Trabalhando com exceções no Spring Boot

Certified Developer
The Ultimate Tech Degree

DigitalHouse >

Índice

1 Anotação @ExceptionHandler

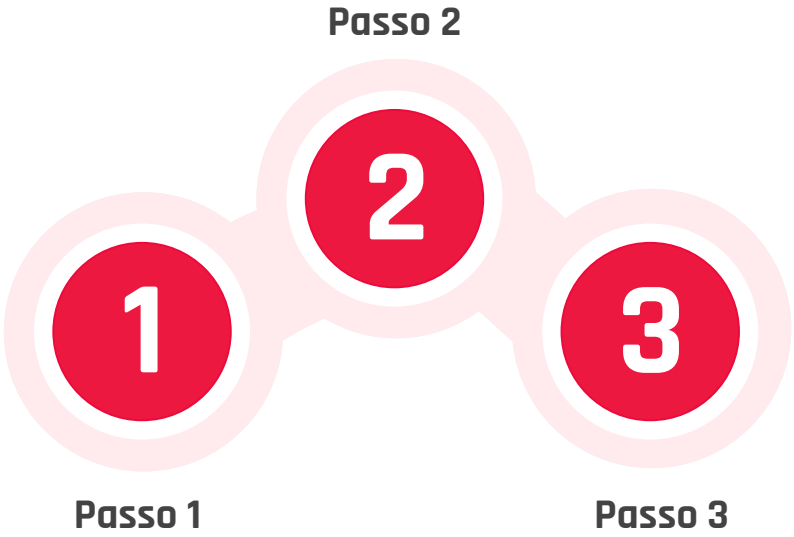
@ExceptionHandler

A primeira opção é usar esta anotação para trabalhar com exceções em relação ao controller, ou seja, dentro do nosso controller implementamos um método que será executado em caso de erro.

Por exemplo, quando temos um erro em que o recurso não foi localizado, aparece o erro 404:

```
{
      "timestamp": 1512713804164,
      "status": 404,
      "error": "Not Found",
      "message": "No message available",
      "path": "/some-url"
}
```

Podemos especificar o status da resposta para uma exceção específica, juntamente com a definição da exceção com a anotação @ResponseStatus. Vamos ver quais são os passos a seguir.



Passo 1

Se precisamos capturar uma exceção criada por nós, devemos primeiro criar nossa própria exceção.

```
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends Exception{

  public ResourceNotFoundException(String  message){
    super(message);
  }
}
```

Passo 2

A partir de um método do controller, poderia ser lançada uma exceção do tipo ResourceNotFoundException.

```
@RestController
public class MeuController{
  @RequestMapping(value = "/files/{id}")
   public String getFile(@PathVariable("id")long id){

    if(id <= 0){
      String messageError = "Nenhum arquivo foi encontrado com o id " + id;
      throw new ResourceNotFoundException(messageError);
    }else{
        ...
    }
  }
}
```

Passo 3

Então, capturamos a exceção com @ExceptionHandler.

```
@RestController
public class MeuController{

  @ExceptionHandler(Exception.class)
  private ResponseEntity<?> exception(ResourceNotFoundException ex, WebRequest request)
}
```

Este método captura as exceções do tipo Exception, ou seja, qualquer exceção Java. Com o ExceptionHandler, podemos capturar exceções do Java (Exception, NullPointerException, etc) e as exceções criadas por nós.

Então, capturamos a exceção com @ExceptionHandler.

```
log.error(ex);
return new ResponseEntity<>("Erro tratado pelo Exception Handler.",
HttpStatus.NOT_FOUND);
```

Captura de exceções do tipo ResourceNotFoundException que ocorrem dentro da controller MyController.

```
@ExceptionHandler(ResourceNotFoundException.class)
public ResponseEntity<?>
resourceNotFoundException(ResourceNotFoundException ex,
                          WebRequest request) {
}
```

É uma boa prática registrar erros com o Log4J:

```
log.error(ex);
return new ResponseEntity<>("Erro tratado pelo Exception Handler.",
HttpStatus.NOT_FOUND);
```

Essa abordagem tem uma grande desvantagem: o método anotado com **@ExceptionHandler** fica ativo apenas para esse controller em específico, não globalmente para toda a aplicação. Ou seja, teríamos que copiar e colar o código em todos os controllers para poder processar este erro. É claro que adicionar isso a cada controller o tornar inadequado para um mecanismo genéricos de tratamento de exceções.

Podemos contornar essa limitação fazendo com que todos os controllers herdem uma classe de controller base. No entanto, essa solução pode ser um problema para aplicativos onde, por qualquer motivo, isso não seja possível. Por exemplo, os controllers já podem ser estendidos de outra classe base, ou eles próprios podem não ser modificados diretamente.

2 Exceções globais @ControllerAdvice

@ControllerAdvice

A anotação @ControllerAdvice nos permite agrupar nossos vários @ExceptionHandlers dispersos em um único componente global de tratamento de erros.

Esse mecanismo é extremamente simples e muito flexível:

- Nos permite controle total sobre o corpo da resposta (response body), assim como o código de status (response status);
- Fornece o mapeamento de várias exceções para o mesmo método, para que possam serem tratados juntos;
- Também podemos usar o ResponseEntity para a resposta.

Vejamos um exemplo:

```
@ControllerAdvice
public class GlobalExceptionHandler {

      @ExceptionHandler(ResourceNotFoundException.class)
      public ResponseEntity<?> resourceNotFoundException(ResourceNotFoundException ex, WebRequest
request) {

            return new ResponseEntity<>(“Erro tratado pelo Exception Handler”, HttpStatus.NOT_FOUND);
      }

      @ExceptionHandler(Exception.class)
      public ResponseEntity<?> globleExcpetionHandler(Exception ex, WebRequest request) {

            return new ResponseEntity<>(“Erro tratado pelo Exception Handler”,
HttpStatus.INTERNAL_SERVER_ERROR);
      }
}
```

3 Exceções a nível de método

ResponseStatusException

No caso de precisar tratar um erro específico, e usar @ExceptionHandler na classe não será eficiente, podemos usar a classe **ResponseStatusException** em um método específico de um controller. Vejamos um exemplo:

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

      return persona;
      }

    catch (MyResourceNotFoundException ex){
       throw new ResourceNotFoundException(HttpStatus.NOT_FOUND, "Persona Not Found", ex);
    }
}
```

DigitalHouse>