModelAttribute("model") : Lie les données de la requête à un modèle (ex: formulaire).
- @SessionAttributes("user") : Stocke des attributs dans la session HTTP.
- @CrossOrigin("*") : Active la prise en charge de CORS (Cross-Origin Resource Sharing) pour un contrôleur.

Gestion des exceptions

- @ExceptionHandler(Exception.class) : Intercepte les exceptions au niveau du contrôleur.

- @ResponseStatus(HttpStatus.NOT_FOUND) : Définit le code de statut HTTP à retourner en cas d’exception (ex : 404).

- @ControllerAdvice : Définit une gestion globale des exceptions pour plusieurs contrôleurs.

Spring Cloud & Microservices

- @EnableEurekaClient : Active le client Eureka pour l’enregistrement auprès du serveur Eureka.
- @FeignClient(name="user-service") : Déclare un client Feign pour appeler un service distant (ex : user-service).
- @LoadBalanced : Active le load balancing côté client (grâce à Ribbon).
- @CircuitBreaker : Implémente une logique de circuit breaker (protection contre les pannes de services).
- @RateLimiter : Applique une limitation de débit (pour éviter la surcharge d’un service).
- @EnableConfigServer : Marque une application comme serveur de configuration Spring Cloud.

- @EnableDiscoveryClient : Active la découverte de services (avec Consul, Eureka ou Zookeeper).
- @EnableConsulServer : Marque une application comme un serveur Consul.
- @EnableZuulProxy : Marque une application comme un proxy Zuul (passerelle API Gateway).

Spring Messaging & WebSockets

- @EnableWebSocket : Active la prise en charge des WebSockets dans l’application.

- @MessageMapping("/chat") : Lie les messages WebSocket à des méthodes de traitement (comme @RequestMapping mais pour WebSocket).

- @SendTo("/topic/updates") : Envoie la réponse à une destination spécifique après le traitement d’un message WebSocket.

Spring Kafka

- @EnableKafka : Active la prise en charge de Kafka dans l’application Spring.

- @KafkaListener(topics="myTopic") : Indique qu’une méthode écoute un topic Kafka (ex : myTopic).

Spring Caching

- @EnableCaching : Active la gestion du cache dans l'application Spring.
- @Cacheable("cacheName") : Met en cache le résultat d'une méthode (si le cache existe déjà, il est renvoyé sans exécuter la méthode).
- @CachePut("cacheName") : Met à jour le cache avec des données fraîches, même si une valeur existe déjà.
- @CacheEvict("cacheName") : Supprime une ou plusieurs entrées du cache.
- @CacheConfig(cacheNames={"defaultCache"}) : Définit la configuration de cache par défaut au niveau de la classe.

Spring Security

- @EnableWebSecurity : Active la configuration de Spring Security.
- @Secured("ROLE_ADMIN") : Protège l'accès à une méthode selon le rôle (ex : uniquement les admins).
- @RolesAllowed({"ROLE_USER", "ROLE_ADMIN"}) : Définit les rôles autorisés à exécuter une méthode.
- @AuthenticationPrincipal : Injecte l'utilisateur actuellement authentifié.
- @PreFilter : Filtre les paramètres d'une méthode avant son exécution.
- @PostFilter : Filtre le résultat d'une méthode après son exécution.

- @PreAuthorize("hasRole('ROLE_USER')") : Vérifie l’autorisation avant l’exécution d’une méthode, en fonction du rôle.

- @PostAuthorize("returnObject.owner == authentication.name") : Vérifie l’autorisation après exécution, en se basant sur le résultat retourné.

Spring Data MongoDB

- @Document : Indique qu'une classe représente un document MongoDB (équivalent d'une table en base relationnelle).

- @Field : Spécifie le nom du champ dans le document MongoDB (si différent du nom de l'attribut Java).

- @Id : Marque un champ comme étant la clé primaire du document.

- @Indexed : Crée un index sur un champ MongoDB pour améliorer les performances des requêtes.

Spring Data Cassandra

- @Table : Indique que la classe correspond à une table Cassandra (comme @Entity en JPA).
- @PrimaryKeyColumn : Marque un champ comme faisant partie de la clé primaire.
- @Column : Spécifie qu'un champ doit être lié à une colonne Cassandra.
- @PartitionKey : Définit un champ comme clé de partition, utilisée pour répartir les données dans Cassandra.

Tests avec Spring

- @SpringBootTest : Charge tout le contexte de l'application Spring pour des tests complets.
- @WebMvcTest : Permet de tester uniquement les contrôleurs Spring MVC, sans charger le reste de l'application.
- @MockBean : Crée des faux beans (mocks) dans les tests, pour simuler le comportement de certaines dépendances.
- @DataJpaTest : Permet de tester les composants JPA (comme les repositories) dans un environnement isolé.
- @TestConfiguration : Définit des configurations spécifiques aux tests.

- @BeforeEach et @AfterEach : Exécutent du code avant ou après chaque test (par exemple, initialisation ou nettoyage).

- @Test : Marque une méthode comme étant un test (annotation JUnit).

- @Rollback : Indique si la transaction doit être annulée après un test (utile pour garder la base propre).

Spring Data JPA & Transactions

Mapping des entités

- @Entity : Déclare une entité JPA (table représentée par une classe Java).
- @Table(name="users") : Spécifie le nom de la table dans la base de données.
- @Id : Marque un champ comme étant la clé primaire.
- @GeneratedValue(strategy=GenerationType.AUTO) : Génère automatiquement les valeurs de clé primaire.
- @Column(name="username") : Lie un champ de la classe à une colonne de la base.
- @OneToOne, @OneToMany, @ManyToOne, @ManyToMany : Définit les relations entre entités.

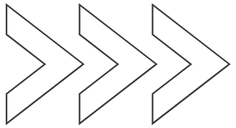
- @Embeddable : Marque une classe comme pouvant être intégrée dans une autre entité.
- @Embedded : Permet d’intégrer une classe marquée @Embeddable dans une entité.
- @Enumerated(EnumType.STRING) : Enregistre un enum sous forme de texte dans la base.
- @Fetch : Spécifie comment charger les relations (chargement "eager" ou "lazy").
- @MappedBy("user") : Définit le côté inverse d’une relation, pour éviter les doublons.

Repository & Transactions

- @Transactional : Gère les transactions au niveau d’une méthode ou d’une classe.
- @Query("SELECT u FROM User u WHERE u.email = ?1") : Permet d’écrire des requêtes JPA personnalisées.
- @Modifying : Indique qu'une requête est de type UPDATE ou DELETE.
- @EnableJpaRepositories : Active les repositories Spring Data JPA.
- @Param : Lie les paramètres de méthode aux paramètres de la requête dans @Query.

Spring Validation

- @NotBlank : Vérifie qu’une chaîne de caractères n’est ni vide ni nulle (espaces exclus).
- @NotEmpty : Vérifie qu’une collection, map ou tableau n’est pas vide.
- @Min(valeur) : Vérifie qu’un champ numérique est supérieur ou égal à une certaine valeur.
- @Max(valeur) : Vérifie qu’un champ numérique est inférieur ou égal à une certaine valeur.
- @Email : Vérifie qu’une chaîne est une adresse e-mail valide.
- @Pattern(regexp) : Vérifie qu’un champ respecte une expression régulière.
- @Size(min, max) : Vérifie que la taille d’un champ est comprise dans une plage donnée.



Merci

Philémon Globléhi, Back-end Developer & DevOps Lover
Linkedin: https://www.linkedin.com/in/philemon-globlehi/