Introdução ao Tratamento de Exceções

Prof. Julio Cesar Nardi.

Este tutorial apresenta uma introdução ao tratamento de exceções no Sprint Boot. Uma aplicação *web* de um Sistema de Vídeo Locadora é utilizada para exemplificação dos códigos-fontes. A API (*Application Programming Interface*) do *backend* desse sistema é acessada via Postman (uma aplicação *HTTP Client*) a fim de realizar os exemplos.

O Contexto

Considere uma classe *AtorResource.java*, a qual funciona como *@RestController*. Nesta classe há métodos que implementam o CRUD do cadastro de atores. Assim, nesta classe há o método *obterAtor*, o qual retorna um ator, dado seu Id, conforme apresentado abaixo.

```
@GetMapping (value="/locadora/resouces/atores/{id}")
public ResponseEntity<Ator> obterAtor(@PathVariable Long id){

      Ator aux = repositorioAtores.findById(id).get();

      return ResponseEntity.ok().body(aux);

}
```

Este método não tem qualquer tratamento de exceção, inclusive para o caso de haver uma solicitação de recuperação de ator cujo Id não existe no banco de dados. Essa será nossa situação exemplo para a implementação do tratamento de exceção, que poderá ser estendido para outras situações de exceção.

A Figura 1 exibe um cenário de acesso “com sucesso”, no qual um Id válido é passado na requisição HTTP e um Json contendo as informações do Ator correspondente é retornado. Já na Figura 2 exibe um caso de disparo de exceção, no qual o Id passado como parâmetro não corresponde a um Ator existente no banco de dados. Podemos observar que esta mensagem não está tratada e aparece conforme o container a dispara para o consumidor da API. Observe como, atualmente, são as informações da mensagem de exceção. Nosso propósito, então, é, em caso de exceção, tratar a mensagem e retorná-la de maneira mais amigável e clara para o consumidor da API.

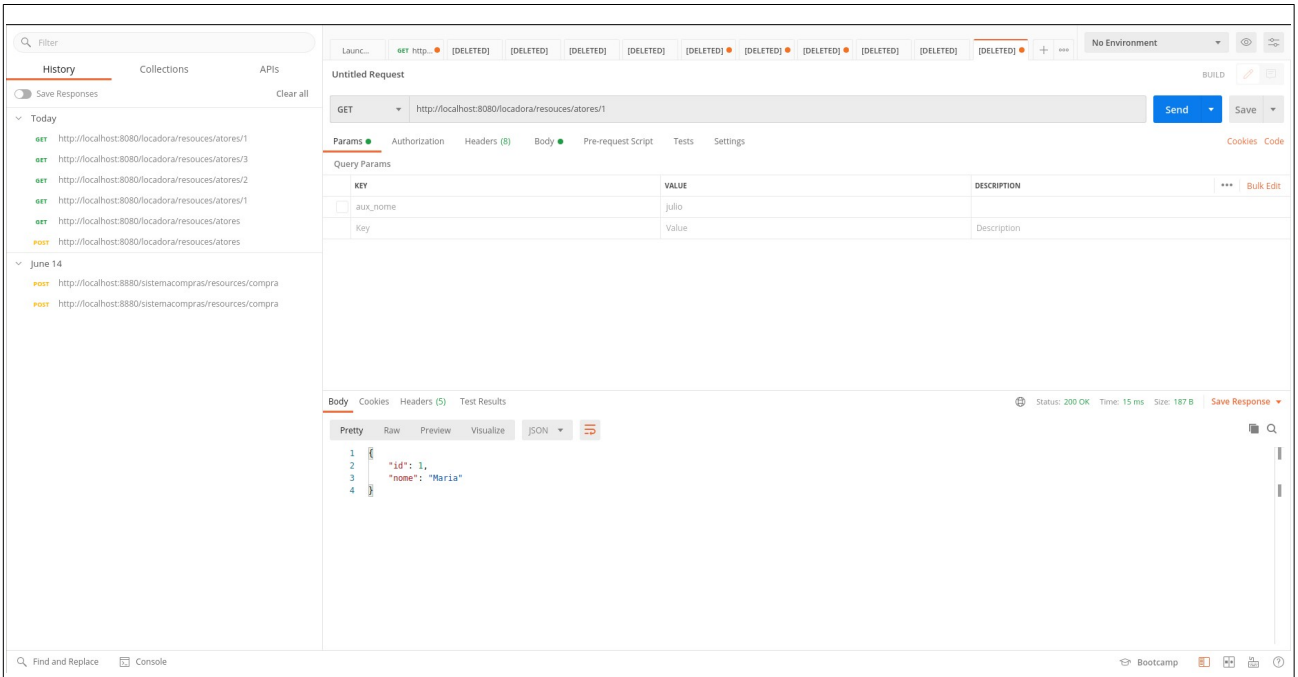

Figura 1: Cenário de acesso com sucesso.

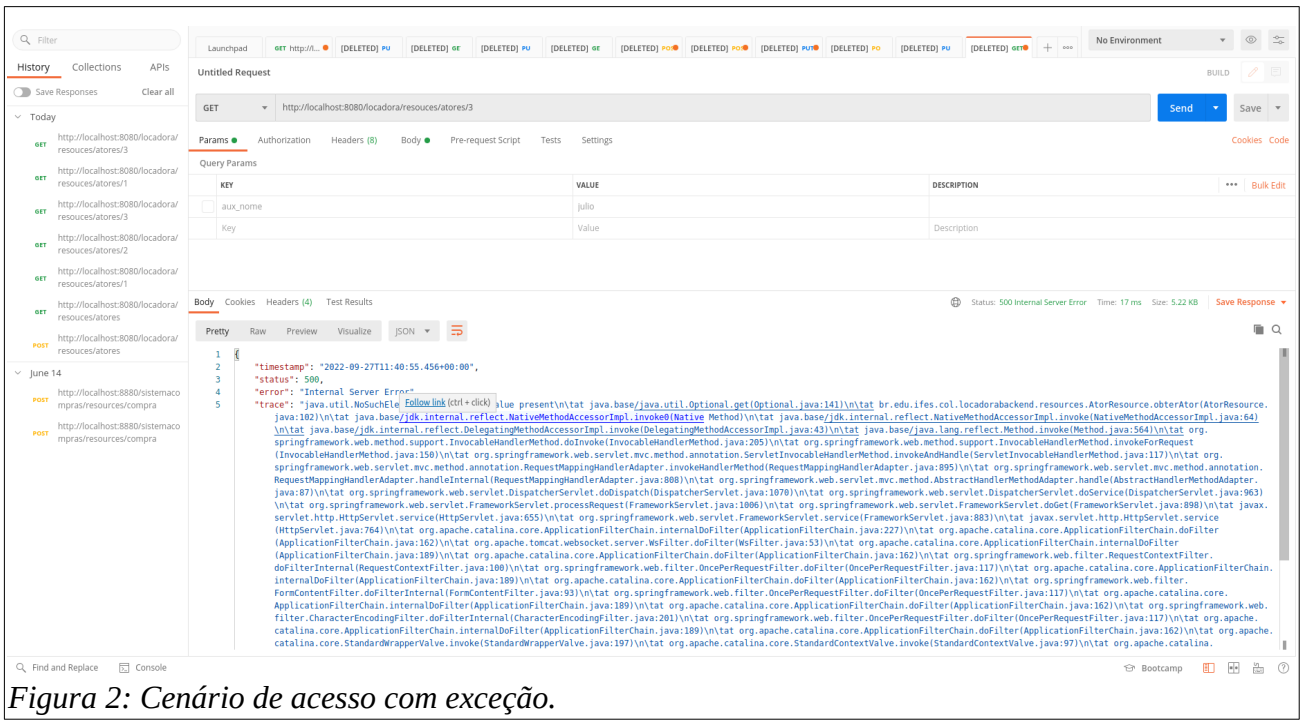

Figura 2: Cenário de acesso com exceção.

Criando Exceções Personalizadas

Para tratar as exceções de maneira particular, vamos criar classes de exceções personalizadas buscando tratar cada exceção/erro. Para tanto, foi criado um pacote no qual serão organizadas todas as exceções personalizadas criadas. Nesta aplicação o pacote é *br.edu.ifes.col.locadorabackend.exceptions*. Neste pacote, foi criada a classe *AtorNaoEncontradoException.java*, cujo código conte é apresentado na listagem abaixo.

```
public class AtorNaoEncontradoException extends RuntimeException{

	private static final long serialVersionUID = 1L;

	public AtorNaoEncontradoException (String msg) {

		super(msg);

	}

}
```

Em seguinda, vamos ajustar o método *obterAtor* da classe *AtorResource* a fim de este método, em caso de não ter encontrado um determinado ator, possa disparar nossa exceção personalizada. Assim, a linha com "*findById*" foi ajustada para.

```
@GetMapping (value="/locadora/resouces/atores/{id}")
public ResponseEntity<Ator> obterAtor(@PathVariable Long id){


	Ator aux = repositorioAtores.findById(id).orElseThrow(
		() -> new AtorNaoEncontradoException("Não foi encontrado ator com o
ID informado!"));

	return ResponseEntity.ok().body(aux);

	}
```

Perceba na Figura 3 que a mensagem já está personalizada.

