

Introdução à SERVLETS & JSP

Breno Alves Beirigo



Limitações do web server

1 - Gerar conteúdo dinâmico

- Um servidor web apenas retorna páginas estáticas.
- Uma aplicação auxiliar (CGI) é capaz de construir páginas dinamicamente e enviá-las ao servidor.

2 - Armazenar dados no servidor



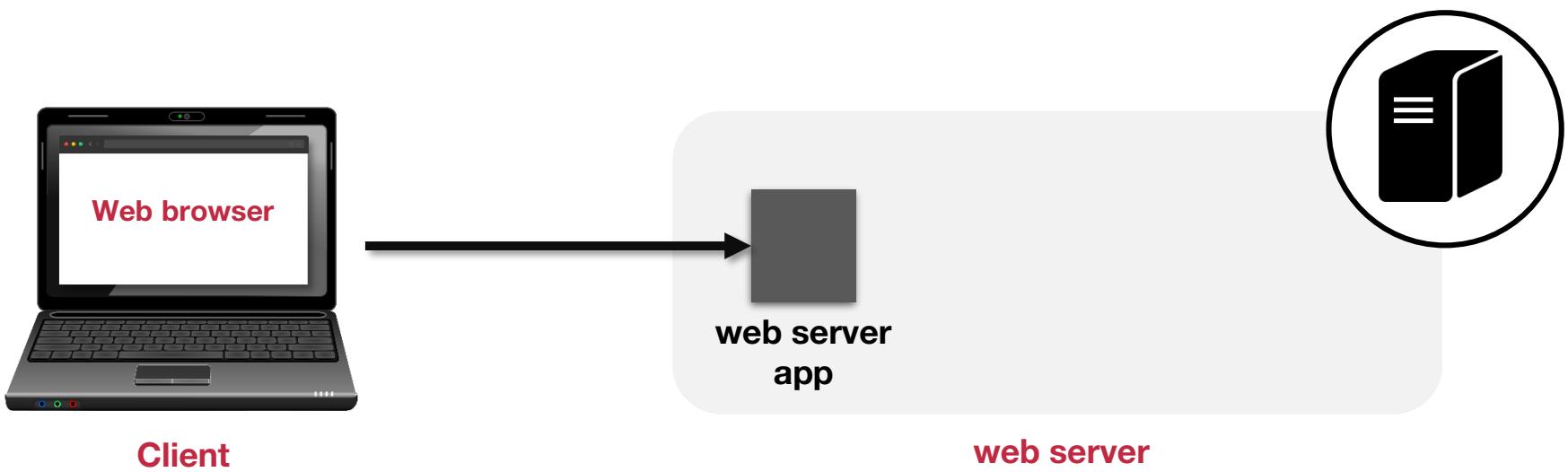
Common Gateway Interface (CGI)

Primeira tentativa de prover conteúdo dinâmico:

- Ambiente padrão que faz a interface entre o servidor web e programas executáveis capazes gerar páginas dinamicamente.
- Esses programas são armazenados em alguma pasta do servidor e codificados numa linguagem compatível com o SO.

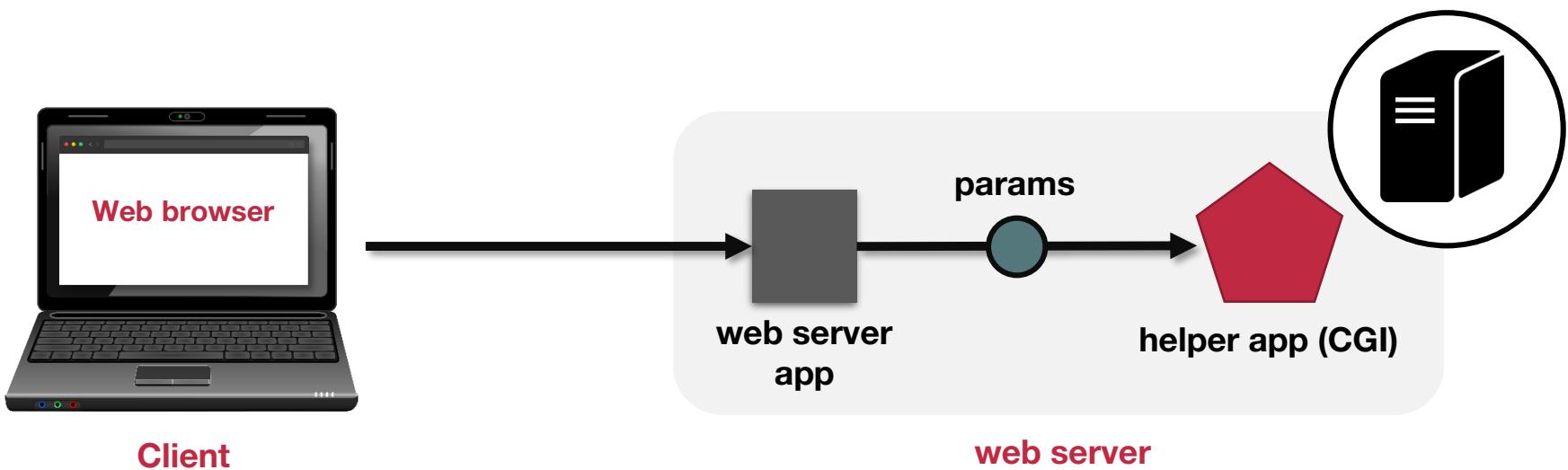


Common Gateway Interface (CGI)





Common Gateway Interface (CGI)

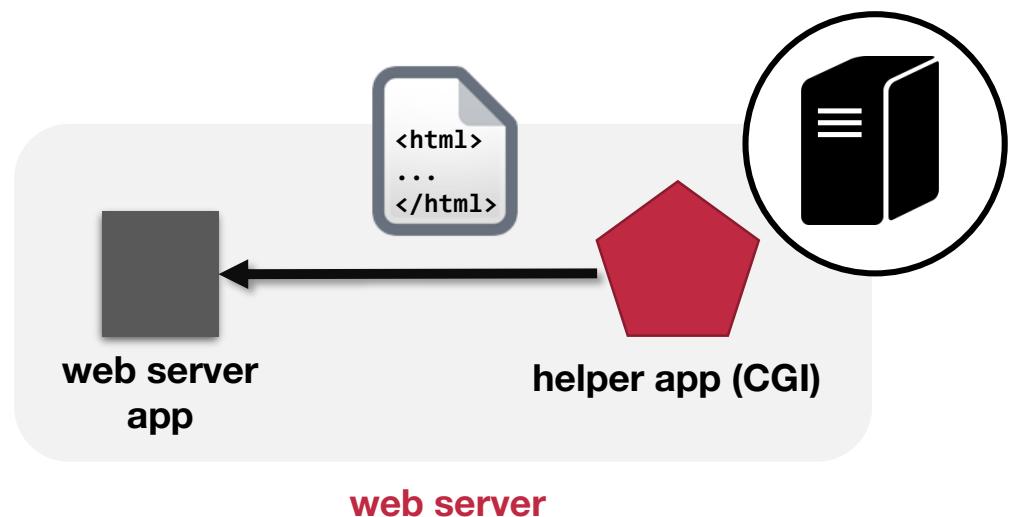




Common Gateway Interface (CGI)

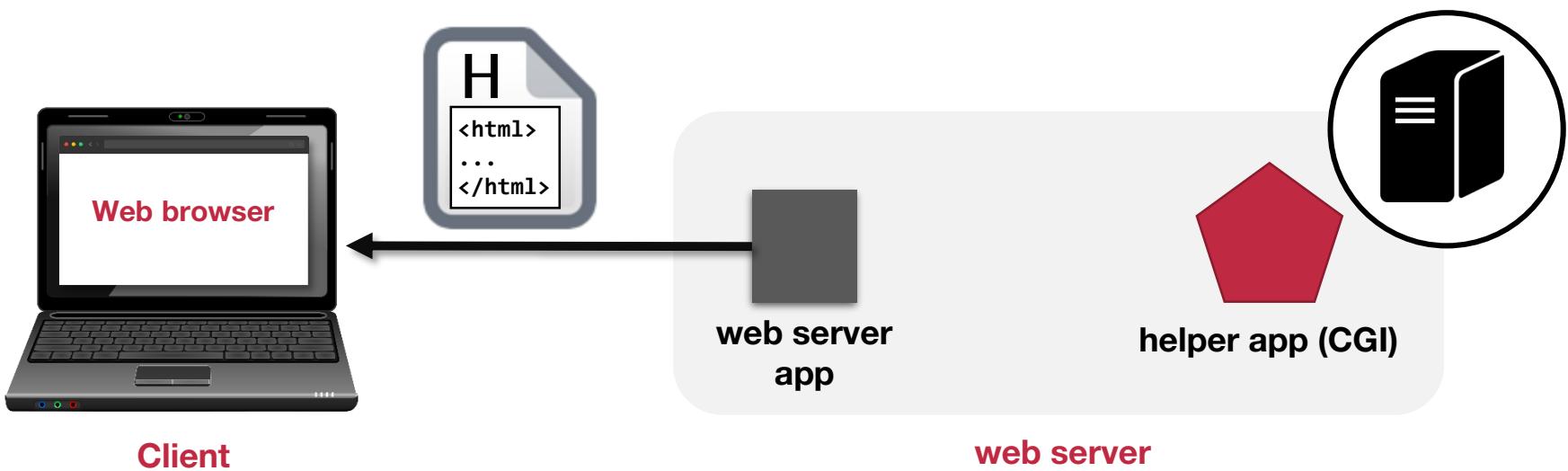


Client





Common Gateway Interface (CGI)





Servlet

- Implementação de Java que provê o mesmo serviço oferecido pelo CGI.
- Programa (já compilado) executado no servidor capaz de gerar conteúdo dinâmico.



