04

Serviço com JSON

Transcrição

Olá! Nesse vídeo iremos implementar nosso primeiro serviço, trabalhando com HTTP, JSON e XML.

Quando precisamos de um protocolo HTTP, já sabemos que um Servlet atenderá nossas necessidades, pois com ele conseguimos executar uma requisição através do protocolo HTTP.

Para criarmos nosso primeiro Webservice com JSON, poderíamos aproveitar a estrutura MVC que construímos nesse curso. Inclusive, usuários de Spring MVC conseguiriam devolver tanto HTML quanto JSON, pois esse controlador já tem essa funcionalidade embutida.

No entanto, precisaríamos mudar um pouco a estrutura da nossa aplicação, pois existem algumas coisas que nosso controlador não seria capaz de fazer. Por exemplo, a classe `ListaEmpresas` faz um *forward* para chamar um `.jsp`. Para trabalharmos com JSON, seria necessário sinalizar, de alguma forma, para o controlador devolver um `json`.

Como queremos economizar tempo, já que existem controladores que já têm essa funcionalidade, criaremos um novo Servlet clicando com o botão direito no pacote `servlets` e em seguida em "New > Servlet".

Chamaremos esse Servlet de `EmpresasService`. Na próxima página, mudaremos a URL para `/empresas` - dessa forma, não passaremos pelo nosso

controlador. O método que daremos suporte é o `service()` (que dá suporte a `doGet()`, `doPost()` e qualquer requisição). Assim, teremos:

```
@WebServlet("/empresas")
public class EmpresasService extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void service(HttpServletRequest request, HttpServ

    }

}
```

COPIAR CÓDIGO

Nesse Servlet queremos devolver as empresas em um formato genérico. Para isso, primeiro precisaremos listar as empresas, o que é feito a partir da nossa classe `Banco`, na qual temos um método `getEmpresas()`. Esse método nos devolve uma lista de empresas:

```
package br.com.alura.gerenciador.servlet;

import java.io.IOException;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.alura.gerenciador.modelo.Banco;
import br.com.alura.gerenciador.modelo.Empresa;

@WebServlet("/empresas")
public class EmpresasService extends HttpServlet {
```

```
    private static final long serialVersionUID = 1L;

    protected void service(HttpServletRequest request, HttpSer

        List<Empresa> empresas = new Banco().getEmpresas();
    }

}
```

COPIAR CÓDIGO

Na página https://json.org/example.html (https://json.org/example.html) podemos encontrar exemplos de JSON, como esse:

```
{
    "glossary": {
        "title": "example glossary",
        "GlossDiv": {
            "title": "S",
            "GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
                    "SortAs": "SGML",
                    "GlossTerm": "Standard Generalized Markup
                    "Acronym": "SGML",
                    "Abbrev": "ISO 8879:1986",
                    "GlossDef": {
                        "para": "A meta-markup language, used
                        "GlossSeeAlso": ["GML", "XML"]
                    },
                    "GlossSee": "markup"
                }
            }
        }
    }
}
```

COPIAR CÓDIGO

Algumas convenções clássicas do JSON (*JavaScript Object Notation*) são as chaves `{}` e os atributos, que vêm entre aspas duplas `" "`. Por exemplo, ao atributo `"glossary"` é atribuído um novo JSON. Nele, temos o atributo `"title"`, cujo valor é `"example gossary"`.

Na verdade, esse código é uma anotação para definir um objeto Java que tem esses atributos. Por isso o JSON é tão famoso com bibliotecas como Angular e React, já que elas conseguem transformar esse código em um objeto JavaScript.

Nesse momento, precisamos de uma biblioteca do mundo Java que saiba trabalhar com JSON. A ideia é passarmos nossa lista de Empresas para essa biblioteca gerar um JSON, pois não queremos fazer essa concatenação de *strings* manualmente.