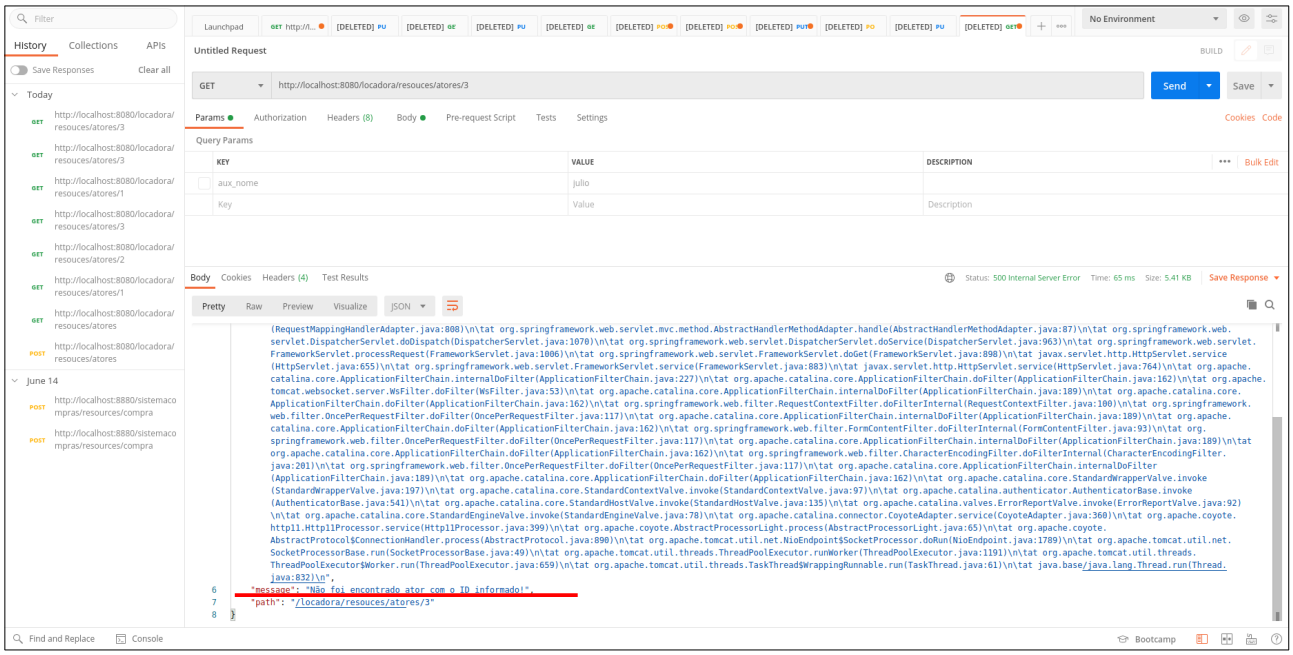

Figura 3: Exceção disparada já com o campo "message" tratado.

Criando Estrutura de Erro Padrão

O nosso próximo ajuste é fazer com que tanto o código de erro HTTP quanto demais informações retornadas ao consumidor da API sejam mais adequados para a exceção em questão. Por exemplo, ao invés de retornar simplesmente "*500 Internal Server Error*" (que de fato não é o código de erro correto para o caso do exemplo em questão), que seja retornado, por exemplo, "404 not found" (ou seja, o recurso/ator buscado no banco não existe). Para tanto, vamos criar uma classe padrão de erro com os mesmos campos geralmente retornados pelo container em caso de exceção. De posse dessa classe, poderemos personalizar seus valores de acordo com os erros da aplicação que vierem a ocorrer. A classe criada é *ErroPadrao.java*, cujo código está listado abaixo.

```
public class ErroPadrao {

	private Instant timestamp;
	private Integer status;
	private String error;
	private String message;
	private String path;

	public Instant getTimestamp() {
		return timestamp;
	}
	public void setTimestamp(Instant timestamp) {
		this.timestamp = timestamp;
	}
	public Integer getStatus() {
		return status;
	}
	public void setStatus(Integer status) {
		this.status = status;
	}
	public String getError() {
		return error;
	}
	public void setError(String error) {
		this.error = error;
	}
	public String getMessage() {
		return message;
	}

	...
```

Essa classe padrão de erro será, então, utilizada para troca de informações entre *backend* e consumidor da API, quando da ocorrência das exceções.