Servlet

Vantagens:

- Independente de plataforma (Bytecode + JVM).
- Servlets utilizam um espaço de memória compartilhado na JVM.



JSP (Java Server Pages)

.java

```
class Time{  
    void main(){  
        Date x;  
        x = new Date();  
    }  
}
```

HEY BABY...
NICE TAGS.

.jsp

```
<html>  
    <body>  
        O tempo atual é  
        <%= new  
            java.util.Date()  
        %>  
    </body>  
</html>
```

html

```
<html>  
    <body>  
        Sempre é  
        4:20PM no  
        Servidor.  
    </body>  
</html>
```



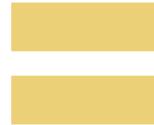
JSP (Java Server Pages)

JSP é uma linguagem interpretada capaz de gerar conteúdo dinâmico.

```
.java  
  
class Time{  
    void main(){  
        Date x;  
        x = new Date();  
    }  
}
```



```
.html  
  
<html>  
    <body>  
        Sempre é  
        4:20PM no  
        Servidor.  
    </body>  
</html>
```



```
.jsp  
  
<html>  
    <body>  
        O tempo atual é  
        <%= new  
        java.util.Date()  
        %>  
    </body>  
</html>
```



JSP & Servlets

- Servlets são mais rápidos do que programas em JSP.
- Scripts JSP podem ser compilados em Java Servlets.
- Atualmente, JSP e Java Servlets são combinados.



Exercício

Crie uma página HTML que envie um formulário via POST e via GET para um servlet. Seu formulário deve conter apenas 2 campos: nome e sobrenome. O servlet deve retornar uma página com os dados inseridos.

Envio via POST:

Nome:

Sobrenome:



Passou pelo POST

Nome do servlet: RecebeDados
Resultado: Michael Phelps

Envio via GET:

Nome:

Sobrenome:



Passou pelo GET

Nome do servlet: RecebeDados
Resultado: Michael Phelps



Introdução

- Quem chama os métodos `doPost()` e `doGet()` do servlet?
- Quem é responsável por passar os argumentos HTTP *request* e HTTP *response* para esses métodos?
- Quem gerencia a vida, morte e recursos de um servlet?



Introdução

- Quem chama os métodos `doPost()` e `doGet()` do servlet?
- Quem é responsável por passar os argumentos HTTP *request* e HTTP *response* para esses métodos?
- Quem gerencia a vida, morte e recursos de um servlet?

WEB CONTAINER



Web container

Responsável pela execução dos servlets.

Exemplos:





O que o container provê?

- Suporte à comunicação
 - Interface entre servidor web e servlets.
- Gerenciamento de ciclo de vida
 - Carregar classes, instanciar e iniciar servlets, invocar métodos dos servlets, suporte à *garbage collector*.
- Suporte à *multithreading*
 - Cada requisição a um servlet é tratada como uma thread Java.

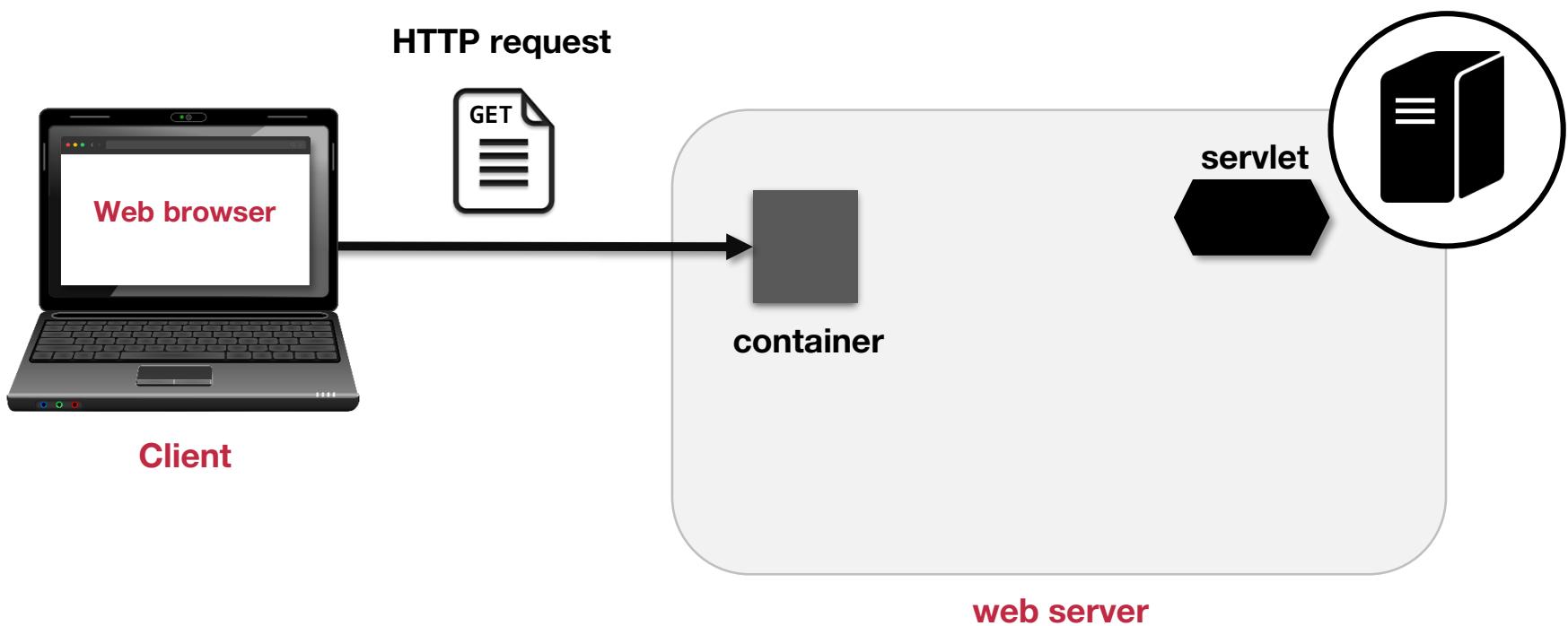


O que o container provê?

- Segurança declarativa
 - Parâmetros de segurança descritos num XML de implantação (gerencia de segurança sem recompilar código).
- Suporte à JSP



