08

Finalmente, o primeiro servlet

Transcrição

Nesta aula criaremos nosso primeiro Servlet, mas antes faremos uma breve recapitulação do que fizemos até agora.

No Eclipse, ao acionarmos a execução do Tomcat, estamos inicializando uma máquina virtual que requer uma classe com um método `main`. O Tomcat é um método `main` que sobe um servidor com várias classes e executa diferentes ações, algo muito sofisticado.

Nós usamos o navegador para realizar uma requisição para o Tomcat por meio do protocolo HTTP (no qual o navegador é especialista). O servidor recebeu a requisição e devolveu a resposta, que no princípio era apenas uma página de erro 404, afinal ainda não havíamos produzido um conteúdo a ser exibido.

Posteriormente, adicionamos uma página HTML dentro do projeto **gerenciador**. Feito isso, conseguimos acessar o caminho http://localhost:8080/gerenciador/bem-vindo.html (http://localhost:8080/gerenciador/bem-vindo.html), isto é, acessar o Tomcat inserindo a porta correta, o projeto **gerenciador** e o conteúdo da página `bem-vindo.html`.

O navegador envia as informações na requisição HTTP, e o Tomcat reconhece essa requisição e devolve o conteúdo solicitado na reposta HTTP. **O protocolo HTTP funciona sempre na dinâmica de requisição e reposta**.

Ao final, conseguimos exibir a mensagem "Bem-vindo no curso de Servlets d Alura" no navegador.

Nossa próxima meta é adicionar mais um recurso ao **gerenciador** - não uma página estática, mas sim um conteúdo dinâmico. Ao receber uma requisição podemos configurar o envio de um e-mail, cadastrar alguma informação no banco de dados ou ler dados do banco e gerar um HTML dinamicamente, por exemplo.

Todas essas possibilidades não são possíveis dentro de uma página HTML, já que se trata apenas de um arquivo de texto. Portanto, precisaremos programar, ou seja, escrever uma classe, criar objetos e executar lógicas no banco dados. Essa é a tarefa do Servlet.

Servlet é um objeto especial armazenado dentro do projeto, e sua particularidade consiste na possibilidade de o chamarmos via protocolo HTTP.

Quando o Tomcat recebe a requisição do navegador com relação aos dados do projeto **gerenciador**, ao abrirmos a página não estamos mais lidando com um arquivo, mas com um Servlet. Isto é, um objeto especial executado para gerar uma resposta HTTP dinâmica.

O termo *let* de *Servlet* é um sufixo diminutivo no inglês, e uma tradução livre seria algo como "Servidorzinho". A ideia é que o Tomcat é um servidor principal, e o Servlet opera de forma semelhante e auxiliar, afinal ele pode receber requisições e gerar respostas dinâmicas por meio do protocolo HTTP.

Agora faremos a implementação do Servlet em nosso projeto. Precisamos criar um objeto, mas antes, naturalmente, devemos criar a classe. Essa nova classe será criada dentro da pasta "src", clicando com o botão direito nela e em seguida em "New > Class". A inseriremos no pacote `br.com.alura.gerenciador.servlet`, e seu nome será `OiMundoServlet`.

Essa nova classe será o Servlet. Um relacionamento importante no mundo Java é a herança, e para transformar essa classe em um Servlet precisamos estendê-la com `extends`. O protocolo que este Servlet irá estender será o `Http`.

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

public class OiMundoServlet extends HttpServlet {

}
```

COPIAR CÓDIGO

O Eclipse irá indicar um erro, afinal ainda não realizamos a importação desta classe, mas para fazê-lo basta clicar com o botão direito sobre o trecho com erro e escolher a opção "*Import HttpServlet*".

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

import javax.servlet.http.HttpServlet;

public class OiMundoServlet extends HttpServlet {

}
```

COPIAR CÓDIGO

Adicionaremos um método que atende uma requisição HTTP, que chamaremos de `service()`. Na verdade, iremos sobrescrever um método já definido na classe mãe. Por isso, pressionaremos "Ctrl + Espaço" para conferir as sugestões para o método `service()`, dentre as quais escolheremos implementar aquela que recebe `HttpServletRequest arg0, HttpServletResponse`.

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

import java.io.IOException;

import javax.servlet.http.HttpServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;

public class OiMundoServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletI
        // TODO Auto-generated method stub
        super.service(req, resp);
    }
}
```

COPIAR CÓDIGO

O método trouxe várias informações novas para o nosso código, mas entenderemos tudo com calma. Primeiramente, faremos uma simplificação e retiraremos as exceções `throws ServletException, IOException` , o método `super.service(req, resp)` e os comentários.

`@Override` será mantido, pois de fato estamos sobrescrevendo o método. O uso do `void` está correto, afinal o método não retorna nada, o que pode soar estranho, mas toda a resposta que precisamos não será por meio do retorno do método. O `service()` recebe dois parâmetros: `HttpServletRequest req` e `HttpServletResponse resp` , já que o protocolo HTTP funciona pela dinâmica de *request* e *response*, apresentadas por meio desses dois objetos no mundo Java.

Quando o método `service()` é chamado, automaticamente recebemos as referências que apontam para os objetos que representam a requisição e resposta.

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

import java.io.IOException;

import javax.servlet.http.HttpServletException;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class OiMundoServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletI

    }
}
```

COPIAR CÓDIGO

Queremos cumprir duas metas: a primeira é fornecer um apelido no endereço URL que remetesse ao Servlet do projeto **gerenciador**, algo como `http://localhost:8080/gereneciador/oi`, sendo que `oi` é esta referência. A segunda meta é desenvolver a resposta do Servlet para o navegador.

