



Bài 3. Cơ bản về Servlet



Nội dung

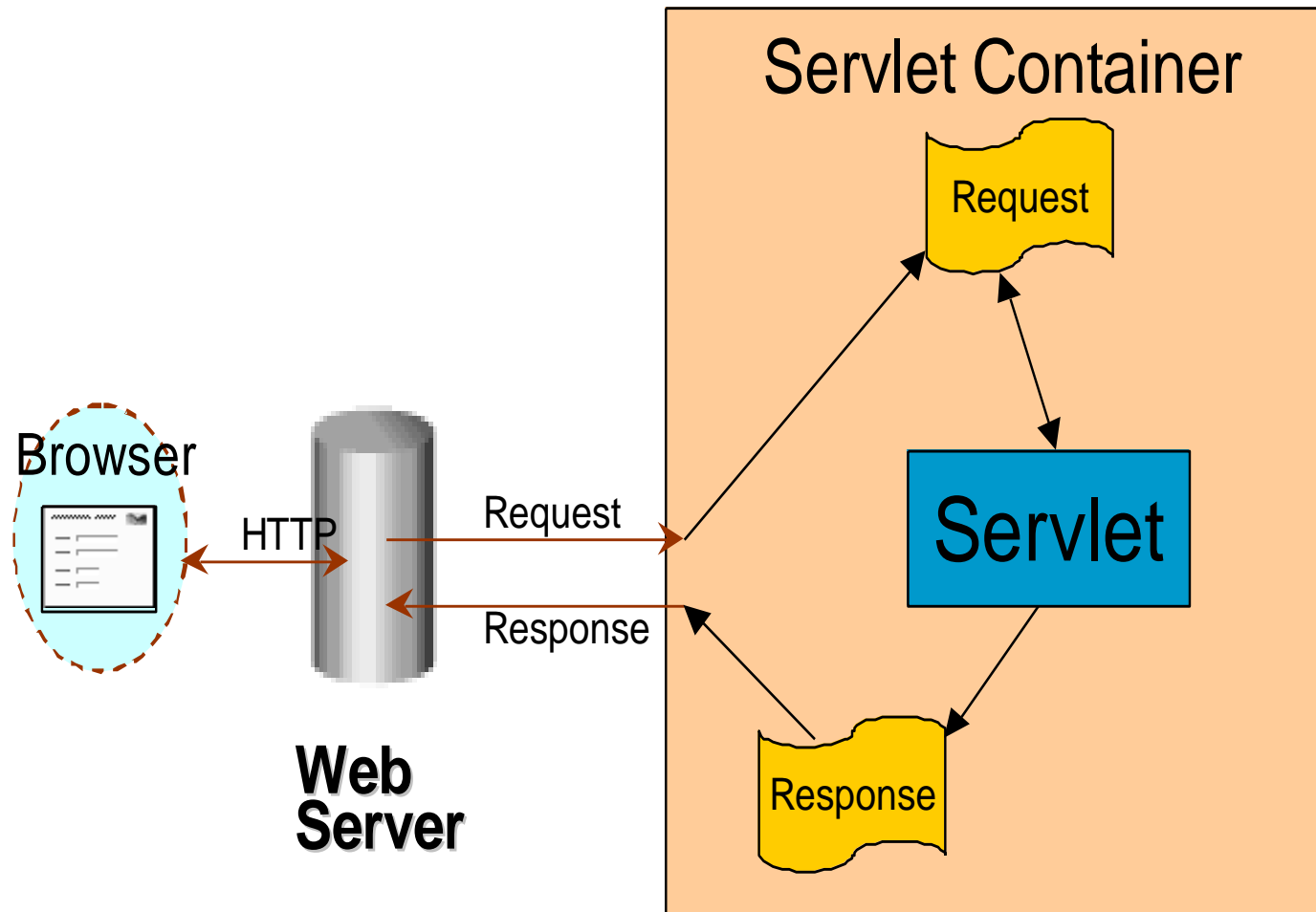
- 1. Servlet là gì
- 2. Các phương thức HTTP
- 3. Vòng đời của Servlet
- 4. Servlet scope objects
- 5. Servlet request
- 6. Servlet response: Status, Header, Body
- 7. Xử lý lỗi (Error)



1. Servlet là gì?

- Các đối tượng Java™, mở rộng chức năng của 1 HTTP server.
- Được ánh xạ (mapped) với 1 URL và được quản lý bởi **container** tương ứng
- Chạy được trên tất cả các **web servers** và các **app servers** chuẩn

Mô hình Servlet Request & response





Nhiệm vụ của Servlet?

- Nhận client request (hầu hết ở dạng HTTP request)
- Trích xuất 1 số thông tin từ request
- Xử lý nghiệp vụ (truy cập DB, tính toán...) hoặc sinh động nội dung
- Tạo và gửi trả response cho client (hầu hết ở dạng HTTP response) hoặc forward request cho servlet khác/cho trang JSP



Requests và Responses

■ Request là gì?

