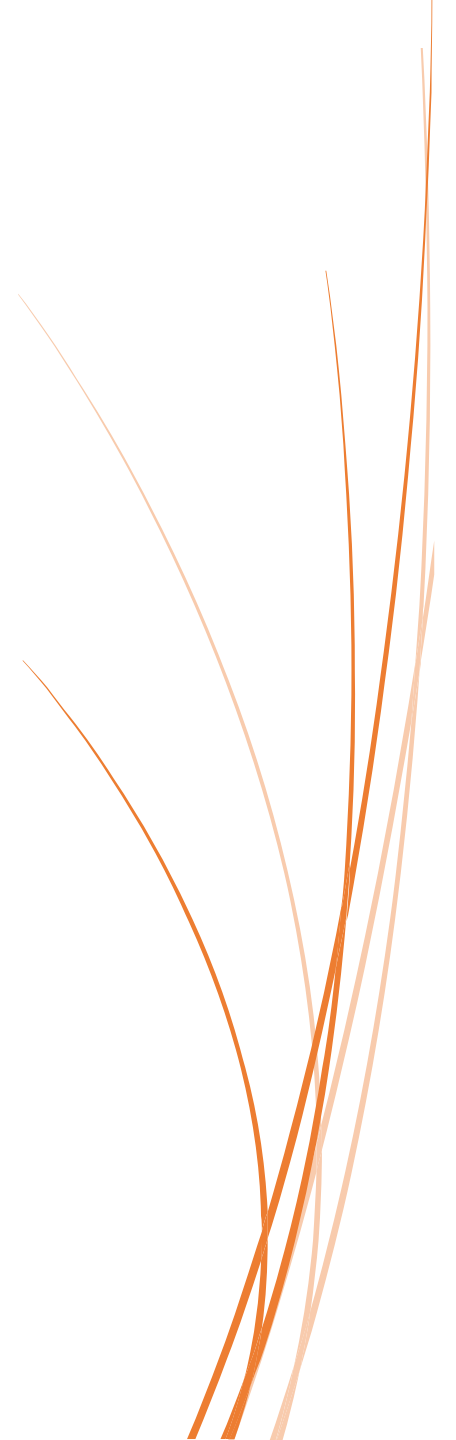


Validation in Spring Boot

Lecture notes



Many ways to validate in Spring Boot

- There are multiple ways to validate in Spring Boot
 - Boilerplate
 - Check data with standard Java code
e.g. if statements
 - Can be avoided in Spring Boot
 - Bean validation
 - Using validation annotations for a class' fields
 - Built-in validation
 - Using `@Validated` and `@Valid` annotations
 - Custom validator
 - A class that implements the [Validator](#) interface

Validation dependency in Spring Boot

- When starting a Spring Boot application at <http://start.spring.io...>

Validation

I/O

Bean Validation with Hibernate validator.

- For an existing project, add the starter validation dependency...

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-validation</artifactId>  
</dependency>
```

Bean validation

The [Hibernate Validator](#) documentation has more details

- Some frequently used validation annotations for the fields in a class
 - **@NotNull** – the field must not be null
 - **@NotBlank** – the string field must not be an empty string (i.e. "")
 - **@Pattern** – the string field is only valid when it matches the specified regular expression
 - **@Email** – the string field must be a valid email address
 - **@Min** and **@Max** – the numerical field is only valid when it's value meets these criteria
 - **@NotEmpty** – the list field must not empty

Using bean validation annotations

- Fields in a class can be annotated with validation constraints

```
public class BookDTO
{
    @Min(1)
    private final int id;

    @NotBlank
    private final String title;

    @NotBlank
    private final String author;

    @NotBlank
    private final String isbn;

    @Min(0)
    private int numberOfCopies;

    private Collection<CopyDTO> copies;
}
```

Bean validation is not automatic

In BookRestController

```
@PostMapping(path = "/create")
public BookDTO createBook(@RequestBody BookDTO bookDTO)
{
    return bookService.createBook(bookDTO);
}
```

In BookService

```
public BookDTO createBook(BookDTO b)
{
    Book book = new Book(b.getId(), b.getTitle(), b.getAuthor(), b.getIsbn(), copies: null);
    book = bookRepository.save(book);

    return dto_factory.createDTO(book);
}
```

HTTP post request with values that violate the constraints defined in BookDTO










POST http://localhost:8080/book/create

Content-Type: application/json

```
{
  "id": -1,
  "title": "",
  "author": "",
  "isbn": "",
  "numberOfCopies": -1
}
```

Bean validation is not automatic

- A book row is created in the database with invalid values

← T →		▼		id	title	author	isbn
<input type="checkbox"/>	 Edit	 Copy	 Delete	-1			
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	Java is brilliant!	Graham Mansfield	1234567890123
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	Spring Boot for beginners	Luke Moonwalker	3210987654321

- No errors are detected because validation has not been invoked

Built-in validation of request body

- To invoke Spring Boot's built-in validation process, use the `@Valid` annotation

In BookRestController

```
@PostMapping(path = "/create")
public BookDTO createBook(@RequestBody @Valid BookDTO bookDTO)
{
    return bookService.createBook(bookDTO);
}
```

- Spring Boot injects a validator which validates the payload data according to the constraints defined in `BookDTO`
 - The POST request's payload is judged to be invalid and the request is rejected

```
{
  "timestamp": "2022-10-19T13:38:22.896+00:00",
  "status": 400,
  "error": "Bad Request",
  "path": "/book/create"
}
```


Built-in validation of path variables

- Validation annotations are embedded in the endpoint method's parameter list

```
@GetMapping(path =("/{id}")  
public BookDTO getBookDetails(  
    @PathVariable("id")  
    @Min(value = 1, message = "Id must be greater than zero") int bookId)  
{  
    return bookService.getBookDetails(bookId);  
}
```

In BookRestController

- The class is annotated with `@Validated` to cause Spring Boot to perform the validation of path variables

```
@RestController  
@RequestMapping(path = "/book")  
@AllArgsConstructor  
@Validated  
public class BookRestController
```

Note: Request parameters are validated in the same way as path variables

Built-in validation of other components

- Built-in validation can be performed on any Spring components, such as services
 - Annotate the class with `@Validated`, and
 - Annotate method parameters for classes with bean validation with `@Valid`

In BookService

```
public BookDTO createBook(@Valid BookDTO b)
{
    Book book = new Book(b.getId(), b.getTitle(), b.getAuthor(), b.getIsbn(), copies: null);
    book = bookRepository.save(book);

    return dto_factory.createDTO(book);
}
```

Additional reading

- Do some background reading about the terms introduced in these notes
 - Annotations for bean validation:
 - https://docs.jboss.org/hibernate/validator/6.0/reference/en-US/html_single/#section-builtin-constraints
 - More about validation in Spring Boot:
 - <https://reflectoring.io/bean-validation-with-spring-boot/>
 - <https://fullstackcode.dev/2022/05/10/complete-guide-to-spring-boot-validation/>
 - Validation groups
 - Validating JPA entities
 - Validating programmatically
 - Validating form input
<https://spring.io/guides/gs/validating-form-input/>