Programação Servidor em Sistemas Web

Recursos Adicionais dos Servlets

Prof. Dr. Daniel Caetano

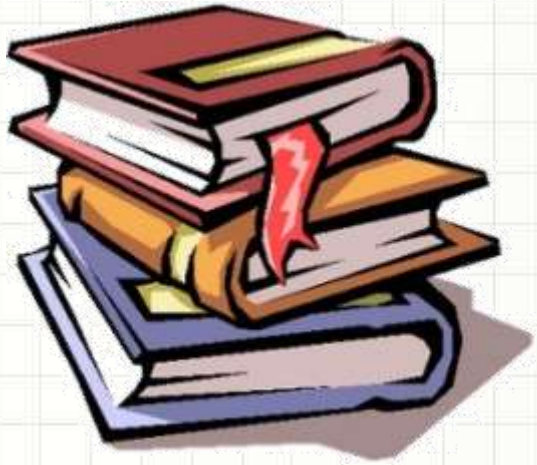
2012 - 1

Objetivos

- Apresentar os dois principais tipos de requisição
- Apresentar os dois tipos mais comuns de redirecionamento
- Armazenamento de Dados na **request**

- **Trabalho disponível online!**

Material de Estudo

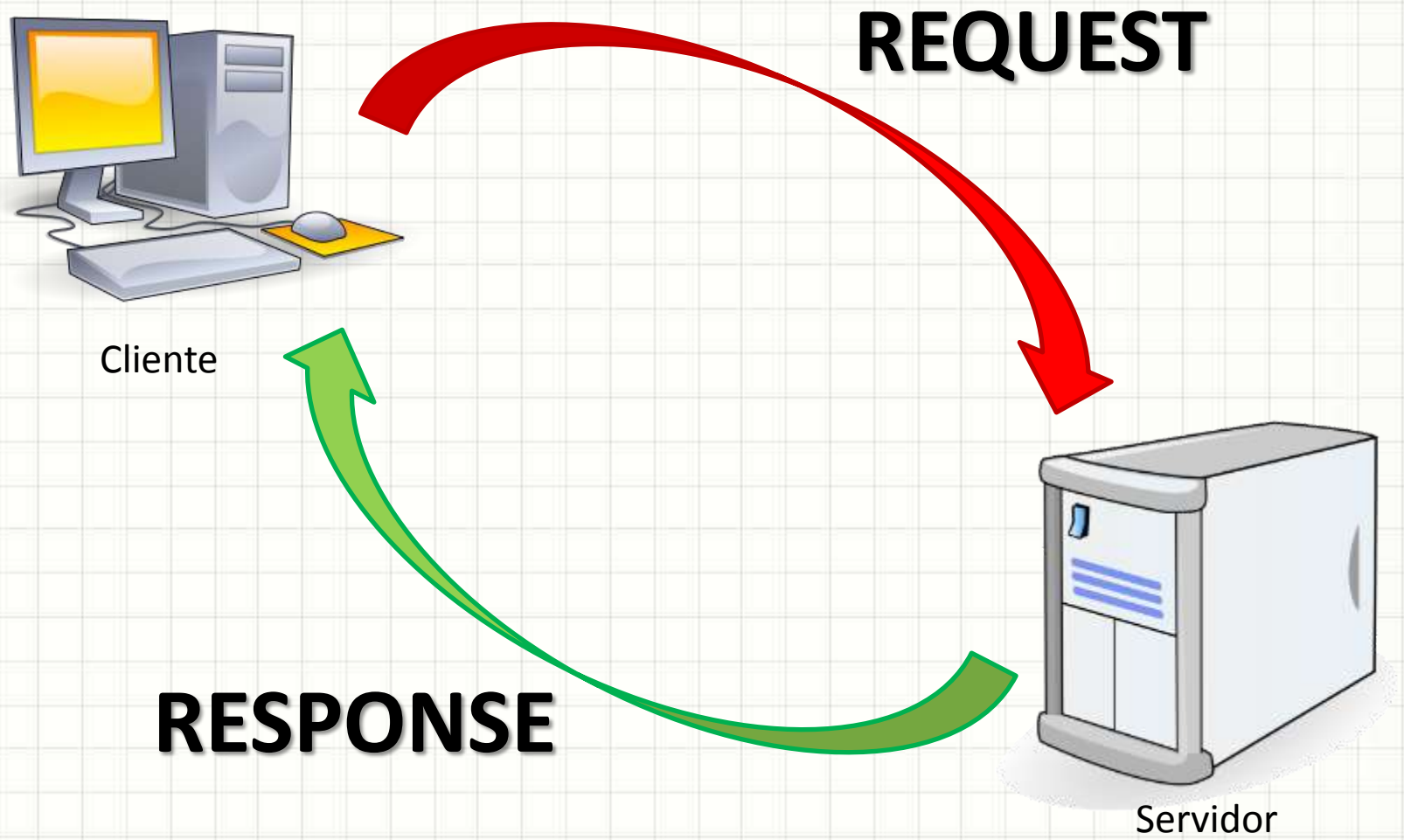


Material	Acesso ao Material
Notas de Aula	http://www.caetano.eng.br/aulas/psw/ (Aula 5)
Apresentação	http://www.caetano.eng.br/aulas/psw/ (Aula 5)
Material Didático	Big Java, páginas 989 a 997
Java: Como Programar	(6ª Edição) Páginas 928 a 948

REQUISIÇÕES POST E GET

Tipos de Requisição

- Já vimos várias vezes esse esquema:



Tipos de Requisição

- De maneira rápida, vimos também que as **request**s podem ser de dois tipos:
  - POST → Função é enviar dados para o servidor
  - GET → Função é requisitar dados do servidor

- POST: usualmente por forms
- GET: usualmente por links/barra de url

Tipos de Requisição

- Na aula passada:
  - Formulário **index.jsp**
    - → Cria a request
    - → Envia para a servlet **Imc** por POST
- Podemos fazer o mesmo com GET?
  - Digite na barra de endereços:

  http://localhost:8080/WProjeto1/Imc?peso=75&altura=1.7

- O que acontece?