- Thông tin được gửi từ client tới 1 server
 - Ai tạo ra request
 - Dữ liệu gì được user nhập vào và gửi đi
 - Các HTTP headers

■ Response là gì?

- Thông tin được gửi đến client từ 1 server
 - Dữ liệu Text (html, thuần text) hoặc dữ liệu binary (image)
 - HTTP headers, cookies, ...



2. Các phương thức HTTP



HTTP

- HTTP request bao gồm
 - header
 - Phương thức
 - Get: Thông tin nhập vào trong form được truyền như 1 phần của URL
 - Post: Thông tin nhập vào trong form được truyền trong nội dung thông điệp (**message body**)
 - Put: Đặt một thông tin đính kèm vào request
 - Delete: Xóa một tài nguyên nào đó
 - ...
 - Dữ liệu trong request (**request data**)



Phương thức GET và POST

- Các phương thức thông dụng nhất
 - GET & POST
- GET requests:
 - Thông tin người dùng nhập vào **đính kèm** trong URL dưới dạng 1 query string
 - Chỉ gửi được lượng dữ liệu giới hạn
 - `.../servlet/ViewCourse?FirstName=Sang&LastName=Shin`
- POST requests:
 - Thông tin người dùng nhập vào được gửi dưới dạng dữ liệu (không đính kèm vào URL)
 - Gửi được lượng dữ liệu bất kỳ



Nên sử dụng GET hay POST

- GET: getting
 - nhận dữ liệu từ server để hiển thị
 - không thay đổi điều gì phía server
 - các vấn đề khác: không có tính an ninh, bookmark...
- POST: update
 - thay đổi điều gì đó trên server như thêm bản ghi mới...
 - các vấn đề khác: truyền dữ liệu đảm bảo an ninh, không bookmark...

Phương thức “idempotent” và “non idempotent”

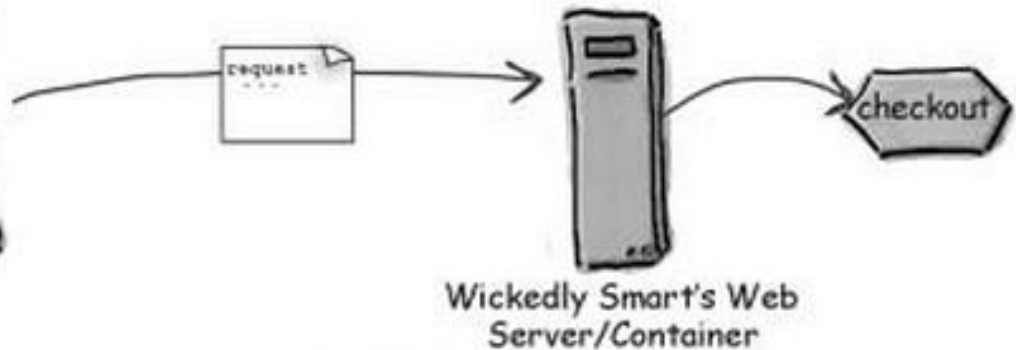
①

Diane hits the **CHECKOUT** button. (She submitted her bank account info earlier.)



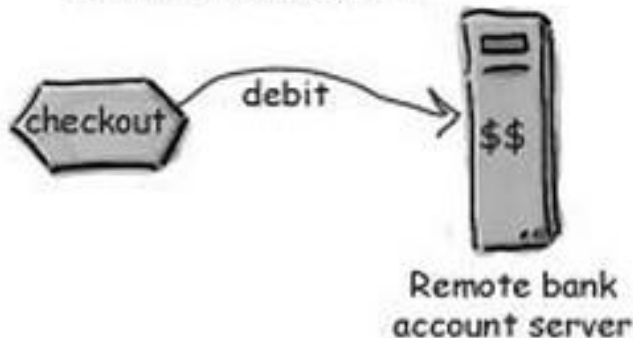
Browser sends an HTTP request to the server with the book purchase info and Diane's customer ID number.

The Container sends the request to the Checkout servlet for processing.



②

Servlet electronically debits Diane's bank account.



