Частичное обновление

- Установка
- Подключение к DTO
- Подключение к MapStruct
- Валидация

Из всех CRUD-операций, обновление сложнее всего реализовать правильно. В этом уроке вы узнаете, как правильно написать код для обновления сущностей с возможностью делать это частично.

Предположим, что мы написали такой код для обновления сущности `Post`:

```
@RestController
@RequestMapping("/api")
public class PostsController {
    @Autowired
    private PostRepository repository;

    @Autowired
    private PostMapper postMapper;


    @PutMapping("/posts/{id}")
    @ResponseStatus(HttpStatus.OK)
    public PostDTO update(@RequestBody @Valid PostUpdateDTO postData, @PathVariable Long i
        var post = repository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Not Found"));
        postMapper.update(postData, post);
        repository.save(post);
        var postDTO = postMapper.map(post);
        return postDTO;
    }
}
```

При этом код DTO выглядит так:

```java
@Setter
@Getter
public class PostUpdateDTO {
    private String name;
    private String body;
}
```

Код маппера выглядит так:

```java
public abstract class PostMapper {
    public abstract PostDTO map(Post model);
    public abstract void update(PostUpdateDTO dto, @MappingTarget Post model);
}
```

Представим, что мы отправляем такой JSON в этот API:

```json
{
  "name": "new name"
}
```

В коде выше мы не передали `body`, поэтому в DTO это свойство будет равно `null`. Это значит, что при копировании данных из DTO в объект `post` оригинальное значение будет стерто.

Такая реализация обновления работает только тогда, когда передаются все поля, указанные в DTO. На практике это порождает проблемы при одновременных обновлениях от разных клиентов — клиенты будут стирать данные друг друга.

Кажется, что проблема решилась бы, если перед установкой значения мы проверили бы его на наличие `null`:

```java
if (dto.getName() != null)
    post.setName(dto.getName());
}
```

На самом деле, этот способ не сработает, потому что может быть ситуация, при которой мы действительно передали такое значение в JSON:

```json
{
  "name": "new name",
```

```
  "body": null
}
```

Эта ситуация возникает из-за того, что у свойств в объектах есть только два возможных значения:

- Либо `null`
- Либо какое-то конкретное значение

Если в DTO оказался `null`, что это значит? Возможны два варианта:

- Либо `null` действительно пришел в JSON снаружи
- Либо свойство не было установлено вообще

В таких условиях мы не можем с уверенностью сказать, какой вариант правильный.

Решить эту проблему можно с помощью модуля jackson-databind-nullable в связке с MapStruct.

Обсудим принцип работы *jackson-databind-nullable* подробнее. Сначала мы оборачиваем в класс `JsonNullable` какое-то свойство, которое может отсутствовать. Дальше применяется следующая логика:

- Если в свойстве находится явный `null`, значение удалено явно
- Если в свойстве находится `JsonNullable.undefined()`, значение не передано — его нужно игнорировать
- Если в свойстве находится реальное значение, обернутое в `JsonNullable`, то нужно его использовать

Пройдем весь путь подключения этой библиотеки к проекту и MapStruct.

Установка

Для начала библиотеку нужно установить:

```
implementation("org.openapitools:jackson-databind-nullable:0.2.6")
```

Сам модуль подключается к Jackson с помощью конфигурационного класса:

```
// src/main/java/io/hexlet/spring/config/JacksonConfig
package io.hexlet.spring.config;
```

```java
import org.openapitools.jackson.nullable.JsonNullableModule;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.converter.json.Jackson2ObjectMapperBuilder;

import com.fasterxml.jackson.annotation.JsonInclude;

@Configuration
public class JacksonConfig {
    @Bean
    Jackson2ObjectMapperBuilder objectMapperBuilder() {
        var builder = new Jackson2ObjectMapperBuilder();
        builder.serializationInclusion(JsonInclude.Include.NON_NULL)
                .modulesToInstall(new JsonNullableModule());
        return builder;
    }
}
```

