



## Enviando parâmetros

### Transcrição

Seguiremos com o projeto e aprenderemos mais funcionalidades do Servlets. Teremos mais algumas explicações teóricas e configurações, mas iremos introduzir esse conteúdo aos poucos.

Um pequeno resumo do que fizemos até o momento: usamos o navegador para enviar uma requisição, o Servlet foi mapeado e devolveu uma resposta que criamos no código Java. Construímos a comunicação remota, isto é, a URL poderia ser aberta em outro computador para acessar o Tomcat por meio da rede. Na verdade, mesmo aqui, usando o mesmo computador, já temos uma conexão remota.

Retomaremos também a construção do nosso código: Em

`OiMundoServlet.java` , temos a configuração `@WebServlet(urlPatterns="/oi")` , a URL que evoca o Servlet. Temos o método `service()` que recebe os objetos `HttpServletRequest` e `HttpServletResponse` , isto é, referências fundamentais de requisição e resposta. A partir de `resp` , usamos o método `getWriter()` para de fato devolver algum conteúdo ao navegador.

Por fim, utilizamos o `System.out.println()` para mostrar que o Servlet realmente está sendo chamado.

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
```

```
import javax.servlet.annotation.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, HttpServletResponse resp) {

        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>");

        System.out.println("o servlet OiMundoServlet foi chamado");
    }
}
```

[COPIAR CÓDIGO](#)

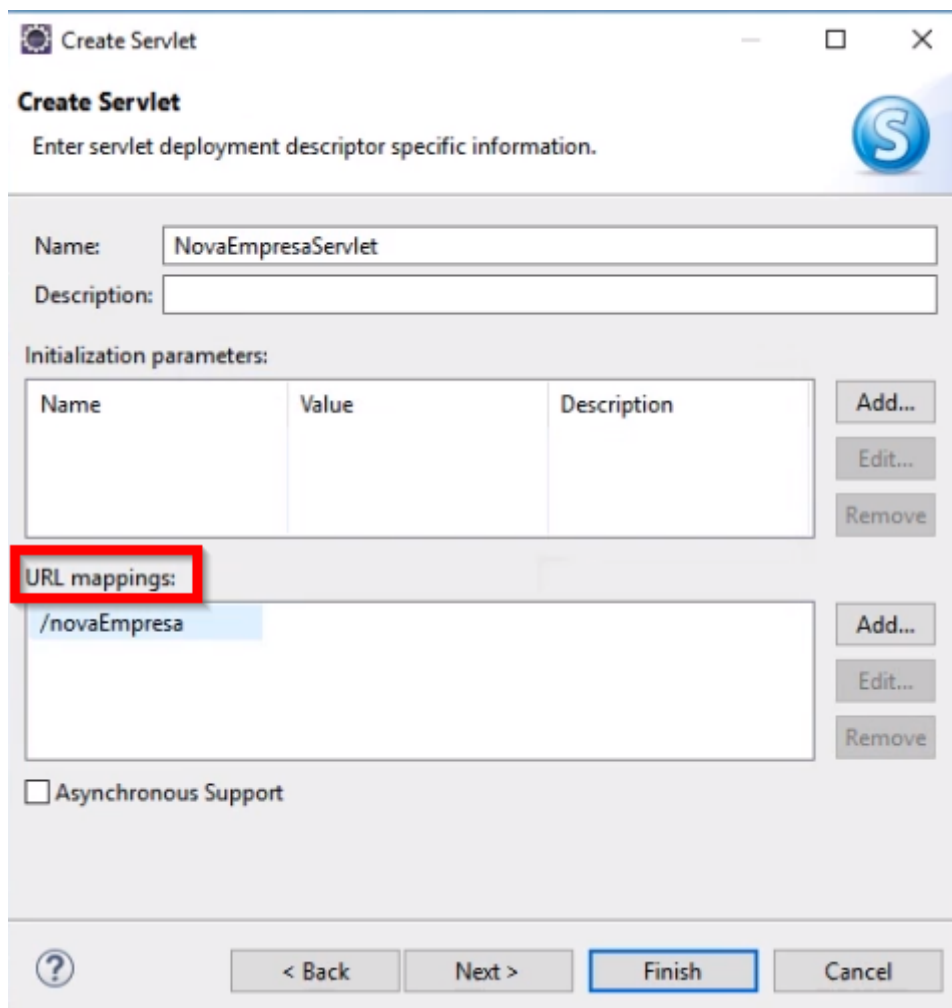
A próxima meta é criar um sistema de cadastro para empresas por meio do Servlet. Passaremos a enviar dados para este pequeno servidor e precisaremos devolver uma resposta de confirmação de cadastro.

Para isso, faremos um novo Servlet. Na pasta "gerenciador > Java Resources > src > br.com.alura.gerenciador", clicaremos com o botão direito e selecionaremos a opção "New > Servlet". Com esse diálogo, podemos criar um Servlet por meio do próprio Eclipse.

Caso essa opção não esteja visível para você, selecione "New > Other" e digite no campo "Wizard" a palavra "Servlet".

Na caixa de diálogo aberta, chamaremos a nova classe de `NovaEmpresaServlet`. A superclasse ("*Superclass*") será estendida de `javax.servlet.http.HttpServlet`. Já poderíamos confirmar a criação da classe, mas pressionaremos o botão "Next" para conhecer mais algumas configurações.

Na página seguinte, temos a opção "*URL mappings*", na qual é possível editar o mapeamento da URL. Clicaremos sobre a opção "Edit". e modificaremos o nome de `/NovaEmpresaServlet` para `/novaEmpresa`.



The screenshot shows the 'Create Servlet' dialog box in Eclipse IDE. The dialog is titled 'Create Servlet' and contains the following fields and sections:

- Name:** `NovaEmpresaServlet`
- Description:** (empty)
- Initialization parameters:** A table with columns 'Name', 'Value', and 'Description'. To the right of the table are buttons 'Add...', 'Edit...', and 'Remove'.
- URL mappings:** A section highlighted with a red box, containing a list with the entry `/novaEmpresa`. To the right of the list are buttons 'Add...', 'Edit...', and 'Remove'.
- Asynchronous Support:** A checkbox that is currently unchecked.
- Navigation buttons:** At the bottom, there are buttons for '< Back', 'Next >', 'Finish' (highlighted with a blue box), and 'Cancel'.

Clicando em "Next", acessaremos outra janela de diálogo. Nessa nova área, podemos escolher métodos que podem ser previamente implementados pelo Eclipse. Iremos desmarcar a opção de construtor padrão "*Constructors from superclass*"; e desmarcaremos também os métodos "doPost" e "doGet".

Which method stubs would you like to create?

- |  |                                    |   |
|--|------------------------------------|---|
| <input type="checkbox"/> Constructors from superclass          |                                    |   |
| <input checked="" type="checkbox"/> Inherited abstract methods |                                    |   |
| <input type="checkbox"/> init                                  | <input type="checkbox"/> destroy   | <input type="checkbox"/> getServletConfig |
| <input type="checkbox"/> getServletInfo                        | <input type="checkbox"/> service   | <input checked="" type="checkbox"/> doGet |
| <input checked="" type="checkbox"/> doPost                     | <input type="checkbox"/> doPut     | <input type="checkbox"/> doDelete         |
| <input type="checkbox"/> doHead                                | <input type="checkbox"/> doOptions | <input type="checkbox"/> doTrace          |

Clicando em "*Finish*", nossa nova classe estará criada:

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

```
import java.io.IOException;
```

```
/**
```

```
 *Servlet implementation class NovaEmpresaServlet
```

```
 */
```

```
@WebServlet("/novaEmpresa")
```

```
public class NovaEmpresaServlet extends HttpServlet {
```

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

