@RestController = @Controller + @ResponseBody

@PostMapping = @RequestMapping + method

JDBC

1. SỬ dụng SQL thuần

JPA được tiến hóa lên từ JDBC

entity mapping table

@Entity

@Table

@Column

spring data jpa không phụ thuộc vào loại database(mysql, sql server)

entity: java class

HQL: hibernate query language

SQL native

JDBC -> JPA -> Spring data jpa

Spring data jpa được phát triển dựa trên JPA, từ spring 4.0 trở lên

@Id = primary key + not null

@GeneratedValue(strategy = GenerationType.IDENTITY) = auto incresement

@MappedSuperclass apply field của cha cho con, lớp cha phải abstract

mỗi loại database khác nhau sẽ có spring.jpa.properties.hibernate.dialect(application.properties) khác nhau

spring.jpa.hibernate.ddl-auto(application.properties) : tạo table. None không làm gì cả(tạo đầy đủ rồi, không thao tác và database nữa)

validate, update cập nhật lại column trong table, creatable : tạo ra table mới, create-drop: xóa hết table và tạo lại table mới

muốn sử dụng jpa bắt bược phải có java class entity vì nó làm việt với java class entity

@ManyToOne

@JoinColumn(name = "category\_id"): name : tên cột lưu vào data

@OneToMany(mappedBy = "category"): tên thuộc tính lớp kia

swagger spring boot : mô tả API

Spring đặt tên cho container là ApplicationContext

và đặt tên cho các dependency là Bean

@Component là một Annotation (chú thích) đánh dấu trên các Class để giúp Spring biết nó là một Bean.

Tôi đánh dấu thuộc tính Outfit của Girl bởi Annotation @Autowired. Điều này nói với Spring Boot hãy tự inject (tiêm) một instance của Outfit vào thuộc tính này khi khởi tạo Girl.

@Autowired đánh dấu cho Spring biết rằng sẽ tự động inject bean tương ứng vào vị trí được đánh dấu.

Tất cả những Bean được quản lý trong ApplicationContext đều chỉ được tạo ra một lần duy nhất và khi có Class yêu cầu @Autowired thì nó sẽ lấy đối tượng có sẵn trong ApplicationContext để inject vào.

@Scope("prototype") mỗi lần sử dụng sẽ là 1 instance mới. Đánh dấu trên @Component

Sau khi tìm thấy một class đánh dấu @Component. thì quá trình inject Bean xảy ra theo cách như sau:

Nếu Class không có hàm Constructor hay Setter. Thì sẽ sử dụng Java Reflection để đưa đối tượng vào thuộc tính có đánh dấu @Autowired.

