

Проверка данных. Перехват исключений



ПРЕПОДАВАТЕЛЬ



Юрий Костяной

Java/Kotlin backend-разработчик

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

ПЛАН ЗАНЯТИЯ

1. Повторение
2. Основной блок
3. Вопросы по основному блоку
4. Домашняя работа

1

ОСНОВНОЙ БЛОК

Введение

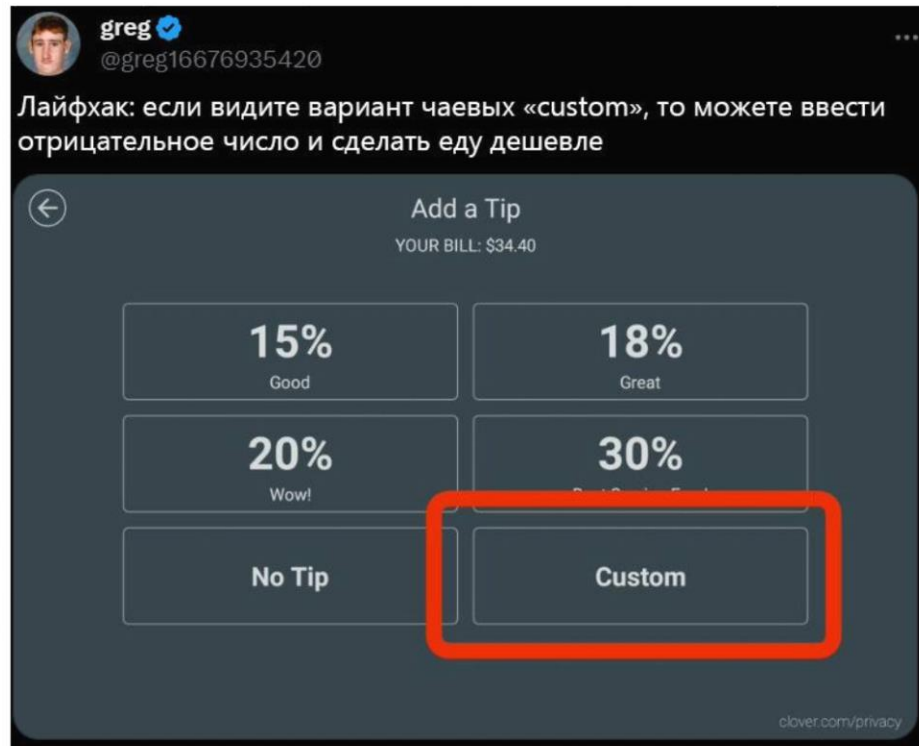
- Раз-раз, проверка
- Действие-противодействие



Проблема

Когда ранее мы писали программы с консольным вводом, нам приходилось делать множество проверок введенных пользователем данных: не введена ли пустая строка там, где она пустой быть не должна, или не указан ли отрицательный возраст или корректен ли адрес электронной почты. В web-приложении приложение постоянно принимает данные извне, а проверки этих данных обычно являются похожими.

Как Spring автоматизирует эти рутинные операции?



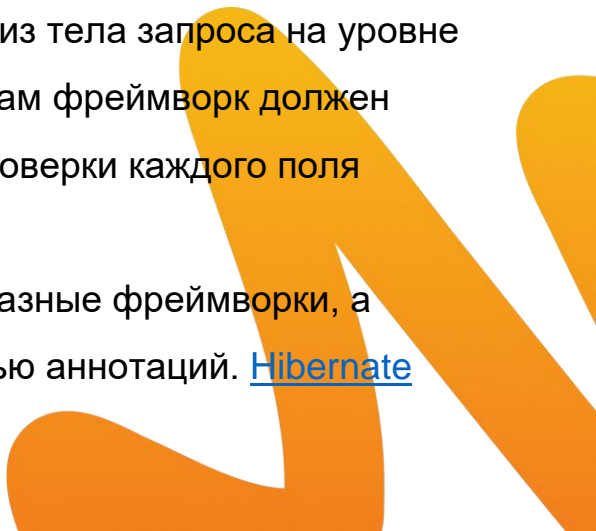
Раз-раз, проверка

JSR-303

Ранее проверку данных (**валидацию**) выполняли в конструкторах и сеттерах классов. Из-за того, что логика валидации была примерно одинаковой для всех классов проекта, такую логику стали выносить в отдельные классы-валидаторы. У таких классов были наборы методов *isNull*, *isEmpty*, *isNegative* и т.д. Это позволило убрать лишний код из сущностей, но требовало вызова этих методов.

