Sử dụng OpenFeign để làm REST Client

Ở bài trước chúng ta đã dùng WebFlux WebClient để làm client kết nối vào REST API. Web Client có rất nhiều tính năng mạnh đặc biệt là non blocking call. Tuy nhiên cú pháp của WebClient thì hơi khó học và nhớ

```
Mono<Player[]> result = webClient.get()
  .uri(uriBuilder -> uriBuilder
      .path("/team")
      .build())
  .accept(MediaType.APPLICATION_JSON)
  .retrieve()
  .bodyToMono(Player[].class)
  .log();

Player[] players = result.block();
```

Thực ra có rất nhiều thư viện trong Java hỗ trợ tạo request lên HTTP server. *REST API cũng là một server trả về dữ liệu định dạng JSON. Việc chuyển body dạng JSON String thục ra là việc của một thư viện khác.*

1. Retrofit hãy dùng trong ứng dụng Android
2. OkHttp cú pháp dễ hiểu. Được đánh giá rất cao. Follow - Start github cực cao
3. OpenFeign cách tiếp cận khá hay, cú pháp trong sáng.
4. RestTemplate thư viện chuẩn của Spring, chỉ hỗ trợ blocking call, đã bị thay thế bởi WebFlux WebClient
5. Eclipse Jersey
6. Resteasy
7. Which Java HTTP client should I use in 2020? tham khảo thêm bài so sánh các thư viện Http Client

Tại sao tôi chọn Open Feign?

Nếu như OKHttp và Retrofit là 2 mã nguồn mở được khởi xướng bởi Square, một start up chuyên về máy POST, thanh toán ở Mỹ. Feign ban đầu được phát triển dự án nội bộ của NetFlix, công ty chuyên cho thuê phim trực tuyến. Sau một thời gian NetFlix quyết định mở mã nguồn Feign. Spring Cloud cũng tích hợp Open Feign như một thư viện thành phần.

Cách khai báo một request bằng Interface method cùng annotation trong Feign giống cách khai báo một phương thức trong interface của JPA repository. Khai báo thay cho lập trình ~ Declaring over Programming.

```
public interface BarcaClient {
  @RequestLine("GET /team")
  Set<Player> getTeam();

  @RequestLine("GET /chooseteam/{team_pattern}")
  Set<Player> chooseTeam(@Param("team_pattern") String team_pattern);
}
```

Feign là một thư viện cấp cao, phía dưới có thể cấu hình để dùng OKHttp như sau:

```
Feign.builder()
  .client(new OkHttpClient())
  .target(GitHub.class, "https://api.github.com");
```

Các bước cấu hình và lập trình REST Client bằng Feign

```
.
├── main
│   ├── java
│   │   ├── com
│   │   │   ├── onemount
│   │   │   │   ├── barcelonateam
│   │   │   │   │   ├── controller
│   │   │   │   │   │   ├── APIController.java  <-- RestController
│   │   │   │   │   │   └── CustomExceptionHandler.java <-- RestControllerAdvice biến Exception thành b
│   │   │   │   │   ├── exceptions
│   │   │   │   │   │   ├── APIError.java  <-- Cấu trúc báo lỗi trả về từ REST API
│   │   │   │   │   │   ├── APIException.java <-- Runtime Exception phía REST Client sẽ throw khi nhận
│   │   │   │   │   │   └── TeamException.java <-- CoachService sẽ throw ra Exception này
│   │   │   │   │   ├── model
│   │   │   │   │   │   ├── Coach.java
│   │   │   │   │   │   ├── Player.java
│   │   │   │   │   │   ├── Position.java
│   │   │   │   │   │   ├── Substitute.java
│   │   │   │   │   │   └── TeamStatus.java
│   │   │   │   │   ├── repository
│   │   │   │   │   │   └── PlayerRepository.java
│   │   │   │   │   ├── restclient <-- Thư mục chưa khai báo Feign request và bộ giải mã lỗi
│   │   │   │   │   │   ├── APIErrorDecoder.java <-- Giải mã lỗi từ JSON string trong body thành APIErr
│   │   │   │   │   │   └── BarcaRequest.java <-- Khai báo các request bằng annotation của Feign
│   │   │   │   │   ├── service
│   │   │   │   │   │   └── CoachService.java
│   │   │   │   │   └── BarcelonateamApplication.java
│   ├── resources
│   │   └── application.properties
├── test
│   ├── java
│   │   ├── com
│   │   │   ├── onemount
│   │   │   │   └── barcelonateam
│   │   │   │       ├── FeignTest.java <-- Kiểm thử dùng OpenFeign
│   │   │   │       └── WebFluxClientTest.java <-- Kiểm thử dùng WebFlux WebClient
```

1. Cấu hình pom.xml Maven

Vào pom.xml bổ xung:

```
<dependency>
  <groupId>io.github.openfeign</groupId>
  <artifactId>feign-core</artifactId>
  <version>11.8</version>
</dependency>
<dependency>
  <groupId>io.github.openfeign</groupId>
  <artifactId>feign-gson</artifactId>
  <version>11.8</version>
</dependency>
<dependency>
  <groupId>io.github.openfeign</groupId>
  <artifactId>feign-okhttp</artifactId>
  <version>11.8</version>
</dependency>
```

Thư viện `feign-okhttp` là tuỳ chọn khi bạn muốn dùng OkHttp là thư viện network phía dưới cho OpenFeign.

