**1. Spring Annotation**

**Trong bài hôm nay chúng ta sẽ đi qua các annotation thường xuyên được sử dụng trong Spring**

**2 @Congiguration**

**Được sử dụng để chỉ ra rằng class khai báo sử dụng annotation @Configuration sẽ khai báo một hoặc nhiều @Bean method trong class đó. Những class khai báo với @Configuration sẽ được Spring container quản lý và tạo bean trong lúc chương trình đang chạy. Thông thường các bean cấu hình cho dự án ta để trong này. Ví dụ cấu hình thymeleaf, đa ngôn ngữ , và nhiều cấu hình khác cho ứng dụng.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | @Configuration  public class WebConfig implements WebMvcConfigurer {  @Bean  @Description("Thymeleaf template resolver serving HTML 5")  public ClassLoaderTemplateResolver templateResolver() {  var templateResolver = new ClassLoaderTemplateResolver();  templateResolver.setPrefix("mytemplates/");  templateResolver.setCacheable(false);  templateResolver.setSuffix(".html");  templateResolver.setTemplateMode("HTML5");  templateResolver.setCharacterEncoding("UTF-8");  return templateResolver;  } |

**3. @Bean**

**Method (phương thức) sử dụng @Bean ở phía trên mình để chỉ ra rằng . Method đó sẽ sản xuất ra đối tượng bean và được quản lý bởi spring container . Bean annotation có thể sử dụng với các tham số như name, initMethod hoặc destroyMethod**

**Ví dụ dưới đây mình sử dụng @Bean để tạo ra object Spring Template .**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @Bean  @Description("Thymeleaf template engine with Spring integration")  public SpringTemplateEngine templateEngine() {  var templateEngine = new SpringTemplateEngine();  templateEngine.setTemplateResolver(templateResolver());  return templateEngine;  } |

**4. @PreDetroy và @PostConstruct**

**Đây là cách dùng khác để quản lý vòng đời của Bean. Ngoài cách sử dụng initMethod và destroyMethod. Ta có thể sử dụng @PreDetroy và @PostConstruct với cùng một mục đích**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class Computer {  @PostConstruct  public void turnOn(){  System.out.println("Load operating system");  }  @PreDestroy  public void turnOff(){  System.out.println("Close all programs");  }  } |

**5. @ComponentScan**

**Chúng ta sử dụng @ComponentScan để thông báo Spring Container biết phải vào package nào trong dự án để quét các Annotation và tạo Bean. Như ví dụ bên dưới. Spring sẽ quét tất cả các file trong package levunguyen.spring. Tìm các Class có annotation để tạo bean và các @autowire để nhúng bean ở trong container vào các Class sử dụng autowire**

|  |  |
| --- | --- |
| 1  2  3  4  5 | @ComponentScan(basePackages = "levunguyen.spring ")  @Configuration  public class SpringComponentScanApp {  // ...  } |

**6. @Component**

**Khi một class được đánh dấu là @Component thì sẽ được tạo thành 1 bean. Khi Spring start thì nó quét qua các annotation có đánh dấu là @Component thì nó sẽ tạo bean cho class đó. Ví dụ ta có class Contact và ta đánh dấu nó là @Component thì Spring khi đọc qua class này nó sẽ tạo 1 bean có tên là contact trong container của nó. Nếu có class nào dùng thì nó sẽ nhúng bean này vào. Dùng @component là để tạo ra một bean**

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Component  @Scope("request")  public class Contact {  } |

**7. @PropertySource và @Value**

**Trong Spring chúng ta sử dụng @PropertySource để cho Spring biết tìm các file properties cấu hình cho hệ thống ở đâu đồng thời sử dụng @Value để lấy các giá trị trong file properties**

**Ví dụ bên dưới ta sử dụng classpath để khai báo file properties ta đặt ở đâu trong dự án. Tiếp đến ta sử dụng @Value để lấy các giá trị trong file properties với key tương ứng và gán vào biến mà ta sẽ sử dụng.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | @Configuration  @ComponentScan(basePackages = { "levunguyen.\*" })  @PropertySource("classpath:config.properties")  public class AppConfigMongoDB {  //1.2.3.4  @Value("${mongodb.url}")  private String mongodbUrl;  //hello  @Value("${mongodb.db}")  private String defaultDb; |

