1. IoC là gì, DI là gì?
   1. DI(Dependency Iversion) là gì?

* Là quy tắc lập trình mà trong đó modules cấp cao không nên phụ thuộc vào modul cấp thấp, mà cả module cấp cao và module cấp thấp nên phụ thuộc vào abstraction. Nguyên tắc DI principle quy định các class phải giao tiếp với nhau thông qua interface chứ không được qua implementation.

VD : Có abstraction interface AbstractDAO, các class implementation là MySqlDAO, PostgreSqlDAO.

Ở Service ta có thể dễ dàng thay đổi các module DAO tùy theo nghiệp vụ bằng cách :

AbstractDAO abstractDAO = new MySqlDao();

or

AbstractDAO abstractDAO = new PostgreSqlDAO ();

Service serviceA = new ServiceA(abstractDAO)

\*Vấn đề gặp pahri khi thực hiện nguyên tắn DI principle :

- Gắn nhiều module vào module khác

- Phải quan tâm tới thứ tự khởi tạo mobule

- Phụ thuộc vòng (A phụ thuộc B, B phụ thuộc A => không biết tạo A hay B trước)

=> Sinh ra khái niệm IoC để giải quyết.

* 1. IoC là gì?
     1. Iversion of Control : Là nguyên lý thiết đảo ngược quyền điều khiển so với lập trình hướng thủ tục. Nó tuân thủ nguyên tắc Dependency Inversion

Diagram

Description automatically generated

+ Mô hình không IoC : Class A cần chủ động tạo ra đối tượng lớp class B và C (A nắm quyền khởi tạo, điều khiển).

+ Mô hình IoC : A không tự khởi tạo cũng k chịu trách nhiệm quản lý B và C. Nó nhận được 2 dependency này từ bên ngoài thông qua cơ chế nào đó (Injection).

* IoC nhằm mục đích đơn giản hóa quá trình tạo đối tượng và mối quan hệ giữa chúng bằng cách tuân theo nguyên tắc : Không tạo đối tượng, chỉ mô tả cách chúng được tạo ra.
  + 1. Spring IoC:
* IoC container(spring container) là thành phần thực hiện IoC.
* Trong Spring, spring container sẽ tạo các đối tượng, lắp ráp chúng lại với nhau, cấu hình và quản lý vòng đời của chúng từ lúc tạo ra đến lúc bị hủy.
* Spring container sử dụng DI(Dependency Injection) để quản lý các thành phần, đối tượng để tạo nên 1 ứng dụng. Các thành phần, đối tượng này gọi là Bean.
* 2 kiểu :
  + BeanFactory
  + AplicationContext

BeanFactory và AplicationContext đề là các interface thực hiện IoC container.

AplicationContext được xây dựng từ BeanFactory nhưng có thêm 1 số chức năng mở rộng như tích hợp Spring AOP, xử lý message, context cho web.

VD :

@Component  
@Primary  
class A implements C{}  
  
@Component  
class B implements C{}  
  
interface C{}  
  
@Component  
class D{  
 @Autowired  
 @Qualifier("b")  
 C c;  
}  
  
@Component  
class E{  
 @Autowired  
 C c;  
  
 public E() {  
 }  
  
 @Autowired  
 public E(C c) {  
 this.c = c;  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 ApplicationContext ctx = new AnnotationConfigApplicationContext("qualifier.primary");  
 D d = ctx.getBean(D.class);  
 System.*out*.println(d.c);  
 E e = ctx.getBean(E.class);  
 System.*out*.println(e.c);  
  
 B b = ctx.getBean(B.class);  
 System.*out*.println(b);  
 }  
}

* + 1. DI(Dependency Injection) :
* Là một trong những design pattern được sử dụng để triển khai IoC.
* Nguyên tắc cơ bản : DI làm cho module cấp cao phụ thuộc vào module cấp thấp thông qua abstraction bằng injector(constructor, method, property). Nghĩa là muốn tạo instance của module cấp cao ta phải tạo instance của modul cấp thấp trước(sử dụng module abstraction chứ không phải implementation).
* Cách thức hoạt động : tìm Bean cần inject trong container xem có không, nếu có thì inject vào, nếu chưa có thì tạo mới, bỏ vào container và inject vào module cần.
* Muon tao instance module cap cao thi phai tao instance cua module cap thap trc, no se tim bean trong contrainer xem co khong, neu khong co no se bo vao container va inject vao module can

VD : module cấp cao OrderService cần có 2 module là ProductService và CategoryService

Muốn instance OrderServiceImpl thì DI sẽ tìm trong container IoC các implementation ProductServiceImpl và CategoryServiceImpl rồi inject vào OrderServiceImpl.

