10

Usando a taglib fmt

Transcrição

Nesta aula estudaremos a sub-biblioteca **fmt** do JSTL (*Java Standard Tag Library*), que lida especificamente com formatação. Obviamente existem muitos conteúdos ainda sobre a **core**, mas aqui aprenderemos o essencial para depois realizarmos um curso específico sobre o assunto, inclusive de como podemos criar nossas próprias tags.

Vamos refletir sobre algumas melhorarias em nosso formulário de cadastramento: seria interessante que, ao cadastrarmos uma empresa, surgisse a data em que esse cadastro foi realizado, isto é, uma data de abertura que será exibida na página de lista de empresas do banco de dados. Para isso, usaremos o fmt.

Começaremos trabalhando em `listaEmpresas.jsp`. No momento do código que realizamos o laço e é renderizado o elemento da lista, iremos incluir também a data, utilizando novamente a expression language. Agora, se a empresa tem um nome, ela também precisa fornecer a data de cadastramento.

```
<body>
    Lista de empresas: <br />

    <ul>
        <c:forEach items="${empresas}" var="empresa">
            <li>${empresa.nome } ${empresa.dataAbertura }</li>
        </c: forEach>
    </ul>
</body>
```

COPIAR CÓDIGO

Na classe `Empresa`, devemos pensar em como representar a data de cadastro. Em seguida, usaremos o `Date` - apesar de existir uma API mais nova que poderíamos usar nesse caso, `Date` combina melhor com fmt. Chamaremos esse atributo de `dataAbertura`.

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

import java.util.Date;

public class Empresa {

    private Integer id;
    private String nome;
    private Date dataAbertura;
```

COPIAR CÓDIGO

Ao final do código, criaremos os *getters* e *setters*, isto é, os métodos auxiliares para acessarmos o atributo.

```
    public void setNome(String nome) {
        this.nome = nome;
    }

    public void setDataAbertura(Date dataAbertura) {
        this.dataAbertura = dataAbertura;

    public Date getDataAbertura() {
        return dataAbertura;
    }
}
```

COPIAR CÓDIGO

Dessa forma `Empresa` já conhece o novo valor que criamos. Para podermos fazer um teste rápido com esse novo dado, iremos inicializar diretamente a `Date`, utilizando as duas empresas que já temos cadastradas em nosso banco de dados.

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

import java.util.Date;

public class Empresa {

    private Integer id;
    private String nome;
    private Date dataAbertura = new Date();
```

COPIAR CÓDIGO

Reiniciaremos o servidor. Já no navegador, por meio da URL `localhost:8080/gerenciador/listaEmpresas`, acessaremos a lista de empresas cadastradas com as respectivas datas de abertura:

```
Lista de empresas:

.Alura Fri Aug 24 15:29:36 BRT 2018
.Caelum Fri Aug 24 15:29:36 BRT 2018
```

COPIAR CÓDIGO

Ainda são utilizadas as saídas padrão de data, portanto precisamos ainda formatar essa informação, e para isso usaremos a tag fmt. Em `listaEmpresas`, faremos a importação dessa sub-biblioteca e utilizaremos o prefixo `fmt`.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fm
```

```
<!DOCTYPE html>
```

COPIAR CÓDIGO

Feita a importação, dentro de `<c:forEach>`, escreveremos a tag `<fmt:FormateDate>`, que recebe um valor. Neste caso, ele será justamente `${empresa.dataAbertura}`. Passaremos esse conteúdo para dentro da tag `<li>`, de forma que o código ficará com a seguinte forma:

```
<body>
    Lista de empresas: <br />

    <ul>
        <c:forEach items="${empresas}" var="empresa">
            <li>${empresa.nome} <fmt:formatDate value="${empres
        </c: forEach>
    </ul>
</body>
```

COPIAR CÓDIGO

Ao acessarmos a página no navegador, veremos que a data está com uma formatação mais elegante, inclusive em português:

> Lista de empresas:
>
> Alura 24 de ago de 2018
>
> Caelum 24 de ago de 2018

Contudo, podemos nós mesmos podemos definir o formato das datas. Inseriremos um `-` (hífen) entre o nome e a data, e depois utilizaremos o atributo `pattern`, com o qual definiremos o formato `dd/MM/yyyy`.