```
    /**
```

```
     * @see HttpServlet#service(HttpServletRequest request, Http
```

```
     */
```

```
    protected void service(HttpServletRequest request, HttpServ
```

```
        //TODO Auto-generated method stub
```

```
    }
```

```
}
```

COPIAR CÓDIGO

Nesse código, removeremos os comentários. O `serialVersionUID` serve para que desapareça um *warning* relacionado ao Java IO, mas ele é totalmente opcional.

O Servlet está pronto, e podemos testá-lo acionando o `System.out.println()` com a mensagem `Cadastrando nova empresa`, que será impressa somente no console.

Em seguida, estruturaremos também a mensagem HTML por meio do `PrintWriter()`, especializado no envio de caracteres. Usaremos a referência `out`, e o coletaremos por meio de `response.getWriter()`. Lembre-se que é necessário importar o método `PrintWriter()` do pacote `java.io`.

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

import java.io.IOException;

@WebServlet("/novaEmpresa")
public class NovaEmpresaServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("Cadastrando nova empresa");
        PrintWriter out = response.getWriter();
        out.println("<html><body>Empresa cadastrada com sucesso");
    }
}
```

[COPIAR CÓDIGO](#)

Executaremos o Tomcat e verificaremos o console em busca de alguma ocorrência de erro. Não encontramos nenhum, então seguiremos para o navegador, onde digitaremos a URL

<http://localhost:8080/gerenciador/novaEmpresa>  
(<http://localhost:8080/gerenciador/novaEmpresa>).

A mensagem Empresa cadastrada com sucesso! será exibida no navegador como esperávamos, e no console teremos a mensagem Cadastrando nova empresa .

O que fizemos até agora foi uma simples revisão do conteúdo aprendido. A partir desse ponto nós podemos de fato começar a criar o sistema de cadastro de empresas, isto é, enviar dados a partir do navegador para o Servlet - nesse caso, o nome da empresa.

Existem duas formas de fazer isso: a primeira consiste em inserir parâmetros da requisição na URL. Depois do nome da URL, inserimos o caractere `?` que marcará o início do parâmetro, seguido de seu valor. Vejamos um exemplo hipotético, cujo nome do parâmetro será `nome` e o valor `Alura` . Para adicionarmos mais um parâmetro adicionaremos o caractere `&` , seguindo do `cnj` (nome do segundo parâmetro) e o valor `123` .

`localhost:8080/gerenciador/novaEmpresa?nome=Alura&cnj=123`

Ou seja, respeitando essa sintaxe, é possível enviar parâmetros para o servidor. Se utilizarmos essa URL no navegador, os parâmetros serão enviados com sucesso e o servidor foi chamado corretamente, já que a mensagem "Cadastrando nova empresa" é impressa no console. Porém, nosso Servlet ainda não está lendo esses parâmetros.

Gostaríamos de ler o novo parâmetro enviado, o que pode ser feito a partir da requisição ( `request` ) com o método `getParameter()` , que recebe uma string - isso porque precisamos passar o nome do parâmetro, que é `nome` . Será devolvida uma string `nomeEmpresa` .

Por fim, concatenaremos na mensagem HTML a variável `nomeEmpresa` .

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

import java.io.IOException;

@WebServlet("/novaEmpresa")
public class NovaEmpresaServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("Cadastrando nova empresa");
        String nomeEmpresa = request.getParameter("nome");
        PrintWriter out = response.getWriter();
        out.println("<html><body>Empresa " + nomeEmpresa + " cadastrada com sucesso!");
    }
}
```

[COPIAR CÓDIGO](#)

Salvaremos as modificações e executaremos o Tomcat.

Feito isso, acessaremos a URL <http://localhost:8080/gerenciador/novaEmpresa?nome=Alura> (<http://localhost:8080/gerenciador/novaEmpresa?nome=Alura>), e teremos a mensagem "Empresa Alura cadastrada com sucesso!". Caso modifiquemos a URL para <http://localhost:8080/gerenciador/novaEmpresa?nome=Caleum> (<http://localhost:8080/gerenciador/novaEmpresa?nome=Caleum>), a mensagem será atualizada para "Empresa Caelum cadastrada com sucesso!".

Chamamos o Servlet, enviamos um parâmetro na requisição, lemos esse parâmetro e depois o concatenamos com a resposta exibida no navegador. Podemos fazer esse processo ser mais elaborado, mas a primeira parte já conseguimos completar, enviamos os dados por meio da requisição HTTP.

