How to handle custom exceptions in Spring Boot web application and Spring Boot RestAPI webapplication.

① throw → to create exception object
throws → to ignore the exception
→ when e

② @ExceptionHandler → only handles exception getting raised from the controller where it is defined
→ It will not handle exceptions getting raised from other controllers.
@ControllAdvice Annotation is solved the above problem.

③ → @ControllerAdvice annotation is used to define @ExceptionHandler, @InitBinder, and @ModelAttribute methods that apply to all @RequestMapping methods.

---

3 Approaches in Spring MVC having exception handling
① controlled Based handling
② Global Exception Handler
③ Handler Exception Resolver

→ Dispatcher servlet acts as a front controller in spring MVC. will send a request it will received by the dispatcher servlet it will talk to the handler mapping to identify the controller, then dispatcher servlet will call the controller method → control to process the request and given response / return model and view object
→ Model ~~with Controller data~~ will contain data.
→ View is a logical view name → with that view dispatcher servlet will talk to view resolver that will specify the physical location of the view → with the model data and view location dispather servlet will talk to view component which

will render model data on the view component and will return response that response will be dispatched to end user. (This is spring MVC application.

Spring MVC Based Application Execution flow.

→ End user / client send a request from presentation layer (JSP) → come to controller → controller can talk to service layer → service layer will talk to DAO layer → DAO layer will talk to data base.

4 layers
1. Presentation layer
2. controller layer
3. Service layer
4. persistance layer.

→ when the request is executed, here Exception can occur persistence layer (or) service, (or) controller layer. we need to handle that exceptions and display meaningful exception end user.

→ we wont display any exception stack trace and we need handle that exception and we need to display meaningful message to the end user.

---

① controller Based Handling.

→ we can write a method by using @ExceptionHandler

```
@ExceptionHandler(value = WebAppException.class)
public String handle(Model model)
```

(value → exception type of the exception. (user defined exception))

→ when this exception occurs in a controller then the method will be executed which is setting some data to the model scope and which is returning logical view name.

@ExceptionHandler → method level annotation of controller class. ②

```
@ExceptionHandler(value = WebAppException.class)   (User defind Exception class.)
public String handle(Model model)                  (Ex: we can write for null pointer or ArithmeticException)
{
String errMsgDesc = "Problem is to process request, please try some time";

model.addAttribute("errMsg", errMsgDesc);
                            key      value

return "error";
}
```

⇒ when the exception occurs in the controller, the above method is going to handle that.

⇒ If we write this a method inside a controller then it is called controller Based Handling. This is applicable for only that particular controller.

⇒ In project so many controllers in a project, so this is the same / common code of all the controllers then it is called boiler plate code.

So How to handle the exceptions globaley in our Spring MVC ~~class~~ application.

so GlobalExceptionHandler comes in to picture.

→ If we want to handle Exceptions in all the controllers then we need to go for Global Exception Handler.

→ so How to create a Global Exception handler → using annotation provided by a spring → @ControllerAdvice.

@Controller

@ControllerAdvice.

```
public class AppExceptionHandla                    → Customized exception class.
{
    @ExceptionHandler(value = WebAppException.class)
    public String handle(Model model)
    {
        String errMsgDesc = "Problem in processing request
                             please try after sometime";
        model.AddAttribute("errMsg", errMsgDesc); // setting data to model
        return "error";
    }                    → logical view name
}
```

→ Inside a project we will create a one handler class using @ControllerAdvice annotations.
(class level)

→ Inside a class we are writing a method. which is used to handle WebAppException in the project.

→ If any controller webAppException occurs then this method is responsible to handle that.
In that method setting the data to the model obj and return logical view name.

⇒ when ever webAppException occurs in any controller this error page will be displayed to the end user, in that page we will write some meaningful message to convey the user..

→ only ~~single~~ controller based handling (@ExceptionHandler) specific to particular controller ③

→ Global Exception Handler used for All the controllers in the project.

like this we can handle exception handling in spring mvc.

Exception Handling in MVC Flow

Spring Boot with MVC

views are located in folder

welcomeController

com.demo → base package of boot project of Application.

application.properties

spring.mvc.view.prefix = /views/

spring.mvc.view.suffix = .jsp

server.port = 7071

→ ~~package~~ package com.dem.controller

public class welcomeController {

@RequestMapping(value = "/welcome")
public String welcome(Model model)
{ // storing the data in model scope
model.addAttribute("msg", "welcome to ASKOLT school")

key (~~of EVA~~ ~~of jsp~~) → value should display in jsp

(key-value) → data adding.

return "welcome"

↳ logical view name.

