Custom JSON Deserialization with Jackson

by A. N. M. Bazlur Rahman MVB · Dec. 17, 15 · Integration Zone · Tutorial

We consume a REST API as a JSON format and then unmarshal it to a POJO. Jackson’s org.codehaus.jackson.map.ObjectMapper “just works” out of the box and we really don’t do anything in most cases. But sometimes we need a custom dserializer to fulfill our custom needs and this tutorial will guide you through the process of creating your own custom dserializer.

Let’s say we have following entities:

```java
public class User {
private Long id;
private String name;
private String email;

public Long getId() {
return id;
}

public User setId(Long id) {
this.id = id;
return this;
}

public String getName() {
return name;
}

public User setName(String name) {
this.name = name;
return this;
}

public String getEmail() {
return email;
}

public User setEmail(String email) {
this.email = email;
return this;
}

@Override
public String toString() {
final StringBuffer sb = new StringBuffer("User{");
sb.append("id=").append(id);
sb.append(", name='").append(name).append('\'');
sb.append(", email='").append(email).append('\'');
sb.append('}');
return sb.toString();
}
}
```

```java
public class Program {
private Long id;
private String name;
private User createdBy;
private String contents;

  public Program(Long id, String name, String contents, User createdBy) {
    this.id = id;
    this.name = name;
    this.contents = contents;
    this.createdBy = createdBy;
  }

public Program() {}

public Long getId() {
return id;
}

  public Program setId(Long id) {
    this.id = id;
    return this;
  }

  public String getName() {
  return name;
  }

  public Program setName(String name) {
    this.name = name;
    return this;
  }

  public User getCreatedBy() {
  return createdBy;
  }

  public Program setCreatedBy(User createdBy) {
    this.createdBy = createdBy;
    return this;
  }

  public String getContents() {
  return contents;
  }

  public Program setContents(String contents) {
  this.contents = contents;
  return this;
  }

  @Override
  public String toString() {
    final StringBuffer sb = new StringBuffer("Program{");
    sb.append("id=").append(id);
    sb.append(", name='").append(name).append('\'');
    sb.append(", createdBy=").append(createdBy);
    sb.append(", contents='").append(contents).append('\'');
    sb.append('}');
    return sb.toString();
  }
}
```

Let’s serialize/marshal an object first:

```
User user = new User();
user.setId(1L);
user.setEmail("example@example.com");
user.setName("Bazlur Rahman");

Program program = new Program();
program.setId(1L);
program.setName("Program @# 1");
program.setCreatedBy(user);
program.setContents("Some contents");

ObjectMapper objectMapper = new ObjectMapper();

final String json = objectMapper.writeValueAsString(program);
System.out.println(json);
```

The above code will produce the following JSON:

```
{
"id": 1,
"name": "Program @# 1",
"createdBy": {
"id": 1,
"name": "Bazlur Rahman",
"email": "example@example.com"
},
"contents": "Some contents"
}
```

Now can do the opposite very easily. If we have this JSON, we can unmarshall to a program object using ObjectMapper as following:

```
String jsonString = "{\"id\":1,\"name\":\"Program @# 1\",\"createdBy\":{\"id\":1,\"name\":\"Bazlur Rahman\",\"email\":\"exa


final Program program1 = objectMapper.readValue(jsonString, Program.class);
 System.out.println(program1);
```

Now let’s say, this is not the real case, we are going to have a different JSON from an API which doesn’t match with our program class:

```
{
"id": 1,
"name": "Program @# 1",
"ownerId": 1
"contents": "Some contents"
}
```

Look at the json s it has a different field that is *ownerId*.

Now if you want s as we did earlier, you will have exceptions.

There are two wa ns and have this seralized: ignore the unknown fields and write a custom deserializer.

Ignore the Unknown Fields

Ignore the `*onwerId.*` Add following annotation in the Program class

```
@JsonIgnoreProperties(ignoreUnknown = true)
public class Program {}
```

Write Custom Deserializer

But there are cases when you actually need this `owerId` field. Lets say you want to relate it as as an id of the user class.

In such case, you need to write a custom deserializer:

```java
import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.core.ObjectCodec;
import com.fasterxml.jackson.databind.DeserializationContext;
import com.fasterxml.jackson.databind.JsonDeserializer;
import com.fasterxml.jackson.databind.JsonNode;

import java.io.IOException;

public class ProgramDeserializer extends JsonDeserializer<Program> {
  @Override
  public Program deserialize(JsonParser jp, DeserializationContext ctxt) throws IOException, JsonProcessingException {
    ObjectCodec oc = jp.getCodec();
    JsonNode node = oc.readTree(jp);

    final Long id = node.get("id").asLong();
    final String name = node.get("name").asText();
    final String contents = node.get("contents").asText();
    final long ownerId = node.get("ownerId").asLong();

    User user = new User();
    user.setId(ownerId);

    return new Program(id, name, contents, user);
  }
}
```

As you can see, first you have to access the JsonNode from the JonsParser. And then you can easily extract information from a JsonNode using get method. and you have to be make sure about the field name. It should be the exact name, spelling mistake will cause exceptions.

And finally you have to register your ProgramDeserializer to the `*ObjectMapper*`.

```java
ObjectMapper mapper = new ObjectMapper();
SimpleModule module = new SimpleModule();
module.addDeserializer(Program.class, new ProgramDeserializer());

mapper.registerModule(module);

String newJsonString = "{\"id\":1,\"name\":\"Program @# 1\",\"ownerId\":1,\"contents\":\"Some contents\"}";
final Program program2 = mapper.readValue(newJsonString, Program.class);
```

Alternatively you can use annotation to register the deserializer directly:

```java
@JsonDeserialize(using = ProgramDeserializer.<b>class</b>)
public class Program {}
```

Full source code can be found in : https://github.com/rokon12/json-deserializer

Like This Article? Read More From DZone

Jackson Mixin to the Rescue

JSON processing using Jackson Java JSON Processor

Dynamically Filter JSON With Jackson and Squiggly

Free DZone Refcard
RESTful API Lifecycle Management

Topics: JAVA , JSON , JACKSONJSON , JACKSON , DESERIALIZATION

Published at DZone with permission of A. N. M. Bazlur Rahman , DZone MVB. See the original article here.

Integration Partner Resources

The CRM Integration Guide: 8 Things Experts Are Considering in Their API Strategy
Cloud Elements

How event streaming can benefit API design.
Capital One

Modernizing Application Architectures with Microservices and APIs
CA Technologies

A Comprehensive Guide to the Enterprise Integration Cloud
SnapLogic