```
<body>
    Lista de empresas: <br />

    <ul>
        <c:forEach items="${empresas}" var="empresa">
            <li>${empresa.nome} - <fmt:formatDate value="${empr
        </c: forEach>
    </ul>
</body>
```

COPIAR CÓDIGO

Dessa forma, no navegador, a lista será exibida da seguinte maneira:

> Lista de empresas:
>
> Alura - 24/08/2018
>
> Caelum - 24/08/2018

A data exibida é aquela em que estamos gravando este curso, mas gostaríamos de definir datas diferentes no momento de cadastrar uma nova empresa. Para isso, faremos algumas modificações em nosso formulário.

Trabalharemos agora em `formNovaEmpresa.jsp` , que atualmente se encontra com este formato:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %
<c:url value="/novaEmpresa" var="linkServletNovaEmpresa"/>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
```

```
<title>Insert title here</title>
</head>
<body>

    <form action="${linkServletNovaEmpresa}" method="post">

        Nome: <input type="text" name="nome" />

        <input type="submit" />
    </form>

</body>
</html>
```

COPIAR CÓDIGO

Iremos adicionar uma linha que contém a tag `<input>`, que começará com `Data Abertura`. O parâmetro desse input será `data` e manteremos o tipo `text`.

```
    <form action="${linkServletNovaEmpresa}" method="post">

        Nome: <input type="text" name="nome" />
        Data Abertura: <input type="text" name="data" />

        <input type="submit" />
    </form>

</body>
</html>
```

COPIAR CÓDIGO

Ao acessarmos a página http://localhost:8080/gerenciador/formNovaEmpresa.jsp (http://localhost:8080/gerenciador/formNovaEmpresa.jsp), teremos dois campos no formulário: "Nome" e "Data Abertura". Preencheremos os campos

com "Facebook" e "11/01/2001", respectivamente. Ao enviarmos a requisição, precisamos que o parâmetro seja lido no Servlet `novaEmpresa`.

Acessaremos `NovaEmpresaServlet.java` e escreveremos uma nova linha de código responsável pela leitura do novo parâmetro `data`:

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

  protected void doPost(HttpServletRequest request, HttpServletR
          throws ServletException, IOException {
      System.out.println("Cadastrando nova empresa");

      String nomeEmpresa = request.getParameter("nome");
      String paramDataEmpresa = request.getParameter("data");
```

COPIAR CÓDIGO

Contudo, teremos um pequeno problema: o método `getParameter()` sempre devolve uma string, independentemente do que estamos enviando. Em `Empresa.java` tratamos `dataAbertura` com um tipo específico, o `Date`. Portanto, é necessário que transformemos a string neste `Date` específico.

O nome desse tipo de transformação é ***parsing***, e utilizamos uma classe específica para realizá-la, a `SimpleDateFormat`. Essa classe recebe, como podemos verificar no construtor, uma string. Ao analisarmos a documentação sobre a classe SimpleDateFormat (https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html), veremos que há um construtor que recebe uma string e pattern, justamente aquele que definimos no fmt, isto é, `dd/MM/yyyy`.

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

  protected void doPost(HttpServletRequest request, HttpServle
```

```
        throws ServletException, IOException {
    System.out.println("Cadastrando nova empresa");

    String nomeEmpresa = request.getParameter("nome");
    String paramDataEmpresa = request.getParameter("data");

    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
```

COPIAR CÓDIGO

Dessa forma, estamos delimitando o formato de envio da data de cadastramento, fazendo que o parsing funcione do lado do servidor.

Temos o objeto que realiza o parsing da string, só resta utilizá-lo. Para isso, escreveremos `sdf` mais o método `parse()`, que receberá como parâmetro `paramDataEmpresa`. É importante destacar que o método `parse()` devolve uma `Date`, portando escreveremos `Date dataAbetura = sdf.parse(paramDataEmpresa)`. Por fim, realizaremos a importação de `Date(java.util)`.

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

  protected void doPost(HttpServletRequest request, HttpServletR
        throws ServletException, IOException {
    System.out.println("Cadastrando nova empresa");

    String nomeEmpresa = request.getParameter("nome");
    String paramDataEmpresa = request.getParameter("data");