Так как в *JEE* и в *Spring*, в частности, объекты создаются по данным из тела запроса на уровне фреймворка, т.е. автоматизировано, то логично предположить, что сам фреймворк должен делать валидацию. Но тогда надо сообщить фреймворку правила проверки каждого поля создаваемого объекта.

JEE предоставляет [Jakarta Bean Validation API](#), которую реализуют разные фреймворки, а спецификация **JSR-303** предлагает давать такие подсказки с помощью аннотаций. [Hibernate Validator](#) считается эталонной реализацией *Bean Validation*.



Раз-раз, проверка

Основные аннотации

Некоторые из наиболее распространенных аннотаций для валидации:

@NotNull – поле не должно быть null.

@NotEmpty – поле не должно быть пустым (подходит для строк, массивов и коллекций).

@Size – устанавливает размер строки или коллекции.

@NotBlank – строковое поле должно содержать хотя бы один непробельный символ.

@Min и **@Max** – определяют границы допустимых значений числового поля.

@Pattern – значение строкового поля должно соответствовать определенному регулярному выражению.

@Email – строковое поле должно быть действительным адресом электронной почты.

DAY1 OF PROGRAMMING



regex for email validation



Google Search

I'm Feeling Lucky

10 YEARS OF PROGRAMMING



regex for email validation



Google Search

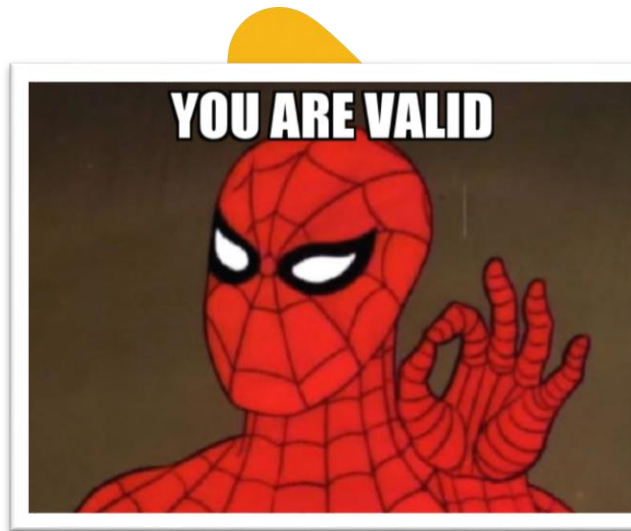
I'm Feeling Lucky

Раз-раз, проверка **@Validated** и **@Valid**

Для других типов данных используются аннотации *@Validated* и *@Valid*.

@Validated – аннотация на уровне класса, которую мы можем использовать, чтобы указать *Spring* проверять параметры, которые передаются в методы аннотированного класса. Т.е. если эта аннотация указана и в контроллере есть метод, у которого есть аргумент, помеченный какой-либо аннотацией для проверки, то *Spring* проверит этот аргумент.

@Valid – аннотация, помещаемая перед аргументом метода, чтобы сообщить *Spring*, что объект, передаваемый в аргумент, должен быть проверен. Если входной аргумент метода не примитивный тип и не *String*, то его следует помечать *@Valid*.

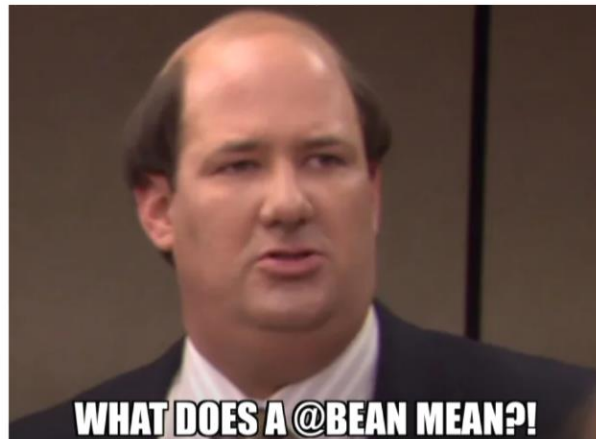


Раз-раз, проверка

Бин для валидации

Сами по себе аннотации не включают работу валидатора. Для активации валидатора нужно создать соответствующий бин в классе конфигурации.

```
@Bean // Добавили для активации валидации
public MethodValidationPostProcessor methodValidationPostProcessor() {
    return new MethodValidationPostProcessor();
}
```



