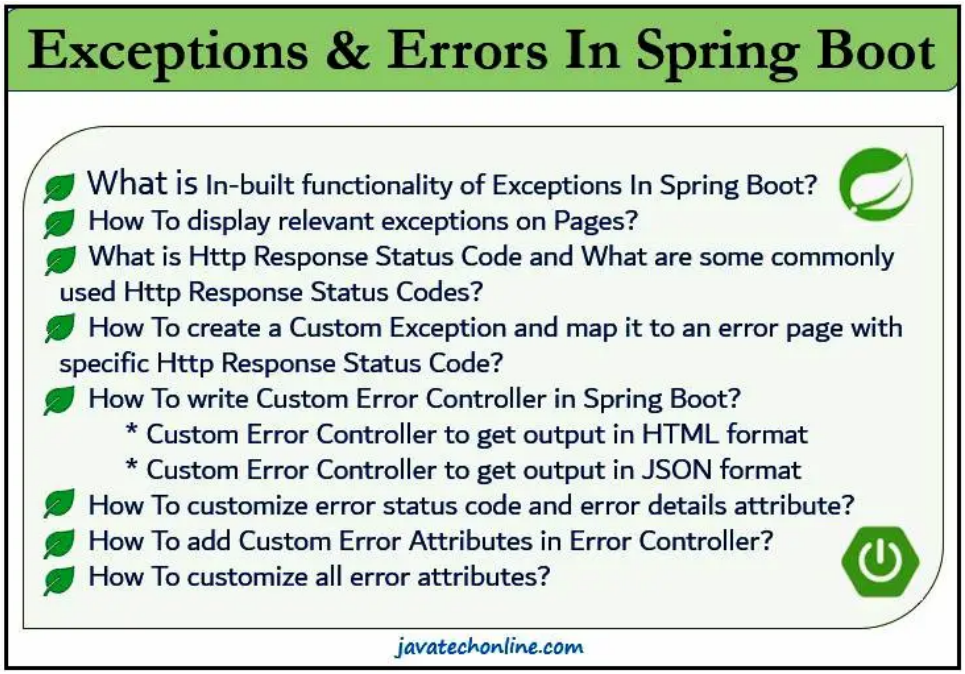
Making Java easy to learn
Java Technology and Beyond

How To Handle Exceptions & Errors In Spring Boot?

Spring Boot Java Spring by Shwetas0000 December 28, 2020 0



Everyone of us spend ample amount of time in learning big topics of Spring & Spring Boot. For example, Spring Boot REST, Spring Boot MVC, Spring Boot Security and many more. But generally we don't think about 'How to handle Exceptions & Errors in Spring Boot?'.

This topic might become the most important for running the application without any interference. Also, it is very helpful in making other developers understand our code flow easily. Even finding out the origin of errors & exceptions becomes very irritating, if we have not handled them properly in our code. Sometimes we have to debug the whole flow of code to find it out & resolve accordingly. In this way Exception handling plays an important role in our software development engagement.

In this topic 'How to handle Exceptions & Errors in Spring Boot?' we will learn handling of Errors & Exceptions step by step covering all aspects. However, Spring Boot has made our job very easy by handling most common exception at the framework level internally. Even we can observe the fascination of Spring Boot while learning & implementing the exceptions. In my opinion, every developer should go through this topic and subsequently apply the concepts learned in the rea

TOP

project. Let's discuss 'How to handle Exceptions & Errors in Spring Boot?' accordingly.

What Can You Expect from This Article as a Whole?

Table of Contents

Once you complete going through this article, you will be able to answer :

1) What are the importance & benefits of implementing Exception Handling in a Spring Boot Application?
2) How does Spring Boot's inbuilt Exception/Error handling work internally?

TOP

3) Further, How does predefined BasicErrorController work in different scenarios?
4) How to display a meaningful custom error/exception pages?
5) Additionally, What is an Http Response Status Code?
6) Also, What are some most commonly used status codes?
7) How to create a specific error page with respect to a Http Response Status Code?
8) How to create a custom exception and map it to a specific status code page?
9) Consequently, How to add custom error attributes in a custom Error Controller?