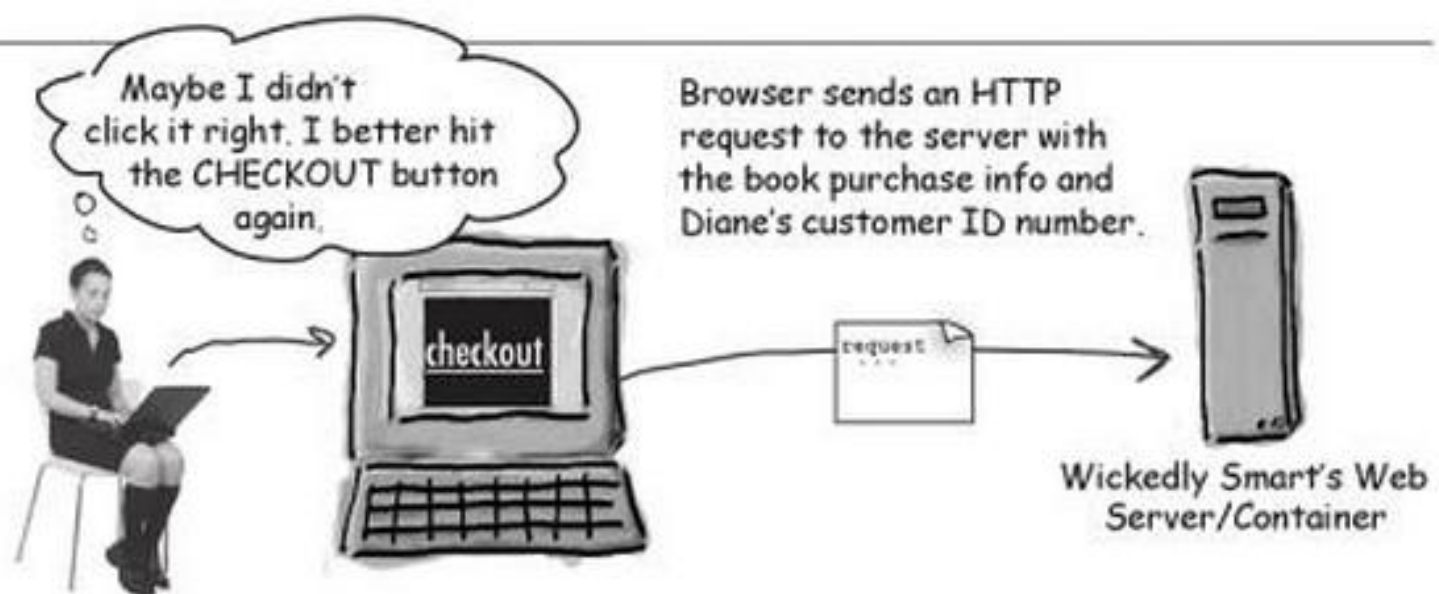
③

Servlet updates the database (takes the book out of inventory, creates a new shipping order, etc.).



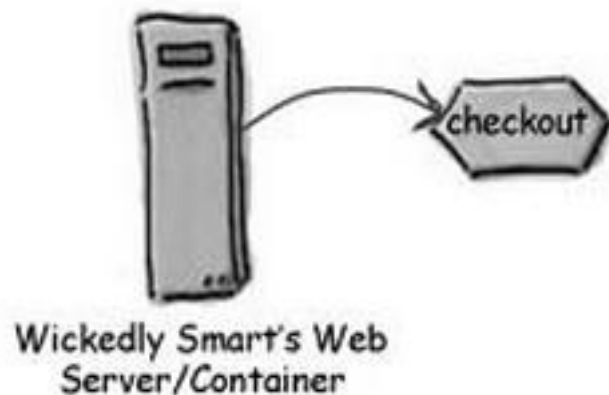
④

Servlet does NOT send an obvious response, so Diane still sees the same shopping cart page and thinks...



⑤

The Container sends the request to the Checkout servlet for processing.

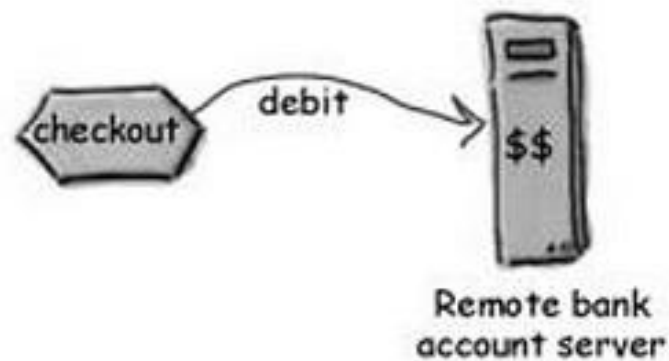


⑥

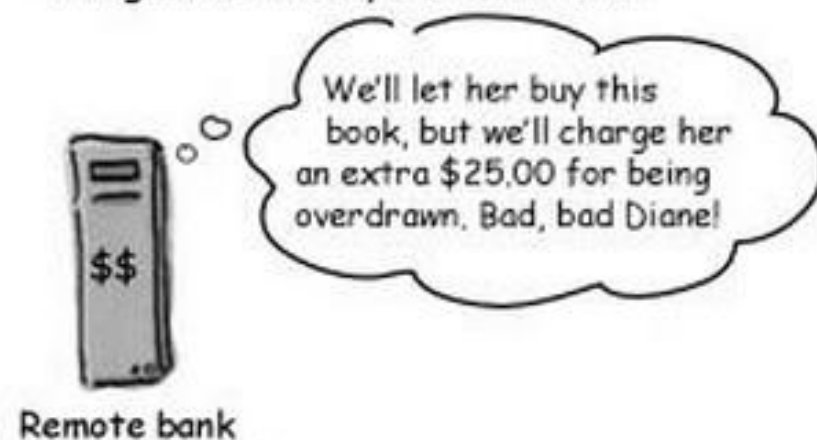
The servlet does not have a problem with Diane buying the same book she bought before.



