**BÀI 1. SPRING MVC**

|  |  |
| --- | --- |
| **- Ý nghĩa của Framework** | - **Framework:**là các đoạn code đã được viết sẵn, cấu thành nên một bộ khung và các thư viện lập trình được đóng gói. Chúng cung cấp các tính năng có sẵn như mô hình, API và các yếu tố khác để tối giản cho việc phát triển các ứng dụng web phong phú, năng động. Các **framework** giống như là chúng ta có khung nhà được làm sẵn nền móng cơ bản, bạn chỉ cần vào xây dựng và nội thất theo ý mình.  - **Spring** là 1 framework  - **Ưu điểm:**   * Giúp LTV tiết kiệm được tối đa thời gian, công sức khi phát triển phần mềm/ứng dụng. * Cho phép sản phẩm ứng dụng kế thừa các tính năng, cấu trúc đã được tiêu chuẩn hóa => giúp cho quá trình vận hành và bảo trì/khắc phục sự cố ứng dụng dễ dàng hơn. * Cho phép người dùng mở rộng tùy ý dựa trên những gì mà **Framework** đã cung cấp.   - Một số Framework phổ biến: Spring MVC, JSF, Struts, Vaadin… |
| **- Phân biệt Framework & Library** | **- Giống:** cả Framework và Library đều cung cấp các chức năng được viết sẵn để có thể tái sử dụng.  **- Khác***: Điểm khác biệt lớn nhất giữa Framework và Library là cơ chế IoC (Inversion Of Control - đảo ngược quyền điểu khiển)*   * Libs chỉ cung cấp công cụ tiện ích để thoải mái sử dụng. Là ứng dụng nắm quyền điều khiển (control) * Framework tạo ra một bộ khung và cung cấp các hướng dẫn cần tuân thủ theo. Là framework nắm quyền điều khiển. |
| **- Spring MVC** | - Spring MVC Framework là một nền tảng mã nguồn mở phổ biến để phát triển ứng dụng Java EE (ứng dụng web)  - Cung cấp một Front Controller để xử lý hoặc lắng nghe mỗi khi có request tới ứng dụng; điều hướng đến các controller con tương ứng.  - Front Controller chính là Dispatcher Servlet  - DispacherServlet gửi các request tới các Controller và quyết định hồi đáp bằng cách gửi lại view    - POJO (plain old Java Object): đối tượng Java khai báo theo cách cũ (Lớp: bao gồm các thuộc tính…)  - Service ~ BO // Entities ~ Model  **- Luồng đi của Spring MVC:**   1. Người dùng điền vào tên website mình cần truy cập trên browser sau đó bấm enter. Lúc này mình gửi 1 request lên server nơi mà mình triển khai ứng dụng Spring. 2. Thành phần DispatcherServlet của Spring MVC sẽ nhận được request (yêu cầu) của người dùng ở bước 1. Dispatcher là thành phần quan trọng nhất trong springmvc. Nó sẽ là nơi đầu tiên nhận request từ client sau đó sẽ chuyển request đó tới các controller tương ứng, đồng thời sẽ là chốt chặn cuối cùng trả về kết quả cho client. 3. Sau khi nhận được request(yêu cầu) DispatcherServlet sẽ chuyển yêu cầu đó tới Controller bằng các cơ chế mapping mà ta khai báo trong Handler Mapping. |
| **- Cấu hình DispatcherServlet** | - Cấu hình của DispatcherServlet được thể hiện trong file web.xml  **<servlet>**  **<servlet-name>dispatcher</servlet-name>**  **<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>**  **<load-on-startup>1</load-on-startup>**  **</servlet>**  **<servlet-mapping>**  **<servlet-name>dispatcher</servlet-name>**  **<url-pattern>/</url-pattern>**  **</servlet-mapping>** |
| **- DI và IoC** | **- DI** *(Dependency Injection – tiêm phụ thuộc)* :   * Là một cơ chế để triển khai IoC. * Là kỹ thuật giảm sự phụ thuộc của 2 module với nhau bằng cách gọi qua Interface * Triển khai DI bằng cách tiêm đối tượng của module A vào module B.   Ví dụ: Controller và ServiveImpl giao tiếp qua Interface Service.  - Có 3 cách triển khai DI:   * Sử dụng Interface * Sử dụng Contructor * Sử dụng Setter   **- IoC** *(Inversion Of Control - đảo ngược quyền điểu khiển):*   * Là một nguyên lý thiết kế trong công nghệ phần mềm. * Cho phép framework nắm giữ quyền điều khiển luồng thực thi của hệ thống và gọi các mã nguồn khác   Lợi ích:   * Tách rời việc thực thi (execution) và việc triển khai (implementation) * Dễ chuyển đổi giữa các implementation * Dễ phân tách module hơn * Dễ kiểm thử hơn, bằng cách tách rời các thành phần riêng lẻ |
| **- Model / ModelMap / ModelAndView** | |  |  |  | | --- | --- | --- | | **Model** | **ModelMap** | **ModelAndView** | | - Là 1 interface | - Là 1 class kế thừa từ Map | - Là sự kết hợp giữa ModelMap và View | | - Dùng để gởi giá trị sang view.  - Sử dụng như một tham số trong method của Controller. | - Sử dụng như một tham số trong method của Controller.  - Có các phương thức của Map (put, putAll…) | - Khi khai báo sẽ kèm theo cả việc khai báo View  - Nó cho phép controller trả về 1 giá trị bao gồm Model và View . | |
| **- IoC Container** | - Được đại diện bởi interface ApplicationContext  - Là lõi của Spring Framework  - IoC Container sẽ tạo ra các đối tượng, nối chúng lại với nhau và quản lý vòng đời của chúng từ khi tạo ra đến khi bị hủy.  - IoC Container sử dụng DI *(*[*Dependency Injection*](https://viettuts.vn/spring/dependency-injection-trong-spring)*)* để quản lý các thành phần tạo nên một ứng dụng. Những đối tượng này được gọi là Spring Bean.  - IoC Container được cung cấp thông tin từ các tập tin XML.  Có hai loại IoC Container, đó là:   1. BeanFactory 2. ApplicationContext |
| **- ViewResolver** | - Là 1 interface (tạo tiền tố WEB-INF/views và hậu tố “.jsp”)  - Là cơ chế để xử lý tầng view của Spring MVC  - Ánh xạ tên của view sang đối tượng view tương ứng  - Là bộ phận quyết định xem Front Controller sẽ trả về View thực tế nào. |
| **- So sánh Spring MVC và MVC** | - Khác nhau ở Front Controller (DispatcherServlet): dùng để điểu hướng Controller thích hợp và View thích hợp |
| **- Phân biệt URI và URL** | |  |  | | --- | --- | | **URL** | **URI** | | - URL [*(Uniform Resource Locator – định vị tài nguyên đồng nhất)*](https://quantrimang.com/url-la-gi-158090)*:* thường được định nghĩa là một chuỗi ký tự dẫn đến một địa chỉ. | - URI *(Uniform Resource Identifier – bộ định dạng tài nguyên đồng nhất):* bao gồm URL và URN *(Uniform Resource Name – tên tài nguyên đồng nhất)* | | - URL được sử dụng để mô tả đặc điểm của một mục. | - URI cung cấp một kỹ thuật để xác định danh tính của một mục. | | - URL liên kết một trang web, một thành phần của trang web hoặc một chương trình trên một trang web với sự trợ giúp của việc truy cập các phương pháp như giao thức. | - URI được sử dụng để phân biệt tài nguyên này với tài nguyên khác bất kể phương pháp được sử dụng. | | - URL cung cấp thông tin chi tiết về loại giao thức sẽ được sử dụng. | - URI không chứa đặc tả giao thức. | | - URL là một loại URI. | - URI là "tập cha" của URL. | |