2. Tạo ra một Interface khai báo các request

2.1 Các annotation Feign dùng để tạo ra request

Bình thường, khi tạo HTTP request, chúng ta thường sử dụng truyền tham số hoặc gán thuộc tính. Nhưng Feign lại dùng khai báo annotation.

- `@RequestLine` : định nghĩa loại request (GET, POST, PUT, PATCH, DELETE) và đường dẫn
- `@Param` : tham số URL. Kiểu như `/api/param1/param2` . Áp dụng phía trên khai báo hàm.
- `@QueryMap` : tham số key value ở Query string `/api?key1=value1&key2=value2` áp dụng ở tham số
- `@Headers` : thuộc tính cho header, áp dụng phía trên khai báo hàm trong interface
- `@HeaderMap` : thuộc tính cho header, truyền qua tham số của hàm
- `@Body` : khai báo cấu trúc tham số cho body.

Xem chi tiết ở đây https://github.com/OpenFeign/feign để hiểu rõ ý nghĩa của các annotation này.

2.2 Khai báo danh sách các Request

Xem file BarcaRequest.java chúng ta định nghĩa 2 request kiểu GET đường dẫn lần lượt là `/team` và `/chooseteam/{team_pattern}`

```
public interface BarcaClient {
  @RequestLine("GET /team")
  Set<Player> getTeam();

  @RequestLine("GET /chooseteam/{team_pattern}")
  Set<Player> chooseTeam(@Param("team_pattern") String team_pattern);
}
```

3. Khởi tạo Feign Client

Xem FeignTest.java

```
@TestInstance(Lifecycle.PER_CLASS)
public class FeignTest {
  private BarcaRequest barcaClient;

  @BeforeAll
  public void buildFeignClient() {
    final Decoder decoder = new GsonDecoder();  //Dùng để decode JSON string

    barcaClient = Feign
    .builder()
    .client(new OkHttpClient())  //Dùng OkHttp
    .decoder(decoder)
    .errorDecoder(new APIErrorDecoder(decoder)) //Xử lý lỗi
    .target(BarcaRequest.class, "http://localhost:8080"); //Truyền vào Interface liệt kê
  }
}
```

4. Viết hàm kết nối và lấy dữ liệu

```
@Test
@Order(1)
@DisplayName(" .1 GET /team must return 11 players")
void getTeam() {
  Set<Player> players = barcaClient.getTeam();
  assertThat(players).hasSize(11);

  long goalKeeperCount = players.stream()
  .filter(p -> p.getPosition() == Position.GK)
  .count();
  assertThat(goalKeeperCount).isEqualTo(1);
}
```

5. Bắt lỗi khi REST trả về

Xem APIErrorDecoder.java

```
public class APIErrorDecoder implements ErrorDecoder {
  final Decoder decoder;
  final ErrorDecoder defaultDecoder = new ErrorDecoder.Default();

  public APIErrorDecoder(Decoder decoder) {
    this.decoder = decoder;
  }

  @Override
  public Exception decode(String methodKey, Response response) {
    try {
      //Cố gắng parse JSON trả về và chuyển thành đối tượng APIError
      APIError apiError = (APIError) decoder.decode(response, APIError.class);

      //Tạo ra APIException để throw
      return new APIException(HttpStatus.valueOf(response.status()),
      apiError.message, apiError.details);

    } catch (final IOException fallbackToDefault) {
      return defaultDecoder.decode(methodKey, response);
    }
  }
}
```

Ở đây chúng ta dùng `try catch` để bắt `APIException`

```
try {
  barcaClient.chooseTeam("118");
} catch (APIException e) {
  assertThat(e.getMessage()).isEqualTo("TeamException : Request players more than availa
}
```

Kết luận

Bạn có thể thay thế WebFlux WebClient bằng OpenFeign + OkHttpClient. Ưu điểm dễ nhận thấy:

1. Code viết ngắn và dễ hiểu hơn hẳn
2. Thay vì phải lập trình, cài đặt thuộc tính phức tạp, OpenFeign để lập trình viên khai báo cấu trúc request
3. Cú pháp của OpenFeign rất mạnh vừa dễ kế thừa mà vừa dễ tuỳ biến

Bài tập lập trình tại lớp

Hãy viết bổ xung các Test Case sử dụng OpenFeign

1. Số lượt thay cầu thủ không được phép vượt quá 5.
2. Không được thay cầu thủ có số áo không phải cầu thủ đang đá.
3. Báo lỗi khi vị trí cần thay vào không còn cầu thủ đáp ứng.