Раз-раз, проверка

Исключения при валидации

Если в контроллере сущность из тела запроса не прошла проверку по `@Valid`, то валидатор выбрасывает `MethodArgumentNotValidException`. *Spring* интерпретирует такое исключение как неправильный запрос и отправляет ответ со статусом HTTP 400.

Если проверку не прошли параметры типа `String`, примитивных типов или их обёрток, то будет выброшено

`ConstraintViolationException`. По умолчанию последует ответ со статусом HTTP 500 (Internal Server Error), так как *Spring* не регистрирует обработчик для этого исключения по умолчанию.



ARE YOU AN EXCEPTION?

BECAUSE I CAN'T WAIT TO CATCH YOU.

Раз-раз, проверка

Исключения при валидации

Валидацию можно проводить не только в контроллере:

- на уровне сервиса. Если проверка объекта по `@Valid` не пройдена, то будет выброшено *ConstraintViolationException*, а контроллер вернёт 500 статус;
- на уровне хранения данных (*persistence layer*). Когда приложение пытается сохранить не валидный объект, выбрасывается *ConstraintViolationException*.

Лучше проводить валидацию на наиболее раннем этапе, т.е. в контроллере. Но если сервисы вызывают друг друга в приложении, то один сервис сможет передавать невалидные параметры в другой, поэтом валидацию следует спустить или продублировать на уровень сервисов.

При изучении *Spring Boot* мы также узнаем об аннотации `@ConfigurationProperties`, которая позволяет создавать объекты классов, поля которых заполнены значениями из конфигурационного файла. Эти значения тоже могут быть провалидированы.

Раз-раз, проверка

Пример организации проверки



SpringValidationExample.zip



Задание

Создайте проект, который позволяет регистрировать спортсменов на марафон.

Зарегистрированы могут быть только совершеннолетние спортсмены, указавшие имя, фамилию и дающие согласие на обработку персональных данных.

1 добавьте зависимость

<https://mvnrepository.com/artifact/org.hibernate.validator/hibernate-validator>

2 создайте бин MethodValidationPostProcessor в конфигурации.

3 укажите нужные аннотации в контроллерах и, при необходимости, в сервисах.

4 Для создания чек-бокса (галочки) используйте thymeleaf-код

```
<form... >
```

```
  <label>
```

```
    <input type="checkbox" th:checked="${flag}"/> Flag activated
```

```
  </label>
```

```
</form>
```

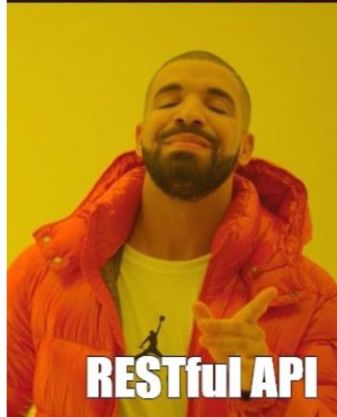
Проблема

Как мы увидели, MVC-контроллеры в случае ошибки отдают представление для информирования пользователя об ошибке. Но в RESTful API Spring-контроллеры возвращают стандартные html-ответы с кодом ошибки 500, что нарушает RESTful API, т.к. работа производится через объекты JSON.

Можно ли заменить стандартный ответ на отправку JSON с ошибкой?