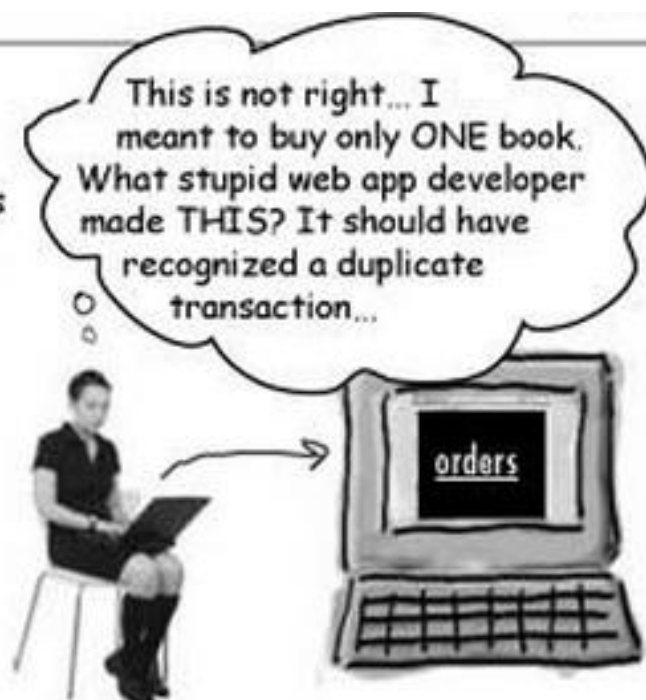
- 7 Servlet electronically debits Diane's bank account for the second time.



- 8 Diane's bank accepts the debit, but charges her a hefty overdraft fee.



- 9 Eventually Diane navigates to the Check Order Status page and sees that she has TWO orders for the knitting book...



- 10 Hello bank? This wickedly stupid web programmer made a mistake...





Phương thức “idempotent” và “non idempotent”

- Phương thức idempotent (cố định)
 - Nếu sự thực thi của $n > 0$ request có tác động giống như sự thực thi của 1 request riêng
 - Các phương thức: GET, PUT, DELETE, HEAD
- Phương thức non idempotent
 - Phương thức: POST

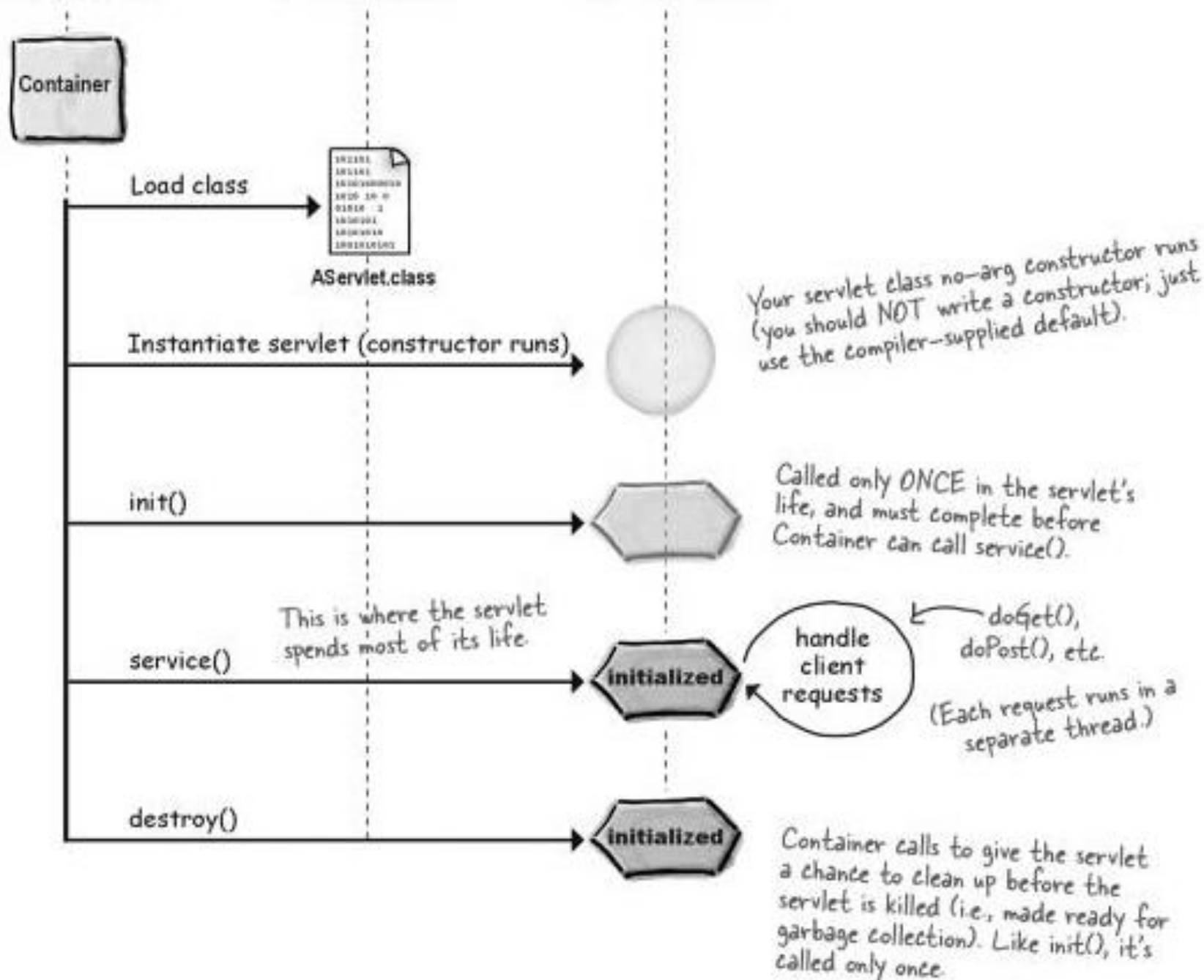


Phương thức GET và POST

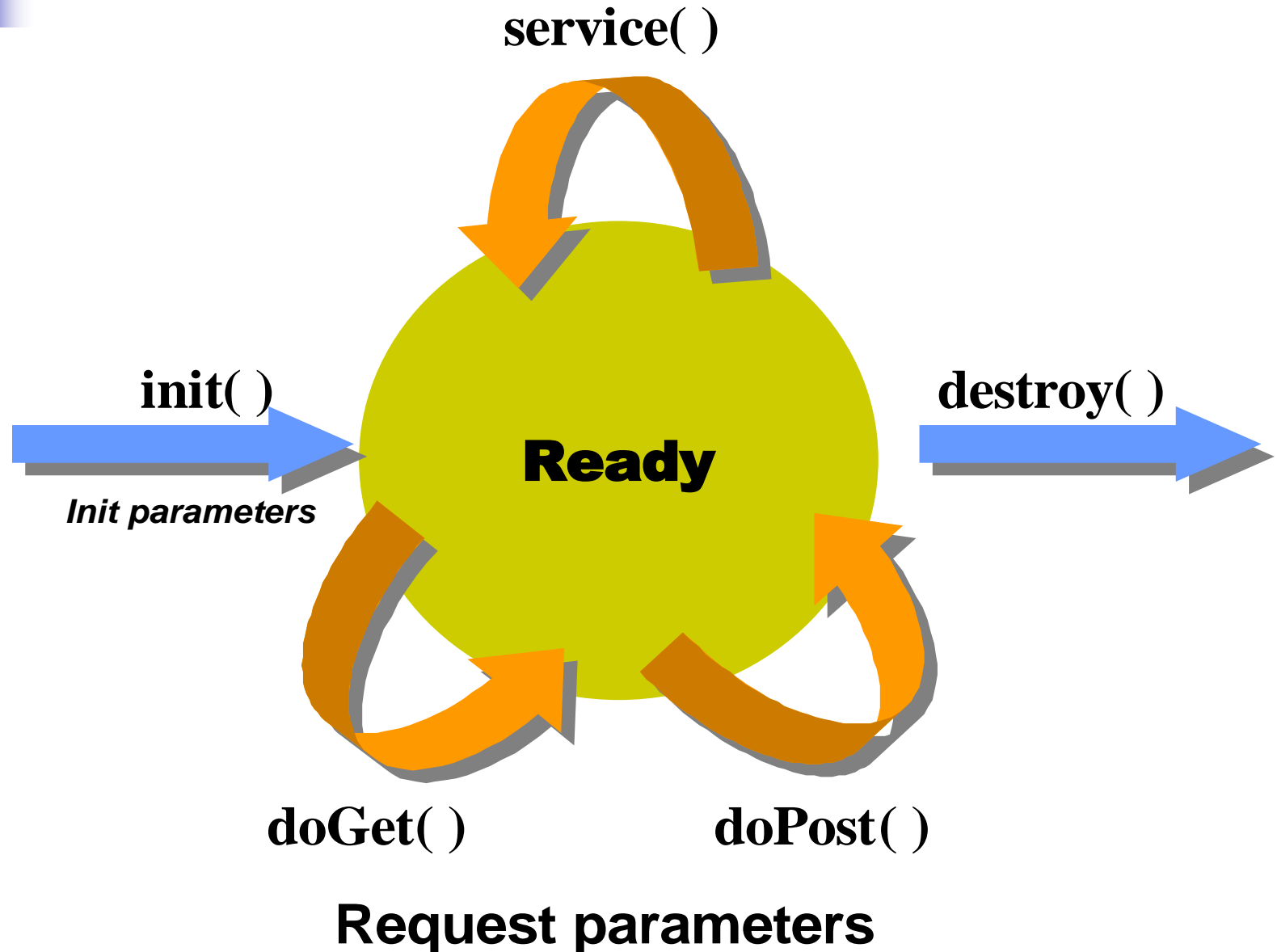
- Thiết lập cách truyền GET
 - `<form method =“GET” action=“SelectBeer.do”>`
 - `click here`
- Thiết lập cách truyền POST
 - `<form method =“POST” action=“SelectBeer.do”>`
- Phương thức mặc định là GET



