

# Protocolo HTTP

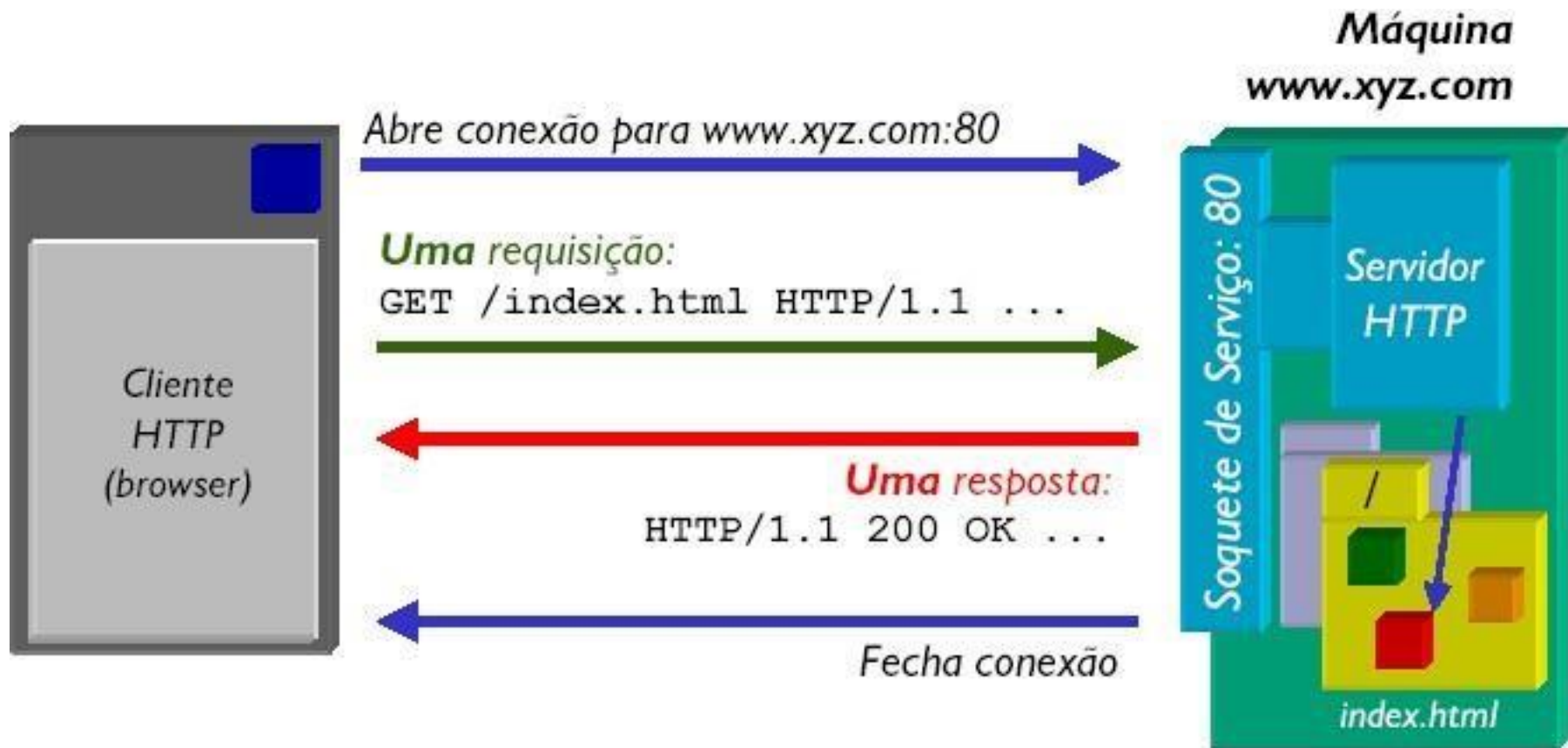


## Protocolo HTTP

- A web que conhecemos hoje funciona com o protocolo HTTP, que significa Hyper Text Transfer Protocol;
- HTTP é um protocolo, bastante simples, de transferência de dados;
- É um padrão, definido pela W3C (http 1.1 → RFC 2616)



# Protocolo HTTP



# Protocolo HTTP

- HTTP é um protocolo stateless, ou seja, não mantém estado;
- Cada requisição estabelece uma conexão com o servidor, recebe os dados e fecha a conexão.



## HTTP - Requisição

- Quando um cliente faz uma requisição, ele especifica um comando, chamado método, que especifica o tipo de ação que ele quer efetuar;
- A primeira linha da requisição também especifica a URL e a versão do protocolo:

```
GET /index.jsp HTTP/1.0
```



## HTTP - Requisição

- Após enviado o comando, o cliente pode enviar informações adicionais (tipo do browser, versão, etc):

```
GET /index.jsp HTTP/1.0
```

```
User-Agent: Mozilla/4.0 (compatible; MSIE 4.0;  
Windows 95)
```

```
Accept: image/gif, image/jpeg, text/*, */*
```



## HTTP - Requisição

- A requisição também pode ser feita com um método POST

```
POST /index.jsp HTTP/1.0
```

```
User-Agent: Mozilla/4.0 (compatible; MSIE 4.0;  
Windows 95)
```

```
Accept: image/gif, image/jpeg, text/*, */*
```



## HTTP - Requisição

Qual a diferença entre o método GET e o POST?

- GET, envia dados através da URI (tamanho limitado)

`<uri>?dados`

- Um exemplo de um site que usa GET para enviar informações:

<http://www.google.com/search?q=cejug>





## HTTP - Requisição

Qual a diferença entre o método GET e o POST?

- POST envia os dados para o servidor como um fluxo de bytes – dados não aparecem na URL;
- Útil para grandes formulários a informação não deve ser exibida na URL (senhas, por exemplo);
- Desvantagem – usuário não pode fazer bookmarks da URL.



## HTTP - Resposta

- Feita uma requisição, o servidor envia uma resposta;
- A primeira linha da resposta indica um código de status e sua descrição

**HTTP/1.0 200 OK**



## HTTP - Resposta

Alguns códigos HTTP:

- 200 → Código default, sucesso;
- 400 → Erro de sintaxe na requisição;
- 404 → Recurso não existe no servidor;
- 405 → Método não permitido.



Para consultar a lista completa, acesse:

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.ht>



## HTTP - Resposta

- Após a linha de status, o servidor envia os cabeçalhos de resposta:

Date: Saturday, 23-May-00 03:25:12 GMT

Server: Tomcat Web Server/3.2

MIME-version: 1.0

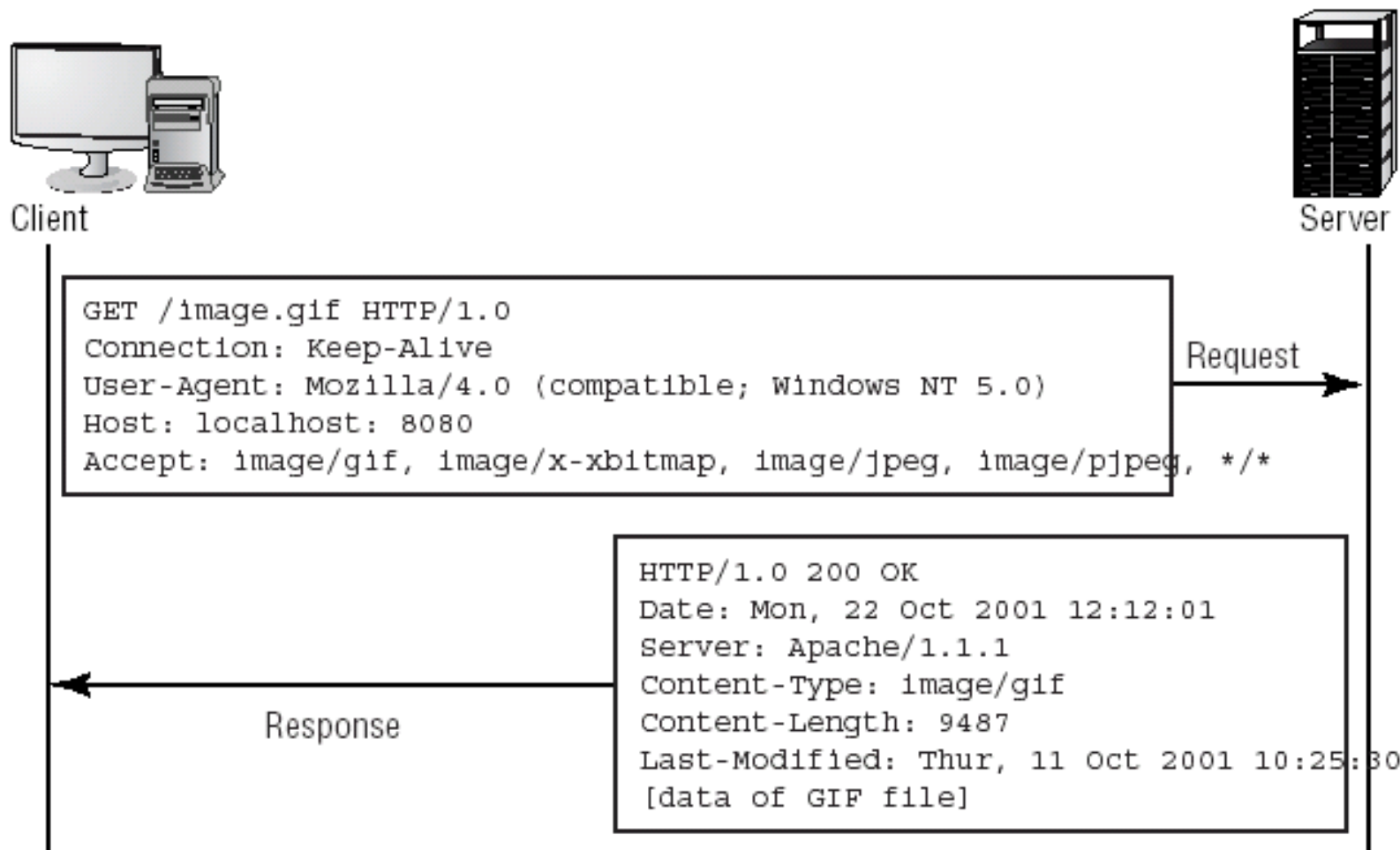
Content-type: text/html

Content-length: 1029

Last-modified: Thursday, 7-May-00 12:15:35 GMT



# HTTP – Resumindo



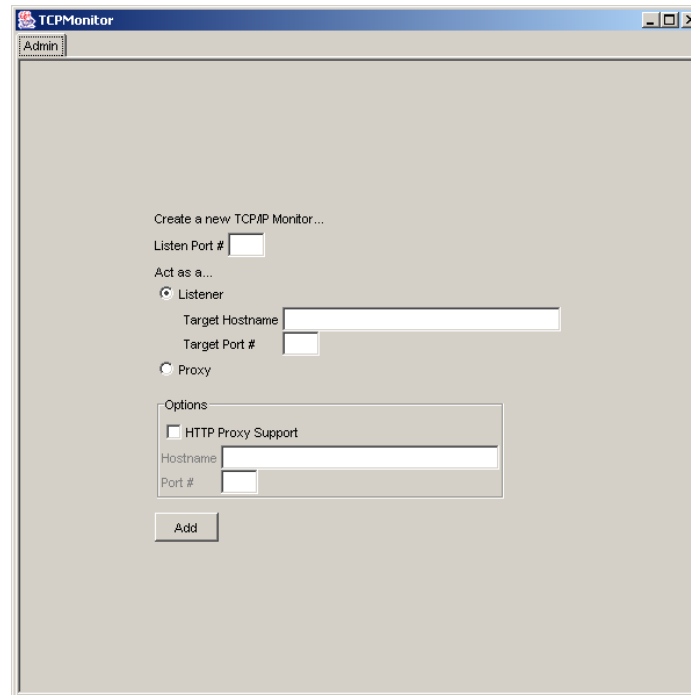
## TCP Monitor

- Uma boa forma de entender o protocolo HTTP é vê-lo funcionando;
- O programa TCP Monitor exibe as requisições feitas pelos clientes e a resposta feita por um servidor;
- O TCP Monitor é um utilitário, parte do projeto AXIS, do projeto Jakarta, e pode ser baixado em <http://ws.apache.org/axis/>



# Configurando o TCP Monitor

- Primeiro passo – carregar o programa:  
`java -cp axis.jar org.apache.axis.utils.tcpmon`



The screenshot shows the TCPMonitor application window with a title bar and a menu bar containing 'Admin'. The main area is titled 'Create a new TCP/IP Monitor...'. It features a 'Listen Port #' field, an 'Act as a...' section with radio buttons for 'Listener' (selected) and 'Proxy', and an 'Options' section with a checkbox for 'HTTP Proxy Support'. Below these are fields for 'Target Hostname', 'Target Port #', 'Hostname', and 'Port #'. An 'Add' button is at the bottom.



# Configurando o TCP Monitor

## Segundo passo – configurar

Create a new TCP/IP Monitor...

Listen Port #

Act as a...

☒ Listener

Target Hostname

Target Port #

☐ Proxy

Options

☐ HTTP Proxy Support

Hostname

Port #

Porta que será criada  
na nossa máquina

O host para qual a  
requisição será  
direcionada

Clique em adicionar  
depois de configurar





## Configurando o TCP Monitor

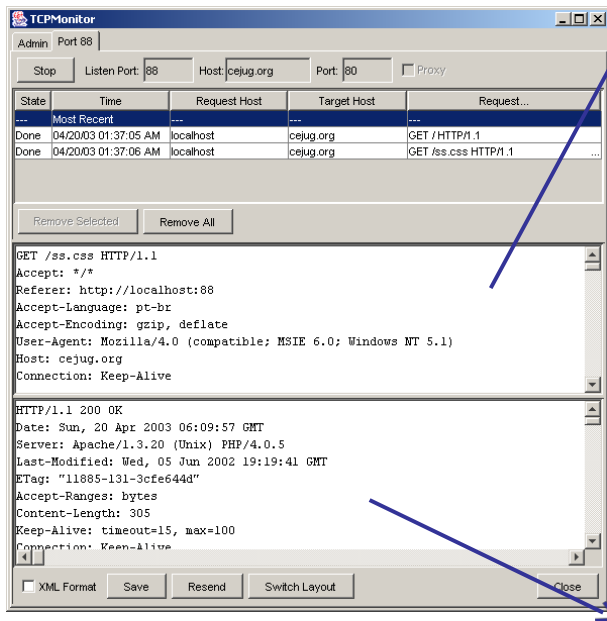
Terceiro passo – faça uma requisição normalmente do seu browser para a porta configurada:

`http://localhost:88`



# Configurando o TCP Monitor

E finalmente, veja os detalhes da requisição HTTP:



```
GET /ss.css HTTP/1.1
Accept: */*
Referer: http://localhost:88
Accept-Language: pt-br
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: cejug.org
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Date: Sun, 20 Apr 2003 06:09:57 GMT
Server: Apache/1.3.20 (Unix) PHP/4.0.5
Last-Modified: Wed, 05 Jun 2002 19:19:41 GMT
ETag: "11885-131-3cfe644d"
Accept-Ranges: bytes
Content-Length: 305
Keep-Alive: timeout=15, max=100
```



# Trabalhando com dados de formulários



## Dados vindos de formulários

- Conforme vimos anteriormente, dados podem ser postados utilizando os métodos GET e POST;
- Iremos aprender nesse módulo como “capturar” esses dados nos nossos servlets.



# Relembrando HTML e Formulários...

Para onde o formulário  
será submetido

Como o formulário  
será submetido

`<html>`

`<form action="meuServlet" method="get">`

`<input name="nome" type="text"  
value="Digite seu nome aqui!" size="40"  
maxlength="40">`

`</form>`

`</html>`



## Dados vindos de formulários

- No exemplo anterior, o formulário irá postar os dados para uma URL, no mesmo servidor, chamada meuServlet;
- Precisamos criar um servlet para receber os dados desse formulário
- Já sabemos criar um servlet, mas como receber os parâmetros passados?



# Relembrando o HelloWorld

```
import java.io.*; import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet
{
    public void doGet ( HttpServletRequest request,
                        HttpServletResponse response )
        throws ServletException, IOException
    {
        PrintWriter out;
        response.setContentType("text/html");
        out = response.getWriter();
        out.println( "Hello World!" );
        out.close();
    }
}
```



## A Interface HttpServletRequest

- A instância de HttpServletRequest passada como parâmetro para o método doGet encapsula os dados vindos da requisição;
- Através dela poderemos saber todos os dados relativos a requisição;
- A interface HttpServletRequest herda de ServletRequest.





## A Interface ServletRequest

Principais métodos:

- `getParameter( String name )` →
  - Retorna o valor relativo a um determinado parâmetro passado;
  - Esse parâmetro pode ser um dado de um formulário submetido ou um dado passado na URL;
  - O retorno de `getParameter` sempre é `String`!
  - **Cuidado** → O nome do parâmetro é sensitive case.
  - **Cuidado** → Se o parâmetro não existir na requisição, `getParameter` retorna `null`.



## ...A Interface ServletRequest

Principais métodos:

- `getParameterValues( String name )` →
  - Retorna os valores relativos a um determinado parâmetro passado;
  - Use esse método quando um dado de um formulário tiver mais de um valor (um “select”, por exemplo)
  - O retorno de `getParameterValues` é um array de Strings.
  - **Cuidado** → O nome do parâmetro é sensitive case.
  - **Cuidado** → Se o parâmetro não existir na requisição, `getParameterValues` retorna null.



## ...A Interface ServletRequest

Principais métodos:

- `java.util.Map getParameterMap()` →
  - Retornar todos os parâmetros passados na requisição, no formato de um Map.



## ...A Interface ServletRequest

Principais métodos:

- `getMethod()` →
  - Retorna o método utilizado para submeter os dados (GET, POST ou PUT).
- `getQueryString()` →
  - Retorna a query postada (URL).
- `getParameterNames()` →
  - Retorna um Enumeration de Strings contendo o nome de todos os parâmetros passados na requisição.



## Construindo um formulário

```
<html>
<form action="./servlet/Formulario" method="get">
  <p> <input name="nome" type="text" value="Digite
seu nome aqui!" size="40" maxlength="40"> </p>
  <p>Interesses:</p>
  <SELECT NAME="preferencias" MULTIPLE>
    <OPTION VALUE="1"> Musica </OPTION>
    <OPTION VALUE="2"> Computadores </OPTION>
    <OPTION VALUE="3"> Ocio </OPTION>
  </SELECT>
  <p> <input type="submit" name="Submit"
value="Submit"> </p>
</form>
</html>
```



```
public class Formulario extends HttpServlet
{
    public void doGet ( HttpServletRequest request,
                        HttpServletResponse response
                        ) throws ServletException, IOException {
        PrintWriter out;
        response.setContentType("text/html");
        out = response.getWriter();
        out.println("<HTML><HEAD><TITLE></TITLE>");
        String nome = request.getParameter("nome");
        out.println("Nome --> " + nome + "<p>");
        String[] dados = request.getParameterValues("preferencias");
        if ( dados == null )
            out.println("Dados é null." );
        else
            for( int i=0; i < dados.length; i++ )
                out.println( dados[i] );
    }
}
```



## Entendido o exemplo

- Os dados do formulário são submetidos via GET;
- Isso significa que também podemos disparar o servlet fazendo uma requisição na URL do tipo:

```
http://localhost:8080/curso/servlet/Formulario  
?nome=blabla&preferencias=1&  
preferencias=2
```



## Alterando o exemplo

- Vamos alterar o exemplo anterior para que ele exiba todos os parâmetros passados;
- Vamos utilizar o método `getParameterNames` para descobrir quais parâmetros foram passados para o formulário;
- E o `getParameterValues` para obter os valores da requisição.





```
public class Formulario2 extends HttpServlet
{
    public void doGet ( HttpServletRequest request, HttpServletResponse
response )    throws IOException
    {
        PrintWriter    out;
        response.setContentType("text/html");
        out = response.getWriter();
        Enumeration paramNames = request.getParameterNames();
        while(paramNames.hasMoreElements())
        {
            String paramName = (String)paramNames.nextElement();
            out.println( "<p><b> " + paramName + "</b><p>");

            String[] paramValues = request.getParameterValues(paramName);
            for( int i=0; i < paramValues.length; ++i )
            {
                out.print( "--> " + paramValues[i]);
            }
        }
    }
}
```

## Suportando Post

- Para fazer com que o servlet suporte GET, implementamos o método `doGet`;
- Para fazer com que o servlet suporte POST, devemos implementar o método `doPost`.

```
public void doPost( HttpServletRequest  
    request,  
                    HttpServletResponse response  
    )
```



## Suportando Post

- Normalmente, é interessante suportar tanto get quanto post;
- Para isso, é só implementar os dois métodos

```
public void doPost( HttpServletRequest request,  
                   HttpServletResponse response  
                   ) { ... }  
public void doGet ( HttpServletRequest request,  
                   HttpServletResponse response  
                   ) { ... }
```

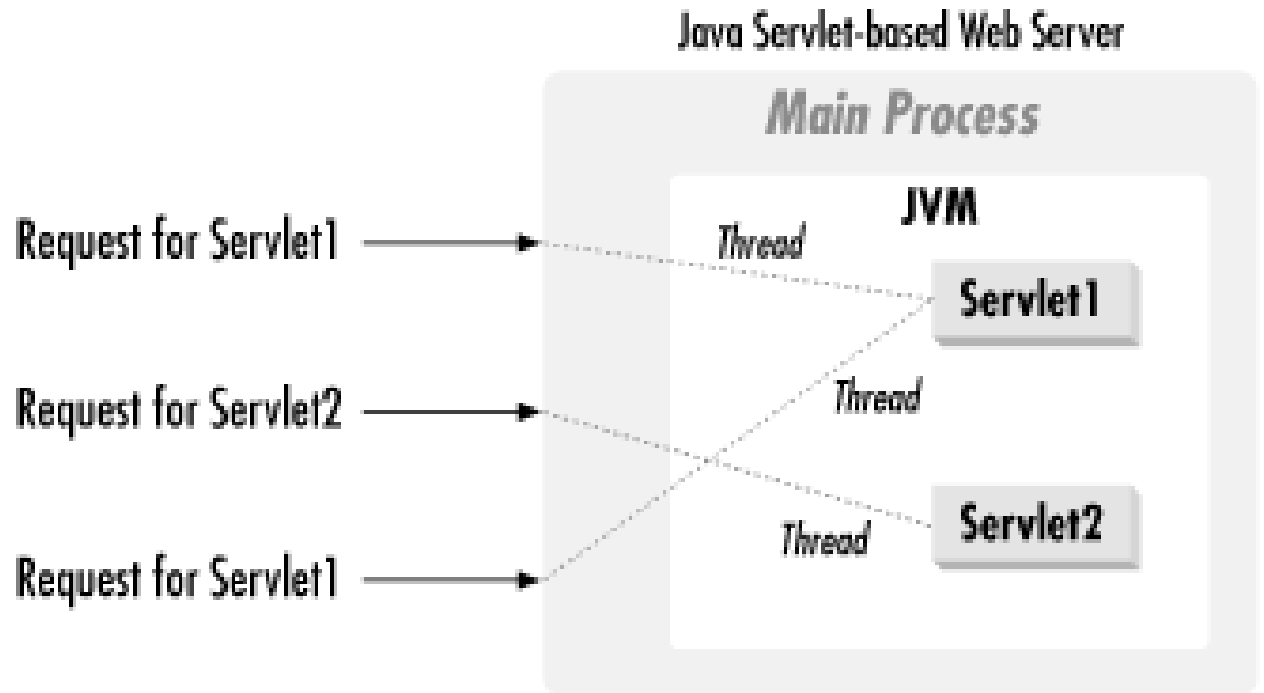


# Processamento e ciclo de vida de um Servlet

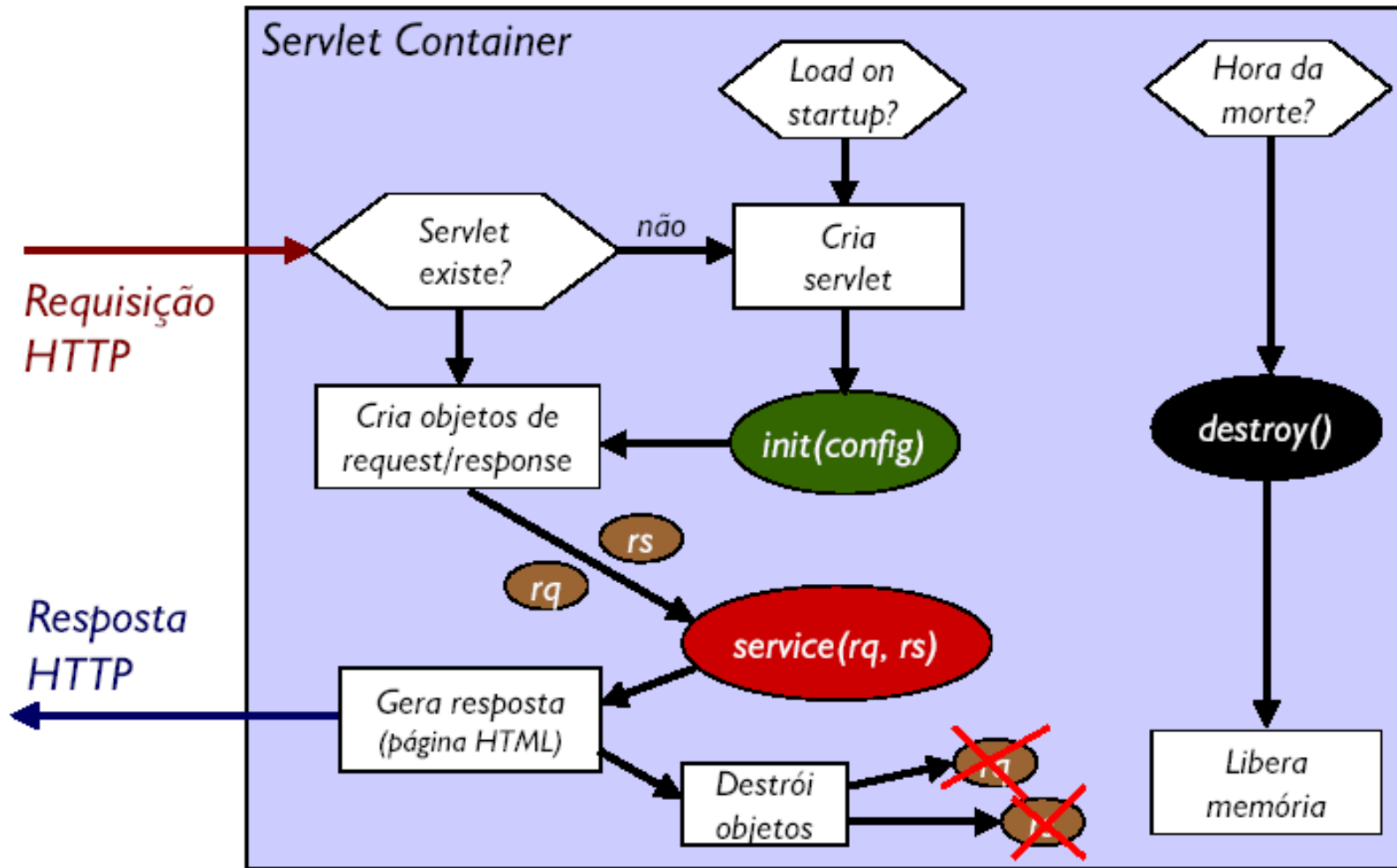


## Servlet = Multi-Thread

- Como já mencionamos, servlets são multi-thread;
- A mesma instância do servlet processa diversas requisições.



## Ciclo de vida de um Servlet



## Ciclo de vida de um Servlet

- O ciclo de vida de um servlet é controlado pelo container;
- Quando o servidor recebe uma requisição, ela é repassada para o container que a delega a um servlet. O container:
  - 1 – Carrega a classe na memória;*
  - 2 – Cria uma instância da classe do servlet;*
  - 3 – Inicializa a instância chamando o método `init()`*
- Depois que o servlet foi inicializado, cada requisição é executado em um método `service()`
  - O container cria um objeto de requisição (*`ServletRequest`*) e de resposta (*`ServletResponse`*) e depois chama *`service()`* passando os objetos como parâmetros;
  - Quando a resposta é enviada, os objetos são destruídos.
- Quando o container decidir remover o servlet da memória, ele o finaliza chamando o método *`destroy()`*

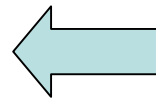


## Ciclo de vida de um Servlet – init()

- O método init() é disparado quando o servlet é carregado;
- Você deve sobrescrevê-lo quando necessitar inicializar o servlet de alguma forma;
- A sintaxe do método é:

```
public void init(ServletConfig config)  
    throws ServletException
```

```
public void init()  
    throws ServletException
```



**Sobrescreva esse  
método caso  
necessite inicializar algo**





## Ciclo de vida de um Servlet – init()

- Caso você tenha algum problema na inicialização do Servlet, você deverá disparar a exceção `UnavailableException` (subclasse de `ServletException`).

```
...  
public void init()  
    throws ServletException  
{  
    String dir = getInitParameter("diretorio");  
    // Problemas ao ler configuracao?  
    if ( dir == null )  
        throw new UnavailableException("Erro na  
        configuracao!")  
}
```



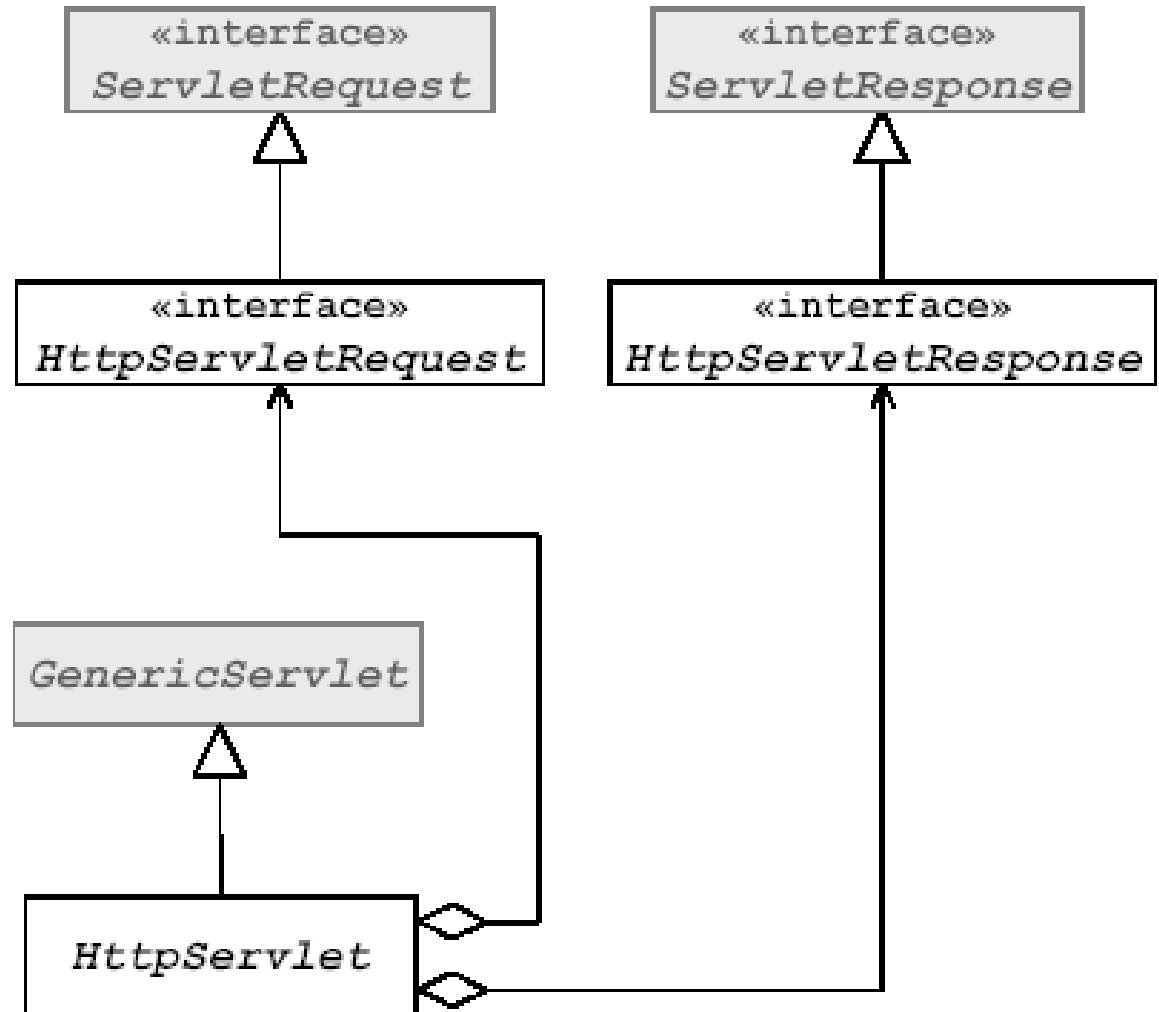
## Ciclo de vida de um Servlet – service()