Requisição com GET

- Podemos passar alguns **parâmetros** pela requisição do tipo **GET** através do seguinte esquema:

  http://servidor/servlet?param1=valor1

- O “truque” é a interrogação: **?**
- Esse caractere indica que:
  - o endereço **já acabou**
  - tudo que vem em seguida é parâmetro

Requisição com GET

http://servidor/servlet?param1=valor1

- Depois da ?
  - Nome do parâmetro (no exemplo, param1)
  - Sinal de igualdade
  - Valor do parâmetro (no exemplo, valor1)

- E se quiser passar mais de um parâmetro?

Requisição com GET

http://servidor/servlet?param1=valor1&param2=valor2

- Para indicar valores de mais parâmetros, basta separá-los com o uso de um **&**
- Podemos passar quantos parâmetros quisermos?
- **NÃO** com o GET: até cerca de 1KB

Requisição com GET

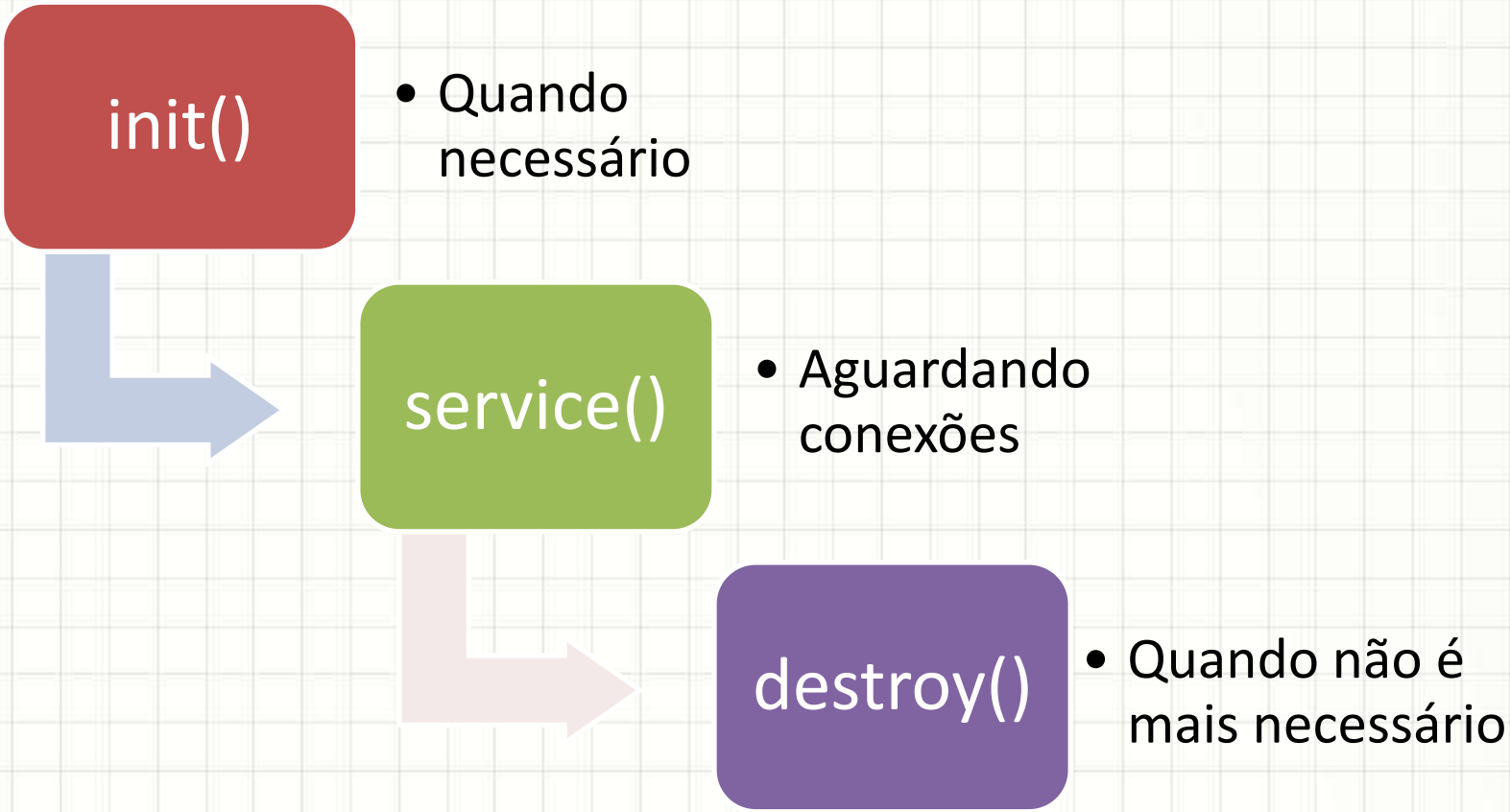
- Do ponto de vista do **servlet**, o que muda?

- Usando NetBeans, **NADA**, os dados chegam, com o **GET**, da mesma forma que com o **POST**

- Quer dizer que não temos como diferenciar um do outro no **servlet**?

Recebendo GET e POST no Servlet

- Na verdade, é possível
- Lembram-se deste esquema?



Recebendo GET e POST no Servlet

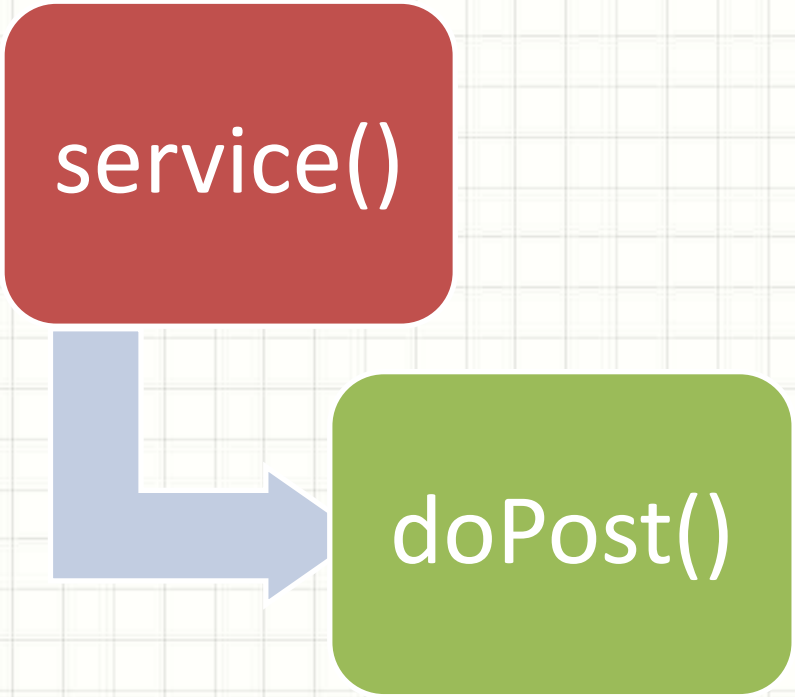
- Vamos pensar só no **service()**

- Aguardando conexões

- O **service** aguarda requisições
- Se uma **POST** chega...

