(http://baeldung.com)

Spring ResponseStatusException

Última modificación: 4 de abril de 2018

por baeldung (http://www.baeldung.com/author/baeldung/)
(http://www.baeldung.com/author/baeldung/)

excepciones (http://www.baeldung.com/category/exceptions/)
Primavera (http://www.baeldung.com/category/spring/) +

1. Información general

En este tutorial rápido, analizaremos la nueva clase *ResponseStatusException* introducida en Spring 5. Esta clase admite la aplicación de código de estado HTTP a respuestas HTTP.

A RESTful application can communicate the success or failure of an HTTP request by **returning the right status code in the response to the client**. Simply put, an appropriate status code can help the client to identify problems that might have

occurred while the application was dealing with the request.

2. *ResponseStatus*

Before we delve into *ResponseStatusException*, let's quickly take a look at the *@ResponseStatus* annotation. This annotation was introduced in Spring 3 for applying HTTP Status code to an HTTP response.

We can use the *@ResponseStatus* annotation to set the status and reason in our HTTP response:

```
@ResponseStatus(code = HttpStatus.NOT_FOUND, reason = "Actor Not Found")
public class ActorNotFoundException extends Exception {
    // ...
}
```

If this exception is thrown while processing an HTTP request, then the response will include the HTTP status specified in this annotation.

One drawback of the *@ResponseStatus* approach is that it creates tight coupling with the exception. In our example, all exceptions of type *ActorNotFoundException* will generate the same error message and status code in the response.

3. *ResponseStatusException*

ResponseStatusException is a programmatic alternative to *@ResponseStatus* and is the base class for exceptions used for applying a status code to an HTTP response. It's a *RuntimeException* and hence not required to be explicitly added in a method signature.

Spring provides 3 constructors to generate *ResponseStatusException*:

```
ResponseStatusException(HttpStatus status)
ResponseStatusException(HttpStatus status, java.lang.String reason)
ResponseStatusException(
  HttpStatus status,
  java.lang.String reason,
  java.lang.Throwable cause
)
```

ResponseStatusException, constructor arguments:

- status – an HTTP status set to HTTP response
- reason – a message explaining the exception set to HTTP response
- cause – a *Throwable* cause of the *ResponseStatusException*

Note: in Spring, *HandlerExceptionResolver* intercepts and processes any exception raised and not handled by a Controller.

One of these handlers, *ResponseStatusExceptionResolver,* looks for any *ResponseStatusException* or uncaught exceptions annotated by *@ResponseStatus* and then extracts the HTTP Status code & reason and includes them in the HTTP response.

3.1. *ResponseStatusException* Benefits

ResponseStatusException usage has few benefits:

- Firstly, exceptions of the same type can be processed separately and different status codes can be set on the response. Reducing tight coupling
- Secondly, it avoids creation of unnecessary additional exception classes
- Finally, it provides more control over exception handling, as the exceptions can be created programmatically

4. Examples

4.1. Generate *ResponseStatusException*

Now, let's us see an example which generates a *ResponseStatusException* :

```
@GetMapping("/actor/{id}")
public String getActorName(@PathVariable("id") int id) {
    try {
        return actorService.getActor(id);
    } catch (ActorNotFoundException ex) {
        throw new ResponseStatusException(
          HttpStatus.NOT_FOUND, "Actor Not Found", ex);
    }
}
```

Spring Boot provides a default */error* mapping, returning a JSON response with HTTP status and the exception message.

Here's how the response looks:

```
$ curl -i -s -X GET http://localhost:8080/actor/8
HTTP/1.1 404
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Sun, 28 Jan 2018 19:48:10 GMT

{
    "timestamp": "2018-01-28T19:48:10.471+0000",
    "status": 404,
    "error": "Not Found",
    "message": "Actor Not Found",
    "path": "/actor/8"
}
```

4.2. Código de estado diferente: el mismo tipo de excepción

Ahora, veamos cómo un código de estado diferente se establece en respuesta HTTP cuando se plantea el mismo tipo de excepción:

```
@PutMapping("/actor/{id}/{name}")
public String updateActorName(
  @PathVariable("id") int id,
  @PathVariable("name") String name) {

    try {
        return actorService.updateActor(id, name);
    } catch (ActorNotFoundException ex) {
        throw new ResponseStatusException(
          HttpStatus.BAD_REQUEST, "Provide correct Actor Id", ex);
    }
}
```

Así es como se ve la respuesta:

```
$ curl -i -s -X PUT http://localhost:8080/actor/8/BradPitt
HTTP/1.1 400
...
{
    "timestamp": "2018-02-01T04:28:32.917+0000",
    "status": 400,
    "error": "Bad Request",
    "message": "Provide correct Actor Id",
    "path": "/actor/8/BradPitt"
}
```


5. Conclusión

En este tutorial rápido, discutimos cómo construir una *ResponseStatusException* en nuestro programa.

También enfatizamos cómo es programáticamente una mejor forma de establecer los códigos de estado HTTP en la Respuesta HTTP que la anotación *@ResponseStatus* .

Como siempre, el código fuente completo está disponible en GitHub (https://github.com/eugenp/tutorials/tree/master/spring-5) .

CATEGORÍAS

PRIMAVERA (HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)

DESCANSO (HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SEGURIDAD (HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCIA (HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/)
HTTPCLIENT (HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIE

TUTORIAL "VOLVER A LO BÁSICO" DE JAVA (HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (HTTP://WWW.BAELDUNG.COM/JACKSON)
TUTORIAL DE HTTPCLIENT 4 (HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE)
REST CON SPRING TUTORIAL (HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/)
TUTORIAL DE SPRING PERSISTENCE (HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/)
SEGURIDAD CON SPRING (HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING)

ACERCA DE

ACERCA DE BAELDUNG (HTTP://WWW.BAELDUNG.COM/ABOUT/)
LOS CURSOS (HTTP://COURSES.BAELDUNG.COM)
TRABAJO DE CONSULTORÍA (HTTP://WWW.BAELDUNG.COM/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
EL ARCHIVO COMPLETO (HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE)
ESCRIBIR PARA BAELDUNG (HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES)
CONTACTO (HTTP://WWW.BAELDUNG.COM/CONTACT)
INFORMACIÓN DE LA COMPAÑÍA (HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
8898 (SIN TÍTULO) (HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY)
EDITORES (HTTP://WWW.BAELDUNG.COM/EDITORS)
KIT DE MEDIOS (PDF) (HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF)