Spring Boot REST web service – Part 3 – Exception Handling and Validation using @ControllerAdvice, @Valid and Custom Annotations

March 13, 2017 (https://mydevgeek.com/spring-boot-rest-web-service-part-3-exception-handling-validation-using-controlleradvice-valid-custom-annotations/) Damith Ganegoda (https://mydevgeek.com/author/fffpt/) REST (https://mydevgeek.com/category/rest/), Spring (https://mydevgeek.com/category/spring/)

This is the 4th tutorial in the Spring Boot REST web service tutorial series.

Building a simple REST Service with Spring Boot (http://mydevgeek.com/building-a-simple-rest-service-with-spring-boot/)

Part 1 – Spring Data and MySQL (http://mydevgeek.com/spring-boot-rest-web-service-part-1-spring-data-mysql/)

Part 2 – CRUD, Service Layers, Assemblers and Utility classes (http://mydevgeek.com/spring-boot-rest-web-service-part-2-crud-operations-service-layers-assemblers-utility-classes/)

1) Overview

In this tutorial, we will discuss how to handle exceptions. For an example, we want to add a new user through REST service but the user already exists. After that, we will discuss how to validate inputs like how to send a proper and readable message when missing user first name.

2) Exception Handling using `@ControllerAdvice`

Get the full source code from GitHub that related to Part 2 (https://github.com/damithme/spring-boot-REST/tree/master/spring-boot-part2). I think you remember, we get specific user details sending user ID through the Postman (https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop?hl=en). What happens when we send the user ID that doesn't exist in the database.

http://localhost:8080/user/645 (http://localhost:8080/user/645)

Output

```
{"timestamp":1489114177875,"status":500,"error":"Internal Server Err
"exception":"java.lang.NullPointerException",
"message":"No message available","path":"/user/645"}
```

This error message is not meaningful.

2.1) `@ControllerAdvice`

Actually, there are a couple of ways to handle exceptions in the Spring. In this tutorial, we are going to use @ControllerAdvice (https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#mvc-ann-controller-advice).

> The `@ControllerAdivce` annotation is a component annotation allowing implementation classes to be auto-detected through classpath scanning.

The `@ControllerAdvice` listens across the whole application for exceptions. When throws an exception, it will catch and convert it to the meaningful message.

2.2) ResourceNotFoundException

Now we are going to define a custom exception. Let say, we need to throw an exception when the application could not find the user for specific user ID. The custom exception class we can call it *UserNotFoundException* or *ResourceNotFoundException*. The **UserNotFoundException** *specifically defines for throwing when user not found in the database. But the* ***ResourceNotFoundException*** can be used for commonly for any resources. Totally, it depends on the application complexity. You can decide use common exception or define exceptions for each resource. In this tutorial, we selected the ***ResourceNotFoundException.***

Create a package **com.mydevgeek.exception** and create a new class **ResourceNotFoundException.java**

```java
package com.mydevgeek.exception;

public class ResourceNotFoundException extends RuntimeException {

    private Long resourceId;

    public ResourceNotFoundException(Long resourceId, String messag
        super(message);
        this.resourceId = resourceId;
    }
}
```

2.3) Implement ControllerAdvice

Firstly, we need a POJO class to pass a response. Create a class that name is **ExceptionResponse.java** in the **com.mydevgeek.exception**.

```java
package com.mydevgeek.exception;

public class ExceptionResponse {

    private String errorCode;
    private String errorMessage;

    public ExceptionResponse() {
    }

    public String getErrorCode() {
        return errorCode;
    }

    public void setErrorCode(String errorCode) {
        this.errorCode = errorCode;
    }

    public String getErrorMessage() {
        return errorMessage;
    }

    public void setErrorMessage(String errorMessage) {
        this.errorMessage = errorMessage;
    }
}
```

Now, create an **ExceptionHandlingController.java** class in the same package.

06 Mar , 2017

Spring Boot REST web s
Part 1 – Spring Data and
(https://mydevgeek.cor
boot-rest-web-service-p
spring-data-mysql/)
28 Feb , 2017

(https://
mydevge
ak.com/

Angular 4 CRUD applica
Spring Boot REST servic
(https://mydevgeek.cor
4-crud-application-with
boot-rest-service-part-1
18 Sep , 2017

Building a simple REST
with Spring Boot
(https://mydevgeek.cor
a-simple-rest-service-w
boot/)
14 Feb , 2017

Deploying a Spring Boo
Application on AWS usi
RDS
(https://mydevgeek.cor
spring-boot-application
ec2-rds/)
12 Apr , 2017

CATEGORIES

- Angular 4 (https://mydevgeek.com/category/angular-4/) (3)
- Apache Spark (https://mydevgeek.com/category/apache-spark/) (1)
- AWS (https://mydevgeek.com/category/aws/) (1)
- Data Structures (https://mydevgeek.com/category/data-structures/) (6)
- DesignPattern (https://mydevgeek.com/category/designpattern/) (2)
- Docker (https://mydevgeek.com/category/docker/) (1)
- Dropwizard (https://mydevgeek.com/category/dropwizard/) (1)
- IntelliJ (https://mydevgeek.com/category/intellij/) (4)
- Java (https://mydevgeek.com/category/java/) (26)

```java
package com.mydevgeek.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice
public class ExceptionHandlingController {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ExceptionResponse> resourceNotFound(Resou
        ExceptionResponse response = new ExceptionResponse();
        response.setErrorCode("Not Found");
        response.setErrorMessage(ex.getMessage());

        return new ResponseEntity<ExceptionResponse>(response, Http
    }
}
```

Then, we have to change **UserServiceImpl.java**. If it could not find a user for given id then it will be thrown the *ResourceNotFoundException*.

```java
@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserRepository userRepository;

    public User getUserById(Long id) {
        User user = userRepository.findOne(id);

        if (user == null) {
            throw new ResourceNotFoundException(id, "user not found
        }
        return user;
    }

    //other methods
}
```

Then build and run the application. And send GET request via Postman.

http://localhost:8080/user/645 (http://localhost:8080/user/645)

Output

```
{"errorCode":"Not Found","errorMessage":"user not found"}
```

It throws the **404** not-found error. Using the ControllerAdvice you can add any no of exceptions.

Frequently used HTTP status codes

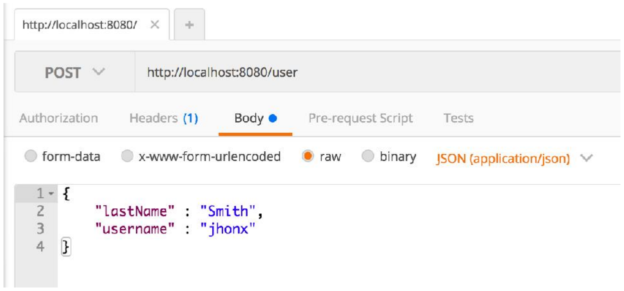
- **200 OK –** the request has succeeded.
- **201 Created –** the request has succeeded and a request was created a new resource (ex:- create a new user).
- **204 Not Content –** the request has succeeded and no need to return any content (ex:- such as update operation)
- **400 Bad Request –** the request could not be understood.
- **401 Unauthorized –** the request require user authorization.
- **403 Forbidden –** the server refusing to fulfill the request.
- **404 Not found –** the server can not find anything that related to the request (ex:- try to get user by id that not in the server).
- **409 Conflict –** the request can not be completed due to a resource conflict. (ex:- user id already exist).

You can find more HTTP status code here (http://www.restapitutorial.com/httpstatuscodes.html).

3) Validation

In Part – 2 (http://mydevgeek.com/spring-boot-rest-web-service-part-2-crud-operations-service-layers-assemblers-utility-classes/), We were able to create and update a user by sending a request to the application. How do you handle the end user send request missing or null property?

We sent a request for creating a user without the first name.



http://localhost:8080/user

```
{
    "lastName" : "Smith",
    "username" : "jhonx"
}
```

Output

```
{"userId":12,"fullName":"null Smith","username":"jhonx"}
```

It creates a user with null value. Let say, we need to validated it.

3.1) Validate using `@Valid` and `@NotNull`

`@Valid` is an annotation that use for Bean validation.

Add `@Valid` annotation into the **UserController.java** as mentioned below.

```
@RestController
@RequestMapping("/user")
public class UserController {

    @RequestMapping(method = RequestMethod.POST)
    public UserVO createUser(@Valid @RequestBody CreateUserVO userV
        //convert to User
        User user = userAssembler.toUser(userVO);
        //save User
        User savedUser = userService.createUser(user);
        //convert to UserVO
        return userAssembler.toUserVO(savedUser);
    }

    //other methods
}
```

After that, we are going to add the `@NotNull` annotation to the **CreateUserVO.java** class.

```
package com.mydevgeek.vo;

import javax.validation.constraints.NotNull;

public class CreateUserVO {

    @NotNull(message = "first name can not be null.")
    private String firstName;

    @NotNull(message = "last name can not be null.")
    private String lastName;

    //other property and methods
}
```

Now run the application and try to send a null value for the first name.

```
{
"firstName" : null,
"lastName" : "Smith",
"username" : "jhonx"
}
```

Output

```
{"timestamp":1489216085324,"status":400,"error":"Bad Request","excep
org.springframework.web.bind.MethodArgumentNotValidException","error
["createUserVO.firstName","firstName"],"arguments":null,"defaultMess
"defaultMessage":"first name can not be null.","objectName":"createU
"bindingFailure":false,"code":"NotNull"}],"message":"Validation fail
```

Now we got an exception message but it's not a meaningful for the end user. If you look at the error message you can see it threw ***MethodArgumentNotValidException***.

What we are going to do is, add a new ExceptionHandler to the ControllerAdvice and exception converts to the meaningful error message. Add following method into the **ExceptionHandlingController.java**

```
@ControllerAdvice
public class ExceptionHandlingController {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ExceptionResponse> invalidInput(MethodArg
        ExceptionResponse response = new ExceptionResponse();
        response.setErrorCode("Validation Error");
        response.setErrorMessage(ex.getMessage());
        return new ResponseEntity<ExceptionResponse>(response, Http
    }
}
```

this will be called if MethodArgumentNOtvalid Exception happens

try to send null first name and get an error message.

output

```
{"errorCode":"Validation Error","errorMessage":"Validation failed fo
public com.mydevgeek.vo.UserVO com.mydevgeek.controller.UserControll
[NotNull.createUserVO.firstName,NotNull.firstName,NotNull.java.lang.
arguments [org.springframework.context.support.DefaultMessageSourceR
arguments []; default message [firstName]]; default message [first n
```

The message still not readable and understandable. What happens if we send first and last names as null values? We will get 2 error messages like above mentioned. So that we define the ***List*** to keep error messages in **ExceptionResponse.java**.

```
public class ExceptionResponse {

    private String errorCode;
    private String errorMessage;
    private List<String> errors;
    //getters and setters
}
```

Then modify the **ExceptionHandlingController.java** to take all error messages that define in the **CreateUserVO.java**.

```
package com.mydevgeek.exception;

import com.mydevgeek.util.ValidationUtil;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.MethodArgumentNotValidException
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice
public class ExceptionHandlingController {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ExceptionResponse> invalidInput(MethodArg
        BindingResult result = ex.getBindingResult();
        ExceptionResponse response = new ExceptionResponse();
        response.setErrorCode("Validation Error");
        response.setErrorMessage("Invalid inputs.");
        response.setErrors(ValidationUtil.fromBindingErrors(result)
        return new ResponseEntity<ExceptionResponse>(response, Http
    }
}
```

Send the same request with null first and last name.

output

```
{"errorCode":"Validation Error","errorMessage":"Invalid inputs.",
"errors":["last name can not be null.","first name can not be null."
```

Now it looks pretty good. Apart from `@NotNull` there are many annotations to validate such as `@NotBlank, @Digits, @Email, @Max, @Min` etc.

3.2) Custom annotation for validating

We can implement custom annotation for validating. Once we implemented, we can use across the whole application. We will implement simple validator. Our requirement is, the **username** can not be ***"mydevgeek"*** it must be other value.

Add new package **com.mydevgeek.util.validators**. Add new annotation interface that name is **Username.java**.

```
package com.mydevgeek.util.validators;

import javax.validation.Constraint;
import javax.validation.Payload;
import java.lang.annotation.*;

@Documented
@Constraint(validatedBy = UsernameValidator.class)
@Target( { ElementType.METHOD, ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
public @interface Username {

    String message() default "{Username}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};
}
```

create custom annotation

Class is a parametrizable class, hence you can use the syntax Class<T> where T is a type. By writing Class<?>, you're declaring a Class object which can be of any type (? is a wildcard). The Class type is a type that contains metainformation about a class.

It's always good practice to refer to a generic type by specifying his specific type, by using Class<?> you're respecting this practice (you're aware of Class to be parametrizable) but you're not restricting your parameter to have a specific type.

