Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Дисциплина: Технологии веб-сервисов

Лабораторная работа 3

Выполнили:
Кривоносов Егор Дмитриевич

Группа: P4214

Преподаватель:
Сафронов Андрей Геннадьевич

2024 г.

Санкт-Петербург

Оглавление

Техническое задание **3**
Постановка задачи **3**
Этапы выполнения **3**
Комментарии к архитектурным и функциональным аспектам реализации **4**
Ссылка на код **6**
Вывод **7**

Техническое задание

Основываясь на информации из раздела 2.8, добавить поддержку обработки ошибок в сервис. Возможные ошибки, которые могут происходить при добавлении, изменении или удалении записей: неверное значение одного из полей; попытка изменить или удалить несуществующую запись.

В соответствии с изменениями сервиса необходимо обновить и клиентское приложение.

Постановка задачи

Задача заключалась в добавлении обработки ошибок в существующий SOAP-сервис и обновлении клиентского приложения. На сервере нужно было внедрить детализированные сообщения об ошибках, которые будут возвращаться клиенту при некорректных запросах. Клиентское приложение должно корректно обрабатывать ошибки сервиса и отображать их пользователю.

Этапы выполнения

1. **Расширение SOAP-сервиса**
    - Добавлен новый класс PersonServiceException для возврата ошибок. Этот класс включает в себя FaultBean, который используется для передачи подробной информации об ошибке. PersonServiceException реализован с аннотацией @WebFault, чтобы его можно было обрабатывать в soap:Fault ответа.
    - Добавлен класс PersonValidationService для проверки данных, поступающих в сервис. Этот класс валидирует поля объекта PersonDto, возвращая все найденные ошибки сразу, если их несколько. Например, некорректные значения для имени или номера телефона передаются в FaultBean.
    - Методы createPerson, updatePerson, deletePersonById, и findPerson были обновлены для использования PersonServiceException с целью обработки ошибок.
2. **Обновление клиентского приложения**

- В клиентское приложение добавлена обработка PersonServiceException для команд create, update, delete, findById, и search. Метод getPersonDtoFromInput был модифицирован с использованием рефлексии для более универсального считывания полей.

Комментарии к архитектурным и функциональным аспектам реализации

Добавление PersonServiceException позволяет передавать ошибки сервиса в формате soap:Fault, что делает их более структурированными и удобными для обработки. Использование FaultBean для передачи деталей ошибки делает сообщения более информативными и помогает клиенту понять источник проблемы.

Использование рефлексии в getPersonDtoFromInput упростило ввод данных, позволяя универсально получать от сервера класс и не хардкодить на клиенте типы данных и задавать значения полей PersonDto.

Пример PersonServiceException:

```
@Getter
@WebFault(name = "PersonServiceFault")
public class PersonServiceException extends Exception {
    private final FaultBean faultInfo;

    public PersonServiceException(String message, FaultBean faultInfo) {
        super(message);
        this.faultInfo = faultInfo;
    }

    public PersonServiceException(String message, FaultBean faultInfo, Throwable
cause) {
        super(message, cause);
        this.faultInfo = faultInfo;
    }
}
```

Обновленные методы в PersonWebService:

```
    @WebMethod
    public List<Person> searchPersons(@WebParam(name = "arg0") PersonListRequestDto
personListRequestDto) throws PersonServiceException {
        if (personListRequestDto == null) {
            System.out.println("Received null PersonListRequestDto");
            return Collections.emptyList();
        }
```

```
        int limit = personListRequestDto.getLimit() != null ?
personListRequestDto.getLimit() : 10;
        int offset = personListRequestDto.getOffset() != null ?
personListRequestDto.getOffset() : 0;

        return personService.searchPersons(personListRequestDto.getQuery(), limit,
offset);
    }

    @WebMethod
    public int createPerson(@WebParam(name = "personDto") PersonDto personDto)
throws PersonServiceException {
        PersonValidationService.validatePersonDto(personDto);
        return personService.createPerson(personDto);
    }

    @WebMethod
    public boolean updatePerson(
            @WebParam(name = "id") int id,
            @WebParam(name = "personDto") PersonDto personDto
    ) throws PersonServiceException {
        PersonValidationService.validatePersonDto(personDto);
        boolean updated = personService.updatePerson(id, personDto);

        if (!updated) {
            throw new PersonServiceException("Person not found", new FaultBean("No
person found with id: " + id));
        }
        return true;
    }
```

Метод для валидации DTO, которое мы получаем от клиента:

```
public class PersonValidationService {
    private static final String PHONE_REGEX = "^\\+?[0-9]{10,15}$";

    public static void validatePersonDto(PersonDto personDto) throws
PersonServiceException {
        StringBuilder errorMessageBuilder = new StringBuilder();

        if (personDto == null) {
            throw new PersonServiceException("PersonDto is null", new
FaultBean("Provided PersonDto is null"));
        }
        if (personDto.getName() == null || personDto.getName().isEmpty()) {
            errorMessageBuilder.append("Name field cannot be empty. ");
        }
        if (personDto.getSurname() == null) {
            errorMessageBuilder.append("Surname field cannot be empty. ");
        }
        if (personDto.getAge() < 0) {
            errorMessageBuilder.append("Age must be a non-negative integer. ");
        }
        if (personDto.getPhoneNumber() != null &&
!personDto.getPhoneNumber().matches(PHONE_REGEX)) {
            errorMessageBuilder.append("Phone number must match the format:
+79996665544. ");
```

```
        }
        if (!errorMessageBuilder.isEmpty()) {
            throw new PersonServiceException(
                    "Validation failed for PersonDto",
                    new FaultBean(errorMessageBuilder.toString().trim())
            );
        }
    }
}
```

Метод getPersonDtoFromInput с рефлексией в клиенте:

```
    public static PersonDto getPersonDtoFromInput(Scanner scanner) {
        PersonDto personDto = new PersonDto();
        System.out.println("Enter person details:");
        for (Field field : PersonDto.class.getDeclaredFields()) {
            field.setAccessible(true);
            System.out.print(field.getName() + " (" +
field.getType().getSimpleName() + "): ");
            try {
                if (field.getType() == String.class) {
                    String input = scanner.nextLine();
                    field.set(personDto, input);

                } else if (field.getType() == int.class) {
                    while (true) {
                        try {
                            int input = Integer.parseInt(scanner.nextLine());
                            field.set(personDto, input);
                            break;
                        } catch (NumberFormatException e) {
                            System.out.print("Invalid format. Please enter a valid
integer for " + field.getName() + ": ");
                        }
                    }
                }
            } catch (IllegalAccessException e) {
                System.out.println("Error setting value for field " +
field.getName() + ": " + e.getMessage());
            }
        }
        return personDto;
    }
```

Ссылка на код

https://github.com/RedGry/TVS-LABS/tree/lab3

Вывод

В данной лабораторной работе была успешно реализована обработка ошибок в SOAP-сервисе с использованием PersonServiceException и FaultBean. Это позволило улучшить взаимодействие клиента с сервером, делая сообщения об ошибках более понятными и информативными. Клиентское приложение также было обновлено для корректного отображения сообщений об ошибках сервиса.