Migwel

Zeste de savoir

Déserialiser une API variable en Java avec Jackson

29 novembre 2020

Table des matières

Depuis le début du confinement, les échecs ont connu un fameux regain en popularité et le site de référence quand vous voulez apprendre et jouer aux échecs est chess.com . Bien que leur site est plutôt bien conçu, leur API contient quelques défauts et l'un des plus énervants quand on veut l'utiliser est le manque de cohérence dans les réponses retournées. Prenons l'exemple des participants à une partie (white et black). Selon leur documentation , si vous souhaitez voir les parties en cours pour un joueur, voici le format de la réponse :

```
{
  "games": [
    {
      "white": "string", // URL of the white player's profile
      "black": "string", // URL of the black player's profile
      ...
    }
  ]
}
```

Et maintenant, si vous souhaitez voir les parties jouées et terminées durant un certain mois, voici ce que ça donne :

```
{
  "games": [
    {
      "white": { // details of the white-piece player:
          "username": "string", // the username
          "rating": 1492, // the player's rating at the start of
              the game
          "result": "string", // see "Game results codes" section
          "@id": "string", // URL of this player's profile
      },
      "black": { // details of the black-piece player:
          "username": "string", // the username
          "rating": 1942, // the player's rating at the start of
              the game
          "result": "string", // see "Game results codes" section
          "@id": "string", // URL of this player's profile
```

```
      },
      ...
    }
  ]
}
```

Bon, d'accord, c'est un peu ennuyant mais on peut facilement gérer ce genre de cas, il nous suffit d'utiliser des objets différents selon l'appel utilisé mais rien dramatique.

Là où ça se corse, c'est que pour certains appels, les deux formats sont utilisés. Parfois vous recevrez une chaine de caractères, parfois un objet complet. Que faire ?

La première étape est de se plaindre parce que ça fait toujours du bien. La deuxième, c'est de regarder les possibilités offertes par le framework de désérialization que nous utilisons.

1. Introduction à Jackson

Dans mon cas, j'utilise Jackson. Si vous ne connaissez pas, Jackson est l'un des frameworks les plus utilisés dans le monde Java pour sérialiser des objets de Java vers JSON (ou XML, ou Avro, ou protobuf, …) ou désérialiser de JSON vers Java. C'est utile quand vous devez par exemple stocker des objets en base de données et encore plus lorsque vous interagissez avec un API qui retourne des réponses au format JSON.

Voici un exemple très simple :

```java
public class NameDeserializationTest {

    private static record Name(String firstName, String lastName){}

    @Test
    void testSerializeSimpleObject() throws JsonProcessingException
        {
        ObjectMapper objectMapper = new JsonMapper();
        Name name = new Name("Donald", "Duck");

            assertEquals("{\"firstName\":\"Donald\",\"lastName\":\"Duck\"}",
            objectMapper.writeValueAsString(name));
    }

    @Test
    void testDeserializeSimpleObject() throws
        JsonProcessingException {
        ObjectMapper objectMapper = new JsonMapper();
        String json =
            "{\"firstName\":\"Donald\",\"lastName\":\"Duck\"}";
        Name name = objectMapper.readValue(json, Name.class);
        assertEquals("Donald", name.firstName);
```

```
18          assertEquals("Duck", name.lastName);
19
20      }
21  }
```

Très bien, Jackson fonctionne et est simple d'utilisation, mais que faire lorsque le json qu'on reçoit change de format de temps en temps ? Écrivons notre propre méthode de désérialisation.

2. Désérialisation maison

Premièrement, nous allons créer une nouvelle classe fille de `StdDeserializer` et redéfinir la méthode `deserialize`.

```
1   public class ParticipantDeserializer extends
        StdDeserializer<Participant> {
2
3       private final static String URL_REGEXP =
            "https:\\/\\/api.chess.com\\/pub\\/player\\/(.+)";
4       private final static Pattern urlPattern =
            Pattern.compile(URL_REGEXP);
5   ...
6
7       @Override
8       public Participant deserialize(JsonParser jsonParser,
            DeserializationContext ctxt) throws IOException {
9           String participantUrl = jsonParser.getValueAsString();
10          if (participantUrl == null) {
11              return jsonParser.readValueAs(Participant.class);
12          }
13
14          Matcher matcher = urlPattern.matcher(participantUrl);
15          String username = participantUrl;
16          if (matcher.find()) {
17              username = matcher.group(1);
18          }
19          return new Participant(username, participantUrl);
20      }
21  }
```

Dans notre application, quoi qu'on reçoive de chess.com , nous voulons toujours créer un objet de type `Participant` qui est composé d'un nom d'utilisateur et d'un url. On vérifie d'abord si ce qu'on a reçu de chess.com est une simple chaine de caractères. Si ce n'est pas le cas, on n'a rien de spécial à faire et on peut laisser le framework faire le travail.

Par contre, si on a reçu une chaine de caractère, on va construire une instance de Participant "à la main". J'ai ajouté un peu de logique car la chaine de caractère semble toujours être du

format `https://api.chess.com/pub/player/username` donc j'ai ajouté un peu de logique qui essaie d'extraire le nom d'utilisateur hors de l'url. Si cela ne fonctionne pas, tant pis et on utiliser l'URL en tant que nom d'utilisateur (cette décision est purement arbitraire ; on pourrait aussi imaginer laisser le champ nom d'utilisateur vide).

Maintenant que notre *deserializer* est créé, il ne nous reste plus qu'à ajouter une annotation pour informer Jackson de l'utiliser sur les champs white et black en utilisant `@JsonDeserialize`.

```
1 public record GroupGame(@JsonProperty("white")
      @JsonDeserialize(using = ParticipantDeserializer.class)
      Participant white,
2                       @JsonProperty("black")
                            @JsonDeserialize(using =
                            ParticipantDeserializer.class)
                            Participant black,
3                       ...)
4 {}
```

L'annotation `@JsonProperty` permet de définir le nom du champ dans le message JSON. Une autre annotation sympa de Jackson !

J'espère que ceci pourra aider certains d'entre vous qui doivent utiliser des API irrégulières. Le code présenté ci-dessus est disponible sur Github et plus particulièrement, ce commit qui a été créé pour résoudre ce problème.