**Sử dụng để khai báo với Spring đọc các cấu hình trong file resource vào ứng dụng**

**8. @Service**

**Nếu một class được đánh dấu là @Service thì nó là kiểu đặt biệt của @Component. Nó được dùng để xử lý các nghiệp vụ của ứng dụng. Ví dụ như kế toán thì có nghiệp vụ là kiểm tra chi, quản lý thu. Lớp BookServiceImpl dưới đây được đánh dấu là @Service thì nó sẽ phụ trách xử lý các vấn đề liên quan đến nghiệp vụ.**

|  |  |
| --- | --- |
| 1  2  3  4 | @Service  public class BookServiceImpl implements BookService {  } |

**9. @Repository**

**Nếu một class được đánh dấu là @Repository thì nó là kiểu đặt biệt của @Component . Nó được sử dụng để nói bean này dùng để truy cập và thao tác xuống cơ sở dữ liệu. Class BookDaoImpl được đánh dấu với @Repository nghĩa là lớp này có nhiệm vụ thực hiện các câu lệnh truy vấn xuống database.**

|  |  |
| --- | --- |
| 1  2  3  4 | @Repository  public class BookDaoImpl implements BookDao {  } |

**10. @Autowire**

**Tự động nhúng các bean được Spring Container sinh ra vào Class có khai báo @Autowire. Khi Spring nó sẽ tìm kiếm bean có tên là BookDao trong container của nó ,sau đó nhúng (hoặc tiêm) vào lớp BookServiceImple. Đây chính là cơ chế DI (dependency injection). Khi Spring bắt đầu chạy nó sẽ quét qua các lớp có sử dụng annotation để tạo bean đồng thời nó cũng quét bên trong các bean xem có khai báo @Autowire không nếu có nó sẽ tìm kiếm bean tương ứng mà nó quản lý và nhúng vào.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | @Service  public class BookServiceImpl implements BookService {  @Autowired  private BookDao bookDao;  @Autowired  private CustomerDao customerDao;  ...  } |

**11 @Scope**

**Khi bean được tạo ra thì nó có nhiều scope khác nhau. @Scope ở đây là phạm vi bean được sinh và và bị phá huỷ dưới sự quản lý của Spring Container. Khi bean được sinh ra nó có 5 scope (phạm vi được sử dụng)**

* **singleton : đây là scope mặc định của 1 bean khi được sinh ra. Nếu ta không khai báo scope cụ thể thì bean sẽ lấy singleton scope. Singleton bean có nghĩa là bean chỉ tạo ra 1 lần và được sử dụng trong container. Chỉ duy nhất 1 bean tồn tại trong container.**
* **prototype : ngược lại với singleton ta muốn có nhiều bean (đối tượng) thì ta sử dụng scope prototype.**
* **Request : Bean được sinh ra thông qua các request http (yêu cầu) từ người dùng. Chỉ được dùng trong các ứng dụng web.**
* **Session : Bean được sinh ra thông qua các http session.**
* **Global-session : Bean được sinh ra thông qua các request http (yêu cầu) từ người dùng. Chỉ được dùng trong các ứng dụng web.**

**Ví dụ sử dụng @Scope với phạm vi là request**

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Component  @Scope("request")  public class Contact {  } |

**12. @Valid**

**Dùng để kiểm tra dữ liệu có đúng như mình mong muốn hay không. Ví dụ dưới đây mình mong muốn name là không được rỗng , author không được rỗng. Nếu dữ liệu bị rỗng thì @Validate sẽ bắt lỗi.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | @Entity  public class Book {  @Id  @GeneratedValue  private Long id;  @NotEmpty(message = "Please provide a name")  private String name;  @NotEmpty(message = "Please provide a author")  private String author;  @NotNull(message = "Please provide a price")  @DecimalMin("1.00")  private BigDecimal price;  //...  }  @RestController  public class BookController {  @PostMapping("/books")  Book newBook(@Valid @RequestBody Book newBook) {  return repository.save(newBook);  }  //...  } |

**13. @Controller**

**Một class được đánh dấu là @Controller thì để khai báo Class đó là một controller và có nhiệm vụ mapping request trên url vào các method tương ứng trong controller. Ví dụ dưới đây mình khai báo Class HomeController là một Controller . Khi người dùng gõ vào http://localhost:8080/ thì sẽ được xử lý bởi Class HomeController. Như vậy nhiệm vụ của Controller là điều hướng các request (yêu cầu) người dùng vào method xử lý tương**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @Controller  public class HomeController {  @RequestMapping("/")  public String homePage() {  return "home";  }  } |

**14. @RequestMapping**

**Có nhiệm vụ ánh xạ các request (yêu cầu) người dùng vào method tương ứng trong controller. Ví dụ : Khi ta nhập vào url là http://localhost:8080/method2 thì nó sẽ được xử lý bởi phương thức là public String method2().**

**Ví dụ : Khi ta nhập vào url là http://localhost:8080/method3 thì nó sẽ được xử lý bởi phương thức là public String method3().**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | //Xử lý cho request có phương thức http là post và url hoặc action method là "/home/method2  @RequestMapping(value = "/method2", method = RequestMethod.POST)  public String method2() {  return "method2";  }  //Xử lý cho request có phương thức http là post hoặc get có url hoặc  // form action method là "/home/method3  @RequestMapping(value = "/method3", method = {RequestMethod.POST, RequestMethod.GET})  public String method3() {  return "method3";  } |

**15. @PathVariable**

**@PathVariable được sử dụng để xử lý những URI động, có một hoặc nhiều parameter trên URI.**

**Ví dụ bên dưới khi người dùng gõ vào là http://localhost:8080/test2/10/nguyen.**

**test2 : sẽ ứng với method test2 trong controller thông qua @Request**

**10 : số 10 sẽ được gán vào biên id nhờ PathVariable (@PathVariable(“id”) int id)**

**nguyen : chữ nguyên sẽ gán vào biến name nhờ PathVariable (@PathVariable(“name”) String name)**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @RequestMapping("/test2/{id}/{name}")  public String test2(@PathVariable("id") int id, @PathVariable("name") String name, Model model) {  model.addAttribute("id", id);  model.addAttribute("name", name);  return "test2";  } |

**16. @RequestParam**

**Chúng ta sử dụng @RequestParame để bắt các giá trị các tham số mà người dùng truyền vào trên url theo định dạng key và value.**

**Ví dụ mình có cái link sau http://localhost:8080/api/foos?id=abc . Bây giờ mình muốn lấy giá trị abc của tham số id trên url thì mình sẽ dùng @RequestParam . Ở đây mình khai báo giá trị tham số trên URL theo định dạng key = value (id=abc).**

**Chúng ta khai báo @RequestParame trong method getFoos(@RequestParam String id) . Như vậy biến id sẽ có giá trị là abc nhờ cơ chế mapping.**

|  |  |
| --- | --- |
| 1  2  3  4  5 | @GetMapping("/api/foos")  @ResponseBody  public String getFoos(@RequestParam String id) {  return "ID: " + id;  } |

**17. @ModelAttribute**

**Một trong những annotation quan trọng trong Spring đó là @ModelAttribute. Chúng ta sử dụng ModelAttribute như một cầu nối giữa Controller và View. Từ Controller chúng ta truyền các dữ liệu qua cho View thông qua ModelAttribute. Từ View chúng ta sẽ sử dụng Thymeleaf để đọc các dữ liệu từ model và hiển thị ra cho người dùng.**

**Tầng View chúng ta sử dụng model để lấy các giá trị từ người dùng và gắn vào thuộc tính modelAttribute.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <form:form method="POST" action="/spring-mvc-basics/addEmployee"  modelAttribute="employee">  <form:label path="name">Name</form:label>  <form:input path="name" />  <form:label path="id">Id</form:label>  <form:input path="id" />  <input type="submit" value="Submit" />  </form:form> |

**Ở tằng Controller ta sử dụng @ModelAttribute là tham số trong phương thức để lấy các giá trị từ view truyền vào.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @RequestMapping(value = "/addEmployee", method = RequestMethod.POST)  public String submit( @ModelAttribute("employee") Employee employee) {  model.addAttribute("name", employee.getName());  model.addAttribute("id", employee.getId());  employeeMap.put(employee.getId(), employee);  return "employeeView";  } |

**18. @RequestBody**

**được sử dụng để lấy các giá trị mà người dùng gửi lên server mà các giá trị đó được chứa trong phần thân (body) của request**