    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
    Date dataAbertura = sdf.parse(paramDataEmpresa);
```

COPIAR CÓDIGO

O Eclipse ainda irá apontar um erro, pois o método `parse()` joga uma exceção `ParseException`, portanto é preciso fazer um tratamento. Poderíamos tentar escrever essa exceção na parte superior do código, na linha de `throws`, mas isso não é permitido, afinal o método `doPost()` está sobrescrevendo a classe Servlet, e sua assinatura não pode ser modificada, pois a herança não permite uma nova exceção.

A única forma de resolvermos essa questão é utilizando um `try/catch`.

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

protected void doPost(HttpServletRequest request, HttpServletR
        throws ServletException, IOException {
    System.out.println("Cadastrando nova empresa");

    String nomeEmpresa = request.getParameter("nome");
    String paramDataEmpresa = request.getParameter("data");

    try {
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyy
        Date dataAbertura = sdf.parse(paramDataEmpresa);
    } catch (ParseException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

COPIAR CÓDIGO

Temos uma pergunta importante a ser feita: caso ocorra um erro no processo de parsing, é interessante continuar o processo de cadastro no Servlet? Isto é, faz sentido criarmos uma empresa sem data de abertura? Aparentemente não. Portanto, iremos continuar lançando exceções nesta parte do código, e criaremos um objeto do tipo `ServletException`. Para não perdermos de vista o problema original da exceção, adicionaremos `e` em `ServletException`, isto é, a exceção original.

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

    protected void doPost(HttpServletRequest request, HttpServletR
            throws ServletException, IOException {
        System.out.println("Cadastrando nova empresa");

        String nomeEmpresa = request.getParameter("nome");
        String paramDataEmpresa = request.getParameter("data");

        try {
            SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyy
            Date dataAbertura = sdf.parse(paramDataEmpresa);
        } catch (ParseException e) {
                throw new ServletException(e);
        }
```

COPIAR CÓDIGO

Esse padrão é conhecido por *catch and rethrow*, isto é, capturar e relançar a exceção. É exatamente isso que estamos fazendo. O *parsing* está pronto, e agora resta preenchermos a empresa com a data de abertura. Logo, escreveremos `empresa.setDataAbertura()` passando `dataAbertura`

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

    protected void doPost(HttpServletRequest request, HttpServletR
            throws ServletException, IOException {
        System.out.println("Cadastrando nova empresa");

        String nomeEmpresa = request.getParameter("nome");
        String paramDataEmpresa = request.getParameter("data");

        try {
            SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyy
            Date dataAbertura = sdf.parse(paramDataEmpresa);
        } catch (ParseException e) {
```

```
            throw new ServletException(e);
        }

        Empresa empresa = new Empresa();
        empresa.setNome(nomeEmpresa);
        empresa.setDataAbertura(dataAbertura);
```

COPIAR CÓDIGO

A compilação não está ocorrendo, pois `dataAbertura` é uma variável local, que apenas existe dentro do seu próprio bloco. Apagaremos o título `Data` e declararemos a variável fora do bloco, e ela irá inicializar com o valor `null`.

```
        Date dataAbertura = null;
        try {
            SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyy
            dataAbertura = sdf.parse(paramDataEmpresa);
        } catch (ParseException e) {
                throw new ServletException(e);
        }

        Empresa empresa = new Empresa();
        empresa.setNome(nomeEmpresa);
        empresa.setDataAbertura(dataAbertura);
```

COPIAR CÓDIGO

Percebam que o Servlet, neste caso, ainda é de baixo nível. Existem bibliotecas de alto nível que podem nos ajudar com a leitura de parâmetros e com o trabalho de *parsing*, como preenchimento de dados. Bibliotecas como o **SpringMVC** podem realizar todo esse trabalho e tornar o código mais enxuto.

Imagine esse mesmo código trabalhando com dez parâmetros? É fácil imaginar que o Servlet não poderá resolver todas as questões que encontramos no dia-a-dia.

Podemos testar a nossa aplicação, acessando http://localhost:8080/gerenciador/formNovaEmpresa.jsp (http://localhost:8080/gerenciador/formNovaEmpresa.jsp). Preencheremos os campos "Nome" e "Data Abertura", incluindo a empresa "Facebook" e a data "11/01/2001". A empresa será cadastrada com sucesso.

Em seguida, acessaremos a lista de empresas cadastradas via http://localhost:8080/gerenciador/listaEmpresas (http://localhost:8080/gerenciador/listaEmpresas), e receberemos a seguinte mensagem:

> Lista de empresas:
>
> Alura - 24/02/2018
>
> Caelum - 24/08/2018
>
> Facebook - 11/01/2001

Tudo opera como o esperado. A formatação de `listaEmpresas` estava pronta, e fizemos a leitura de novos parâmetros em `NovaEmpresaServlet`. Concluímos a parte de fmt, o foco da aula, aprendendo como funciona essa biblioteca de formatação e, por fim, aprendemos que utilizar Servlet na web é bastante trabalhoso dependendo do tipo de projeto. Por isso, muitas vezes é importante simplificarmos nosso trabalho com o auxílio de outras bibliotecas de alto nível.