1. Bean :

* Là các object trong spring framework, là những module chính của chương trình, được tạo ra và quản lý bởi Spring IoC container.
* Bean Scope :

+ Singleton : Chỉ duy nhất một thể hiện của bean sẽ được tạo cho mỗi container. Đây là scope mặc định cho spring bean. Khi sử dụng scope này cần chắc chắn rằng các bean không có các biến/thuộc tính được share.

VD :

+ Prototype : Một thể hiện của bean sẽ được tạo cho mỗi lần được yêu cầu(request)

+ Request : web

+ Session : web

+ Aplication : web

+ WebSocker: web

VD: Singleton và Prototype

@Service  
class SingletonBean {  
 private String content;  
  
 public SingletonBean(String content) {  
 this.content = content;  
 }  
}  
  
@Component  
class PrototypeBean {  
 private String content;  
  
 public PrototypeBean(String content) {  
 this.content = content;  
 }  
}  
  
@Configuration  
class BeanConfig {  
  
 @Bean  
 public SingletonBean singletonBean(){  
 return new SingletonBean("asd");  
 }  
  
 @Bean  
 @Scope("prototype")  
 public PrototypeBean prototypeBean(){  
 return new PrototypeBean("asd");  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 ApplicationContext applicationContext = new AnnotationConfigApplicationContext("beanscope");  
  
 SingletonBean singletonBean = (SingletonBean) applicationContext.getBean("singletonBean");  
 SingletonBean singletonBean2 = (SingletonBean) applicationContext.getBean("singletonBean");  
  
 PrototypeBean prototypeBean = (PrototypeBean) applicationContext.getBean("prototypeBean");  
 PrototypeBean prototypeBean2 = (PrototypeBean) applicationContext.getBean("prototypeBean");  
  
 System.*out*.println("singletonBean : " + singletonBean.hashCode());  
 System.*out*.println("singletonBean2 : " + singletonBean2.hashCode());  
 System.*out*.println("prototypeBean : " + prototypeBean.hashCode());  
 System.*out*.println("prototypeBean2 : " + prototypeBean2.hashCode());  
 }  
}

Text

Description automatically generated

Diagram

Description automatically generated- Vòng đời của Bean :

+ IoC container tạo Bean bằng cách gọi Constructor(có thể inject các bean vào đây).

+ Gọi setter method để inject bean.

+ Gọi 1 số method khác

+ Method @PostConstruct được gọi sau khi bean được khởi tạo.

+ Bean được sử dụng.

+ Sử dụng xong thì đến quá tình hủy bean.

+ Method @PreDestroy được gọi trước khi bean bị hủy.

+ Destroy bean.

Goi setter method de inject bean, roi goi 1 so method khac, method @preConstructor duoc goi sau khi bean duoc khoi tao, bean duocj su dung, su dung xong den qua trinh huy bean, method predestroy dc goi khi bean bi huy, huy bean

1. Phân biệt @component, @controller, @service, @repository? (Tại sao lại chia @Repo với @Service?)

\*\*\* : Chia layer làm việc cho các bean. Dùng cho class (@Bean dùng cho function)

* @Component : Class dùng cho toàn bộ dự án hoặc 1 profile cụ thể.
* @Repository : Dùng để đánh dấu các class thuộc tầng giao tiếp với database.
  + Annotation @Repository có 1 vai trò đặc biệt dùng để chuyển đổi các database exception sang Spring-based (Các exception mà Spring có thể hiểu và xử lý được) unchecked exception. (Trả lời cho câu hỏi : Tại sao lại chia @Repo với @Service?)