Nếu có hàm Constructor thì sẽ inject Bean vào bởi tham số của hàm

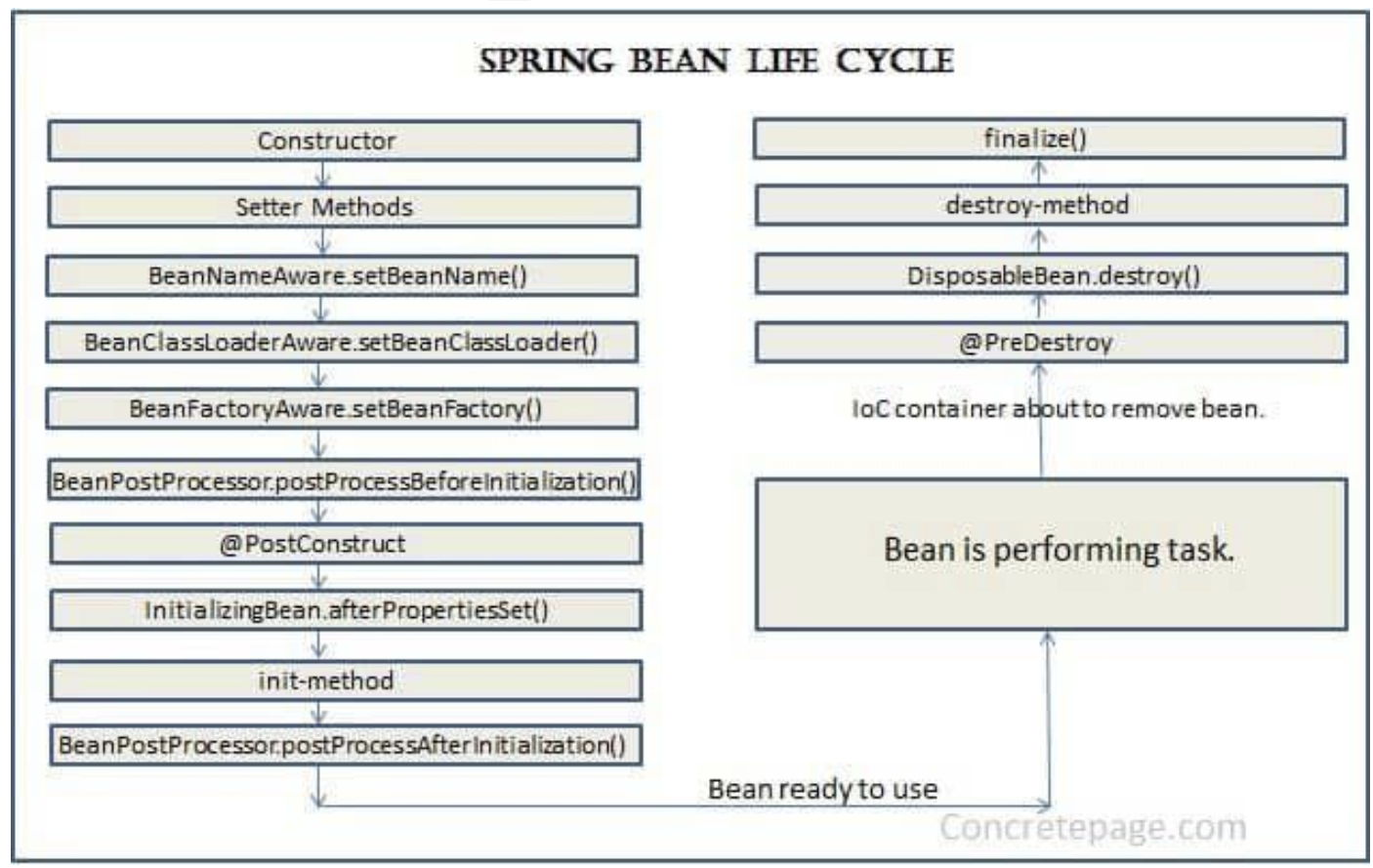
Mức độ ưu tiên @Autowried ưu tiên : setter -> properties -> constructer .

Nếu có hàm Setter thì sẽ inject Bean vào bởi tham số của hàm

@Primary là annotation đánh dấu trên một Bean, giúp nó luôn được ưu tiên lựa chọn trong trường hợp có nhiều Bean cùng loại trong Context.

@Qualifier xác định tên của một Bean mà bạn muốn chỉ định inject.