Criando o Tratador da Exceção (*Handler*)

Usaremos o elemento *Controller Advice* para interceptar a transação disparada e tratá-la em uma classe em separado a fim de não poluir o código do controlador *AtorResource*. Vamos criar, então, a classe *AtorResourceExceptionHandler.java,* cujo código-fonte está listado abaixo.

```java
@ControllerAdvice
public class AtorResourceExceptionHandler {

	@ExceptionHandler(AtorNaoEncontradoException.class)
	public ResponseEntity<ErroPadrao>
atorNaoEncontradoHandle(AtorNaoEncontradoException e, HttpServletRequest req){

		ErroPadrao err = new ErroPadrao();

		err.setError("Resource not found");
		err.setMessage(e.getMessage());
		err.setPath(req.getRequestURI());
		err.setStatus(HttpStatus.NOT_FOUND.value());
		err.setTimestamp(Instant.now());

		return ResponseEntity.status(HttpStatus.NOT_FOUND).body(err);

	}
}
```

Após implementar o método, vamos acessar, via Postman, a URL solicitando um ator cujo Id não consta no banco de dados a fim de testar o tratamento de exceções implementado. A Figura 4 exibe esse cenário, em que a mensagem é apresentada com todos os campos já personalizados, conforme informado no tratamanto de exceção.

Dessa forma, temos, então, nosso tratamento para essa exceção implementado e funcionando.

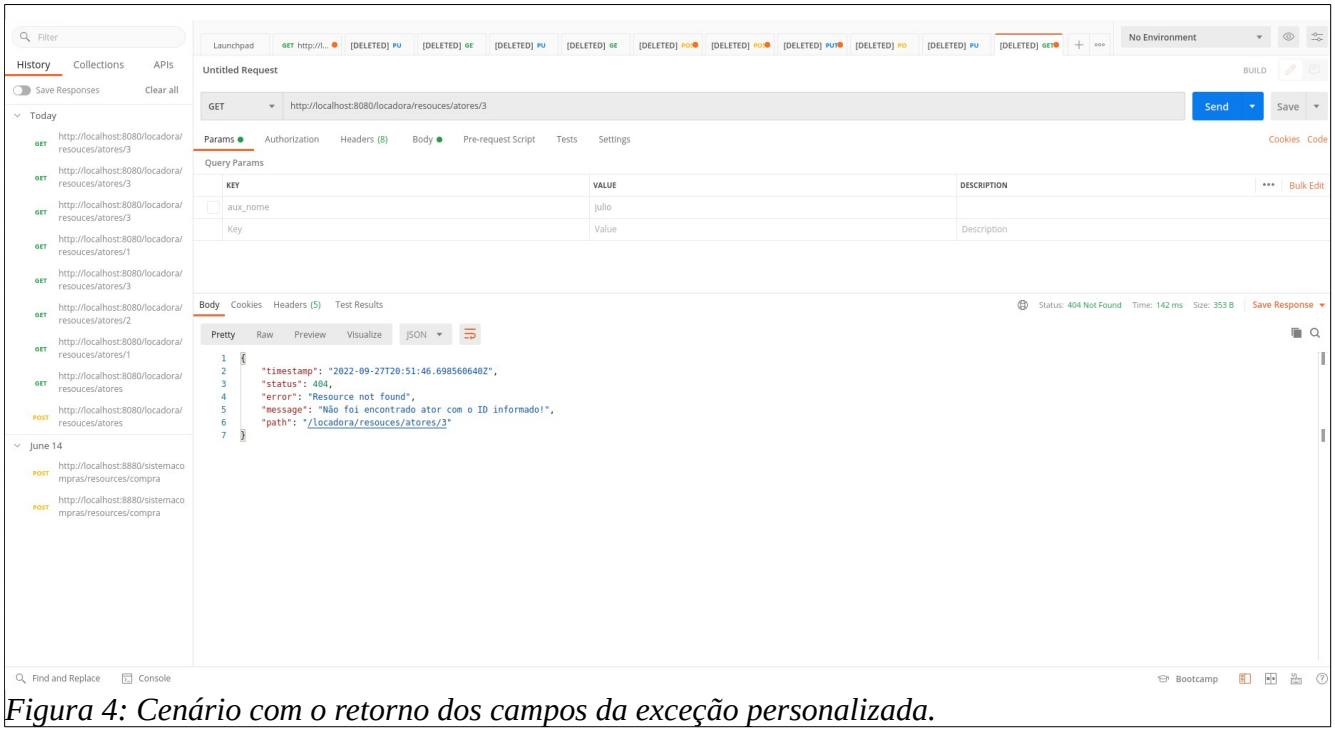

Figura 4: Cenário com o retorno dos campos da exceção personalizada.

Tutorial baseado em:
Padrão camadas e exceções em Java web Spring Boot – Aulão #12 (by Dev Superior) (https://www.youtube.com/watch?v=MAv7xgnSD-s)