Подключение к DTO

JacksonNullable подключается к опциональным свойствам DTO, то есть эти свойства могут быть пропущены при формировании JSON с клиентской стороны:

```java
package io.hexlet.spring.dto;

import org.openapitools.jackson.nullable.JsonNullable;

import jakarta.validation.constraints.NotNull;
import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class PostUpdateDTO {
    private JsonNullable<String> name;

    private JsonNullable<String> body;
}
```

Подключение к MapStruct

По умолчанию MapStruct ничего не знает о `JsonNullable`. Чтобы добавить нужную нам условную логику, проверяющую наличие реального значения, надо добавить специальный маппер:

```
// src/main/java/io/hexlet/spring/mapper/JsonNullableMapper.java
package io.hexlet.spring.mapper;

import org.mapstruct.Condition;
import org.mapstruct.Mapper;
import org.mapstruct.MappingConstants;
import org.openapitools.jackson.nullable.JsonNullable;

@Mapper(
    componentModel = MappingConstants.ComponentModel.SPRING
)
public abstract class JsonNullableMapper {

    public <T> JsonNullable<T> wrap(T entity) {
        return JsonNullable.of(entity);
    }

    public <T> T unwrap(JsonNullable<T> jsonNullable) {
        return jsonNullable == null ? null : jsonNullable.orElse(null);
    }

    @Condition
    public <T> boolean isPresent(JsonNullable<T> nullable) {
        return nullable != null && nullable.isPresent();
    }
}
```

Это универсальный маппер, который можно подключить к любым другим мапперам. В нашей ситуации он понадобится для реализации маппера `PostMapper`:

```
package io.hexlet.spring.mapper;

import org.mapstruct.InheritConfiguration;
import org.mapstruct.InheritInverseConfiguration;
import org.mapstruct.Mapper;
```

```java
import org.mapstruct.Mapping;
import org.mapstruct.MappingConstants;
import org.mapstruct.MappingTarget;
import org.mapstruct.NullValuePropertyMappingStrategy;
import org.mapstruct.ReportingPolicy;

import io.hexlet.blog.dto.PostCreateDTO;
import io.hexlet.blog.dto.PostDTO;
import io.hexlet.blog.dto.PostUpdateDTO;
import io.hexlet.blog.model.Post;

@Mapper(
    // Подключение JsonNullableMapper
    uses = { JsonNullableMapper.class },
    nullValuePropertyMappingStrategy = NullValuePropertyMappingStrategy.IGNORE,
    componentModel = MappingConstants.ComponentModel.SPRING,
    unmappedTargetPolicy = ReportingPolicy.IGNORE
)
public abstract class PostMapper {
    // Остальные методы
    public abstract void update(PostUpdateDTO dto, @MappingTarget Post model);
}
```

После такого изменения реализация метода `update()` в сгенерированном маппере значительно меняется:

```java
@Component
public class PostMapperImpl extends PostMapper {

    @Autowired
    private JsonNullableMapper jsonNullableMapper;

    @Override
    public void update(PostUpdateDTO dto, Post model) {
        if ( dto == null ) {
            return;
        }
        if ( jsonNullableMapper.isPresent( dto.getName() ) ) {
            model.setName( jsonNullableMapper.unwrap( dto.getName() ) );
        }
        if ( jsonNullableMapper.isPresent( dto.getBody() ) ) {
            model.setBody( jsonNullableMapper.unwrap( dto.getBody() ) );
```

```
        }
    }
}
```

Валидация

Как в таком случае использовать валидацию? Валидация же должна применяться к оригинальному значению, а не свойству в целом — иначе мы не сможем использовать `null` как значение. Хорошая новость в том, что это происходит автоматически. Все добавленные аннотации применяются не к обертке, а к тому, что находится внутри нее. Соответственно, если значение не передано, то и валидация не применяется. Значит, мы можем писать так:

```
package io.hexlet.spring.dto;

import org.openapitools.jackson.nullable.JsonNullable;

import jakarta.validation.constraints.NotNull;
import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class PostUpdateDTO {
    @NotNull
    private JsonNullable<String> name;

    @NotNull
    private JsonNullable<String> body;
}
```

Дополнительные материалы

1. Реализация частичного обновления (EN)