Recebendo GET e POST no Servlet

- Vamos pensar só no **service()**



- O **service** aguarda requisições
- Se uma **POST** chega...

Recebendo GET e POST no Servlet

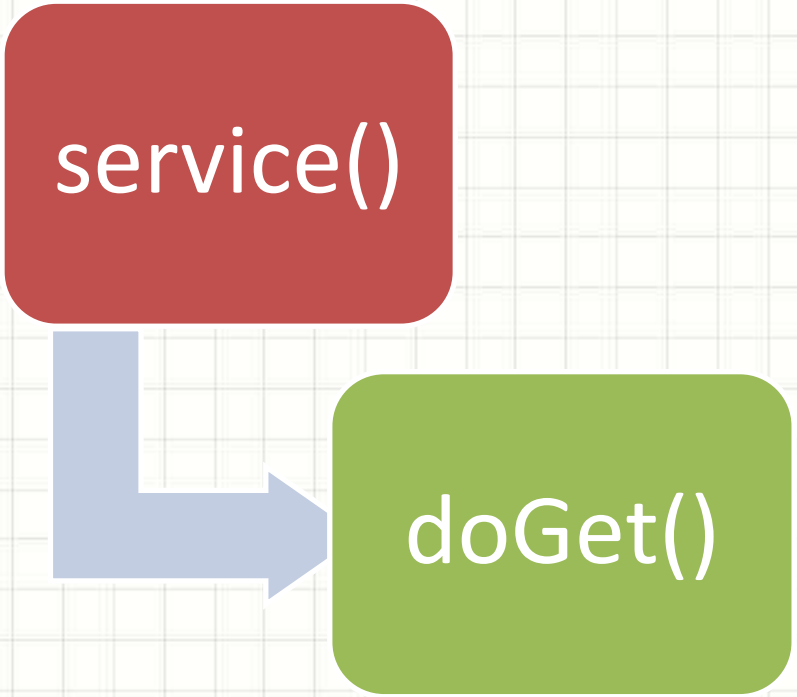
- Vamos pensar só no **service()**

- Aguardando conexões

- O **service** aguarda requisições
- Mas... E se uma **GET** chega?

Recebendo GET e POST no Servlet

- Vamos pensar só no **service()**



- O **service** aguarda requisições
- Mas... E se uma **GET** chega?

Recebendo GET e POST no Servlet

- Vamos pensar só no **servic)**

Entendendo o Servlet

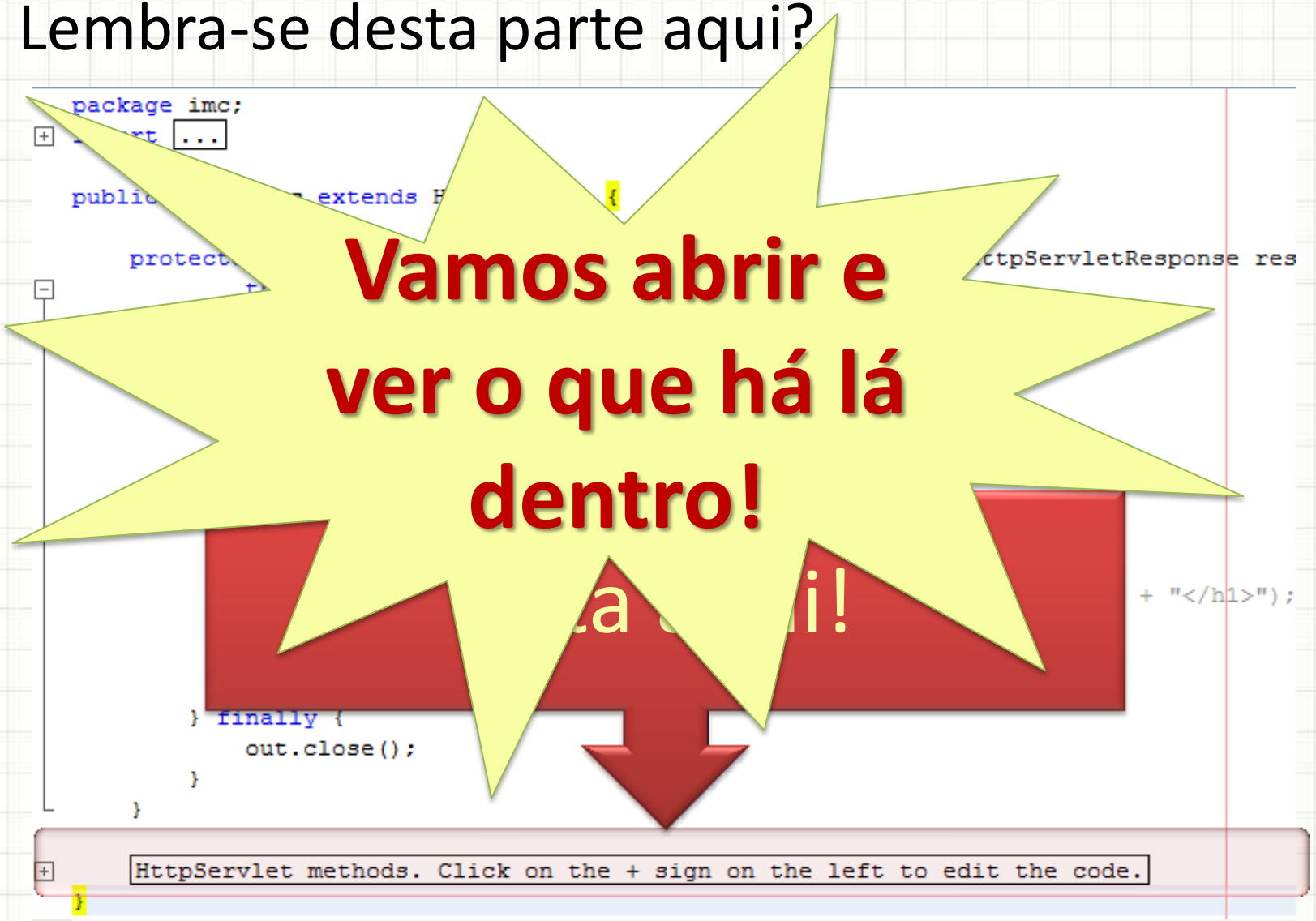
- Lembra-se desta parte aqui?

```
package imc;
import ...

public class Imc extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse res
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            /* TODO output your page here
            out.println("<html>");
            out.println("<head>");

                                                                    + "</h1>");

        } finally {
            out.close();
        }
    }

    HttpServlet methods. Click on the + sign on the left to edit the code.
}
```

Esta aqui!