}

~~}~~

~~application~~.

welcom.jsp

<body> ${msg}

</body>

${msg} → is the key of msg using this we can access the data

~~@ Enable Dispatcher~~

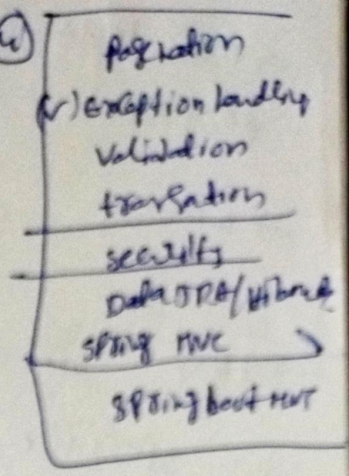
→ Request is going to controller → controller returning view
→ in that view we are accessing the models code
by using expression language.
→ let as thing that, we are getting the exception
in that method.

```
String name = null;
name.length();        // we can get null pointer exception.
return welcome;
```

→ we are added dev tools dependency, application got relaoded again we see the response.
earier success
Now 500 → Internal server.
→ The end user when we try to give request the exception occur and user can not understand it is not at all recomended stacktrace like this so we need handle the null pointer exception here we need display meaningful message to the user.

@Controller

→ class WelcomeController {



@ExceptionHandler

public String handleNullPointException(Model model)

{

model.addAttribute("errMsg", "some Problem occured, Please Try After Sometime.!");

return "error";

}

}

when ever Null pointer exception occur in this controller class the above method is responsible to handle that. we are storing the data in a model scope and return error.jsp page.

→ when ever null pointer exception occur in this method the above method is the handler binded by that exception so the handleNullPointerException will be executed which is returning logical view name is error.

→ we create error.jsp

```
<body>
    ${errMsg}
</body>
```

(errMsg ← key)

→ Dev tools is the project reloaded → clean the project.

→ we can display errMsg.

→ here we are writing exception handler method in specific controller. this is called controllerBased exception handler, this is applicable to only the current controller.

→ so that how to handle the all the controllers globally? (or)

controllerBasedExceptionHandler → If we handling exception specific to a controller is called.

How to used it.

→ writing exceptionhandler method and annotate that method into @ExceptionHandler.

→ Now Global Exception Handler in MVC

→ we need to create GlobalExceptionHandler which will handle the exceptions all the controller.

→ write a one more controller.

@Controller

```
public class DateController {
{
    @RequestMapping("/date")
    public String displayDate(Model model)
    {
        model.addAttribute("dateMsg", "today date :" + new Date());
        return "dateDisplay";
    }
}
```

↳ LVN

dateDisplay.jsp

```
<body>
${dateMsg}
</body>
```

Here also we are getting.

```
String s = null;
s.length();
return "dateDisplay";
```

// Here we are getting NULL Pointer Exception

→ Both controller we get NULL Pointer Exception How we need Handle the Exception for both controller.

→ Let us create global Exception Handler.

package com.demo.Exceptions

class ExceptionHandler

{

}

```
Package com.demo.exceptions
@Controller
@ControllerAdvice
Public class APPExceptionHandler {

                                        handle exception name.
                                        ↓
    @ExceptionHandler (value = nullPointException.class)
    public String handleNullPointException(Model model) {

        model.addAttribute("errMsg", "some Problem occured.
                                please try Again After sometime..!");

        return "error";

    }
```

→ This is the global exception handler whenever null pointer exception occurs in our project/application then this method going to executed. this is called Global Exception handler in our project.

→ Both try to send Output

localhost:7071/dele , localhost:7071/welcome

→ Both classes are throughing the null pointer exception instead of writing the exception handling logic in the both controller writing in a global class i.e. APPExceptionHandler.

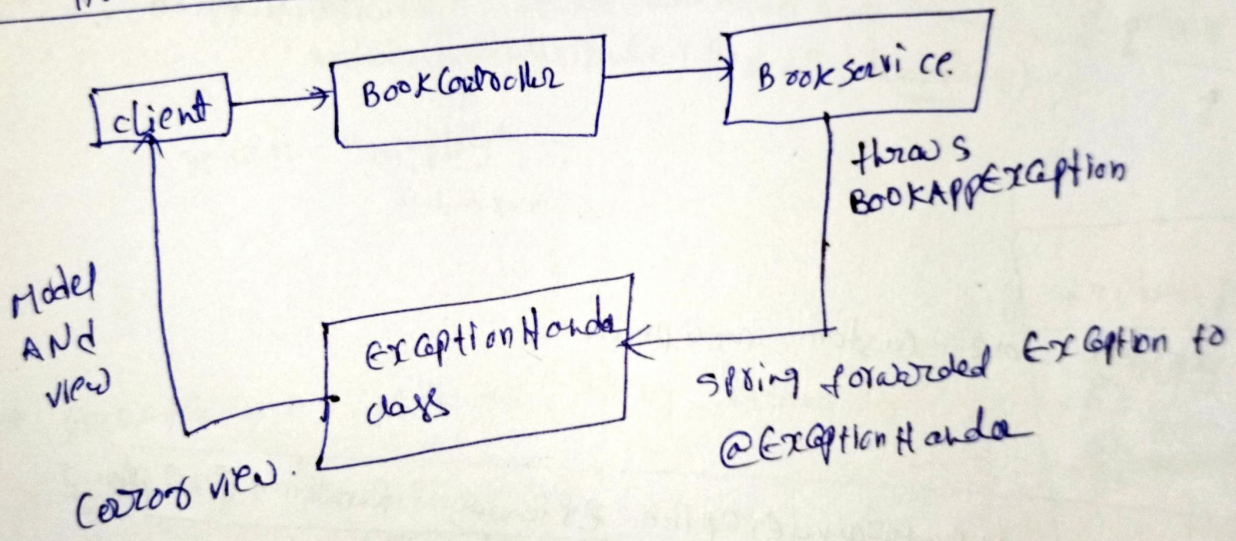
this is another mechanism which is recommended to handle.

→ TO create a exception handler method inside a controller and use @ExceptionHandler annotation only

one controller i.e. controller based exception.

→ If we want handle the same / common exceptions in all the controller then we need to create exception handler class using annotation @ControllerAdvice (this annotation privided by spring 3.2 onward).

→ How to create user defined exception in Java
How to handle. ~~that exception~~ in spring MVC.



BookService.java

```
@Service
public class BookServiceImpl implements BookService {

    @Override
    public Double findPriceById(String bookId) {
        if (bookId.equals("B101")) {
            return 450.00;
        }
        else { // throws custom exception if bookId not matched
            throw new NoBookFoundException("No Book found with GivenId");
        }
    }
```

Refactor create one custom exception:

```
public class NoBookFoundException extends RuntimeException {
    // constructor // override serialVersionUID
    public NoBookFoundException(String msg) {
        super(msg);
    }
}
```

How to Handle the user defined exception. ①

BookController.java

```
@Controller
public class BookController {
    @Autowired
    private BookService bookService;

    @RequestMapping(value="/findPriceById")
    public String getPriceById(@RequestParam("bookId")
            String bookId, Model model) {
        Double bookPrice = bookService.findPriceById(bookId);
        // this price we are add model scope.
        model.addAttribute("price", bookPrice);
        return "display";
    }
}
```

So where we need to get price from the service layer inject service object here

→ coming from request of request URL

→ RequestParam i.e. our query param by are of /findPriceById.

→ Deploy as project.

display.jsp

```
<body>
${price}
```

↳ Key as price a controller

use same key so that value displayed on jsp.

@Service → we can not give means auto scanning is done not done. it is component of a spring using @service.

localhost:7071/findPriceById?bookId=B101

→ @RequestParam("bookId")

↳ value of bookId.

BookPrice : 450.0.

let me give

localhost:7071/findByBookId?bookId=B102 // we can get page blank.

→ we can see the compl throwing the user defined exception.

→ so we create a global Exception handa.

@Controller
@ControllerAdvice.
public class APPExceptionHanda {

@ExceptionHanda(value = NoBookFoundException.class)
public String HandleNoBookFoundException() {

return "customError";

}

→ so when ever we encountered that no book found exception then my exception handler is having a method to handler that it is returning a custom error. the custom error we are displaying page that will contains meaning full image error in the web.

customError.jsp

we ta error methods
resources/static/images/error-image.png>

<body>
(How to Load image in jsp use image tag)
<img alt="error-image"
src="image/error-image.png">
</body>

↗ @RequestParam(value = "bookId")

localhost : 7071 /findByPriceid ? bookId = B102 ③

Something went wrong!

Go Home

} image display.