10) How does a predefined ErrorController handles exception raised by a REST call?
11) How can we customize error status code & error details attribute?
12) In the end, How can we customize all error attributes?
13) Use of annotations @ControllerAdvice, @RestControllerAdvice, @ExceptionHandler, @ResponseStatus, @ResponseBody etc.
14) Last but not the Least, 'How to handle Exceptions & Errors in Spring Boot?'.

How does Spring Boot application handle errors/exceptions internally via inbuilt functionality?

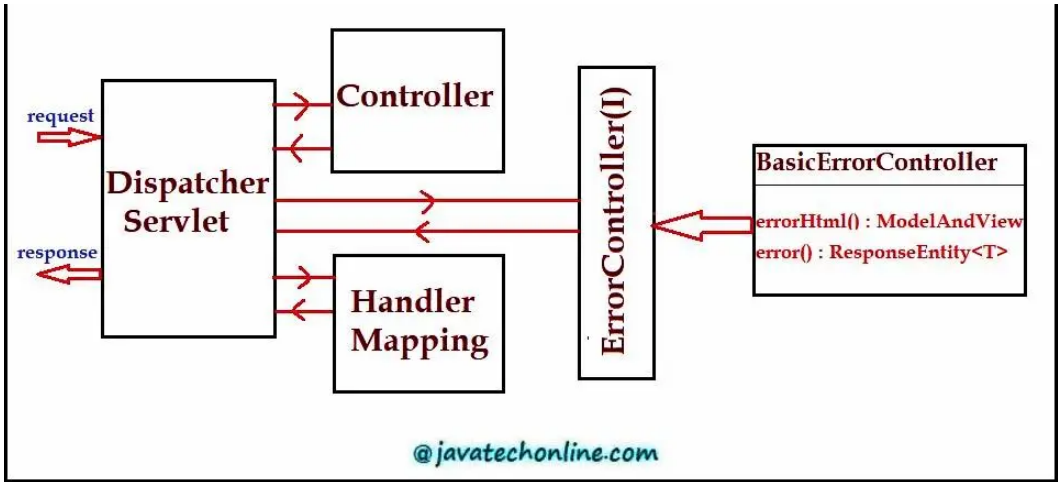
To illustrate the inbuilt exception handling in a Spring Boot Project, we will consider the most commonly used flows which are Spring Boot MVC and Spring Boot REST. Both flows work based on a Controller, either it is a normal controller or a RestController. Also in both the cases, any request first interacts with DispatcherServlet. Further, any request interacts with the DispatcherServlet before sending the response to the client, whether it is returning a successful result or failure after raising any exception. If it returns any failure result, DispatcherServlet calls a predefined functional interface ErrorController. In this case BasicErrorController (an implementation class of interface ErrorController) handles the request. BasicErrorController has below two methods to handle such type of requests.

♦ **errorHtml()** – This method is called when the request comes from a browser (MVC call)
♦ **error()** – This method is called when the request comes from non-browser medium such as postman tool/client app/other app (REST call)

BasicErrorController shows error/exception because of below default path at @RequestMapping annotation.
♦ @RequestMapping("${server.error.path:${error.path:/error}}")

Finally a default whitelabel Error page with some status code appears on the screen in case of MVC call. Similarly, if it is REST call, you will receive an error message as a JSON response in the form of default error attributes like status, error, message, timestamp etc. Further If we want to display a meaningful custom page on the screen, then what will we do? Check it in the next section.

TOP



How to display meaningful Custom error/exception pages?

