|  |  |
| --- | --- |
| 1 | Trình bày cách Spring MVC xử lý request |
| 2 | Trình bày cơ chế Dependency Injection trong Spring |
| 3 | Có bao nhiêu cách để thực hiện Dependency Injection? Trình bày? |
| 4 | Framework là gì ? Framework khác Library chỗ nào ? |
| 5 | Spring Framework là gì ? |
| 6 | Lợi ích của Spring Framework? |
| 7 | Nguyên lý đảo ngược quyền điều khiển (Inversion of Control) là gì ? |
| 8 | Bean là gì? |
| 9 | Trong Spring có bao nhiêu Bean Scope? |
| 10 | @Autowire là gì? |
| 11 | @Component có ý nghĩa gì? |
| 12 | Trình bày ý nghĩa của Controller |
| 13 | Trình bày ý nghĩa của ModelAndView Interface |
| 14 | Trình bày ý nghĩa của ModelMap |
| 15 | Trình bày ý nghĩa của ViewResolver Interface |
| 16 | Phân biệt POST với PUT thường sử dụng để làm gì? |
| 17 | @RequestMapping làm gì? |
| 18 | Thuộc tính consumes trong các Request Mapping là gì ? |
| 19 | Thuộc tính produces trong các Request Mapping là gì ? |
| 20 | Trình bày cơ chế Data Binding trong Spring |
| 21 | So sánh RequestParam và PathVariable ? |
| 22 | Thymeleaf là gì? |
| 23 | Sử dụng lặp trong Thymeleaf như thế nào? Sử dụng điều kiện trong Thymeleaf như thế nào? |
| 24 | Formatter là gì ? Converter là gì ? |
| 25 | ORM là gì ? |
| 26 | JPA là gì? Spring Data JPA là gì ? |
| 27 | Hibernate là gì ? Hibernate làm gì với database |
| 28 | Liệt kê một số annotation của hibernate? |
| 29 | Entity là gì? |
| 30 | Persistence Context & Entity Manager là gì? |
| 31 | Câu lệnh truy vấn động là gì? Câu lệnh truy vấn tĩnh là gì? |
| 32 | Trạng thái của Entity bao gồm những gì? |
| 33 | Có bao nhiêu loại mapping trong Hibernate? |
| 34 | Để sử dụng interface JpaRepository cần cung cấp những thông tin nào? |
| 35 | Validation dữ liệu là gì? |
| 36 | Trình bày cách triển khai validate dữ liệu trong Spring |
| 37 | Binding Result là gì? |
| 38 | AOP là gì? AOP gồm những yếu tố nào? |
| 39 | Các loại Advice? |
| 40 | JoinPoint là gì? |
| 41 | Nêu cách triển khai AOP mà bạn dùng trong dự án |
| 42 | Mục đích xử lý ngoại lệ trong Spring Web MVC ? |
| 43 | Các cách xử lý ngoại lệ trong Spring Web MVC ? |
| 44 | Cookie là gì? |
| 45 | Phân biệt Session và Cookie |
| 46 | Session là gì ? Cho ví dụ về session ? |
| 47 | Nêu cách triển khai Session mà bạn dùng trong dự án |
| 48 | Web Service là gì? Lấy ví dụ về web service |
| 49 | Phân biệt Web Service và Web truyền thông |
| 50 | SOAP là gì? RESTful là gì? |
| 51 | jQuery là gì? Ajax là gì? |
| 52 | Nêu ví dụ về cách bạn triển khai Ajax trong dự án của bạn |
| 53 | @RequestHeader và @ResponseHeader có ý nghĩa gì |
| 54 | I18n và L10n là gì? |
| 55 | Nêu cách triển khai I18N và I10N trong dự án Spring |
| 56 | Interceptor là gì? |
| 57 | Bảo mật là gì? Cơ chế bảo mật trong Spring như thế nào? |
| 58 | Authentication là gì ? Authorization là gì ? |
| 59 | Các cơ chế xác thực ? |
| 60 | CSRF là gì |
| 61 | CORS là gì |
| 62 | Spring Boot là gì? |
| 63 | Sự khác nhau giữa SpringBoot và SpringMVC là gì? |
| 64 | Cấu hình cho Spring Boot Tìm kiếm các Bean ở nhiều package khác nhau bằng cách nào? |
| 65 | Trình bày Spring Security? @EnableWebSecurity làm gì? |
| 66 | Trình bày Spring Boot JPA? Cài đặt? |

1, Trình bày cách Spring MVC xử lý request.

+ Luồng thực thi:

🡪 Người dùng điền tên website cần truy cập trên brower 🡪 nhấn Enter

🡪 gửi request lên server (nơi triển khai ứng dụng Spring)

🡪 DispatcherServlet tiếp nhận request

🡪 chuyển tới controller tương ứng (thông qua các cơ chế mapping – ánh xạ đường dẫn)

🡪 controller nhận yêu cầu và gọi phương thức thích hợp (GET / POST)

🡪 controller tiến hành render dữ liệu và trả về tên view cho DispatcherServlet.

🡪 DispatcherServlet sẽ nhận trợ giúp từ ViewResolver để xác định view cho các yêu cầu.