Flow

→ whenever request comes to controller → controller called service layer method → service layer method is throwing one exception i.e. user defined exception (i.e NoBookFoundException), → In service layer we dont have logic handle that exception, then JVM will terminate that service method execution → It will go to the caller of service method:

→ so controller called that service method, there. do we have a logic to handle the exception in controller? NO

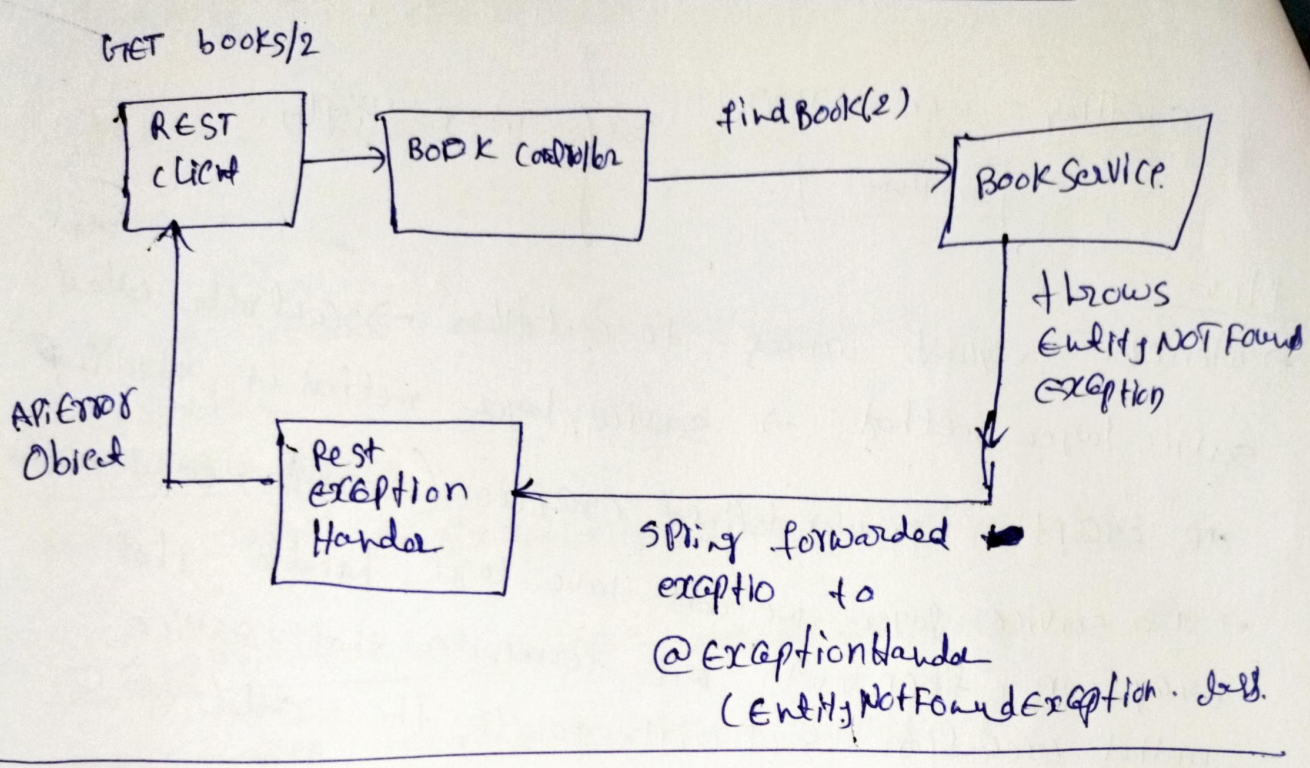
→ we have a global exception handler to handle that NO book found exception using handler method.

@ControllerAdvice and

@ExceptionHandler (value = "NoBookFoundException.class").

→ it is returning jsp message called customerror.jsp. it display one image meaning fal message one error.

How to handle Exceptions in RestAPI



→ Here we are going to use Rest client to send request to the rest controller, that controller will talk to Book service → If the record is available it will return to the response to the user.

→ If the Given Id is not matching then service class will going throw entity not found exception.. this is user defined exception.

→ when ever it throws this exception it will forward to the exceptionHandler in that Exception Handler we are going to convey a meaningful message to the rest client appropriate status code and status messages.

Here we are going to create a rest controller and a service class → and a user defined exception → and we are create global exception handler to handle the exception in our Rest API.