VD : Lấy ví dụ khi chúng ta làm việc với Hibernate mà không sử dụng Spring Template (ví dụ như JdbcTemplate) để thao tác với database. Trong trường hợp này, nếu Hibernate ném ra một exception thì lúc này spring sẽ không chuyển đổi nó sang Spring-based exception.

Tầng repository, phục vụ trao đổi giữ liệu với database

* @Service : Class thuộc về tầng service nắm giữ code xử lý business.

Tầng service, phục vụ các logic nghiệp vụ, code xử lý

* @Controller : Xử lý request đến server.

\*\*\* : @Repository, @Service, @Controller đều được chú thích annotation @Component để Spring tự động quét và đăng ký với Spring container.

1. @Controller và @RestController

- @RestController chỉ đơn giản là kết hợp @Controller và @ResonseBody annotation lại với nhau nhầm loại bỏ việc chúng ta phải chú thích tất cả các hàm bên trong với @ResponseBody annotation.

VD :

@Controller

Text

Description automatically generated

@RestController

Text

Description automatically generated

* @service có thao tác trực tiếp với DB đc k? tại sao?

Trả lời : Có. Tại vì @service cũng đc đánh dấu @component giống repo.

1. Rest, Restfull, API

\*\*\* : Nguyên lý REST và cấu trúc dữ liệu RESTfull.

* Api : là 1 tập các quy tắc và cơ chế mà theo đó, một ứng dụng hay một thành phần sẽ tương tác với một ứng dụng hay thành phần khác . Api có thể trả về Json, xml, …

1 tập các quy tắc và cơ chế trong đó 1 ứng dụng hay thành phần sẽ tương tác với ứng dụng hay thành phần khác

* Rest : là 1 dạng chuyển đổi cấu trúc dữ liệu. Nó sử dụng HTTP method đơn giản để tạo giao tiếp giữa các máy. Vì vậy, thay sử dụng URL cho việc xử lý một số thông tin người dùng, REST gửi một yêu cầu HTTP như GET, POST, DELETE, vv đến một URL để xử lý dữ liệu.
* Restfull api: là một tiêu chuẩn dùng trong việc thiết kế các API cho các ứng dụng web để quản lý các resource.

- Cơ chế hoạt động :

Diagram

Description automatically generated

- Đặc điểm :

+ Authentication request bằng access\_token

+ Status code response

+ Quản lý version : “api/v1/products”

1. So sánh REST và SOAP :

|  |  |  |
| --- | --- | --- |
| STT | SOAP | REST |
| 1) | Soap là một giao thức | Rest là một cách thiết kế kiến trúc |
| 2) | SOAP là từ viết tắt của Simple Object Access Protocol(giao thức truy cập đối tượng đơn giản) | REST viết tắt của REpresentational State Transfer |
| 3) | SOAP can't use REST because it is a protocol | REST có thể dùng các web services sử dụng SOAP vì nó có thể dùng bất kỳ giao thức nào như HTTP, SOAP |
| 4) | SOAP cung cấp các giao diện dịch vụ(services interfaces) cho các thành phần bên ngoài sử dụng | REST sủ dụng đỉa chỉ URI để cung cấp các dịch vụ |
| 5) | JAX-WS là java API cài đặt web services theo giao thức SOAP | JAX-RS là java API cài đặt web services theo kiến trúc RESTful |
| 6) | SOAP định nghĩa các chuẩn và quy tắc chặt chẽ | REST không định nghĩa nhiều chuẩn như SOAP |
| 7) | SOAP sử dụng băng thông và tài nguyên nhiều hơn REST | REST sử dụng băng thông và tài nguyên ít hơn SOAP |
| 8) | SOAP định nghĩa chuẩn bảo mật của riêng nó | RESTful kế thừa chuẩn bảo mật tầng vận tải của giao thức mạng |
| 9) | SOAP chỉ hỗ trợ định dạng dữ liệu XML | REST hỗ trợ các định dạng dữ liệu khác nhau như text, HTML, XML, JSON |
| 10) | SOAP ít được dùng hơn REST | REST được ưa chuộng hơn SOAP |
| 11) | Được thiết kế để dùng trong tính toán phân tán | Thương không được dùng trong môi trường tính toán phân tán |
| 12) | Tin cậy hơn | Ít tin cậy hơn – chẳng hạn, HTTP DELETE có thể trả về trạng thái OK ngay cả khi tài nguyên không được xóa |
| 13) | Hỗ trợ hầu hết các chuẩn bảo mật, tin cậy và giao dịch | Sử dụng tốt với các giao thức như: HTTP, SSL. Các phương thức DELETE và PUT thường bị vô hiệu hóa bởi tường lửa hoặc vấn đề bảo mật |
| 14) | SOAP hỗ trợ cả hai giao thức SMTP và HTTP | REST gắn với giao thức HTTP |