Instead of getting Whitelabel Error Page, we can just create 'error.html' page under src/main/resources /templates folder as below.

error.html

```
<!DOCTYPE html>
<html xmlns:th="https://www.thymeleaf.org/">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.
</head>
<body>
<div class="container">
<h3>SOMTHING WENT WRONG! PLZ CONTACT MAINTENANCE TEAM</h3>
<table class="table table-bordered">
    <tr>
        <td>DATE</td>
        <td th:text="${timestamp}"></td>
    </tr>

    <tr>
        <td>status</td>
        <td th:text="${status}"></td>
    </tr>

```

TOP

```
22      <tr>
23          <td>error</td>
24          <td th:text="${error}"></td>
25      </tr>
26
27      <tr>
28          <td>path</td>
29          <td th:text="${path}"></td>
30      </tr>
31
32      <tr>
33          <td>Trace</td>
34          <td th:text="${trace}"></td>
35      </tr>
36  </table>
37  </div>
38  </body>
39  </html>
```

On creating our custom error.html page we can get a meaningful error information separated by default error attributes.

error.html will handle all the errors with status 4xx & 5xx. For example 400, 401 , 403 , 404 , .. 500 , 501 etc. only error.html page will be executed.

What is Http Response Status code?

In a client-server request-response model Http Response Status code is a three digit code which is provided by a server to the client. This Status code indicates whether the requested action is successful or any error occurred during the processing. For example, when a search engine or website visitor makes a request to a web server, a three digit HTTP Response Status Code is returned. This cod

TOP

indicates what is about to happen. A response code of 200 means “OK” which indicates that the request has succeeded. Similarly, other status codes have some meaning.

Generally Http Status Codes come under below five categories.

Informational responses (**100–199**)
Successful responses (**200–299**)
Redirects (**300–399**)
Client errors (**400–499**) : denotes Errors in Java
Server errors (**500–599**) : denotes Exceptions in Java

The below status codes are defined by section 10 of RFC 2616. You can find an updated specification in RFC 7231.

♥ If you receive a response that is not in this list, it is a non-standard response, possibly custom to the server’s software. If error status code is of type 4xx or 5xx, aforementioned error.html will handle that request.

What are some commonly used Http Response Status codes?

OK : 200
CREATED : 201
NO_CONTENT : 204
MULTIPLE_CHOICES : 300
NOT_MODIFIED : 304
PERMANENT_REDIRECT : 308
BAD_REQUEST : 400
UNAUTHORIZED : 401
PAYMENT_REQUIRED : 402
FORBIDDEN : 403
NOT_FOUND : 404
METHOD_NOT_ALLOWED : 405
PROXY_AUTHENTICATION_REQUIRED : 407
REQUEST_TIMEOUT : 408
CONFLICT : 409
URI_TOO_LONG : 414
UNSUPPORTED_MEDIA_TYPE : 415
TOO_MANY_REQUESTS : 429
REQUEST_HEADER_FIELDS_TOO_LARGE : 431

TOP

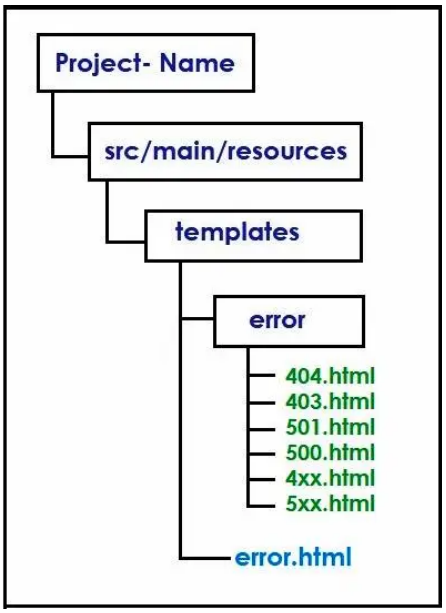
INTERNAL_SERVER_ERROR : 500
NOT_IMPLEMENTED : 501
BAD_GATEWAY : 502
SERVICE_UNAVAILABLE : 503
GATEWAY_TIMEOUT : 504
HTTP_VERSION_NOT_SUPPORTED : 505
NETWORK_AUTHENTICATION_REQUIRED : 511

How to create a specific error page for a particular Http Response Status code?