- O método service() é o método que irá disparar o doGet ou doPost.
- Ele é um método abstrato na sua declaração em GenericServlet. HttpServlet implementa esse método repassando a chamada para doGet ou doPost.



## Diagrama de classes

- Servlets não necessariamente estão ligados a aplicações web. Essa é a razão de existir a classe `GenericServlet` e a implementação `HttpServlet`



# Ciclo de vida de um Servlet – destroy()

- O método destroy() é chamado pelo container antes do servlet ser destruído;
- O momento em quem o servlet será desativado é decidido pelo container;
- É a última oportunidade para “limpar a casa”.

```
public void destroy()  
{  
    ...  
}
```



# Processamento em Background

- Um servlet pode fazer mais que simplesmente servir clientes;
- Ele pode trabalhar entre os acessos;
- Essa é uma habilidade interessante para processos muito demorados;
- Um cliente pode, por exemplo, disparar um processo e eventualmente ficar consultando o resultado do processamento.



## ...Processamento em Background

- Para exemplificar, vamos criar um servlet que calcule números primos;
- Para fazer com que o processamento demore bastante, vamos iniciar a busca por números primos a partir de 1.000.000 (um milhão);
- Vamos utilizar um thread para ficar realizando o processamento em background.



# ServletBackground

```
import java.io.*;                import java.util.*;
import javax.servlet.*;          import javax.servlet.http.*;

public class ServletBackground extends HttpServlet
implements Runnable
{

private long lastprime = 0; // Ultimo primo encontrado
Date lastprimeModified = new Date();
Thread searcher; // Thread para busca

public void init() throws ServletException {
    searcher = new Thread( this );
    searcher.setPriority( Thread.MIN_PRIORITY );
    searcher.start();
}

// ... Continua →
```



// → ... Continuação

```
public void run() {  
    long candidate = 1000001L; // um milhão e um  
    // Loop para buscar eternamente primos...  
    while (!searcher.interrupted()) {  
        if (isPrime(candidate)) {  
            lastprime = candidate;  
            lastprimeModified = new Date();  
        }  
        candidate += 2; // Nos. pares não são primos  
        // Dormir por 200ms...  
        try {  
            Thread.sleep(200);  
        }  
        catch (InterruptedException ignored) { }  
    }  
}
```





// → ... Continuação

```
private static boolean isPrime(long candidate) {
    long sqrt = (long) Math.sqrt(candidate);
    for (long i = 3; i <= sqrt; i += 2) {
        if (candidate % i == 0) return false;
    }
    return true;
}

public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    if (lastprime == 0) {
        out.println("Procurando o 1o. primo...");
    }
    else {out.println("O último primo encontrado foi " +
        lastprime);
        out.println(" em " + lastprimeModified);
    }
}
```



// → ... Continuação

```
public void destroy() {  
    // Sinalizar para o thread parar...  
    searcher.interrupt();  
}  
}
```



KEEP  
CALM  
AND  
BORA ESTUDAR

@Gustavo  
Molina

