Spring MVC Annotations

LAHIRU
LIYANAPATHIRANA

What is Spring MVC?

Spring MVC is a powerful web framework that is part of the Spring ecosystem.

Spring MVC simplifies web development with its separation of concern and robust and scalable capabilities.

Spring MVC provides a variety of annotations that simplify the development of web applications by allowing developers to define behavior and configurations directly in their code.

@Controller

Classes annotated with the @Controller, handle HTTP requests and return responses. They can contain methods that process input and return view names with model data through ModelAndView objects or String view names.

```java
@Controller
public class HomeController {
    @RequestMapping("/home")
    public String home() {
        return "home"; // Return view name
    }
}
```

LAHIRU
LIYANAPATHIRANA

@ResponseBody

Indicates that the method returns the data directly to the HTTP response body, bypassing the view resolution. This is used in RESTful APIs and can automatically serialize objects to JSON/XML.

```java
@RequestMapping(value = "/hello", method = RequestMethod.GET)
@ResponseBody
public String sayHello() {
    return "Hello, World!"; // Return response
}
```


@RestController

It is a specialized version of the @Controller that combines @Controller and @ResponseBody. It is specifically designed for RESTful web services where every method returns data rather than a view.

```java
@RestController
public class RestApiController {

    @GetMapping("/hello")
    public String greeting() {
        return "Hello, World!"; // Return response
    }
}
```

LAHIRU
LIYANAPATHIRANA

@ResponseStatus

Marks method or exception class with HTTP status code and reason. It is possible to provide a custom reason message.

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException(String message) {
        super(message);
    }
}
```

LAHIRU
LIYANAPATHIRANA

@RequestMapping

Maps incoming HTTP requests to specific handler classes or handler methods within a controller.

It supports multiple attributes:

- **value/path:** URL pattern (e.g., "/users", "/products/{id}")
- **method:** HTTP method (GET, POST, PUT, DELETE, etc.)
- **params:** Request parameters
- **headers:** Request headers (e.g., headers = "content-type=text/*")
- **consumes:** Content-Type (e.g., consumes = "application/json")

@RequestMapping

- **produces:** Response Content-Type (e.g., produces= "application/json")

- Path variables (e.g., "/products/{id}")

```
@Controller
@RequestMapping("/products")
public class ProductController {

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public String getProduct(@PathVariable int id, Model model) {
        model.addAttribute("productId", id);
        return "productDetails"; // View name
    }
}
```


@GetMapping

It is a shortcut for @RequestMapping(method = RequestMethod.GET).

It maps HTTP GET requests to specific handler methods and is ideally used for read/retrieve operations. Supports all @RequestMapping attributes.

```java
@RestController
public class ProductController {

    @GetMapping("/products")
    public List<String> listProducts() {
        return List.of("Product A", "Product B", "Product C");
    }
}
```


@PostMapping

It is a shortcut for @RequestMapping(method = RequestMethod.POST).

It maps HTTP POST requests to specific handler methods and is ideally used to create new resources and form submissions.

```java
@RestController
public class ProductController {

    @PostMapping("/products")
    public String addProduct(@RequestBody String productName) {
        return "Product added: " + productName;
    }
}
```

LAHIRU
LIYANAPATHIRANA

@PutMapping

It is a shortcut for @RequestMapping(method = RequestMethod.PUT).

It maps HTTP PUT requests to specific handler methods and is ideally used to update existing resources.

```
@RestController
public class ProductController {

    @PutMapping("/products/{id}")
    public String updateProduct(@PathVariable int id, @RequestBody String productName) {
        return "Updated Product " + id + " to " + productName;
    }
}
```


@DeleteMapping

It is a shortcut for @RequestMapping(method = RequestMethod.DELETE).

It maps HTTP DELETE requests to specific handler methods and is ideally used to remove resources.

```
@RestController
public class ProductController {

    @DeleteMapping("/products/{id}")
    public String deleteProduct(@PathVariable int id) {
        return "Deleted Product with ID: " + id;
    }
}
```


@PatchMapping

It is a shortcut for @RequestMapping(method = RequestMethod.PATCH).

It maps HTTP PATCH requests to specific handler methods and is ideally used to update and process resources partially.

```java
@RestController
public class ProductController {

    @PatchMapping("/products/{id}")
    public String patchProduct(@PathVariable int id, @RequestBody String productName) {
        return "Partially Updated Product " + id + " to " + productName;
    }
}
```


@RequestParam

Binds request parameters to the method parameters and extracts query parameters from the URL. It can handle multiple parameter types.
It supports the following attributes:

- **required:** Specifies if the parameter is mandatory
- **defaultValue:** Default value if the parameter is missing
- **value/name:** Parameter name

```
@RestController
public class UserController {

    @GetMapping("/users")
    public String getUser(@RequestParam(name = "id", required = false,
    defaultValue =     "1") int id) {
        return "User ID: " + id;
    }
}
```

LAHIRU
LIYANAPATHIRANA

@PathVariable

Extracts values from the URL template variables, and binds them to the method parameters. In the URL ("/products/{id}"), “id” can be extracted to a method parameter. It can be used to make dynamic URLs.

```java
@RestController
public class ProductController {

    @GetMapping("/products/{id}")
    public String getProductById(@PathVariable int id) {
        return "Product ID: " + id;
    }
}
```

LAHIRU
LIYANAPATHIRANA

@RequestBody

Binds HTTP request body to method parameter and automatically deserialize JSON/XML. It is used in POST or PUT requests to read data sent by the client and supports content negotiations.

```
@RestController
public class UserController {

    @PostMapping("/users")
    public String addUser(@RequestBody User user) {
        return "Added User: " + user.getName();
    }
}
```


@RequestHeader

Binds HTTP request header to method parameter and it is useful when extracting specific headers. Request headers can be marked as required or optional and support default values.

```java
@RestController
public class HeaderController {

    @GetMapping("/headers")
    public String getHeaders(@RequestHeader("User-Agent") String userAgent) {
        return "User-Agent: " + userAgent;
    }
}
```


@ModelAttribute

Binds a method parameter or method return value to a named model attribute and automatically populates the object with data from the form submissions.

```
@Controller
public class UserController {

    @PostMapping("/register")
    public String registerUser(@ModelAttribute User user) {
        return "User registered: " + user.getName();
    }
}
```


@SessionAttributes

Specifies attributes that are stored in the session, which enables maintaining the state between requests. It is used for multi-step forms and session-bound tasks.

```java
@Controller
@SessionAttributes("user")
public class UserController {

    @ModelAttribute("user")
    public User createUser() {
        return new User();
    }

    @GetMapping("/profile")
    public String getUserProfile(@ModelAttribute("user") User user) {
        return "User Profile: " + user.getName();
    }
}
```


@CrossOrigin

Enables cross-origin requests and can be applied at the class or method level.
It supports the following attributes:

- **origins:** Allowed origins
- **methods:** Allowed methods
- **allowedHeaders:** Allowed headers
- **exposedHeaders:** Exposed headers
- **allowCredentials:** Allows credentials

```java
@RestController
@CrossOrigin(origins = "http://example.com", allowedHeaders = "*")
public class ProductController {

    @GetMapping("/products")
    public List<String> getProducts() {
        return List.of("Product A", "Product B", "Product C");
    }
}
```

@CookieValue

Binds cookie values to handler method parameters.
It supports the following attributes:

- **required:** Specifies if the cookie is mandatory
- **defaultValue:** Default value if the cookie is missing
- **name/value:** Cookie name

```java
@RestController
public class CookieController {

    @GetMapping("/cookie")
    public String getCookie(@CookieValue(name = "sessionId",
    defaultValue = "defaultSession") String sessionId) {
        return "Session ID: " + sessionId;
    }
}
```


@Valid

Triggers validation of an object on the method level. It is used with the BindingResult interface to capture validation errors.

This is used in conjunction with the following validation annotations and custom validations:

- @NotNull: Ensures field is not null
- @NotEmpty: Ensures collection/string is not empty
- @NotBlank: Ensures string is not null/empty/whitespace
- @Size: Validates collection size or string length
- @Min/@Max: Validates numeric bounds
- @Pattern: Validates string against regex
- @Email: Validates email format

@Valid

- @Positive/@PositiveOrZero: Validates numbers
- @Future/@Past: Validates dates
- @Range: Validates numeric ranges

```
@RestController
public class UserController {

    @PostMapping("/users")
    public String addUser(@Valid @RequestBody User user, BindingResult result) {
        if (result.hasErrors()) {
            return "Validation errors: " + result.getAllErrors();
        }
        return "Added User: " + user.getName();
    }
}
```

```
public class User {
    @NotBlank
    private String name;

    @Email
    private String email;
}
```

LAHIRU
LIYANAPATHIRANA

@Validated

Marks class for Spring validation, and enabled validation groups.

```java
@RestController
@Validated
public class UserController {

    @GetMapping("/users")
    public String validateAge(@RequestParam @Min(18) int age) {
        return "Valid Age: " + age;
    }
}
```


@Validated

Defines a global exception handler for all controllers. It is used for centralized exception handling.

It supports:

- **basePackages:** Limits scanning to specific packages
- **annotations:** Limits to controllers with specific annotations
- **assignableTypes:** Limits to specific controller types

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<String> handleNotFound(ResourceNotFoundException ex) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
    }
}
```


@RestControllerAdvice

It is a specialized version of @ControllerAdvice that combines @ControllerAdvice and @ResponseBody.

It is used for global RESTful exception handling.

```java
@RestControllerAdvice
public class GlobalRestExceptionHandler {

    @ExceptionHandler(RuntimeException.class)
    public ResponseEntity<String> handleRuntimeException(RuntimeException ex) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
        .body("Error: " + ex.getMessage());
    }
}
```

LAHIRU
LIYANAPATHIRANA

@ExceptionHandler

Handles exceptions thrown by controller methods. It can be configured to handle specific exceptions and custom error responses.

```
@RestController
public class ProductController {

    @ExceptionHandler(IllegalArgumentException.class)
    public ResponseEntity<String> handleIllegalArgumentException(IllegalArgumentException
    exception) {
        return ResponseEntity.badRequest().body("Error: " + exception.getMessage());
    }
}
```


@InitBinder

Registers custom editors for data binding, validation, type conversions, and pre-processing during binding.

```java
@Controller
public class BinderController {

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        binder.registerCustomEditor(LocalDate.class, new PropertyEditorSupport() {
            @Override
            public void setAsText(String text) {
                setValue(LocalDate.parse(text));
            }
        });
    }

    @GetMapping("/date")
    public String getDate(@RequestParam("date") LocalDate date) {
        return "Parsed Date: " + date;
    }
}
```

LAHIRU
LIYANAPATHIRANA

Did You Find This Post Useful?

Stay Tuned for More Spring Related Posts Like This

LAHIRU
LIYANAPATHIRANA