Tratamento de requisições

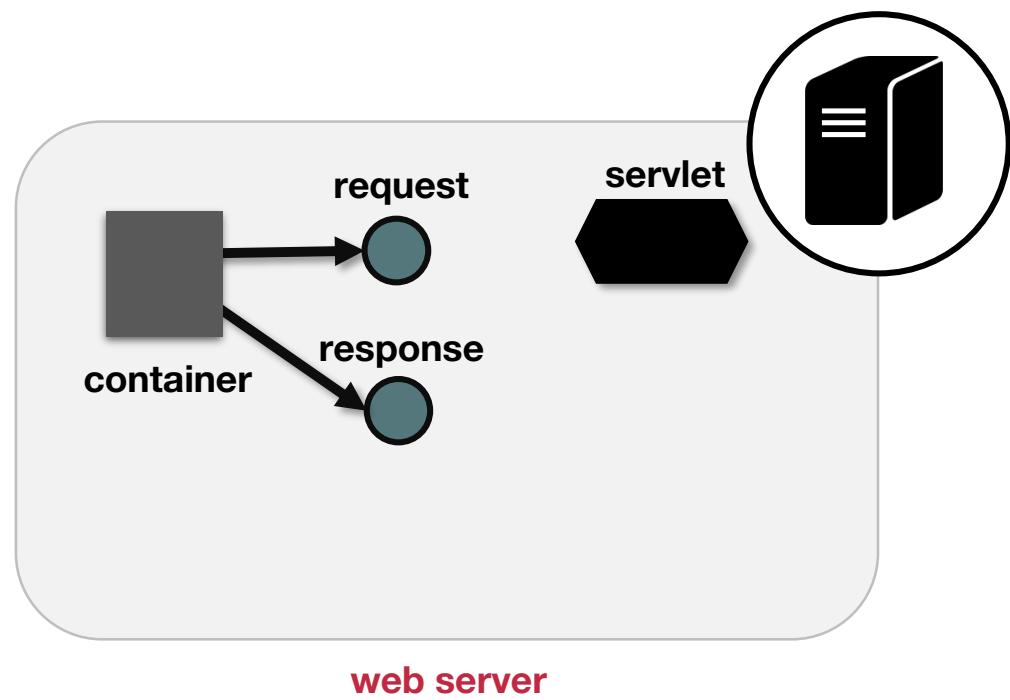




Tratamento de requisições

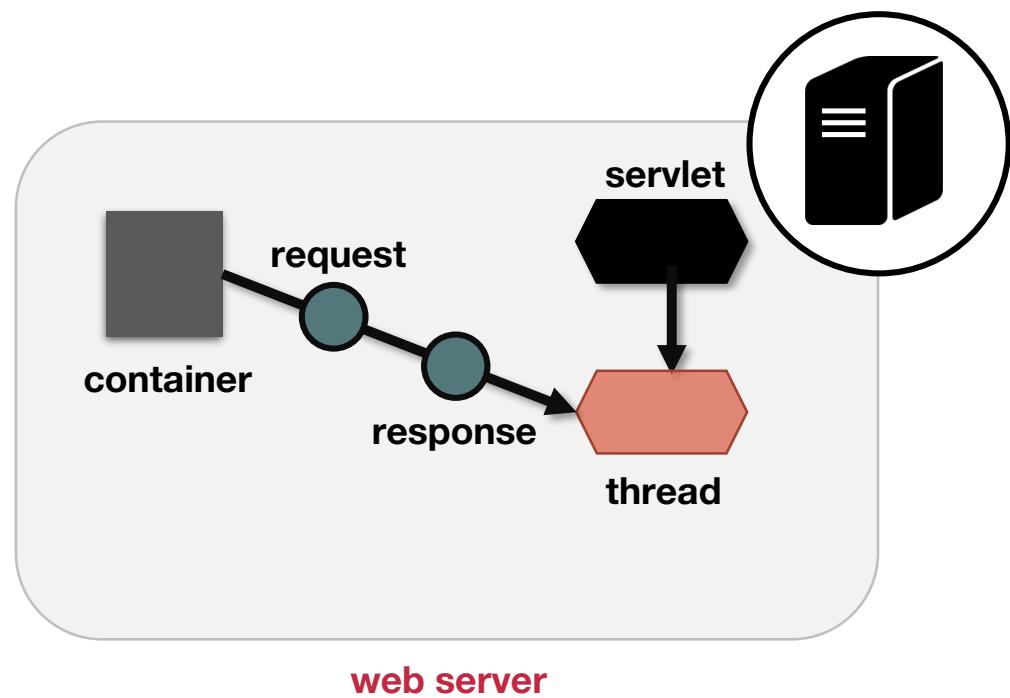


Client



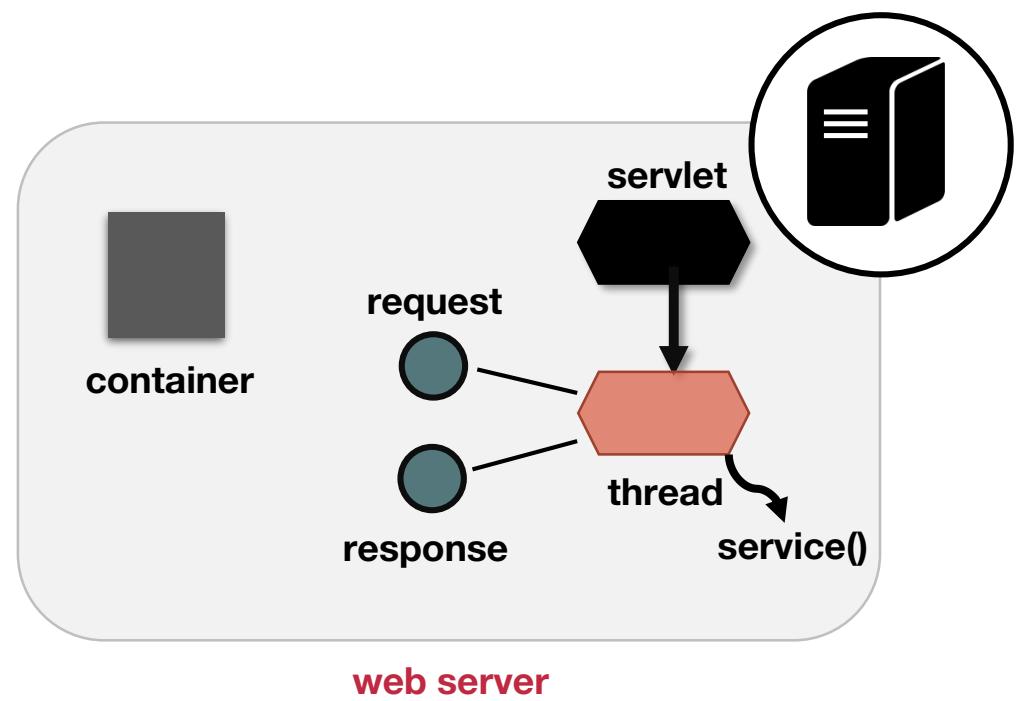


Tratamento de requisições



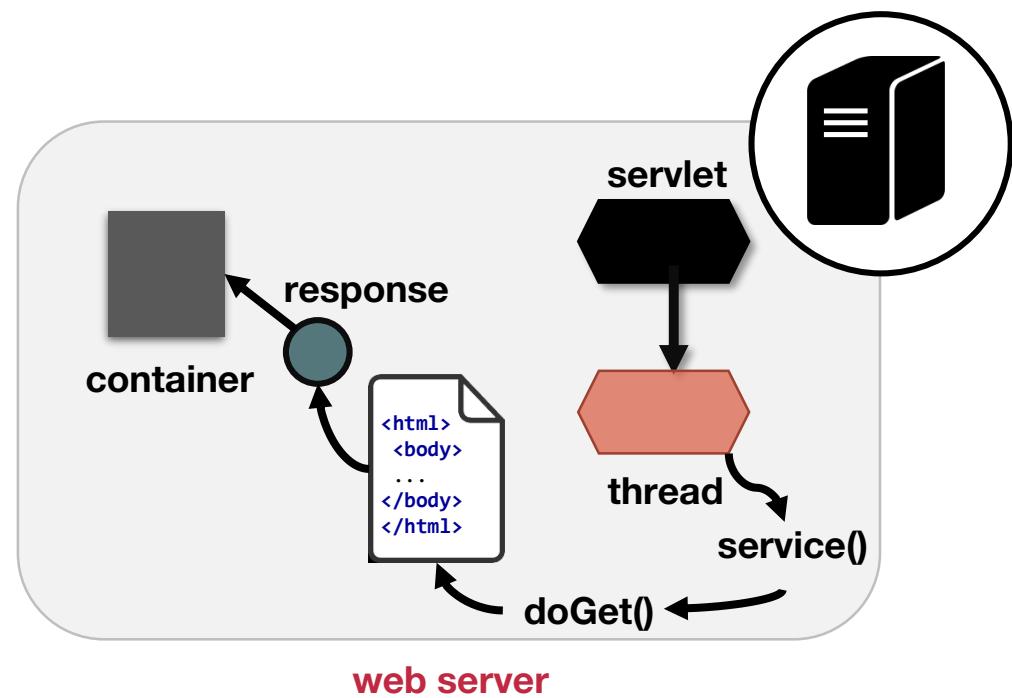


Tratamento de requisições



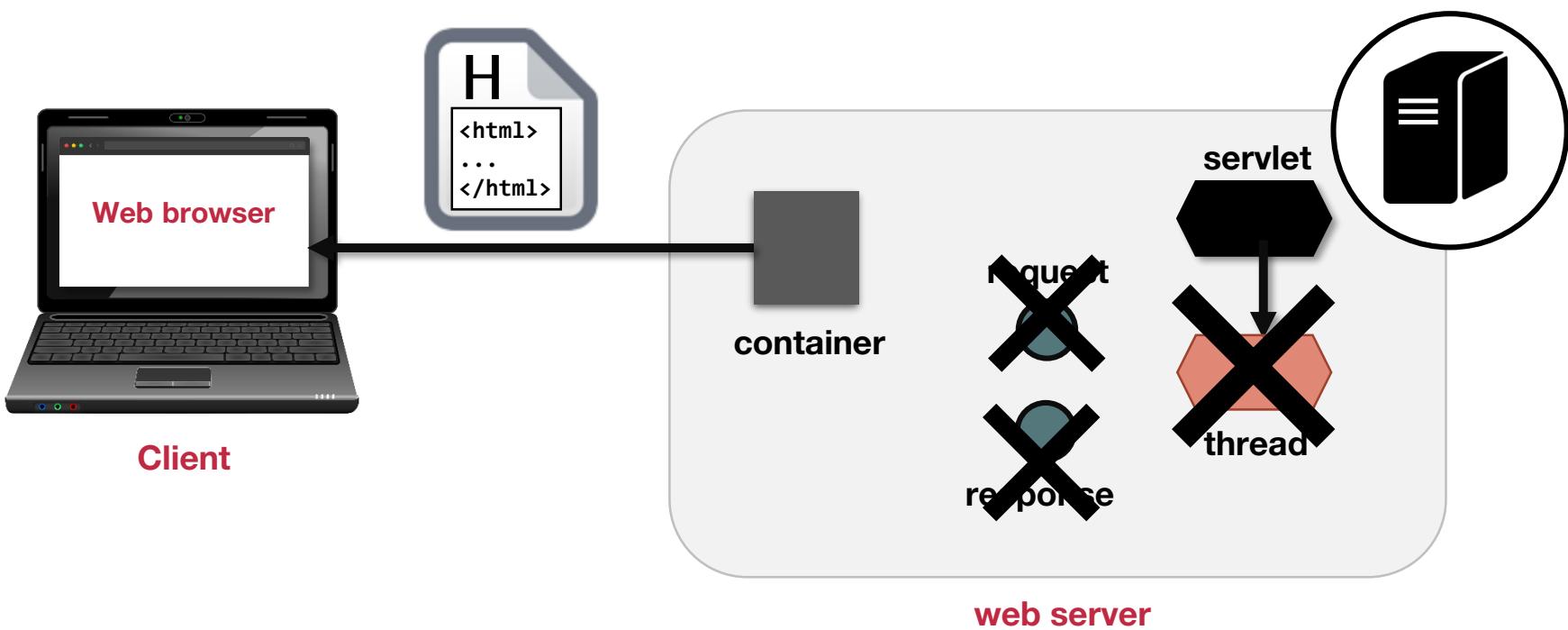


Tratamento de requisições





Tratamento de requisições





Mapeamento de nomes

A URL que faz parte da requisição do cliente é mapeada em um servlet específico no servidor. Um Servlet pode ter 3 nomes:

- Nome do caminho:
 - Ex.: classes/registration/SignUpServlet.class.
- Nome de implantação (deployment):
 - Ex.: EnrollServlet.
- URL público:
 - Ex.:register/registerMe



Por que mapear?

Flexibilidade

- Mudanças na estrutura de diretório não são percebidas por terceiros.

Segurança

- O cliente não deve ser capaz de enxergar a estrutura de diretório do servidor.



Deployment Descriptor

- Indica ao container como executar servlets e JSPs.
- Armazenado no arquivo **web.xml**. Exemplo:

```
<web-app ...>
  <servlet>
    <servlet-name>Internal name 1</servlet-name>