Existem muitas bibliotecas que têm essa funcionalidade. Nesse curso, usaremos o GSON (https://github.com/google/gson). No Guia de Uusário dessa biblioteca ("*User Guide*") encontramos um exemplo dessa concatenação:

```
class BagOfPrimitives {
  private int value1 = 1;
  private String value2 = "abc";
  private transient int value3 = 3;
  BagOfPrimitives() {
    // no-args constructor
  }
}

// Serialization
BagOfPrimitives obj = new BagOfPrimitives();
Gson gson = new Gson();
String json = gson.toJson(obj);

// ==> json is {"value1":1,"value2":"abc"}
```

COPIAR CÓDIGO

Repare que temos uma classe chamada `gson`. A partir dela, chamamos o método `toJson()` e passamos o objeto `obj`. Isso é exatamente o que queremos fazer na nossa aplicação:

```
List<Empresa> empresas = new Banco().getEmpresas();

Gson gson = new Gson();
String json = gson.toJson(empresas);
```

COPIAR CÓDIGO

Porém, nosso código evidentemente não irá copiar, já que essa é uma biblioteca externa e não vêm nativamente no Java. Precisaremos importá-la no nosso projeto. Para facilitar, você pode baixá-la diretamente aqui (https://caelum-online-public.s3.amazonaws.com/1001-servlets-parte2/06/gson-2.8.5.jar.zip).

Agora basta copiar o arquivo `gson-2.8.5.jar` extraído desse `.zip` para dentro da pasta "WEB-INF/lib", na qual já temos o arquivo `jstl-1.2.jar`. O Eclipse reconhecerá a biblioteca automaticamente, portanto só precisaremos importar `com.google.gson.Gson` no nosso `EmpresasService` ("Ctrl + Shift + O").

Para devolvermos o JSON, usaremos `response.getWriter().print(json)` - um método que devolve uma string e imprime na saída o JSON.

```
List<Empresa> empresas = new Banco().getEmpresas();

Gson gson = new Gson();
String json = gson.toJson(empresas);

response.getWriter().print(json);
```

COPIAR CÓDIGO

Quando o navegador ou o celular fazem uma chamada para nosso *Web Service*, eles esperam, na resposta, um cabeçalho específico que define qual o conteúdo dessa resposta. Esse conteúdo pode variar: pode ser um HTML, um JSON, um XML ou mesmo um PDF, dependendo do que escrevemos na resposta.

Para implementarmos um serviço bem refinado, iremos definir na resposta que o usuário está recebendo um JSON. Isso não era necessário quando estávamos trabalhando com JSP, pois ele faz isso de maneira automática.

Nesse caso, teremos que escrever a resposta manualmente:

```
List<Empresa> empresas = new Banco().getEmpresas();

Gson gson = new Gson();
String json = gson.toJson(empresas);

response.setContentType("application/json");
response.getWriter().print(json);
```

COPIAR CÓDIGO

Aqui estamos definindo o tipo de conteúdo na resposta. Existem algumas regras em relação a como devolver cada tipo - no caso, para JSON, utilizamos `application/json`. Se estivéssemos trabalhando com um HTML, seria `text/html`.

E... estamos prontos. Muito simples, não é? Vamos testar acessando a URL http://localhost:8080/gerenciador/empresas (http://localhost:8080/gerenciador/empresas). Como resposta, teremos:

```
[{"id":1,"nome":"Alura","dataAbertura":"Oct 31, 2018, 3:06:28 PM"},
{"id":2,"nome":"Caelum","dataAbertura":"Oct 31, 2018, 3:06:28 PM"}]
```

Repare que isso é exatamente um JSON: temos a primeira empresa entre chaves, uma vírgula e depois a segunda empresa também entre chaves. Esses

dois objetos JavaScript estão dentro de um array de dois elementos(que é representado pelos colchetes `[]`). Dentro dos objetos JavaScript temos o elemento `id` com o valor `1`, o nome da empresa e a data de abertura.

Os dados foram renderizados automaticamente dessa forma, porém, assim como no HTML, é possível manipular o formato em que eles são apresentados. Por enquanto vamos mantê-los assim, pois já atendem às nossas necessidades.

No próximo vídeo trabalharemos com XML. Não se esqueça de resolver os exercícios e até lá!