Vời đời của Bean

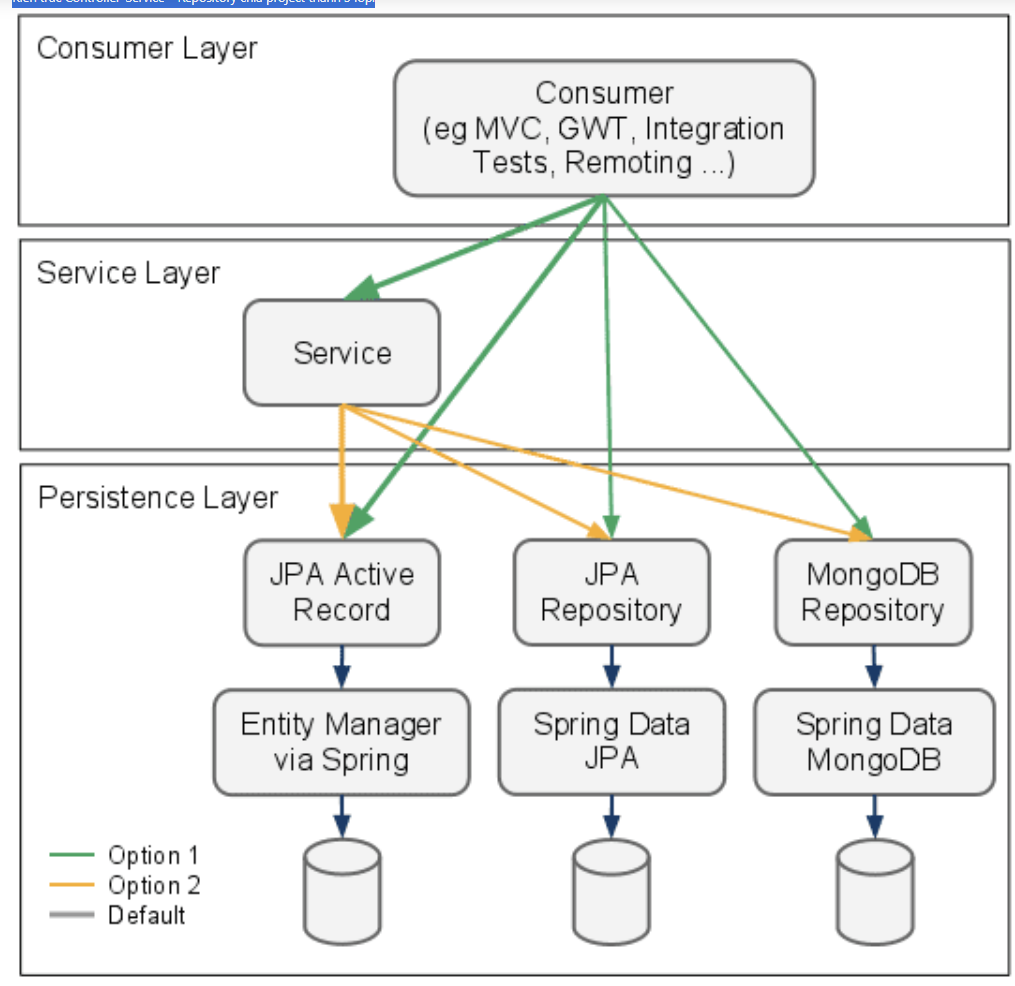


@PreDestroy

@PreDestroy được đánh dấu trên một method duy nhất bên trong Bean. IoC Container hoặc ApplicationContext sẽ gọi hàm này trước khi một Bean bị xóa hoặc không được quản lý nữa.

1. Khi IoC Container (ApplicationContext) tìm thấy một Bean cần quản lý, nó sẽ khởi tạo bằng Constructor
2. inject dependencies vào Bean bằng Setter, và thực hiện các quá trình cài đặt khác vào Bean như setBeanName, setBeanClassLoader, v.v..
3. Hàm đánh dấu @PostConstruct được gọi
4. Tiền xử lý sau khi @PostConstruct được gọi.
5. Bean sẵn sàng để hoạt động
6. Nếu IoC Container không quản lý bean nữa hoặc bị shutdown nó sẽ gọi hàm @PreDestroy trong Bean
7. Xóa Bean.

Kiến trúc Controller-Service – Repository chia project thành 3 lớp:



**Consumer Layer hay Controller:** là tầng giao tiếp với bên ngoài và handler các request từ bên ngoài tới hệ thống.

**Service Layer:** Thực hiện các nghiệp vụ và xử lý logic

**Repository Layer:** Chịu trách nhiệm giao tiếp với các DB, thiết bị lưu trữ, xử lý query và trả về các kiểu dữ liệu mà tầng Service yêu cầu.

* @Service Đánh dấu một Class là tầng Service, phục vụ các logic nghiệp vụ.
* @Repository Đánh dấu một Class Là tầng Repository, phục vụ truy xuất dữ liệu.

Về bản chất @Service và @Repository cũng chính là @Component. Nhưng đặt tên khác nhau để giúp chúng ta phân biệt các tầng với nhau.

* @Service gắn cho các Bean đảm nhiệm xử lý logic
* @Repository gắn cho các Bean đảm nhiệm giao tiếp với DB
* @Component gắn cho các Bean khác.

Trong trường hợp bạn muốn tuỳ chỉnh cấu hình cho **Spring Boot** chỉ tìm kiếm các bean trong một package nhất định thì có các cách sau đây:

1. Sử dụng @ComponentScan
2. Sử dụng scanBasePackages tromg @SpringBootApplication.

Có thể sử dụng multiScan

Đằng sau chương trình, **Spring Boot** lần đầu khởi chạy, ngoài việc đi tìm các @Component thì nó còn làm một nhiệm vụ nữa là tìm các class @Configuration.

1. Đi tìm class có đánh dấu @Configuration
2. Tạo ra đối tượng từ class có đánh dấu @Configuration
3. tìm các method có đánh dấu @Bean trong đối tượng vừa tạo
4. Thực hiện gọi các method có đánh dấu @Bean để lấy ra các Bean và đưa vào `Context.

Ngoài ra, về bản chất, @Configuration cũng là @Component. Nó chỉ khác ở ý nghĩa sử dụng. (Giống với việc class được đánh dấu @Service chỉ nên phục vụ logic vậy).

Hoặc bạn muốn log của chương trình chi tiết hơn. Hãy chuyển nó sang dậng Debug bằng cách config như sau:

logging.level.root=DEBUG

@Value được sử dụng trên thuộc tính của class, Có nhiệm vụ lấy thông tin từ file properties và gán vào biến.

Thư mục mặc định của **Thymeleaf** là resources/templates

@Controller cũng là 1 component