**BÀI 2. SPRING CONTROLLER**

|  |  |
| --- | --- |
| **@GetMapping/**  **@PostMapping/**  **@RequestMapping** | - Đều sử dụng để ánh xạ các request tới các action tương ứng của controller  - @GetMapping ~ phương thức Get  - @PostMapping ~ phương thức Post  - @RequestMapping : gồm các phương thức   * @GetMapping * @PostMapping * @PutMapping (@RestController) * @DeleteMapping (@RestController) * @PatchMapping (@RestController) |
| **@RequestParam &**  **@PathVariable** | **- Giống:**   * Đều lấy các tham số từ người dùng truyền lên (đều sử dụng để gởi giá trị từ client đến server) * Đều là 1 phần của URL   - **Khác:**   |  |  | | --- | --- | | **RequestParam** | **PathVariable** | | - Truyền giá trị theo dạng key – value và được phân tách bằng dấu & | - Truyền giá trị theo value (không cần key) | | - Sử dụng annotation:  **@ RequestParam** | - Sử dụng annotation:  **@ PathVariable** | | - Thường được use submit form với phương thức là GET, khi đó trên thanh URL sẽ chứa các giá trị các ô input được submit. (URL như bình thường).  Vd: local8080/student/detail&id=123 | - Thường được sử dụng để xử ký những URI động, có 1 hoặc nhiều parameter bên trong URI (khai báo URL theo dạng template)  Vd: local8080/student/detail/123  (123,111…là URI động) | | - Dùng khi muốn truy vấn DL trên thanh URL | - Dùng khi muốn che giấu thông tin cần thiết | |

**BÀI 3. DATA BINDING**

|  |  |
| --- | --- |
| **Data Binding** | - Cơ chế liên kết DL đầu vào và đầu ra với các đối tượng Model  Vd: Liên kết giữa các thẻ input có path = “…” với các thuộc tính bên trong Model. \_ Đầu vào: @ModelAtribute \_ Đầu ra: Dùng Model  - DataBinding được sử dụng để xác thực trường cho mỗi yêu cầu ứng dụng (validator), định dạng trường (formatter) và cách sử dụng Dịch vụ chuyển đổi để chuyển đổi các giá trị thuộc tính (conversion service, property editor)  - DataBinding trong Spring hoạt động dựa trên Data Binder. |
| **Redirect & FlashAttribute** | - FlashAttribute: được sử dụng ở interface RedirectAttribute  (như là tham số ở phương thức trong Controller)  - Tác dụng: truyền DL sau khi redirect (chuyển hướng)  - Redirect: chuyển hướng trang (ko kèm DL) |
| **@ModelAttribute** | **-** Sử dụng @ModelAttribute để truy cập đến thuộc tính của một model (hoặc khởi tạo model nếu chưa có)  - Các thuộc tính của model được liên kết với các trường dữ liệu có cùng tên |