Entendendo o Servlet

- Lembra-se desta parte aqui?



Entendendo o Servlet

- Observe os métodos **doGet** e **doPost**

```
// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the
/**...*/
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}

/**...*/
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}

/**...*/
@Override
public String getServletInfo() {
    return "Short description";
}// </editor-fold>
```

Entendendo o Servlet

- Observe os métodos **doGet** e **doPost**

```
// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the
/**...*/
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}

/**...*/
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}

/**...
@Overr
public
    re
}// </
```

Eles simplesmente redirecionam a chamada para o método processRequest()!

Por isso podemos usar o processRequest para acessar GET e POST

```
protected void doGet(HttpServletReque      equest, HttpServletResponse response)
        throws ServletException, IOEx      on {
    processRequest(request, response);
}

/**...*/
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}

/**...*/
@Override
public String getServletInfo() {
    return "Short description";
}// </editor-fold>
```

Rejeitando GET

- E se não quisermos que nosso **servlet** responda com requisições GET?
- Podemos ir até o método **doGet()** e simplesmente remover a chamada ao **processRequest()**
- Um outro jeito é encaminhando o usuário para uma página de erro específica...
- Ou simplesmente para a tela correta de preenchimento!

SOLICITANDO REDIRECIONAMENTO

Solicitando Redirecionamento

- O jeito mais fácil de fazer isso é modificar a **response**
- Na resposta, vamos dizer ao navegador que ele deve ir para outra página
- Isso é feito com o seguinte comando:

```
Response.sendRedirect("http://www.endereco.com/");
```

SendRedirect para Rejeitar GET

- Observe o novo método **doGet**

```
/**...*/
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.sendRedirect("http://www.uol.com.br/");
}

/**...*/
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}
```

- Se for tentado um acesso por GET, o navegador redireciona para a página da UOL

Encaminhamento de Requisição

Encaminhando uma Requisição

- Pode ser que, por alguma razão, seu **servlet** detecte que aquela requisição não pode ser processada por ele

- Neste caso, se ele souber qual **servlet** é responsável pela execução, ele pode **encaminhar a requisição**

Encaminhando uma Requisição

- Para isso, é preciso arrumar um "entregador"... Quem sabe dele é a própria **request**

Nome do Servlet Destino!

request.getRequestDispatcher("/Servlet");

- Isso **não faz redirecionamento**, apenas devolve um objeto **RequestDispatcher** (ou, em bom português, um **entregador de requisições**)

Encaminhando uma Requisição

- Assim, devemos guardar esse objeto em uma variável

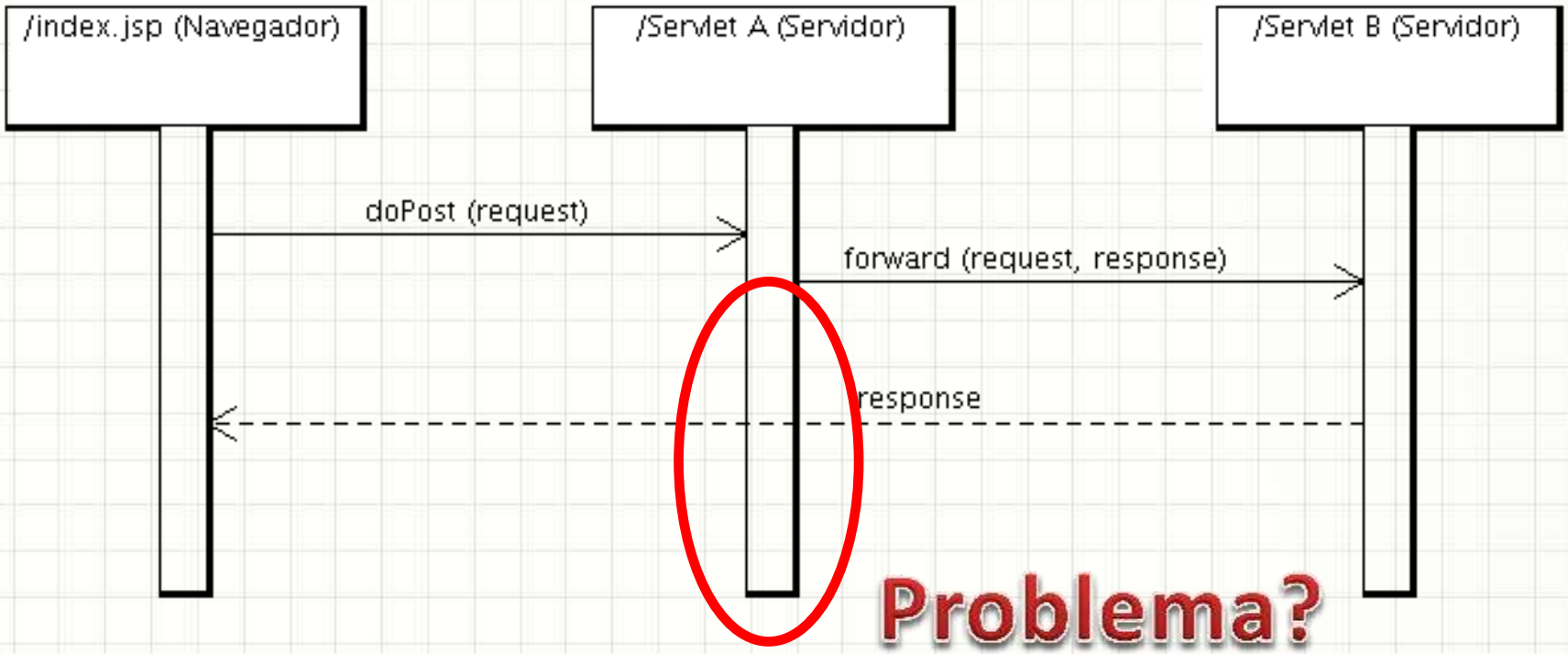
```
RequestDispatcher rd  =
request.getRequestDispatcher("/Servlet");
```

- E, para redirecionar, damos o comando **forward()** para o entregador:

```
rd.forward(request, response);
```

Encaminhando uma Requisição

- Diagrama de Sequência (forward)



- Execução do Servlet A continua!

Encaminhando uma Requisição

- Como resolver?
- Usando um **return** após o forward!
- Assim, a sequência de despacho fica...

```
RequestDispatcher rd  =
request.getRequestDispatcher("/Servlet");
rd.forward(request, response);
return;
```

Encaminhando uma Requisição

- Encaminhar requisições pode ser útil...
- Mas se pudéssemos acrescentar informações na requisição...

- **Seria bem mais útil, não?**

Modificando uma Requisição

Guardando Coisas na Requisição

- Vimos como ler **parâmetros** de uma requisição
  - **request.getParameter(“nome”);**

- Parâmetros são dados armazenados pelo **navegador** na requisição
- Não podemos inserir parâmetros através do **servlet**

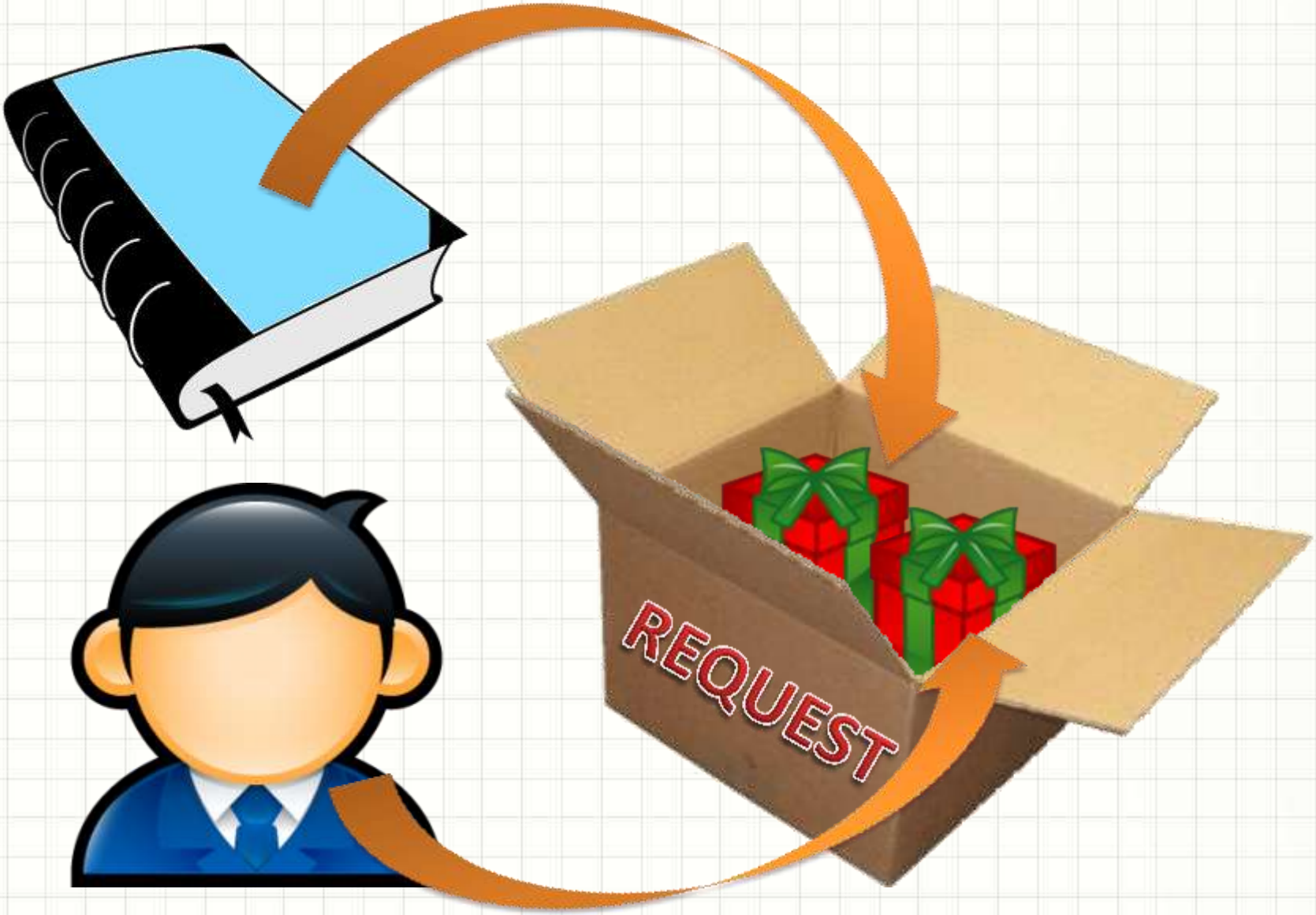
Guardando Coisas na Requisição

- Quando nosso programa acrescenta dados na requisição, dá-se o nome de **atributo**
  - **request.setAttribute(“nome”,objeto)**

- Para ler esses valores, usamos...
  - **request.getAttribute(“nome”);**

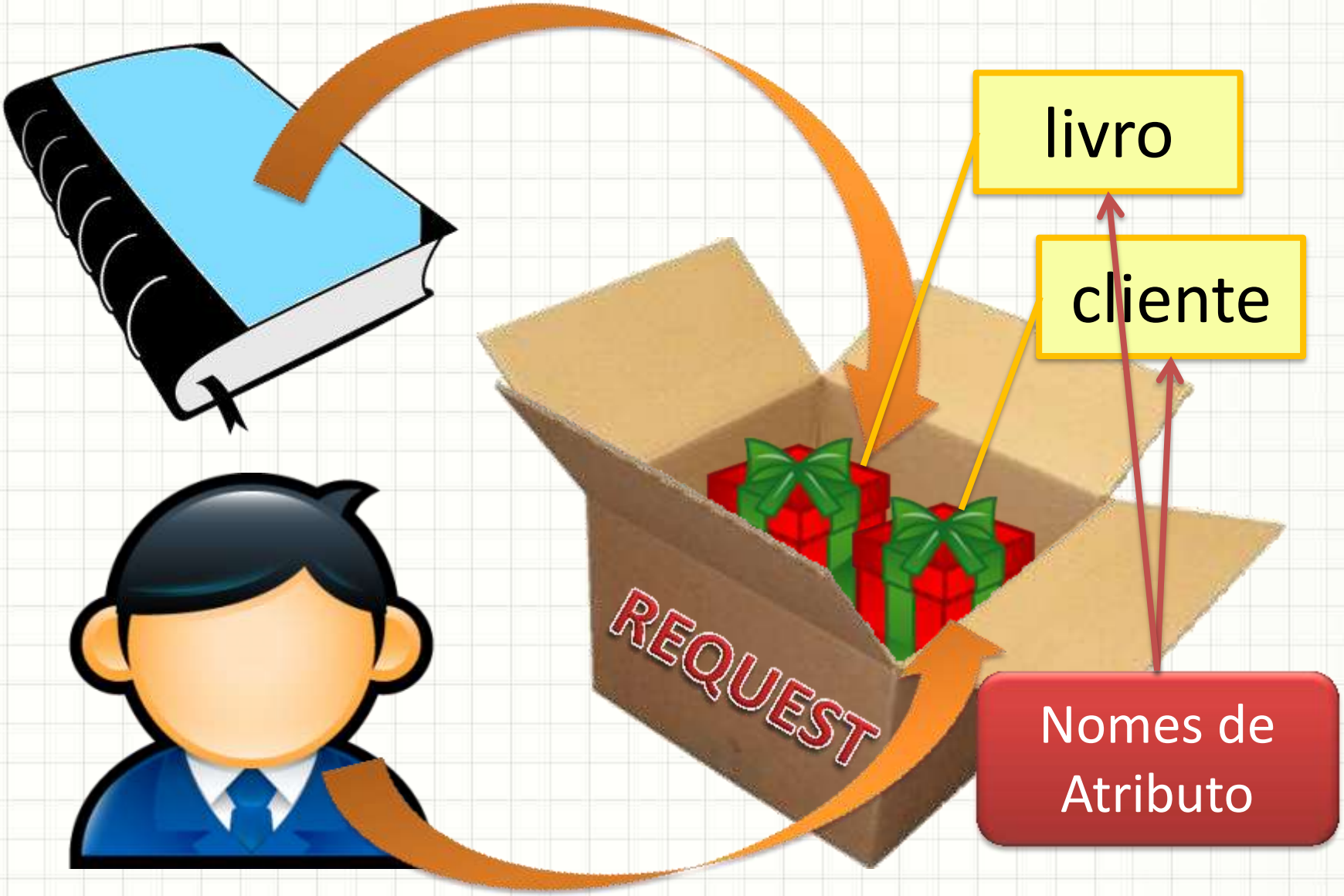
- **Pegadinha**: só posso guardar OBJETOS (tipos não nativos) na requisição

Guardando Coisas na Requisição

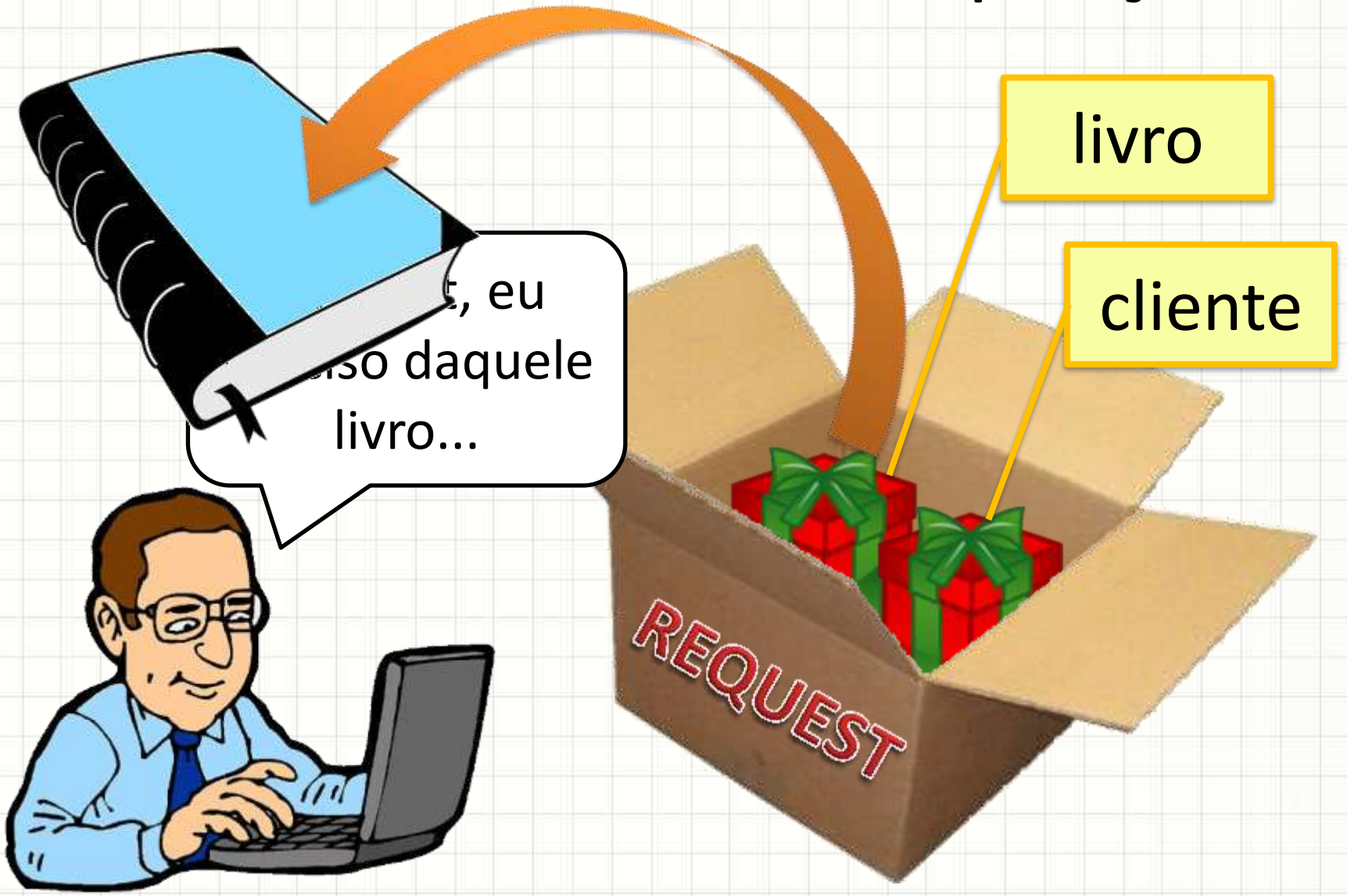


Guardando Coisas na Requisição

Guardando Coisas na Requisição



Guardando Coisas na Requisição



Por que isso é útil?

- Por que é útil guardar coisas na requisição?
- Que tal o nosso aplicativo **Imc** ser composto por dois **servlet**s?
  - Imc → Faz os cálculos
  - ImcView → Apresenta os resultados

- Por hoje, acreditem que isso facilita o reuso de código, por separar processamento de apresentação
- Na aula que vem veremos mais razões para isso ser útil...

TUTORIAL

Tutorial

- Siga o professor:
  - Construção do Servlet ImcView

Tutorial

- Siga o professor:
  - Modificação do Servlet Imc
    - Para usar IMC View
  - RequestDispatcher: “corrigir importações”

Tutorial

- Siga o professor:
  - Criação do Servlet ImcError
  - Modificação do Servlet Imc para usar ImcError

Tutorial

- Siga o professor:
  - Opcional:
    - Construir aplicativo WProjeto2 – CalcMedia

Incluindo um Servlet em Outro

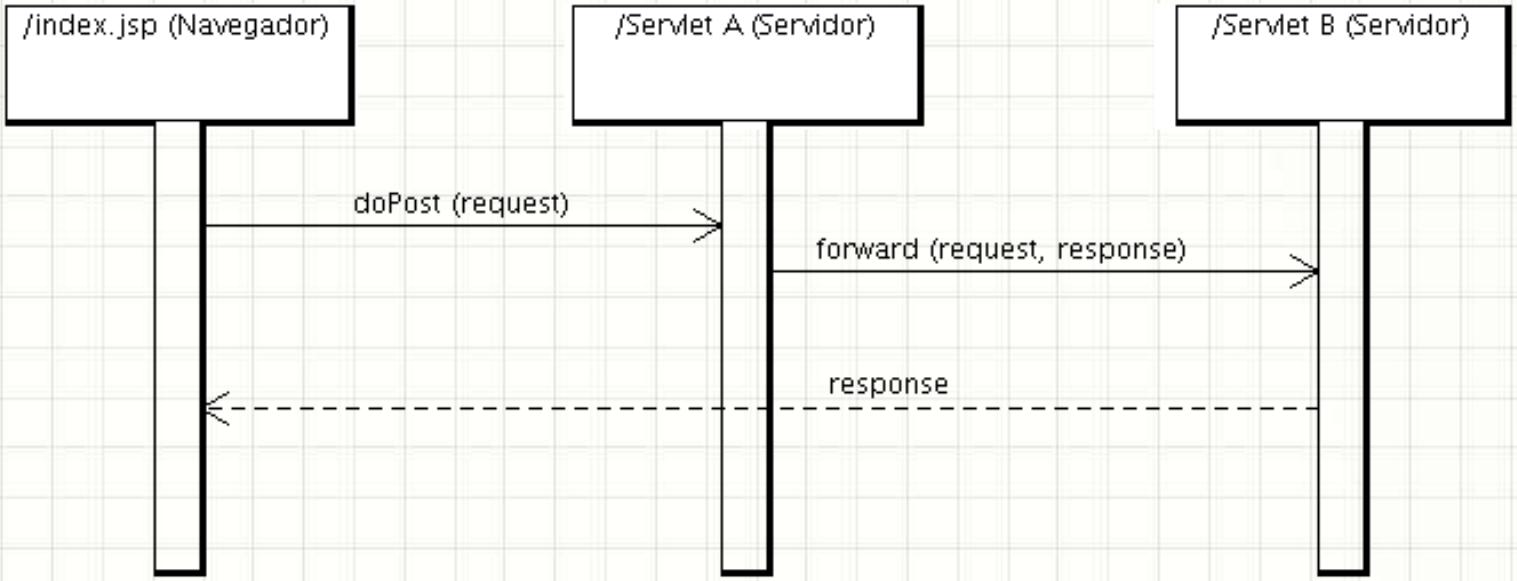
Transferência Temporária de Exec.

- Transferência Permanente: forward

RequestDispatcher rd = request.getRequestDispatcher(“/Servlet”);

rd.forward(request, response);

return;