Then add a new class that name is **UsernameValidator.java** and *implement ConstraintValidator<A,T>*.

```
package com.mydevgeek.util.validators;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

public class UsernameValidator implements ConstraintValidator<Usern
    public void initialize(Username constraintAnnotation) {

    }

    public boolean isValid(String value, ConstraintValidatorContext
        if ("mydevgeek".equalsIgnoreCase(value)) {
            return false;
        }
        return true;
    }
}
```

Finally, add `@Username` annotation into the **CreateUserVO.java**.

```
public class CreateUserVO {

    @NotNull(message = "first name can not be null.")
    private String firstName;

    @NotNull(message = "last name can not be null.")
    private String lastName;

    @Username(message = "Invalid username.")
    private String username;
}
```

Send a request that username is “mydevgeek”.

output

```
{"errorCode":"Validation Error","errorMessage":"Invalid
inputs.","errors":["Invalid username."]}
```

Project Code :- GitHub (https://github.com/damithme/spring-boot-REST/tree/master/spring-boot-part3)

Next tutorials will be focused on Unit testing, CORS, Health Check, Logback integration.

(http://www.facebook.com/sharer.php?u=https://mydevgeek.com/spring-boot-rest-web-service-part-3-exception-handling-validation-using
annotations/&t=Spring+Boot+REST+web+service+%26%238211%3B+Part+3+%26%238211%3B+Exception+Handling+and+Validation+using+%40

(http://twitter.com/share?text=Spring+Boot+REST+web+service+%26%238211%3B+Part+3+%26%238211%3B+Exception+Handling+and+V
exception-handling-validation-using-controlleradvice-valid-custom-annotations/&via=MyDevGeek)

(https://plusone.google.com/_/+1/confirm?hl=fr-FR&url=https://mydevgeek.com/spring-boot-rest-web-service-part-3-exception-handling-v

(http://www.tumblr.com/share/photo?source=https%3A%2F%2Fi0.wp.com%2Fmydevgeek.com%2Fwp-content%2Fuploads%2F2017%2F02
logo.png%3Ffit%3D300%252C300%26ssl%3D1&caption=Spring+Boot+REST+web+service+%26%238211%3B+Part+3+%26%238211%3B+Exceptio
boot-rest-web-service-part-3-exception-handling-validation-using-controlleradvice-valid-custom-annotations%2F)

(http://www.linkedin.com/shareArticle?mini=true&url=https://mydevgeek.com/spring-boot-rest-web-service-part-3-exception-handling-vali
annotations/&title=Spring+Boot+REST+web+service+%26%238211%3B+Part+3+%26%238211%3B+Exception+Handling+and+Validation+using+%

(https://www.blogger.com/blog-this.g?u=https://mydevgeek.com/spring-boot-rest-web-service-part-3-exception-handling-validation-using-
annotations/&n=Spring+Boot+REST+web+service+%26%238211%3B+Part+3+%26%238211%3B+Exception+Handling+and+Validation+using+%4

Related Post

Spring 4.3 – Event Listener (https://mydevgeek.com/spring-4-	Building a REST service with Dropwizard (https://mydevge	Angular 4 CRUD application with Spring Boot REST s...	Spring Boot REST web service – Part 1 –... (https://mydevge

12166 total visits, 47 visits today

@ControllerAdvice (https://mydevgeek.com/tag/controlleradvice/)

annotation (https://mydevgeek.com/tag/annotation/)

java 8 (https://mydevgeek.com/tag/java-8/) rest (https://mydevgeek.com/tag/rest/)

spring (https://mydevgeek.com/tag/spring/)

spring-boot (https://mydevgeek.com/tag/spring-boot/)

1 Comment mydevgeek.com Login

Recommend 1 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Yuriy Tkachenko • 3 months ago
Thank you for great tutorials!
But what about ValidationUtil.java? It not mentioned in this article.
3 • Reply • Share ›

Subscribe Add Disqus to your siteAdd DisqusAdd Privacy

Spring Boot REST web service – Part 2 – CRUD operations, Service Layers, Assemblers and Utility classes (https://mydevgeek.com/spring-boot-rest-web-service-part-2-crud-operations-service-layers-assemblers-utility-classes/)

Spring 4.3 – Event Listener (https://mydevgeek.com/spring-4-3-event-listener/)

PAGES

Contact (https://mydevgeek.com/contact/)

SUBSCRIBE TO BLOG VIA EMAIL

Enter your email address to subscribe to MyDevGeek blog and receive notifications of new posts by email.

Join 58 other subscribers

Email Address

SUBSCRIBE