**BÀI 4. Thymeleaf**

|  |  |
| --- | --- |
| **Thymeleaf** | - Là bộ cộng cụ dùng để xử lý view, dựa trên ngôn ngữ Java (Template engine Java – công cụ mẫu Java)  - Thymeleaf có thể xử lý HTML, XML, JavaScript, CSS và cả text.  - Thymeleaf được xây dựng phù hợp với các tiêu chuẩn của web, đặc biệt là HTML5.  - Những loại template được Thymeleaf hỗ trợ: HTML, XML, TEXT, JAVASCRIPT , CSS, RAW  **- Lợi ích:**   * Với thymeleaf, ta chỉ cần sử dụng file HTML là có thể hiển thị tất cả mọi thứ (không cần jsp …). * Thymealeaf sẽ tham gia vào rendered (kết xuất) các file HTML dưới dạng các thuộc tính trong các thẻ HTML –> do đó ta không cần phải thêm bất kỳ thẻ non-HTML nào. * Vì là HTML nên ta có thể xem các file view mà không cần khởi chạy server. * Thymeleaf hỗ trợ cơ chế cache, do đó ta có thể cache dữ liệu hoặc custom để hiển thị view khi có thay đổi mà không cần restart server. |
| **Cấu hình Thymeleaf** | **-** Thymeleaf có thể được cấu hình bằng file XML hoặc qua các lớp config.    Thông qua Thymeleaf ViewResolver.   * Prefix : đường dẫn của file html * Suffix : tập tin mở rộng (.html) |
| **Các biểu thức cơ bản** | - Biểu thức với biến: ${...}  - Biểu thức với thuộc tính: \*{...}  - Biểu thức với Message ( validation, i18n): #{...}  - Biểu thức với URL: @{...}  - Biểu thức với phân đoạn: ~{...}  **!!! Lưu ý: Với th:href trong:**   * **RequestParam:**   @{/student/detail(id=${student.id},name=${student.name})}   * **PathVariable:**   @{/student/detail**/{id}**(id=${student.id})} |
| **Các thuộc tính hay sử dụng** | **- Biểu thức điều kiện:**  **th:if** trả về true trong các trường hợp:   * Một giá trị boolean là true. * Một số có giá trị là non-zero (khác 0) * Một ký tự có giá trị là non-zero (khác 0) * Một chuỗi có giá trị khác với “false”, “off” hoặc“no” * Nếu một giá trị không phải là boolean, số, ký tự hoặc chuỗi.   **th:switch…th:case**   * th:case = “\*” ~ default   **- Vòng lặp: th:each**  Vòng lặp th:each cung cấp các trạng thái sau:   * index: chỉ số của lần lặp hiện tại, bắt đầu từ giá trị 0 * count: chỉ số của lần lặp hiện tại, bắt đầu từ giá trị 1 * size: tổng số các phần tử * current: phần tử của vòng lặp hiện tại * even hoặc odd: vòng lặp chẵn hay lẻ * first: vòng lặp đầu tiên * last: vòng lặp cuối cùng   **- Output:**   * th:text ~ c:out : bảo mật hơn * th:utext ~ ${} |
| **Header/Footer** | - **Fragment** là các đoạn mã riêng biệt, có thể được nhúng vào các vị trí khác nhau trong template  - Có thể sử dụng th:insert hoặc th: replace để nhúng fragment   * th:replace sẽ thay thế thẻ mục tiêu bởi Fragment. * th:insert sẽ trèn Fragment vào thành con của thẻ mục tiêu (Target tag).   - Sử dụng fragment để định nghĩa header/ footer  Vd: <div th:insert = “~{common::header}”>…</div>  **- th:block**   * Không phải là một thuộc tính, mà là vùng chứa thuộc tính * Là bộ đệm của Thymeleaf |
| **So sánh JSP & Thymeleaf** | |  |  | | --- | --- | | **Thymeleaf** | **JSP** | | - Thymeleaf là thuần HTML, để hiển thị dữ liệu từ server nó sẽ dùng các thẻ html do thymeleaf cung cấp. | - File JSP có thể nhúng code java (cái này hạn chế, tốt nhất là không làm), hiển thị dữ liệu bằng JSTL. | |

**BÀI 5. Thao Tác Với CSDL và ORM**

|  |  |
| --- | --- |
| **ORM**  **(Kết nối quan hệ giữa các đối tượng)** | - ORM (Object-Relational Mapping) là **kỹ thuật** liên kết giữa các đối tượng trong lập trình với các đối tượng trong CSDL  - ORM cho phép truy xuất dễ dàng đến dữ liệu thông qua các đối tượng lập trình - ORM giúp lập trình viên tập trung thao tác với các đối tượng, không cần quá quan tâm đến CSDL thực tế đang dùng  **-** Là một quá trình ánh xạ (chuyển đổi) dữ liệu từ ngôn ngữ hướng đối tượng sang Database quan hệ và ngược lại. ORM giúp mình ánh xạ các tables,column,kiểu dữ liệu và mối quan hệ (1-1,1-n,n-n) trong database thành các Class và thuộc tính trong Java.    **Ưu điểm:**   * Ngắn gọn, xúc tích về CRUD * Quản lý về Java là chính, không quan tâm đến Database * OOP: ORM giúp lập trình viên tập trung vào lập trình hướng đối tượng * Tính độc lập: Làm việc được với nhiều loại database (hệ quản trị cơ sở dữ liệu), nhiều kiểu dữ liệu khác nhau. Dễ dàng thay đổi loại database hơn. Các câu lệnh SQL không phụ thuộc vào loại database. * Đơn giản, dễ sử dụng: Hỗ trợ HSQL, cung cấp nhiều nhiều API truy vấn. * Năng suất hơn: viết code ít hơn, dễ hiểu hơn. Phù hợp các case CRUD (Create, Read, Update, Delete) * Khả năng sử dụng lại code.     **Nhược điểm:**   * Phải học Hibernate Query Language * Khả năng truy vấn bị hạn chế, những câu truy vấn phức tạp thì phải sử dụng SQL thuần * Khó tối ưu câu lệnh SQL (do câu lệnh SQL được ORM tự động sinh ra). |
| **Hibernate và cấu hình** | **-** Hibernate là framework triển khai kỹ thuật ORM   1. Cấu hình file hibernate.xml 2. Tạo Entity Manager (Base Repository) 3. Định nghĩa Entity bởi annotation Entity -> POJO  * Thuộc tính : private, protected * Constructor mặc định * Getter / Setter |
| **Ưu, nhược điểm của Hibernate** | **Ưu điểm:**   * *Dễ sử dụng:* dễ dàng quản lí các kết nối database và dễ fix bug, cung cấp sẵn nhiều API truy vấn * *Tính độc lập:* không cần quan tâm tới cơ sở dữ liệu sử dụng khi viết câu lệnh SQL. * *Tính hướng đối tượng:* tập trung xử lý theo hướng đối tượng, phù hợp sử dụng trong các case CRUD(Create, Read, Update, Delete) * *Tính tin cậy:* Hibernate đã được kiểm thử và khá an toàn trong các truy vấn   **Nhược điểm:**   * Không hỗ trợ các câu truy vấn phức tạp * Một số trường hợp vẫn phải dùng SQL thuần do Hibernate không thể cover hết tất cả các cú pháp của các hệ quản trị cơ sử dữ liệu. * Bị hạn chế sự can thiệp vào câu lệnh SQL do nó được tự động sinh ra. |
| **CRUD** | - C: persist (object)  - R: typed query -> select… -> getResultList () hoặc getSingleResult ()  - U: merge (object) (object phải chứa id phải tồn tại trong bảng)  - D: remove (object)  \*\*\* Save, Update, Remove cần có thêm Transaction |

**BÀI 6. JPA**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| JPA(****Java Persistence API)**** | |  |  |  | | --- | --- | --- | | **ORM** | **JPA** | **Hibernate** | | - Là kỹ thuật liên kết giữa Object trong lập trình và các object ở DB | - Là đặc tả (giao diện ~ interface) nó sẽ triển khai khái niệm ORM, tức là các table (entity), column, id, CRUD | - Là framework triển khai JPA thành các hành động cụ thể, tức là @Entity, @Column, @ Id, persist (), merge ()…  - Xây dựng dựa trên những khái niệm của JPA |     - Tầng Persistent có nhiệm vụ thao tác với database như query lấy dữ liệu , lưu dữ liệu xuống database . JPA cung cấp cho mình cơ chế ORM mapping các bảng, column , mối quan hệ trong database thành các lớp java và đồng thời cung cấp cho mình các method cần thiết để thao tác dữ liệu trong database. |
| **Phân biệt JPA & Hibernate** | |  |  | | --- | --- | | **JPA** | **Hibernate** | | - Là tập các giao diện (interfaces), qui chuẩn, định nghĩa | Thực hiện cài đặt (implements) các giao diện đó, định nghĩa đó. | | - Tương tự interface, triển khai khái niệm ORM (các table, column, id, CRUD) | - Là 1 framework triển khai JPA thành các hành động cụ thể (@Entity, @Column, @Id, persist(), merge()…) | |
| **Tại sao cần JPA?** | - Chúng ta chỉ tập trung vào viết chức năng của chương trình còn các việc như quản lý connection , cách query thì JPA sẽ lo  - Khả năng thay đổi database không bị ảnh hưởng . Ví du hôm nay ta dùng Mysql ngày mai ta dùng Postgres thì không ảnh hưởng tới chương trình của mình. |
| **Các interface trong Spring JPA** | - Repository <T,ID>  - Crud Repository <T,ID>  - PagingAndSortingRepository <T,ID>  - JPARepository <T,ID> *(cũng có thể phân trang và sort đc do JPARepository <T,ID> kế thừa PagingAndSortingRepository <T,ID>)*  ***\*\*\* Tất cả Extends từ* Serializable interface** |
| **CRUD** | - C: save ()  - R: findById () , findAll()  - U: save () -> giữ lại ID đề tồn tại trong database  - D: deleteById() , deleteAll () |

**BÀI 7. Spring Data Repository**

|  |  |
| --- | --- |
| **Native query**  **(truy vấn gốc)** | **-** Để sử dụng native query ta sử dụng thuộc tính nativeQuery = true bên trong annotation @Query -> sử dụng SQL thuần  **[Đọc thêm]**  - Để sử dụng annotation @Query, ta sẽ khai báo nó trước các method của interface extends từ JPARepository, CrudRepository , và truyền câu HQL vào bên trong  Nhược điểm:   * Việc sử dụng các [method có sẵn](https://stackjava.com/spring/code-vi-du-spring-boot-data-jpa-jparepository.html) khi extends interface JPARepository, CrudRepository  không đáp ứng được yêu cầu. * Việc đặt tên method theo chuẩn [Query Creation](https://stackjava.com/spring/code-vi-du-spring-data-query-creation-dat-ten-method.html) quá dài hoặc tối nghĩa. (Ví dụ bạn muốn truy vấn theo 5 điều kiện thì tên method của bạn sẽ gồm 5 điều kiện đó => quá dài) * Kết quả trả về của method sử dụng @Query có thể là: 1 list, 1 stream, 1 page, 1 đối tượng |
| **Phân trang**  **(paging)** | - Page <Student> findAll (Pageable pageable)  **Lưu ý:**   * Sử dụng pageable là tham số ở Controller ( với @PageableDefault để set size cho từng trang)   **- Phân biệt phân trang Back-end và phân trang Front-end:**   |  |  | | --- | --- | | **Back-end** | **Front-end** | | - Viết ở Controller | - Viết ở view, tầng giao diện | | - Dùng pageable | - Dùng datatable | | - Nhẹ hơn, lấy đúng số lượng ở 1 trang để hiển thị (ví dụ: 1 trang có 10 record thì hiển thị 10 record) | - Nặng hơn, lấy toàn bộ DL lên rồi mới phân trang | | - Đồng bộ DL | - Không đồng bộ DL | |
| **Tìm kiếm** | - Sử dụng Optional <String> để check việc kiểm tra keywork có tồn tại trên URL hay không   * .isPresent () : check có tồn tại trên URL * .get () : lấy giá trị gốc từ Optional |
| **Mối quan hệ**  **1-1 / 1-n / n-n** | - @OneToOne  - @OneToMany / @ManyToOne  - @ManyToMany |

**BÀI 8. Validation**

|  |  |
| --- | --- |
| **Validation** | - Kiểm tra tính hợp lệ DL đầu vào của người dùng |
| **@NotEmpty / @NotNull/**  **@NotBlank** | |  |  |  | | --- | --- | --- | | **Empty** | **Null** | **Blank** | | - Là rỗng “” (Không nhập gì) | - Có địa chỉ ở Stack mà không có ở Heap | - Có thể hiểu là null hoặc Empty | | - Có thể dùng khoảng trắng “ ” | - Bỏ qua không nhập gì cả, thì giá trị trả về là null (nếu là integer) và “” (nếu là String) | - Không được nhập khoảng trắng, không được để trống | |
| **Annotation Validation** | 1. Gắn thêm annotation cho các thuộc tính ở Entity 2. Sử dụng BindingResult ở Controller *(BindingResult phải đặt cạnh object muốn so sánh) – xác nhận các ràng buộc xem đối tượng có bị xâm phạm hay không* 3. Hiển thị lỗi ở view (dùng th:errors)   @Email, @Pattern, @Size, @Min, @Max… |
| **Custom Validation** | 1. Implement Validatior cho model muốn Validate 2. Cấu hình file message  * @Configuration , @Bean * Tạo ra message = resource bundle  1. Sử dụng hàm validate của Model ở Controller 2. Hiển thị lỗi ở view (dùng th:errors) |

**BÀI 9. AOP & Exception Handling**

|  |  |
| --- | --- |
| **AOP**  **(Aspect Oriented Programming)** | **-** Aspect Oriented Programming (AOP) – lập trình hướng khía cạnh: là một kỹ thuật lập trình (kiểu như lập trình hướng đối tượng) nhằm phân tách chương trình thành cách moudule riêng rẽ, phân biệt, không phụ thuộc nhau (tức là phân chia project thành các module độc lập)  - AOP không phải dùng để thay thế OOP mà để bổ sung cho OOP.    **Ưu điểm:**   * Khi hoạt động, chương trình sẽ kết hợp các module lại để thực hiện các chức năng nhưng khi sửa đổi 1 chức năng thì chỉ cần sửa 1 module. * Tái sử dụng dễ dàng.   **Nhược điểm:**   * Khái nhiệm khá trừu tượng, độ trừu tượng của chương trình cao * Luồng chương trình phức tạp. |
| **Aspect ->**  **Advice ->**  **Pointcut/**  **JoinPoint** | - Aspect: khối module độc lập với project (nằm trong khái niệm AOP)  - Advice: là hành động để xử lý exception *(try/catch từng phương thức trong Controller) (thể hiện qua @AfterThrowing sau đó ném để xử lý exception)*  - Pointcut: là tập hợp các JoinPoint *(lấy tất cả thông tin trong StudentService)*  - JoinPoint: là các điểm có khả năng xảy ra Exception (với JoinPoint thì sẽ lấy được tất cả thông tin của phương thức)  Vd: Tên của class, Tên phương thức, Kiểu trả về… |
| **@Exception Handle** | - Bỏ đi các try/ catch lặp lại nhiều lần ở các phương thức  **Ưu điểm:**   * Giúp code ở project trở nên độc lập và rõ ràng   **Nhược điểm:**   * Làm source code trừu tượng, không biết Advice đang handle cho phương thức nào. |

**BÀI 10. Session & Cookie**

|  |  |
| --- | --- |
| **Session** | **-** [**Session**](https://topdev.vn/blog/?s=Session) là một phiên làm việc được dùng trong lập trình web có kết nối với database.  Vd: chức năng như đăng nhập, đăng xuất người dùng  - Một [session](https://topdev.vn/blog/?s=Session) bắt đầu khi client gửi request đến server, nó tồn tại xuyên suốt từ trang này đến trang khác trong ứng dụng web và chỉ kết thúc khi hết thời gian timeout hoặc khi bạn đóng ứng dụng. Giá trị của session sẽ được lưu trong một file trên server.  - **Session**: kiểu DL là gì cũng được |
| **Cookie** | **- Cookie** dùng để lưu những thông tin tạm thời.  - **Cookie** thường được tạo ra khi người dùng truy cập một website, **cookie**sẽ ghi nhớ những thông tin như tên đăng nhập, mật khẩu, các tuỳ chọn do người dùng lựa chọn đi kèm. Các thông tin này được lưu trong máy tính để nhận biết người dùng khi truy cập vào một trang web. Mỗi cookie thường có khoảng thời gian timeout nhất định do lập trình viên xác định trước.  - **Cookie** : kiểu DL chỉ là String |
| **Ý nghĩa của Session & Cookie** | **-** Lưu DL trong 1 khoảng thời gian nào đó. |
| **Phân biệt Session & Cookie** | **- Giống:** Lưu trữ DL  **- Khác:**   |  |  |  | | --- | --- | --- | |  | **Session** | **Cookie** | | **Nơi lưu trữ** | **- Session** không được lưu trữ trên trình duyệt.  - Dữ liệu **session** được lưu trữ ở phía server. | **- Cookie** được lưu trữ trên trình duyệt của người dùng.  - Dữ liệu **cookie** được lưu trữ ở phía client. | | **Bảo mật** | **-** DL **session** không  dễ dàng sửa đổi vì chúng được lưu trữ ở phía máy chủ. 🡪 Bảo mật hơn | Dữ liệu **cookie** dễ dàng sửa đổi hoặc đánh cắp khi chúng được lưu trữ ở phía client. | | **Vòng đời** | **-** Sau khi đóng trình duyệt sẽ hết phiên làm việc (session) | **-** Dữ liệu **cookie** có sẵn trong trình duyệt đến khi hết hạn. *(Khi nào người dùng xóa hoặc kết thúc vòng đời setMaxAge() ).* | | **Ví dụ** | - Login (lưu trữ username lâu dài) | - Remember 🡪 thông tin người dùng được lưu lại | |
| **Cách sử dụng** | **- Session:**  - @SessionAttributes thì sử dụng ở đầu Controller nào muốn dùng session.  - @ SessionAttribute thì sử dụng ở tham số ở các method muốn dùng session  **- Cookie:**  - Tạo ra: Sử dụng HttpServletResopne để add cookie  - Sử dụng: Dùng @CookieValue |

**BÀI 11. Webservice – RESTful**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RESTful & SOAP** | - Webservice: là dịch vụ đùng để giao tiếp giữa 2 ứng dụng hoặc 2 hệ thống với nhau thông qua các tiêu chuẩn và giao thức  - SOAP (Simple Object Access Protocol - Giao thức truy cập đối tượng đơn giản) : sẽ theo tiêu chuẩn riêng  - RESTful (Representational State Transfer - Chuyển trạng thái biểu hiện): việc giao tiếp giữa Client và Server sẽ thông qua JSON.   |  |  | | --- | --- | | **SOAP** | **RESTful** | | - SOAP là một giao thức | - REST là một cách thiết kế kiến trúc | | - SOAP không thể sử dụng REST vì nó là 1 giao thức | - REST có thể dùng các web services sử dụng SOAP vì nó có thể dùng bất kỳ giao thức nào như HTTP, SOAP | | - SOAP cung cấp các giao diện dịch vụ (services interfaces) cho các thành phần bên ngoài sử dụng | - REST sử dụng đỉa chỉ URI để cung cấp các dịch vụ | | - SOAP định nghĩa các chuẩn và quy tắc chặt chẽ | - REST không định nghĩa nhiều chuẩn như SOAP | | - SOAP sử dụng băng thông và tài nguyên nhiều hơn REST | - REST sử dụng băng thông và tài nguyên ít hơn SOAP | | - SOAP định nghĩa chuẩn bảo mật của riêng nó | - RESTful kế thừa chuẩn bảo mật tầng vận tải của giao thức mạng | | - SOAP chỉ hỗ trợ định dạng dữ liệu XML | - REST hỗ trợ các định dạng dữ liệu khác nhau như text, HTML, XML, JSON | | - SOAP ít được dùng hơn REST | - REST được ưa chuộng hơn SOAP | | - Được thiết kế để dùng trong tính toán phân tán | - Thường không được dùng trong môi trường tính toán phân tán | | - 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 | | - 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 | | - SOAP hỗ trợ cả hai giao thức SMTP và HTTP | - REST gắn với giao thức HTTP | |
| **Webservice & Website** | **-** Web service cho phép client và server tương tác được với nhau ngay cả trong những môi trường khác nhau.  **-** Webservice: output sẽ là JSON  **-** Web service gồm có 3 chuẩn chính:   * SOAP (Simple Object Access Protocol), * WSDL (Web Service Description Language) * UDDI (Universal Description, Discovery, and Integration).   **Ưu, nhược điểm của Webservice.**   |  |  | | --- | --- | | **Ưu điểm** | **Nhược điểm** | | **-** Hoạt động trên các ứng dụng, nền tảng, hệ điều hành, ngôn ngữ khác nhau. | * - Khi một web service chết hoặc dừng hoạt động sẽ gây lỗi, thiệt hại lớn trên tất cả các hệ thống, thiết bị đang sử dụng web service đó. | | * **-** Khả năng tái sử dụng cao. | - Cần quan tâm đến vấn đề an toàn và bảo mật nhiều hơn khi sử dụng web service. | | * **-** Tạo mối quan hệ tương tác, mềm dẻo trong hệ thống phần mềm, dễ dàng cho việc phát triển ứng dụng phân tán. | - Việc có quá nhiều chuẩn cho web service dẫn đến người sử dụng khó nắm bắt. | | * **-** Giảm sự phức tạp của hệ thống, giảm thời gian phát triển hệ thống, hạ giá thành hoạt động, dễ dàng tương tác giữa các hệ thống với nhau. |  |   **Phân biệt Webservice và Website.**   |  |  | | --- | --- | | **Website** | **Webservice** | | - Output sẽ là view (tạo từ Front End: HTML, JS, CSS…) | - Output sẽ là JSON | | - Có giao diện người dùng | - Không có giao diện người dùng | | - Nói đến Website được hiểu sử dụng bởi con người | - Nói đến Web Service được hiểu sử dụng bởi các ứng dụng được tương tác với nhau qua internet | | - Website hoạt động đa nền tảng, vì chúng yêu cầu tinh chỉnh để hoạt động trên các trình duyệt hay hệ điều hành khác nhau. | - Web Service độc lập về nền tảng và tất cả dạng truyền thông đều sử dụng giao thức chuẩn. | | - Website được truy cập bởi các thành phần trong giao diện người dùng như button, textbox, form ... | - Webservice được truy cập bởi phương thức HTTP – PUT, GET, POST, DELETE ... | | - Website là một ứng dụng đầu cuối. Người sử dụng truy cập website qua URL qua đó nhận được những dữ liệu text, hình ảnh, âm thanh… có thể dễ dàng hiểu được. | - Web service là một khái niệm rộng hơn, dữ liệu trả ra từ web service người sử dụng thông thường khó để hiểu. Dữ liệu đó được các ứng dụng (trong đó có web site) sử dụng và chế biến thành dữ liệu có thể đọc cho người sử dụng. | | - Ví dụ. ArtOfTesting.com là trang web có bộ sưu tập các trang web có liên quan chứa hướng dẫn | - Ví dụ: Google Maps API là một dịch vụ web có thể được sử dụng bởi các trang web để hiển thị bản đồ bằng cách chuyển tọa độ tới đó. | |
| **Web service & API** | - Web service**tạo thuận lợi cho sự tương tác giữa hai máy qua mạng.**  - API hoạt động như một giao diện giữa hai ứng dụng khác nhau để chúng có thể giao tiếp với nhau.  - Webservice sẽ chứa nhiều API  **Sự khác nhau giữa Web Service & API**  1. Tất cả Web services là APIs nhưng tất cả các APIs không phải là Web services.  2. Web services không thể thực hiện được tất cả các thao tác mà API sẽ thực hiện.  3. Một Web service sử dụng 3 chuẩn chính: SOAP, REST và XML-RPC trong quá trình giao tiếp, ngược lại API có thể sử dụng bất kỳ chuẩn nào để giao tiếp.  4. Một Web service đòi hỏi luôn luôn phải có mạng để nó hoạt động nhưng API thì không cần.  5. API tạo điều kiện liên kết trực tiếp với một ứng dụng trong khi Web service thì không. |
| **Triển khai Webservice ở Spring** | - Bao gồm các phương thức:   * GET: lấy thông tin * POST: đăng ký mới * PUT: update toàn bộ * PATCH: update 1 số thông tin * DELETE: xóa |