Package com.demo.model

(1)

```
@Data
class Product {
    private String pid;
    private String pname;
    private Double price;
```

(added lombok)
↳ to avoid boilerplate code we can use @Data (it gives getters/setters, tostring. and constructor (default, parameterized constructor) ↓ to set the data easily)

(2) create service class where we create data.

```
@Service
class ProductService {
}
```

↗ select

```
public Product findProductById(Integer pid)
{
    // return a product object based on id.
    if (pid = 101) {
        return new Product("101", "Keyboard", 800.00);
    }
    else { // throw custom exception.
        throw new NoProductFoundException("No Product Found");
    }
}
```

class NoProductFoundException extends RuntimeException {

public NoProductFoundException(String msg)

{

super(msg);

}

}

// create Rest controller., we are Rest API so we use Rest Controller.

@RestController

public class ProductController {

@Autowired.

private ProductService productService.

Use word → public Product getProductId(@RequestParam("pid") String pid)

{

return productService.findProductById(Integer.parseInt(pid));

↓ coming into request

↳ product object.

}

→ so this method should be binded to http designed method so we are using @GetMapping(value = "/getProduct")

↳ Instead of @RequestMapping in normal case

@GetMapping(value = "/getProduct", produces = {"application/json"})

→ so we are writing a rest controller which is ~~binded to~~ having a method and binded to get request which is excepting request param, based on request param it is talking to

service layer to return the product data in json format..

→ Deploying Project and try to hit URL.

→ localhost : 7071 / getProduct/(Pid = 101) → success.

{"Pid" : "101", "Pname" : "Keyboard", "Price" : 800.0}

→ Localhost : 7071 / getProduct / pid = 102 → no data in browser.

→ the custom exception displayed on console but what about rest client →

so we need to create a RestExceptionHandler class and we need to send the error object details to rest client.

→ we are creating global exception class with name restExceptionHandler.

@RestController → this is for the rest

@ControllerAdvice

public class RestExceptionHandler {

} // write a method to handle method.

@ExceptionHandler (value = NoProductFoundException.class)

public ResponseEntity handleNoProductFoundException()

{

return new ResponseEntity<> (HttpStatus.BAD_REQUEST);

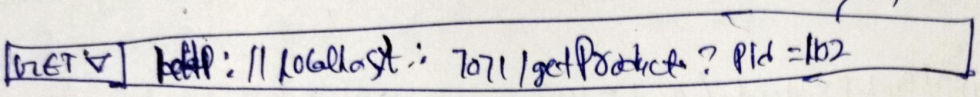
↳ (code)

}

Postman select the GET request in Postman

http://localhost:7071/getProduct?pid=101

```
{
    "pid" : "101"
    "pname" : "keyboard",
    "price" : 800
}
```



status : 400 Bad Request

↑ status code

→ we wont return any default message

So we try to return default message.

→ go to com.demoException package and create one class → ApiError.java.

ApiError.java

```
@Data
public class ApiError {
    Private Integer errorCode;
    Private String errorDesc;
    Private Date date;
}
```

setters / getters parameterised constructor to set data.

→ so this is a model class where we store our api error information like errorcode, desc, etc.

```
@RestController
@ControllerAdvice
public class RestExceptionHandler {

    @ExceptionHandler(value = NoProductFoundException.class)
    public ResponseEntity<APIError>
                handleNoProductFoundException() {

        APIError apiError = new APIError(400, "No product found",
                                         new Date());

        return new ResponseEntity<APIError>(error, HttpStatus.BAD_REQUEST);

    }
```

→ So we are handling exception for Spring Boot Rest API.

→ We have a Product controller where it rest client will send rest request based on product id

→ It is talking to service layer method with that id if the id = 101, then we are returning Product object with the data.

→ if it is not 101 then service layer method throwing the exception called no product found exception.

→ service layer method don't handle the exception → so JVM will kill this method execution and it will throw this NoProductFoundException to the caller.

→ the controller which called this method, controller don't have handler that exception logic.

→ For that exception we have RestExceptionHandler class and productNotFoundException here, we are taking model object to set the data → to Rest client knowing the information about bad request. and we are returning a ResponseEntity object with this api error object as response body and status as bad request.

→ So let us a send request a postmanl

http://localhost:7071/getProduct?pid=102.

{
"errorCode" : 400,
"errorDesc" : "No Product found",
"date" : "2021-.."
}

we are setting error object.

In Browser also

{"errorCode": 400, "errorDesc": "no product found", "date": "..."}

like this we can handle Rest API