**Ví dụ như mình request sau gửi lên server dữ liệu (sendInfo) là một json gồm có tên, địa chỉ bằng method post và dữ liệu được gửi trong phần thân của request. Để nhận được dữ liệu json này từ client thì chúng ta dùng @RequestBody trong method để lấy kết quả.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | var name = $("#id-manuf-name").val();  var address = $("#id-manuf-address").val();  var phone = $("#id-manuf-phone").val();  var sendInfo = {  Name: name,  Address: address,  Phone: phone  $.ajax({  type: "POST",  url: "/Home/Add",  dataType: "json",  success: function (msg) {  if (msg) {  alert("Somebody" + name + " was added in list !");  location.reload(true);  } else {  alert("Cannot add to list !");  }  },  data: sendInfo  }); |

**Trong method handle ta sử dụng @RequestBody để lấy dữ liệu json (sendInfo) từ client gửi lên và gán giá trị đó cho biến body**

|  |  |
| --- | --- |
| 1  2  3  4 | @RequestMapping(path = "/something", method = RequestMethod.PUT)  public void handle(@RequestBody String body, Writer writer) throws IOException {  writer.write(body);  } |

**19. @ResponseBody**

**Chúng ta sử dụng @ResponseBody để nói cho controller biết rằng ta sẽ trả về một đối tượng Object kiểu Json cho client chứ mình không render ra một trang view.**

|  |  |
| --- | --- |
| 1  2  3  4 | @RequestMapping(path = "/something", method = RequestMethod.PUT)  public @ResponseBody String helloWorld() {  return "Hello World";  } |

**20. @RequestHeader và @ResponseHeader**

**@RequestHeader được sử dụng khi ta muốn lấy dữ liệu được truyền bằng Header của một request (yêu cầu từ client)**

**Ví dụ sau ta truyền thêm biến my-number trong phần header của request gửi lên server. @RequestHeader được khai báo trong phương thức doubleNumber có nhiệm vụ lấy giá trị từ header truyền vào biên**

|  |  |
| --- | --- |
| 1  2  3  4  5 | @GetMapping("/double")  public ResponseEntity<String> doubleNumber(@RequestHeader("my-number") int myNumber) {  return new ResponseEntity<String>(String.format("%d \* 2 = %d",  myNumber, (myNumber \* 2)), HttpStatus.OK);  } |

**@ResponseHeader**

**Chúng ta sử dụng @ResponseHeader khi mình muốn trả về thêm dữ liệu cho client ở phần trên cùng của mỗi response**

**Ví dụ sau ta trả thêm các giá trị ở trên phần header cho client thông qua phương thức response.setHeader**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public String addUser(@Valid User user, BindingResult bindingResult,HttpServletRequest request,HttpServletResponse response)  {  if(bindingResult.hasErrors())  {  bindingResult.getFieldError();  return"edit";  }  response.setHeader("Cache-Control","no-cache,no-store,must-revalidate");  response.setHeader("Pragma","no-cache");  response.setDateHeader("Expires", 0);  return "redirect:/welcome/profile/"+user.getName();  } |

**21. @SessionAttribute**

**Chúng ta sử dụng @SessionAttribute để lưu trữ các giá trị trong một phiên làm việc. Giống như mình làm một ứng dụng shopping cart . Khi người dùng chọn 1 sản phẩm thì mình dùng session mình lưu lại. Khi khách hàng thanh toán giỏ hàng thì mình lấy hết tất cả các mặt hàng chứa trong session ra và tính toán**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | @Controller  @SessionAttributes("shoppingCart")  public class AddToCartController {  @PostMapping("/addToCart")  public String addToCart(final Model model, @ModelAttribute ShoppingCart shoppingCart, final String productCode) {  if (shoppingCart != null) {  //add product to the shopping cart list  shoppingCart.setProduct(productCode);  model.addAttribute("cart", shoppingCart);  } else {  ShoppingCart cart = new ShoppingCart();  cart.setCustomerName("Super customer");  cart.setProduct(productCode);  model.addAttribute("cart", cart);  }  return "redirect:" + "product-detail-page";  }  @ModelAttribute("shoppingCart")  public ShoppingCart shoppingCart() {  return new ShoppingCart();  }  } |