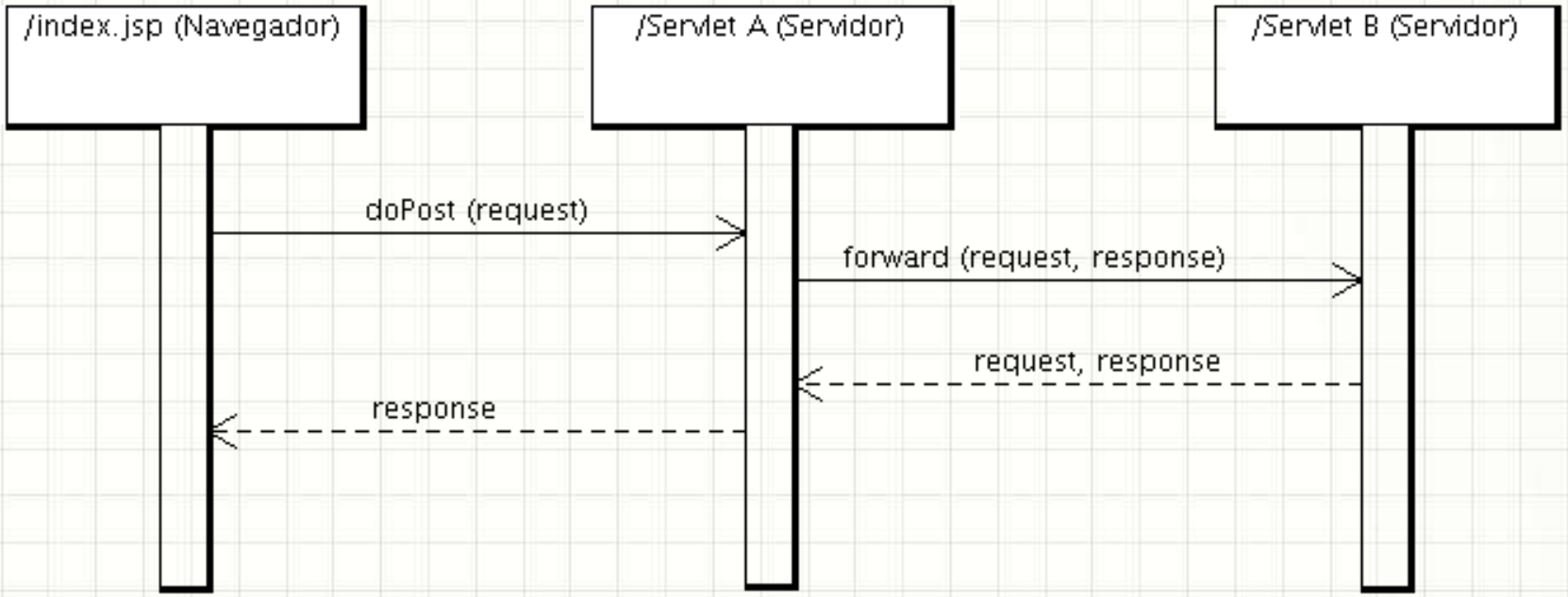


Transferência Temporária de Exec.

- Transferência Temporária: include

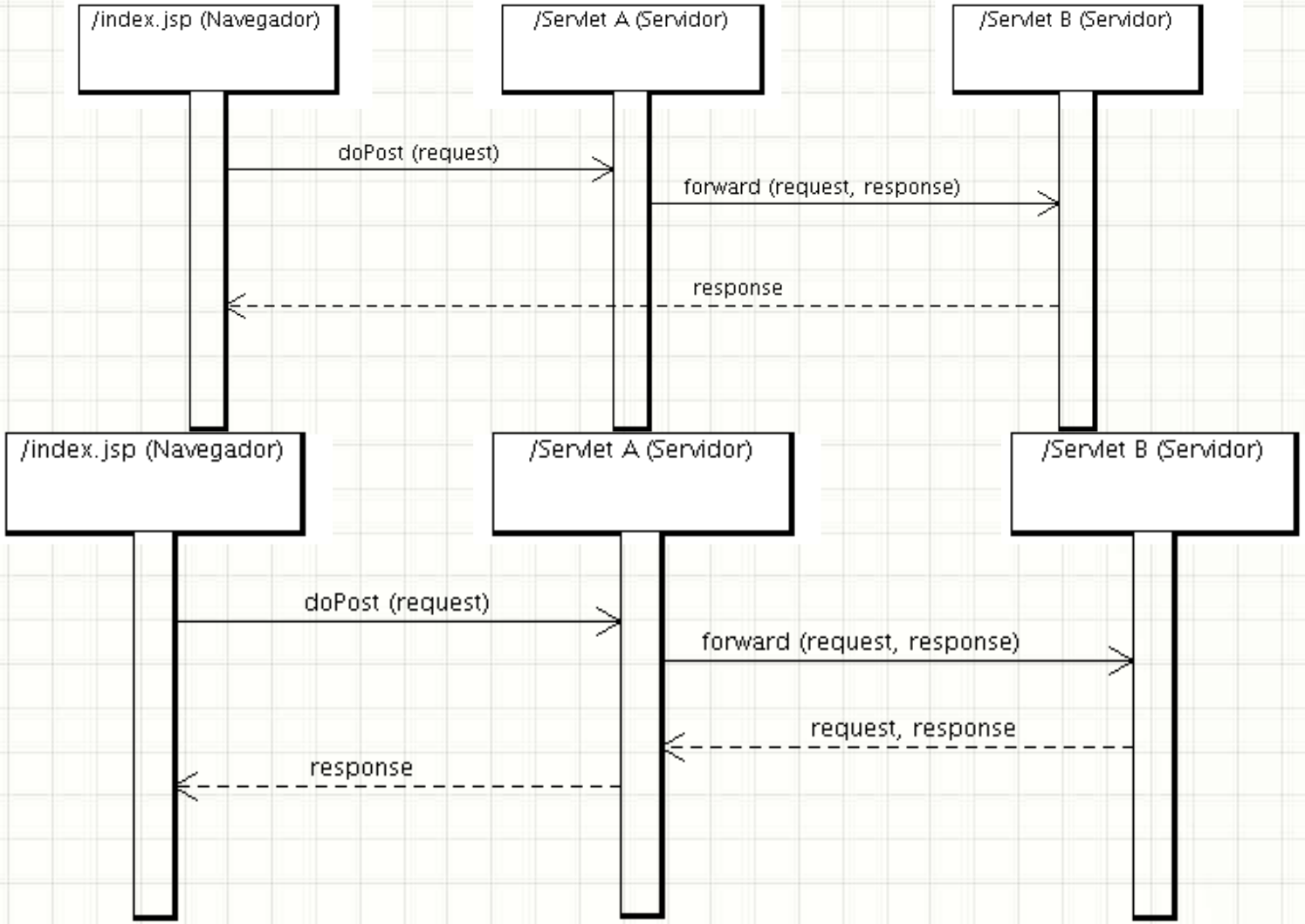
RequestDispatcher rd = request.getRequestDispatcher(“/Servlet”);

rd.include(request, response);



Transferência Temporária de Exec.

- Qual a diferença?



Atividade Estruturada

Orientação Atividades Estruturadas

- Esta disciplina possui Atividades Estruturadas
- Elas serão disponibilizadas no momento oportuno
- A primeira consiste em uma pesquisa (leitura e redação)
- A segunda consiste em compreender e modificar um sistema funcional
- Aguardem maiores informações!

Conclusões

Resumo

- Há dois tipos de requisições: POST e GET
- Os servlets podem diferenciar as requisições por tipo
- Um servlet pode redirecionar o navegador
- Um servlet pode encaminhar uma requisição para outro servlet
- Um servlet pode acrescentar dados em uma requisição

- **TAREFA**
  - **Trabalho 2 Online!**

Próxima Aula

- Complicado fazer layout/html usando Servlet!
  - Será que não tem um jeito mais simples?
  - Ahá! Existem as Java Server Pages (JSPs!)

PERGUNTAS?

Bom Descanso a Todos!