



## ROTEIRO DE LABORATÓRIO - SERVLETS

Nesta aula você aprenderá a criar classes acessíveis ao navegador. Essas classes poderão interagir com o usuário através de formulários. Aprenderemos a receber e converter parâmetros enviado por uma página e, por fim, distinguir entre os métodos HTTP.

## SERVLETS

Os **Servlets** são a primeira forma que veremos de criar páginas dinâmicas com Java. Usaremos a própria linguagem Java para isso, criando uma classe que terá capacidade de gerar conteúdo HTML.

Uma primeira ideia do servlet seria que cada uma deles é responsável por uma página, sendo que este lê dados da requisição do cliente e responde com outros dados. Como no Java tentamos sempre que possível trabalhar orientado a objetos, nada mais natural que uma servlet seja representada como um objeto a partir de uma classe Java.

Cada servlet é, portanto, um objeto Java que recebe tais requisições (**request**) e produz algo (**response**), como uma página HTML dinamicamente gerada.

O comportamento das servlets que vamos ver nesta aula é definido na classe `HttpServlet` do pacote `javax.servlet`. A interface `Servlet` é a que define exatamente como uma servlet funciona, mas não é o que vamos utilizar, uma vez que ela possibilita o uso de qualquer protocolo baseado em requisições e respostas, e não especificamente o HTTP.

Para escrevermos uma servlet, criamos uma classe Java que estenda `HttpServlet` e sobrescreva um método chamado `service`. Esse método será o responsável por atender requisições e gerar as respostas adequadas.

A Figura 1 apresenta a assinatura do método `service`.

```
protected void service(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {  
    // TODO Auto-generated method stub  
}
```

Figure 1. Assinatura do método `service()`.

Repare que o método recebe dois objetos que representam, respectivamente, a requisição feita pelo usuário e a resposta que será exibida no final. É possível utilizar esses objetos para obter informações sobre a requisição e para construir a resposta final para o usuário.

Nosso primeiro exemplo de implementação do método `service` não executa nada de lógica e apenas mostra uma mensagem estática de bem vindo para o usuário. Para isso, precisamos construir a resposta que a servlet enviará para o cliente.

É possível obter um objeto que represente a saída a ser enviada ao usuário através do método `getWriter` da variável `response`. E, a partir disso, utilizar um `PrintWriter` para imprimir algo na resposta do cliente. Veja o exemplo da Figura 2, abaixo.

```
protected void service(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // TODO Auto-generated method stub
    PrintWriter out = response.getWriter();

    out.println("<html>");
    out.println("<body>");
    out.println("Olá Mundo..");
    out.println("</body>");
    out.println("</html>");
}
```

Figure 2. Primeiro exemplo de servlet.

O único objetivo da servlet acima é exibir uma mensagem HTML simples para os usuários que a requisitarem. Mas note como seria muito fácil escrever outros códigos Java mais poderosos para gerar as Strings do HTML baseadas em informações dinâmicas vindas, por exemplo, de um banco de dados.

## ENVIANDO PARÂMETROS NA REQUISIÇÃO:

Ao desenvolver uma aplicação Web, sempre precisamos realizar operações no lado do servidor, operações com dados informados pelo usuário, seja através de formulários ou seja através da URL.

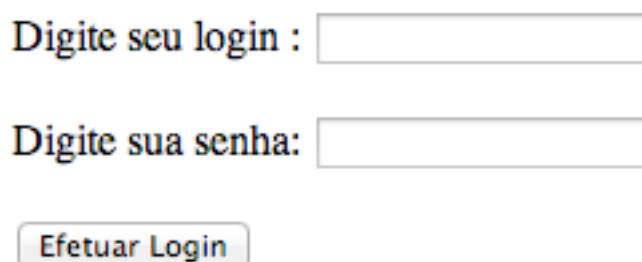
Por exemplo, para efetuarmos login em um sistema de envio de mensagens eletrônicas precisamos coletar do usuário suas credenciais de acesso, ou seja, login e senha. Temos uma página com um formulário que o usuário possa preencher e ao clicar em um botão esses dados devem, de alguma forma, ser passados para uma servlet. Já sabemos que a servlet responde por uma determinada URL, portanto, só precisamos indicar que ao clicar no botão devemos enviar uma requisição para essa servlet. Cria a página *formlogin.html* com o código abaixo.

```
<html>
<head>
<title>Login</title>
</head>
<body>
<form id="formulario" name="formulario1" method="post" action="Login">
  <p>Digite seu login :
    <input type="text" name="usuario" id="usuario" />
  </p>
  <p>Digite sua senha:
    <input type="password" name="senha" id="senha" />
  <p>
    <input type="submit" name="button" id="button" value="Efetuar Login"
  />
</p>
```

```
</form>
<body>
</html>
```

O código apresentado pela acima possui um formulário, determinado pela tag `<form>`. O atributo `action` indica qual endereço deve ser chamado ao submeter o formulário, ao clicar no botão *Efetuar Login*. Nesse caso, estamos apontando o `action` para um endereço (*Login*) que será uma Servlet que já vamos criar.

Ao acessar a *formlogin.html*, o resultado deverá ser similar à figura abaixo:



Digite seu login :

Digite sua senha:

## PEGANDO OS PARÂMETROS DE REQUISIÇÃO

Para recebermos os valores que foram preenchidos na tela e submetidos, criaremos uma Servlet, cuja função será receber de alguma maneira esses dados e convertê-los, se necessário.

Dentro do método `doPost` da nossa Servlet para validação de login, vamos buscar os dados que foram enviados na **requisição**. Para buscarmos esses dados, precisamos utilizar o parâmetro `request` do método `service` chamando o método `getParameter("nomeDoParametro")`, onde o nome do parâmetro é o mesmo nome do input que você quer buscar o valor. Isso vai retornar uma `String` com o valor do parâmetro. Caso não exista o parâmetro, será retornado `null`:

```
String valorDoParametro = request.getParameter("nomeDoParametro");
```

O código abaixo representa as instruções do método `doPost` do servlet `Login`.

```
String nomeUsuario = request.getParameter("usuario");
String senhaUsuario = request.getParameter("senha");

PrintWriter out = response.getWriter();

if (nomeUsuario.equals("admin") &&
senhaUsuario.equals("123"))
{
    request.getSession().setAttribute("usuario", nomeUsuario);
    response.sendRedirect("email.jsp");
}
```

```
        } else
        {
            RequestDispatcher dispatcher =
request.getRequestDispatcher("formlogin.html");
            dispatcher.forward(request, response);
        }
    }
}
```

O código acima verifica se as credenciais de login e senha são admin e 123, respectivamente. Caso essa validação ocorra, o usuário deve ser redirecionado para a página email.jsp, caso contrário deverá retornar à página *formlogin.html*.

Crie uma página email.jsp, e execute o formulário formlogin.html e veja se tudo está funcionando corretamente.

## LISTA DE EXERCÍCIOS

Descreva a funcionalidade das instruções:

1. `request.getSession().setAttribute("usuario", nomeUsuario);`
2. `response.sendRedirect("email.jsp");`
3. `RequestDispatcher dispatcher =  
request.getRequestDispatcher("formlogin.html");  
dispatcher.forward(request, response);`