🡪 DispatcherServlet chuyển dữ liệu tới view đó 🡪 chuyển tới trình duyệt

2,Trình bày cơ chế Dependency Injection trong Spring

- DI (Dependency Injection) – là design pattern cho phép xóa bỏ sự phụ thuộc, là một cơ chế để triển khai IoC, trong đó thao tác tiêm (inject) vào các đối tượng khác được thực hiện bởi container hoặc framework.

(Giống với Nguyên tắc D trong S.O.L.I.D là Dependency inversion principle - Nguyên lý đảo ngược phụ thuộc: các module cấp cao không nên phụ thuộc vào các module cấp thấp, cả 2 nên phụ thuộc vào abstraction).

=> Giảm sự phụ thuộc giữa các module, object giúp cho việc mở rộng code sau này được dễ dàng.

=> Giúp cho việc quản lý và kiểm thử ứng dụng dễ dàng hơn.

=> Giúp chúng ta tập trung vào việc viết logic nghiệp vụ của ứng dụng, còn việc tạo và quản lý các đối tượng đc giao cho framework lo.

🡺 Có 3 cách triển khai:

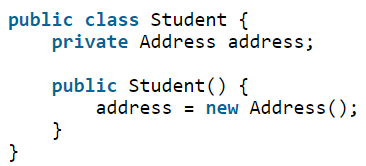
+ Constructor - Dùng inject (tiêm) các module bắt buộc. Các module được tiêm nằm trong constructor, và được gán lần lượt vào các field.

+ Setter method - Dùng inject các module tùy chọn. Mỗi module sẽ được inject thông qua setter, nằm ở tham số và cũng gán cho field nào đó.

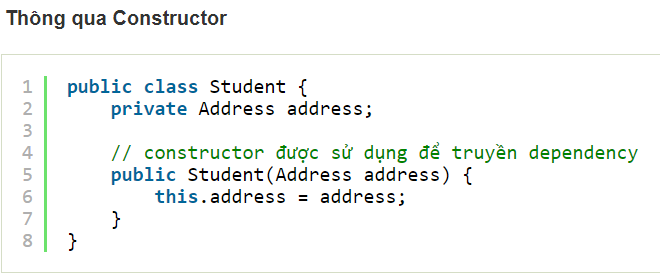
+ Interface (field) - Các client phải triển khai một Interface mà có một setter method dành cho việc nhận dependency.

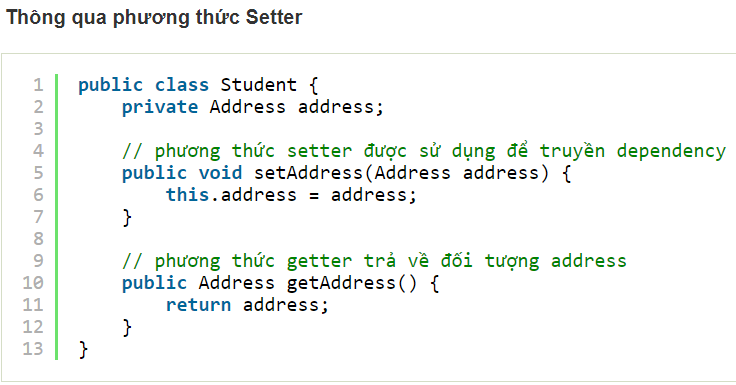
🡺 Ví dụ:

+ Khi không sử dụng DI: đối tượng address sẽ bị phụ thuộc vào Student. Mỗi khi muốn thay đổi đối tượng này thì phải thay đổi đối tượng student.



+ Khi sử dụng DI: để student và address giảm phụ thuộc vào nhau, chúng ta có thể khởi tạo đối tượng address ở bên ngoài lớp Student và truyền vào thông qua Constructor hoặc phương thức Setter.





3,Có bao nhiêu cách để thực hiện Dependency Injection? Trình bày?

🡺 Có 3 cách triển khai:

+ Constructor - Dùng inject (tiêm) các module bắt buộc. Các module được tiêm nằm trong constructor, và được gán lần lượt vào các field.

+ Setter method - Dùng inject các module tùy chọn. Mỗi module sẽ được inject thông qua setter, nằm ở tham số và cũng gán cho field nào đó.

+ Interface (field) - Các client phải triển khai một Interface mà có một setter method dành cho việc nhận dependency.

4,Framework là gì ? Framework khác Library chỗ nào ?

- Framework – là 1 ứng dụng phần mềm có tính trừu tượng cao, cung cấp các tính năng chung thông dụng nhất và có khả năng tùy biến.

(framework ràng buộc, ép người dùng phải theo đúng luật chơi của nó, nó nắm quyền điều khiển nên phải code sao cho nó hiểu, ví dụ phải đứng ở trạm xe buýt thì mới xe mới đón khách,…).

- Phân biệt Framework/Library:

+ Giống: đều cung cấp tính năng viết sẵn để có thể tái sử dụng.

+ Khác: Điểm khác biệt chính giữa framework và library chính là cơ chế IoC.

|  |  |
| --- | --- |
| Library | Farmework |
| - Ứng dụng nắm quyền điều khiển (code của mình sẽ gọi code của thư viện).  - Là tập hợp các tính năng, các lớp.  - Hoạt động bị động.  - Sử dụng trực tiếp mà không cần thay đổi cấu trúc dự án.  - Không có cơ chế IoC. | - Framework nắm quyền điều khiển ứng dụng (code của framework sẽ gọi code của mình).  - Tập hợp các thư viện.  - Hoạt động chủ động.  - Sử dụng framework phải thay đổi cấu trúc dự án.  - Có cơ chế IoC. |

5,Spring Framework là gì ?

