03-2: Spring Rest-Request Data Binding–Exception-OpenAPI

Trần Văn Thịnh

Spring Rest-
Request Data
Binding–
Exception-
OpenAPI

- Request data binding
- Validate Request Data
- OpenAPI
- Exception

Request Data Binding

@RequestParam

@PathVariable

@RequestBody

@RequestParam

- Lấy dữ liệu (query parameter) từ query String trong request url. Ví dụ http://localhost:8080/foods?id=abc&type=mexico thì query String là phần đặt sau dấu ?, được ghi dưới dạng:key=value.

```
//http://localhost:8080/foods?id=abc&type=mexico
@GetMapping("/foods")  //Nếu đặt trùng @GetMapping("/food") sẽ báo lỗi khi biên dịch
public String getFoodByIdAndType(@RequestParam(name = "id") String foodId, @RequestParam String type) {
  return "Food ID: " + foodId + ", type: " + type;
}
```

- Mặc định trường nào dùng với ***@RequestParams*** phải có trong query param, nếu không sẽ báo lỗi. Nếu trường đó là không bắt buộc, thì khai báo thêm required=false

```
//http://localhost:8080/foods-plus?id=abc&type=mexico
@GetMapping("/foods-plus")
public String getFoodByIdAndTypeAndDescription(@RequestParam(name = "id") String foodId,
  @RequestParam String type, @RequestParam(required = false) String description) {
  return "Food ID: " + foodId + ", type: " + type + ", description: " + (description == null ? "No description" : description);
}
```

@RequestParam

- Nếu muốn lấy tất cả các param trong query string, thì sử dụng @RequestParam Map<String,String> allParams

```
//http://localhost:8080//api/foos?id=abc&type=mexico
@GetMapping("/api/foos")
public String getFoos(@RequestParam Map<String,String> allParams) {
  return "Parameters are " + allParams.entrySet();
}
```

- Với @RequestParam thì parameter được sử dụng URL decoded. Ví dụ: “id=abc+d” thì khi xử lý lấy ra RequestParam id sẽ có giá trị là “abc d”

@PathVariable

- Lấy dữ liệu (path variable) từ uri path (đường dẫn)

```
//http://localhost:8080/city/id/1
@GetMapping("/city/id/{cityId}")
public String getCityID(@PathVariable int cityId) {
  return "City id: " + cityId;
}
```

- Mặc định trường nào dùng với ***@PathVariable*** phải có trong uri path, nếu không sẽ báo lỗi. Nếu trường đó là không bắt buộc, thì khai báo thêm required=false

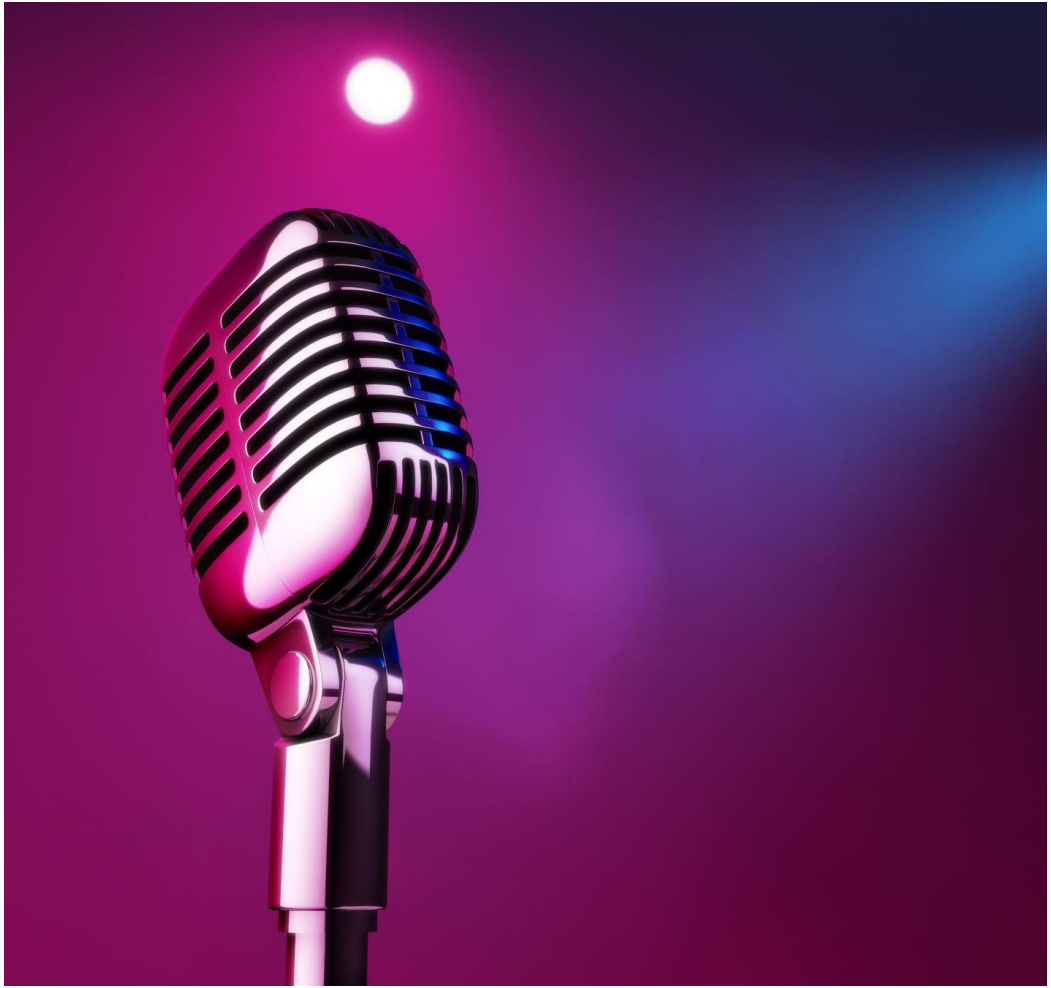
```
//http://localhost:8080/city/vietnam/hanoi
@GetMapping("/city/{countryName}/{cityName}")
public String getCity(@PathVariable(required=false) String countryName, @PathVariable String cityName) {
  return "City name: " + cityName + (countryName == null ? "" : " of country: " + countryName);
}
```