1. Cách tạo 1 Rest API? ( thiết kế API, sử dụng các antonation nào? (Get, post, put, delete)

VD :

@RestController  
@RequestMapping("/api/v1/tests")  
public class RestAPIController {  
 private final CouponService couponService;  
  
 public RestAPIController(CouponService couponService) {  
 this.couponService = couponService;  
 }  
  
 @GetMapping("/")  
 public List<Coupon> getList(){  
 return couponService.getList();  
 }  
  
 //@PostMapping, @PutMapping, @DeleteMapping  
 //@PartVarible, @RequestParam, @RequestBody  
}

1. So sánh giữa mybatis, JPA, hibernate ?
   1. JPA :

* JPA (Java Persistence API) là 1 giao diện lập trình ứng dụng Java, nó mô tả cách quản lý các mối quan hệ dữ liệu trong ứng dụng sử dụng Java Platform.
* Giao dien lap trinh ung dung java, mo ta cach quan ly cac moi quan he du lieu
* JPA cung cấp một mô hình POJO persistence cho phép ánh xạ các table/các mối quan hệ giữa các table trong database sang các class/mối quan hệ giữa các object.
* Giao dien lap trinh ung dung java Cung cap mo hinh pojo persistence cho phep anh xa cac table,mqh giua cac table trong databse sang cac class/mqh giuacac object
  1. Hibernate & Mybatis.
* Đều là ORM framework cho phép người lập trình thao tác với database một cách hoàn toàn tự nhiên thông qua các đối tượng.
* là cài đặt của JPA (JPA là 1 tập các interface, còn Hibernate & Mybatis implements các interface ấy 1 cách chi tiết).
* Khác biệt :

|  |  |
| --- | --- |
| **MyBatis** | **Hibernate** |
| MyBatis đơn giản và gọn nhẹ hơn | Hibernate sinh ra SQL cho bạn, nghĩa là bạn không phải tốn thời gian viết SQL. |
| MyBatis linh hoạt hơn, giúp thời gian phát triển nhanh hơn. | Hibernate có khả năng mở rộng cao, cung cấp bộ nhớ cache tiên tiến hơn. |
| MyBatis sử dụng SQL, là csdl phụ thuộc. | Hibernate sử dụng HQL, tương đối độc lập với csdl. Chuyển đổi db thành Hibernate dễ dàng hơn. |
| MyBatis ánh xạ ResultSet từ JDBC API đến POJO Objects của bạn, vì vậy bạn không cần phải quan tâm đến các cấu trúc bảng. | Hibernate ánh xạ các đối tượng Java POJO của bạn với các bảng. |
| Sử dụng stored procedure trong MyBatis dễ hơn. | Sử dụng stored procedure trong Hibernate khó hơn. |

* Khi nào sử dụng MyBatis?

+ Bạn muốn tạo ra SQL của riêng bạn và bạn sẵn sàng bảo trì chúng.

+ Môi trường của bạn được định hướng bởi mô hình dữ liệu quan hệ.

Lam db rieng, moi truong duoc dinh huong moi hinh quan he du lieu, lam viec voi schema phuc tap

+ Bạn phải làm việc trên các schema hiện tại và phức tạp.

* Khi nào sử dung Hibernate?

+ Sử dụng Hibernate, nếu môi trường được định hướng bởi mô hình đối tượng và cần tạo ra SQL tự động.

Moi truong dinh huong boi mo hinh doi tuong va can tao sql tu dong