- Spring Framework - là một trong những khung ứng dụng dựa trên Java phổ biến nhất (đc Rod Johnson phát triển vào năm 2003).

Là một khung mã nguồn mở có thể được sử dụng để phát triển các ứng dụng Java một cách dễ dàng và với tốc độ nhanh chóng.

Nó là một khung công tác nhẹ cũng cung cấp hỗ trợ cơ sở hạ tầng được xác định rõ ràng để phát triển các ứng dụng trong Java.

Nói cách khác, có thể nói rằng Spring xử lý cơ sở hạ tầng để LTV có thể tập trung hơn vào việc phát triển ứng dụng của mình.

6,Lợi ích của Spring Framework?

- Lợi ích của Spring Framework:

+ Tính đơn giản và kiểm thử tốt.

+ Đảo ngược quyền điều khiển (IoC).

+ Trọng lượng nhẹ (khung cơ bản Spring chỉ có kích thước khoảng 1MB và chi phí xử lý của Spring không quá nhiều).

+ Hỗ trợ lập trình hướng khía cạnh (AOP).

+ Xây dựng ứng dụng web theo mô hình MVC.

+ Cấu hình và vòng đời của các đối tượng của ứng dụng được quản lý bởi Spring framework.

+ Xử lý ngoại lệ tốt.

7,Nguyên lý đảo ngược quyền điều khiển (Inversion of Controller) là gì ?

- IoC (Inversion of Controller) – “nguyên lý đảo ngược quyền điều khiển”, là nguyên lý trong phát triển phần mềm, trong đó việc điểu khiển các đối tượng hoặc thành phần của hệ thống đc thực hiện bởi framework hoặc các container (LTV không cần tạo đối tượng thay vào đó họ chỉ cần mô tả rằng nó sẽ được tạo như thế nào, framework sẽ lo phần còn lại).

8,Bean là gì?

- Bean – là những module chính của chương trình, là 1 đối tượng đc quản lý bởi Spring IoC container (do framework tự khởi tạo, tự quản lý vòng đời của đối tượng).

9, Trong Spring có bao nhiêu Bean Scope?

- Có 5 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.

+ Prototype – khác với Singleton, 1 thể hiện của bean sẽ được tạo cho mỗi lần được yêu cầu (request).

+ Request – giống với prototype scope, tuy nhiên nó dùng cho ứng dụng web, 1 thể hiện của bean sẽ được tạo cho mỗi HTTP request.

+ Session – mỗi thể hiện của bean sẽ được tạo cho mỗi HTTP Session.

+ Global-Session – được sử dụng để tạo global sesion bean cho các ứng dụng Portlet.

10, @Autowire là gì?

- @Autowired – tự động nhúng các bean được Spring Container sinh ra vào Class có khai báo @Autowire.

11,@Component có ý nghĩa gì?

- @Component – là một annotation của class. Nó dùng để đánh dấu class Java là một bean. Khi Spring start thì nó quét qua các annotation có đánh dấu là @Component thì nó sẽ tạo bean cho class đó.

12, Trình bày ý nghĩa của Controller ?

- Ý nghĩa của Controller:

Trong mô hình MVC, controller là nơi nhận request từ người dùng, xử lý request, xây dựng dữ liệu cho view (model) và chọn view để trả lại kết quả của cho người dùng.

13, Trình bày ý nghĩa của ModelAndView Interface?

+ ModelAndView – Là 1 interface, là sự kết hợp của 2 khía cạnh truyền dữ liệu và view.

14, Trình bày ý nghĩa của ModelMap ?

+ ModelMap – là đối tượng (1 class) implement Map interface trong Java, đc collection tích hợp thêm các phương thức của map.

=> Model và ModelMap chỉ gửi dữ liệu.

15, Trình bày ý nghĩa của ViewResolver Interface ?

+ ViewResolver (trình phân giải view) – là cơ chế ánh xạ “tên” của view sang đối tượng view tương ứng.

16, Phân biệt POST với PUT thường sử dụng để làm gì?

Trả lời :

POST: thêm mới dữ liệu

PUT: chỉnh sửa dữ liệu

|  |  |
| --- | --- |
| **POST** | **PUT** |
| - Thêm mới.  - Đc dùng để gửi các **request** kèm theo một **entity** đến **server** yêu cầu tạo một tài nguyên mới dựa trên **entity** được cung cấp.  - Nếu gọi **POST method** nhiều lần với cùng một **entity** thì nó sẽ tạo ra nhiều tài nguyên ở phía **server** có giá trị tương tự nhau. | - Cập nhật.  - Gửi kèm theo một định danh (thường là **ID**) cùng với 1 **entity**. Nếu một tài nguyên được tìm thấy bởi mã định danh kèm theo thì tài nguyên này sẽ được **thay thế** bởi các giá trị trong **entity** kèm theo.  - Nếu gọi **PUT method** nhiều lần thì sẽ nó tạo hoặc cập nhật cùng một tài nguyên (dựa vào mã định danh). |

17, @RequestMapping làm gì?

- @RequestMapping – đc dùng để ánh xạ các request tới các xử lý ở controller tương ứng.

18, Thuộc tính consumes trong các Request Mapping là gì ?