3. Vòng đời của Servlet

Web Container**Servlet Class****Servlet Object**

Các phương thức trong vòng đời Servlet





Các phương thức trong vòng đời Servlet

- Được gọi bởi container
 - Container điều khiển vòng đời của 1 servlet
- Định nghĩa trong:
 - Lớp `javax.servlet.GenericServlet`
 - `init()`
 - `destroy()`
 - `service()` - là phương thức **abstract**
 - Lớp `javax.servlet.http.HttpServlet`
 - `doGet()`, `doPost()`, `doXxx()`
 - `service()` - implementation



Các phương thức trong vòng đời Servlet

■ init()

- Được gọi **MỘT** lần khi servlet được tạo thể hiện lần đầu tiên
- Thực hiện các **khởi tạo** trong phương thức này
 - Ví dụ: tạo 1 kết nối CSDL

■ destroy()

- Được gọi trước khi hủy 1 servlet instance
- Thực hiện thao tác dọn dẹp
 - Ví dụ: đóng kết nối CSDL đã mở



Ví dụ: init() trong CatalogServlet.java

```
public class CatalogServlet extends HttpServlet {  
    private BookDB bookDB;  
  
    // Perform any one-time operation for the servlet,  
    // like getting database connection object.  
  
    // Note: In this example, database connection object is assumed  
    // to be created via other means (via life cycle event mechanism)  
    // and saved in ServletContext object. This is to share a same  
    // database connection object among multiple servlets.  
    public void init() throws ServletException {  
        bookDB = (BookDB) getServletContext().  
            getAttribute("bookDB");  
        if (bookDB == null) throw new  
            UnavailableException("Couldn't get database.");  
    }  
    ...  
}
```



Ví dụ: init() đọc tham số cấu hình

```
public void init(ServletConfig config) throws
    ServletException {
    super.init(config);
    String driver = getInitParameter("driver");
    String fURL = getInitParameter("url");
    try {
        openDBConnection(driver, fURL);
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

Thiết lập các tham số trong web.xml

```
<web-app>
  <servlet>
    <servlet-name>chart</servlet-name>
    <servlet-class>ChartServlet</servlet-class>
    <init-param>
      <param-name>driver</param-name>
      <param-value>
        COM.cloudscape.core.RmiJdbcDriver
      </param-value>
    </init-param>

    <init-param>
      <param-name>url</param-name>
      <param-value>
        jdbc:cloudscape:rmi:CloudscapeDB
      </param-value>
    </init-param>
  </servlet>
</web-app>
```



Ví dụ: destroy()

```
public class CatalogServlet extends HttpServlet {
    private BookDB bookDB;