Aforementioned error.html will handle all the request irrespective of a particular Http Response Status Code. It works as a generic error page. Suppose we want to display a separate error page in case of status code-404. Additionally, in case of status code-500 we want to display another distinct error page. Then how will we create these pages?

Step#1: Create a sub-folder 'error' under 'src/main/resources/templates' folder.
Step#2: Create an html page with the same name as error status code in 'StatusCode.html' format. For example, for status code 404 create '404.html' and place it under 'error' sub-folder.

Your folder structure should look like below screen.



If no page with specific status code found, it will display error.html by default. If the controller is throwing any exception, then by default Http Status code will be 500 which is 'INTERNAL SERVER ERROR'. To change error code to any other status code, apply @ResponseStatus(code= HttpStatus.StatusCodeName) on top of your custom exception.

TOP

Even we can create a single page for a error status code 400-499 as '4xx.html'. Similarly, for 500-599 as '5xx.html'.

How to create a custom exception and map it to an error page with specific Http Response Status code?

To illustrate this requirement, let's assume that we have an Invoice Processing application. Further in this application we have one controller with a getInvoice() method to serve the request as given below. We will take some random integers in if condition. Therefore this method will sometimes provide successful response. But sometimes it will also throw an exception. We will call it InvoiceNotFoundException and create it as a custom exception. Additionally we want a custom 404.html display this exception.

InvoiceController.java

```
1  package com.dev.spring.exception.controller;
2
3  import java.util.Random;
4
5  import org.springframework.stereotype.Controller;
6  import org.springframework.web.bind.annotation.GetMapping;
7
8  import com.dev.spring.exception.custom.ImplementationNotFoundException;
9  import com.dev.spring.exception.custom.InvoiceNotFoundException;
10
11 @Controller
12 public class InvoiceController {
13
14
15     @GetMapping("/getInvoice")
16     public String getInvoice() {
17         if(new Random().nextInt(10)>5)
18             throw new InvoiceNotFoundException("Invoice Not Found!");
19         return "showInvoice";
20     }
21 }
```

Creating Custom Exception

Create a class InvoiceNotFoundException which will extend java.lang.RuntimeException. Then a

TOP

@ResponseStatus(code= HttpStatus.NOT_FOUND) on top of it as below.

InvoiceNotFoundException.java

```java
package com.dev.spring.exception.custom;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(code= HttpStatus.NOT_FOUND)
public class InvoiceNotFoundException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public InvoiceNotFoundException() {
        super();
    }

    public InvoiceNotFoundException(String message) {
        super(message);
    }

}
```

Here httpStatus.NOT_FOUND indicates error status code 404. Therefore we will create a custom 404.html page so that it can be called when InvoiceNotFoundException occurs.

Creating Custom Error Page (404.html)

Create 404.html as below and place it inside 'src/main/resources/templates/error' folder as aforementioned.

404.html

```html
<!DOCTYPE html>
<html xmlns:th="https://www.thymeleaf.org/">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstr
```

```
</head>
<body>
<div class="container">
<h4>SOME RESOURCE NOT FOUND! PLEASE CONTACT MAINTENANCE TEAM</h4>
<table class="table table-bordered">
    <tr>
        <td>DATE</td>
        <td th:text="${timestamp}"></td>
    </tr>

    <tr>
        <td>STATUS</td>
        <td th:text="${status}"></td>
    </tr>

    <tr>
        <td>ERROR</td>
        <td th:text="${error}"></td>
    </tr>

    <tr>
        <td>PATH</td>
        <td th:text="${path}"></td>
    </tr>

</table>
</div>
</body>
</html>
```

Whenever exception occurs, 404.html will be called and it will display the error description as in below meaningful format.

TOP

How to write custom Error Controller in Spring Boot?