19, Thuộc tính produces trong các Request Mapping là gì ?

Thuộc tính:

+ value (URL pattern).

+ method: GET, POST, PUT, PATCH, DELETE,…

+ consumes: định dạng kiểu dữ liệu khi client request tới server (tương ứng với cài đặt “content- type” của request header).

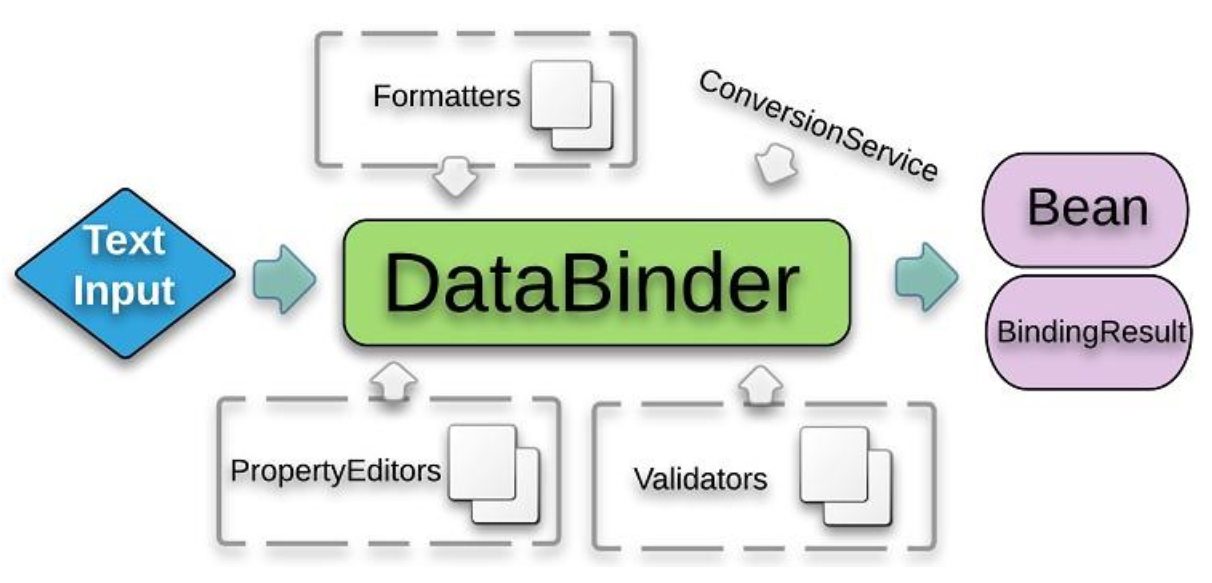
+ produces: định nghĩa kiểu dữ liệu trả về cho client (tương ứng với cài đặt “accept” của request header).

20, Trình bày cơ chế Data Binding trong Spring

- Data binding – là cơ chế liên kết dữ liệu đầu vào / đầu ra với đối tượng model.

+ Các form đều đc liên kết với 1 đối tượng biểu diễn dữ liệu phía sau 🡪 dữ liệu trên form sẽ đc tự động chuyển đổi thành các thuộc tính của đối tượng liên kết form.

+ Data binding hỗ trợ chuyển đổi dữ liệu và validate, hoạt động dựa trên data binder.



- Data binder:

+ Mỗi đối tượng như là 1 form, mỗi thuộc tính tương ứng với 1 ô input.

+ Text input sẽ đc mapping với data binder thông qua propertyEditors (thông qua getter/setter).

+ ConversionService giúp convert (chuyển đổi) dữ liệu 1 cách tự động.

+ Formatters giúp can thiệp sâu về cấu hình (ví dụ như validate dữ liệu ngày, giờ).

+ Validators giúp validate trực tiếp thuộc tính của đối tượng.

+ Đầu ra sẽ là 1 Bean 🡪 gán giá trị vào đối tượng (bắt đối tượng thông qua ModelAttribute).

21, So sánh RequestParam và PathVariable ?

|  |  |
| --- | --- |
| @RequestParam | @PathVariable |
| * Đều có thể dùng để lấy dữ liệu từ client gửi đến server | |
| * Có thể nhận dữ liệu từ cả phương thức Get hoặc Post * Dữ liệu được lấy theo cặp key-value * Dùng nhiều trong web truyền thống | * Chỉ có thể lấy dữ liệu từ phương thức Get * Chỉ cần lấy value * Dùng nhiều trong web service |

22, Thymeleaf là gì?

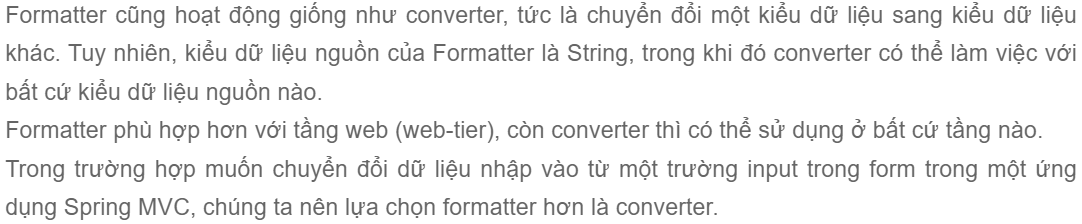
- Thymeleaf – là 1 bộ xử lý view, đc sử dụng cho các ứng dụng web và các ứng dụng độc lập.