**BÀI 12. Jquery & Ajax**

|  |  |
| --- | --- |
| **Jquery -**  **Cú Pháp** | - jQuery là một thư viện kiểu mới của JavaScript ***(Write less, do more - Viết ít hơn, làm nhiều hơn****.)* 🡪 làm đơn giản hóa việc truyền tải HTML, xử lý sự kiện, tạo hiệu ứng động và tương tác Ajax.  - jQuery là một bộ công cụ tiện ích JavaScript làm đơn giản hóa các tác vụ đa dạng với việc viết ít code hơn.  - jQuery là một công cụ cung cấp một tập hợp các phương thức AJAX để phát triển ứng dụng web.  **-** Cú Pháp:  **$ (selector) . event ();**  Selector : id (#) , class (.) , tag (dùng tên thẻ)  Event: Các event lược bỏ tiền tố “on” như:   * Click : xuất hiện khi click chuột * Change : xuất hiện khi phần tử thay đổi * Keydown : xuất hiện khi được nhấn * Keypress : kích hoạt khi phím được nhấn và thả ra * Load : xuất hiện khi tài liệu được tải * Select : kích hoạt khi 1 text được chọn * Scroll : kích hoạt khi cửa sổ được cuộn   ….. |
| **Ajax –**  **Cú Pháp** | **-** AJAX (*Asynchronous JavaScript and XML)*: là **kỹ thuật** xử lý bất đồng bộ của jQuery  - Mục đích:   * Tải dữ liệu từ Server mà không cần Refresh trình duyệt. * Load DL mà không cần phân trang   **- Ví dụ:** bình luận facebook, người A và người B cùng bình luận, không cần load lại trang vẫn hiển thị bình luận 🡪 tự động load theo thời gian |
| **So sánh mô hình thông thường và mô hình Ajax** | **- Mô hình Ajax:**   1. Trình duyệt tạo một lệnh gọi JavaScript để kích hoạt XMLHttpRequest. 2. Ở dưới nền, trình duyệt tạo một yêu cầu HTTP gửi lên server. 3. Server tiếp nhận, truy xuất và gửi lại dữ liệu cho trình duyệt. 4. Trình duyệt nhận dữ liệu từ server và ngay lập tức hiển thị lên trang. Không cần tải lại toàn bộ trang..   **- Mô hình thông thường**   1. HTTP được gửi từ trình duyệt lên máy chủ. 2. Máy chủ nhận, sau đó phản truy xuất thông tin. 3. Server gửi dữ liệu được yêu cầu lại cho trình duyệt. 4. Trình duyệt nhận dữ liệu và tải lại trang để hiển thị dữ liệu lên. |

**BÀI 13. I18N**

|  |  |
| --- | --- |
| **I18N** | - I18N (Internationalization – quốc tế hóa): khái niệm dùng để tùy chỉnh trang web theo nhiều ngôn ngữ khác nhau.  - Còn được gọi là Translation hay Localization Enablement |
| **Cấu hình và sử dụng** | 1. Cấu hình Interceptor:   - Là thành phần gắn vào đuôi request , dùng để bắt các request param có liên quan đến việc thay đổi ngôn ngữ  (Có thể không sử dụng cấu hình này, thì mình dùng if-else liên tục để thay đổi ngôn ngữ 🡪 phức tạp hơn là dùng cấu hình Interceptor)     1. Gởi cho LocaleResolver giải quyết      1. LocaleResolver sẽ refer đến các file MessageResource tương ứng.       **Lưu ý: Set UTF – 8**   * 1. Set default encoding cho MessageResource   2. Set encoding cho file message |

**BÀI 14. Spring Security**

|  |  |
| --- | --- |
| **Ý nghĩa của việc bảo mật trong Spring** | - Bảo mật các thông tin quan trọng từ khách hàng và tài nguyên của mình.  - Spring Security: cung cấp các dịch vụ bảo mật toàn diện cho các ứng dụng doanh nghiệp có nền tảng Java EE.  - Spring Security là một framework tập trung vào việc cung cấp khả năng xác thực (authentication) và phân quyền (authorization) cho ứng dụng Java.   * Authentication: là tiến trình thiết lập một principal. Principal có thể hiểu là một người, hoặc một thiết bị, hoặc một hệ thống nào đó có thể thực hiện một hành động trong ứng dụng. * Authorization hay Access-control: là tiến trình quyết định xem một principal có được phép thực hiện một hành động trong ứng dụng hay không. Trước khi diễn tiến tới Authorization, principal cần phải được thiết lập bởi Authentication. |
| **Các cách xác thực thông dụng** | **- Authentication (xác thực)**: xác nhận 1 user có được vào trang web của mình hay không ( = cách check xem username, password có trùng khớp trong database hay không)  - **Authorization (Cấp quyền/phân quyền)** là quá trình xác định xem một người dùng có quyền truy cập một tài nguyên cụ thể để thực hiện một số hành động hay không.  - Authorization còn được hiểu là sự phân quyền người dùng để họ có quyền truy cập và sử dụng những tính năng mà ứng dụng cung cấp.  - Phân chia các layout các trang website mà người dùng có thể truy cập vào được |
| **Config Authentication** | 1. Tạo Database: User, Role, UserRole  2. Tạo class config Spring Security:   * Login 🡪 thành công, thất bại * Logout   3. Tạo các Entity : User, Role, UserRole  4. Tạo các Repository để get quyền hạn  5. Tạo Service: UserDetailServiceIml  6. Tạo Controller: tạo 1 Request Mapping để hướng đến trang login  7. Tạo View: Với method checkLogin đã được config ở bước 2. |

