Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет информационных технологий и программирования

Лабораторная работа №2

Codegen

Выполнил студенты группы № M3203
Хряков И.В.

Санкт-Петербург
15.04.2022

1. *Написать HTTP-сервер, которые предоставляет несколько методов (в качестве примера, можно взять 2-3 лабораторные второго подпотока). Рекомендуемый язык - Java ввиду простоты поднятия и прочего. Можно использовать любой другой (лучше заранее согласовать). Примеры необходимого функционала:*

1. *GET, POST запросы*
2. *Запросы с аргументами в Query, в Body*
3. *Сложные модели с Response (не примитивы, хотя бы классы с полями)*
4. *Аргументы, которые являются коллекциями, респонсы, которые коллекции содержат*

Мной был выбран SpringBoot в качестве инструмента для поднятия сервера на Java.

Чтобы написать свой сервер создал Maven-проект. Добавил в pom.xml необходимые зависимости. Все необходимые jar-библиотеки автоматически импортировались в проект.

Далее создал несколько POJO-классов.

```java
package com.example.test1.entities;

import java.util.*;

public class Master {
    private String name;
    private String birthday;
    private List<Cat> cats;
    private long id;

    public Master(String newName, String newBirthday) {
        this.name = newName;
        this.birthday = newBirthday;
        cats = new ArrayList<Cat>();
        this.id = (long) (Math.random() * 1000);
    }
    public String getName() {
        return this.name;
    }

    public String getBirthday() {
        return this.birthday;
    }
    public List<Cat> getCats() {
```

```java
package com.example.test1.entities;

import java.util.ArrayList;
import java.util.List;


public class Cat {
    private String name;
    private String species;
    private String birthday;
    private String color;
    private long masterId;
    private List<Cat> friends;
    private long id;

    public Cat(String Name, String Species, String Birthday, String Color, long MasterId) {
        this.name = Name;
        this.species = Species;
        this.birthday = Birthday;
        this.color = Color;
        this.masterId = MasterId;
        friends = new ArrayList<Cat>();
        this.id = (long) (Math.random() * 1000);
    }
```

А также сделал сервис для работы с ними. Далее создал контроллер для отправления запросов и получения ответов от сервера.

```java
package com.example.test1.controller;

import com.example.test1.entities.Cat;
import com.example.test1.entities.Master;
import com.example.test1.service.CatService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;


@RestController
public class Controller {

    @Autowired
    public CatService _catService = new CatService();

    @GetMapping(value = "/masters")
    public ResponseEntity<List<Master>> GetMasters(){
        final List<Master> masters = _catService.GetMasters();
        return new ResponseEntity<>(masters, HttpStatus.OK);
    }

    @GetMapping(path = "/getMaster/{id}", produces =
MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Master> GetMaster(@PathVariable long id){
        return new ResponseEntity<>(_catService.GetMaster(id),
```

```
HttpStatus.OK);
    }

    @PostMapping(path = "/createMaster")
    public ResponseEntity<?> CreateMaster(@RequestBody Master newMaster){
        Master master = new Master(newMaster.getName(),
newMaster.getBirthday());
        _catService.AddMaster(master);
        return new ResponseEntity<>(master.getId(), HttpStatus.OK);
    }


    @GetMapping(path = "/getCat/{id}", produces =
MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Cat> GetCat(@PathVariable long id){
        final Cat cat = _catService.GetCat(id);
        return cat != null
                ? new ResponseEntity<>(cat, HttpStatus.OK)
                : new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }

    @PostMapping(path = "/createCat")
    public ResponseEntity<?> CreateCat(@RequestBody Cat cat){
        Cat newCat = new Cat(cat.getName(), cat.getSpecies(),
cat.getBirthday(), cat.getColor(), cat.getMasterId());
        _catService.CreateCat(newCat);
        return new ResponseEntity<>(newCat.getId(), HttpStatus.OK);
    }
    @PostMapping(path = "/setFriends/{id}")
    public ResponseEntity<?> SetFriends(@RequestBody List<Cat> cats,
@PathVariable long id){
        var cat = _catService.GetCat(id);
        var newCats = new ArrayList<Cat>();
        for(int i = 0; i < cats.size(); i++){
            var buffCat = new Cat(cats.get(i).getName(),
cats.get(i).getSpecies(), cats.get(i).getBirthday(), cats.get(i).getColor(),
cats.get(i).getMasterId());
            newCats.add(buffCat);
        }
        cat.setFriends(newCats);
        return new ResponseEntity<>(cat.getFriends(), HttpStatus.OK);
    }
}
```

REST API запущен и готов работать с портом http://localhost:8080/

2. Написать упрощённый парсер (на C#) для этого сервера, чтобы можно было получить семантическую модель (можно использовать любые библиотеки для этого), а именно:

1. *Описание методов из API - url, список аргументов, возвращаемое значение*
2. *Модели, которые используются в реквестах и респонсах*

Был создан проект на c# для парсинга файлов на java. Необходимо было распарсить непосредственно контроллер и классы POJO (DTO).