- Giá trị sử dụng trong ***@PathVariable*** là giá trị chính xác, không decode như **@RequestParam**

@RequestBody

- Map HttpRequest body với 1 object, có "automatic deserialization" để chuyển thành java object. Mặc định sử dụng định dạng json

```
//http://localhost:8080//api/foods/ (Method=POST) (Use postman to run)
@PostMapping("/api/foods")
public ResponseEntity<Food> createUser(@RequestBody Food food) {
  return ResponseEntity.ok(new Food(10, food.getName(), food.getType(), food.getDescription()));
}
}
```



Demo:
https://github.com/TechMaster/SpringBootBasic/tree/main/form/pathquery

Validate Request Data

Validate Request Data

- Mặc định với Spring MVC xử lý sẵn các trường hợp như thiếu request param, path variable, type không đúng (type miss matched)…
- Trong trường hợp cần validate logic riêng, đơn giản, như lớn hơn bao nhiêu, nhỏ hơn bao nhiêu… thì có thể sử dụng các annotation trong javax validation (**Java Bean Validation**)
- 1 số annotation hay dùng trong **JSR(JSR-303:BeanValidation1, JSR-380 BeanValidation2)**
  - ***@NotNull, @NotEmpty, @NotBlank***
  - ***@AssertTrue***
  - ***@Size***
  - ***@Min, @Max***
  - ***@Email***
  - ***@Positive, @PositiveOrZero, @Negative, @NegativeOrZero***
  - ***@Past, @PastOrPresent, @Future, @FutureOrPresent***

Validate Request Data

- Dependencies để sử dụng Validation API (not spring-boot)
  - Validation API

```
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>2.0.1.Final</version>
</dependency>
```

  - Validation API Reference Implementation

```
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.13.Final</version>
</dependency>
```

- Cho spring boot, sử dụng spring-boot-starter-validation

Validate Request Data

- Với Spring boot, để sử dụng validator, cần sử dụng thêm
  - @Validated (annotation thường sử dụng cho class level hoặc method group)
  - Hoặc @Valid (thường sử dụng cho field(parameter, Object)
  - Với @RequestParam và @PathVariable có thể sử dụng cả @Validated và @Valid, Với @RequestBody và dùng các annotation để validate trong POJO, thường dùng @Valid
  - Ngoài ra có thể viết các Custom validation annotation hoặc viết service code để validate

Validate Request Data

- Demo: https://github.com/TechMaster/SpringBootBasic/tree/main/form/pathquery

OPEN API

OpenAPI

- Swagger hay OpenAP Ià một opensource được sử dụng để phát triển thiết kế và làm các document cho các REST API và còn có một số tính năng khác
- Hỗ trợ các annoutation giúp cho ứng dụng spring của ta có thể mô tả đươc các restapi, và đồng thơi build một giao diện giúp người dùng có thể tương tác trực tiếp với các rest api đó.