    <servlet-class>foo.Servlet1</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>Internal name 2</servlet-name>
    <servlet-class>foo.Servlet2</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Internal name 1</servlet-name>
    <url-pattern>/Public1</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Internal name 2</servlet-name>
    <url-pattern>/Public2</url-pattern>
  </servlet-mapping>
</web-app>
```

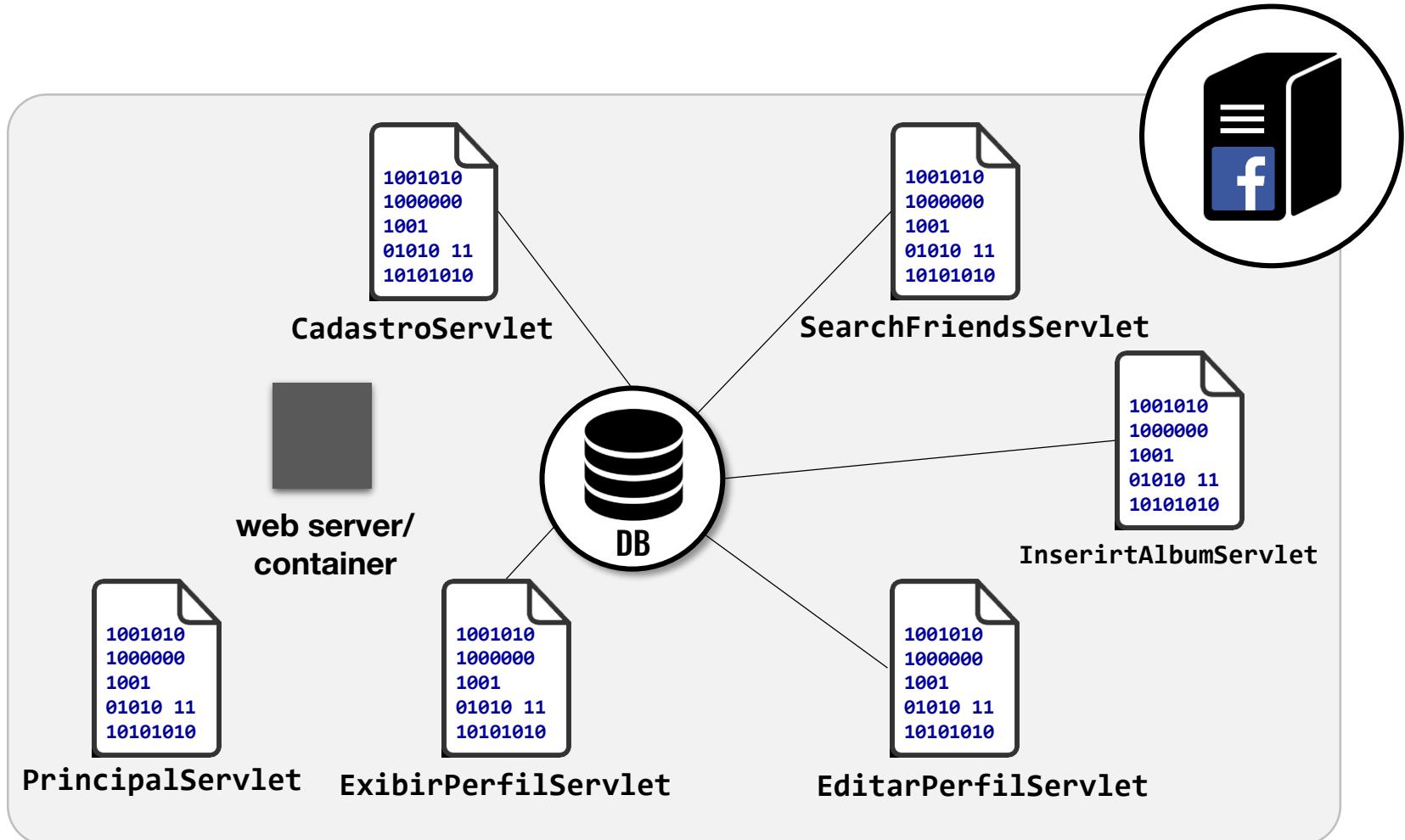


Resumo

- Um container provê ao aplicativo web: suporte à comunicação, gerencia do ciclo de vida, multithreading, segurança declarativa e suporte para JSPs.
- O container cria os objetos *request* e *response*. Esses objetos são usados pelos servlets para coletar informações sobre as requisições e mandar informações para os clientes.
- Um servlet é uma classe que estende `HttpServlet` e sobrescreve alguns serviços que correspondem a métodos HTTP invocados pelo browser (`doGet()`, `doPost()`, etc).
- O *deployer* pode mapear uma classe servlet para uma URL que o cliente pode usar para requisitar um servlet.

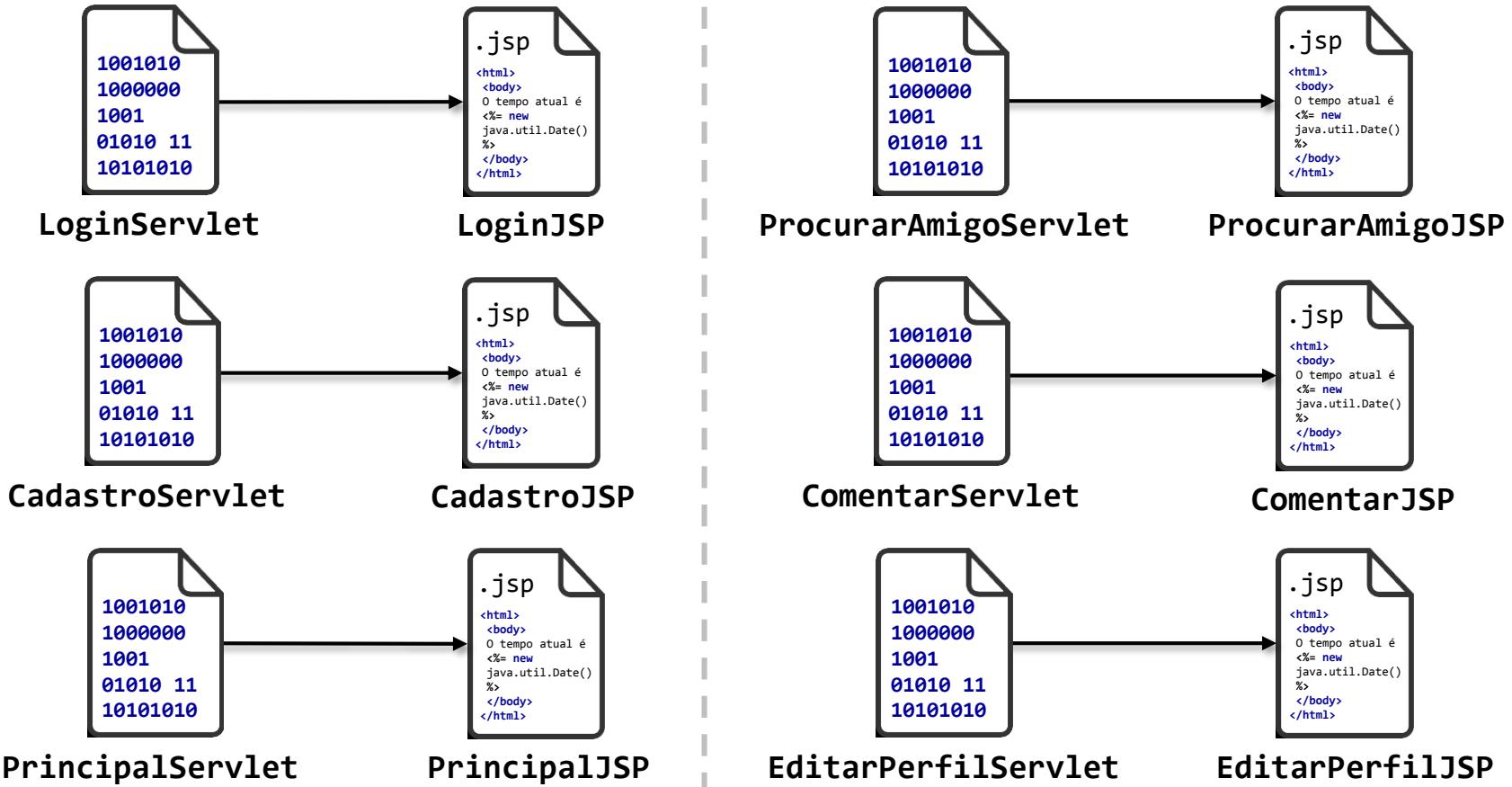


Exemplo de aplicação



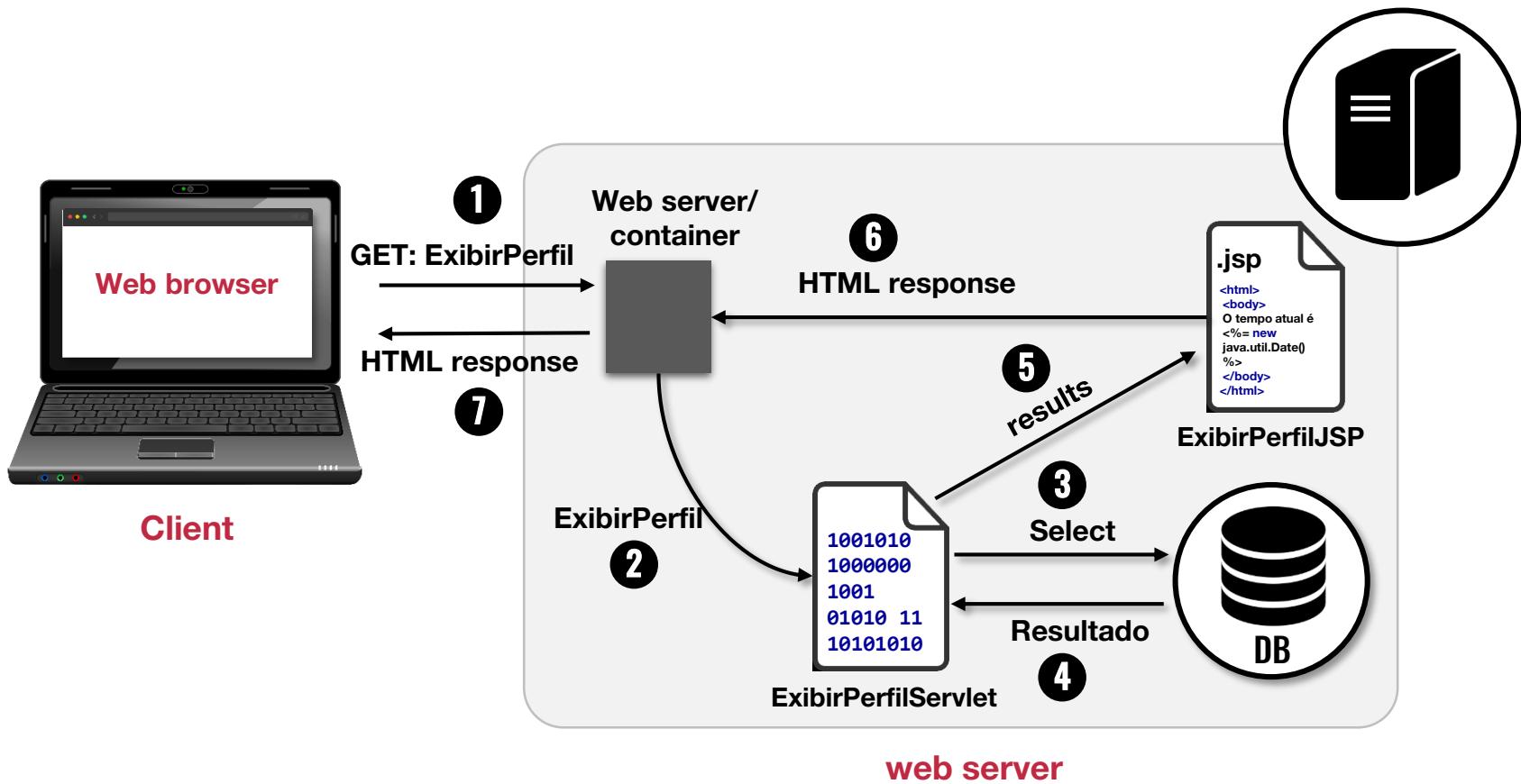


Exemplo de aplicação





Exemplo de aplicação





Servlet Request Dispatcher

Para enviar a resposta para um JSP:

- Adicionar um atributo no objeto de requisição (request) para o JSP usar. Exemplo:
 - `request.setAttribute("USER", user);`
- Instanciar um request dispatcher para o JSP:
 - `RequestDispatcher view;`
 - `view = request.getRequestDispatcher("result.jsp");`
- Encaminhar o pedido para o JSP:
 - `view.forward(request, response);`



Scriptlets

Código Java incorporado em HTML entre as tags <% e %>. Exemplo:

- Recuperar o atributo adicionado via servlet:
 - <% String userName = request.getAttribute("USER");%>
- Mostrar data atual:
 - <p>Data:<%= new Data() %></p>



JavaBeans

Classes que ajudam a separar a lógica do visual. Possuem:

- Interface `java.io.Serializable`
- Construtor sem argumentos;
- Métodos de acesso get e set.



JavaBeans

Um JavaBean:

- Implementa interface `java.io.Serializable`;
- Possui construtor sem argumentos;
- Possui métodos de acesso get e set.

Exemplo:

```
public class User implements java.io.Serializable {  
    private int id;  
    private String name;  
    public User() {}  
    public setId(int id) { this.id = id; }  
    public setName(int name) { this.name = name; }  
    public getId() { return this.id; }  
    public getName() { return this.name; }  
}
```



JavaBeans

Exemplo:

```
<jsp:useBean id="user" scope="page" class="model.User" />
<!-- Modificando valores -->
<!-- Scriptlets:-->
<% user.setId(1);%>
<jsp:setProperty name="user" property="name" value="kleber"/>

<!-- Regatando valores -->
<!-- Scriptlets:-->
<%= user.getName()%>
<!-- Expression Language:-->
${user.id}
<!-- action "jsp:getProperty":-->
<jsp:getProperty name="user" property="id" />
```



JavaBeans

Escopo:

- **page**: Objeto válido até a resposta ser enviada.
- **request**: Objeto disponível enquanto a execução do request.
- **session**: Objeto removido quando a sessão expira.
- **application**: Objeto acessível por páginas do mesmo servidor da aplicação. Removido quando a aplicação é interrompida.



Exemplo de aplicação

Qual é o único fator previsível em relação ao desenvolvimento de software?



Exemplo de aplicação

Qual é o único fator previsível em relação ao desenvolvimento de software?

MUDANÇA





Exemplo de aplicação

Uma vez que as especificações podem mudar:

- A lógica do negócio não deve estar dentro de um servlet.
- Não se deve assumir que o acesso se dará apenas da web.



Exemplo de aplicação

Uma vez que as especificações podem mudar:

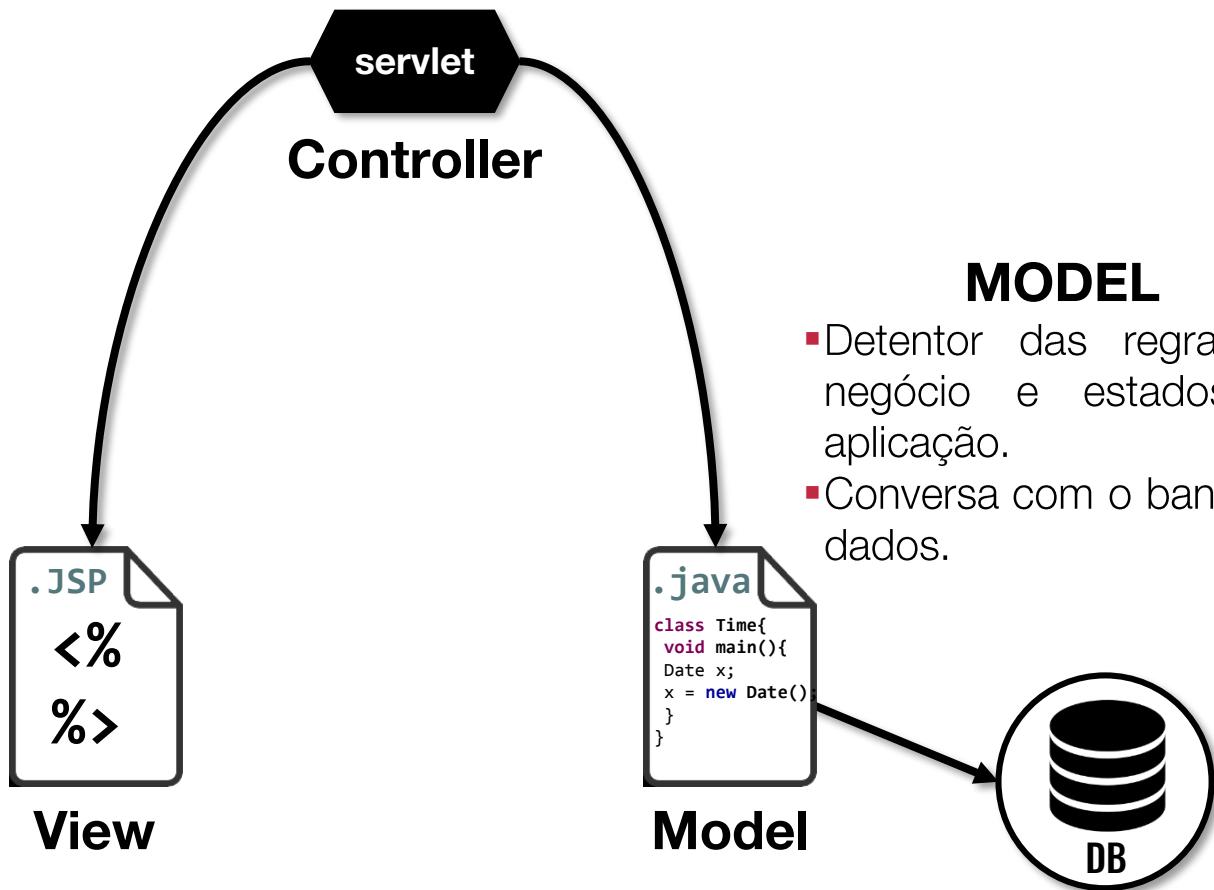
- A lógica do negócio não deve estar dentro de um servlet.
- Não se deve assumir que o acesso se dará apenas da web.

Solução

- Padrão de projeto MVC (Model-View-Controller).

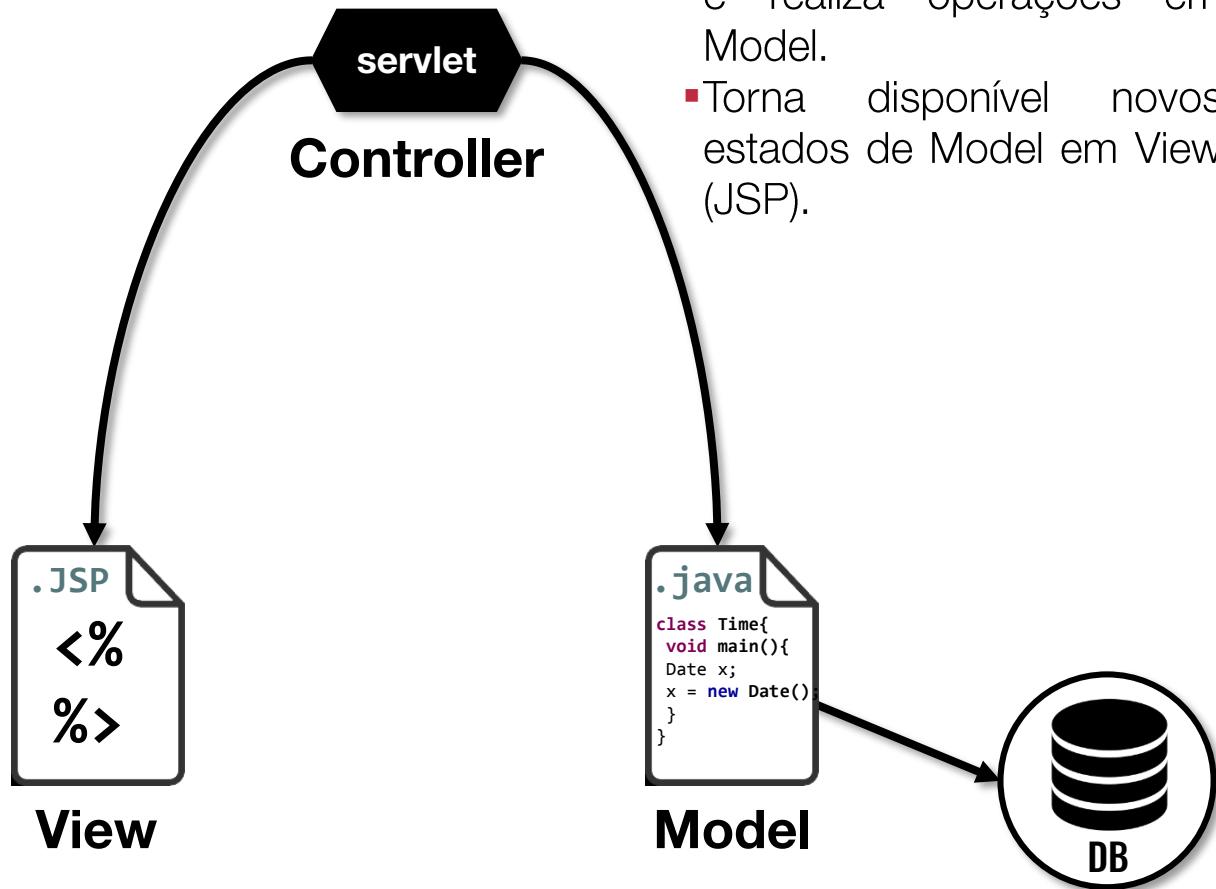


MVC / Servlet / JSP





MVC / Servlet / JSP



CONTROLLER

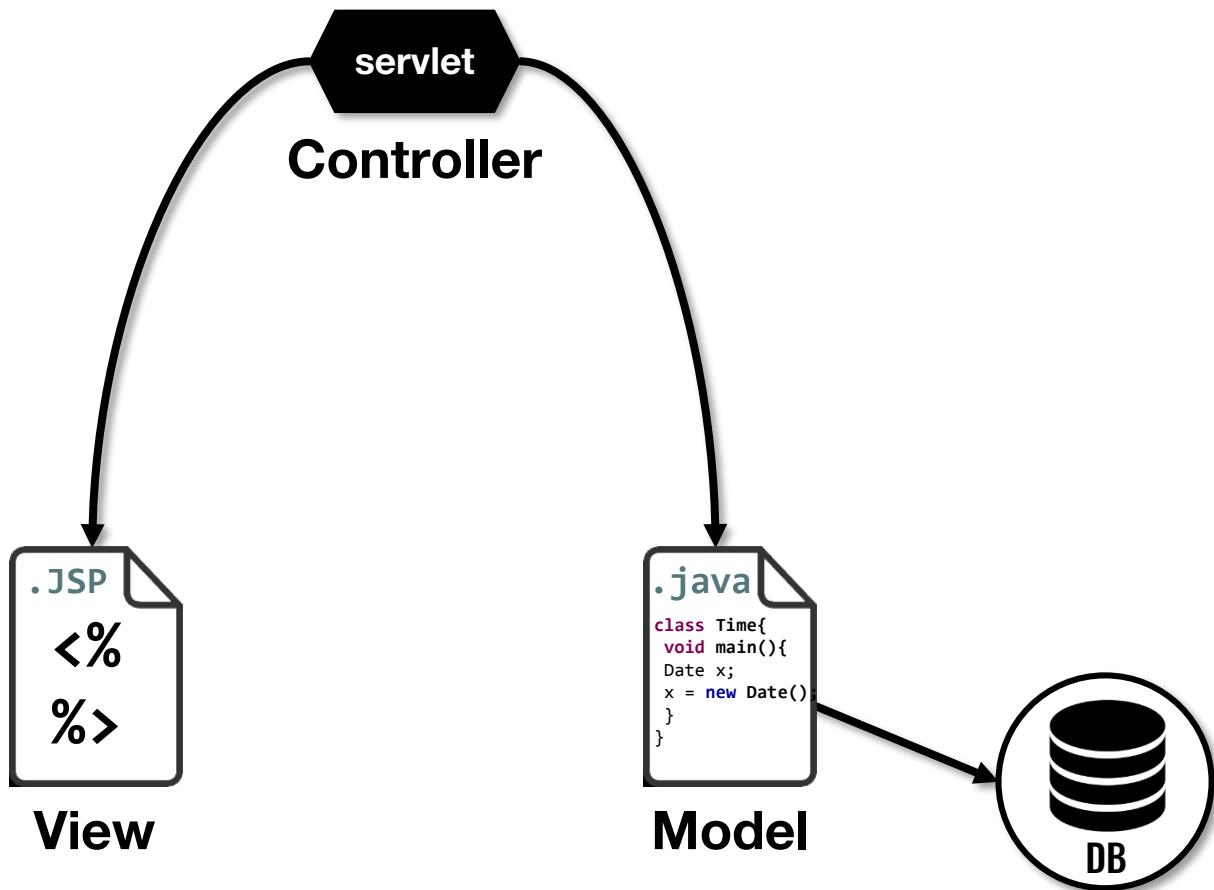
- Recebe entrada do usuário e realiza operações em Model.
- Torna disponível novos estados de Model em View (JSP).



MVC / Servlet / JSP

VIEW

- Responsável pela apresentação.
- Exibe o estado de Model recebido pelo Controller.
- Recebe dados de usuário e envia-os ao Controller.





MVC (Model View Controller)

Model 1 (*page-centric*)

- Construção de uma aplicação como um conjunto de páginas JSP.
- Indicado para aplicação pequena (baixo número de usuários e pouca lógica de negócios).



MVC (Model View Controller)

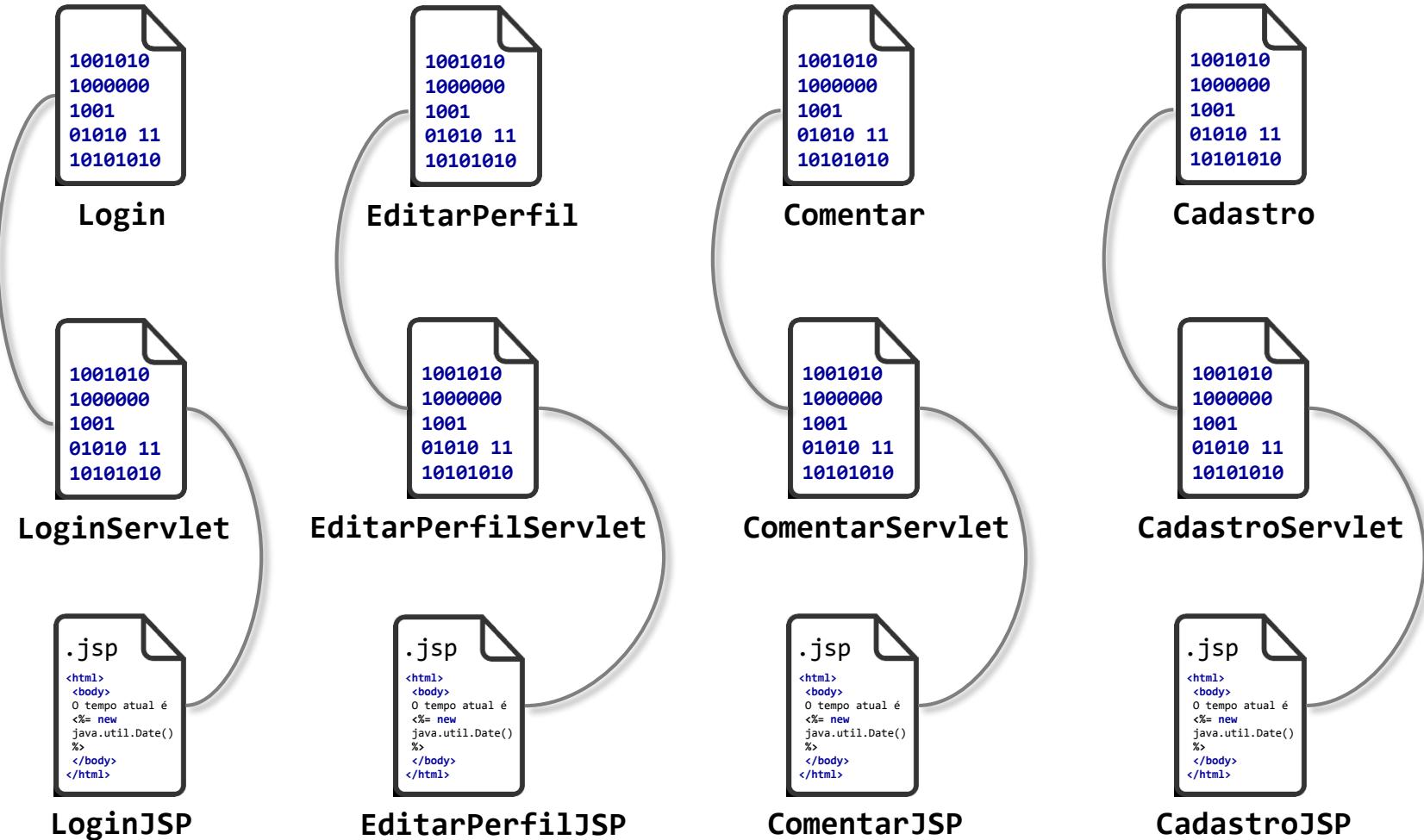
Model 2

- Servlet é usado como controlador(*controller*) :
 - Recebe pedidos de usuário;
 - Efetua mudanças no modelo (*model*) ;
 - Fornece a apresentação (*view*) ao usuário.



MVC (Model View Controller)

Exemplo:





Exercício

Crie uma página "login.jsp" com um formulário de login com os campos nome e senha e um botão de submissão. Ao clicar em submeter, seu formulário deve enviar os dados via POST para um servlet denominado "loginServ".

Se nome = "mphelps" e senha = "123":

- O servlet adiciona um atributo "nome" ao objeto request e redireciona para a página "logado.php". Essa pagina contém a seguinte mensagem: "Bem-vindo (--aqui você resgata o nome do usuário via objeto request --) !".

Caso contrário:

- O servlet adiciona um atributo "erro" (dados inválidos) ao objeto request e redireciona para a página "login.php". Abaixo do formulário de login, deve aparecer a mensagem de erro (usuário não encontrado).



Problemas

Cada lógica de negócios é associada a um servlet diferente.

- E se quiséssemos fazer um Log das chamadas dessas lógicas de negócio?





Problemas

A lógica genérica da aplicação é duplicada em cada servlet:

- Se algo precisa mudar, todos os servlets devem ser alterados.
- O sistema possui várias portas de entrada. Ex.:
 - 10 classes de modelo e cinco lógicas diferentes resultam em 50 formas de acesso.



Solução 1

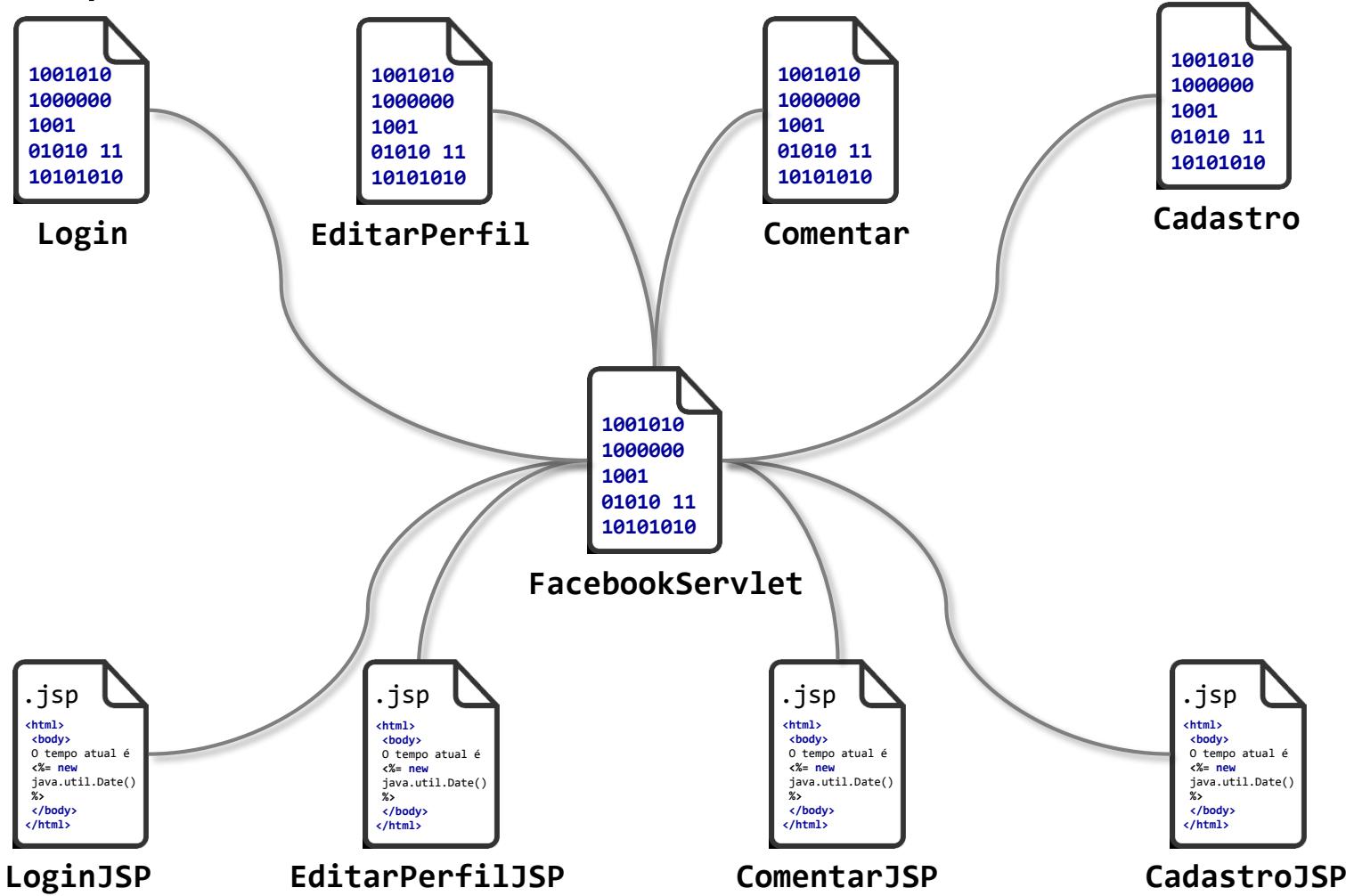
Colocar todo o acesso em apenas um servlet:

```
@WebServlet(urlPatterns = {"/sistema"})
public class SistemaTodoServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String acao = request.getParameter("logica");
        RequestDispatcher rd;
        ContatoDao dao = new ContatoDao();
        if (acao.equals("AdicionaContato")) {
            Contato contato = new Contato();
            contato.setNome(request.getParameter("nome"));
            contato.setEndereco(request.getParameter("endereco"));
            contato.setEmail(request.getParameter("email"));
            dao.adiciona(contato);
            Rd = request.getRequestDispatcher("/contato-adicionado.jsp");
            rd.forward(request, response);
        } else if (acao.equals("ListaContatos")) {
            // busca a lista no DAO // despacha para um jsp
        } else if (acao.equals("RemoveContato")) {
            // faz a remoção e redireciona para a lista
        }
    }
}
```



Solução 1

Exemplo:





Solução 1

Acessar no navegador:

- Para adicionar contatos:

<http://localhost:8080/Projeto/sistema?logica=AdcionaContato>

- Para listar contatos:

<http://localhost:8080/Projeto/sistema?logica=ListaContato>

- Para remover contatos:

<http://localhost:8080/Projeto/sistema?logica=RemoveContato>



Solução 1

Acessar no navegador:

- Para adicionar contatos:

<http://localhost:8080/Projeto/sistema?logica=AdicionaContato>

- Para listar contatos:

<http://localhost:8080/Projeto/sistema?logica=ListaContato>

- Para remover contatos:

<http://localhost:8080/Projeto/sistema?logica=RemoveContato>



Solução 1

Acessar no navegador:

- Para adicionar contatos:

<http://localhost:8080/Projeto/sistema?logica=AdicionaContato>

- Para listar contatos:

<http://localhost:8080/Projeto/sistema?logica=ListaContato>

- Para remover contatos:

<http://localhost:8080/Projeto/sistema?logica=RemoveContato>

PROBLEMA: Servlet muito grande: regras de negócio do sistema inteiro.



Solução 1.2

Extrair as lógicas para diferentes classes.

Exemplo:

```
if (acao.equals("AdicionaContato")) {  
    new AdicionaContato().executa(request, response);  
} else if (acao.equals("ListaContatos")) {  
    new ListaContato().executa(request, response);  
} else if (acao.equals("RemoveContato")) {  
    new RemoveContato().executa(request, response);  
}
```



Solução 1.2

Extrair as lógicas para diferentes classes.

Exemplo:

```
if (acao.equals("AdicionaContato")) {  
    new AdicionaContato().executa(request, response);  
} else if (acao.equals("ListaContatos")) {  
    new ListaContato().executa(request, response);  
} else if (acao.equals("RemoveContato")) {  
    new RemoveContato().executa(request, response);  
}
```

PROBLEMA: A cada lógica nova, lógica removida, alteração, etc., é preciso alterar o servlet.



Solução Final

Generalizar as chamadas de acordo com os parâmetros recebidos.

```
@WebServlet("/sistema")
public class ControllerServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String parametro = request.getParameter("logica");
        String nomeDaClasse = "caminho.da.classe.logica." + parametro;
        try {
            Class<?> classe = Class.forName(nomeDaClasse);
            Logica logica = (Logica) classe.newInstance();
            // Recebe o String após a execução da lógica
            String pagina = logica.executa(request, response);
            // Faz o forward para a página JSP
            request.getRequestDispatcher(pagina).forward(request, response);
        } catch (Exception e) {
            throw new ServletException( "A lógica de negócios causou uma
exceção", e);
        }
    }
}
```



Solução Final

Toda lógica de negócios vai seguir um "contrato" de execução:

```
public interface Logica {  
    String executa(HttpServletRequest req, HttpServletResponse res) throws  
    Exception;  
}
```

A lógica executa o serviço e retorna o JSP onde o resultado será mostrado.

```
public class PrimeiraLogica implements Logica {  
    String executa(HttpServletRequest req, HttpServletResponse res) throws  
    Exception{  
        System.out.println("Executando a primeira lógica e redirecionando...");  
        return "primeiraLogica.jsp";  
    }  
}
```