HTML



JSON

Действие-противодействие

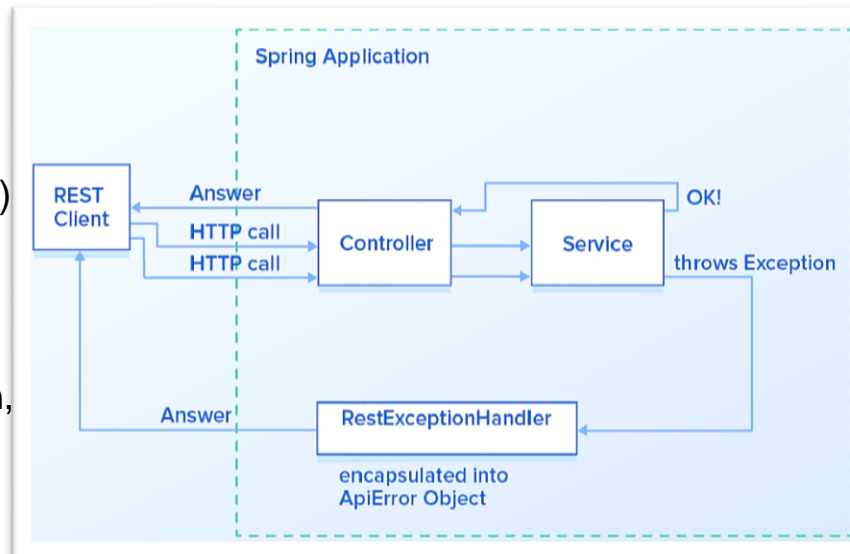
Перехват исключений

До Spring 3.2 для перехвата сообщений нужно было имплементировать

HandlerExceptionResolver или использовать аннотацию **@ExceptionHandler**, предоставляющую механизм обработки исключений, возникающих во время выполнения обработчиков (операций контроллера). Эта аннотация, если она используется в методах классов контроллера, будет служить точкой входа для обработки исключений, создаваемых только внутри этого контроллера.

Для создания исключений для всего приложения (нескольких контроллеров и/или REST-контроллеров) используется класс, помеченный аннотацией **@ControllerAdvice**.

В Spring 5 появился класс **ResponseStatusException**, с которым перехват исключений упростился, т.к. он реализован в Spring в автоматическом режиме.



Действие-противодействие

Назначение **@ControllerAdvice**

- Глобальная обработка исключений: вместо того, чтобы обрабатывать исключения в каждом контроллере отдельно, можно определить класс с аннотацией *@ControllerAdvice*, который будет автоматически перехватывать и обрабатывать исключения, брошенные любым контроллером. Для обработки исключений внутри класса, аннотированного *@ControllerAdvice*, обычно используют аннотацию *@ExceptionHandler*.
- *@ControllerAdvice* позволяет конфигурировать глобальные преобразователи данных и валидаторы, которые будут применяться ко всем контроллерам в приложении.
- Добавление общих атрибутов к моделям перед возвращением представлений.
- Предварительная и постобработка запросов (менее распространенное использование).
Например, для изменения тела запроса или ответа перед их отправкой.

Действие-противодействие

Пример @ControllerAdvice

```
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(value = Exception.class)
    public ResponseEntity<Object> handleException(Exception e) {
        // Логика обработки исключений
        return new ResponseEntity<>(
            "Произошла ошибка: " + e.getMessage(),
            HttpStatus.INTERNAL_SERVER_ERROR
        );
    }
}
```

Действие-противодействие

Пример обработки исключений



SpringControllerAdviceExample.zip



Задание

Создайте RESTful-приложение для сбора данных о животном в ветеринарной клинике.

Создайте методы добавления данных животного, а также методы получения информации – по id, по кличке животного, виду, породе и дате рождения.

Если животное не найдено, то должна вернуться соответствующая ошибка http.



2

Домашнее задание

Домашнее задание

Создайте приложение для хранения дней рождений Ваших друзей. Приложение должно хранить имя (более одного символа) и фамилию (более одного символа) друга, а также его дату рождения (любая дата из прошлого, включая сегодняшнюю). Создайте метод для добавления записи о дне рождения, метод получения всех хранимых записей, метод получения сегодняшних именинников и методы поиска (именинники по дате, дата по имени и фамилии). Если при поиске данные не найдены, то должно быть брошено исключение, а контроллер должен обработать исключение и дать ответ 404. Выполните валидацию данных при создании записи. В ответ на невалидные данные должен быть дан ответ с кодом 400.

ЗАКЛЮЧЕНИЕ