OpenAPI 3.0

- https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.0.md

Swagger2	OpenApi 3.0	Description
@Api	@Tag	Đánh dấu 1 class là nơi chứa các API
@ApiModel	không còn	Đánh dấu 1 class là Swagger Model
@ApiModelProperty	@Schema	Bổ sung các thông tin cho data model
@ApiOperation	@Operation	Mô tả cho một API và response của nó
@ApiParam	@Parameter	Mô tả các parameter
@ApiResponse	@ApiResponse	Mô tả status code của response
@ApiResponses	@ApiResponses	Mô tả danh sách các status code của response

OpenAPI

- Demo https://github.com/TechMaster/SpringBootBasic/tree/main/rest/02OpenAPI/bookstore

Xử lý error cho Spring Rest

- *@ExceptionHandler*
- *HandlerExceptionResolver*
- *@ControllerAdvice*
- *ResponseStatusException*
- Spring Boot Support

Controller-Level @*ExceptionHandler*

- Sử dụng @Controller cho method trong controller class. Method sẽ xử lý những exception được đánh dấu

```
public class FooController{

    //...
    @ExceptionHandler({ CustomException1.class, CustomException2.class })
    public void handleException() {
        //
    }
}
```

- Cách này xử lý được các exception cho từng controller cụ thể, không thể sử dụng cho toàn bộ ứng dụng

HandlerExceptionResolver

Thiết lập một *HandlerExceptionResolver* sẽ giúp xử lý tất cả các exception được throw bởi application. Nó cũng cho phép tạo một cơ chế xử lý exception thống nhất cho cả REST Api

HandlerExceptionResolver là một interface, Spring có sẵn 1 số class thực thi nó

- *ExceptionHandlerExceptionResolver*
- *DefaultHandlerExceptionResolver:*
- *ResponseStatusExceptionResolver*
- *SimpleMappingExceptionResolver* và *AnnotationMethodHandlerExceptionResolver*
- Custom *HandlerExceptionResolver*

- *Core class sử dụng cho @ExceptionHandler*
- *Kích hoạt mặc định bởi Spring Dispatcher Servlet*

ExceptionHandlerExceptionResolver

- *Kích hoạt mặc định bởi Spring Dispatcher Servlet*
- Sử dụng để xử lý các exception thông dụng cùng với các HTTP Status Code tương ứng (4xx client error và 5xx server error). Tham khảo list dưới đây: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-ann-rest-spring-mvc-exceptions
    - Thiết lập đúng các response status code
    - Không thực hiện thiết lập bất cứ thứ gì cho body của response

DefaultHandlerExceptionResolver

- Sử dụng cho @ResponseStatus để map exceptions với HTTP status code
- Giống như *DefaultHandlerExceptionResolver*, class này để body null

```
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class MyResourceNotFoundException extends RuntimeException {
    public MyResourceNotFoundException() {
        super();
    }
    public MyResourceNotFoundException(String message, Throwable cause) {
        super(message, cause);
    }
    public MyResourceNotFoundException(String message) {
        super(message);
    }
    public MyResourceNotFoundException(Throwable cause) {
        super(cause);
    }
}
```

ResponseStatusExceptionResolver

- Sử dụng cho @ResponseStatus để map exceptions với HTTP status code
- Giống như *DefaultHandlerExceptionResolver*, class này để body null

```
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class MyResourceNotFoundException extends RuntimeException {
    public MyResourceNotFoundException() {
        super();
    }
    public MyResourceNotFoundException(String message, Throwable cause) {
        super(message, cause);
    }
    public MyResourceNotFoundException(String message) {
        super(message);
    }
    public MyResourceNotFoundException(Throwable cause) {
        super(cause);
    }
}
```

ResponseStatusExceptionResolver

- *SimpleMappingExceptionResolver* sử dụng cho Spring MVC, map class name với view name
- *AnnotationMethodHandlerExceptionResolver* được thay thế bởi *ExceptionHandlerExceptionResolver*

SimpleMappingExceptionResolver và AnnotationMethodHandlerExceptionResolver

- Có thể tạo một custom exception resolver bằng cách extends AbstractHandlerExceptionResolver

Custom HandlerExceptionResolver (tùy biến)

@ControllerAdvice

- Hỗ trợ tạo một phiên bản global cho ***@ExceptionHandler***
- Class với @ControllerAdvice sẽ có tác dụng với toàn bộ các controller khác trong ứng dụng
- Cơ chế đơn giản nhưng rất linh hoạt, cần dùng chính xác exception khai báo trong @ExceptionHandler với argument của method được đánh dấu(Có thể dùng class cha cho method)
    - Giúp toàn quyền xử lý body của response cũng như status code
    - Map được nhiều exception vào cùng một method, để xử lý cùng với nhau
    - Sử dụng tốt cho Restful ResponseEntity response

```
@ControllerAdvice
public class RestResponseEntityExceptionHandler
  extends ResponseEntityExceptionHandler {

    @ExceptionHandler(value
      = { IllegalArgumentException.class, IllegalStateException.class })
    protected ResponseEntity<Object> handleConflict(
      RuntimeException ex, WebRequest request) {
        String bodyOfResponse = "This should be application specific";
        return handleExceptionInternal(ex, bodyOfResponse,
          new HttpHeaders(), HttpStatus.CONFLICT, request);
    }
}
```

@ResponseStatusException

- Có từ Spring 5, có thể tạo một instance của HttpStatus và các tùy chọn reason và cause
- Lợi ích:
  - Có thể triển khai 1 giải pháp cơ bản khá nhanh
  - Một loại exception có thể trả ra nhiều response khác nhau, giảm phụ thuộc hơn so với @ExceptionHandler
  - Không phải tạo nhiều exception class, có nhiều quyền kiểm soát hơn với việc xử lý các exception
- Nhược điểm
  - Khó thống nhất để xử lý exception
  - Code bị sao chép(Duplicate code) trong nhiều controller

```
@GetMapping(value = "/{id}")
public Foo findById(@PathVariable("id") Long id, HttpServletResponse response) {
    try {
        Foo resourceById = RestPreconditions.checkFound(service.findOne(id));

        eventPublisher.publishEvent(new SingleResourceRetrievedEvent(this, response));
        return resourceById;
     }
    catch (MyResourceNotFoundException exc) {
         throw new ResponseStatusException(
           HttpStatus.NOT_FOUND, "Foo Not Found", exc);
    }
}
```

@Spring Boot Support

- Spring Boot cung cấp một ErrorController implementation để quản lý errors
- Cấu hình bằng các properties
    - *server.error.whitelabel.enabled*:Dùng để disable White label error page và dựa vào servlet container để trả về HTML error message
    - *server.error.include-stacktrace:* Với giá trị là always, chứa stacktrace trong cả HTML và JSON response mặc định
    - *server.error.include-message:* Từ Spring Boot 2.3, message trong response bị giấu đi để tránh lộ thông tin nhạy cảm, có thể sử dụng property này cùng với always value để bật nó lên
- Ngoài các property này, có thể cung cấp một view-resolver tự tạo để mapping cho path /error và override Whitelabel Page

@Spring Boot Support

- Cũng có thể tùy chỉnh các thuộc tính muốn hiển thị trong response bằng cách chứa một ErrorAttributes bean trong context. Có thể extend DefaultErrorAttributes class được cung cấp bởi Spring Boot

```
@Component
public class MyCustomErrorAttributes extends DefaultErrorAttributes {

    @Override
    public Map<String, Object> getErrorAttributes(
      WebRequest webRequest, ErrorAttributeOptions options) {
        Map<String, Object> errorAttributes =
          super.getErrorAttributes(webRequest, options);
        errorAttributes.put("locale", webRequest.getLocale()
            .toString());
        errorAttributes.remove("error");

        //...

        return errorAttributes;
    }
}
```

@Spring Boot Support

- Nếu muốn thiết lập hay override cách mà application xử lý error cho từng loại content cụ thể, có thể đăng kí một ErrorController bean, và đơn giản hơn là extend từ BasicErrorController mặc định của Spring Boot
- Ví dụ để tùy chỉnh cách application xử lý error với XML endpoints, có thể khai báo một public method với @RequestMapping

```
@Component
public class MyErrorController extends BasicErrorController {

    public MyErrorController(
      ErrorAttributes errorAttributes, ServerProperties serverProperties) {
        super(errorAttributes, serverProperties.getError());
    }

    @RequestMapping(produces = MediaType.APPLICATION_XML_VALUE)
    public ResponseEntity<Map<String, Object>> xmlError(HttpServletRequest request) {

    // ...

    }
}
```