+ Được sử dụng cho phù hợp với tiêu chuẩn web, hỗ trợ HTML 5.

+ Thymeleaf cho phép hỗ trợ các template: HTML, XML, JS, CSS, TEXT,...

23, Sử dụng lặp trong Thymeleaf như thế nào? Sử dụng điều kiện trong Thymeleaf như thế nào?

24, Formatter là gì ? Converter là gì ?



25, ORM là gì ?

- ORM (Object-Relational Mapping) – là kỹ thuật liên kết giữa đối tượng trong lập trình với đối tượng trong CSDL (là quá trình biến đổi giữa 1 record trong DB và 1 object trong ngôn ngữ lập trình OOP).

+ ORM ra đời để khắc phục nhược điểm của JDBC, như:

- Cấu hình thủ công (cần phải chuẩn bị DB trước).

- Có kiến thức nền tảng về SQL.

- Phải biết kiểu dữ liệu đầu ra / đầu vào của từng column.

+ ORM có rất nhiều triển khai (Hibernate, Eclipselink, Ibatis,…), trong đó Hibernate đc xem là ORM Framework nổi tiếng nhất và sử dụng rộng rãi trong Java.

26, JPA là gì? Spring Data JPA là gì ?

- JPA (Java Persistence API) – là 1 bộ đặc tả tiêu chuẩn của Java, trong đó mô tả các thao tác quản lý dữ liệu trong Java cung cấp các mô hình cho phép ánh xạ các bảng trong CSDL với các class.

(JPA chỉ chứa interface hướng dẫn triển khai ORM, việc implement sẽ do ORM framework thực hiện).

- Spring Data Jpa – là 1 phần của dự án Spring Data, có thể sử dụng Spring Data JPA với các ORM framework như Hibernate, Open JPA,…

Là tầng được xây dựng phía trên tầng JPA. Có nghĩa là nó sẽ sử dụng tất cả các chức năng của JPA như entity mapping, JPA query, xây dựng một số tính năng như khởi tạo query thông qua tên hàm,…

🡺 Giúp cải tiến JPA tiêu chuẩn, đơn giản hóa tầng truy xuất dữ liệu, tự tạo repository, tạo các truy vấn JPA thông quá tên phương thức.

27,Hibernate là gì ? Hibernate làm gì với database ?

- Hibernate - là một ORM Framework mã nguồn mở giúp LTV viết ứng dụng Java có thể mapping các Object với hệ CSDL, và hỗ trợ thực hiện các khái niệm OOP với CSDL.

28, Liệt kê một số annotation của hibernate?

* @Entity, @Table, @Id, @GenerateValue, @Column, @OneToOne, @OneToMany, @ManyToOne, @ManyToMany

29, Entity là gì?

- Entity – là đối tượng đại diện cho dữ liệu trong ứng dụng.

+ Entity thường là POJO (Plain Old Java Object), nghĩa là lớp đơn giản, chỉ mô tả dữ liệu, không có xử lý logic nghiệp vụ.

+ Mỗi Entity đc mapping với 1 table trong CSDL (mapping 1-1).

+ Mỗi Entity cần phải tuân thủ:

- Được gắn annotation @Entity, @Id.

- Có Constructor là public không tham số.

- Không được khai báo final.

- Thuộc tính có access modifier là private, protected, hoặc ở mức package private (tức là không phải public).

30, Persistence Context & Entity Manager là gì?

31,Câu lệnh truy vấn động là gì? Câu lệnh truy vấn tĩnh là gì?

|  |
| --- |
| Câu lệnh truy vấn động là câu truy vấn có các tham số thay đổi |
| Câu lệnh truy vấn tĩnh là câu truy vấn có các tham số cố định |

32,Trạng thái của Entity bao gồm những gì?

- Trạng thái của các Entity (4 trạng thái):

+ Transient (Tạm thời): Đối tượng không có quan hệ với Session hiện tại của Hibernate. Đối tượng ở trạng thái này chưa từng gắn vào context, nó không có bản ghi tương ứng trong CSDL.

+ Persistent (Bền vững): Đối tượng đang liên hệ với một context, tức là với một đối tượng Session và trạng thái của nó được đồng bộ với cơ sở dữ liệu khi mà ta commit cái Session.

+ Detached (Đã bị tách riêng ra): Đối tượng đã từng có trạng thái persistent nhưng hiện tại đã không còn giữ quan hệ với Session. Nếu nó không được attached trở lại, nó sẽ bị bộ gom rác của Java quét đi theo cơ chế thông thường.

+ Removed (Đã bị xóa): tương tự như detached nhưng bản ghi tương ứng với đối tượng này trước đó đã bị xóa khỏi database.

*33, Có bao nhiêu loại mapping trong Hibernate?*

+ Association Mappings (hay Entity Mappings) – là ánh xạ các liên kết giữa các lớp entity và các mối quan hệ giữa các bảng chính, gồm 4 kiểu liên kết:

- Many-to-One.

- One-to-Many.

- One-to-One.

- Many-to-Many.

***34, Để sử dụng interface JpaRepository cần cung cấp những thông tin nào?***

***35, Validation dữ liệu là gì?***

- **Validation** – thực hiện **đánh giá / xác minh** tính hợp lệ của dữ liệu đầu vào (đối với mọi ứng dụng **web**, dữ liệu luôn quan trọng hơn công nghệ).