Из контроллера доставались HttpMethodName, URL для запросов, список аргументов, тип возвращаемого значения, имя метода.
Из POJO классов получил все поля данных классов а также список аргументов в конструкторе класса.
Результат парсинга контроллера

```
Get /masters List<Master> GetMasters
Get /getMaster/{id} Master GetMaster
Post /createMaster ? CreateMaster
Get /getCat/{id} Cat GetCat
Post /createCat ? CreateCat
Post /setFriends/{id} ? SetFriends
```

```
long id False
Master newMaster True
long id False
Cat cat True
List<Cat> cats True
long id False
```

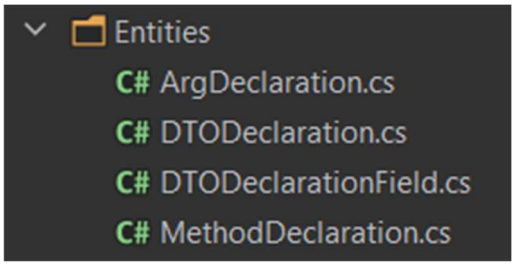
Результат парсинга POJO-классов

```
private string name
private string species
private string birthday
private string color
private long masterId
private List<Cat> friends
private long id

private string name
private string birthday
private List<Cat> cats
private long id
```

Сверху – класс Cat, снизу класс Master.

Данная информация для каждого метода/поля хранится в полях специально предназначенных для этого классах. У меня их 4:



ArgDeclaration отвечает за хранение списка аргументов.
DTODeclaration хранит список DTODeclarationField а также всю необходимую информацию об конструкторе.
DTODeclarationField хранит тип поля, возвращаемый тип и имя поля.
MethodDeclaration отвечает за хранения всей мета-информации о методе в контроллере. хранятся HttpMethodName, URL для запросов, список аргументов, тип возвращаемого значения, имя метода.

3. Генерация клиента
Реализовать генерацию HTTP-клиента для данного сервера. Для API должны генерироваться все нужные модели, методы.
Для генерации клиента использовал синтаксические деревья, точнее статические методы класса SyntaxFactory.
Основными элементами дерева (узлами (нодами)) - являются классы, наследованные от CSharpSyntaxNode. Это все основные конструкции языка. К ним относятся объявления, операторы, выражения, атрибуты, блоки, условия и т.д.
В основном использовал ноды типа SyntaxNode. Это объявления, операторы, выражения и т.п. Также применял ноды типа SyntaxToken для модификаторов доступа и ключевых слов. Для корректной работы url воспользовался SyntaxTrivia для добавления $ перед url.

```
var getter :MethodDeclarationSyntax = SyntaxFactory.MethodDeclaration(
        returnType: SyntaxFactory.IdentifierName(field.GetReturnType()),
        SyntaxFactory.Identifier( text: "Get" + field.GetName()))
    .WithModifiers(publicKey)
    .WithBody(
        SyntaxFactory.Block(
            statements: SyntaxFactory.SingletonList<StatementSyntax>(
                node: SyntaxFactory.ReturnStatement(
                    expression: SyntaxFactory.IdentifierName(field.GetName())))));
```

```
SyntaxFactory
    .InterpolatedStringText()
    .WithTextToken(
        SyntaxFactory.Token(
            leading: SyntaxFactory
                .TriviaList(), SyntaxKind.InterpolatedStringTextToken, text: "http://localhost:8080" +
            method.GetUrl(), valueText: "http://localhost:8080" + method.GetUrl(), trailing: SyntaxFactory.TriviaList()))
```

SourceGenerator позволяет добавлять новые файлы C# в компиляцию. То есть прямо во время компиляции можно указать дополнительный исходный код в качестве входных данных для компиляции.

Алгоритм работы: