AULA 09 - JSON E SERIALIZAÇÃO

Disciplina de Backend - Professor Ramon Venson - SATC 2024

Serialização

Serialização é o processo computacional de converter um objeto na memória do sistema para um formato (bytes ou texto) que possa ser armazenado ou transferido para outro sistema.

Deserialização é o inverso, onde um dado serializado é transformado em um objeto na memória do sistema.

JSON

O JSON (Javascript Object Notation) é um formato de troca de dados em modo texto baseado na notação de objetos do javascript. Ele é utilizado atualmente em muitas APIs na Web para fornecer e receber dados em um formato padronizado e de fácil desconstrução.

Por que usar JSON?

1. Objetos que estão na memória são incompatíveis com outras aplicações/sistemas
2. É necessário um formato que possa ser reconhecido por diferentes plataformas
3. O formato precisa de um custo computacional baixo (para serializar a deserializar)

> Mesmo que o JSON seja menos verboso que outros formatos como o XML, o processo de serialização e deserialização ainda tem um custo considerável para alguns casos.

Características do JSON

- Significa Java Script Object Notation;
- É um formato para troca de dados;
- Independente de linguagem de programação;
- Sintaxe simples e textual;

Exemplo JSON

```
{
    "usuarios": {
        "john": {
            "email": "john@matrix.com",
            "senha": "12345"
        },
        "mary": {
            "email": "m.mary@bol.com.br",
            "senha": "abc123",
            "admin": true
        }
    }
}
```

Sintaxe JSON

O formato JSON é submetido às seguintes regras:

- Os dados são encapsulados em pares (`chave: valor`);
- Os dados são separados por `,` (virgula);
- Um par chave/valor é encapsulado por `{ }` (chaves);
- Vetores são encapsulados por `[ ]` (conchetes);
- As chaves são sempre circundadas por `" "` (aspas duplas).

Um dado vazio no formato JSON é dado pela seguinte sintaxe:

```
{}
```

Um par de chave/valor pode ser descrito como o seguinte exemplo:

```
{ "usuarios" : "john e mary" }
```

Tipos de Dados JSON

O JSON suporta os seguintes tipos de dados:

- Strings (ex.: `"cem"`);
- Números (ex.: `100`, `100.0`, `1.0E+2`, `1E+2`);
- Objeto (ex: `{ "cem" : 100 }` um objeto JSON);
- Vetor (ex: `[100, 100.1, "cem"]`);
- Booleano (ex.: `true` ou `false`);
- Nulo (ex.: `null`).

Valores que não são considerados tipos de dados no JSON:

- Funções;
- Datas;
- `void`

Mapeamento JSON

O Spring Boot possui integração automática com bibliotecas de mapeamento JSON, como o Jackson, GSON e JSON-B. Por padrão, utilizamos o Jackson para realizar a serialização/deserialização implícita de objetos.

Para tal, utilizaremos a anotação `@RequestBody` e o controlador `ResponseEntity`.

Requisições

Imagine que você deseja receber em uma requisição HTTP um tipo específico de dado, como usuário e senha para login na aplicação. Essas informações serão enviadas pelo cliente usando o formato JSON:

```
{ "usuario": "ronaldinho", "senha" : "bruxo123" }
```

Para receber e processar esses dados num controller do Spring, vamos precisar utilizar o `@RequestBody` juntamente com um parâmetro, que deve conter os mesmos atributos esperados na requisição.

```
public class LoginController {
  @PostMapping("/login")
  public ResponseEntity<Object> login(@RequestBody Login login) {
    Boolean isUsuarioCorreto = login.getUsuario().equals("ronaldinho");
    Boolean isSenhaCorreta = login.getSenha().equals("bruxo123");
    if (isUsuarioCorreto && isSenhaCorreta) {
      return ResponseEntity.status(200).body("Login realizado com sucesso");
    }
    return ResponseEntity.status(401).body("Senha ou usuário incorreto");
  }
}
```

A classe a ser usada no mapeamento neste caso será um POJO:

```
public class Login {
  private String usuario;
  private String senha;

  // construtores, getters e setters...
}
```

O Spring então fará o mapeamento automático do corpo da resposta para um objeto.

Respostas

Para responder, por padrão, o Spring também fará a conversão implícita de um objeto para o padrão JSON, caso estejamos utilizando a anotação `@RestController` no controlador e a classe `ResponseEntity` para a resposta.

```
public class GeradorJogadorController {
  @PostMapping("/jogador")
  public ResponseEntity<Object> gerador() {
    Jogador jogador = new Jogador();
    return ResponseEntity.status(200).body(jogador);
}
```

> Nesse caso, também incluímos o código de status usando o método status, que representa um status code HTTP.

O que aprendemos hoje

- O que é e como representar objetos usando o formato JSON;
- Como receber dados JSON no corpo de uma requisição;
- Como responder uma requisição no formato JSON.