***36, Trình bày cách triển khai validate dữ liệu trong Spring?***

|  |
| --- |
| 1. Implement interface Validator vào model cần Validate |
| 2. Overide các phương thức của Interface theo hình thức regex dữ liệu |
| 3. Sử dụng các annotation Validate có sẵn: @NotNull, @NotEmpty, @NotBlank,… |
| 4. Sử dụng @Validated, @Valid, BindingResult ở Controller để thực hiện trả về kết quả khi có lỗi |

***37,Binding Result là gì?***

(**BindingResult** là 1 đối tượng, là nơi **Spring** chứa các kết quả (lỗi) trong quá trình **Validate**).

***38, AOP là gì? AOP gồm những yếu tố nào?***

- **AOP** (**Aspect Oriented Programming**) – là kỹ thuật lập trình hướng khía cạnh, nhằm tách chương trình chính thành các **module** riêng lẻ, phân biệt và không phụ thuộc lẫn nhau (nó chỉ hỗ trợ cho **OOP**, và có phụ thuộc vào **OOP**).

|  |
| --- |
| AOP gồm các yếu tố: |
| + Aspect: dùng để khai báo cho các đoạn chương trình độc lập, không liên quan gì đến nghiệp vụ dự án |
| + Advice: là các hành động mà Aspect dùng để quan sát OOP |
| + Pointcut: là tập hợp các Joinpoint có khả năng xảy ra các hoạt động Advice |
| + Joinpoint: những điểm xảy ra Advice |

***39,*** ***Các loại Advice?***

- **Advice** (**lời khuyên**) – là 1 hành động cụ thể của **Aspect**.

+ **Before advice** – thực thi trước khi **Join point**.

+ **After returning advice** – thực hiện khi **Join point** hoàn thành bình thường.

+ **After throwing advice** – thực hiện khi **Join point** kết thúc có ngoại lệ.

+ **After** (**finally**) – thực thi sau khi **Join point** kết thúc, dù thành công hay thất bại.

+ **Around advice** – thực thi trước và sau khi **join point**.

***40,*** ***JoinPoint là gì?***

- **Join point** (**điểm gia nhập**) – là 1 vị trí nào đó trong luồng thực thi của chương trình mà **advice** sẽ đc đan (**weave**) vào chương trình chính.

***41,*** ***Nêu cách triển khai AOP mà bạn dùng trong dự án?***

***42,*** ***Mục đích xử lý ngoại lệ trong Spring Web MVC ?***

***43, Các cách xử lý ngoại lệ trong Spring Web MVC ?***

- **Các phương thức xử lí ngoại lệ:**

+ Sử dụng **try-catch-finally** (**throw**, **throws**).

+ Sử dụng **@ExceptionHandle** để bắt 1 **Exception** nào đó.

Nếu có **exception** ở bất cứ đâu, sẽ được chuyển tới **method** có **@ExceptionHandler** tương ứng. Thứ tự đặt chúng không quan trọng, **Spring** sẽ tự động tìm cái phù hợp nhất, nếu không có thì chuyển dần lên các **exception** class cha (do đó, nên có một **@ExceptionHandler** để bắt **Exception class**, dành cho các **exception** còn lại).

Các **method** này viết tương ứng với **method** của **Controller**, nhưng thay vì trả **data** về thì chúng ta trả về **message** lỗi, hoặc 1 trang thông báo đến người dùng.

(có thể sử dụng **@RestControllerAdvice** hoặc **@ControllerAdvice** để định nghĩa 1 lớp nào đó để đón và xử lý ngoại lệ, thay vì phải xử lý trực tiếp ở **controller**).

***44, Cookie là gì?***

- **Cookie** – là những trang **web** do người dùng truy cập tạo ra, giúp trãi nghiệm trực tuyến dễ dàng hơn

(ví dụ như ghi nhớ nội dung tùy chọn các dữ liệu đã nhập của trang **web**, thông tin lịch sử truy cập **web**

=> giúp cho **website** đề xuất những nội dung phù hợp với người dùng).

***45, Phân biệt Session và Cookie?***

- **So sánh cookie và session:**

+ **Giống:** đều dùng để lưu thông tin tạm thời.

+ **Khác:**



***46, Session là gì ? Cho ví dụ về session ?***

- **Session** – là phiên làm việc giữa **client** và **server**.

+ Thông thường, sử dụng **session** để lưu trữ thông tin đăng nhập, giỏ hàng, hoặc những dữ liệu mang tính chất tạm thời.

+ 1 **session** đc bắt đầu khi **client** thực hiện **request** lần đầu tiên đến **server** và kết thúc khi **client** dừng làm việc với **server** (đóng trình duyệt, hết time **session**, **server** bị tắt,…).

***47, Nêu cách triển khai Session mà bạn dùng trong dự án?***

***48, Web Service là gì? Lấy ví dụ về web service?***

- **Webservice** – là ứng dụng **web** cung cấp nền tảng cho nhiều ứng dụng khác (giá vàng, chứng khoáng, dự báo thời tiết,…), những ứng dụng này có thể đc viết từ nhiều nền tảng công nghệ khác nhau, nhưng có thể giao tiếp với nhau (thông qua **data**).

+ Kiểu dữ liệu dùng để giao tiếp thường là **XML** hoặc **JSON**.

+ Phân loại:

- **SOAP** (Simple Object Access Protocol) – là giao thức **Internet** cho phép các chương trình trong các hệ điều hành khác nhau hoặc trên các máy chủ riêng biệt liên kết với nhau qua **Internet**. Giao thức sử dụng **XML** (cú pháp nghiêm ngặt, môi trường đóng do dùng trong doanh nghiệp).

- **REST** (Representation State Transfer) – là một tiêu chuẩn trong thiết kế **API** cho các ứng dụng web để quản lý **resource**, sử dụng phương thức **HTTP** đơn giản để giao tiếp giữa các máy. Giao thức sử dụng **XML** hoặc **JSON** (tốc độ nhanh hơn nhưng độ bảo mật và an toàn thấp hơn **SOAP**).

+ Ví dụ: những trang web đc tích hợp bản đồ, giá vàng, chứng khoáng, dự báo thời tiết, biểu đồ thống kê số ca mắc covid,…

***49, Phân biệt Web Service và Web truyền thống?***

**So sánh Web application và Web service:**

|  |  |
| --- | --- |
| **Web application** | **Web service** |
| - Có giao diện.  - Tương tác giữa người dùng và ứng dụng, thông qua **form**, **link**, **image**, **button**,…  - Kết quả trả về cho người dùng là **view** hình ảnh, âm thanh, video,… | - Không có giao diện.  - Tương tác giữa các ứng dụng với nhau, thông qua các phương thức **HTTP** (GET, POST, PUT, DELETE,…) và **HTTP status code**.  - Kết quả trả về cho người dùng chỉ là data **XML** hoặc **JSON** (khó đọc đối với người dùng). |

**50, SOAP là gì? RESTful là gì?**

+ Phân loại:

- **SOAP** (Simple Object Access Protocol) – là giao thức **Internet** cho phép các chương trình trong các hệ điều hành khác nhau hoặc trên các máy chủ riêng biệt liên kết với nhau qua **Internet**. Giao thức sử dụng **XML** (cú pháp nghiêm ngặt, môi trường đóng do dùng trong doanh nghiệp).

- **REST** (Representation State Transfer) – là một tiêu chuẩn trong thiết kế **API** cho các ứng dụng web để quản lý **resource**, sử dụng phương thức **HTTP** đơn giản để giao tiếp giữa các máy. Giao thức sử dụng **XML** hoặc **JSON** (tốc độ nhanh hơn nhưng độ bảo mật và an toàn thấp hơn **SOAP**).

***51, jQuery là gì? Ajax là gì?***

- **Jquery** – là thư viện đc xây dựng từ **JS**, với câu slogan **“Write less, do more”**.

Với các tính năng nổi bật:

+ Thao tác với **DOM** dễ dàng, nhanh chóng (cách truy cập **DOM** gần giống với cú pháp **CSS**).

+ Xử lý **event** tốt.

+ Xử lý **animation** dùng để tạo hiệu ứng chuyển động.

+ Hỗ trợ **Ajax**.

+ Gọn nhẹ, tích hợp **file JS**.

+ Hỗ trợ hầu hết các loại trình duyệt.

- **Ajax** – là 1 kỹ thuật giúp phát triển trang **web** động mà không cần tải lại trang.

+ Tạo các trang **web** nhanh hơn, mượt hơn.

+ Cập nhật nội dung của 1 phần trang **web** mà không cần tải lại trang.

=> Giúp trải nghiệm người dùng tốt hơn.

+ Cú pháp:

**$.ajax({**

**url:** "địa chỉ trang web cần tương tác”,

**type:** "phương thức HTTP method",

**dataType:** "kiểu dữ liệu",

**success: function (**response**) {** //xử lý kết quả trả về khi thành công; **}**

**})**

***52, Nêu ví dụ về cách bạn triển khai Ajax trong dự án của bạn ?***

🡪 Ở **controller**, khai báo **annotation** **@CrossOrigin** (cho phép truy cập các đường dẫn khác).

🡪 Ở **view**, có thể dùng **jQuery** để bắt sự kiện.

🡪 Dùng **$.ajax** để triển khai **Ajax** thông qua các thuộc tính: **url**, **dataType**, **type**, **success**.

**53, @RequestHeader và @ResponseHeader có ý nghĩa gì ?**

- **@RequestHeader** và **@ResponseHeader** đều đc sử dụng trong **WebService** (không có giao diện).

+ **@RequestHeader** – lấy thông tin ở trong **RequestHeader** 🡪 gửi **request** (có đính kèm thông tin phía **client** như ngôn ngữ, method,…) lên **server**.

+ **@ResponseHeader** – đưa thông tin từ **server** vào trong **request** người dùng.

***54, I18n và L10n là gì?***

***55, Nêu cách triển khai I18N và I10N trong dự án Spring?***

***56, Interceptor là gì?***

***57, Bảo mật là gì? Cơ chế bảo mật trong Spring như thế nào?***

- **Security** (**bảo mật**) – là sự hạn chế khả năng lạm dụng tài nguyên và tài sản. Gồm cơ chế xác thực và phân quyền.

***58, Authentication là gì ? Authorization là gì ?***

- **Authentication** (**xác thực**) – là 1 cơ chế / hành động nhằm thiết lập hoặc chứng thực 1 thông điệp / đối tượng là đáng tin cậy.