Instead of utilizing the ErrorController functionalities already provided by Spring Boot, we can implement our own Error Controller. Since already provided ErrorController.class is an interface, we can create an implementer class of it to make it possible. If we define implementation class of ErrorController(I) then Spring Boot selects our custom class on priority to show up the errors/exceptions. Further in this case error.hml, 4xx.html, 5xx.html or any other error page will not work at all. Spring Boot by default provides some error attributes like timestamp, status, message, path, exception, trace etc. We can use them in our custom controller via @Autowired. Now to read values of attributes for current request, use below lines of code.

```
ServletWebRequest swr = new ServletWebRequest(request);
Map<String, Object> errors= errorAttribues.getErrorAttributes(swr,true);
```

Custom Error Controller to Get output in HTML format

To illustrate a custom error controller which will display the error output in HTML format, below is the code.

CustomErrorController.java

```
1  package com.dev.spring.exception.controller;
2
3  import java.util.Map;
4
5  import javax.servlet.http.HttpServletRequest;
6
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.boot.web.servlet.error.ErrorAttributes;
9  import org.springframework.boot.web.servlet.error.ErrorController;
10 import org.springframework.stereotype.Controller;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.ResponseBody;
13 import org.springframework.web.context.request.ServletWebRequest;
14
15 @Controller
16 public class CustomErrorController implements ErrorController {
17
18     @Autowired
19     private ErrorAttributes errorAttributes;
20
21     @Override
```

TOP

```
22      public String getErrorPath() {
23          return "/error";          //mandatory path
24      }
25
26      @RequestMapping("/error")     //mandatory mapping
27      public @ResponseBody String handleError(HttpServletRequest req) {
28
29          ServletWebRequest servletWebRequest = new ServletWebRequest(req);
30
31          @SuppressWarnings("deprecation")
32          Map<String, Object> errors = errorAttributes.getErrorAttributes(servle
33
34          StringBuilder builder = new StringBuilder();
35          builder.append("<html><body>");
36          builder.append("<h2>ERROR SUMMARY</h2>");
37
38          builder.append("<table border='1.5'>");
39          errors.forEach((key, value) -> {
40              builder.append("<tr>").append("<td>").append(key).append("</td>").a
41                      .append("</tr>");
42          });
43          builder.append("</table>");
44          builder.append("</body></html>");
45          return builder.toString();
46      }
47
48  }
```

Output

Below is the Output format.

TOP

Custom Error Controller to Get output in JSON format

To illustrate a custom error controller which will display the error output in JSON format, below is the code.

♥ Remember that Default output of @ResponseBody in Spring Boot is either String or JSON format. Any non-string return type in method will provide output in JSON format. Therefore, make sure that your controller method return type is a non-string (Lis/Set/Map)

Also apply @ResponseBody over method.

CustomErrorControllerWithJSONResponse.java

```
1  package com.dev.spring.exception.controller;
2
3  import java.util.Map;
4
5  import javax.servlet.http.HttpServletRequest;
6
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.boot.web.servlet.error.ErrorAttributes;
9  import org.springframework.boot.web.servlet.error.ErrorController;
10 import org.springframework.stereotype.Controller;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.ResponseBody;
13 import org.springframework.web.context.request.ServletWebRequest;
14
15 @Controller
16 public class CustomErrorControllerWithJSONResponse implements ErrorController
17
18     @Autowired
19     private ErrorAttributes errorAttributes;
20
21     @RequestMapping("/error")
22     public @ResponseBody Map<String, Object> handleError(HttpServletRequest re
23     {
24         ServletWebRequest webRequest = new ServletWebRequest(req);
```

TOP

```
25          @SuppressWarnings("deprecation")
26          Map<String, Object> errors = errorAttributes.getErrorAttributes(webReq
27
28          return errors;
29      }
30
31      @Override
32      public String getErrorPath() {
33          // TODO Auto-generated method stub
34          return null;
35      }
36
37  }
```

Output

Below is the Output format.

♥ To see JSON output use Firefox, JSON View extension with Google Chrome or even Postman tool.

How to add Custom Error Attributes in Custom Error Controller ?