Existem duas formas de realizar a configuração de um apelido, como o `oi` do nosso exemplo. Uma delas é recorrendo ao arquivo xml, essa é uma abordagem mais antiga e cansativa, por isso a veremos posteriormente no curso. A outra maneira de realizar uma configuração no código Java é utilizar **anotações**, que sempre são iniciadas por `@`. Assim como `@Override` é uma configuração voltada para o compilador, existem outras direcionadas para a máquina virtual.

Usaremos a anotação `@WebServlet`, e pressionaremos o atalho "Ctrl + Space" para realizar a importação. É por meio dessa anotação que poderemos definir um `urlPatterns`, que é nada mais que o nome do Servlet na URL que utilizaremos no navegador. Definiremos o nome do Servlet como `/oi`.

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

import java.io.IOException;
```

```
import javax.servlet.http.HttpServletException;
import javax.servlet.http.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


//oi
@WebServlet(urlPatterns="/oi")
public class OiMundoServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletI

    }
}
```

COPIAR CÓDIGO

No momento em que executarmos o Tomcat, essa nova configuração inserida na classe estará ativa. Resta definirmos a resposta do Servlet para o navegador, o que faremos utilizando o objeto `resp`.

No mundo HTTP existem duas formas de devolver uma resposta: a primeira é definir um fluxo binário para, por exemplo, gerar um relatório, arquivo em PDF ou gráfico e devolver essa imagem. Para este caso, utilizaríamos o método `getOutPutStream()`. A segunda alternativa é utilizar o método `getWriter()` para devolver um conteúdo HTML.

O `getWriter()` devolve um objeto chamado `PrintWriter` do pacote `java.io`. O compilador exige que lancemos uma exceção neste caso, por isso a inseriremos em nosso código.

Feito isso, temos o objeto especializado para devolver caracteres, portanto podemos devolver um conteúdo textual por meio do método `out.println()`,

utilizando a estrutura padrão básica HTML. É parecido com o método `system.out.println()`, porém utilizando o objeto `PrintWriter` do `HttpServletResponse`. A nossa mensagem será "oi mundo, parabens vc escreveu o primeiro servlets".

```
//oi
@WebServlet(urlPattern="/oi")
public class OiMundoServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletI

        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("oi mundo, parabens vc escreveu o primeiro
        out.println("</body>");
        out.println("</html>");

    }
}
```

COPIAR CÓDIGO

É importante frisar que o `out.println()` não imprime no console, mas sim no fluxo de resposta para o navegador.

A classe está pronta, temos uma configuração, os métodos e a impressão da mensagem. O conteúdo ainda não é dinâmico, mas nesse ponto poderíamos estruturar um HTML complexo. Vamos testar?

Executaremos o Tomcat e verificaremos se o console aponta algum erro (o que sempre é muito importante). Normalmente, se há algum problema com o Tomcat, ele é exibido na linha de comando.

O Tomcat está ativo com base na máquina virtual, leu nosso projeto e o Servlet, cujo apelido é `/oi`. Se tudo ocorreu corretamente, poderemos digitar o endereço http://localhost:8080/gerenciador/oi (http://localhost:8080/gerenciador/oi) no navegador e visualizar a mensagem.

Caso a mensagem não seja exibida no seu navegador, verifique atentamente no console em busca de alguma mensagem de erro. É interessante também analisar o código fonte no próprio navegador, pois pode existir algum erro no HTML.

Para garantir o sucesso do código, podemos inserir ao final o `System.out.println()` para imprimirmos itens diretamente no console. Faremos esse teste com a mensagem "o servlet OiMundoServlet foi chamado".

```
//oi
@WebServlet(urlPattern="/oi")
public class OiMundoServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletI

        PrintWriter out = rep.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("oi mundo, parabens vc escreveu o primeiro
        out.println("</body>");
        out.println("</html>");

        System.out.println("o servlet OiMundoServlet foi chama
    }
}
```

COPIAR CÓDIGO

O Tomcat percebe as alterações feitas em nosso código e faz o recarregamen (*reload*) automaticamente, porém é mais eficiente reinicializar o servidor

manualmente, já que nem sempre esse recarregamento funciona.

No navegador a mensagem não sofrerá qualquer alteração, pois não realizamos nenhum trabalho extra no HTML, mas a mensagem "O servlet OiMundoServlet foi chamado" será exibida no console, o que significa que a requisição chegou, o Tomcat reconheceu a Servlet, criou o objeto da classe e chamou o método `service()`. Por fim, a resposta foi devolvida através do método `PrintWriter()` e utilizamos o `System.out.println()` para confirmar o processo.