- **Authorization** (**phân quyền** / **cấp quyền**) – là quá trình xác định xem 1 đối tượng / người dùng có quyền truy cập vào 1 tài nguyên cụ thể nào đó để thực hiện 1 số hành động **CRUD**.

***59, Các cơ chế xác thực ?***

+ **HTTP basic**: yêu cầu client cung cấp username và password để thực hiện xác thực.

🡪 **username** và **password** đc gửi cùng với mọi yêu cầu, có khả năng hiển thị chúng, ngay cả khi đc gửi qua kết nối an toàn.

🡪 yêu cầu thay đổi mật khẩu thường xuyên.

+ **Cookies**.

🡪 cần phải nỗ lực nhiều hơn để giảm thiểu các cuộc tấn công **CSRF**.

🡪 không tương thích với **REST**, vì nó đưa vào một trạng thái thành một giao thức không trạng thái

+ **Oauth 2** (Open Authorization): xác thực thông qua ứng dụng thứ 3.

+ **Token**: JWT (Json Web Token).

🡪 là 1 tiêu chuẩn mở định nghĩa cách thức truyền tin an toàn giữa các thành viên bằng 1 đối tượng **JSON**. Thông tin này có thể được xác thực và đánh dấu tin cậy nhờ vào "chữ ký" đã đc mã hóa.

+ **Signatures**.

🡪 là mô hình sử dụng các kỹ thuật mật mã để gắn với mỗi người sử dụng 1 cặp khóa công khai – bí mật và qua đó có thể ký các văn bản điện tử cũng như trao đổi các thông tin mật.

+ **One time password** (OTP).

🡪 người dùng nhập **username** và **password**, sau đó cả máy chủ lẫn máy khách đều tạo mật khẩu một lần.

+ **SSO** (Single sign-on).

🡪 đăng nhập 1 lần có thể vào đc nhiều trang khác nhau (ví dụ như đăng nhập vào trang **james.codegym.vn** thì có thể vào đc trang **ken.codegym.vn** và **bob.codegym.vn**,…).

+ Ngoài ra, còn nhiều cơ chế khác như **Sich trắc học** (vân tay, nhận diện khuôn mặt,…),…

***60,CSRF là gì ?***

- **CSRF** – hay còn gọi là kỹ thuật tấn công “**Cross-site Request Forgery**“, nghĩa là kỹ thuật tấn công giả mạo chính chủ thể của nó.

**CSRF** nói đến việc tấn công vào chứng thực **request** trên **web** thông qua việc sử dụng **Cookies**. Đây là nơi mà các **hacker** có khả năng sử dụng thủ thuật để tạo **request** mà bạn không hề biết. Vì vậy, một **CSRF** là **hacker** lạm dụng sự tin tưởng của một ứng dụng **web** trên trình duyệt của nạn nhân.

***61,CORS là gì ?***

- **CORS** (**Cross-origin resource sharing**) – là một cơ chế cho phép nhiều tài nguyên khác nhau (**fonts**, **Javascript**, v.v…) của một trang **web** có thể được truy vấn từ **domain** khác với **domain** của trang đó.

**CORS** được sinh ra là vì **same-origin policy**, là một chính sách liên quan đến bảo mật được cài đặt vào toàn bộ các trình duyệt hiện nay. Chính sách này ngăn chặn việc truy cập tài nguyên của các **domain** khác một cách vô tội vạ.

***62,Spring Boot là gì?***

- **Spring boot** – là 1 trong số các **module** của **Spring Framework**, đc sử dụng để tạo ra các ứng dụng độc lập có thể chạy ngay với rất ít cấu hình thêm.

***63,Sự khác nhau giữa SpringBoot và SpringMVC là gì?***

- **So sánh Spring MVC và Spring Boot:**

|  |  |
| --- | --- |
| **Spring Boot** | **Spring MVC** |
| - Là một **module** của **Spring** để đóng gói ứng dụng dựa trên **Spring** với các giá trị mặc định hợp lý.  - Cung cấp các cấu hình mặc định để xây dựng khung công tác được cung cấp bởi **Spring**.  - Không cần xây dựng **cấu hình** theo cách thủ công.  - Tránh mã viết sẵn và kết hợp các phần phụ thuộc lại với nhau trong một đơn vị duy nhất.  - Làm giảm thời gian phát triển và tăng năng suất. | - **Spring MVC** là **model-view-controller** web framework trong khuôn khổ **Spring framework**.  - Cung cấp các tính năng sẵn sàng sử dụng để xây dựng một ứng dụng **web**.  - Yêu cầu **cấu hình** xây dựng theo cách thủ công.  - Nó chỉ định từng phụ thuộc riêng biệt.  - Cần nhiều thời gian hơn để phát triển. |

***64, Cấu hình cho Spring Boot Tìm kiếm các Bean ở nhiều package khác nhau bằng cách nào?***

***65, Trình bày Spring Security? @EnableWebSecurity làm gì?***

***66, Trình bày Spring Boot JPA? Cài đặt?***

- **Spring Boot JPA** - là 1 phần trong hệ sinh thái **Spring Data**, nó tạo ra một **layer** ở giữa tầng **service** và **database**, giúp chúng ta thao tác với **database** một cách dễ dàng hơn, tự động **config** và giảm thiểu **code** thừa thãi.