**BÀI 15. Spring Security Authorization**

|  |  |
| --- | --- |
| **Cấu hình Authorization** | - Sử dụng http.authorizeRequest ()  **.** antMatcher () : để định nghĩa quyền cho các Request Mapping  **.** access () : cho quyền tự định nghĩa không có tiền tố “ROLE\_” ; ngược lại dùng: **.** hasRole () ; **.** hasAnyRole (). |
| **Remember me** | - Sử dụng http.rememberMe ()  **Lưu ý:** Sử dụng rememberMeCookie để duy trì đăng nhập |
| **CSRF, XSS** | |  |  | | --- | --- | | **CSRF** | **XSS** | | - Cross Site Request Forgery (*giả mạo yêu cầu trang web chéo*) : là kĩ thuật tấn công giả mạo bằng cách sử dụng quyền chứng thực của người sử dụng đối với 1 website khác. (thông qua 1 trang web khác đã bị hack để lấy thông tin request từ người dùng) | - Cross-site scripting *(Viết kịch bản trang web chéo):* nhúng mã độc vào trong 1 web để đánh cắp thông tin từ web | | - Tấn công gián tiếp | - Tấn công trực tiếp | | - Các ứng dụng web hoạt động theo cơ chế nhận các câu lệnh **HTTP**từ người sử dụng, sau đó thực thi các câu lệnh. | - Sử dụng Javascript | | - Mã độc được lưu trữ trong các trang web của bên thứ ba. | - Trang web chấp nhận mã độc hại |   **- Cách thức hoạt động:**   * **XSS:**      * **CSRF:** |

**? Bean là gì trong Spring Framework ?**

Hiểu Bean đơn giản, **Bean là một instance** từ một bản thiết kế class.**Việc tạo ra instance và quản lý** sẽ được**Spring IOC đảm nhiệm.**

* **Spring IOC**sẽ dựa trên các **meate data** chúng ta khai báo bên trong **class để Spring tạo ra Bean** ( Ví dụ chúng ta khai báo các Bean khác mà Bean này phụ thuộc..etc.. )

**? Có bao nhiêu loại bean scope ?**

1. **Singleton :**Với **Scope**này, **spring chỉ tạo ra 1 Instance duy nhất** của bản thiết kế Bean loại này.

* Ví dụ chúng ta có class UserService. Spring sẽ chỉ tạo 1 instance duy nhất tạo toàn bộ ứng dụng của Spring. Sau đó tất cả yêu cầu bean loại này đều sẽ nhận đường cùng một Instance.

1. **Prototype :**Với loại **Scope**này, mỗi khi có một yêu cầu lấy Bean để sử dụng, Spring sẽ tạo ra 1 Bean mới từ bản thiết kế để trả lại cho bên yêu cầu**.** Nếu bạn có 100 yêu cầu thì sẽ có 100 bean thuộc bản thiết kế đó được tạo ra.
2. **Request**( Dành riêng cho web)**:**Với loại **Scope**này**,** Spring sẽ tạo ra 1 Bean mới cho **mỗi một HTTP request**.
3. **Session**( Dành riêng cho web)**: V**ới loại **scope**này,**mỗi một Session** của user trên website của chúng ta sẽ có một Instance mới được tạo**.**
4. **Application**( dành cho web)**: Với Scope này,**mỗi một ServletContext sẽ có một Bean instance được tạo ra.
5. **Websocket**( dành cho web) : Mỗi một Socket kết nối đến server sẽ có một bean instance được tạo ra.

**? Bao nhiêu cách tạo Bean?**

**Cách 1: Thông qua Classpath**

Chúng ta sử dụng các **Annotation**extends từ**@Component** để Spring tự động phát hiện các Bean ở Classpath để tạo và quản lý Bean.

* Spring sẽ tự động tìm kiếm các Class được khai báo Annotation **@Component**

hoặc cácAnnotation extend **@Component để tạo ra Bean.**

* Một số Annotation chúng ta thường sử dụng mà đang extends từ**@Component là @Configurationb, @RestController, @Controller, @Service, @Repository**

## ****Cách 2: Thông qua @Bean****

**Với cách này,** chúng ta sẽ khai báo @Bean ở trên một method, để Response của method đó sau khi được gọi và trả về thì Spring sẽ tạo ra 1 Bean.

**? Ưu và nhược điểm của 2 cách tạo Bean ?**

* **Class Path :**Nếu chúng ta muốn bean tự tạo ra tự inject các phụ thuộc vào Bena của chúng ta thì chúng ta nên sử dụng Class path để Spring tự tạo và Inject Di giúp chúng ta
* **@Bean :** Nếu chúng ta muốn tự tạo ra 1 instance, muốn tự có các logic và tự inject các phụ thuộc vào trong thì chúng ta có thể sử dụng @Bean. Ưu điểm của loại này mà Class path là nhược điểm là @Bean có thể inject thêm các sự phụ thuộc vào Bena nhưng các sự phụ thuộc đó không phải là một Bean.