REST API Annotations

DeepDive Week 2 Day 1

Материалы и Ресурсы

Итак, сегодня мы почти полностью приведем наш REST API Framework в то состояние, в котором его будут потреблять наши пользователи (другие разработчики). Мы добавим серию аннотаций, чтобы любой разработчик мог всего лишь добавить нужные аннотации себе в код и все работало. Давайте для начала посмотрим на то, как сторонний разработчик будет использовать наш фреймворк.

Предположим, что разработчику нужно написать два HTTP метода:

- первый возвращает список пользователей
- второй создает нового пользователя

Для этого разработчику нужно всего лишь пометить свою логику нашими аннотациями вот так:

```
package academy.kovalevskyi.javadeepdive.week2.day1;

@Controller
public class UsersList {

   @Get
   @Path("/users")
   public String[] users() {
       return new String[] {"one", "two"};
   }

   @Post
   @Path("/users")
   public void addUser(User user) {
   }
}
```

После этого нужно всего лишь запустить наш сервер:

```
var serverThread = new RestServer("academy.kovalevskyi.javadeepdive");
serverThread.start();
```

- сервер сам обойдет все классы в заданном пакете,
- найдет те, которые помечены аннотацией “@Controller”,
- создаст экземпляры каждого такого класса,
- пройдет по каждому методу каждого такого класса и
- запомнит для какого Path и HttpRequest типа, какой именно метод должен вызываться.

Далее (*при новом входящем запросе на сервер*) наш сервер:

- **проверяет** есть ли задекларированный обработчик этого входящего запроса
- если да, то **проверяет** принимает ли на вход метод какой-либо объект (например addUser принимает на вход User)
- если принимает, то наш сервер **преобразовывает** тело (body) входящего запроса в объект данного типа (предполагается, что в теле запроса находится JSON нужного объекта)
- **вызывает** нужный метод
- если метод возвращает что-то, то это “что-то” должно быть преобразовано в JSON
- финал - **отвечает** на входящий запрос

Давайте рассмотрим все на примере немного детальнее.

```
var serverThread = new RestServer("academy.kovalevskyi.javadeepdive");
serverThread.start();
```

В этих двух строчках наш сервер используя рефлексию, пройдет по всем классам, увидит что есть класс UsersList, который помечен аннотацией и создаст объект этого типа, а так же запомнит, что в этом объекте есть методы, которые отвечают за запросы:

- GET /users
- POST /users

и начнет ждать входящие запросы.

Теперь давайте рассмотрим, что будет происходить при вызове вот такого запроса:

```
curl -X POST -d "{'mail': 'test@test.com', 'password': '123'}" -vvv
localhost:8080/users
```

Наш сервер, получив запрос, найдет каким методом его нужно обрабатывать (addUser) и у какого объекта этот метод. Наш сервер увидит, что на вход методу нужен объект User и соответственно создаст такой объект преобразовав JSON из тела запроса ("{'mail': 'test@test.com', 'password': '123'}") в новый объект User, после чего вызовет с этим новым User метод addUser.

Теперь давайте посмотрим на второй пример:

```
curl -vvv localhost:8080/users
```

- сервер поймет, что этот запрос должен обработать метод “public String[] users()”, который на вход ничего не принимает.
- сервер вызовет этот метод
- при формировании HTTP ответа, то что вернул метод, нужно преобразовать в JSON (так как данный метод возвращает String[])
- ответ будет выглядеть так:

```
➜  ~ curl -vvv localhost:8080/users
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8080 (#0)
> GET /users HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Content-Length: 18
<
["one", "two"]
```

API

API довольно простой. Нам нужно создать серию аннотаций:

- academy.kovalevskyi.javadeepdive.week2.day1.Controller
- academy.kovalevskyi.javadeepdive.week2.day1.Put
- academy.kovalevskyi.javadeepdive.week2.day1.Post
- academy.kovalevskyi.javadeepdive.week2.day1.Get
- academy.kovalevskyi.javadeepdive.week2.day1.Delete
- academy.kovalevskyi.javadeepdive.week2.day1.Path

и наш сервер:

```
package academy.kovalevskyi.javadeepdive.week2.day1;
import academy.kovalevskyi.javadeepdive.week1.day2.HttpMethod;
import org.reflections.Reflections;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.net.ServerSocket;
import java.util.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class RestServer extends Thread {
 // TODO
 public RestServer(String packagePath) {
   // TODO
 }
 public void run() {
   // TODO
 }
 public void stopServer() {
   // TODO
 }
 public boolean isLive() {
   // TODO
 }
}
```

Ограничения

Для выполнения задания разрешено использовать следующую зависимость:

```
<dependency>
 <groupId>org.reflections</groupId>
 <artifactId>reflections</artifactId>
 <version>0.9.12</version>
</dependency>
```

Для прохождения тестов наш framework НЕ должен уметь:

- поддерживать пути глубже чем один (т.е. /users - OK, /users/123 - NOT OK)
- поддерживать пути с переменными (т.е. /%userId - NOT OK)

Однако если хотите можете это реализовать ;)