Sometimes we may get a requirement to display some additional attribute with the error details. In that case we can just add the value of attribute in key-value format in the Map itself as shown below. Here we are only providing method code. Subsequently rest part of Custom Controller code will be as it is as in previous code example.

TOP

Adding Custom Error Attribute to Custom Error Controller

```
@RequestMapping("/error")
    public @ResponseBody Map<String, Object> handleError(HttpServletRequest re
    {
        ServletWebRequest webRequest = new ServletWebRequest(req);
        @SuppressWarnings("deprecation")
        Map<String, Object> errors = errorAttributes.getErrorAttributes(webReq
        errors.put("Error Output Format", "JSON");
        return errors;
    }
```

Output

Below is the output.

How does predefined ErrorController handles exception raised by a REST call by default ?

To illustrate this, let's create a Spring Boot Starter project which will implement a simple REST call step by step. Consider an Invoice Processing use case where we will have a RestController as 'InvoiceRestController' with a method as getInvoice(). In addition we will have a model class as 'Invoice' and a Custom exception class as 'InvoiceNotFoundException'.

Step#1 : Create a Spring Boot Starter project in STS(Spring Tool Suite)

TOP

While creating Starter Project select 'Spring Web', 'Lombok' and 'Spring Boot DevTools' as starter project dependencies. Even If you don't know how to create Spring Boot Starter Project, Kindly visit Internal Link. Also, if you want to know more about Lombok, then visit a separate article on 'Lombok'.

Step#2 : Create Model class as Invoice.java

Invoice.java

```
1  package com.dev.spring.exception.entity;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Data;
5  import lombok.NoArgsConstructor;
6
7  @Data
8  @NoArgsConstructor
9  @AllArgsConstructor
10 public class Invoice {
11
12     private Integer id;
13     private String name;
14     private Double amount;
15     private String number;
16 }
```

Step#3 : Create Controller class as InvoiceRestController.java

Here we are intentionally throwing a InvoiceNotFoundException if value of invoice id is 24.

InvoiceRestController.java

```
1  package com.dev.spring.exception.controller;
2
3  import org.springframework.http.ResponseEntity;
4  import org.springframework.web.bind.annotation.GetMapping;
5  import org.springframework.web.bind.annotation.PathVariable;
6  import org.springframework.web.bind.annotation.RestController;
7
8  import com.dev.spring.exception.custom.InvoiceNotFoundException;
9  import com.dev.spring.exception.entity.Invoice;
```

TOP

```
10
11  @RestController
12  public class InvoiceRestController {
13
14      @GetMapping("/find/{id}")
15      public ResponseEntity<Invoice> getInvoice(@PathVariable Integer id){
16
17          if(id ==24) {
18              throw new InvoiceNotFoundException("Invoice with id: " +id +" does
19          }
20          return ResponseEntity.ok(new Invoice(id,"INV001",2378.75,"CRTQW224"));
21      }
22  }
```

Step#4 : Create Custom Exception class as InvoiceNotFoundException.java

InvoiceNotFoundException.java

```
1  package com.dev.spring.exception.custom;
2
3
4  public class InvoiceNotFoundException extends RuntimeException {
5
6      private static final long serialVersionUID = 1L;
7
8      public InvoiceNotFoundException() {
9          super();
10     }
11
12     public InvoiceNotFoundException(String message) {
13         super(message);
14     }
15
16 }
```

Testing the Exception

Enter URL 'http://localhost:8080/find/24' using Postman tool, select 'GET' method and then clicl 'Send' button. Consequently you will see output something like below. Here you can choose an

TOP

browser of your choice to hit the URL.


```
{
"timestamp": "2020-12-29T19:32:29.056+00:00",
"status": 500,
"error": "Internal Server Error",
"trace": "com.dev.spring.exception.custom.InvoiceNotFoundException: Invoice with id: 2
"message": "Invoice with id: 24 does not exist!",
"path": "/find/24"
}
```


Conclusion

Although from the above output it is clear that by default Spring Boot's predefined ErrorController provides Status code as '505' and error as 'Internal Server Error' on any type of exception. Further we can customize the status code & error as per our wish.

How can we customize error status code & error details attribute?

Further to customize error status code & error we can just apply annotation @ResponseStatus(code= HttpStatus.StatusName) on top of custom controller itself as below.

InvoiceNotFoundException.java

```
1  package com.dev.spring.exception.custom;
2
3  import org.springframework.http.HttpStatus;
4  import org.springframework.web.bind.annotation.ResponseStatus;
5
6  @ResponseStatus(code= HttpStatus.NOT_FOUND)
7  public class InvoiceNotFoundException extends RuntimeException {
8
9      private static final long serialVersionUID = 1L;
10
11     public InvoiceNotFoundException() {
12         super();
13     }
14
15     public InvoiceNotFoundException(String message) {
```

TOP

```
16          super(message);
17      }
18
19  }
```

Customized Output

```
{
"timestamp": "2020-12-29T20:17:21.207+00:00",
"status": 404,
"error": "Not Found",
"trace": "com.dev.spring.exception.custom.InvoiceNotFoundException: Invoice with id: 2
"message": "Invoice with id: 24 does not exist!",
"path": "/find/24"
}
```

How can we customize all error attributes ?

To achieve this we will create one new Exception Handler class by using below annotations accordingly.

@ControllerAdvice : At Handler class level
@ExceptionHandler : At method level
@ResponseBody : At method level
♦ Although we can optionally use @RestControllerAdvice at class level. Further in that case we don't need to apply @ResponseBody at method level. In order to get more details on these annotations, kindly visit Annotations on Spring Boot Errors & Exceptions.

Step#1 : Write a new model class as 'ErrorType.java'

In order to customize all attributes, we will create a model class as 'ErrorType.java' and define all desired attributes as the fields. We will also need object of this class in our handler class.

ErrorType.java

```
1  package com.dev.spring.exception.entity;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Data;
5  import lombok.NoArgsConstructor;
6
```

TOP

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorType {

    private String message;
    private String code;
    private String error;
    private String staus;
}
```

Step#2 : Write a new handler class as 'InvoiceExceptionhandler.java'

We have to write a handler class and apply aforementioned annotations as below.

InvoiceExceptionHandler.java

```
package com.dev.spring.exception.handler;

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

import com.dev.spring.exception.custom.InvoiceNotFoundException;
import com.dev.spring.exception.entity.ErrorType;

//@RestControllerAdvice
@ControllerAdvice
public class InvoiceExceptionHandler {

    @ExceptionHandler(InvoiceNotFoundException.class)
    @ResponseBody
    public ResponseEntity<ErrorType> handleInvoiceNotFoundException(
            InvoiceNotFoundException ine){

        return new ResponseEntity<ErrorType>(
                new ErrorType(
```

TOP

```
                                ine.getMessage(),
                                "INVOICE_NOT_FOUND",
                                "DATA NOT FOUND FOR REQUESTED ID",
                                "406"),
                        HttpStatus.NOT_ACCEPTABLE);
        }
}
```

Step#3 : Write a new custom exception class and Rest Controller

You can use InvoiceNotFoundException.java from previous section. Additionally, make sure that @ResponseStatus is not applied on top of it. Similarly, use InvoiceRestController from the previous section itself.

Testing the customized error attributes

Enter URL 'http://localhost:8080/find/24' using Postman tool, select 'GET' method and then click on 'Send' button. Consequently you will see output something like below. Here you can choose any browser of your choice to hit the URL.

Summary

After going through all the theoretical & examples part of 'How to handle Exceptions & Errors in Spring Boot?', finally, we are able to implement Exceptions/Errors handling in a Spring Boot project. Of course, In this article we have thoroughly learned about the Spring Boot Exception handling. Similarly, we expect from you to further extend these examples and implement them in your project accordingly. You can also check other details on Exception Handling in Spring MVC from spring.io. In addition, If there is any update in future, we will also update the article accordingly. Moreover, Feel free to provide your comments in comments section.

TOP