Обработка ошибок

- Исключения Spring Boot
- Ошибки в коде приложения

Во время работы приложения могут возникать ошибки, которые должны транслироваться в правильные коды ответа. Всего в Spring можно выделить две большие группы ошибок:

- Ошибки внутри Spring Boot — это исключения, которые фреймворк генерирует в ответ на разные ситуации
- Ошибки в коде приложения — это исключения, которые возникают в нашем коде или коде библиотек внутри нашего кода

Рассмотрим каждую группу отдельно.

Исключения Spring Boot

Взаимодействие с внешней средой непредсказуемо. Всегда есть вероятность получить несоответствие между ожиданиями Spring Boot и реальностью. Обычно это проявляется в неправильно сформированных HTTP-запросах. Представим, что у нас есть такой обработчик:

```
@GetMapping("/api/users")
public List<User> index(@RequestParam Integer limit) {
    // Какой-то код
}
```

Что будет, если мы выполним запрос с параметром `limit`, передав туда строку? В ответ мы получим ошибку `400`, которая указывает на «плохой запрос»:

```
curl "localhost:8080/api/users?limit=qwer"
```

```
{
    "detail": "Failed to convert 'limit' with value: 'qwer'",
    "instance": "/api/users",
    "status": 400,
    "title": "Bad Request",
```

```
    "type": "about:blank"
}
```

Эту ситуацию Spring Boot обрабатывает внутри себя. Точно так же он обрабатывает и множество других ситуаций, среди которых:

- Используется неподдерживаемый HTTP-метод
- Используется неподдерживаемый media type
- Параметры обработчика не проходят валидацию

Всего есть около 15 ошибок-исключений. Каждое исключение обрабатывается автоматически. После обработки мы получаем соответствующий HTTP-код и тело с описанием ошибки:

```
// Список всех автоматически обрабатываемых ошибок
HttpRequestMethodNotSupportedException
HttpMediaTypeNotSupportedException
HttpMediaTypeNotAcceptableException
MissingPathVariableException
MissingServletRequestParameterException
MissingServletRequestPartException
ServletRequestBindingException
MethodArgumentNotValidException
NoHandlerFoundException
AsyncRequestTimeoutException
ErrorResponseException
ConversionNotSupportedException
TypeMismatchException
HttpMessageNotReadableException
HttpMessageNotWritableException
BindException
```

Обработкой этих ошибок в Spring Boot занимается класс ResponseEntityExceptionHandler. Он вызывается автоматически для обработки любых ошибок, возникающих во время обработки маршрута (диспетчеризации). Причем это работает не только для кода самого Spring Boot, но и для кода внутри обработчика.

Чтобы лучше понять обработку ошибок, советуем открыть исходники класса `ResponseEntityExceptionHandler` и изучить их.

Ошибки в коде приложения

Если ничего специально не делать, то любые ошибки внутри кода приложения будут автоматически приводиться к HTTP-коду `500`. Для большинства ошибок это желаемое поведение, но иногда нужна и особая обработка. Например, важно выдать ошибку `404`, когда запись в базе данных не найдена.

Посмотрим, что произойдет в коде ниже, если данных в базе нет:

```
@GetMapping("/users/{id}")
@ResponseStatus(HttpStatus.OK)
public User show(@PathVariable Long id) {
    var user = userRepository.findById(id).get();
    return user;
}
```

В этом случае вернется ошибка с кодом `500`. С точки зрения HTTP, это неверное поведение. В этом случае правильный код возврата — `404`. Технически мы можем вернуть код `404` с помощью `ResponseEntity`. Нам понадобится метод `ResponseEntity.of()`. Он принимает на вход `Optional` и возвращает коды `200` или `404` в зависимости от того, есть результат или нет:

```
@GetMapping("/users/{id}")
@ResponseStatus(HttpStatus.OK)
public ResponseEntity<User> show(@PathVariable Long id) {
    var user = userRepository.findById(id);
    return ResponseEntity.of(user);
}
```

Но бывают и более сложные ситуации, в которых одним таким методом не обойтись. Например, мы не всегда имеем дело с `Optional`, иногда требуется дополнительная логика для формирования ответа. В таких ситуациях, удобнее определить свой обработчик ошибки на базе `ResponseEntityExceptionHandler`.

Для централизованной обработки ошибки `404` нужно выполнить три шага.

Шаг 1. Создать исключение:

```
// src/main/java/io/hexlet/spring/exception/ResourceNotFoundException.java
package io.hexlet.spring.exception;
```

```java
// Имя класса исключения не принципиально
public class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException(String message) {
        super(message);
    }
}
```

Шаг 2. Реализовать глобальный обработчик этого исключения:

```java
// src/main/java/io/hexlet/spring/handler/GlobalExceptionHandler.java
package io.hexlet.spring.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.servlet.mvc.method.annotation.ResponseEntityExceptionHandle

// Используем созданное исключение
import io.hexlet.spring.exception.ResourceNotFoundException;

@ControllerAdvice
public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<String> handleResourceNotFoundException(ResourceNotFoundExceptio
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
    }
}
```

Чтобы Spring Boot начал использовать наш обработчик ошибок для всех контроллеров сразу, нужно использовать несколько аннотаций:

- `@ControllerAdvice` указывает, что этот класс отвечает за централизованную обработку исключений
- `@ExceptionHandler` указывает, какое исключение должно обрабатываться проаннотированным методом

Сам метод похож на обычный обработчик маршрута. Разница только в том, что ему на вход подается возникшее исключение. Далее мы сами описываем логику и возвращаем `ResponseEntity` с правильным кодом и телом ответа.

Шаг 3. Внедрить использование исключения в обработчики маршрутов:

```
@GetMapping("/users/{id}")
@ResponseStatus(HttpStatus.OK)
public User show(@PathVariable Long id) {
    var user = userRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException(id + " Not Found"));
    return user;
}
```

Далее →