    public void init() throws ServletException {
        bookDB = (BookDB)getServletContext().
            getAttribute("bookDB");
        if (bookDB == null) throw new
            UnavailableException("Couldn't get database.");
    }
    public void destroy() {
        bookDB = null;
    }
    ...
}
```




Các phương thức trong vòng đời Servlet

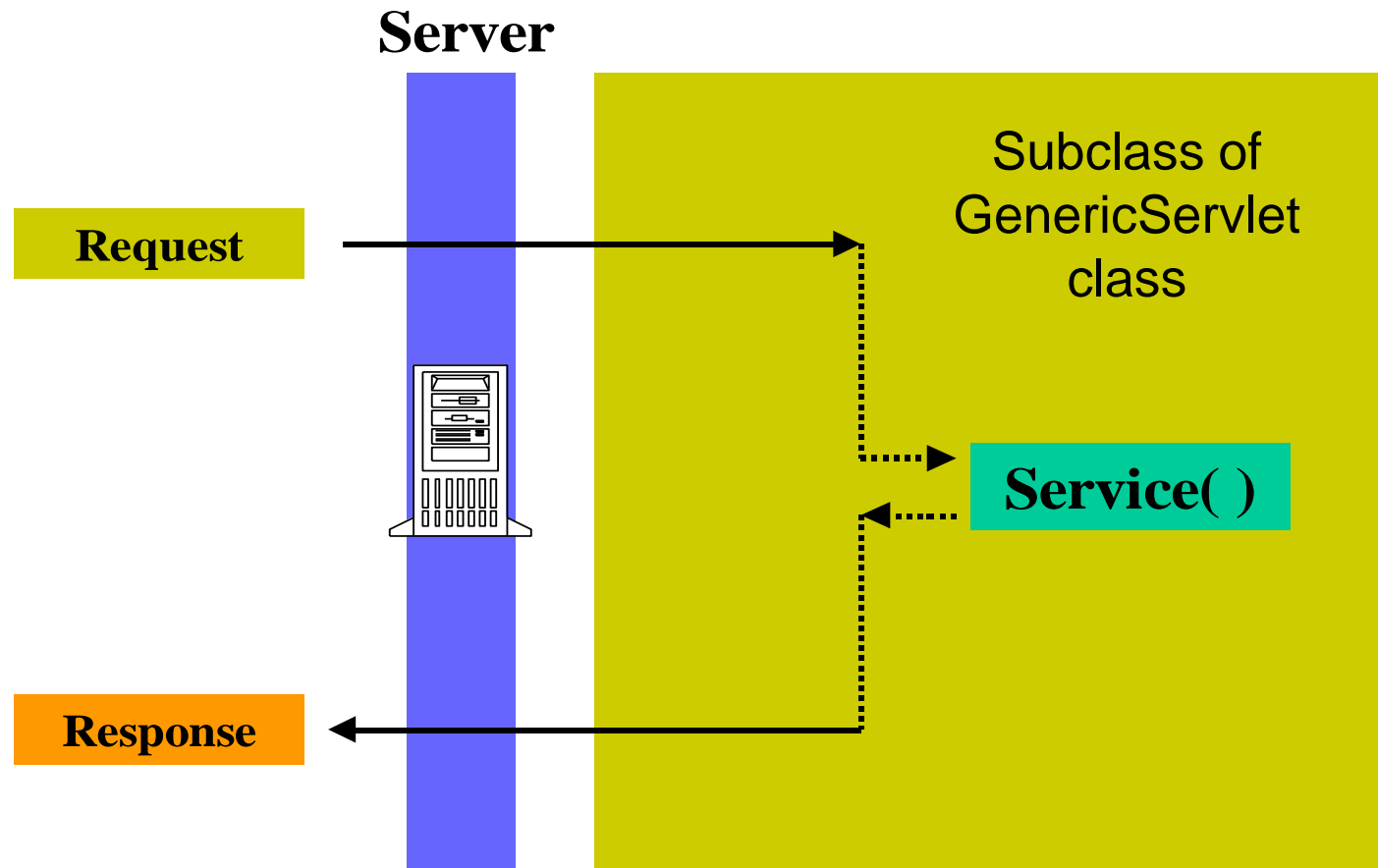
- service() trong `javax.servlet.GenericServlet`
 - Phương thức **Abstract**
- service() trong lớp `javax.servlet.http.HttpServlet`
 - Phương thức cụ thể (đã cài đặt)
 - gọi tới (**dispatch**) `doGet()`, `doPost()`
 - **KHÔNG** override phương thức này!
- `doGet()`, `doPost()`, `doXxx()` trong `javax.servlet.http.HttpServlet`
 - Xử lý các HTTP GET, POST requests
 - Lập trình viên override những phương thức này trong servlet của mình để có xử lý phù hợp



service() & doGet()/doPost()

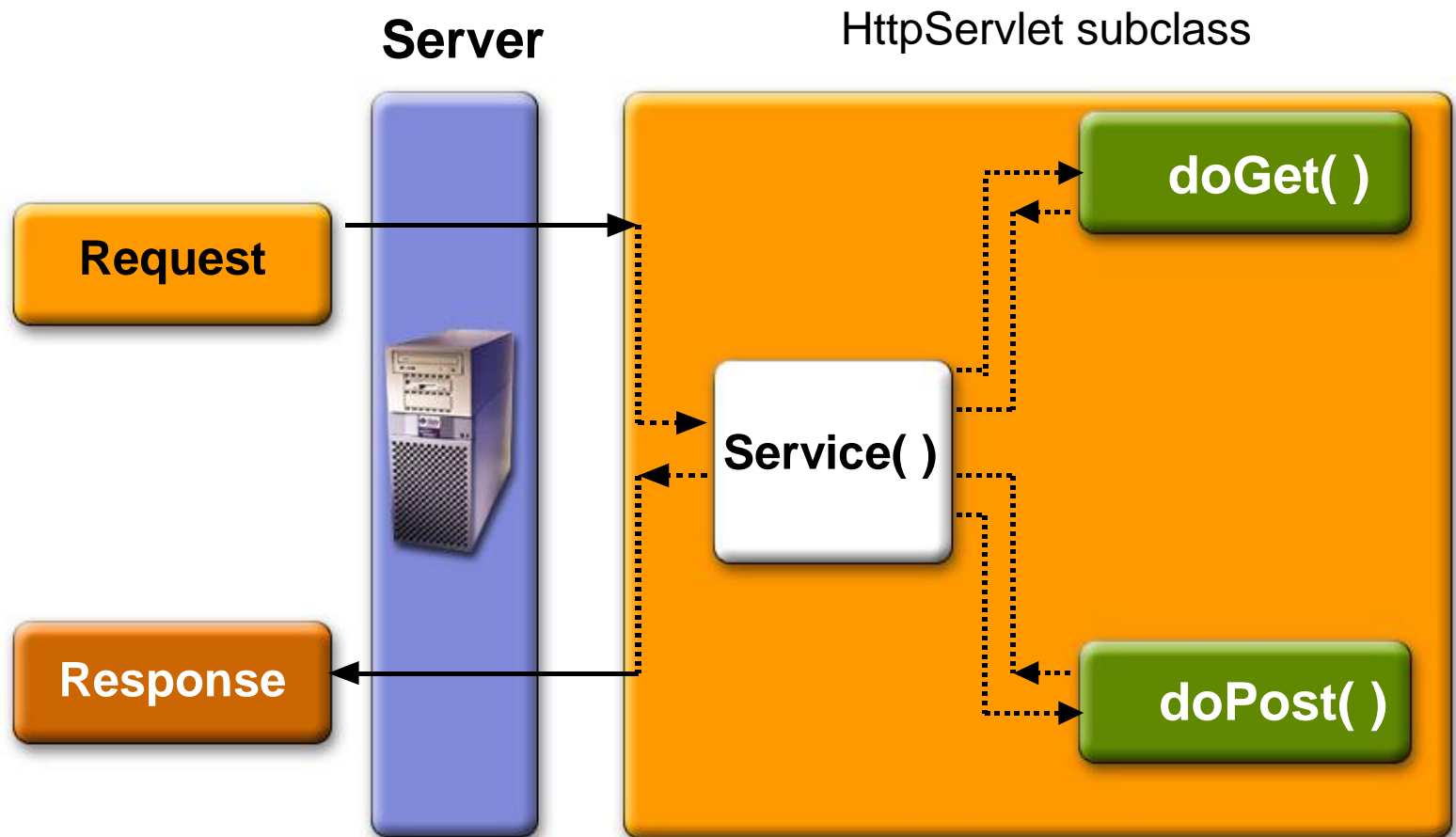
- Phương thức **service()** nhận các requests và responses tổng quát:
 - `service(ServletRequest request, ServletResponse response)`
- **doGet()** và **doPost()** nhận các HTTP requests và responses:
 - `doGet(HttpServletRequest request, HttpServletResponse response)`
 - `doPost(HttpServletRequest request, HttpServletResponse response)`

Phương thức Service()



Key: Implemented by subclass

Phương thức doGet() và doPost()



Key:  Implemented by **subclass**



Những việc cần làm trong doGet() & doPost()

- Trích xuất các thông tin gửi từ client (HTTP parameter) từ HTTP request
- Thiết lập/truy cập các thuộc tính của các **Scope objects**
- Thực hiện các xử lý nghiệp vụ (**business logic**) hoặc truy cập CSDL
- Tùy chọn forward request tới các Web components khác (Servlet hoặc JSP)
- Sinh HTTP response và trả về cho client



Ví dụ 1 phương thức doGet() đơn giản

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

Public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        // Just send back a simple HTTP response
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<title>First Servlet</title>");
        out.println("<big>Hello J2EE Programmers! </big>");
    }
}
```



Các bước tạo một HTTP Response

- Thiết lập loại nội dung trả về (**content type**)
- Lấy 1 đối tượng **output stream** từ response đang xét
- Viết nội dung cần trả về cho client vào output stream



Ví dụ một Response đơn giản

```
Public class HelloServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request,  
        HttpServletResponse response)  
        throws ServletException, IOException {  
  
        // Fill response headers  
        response.setContentType("text/html");  
        // Get an output stream object from the response  
        PrintWriter out = response.getWriter();  
        // Write body content to output stream  
        out.println("<title>First Servlet</title>");  
        out.println("<big>Hello J2EE Programmers! </big>");  
    }  
}
```




Các loại nội dung - ContextType

- Các MIME type phổ biến
 - text/html
 - application/pdf
 - application/java
 - application/jar
 - application/octet-stream
 - application/x-zip
 - image/jpeg
 - vide/quicktime
 - ...



4. Scope Objects



Scope Objects

- Thông tin được **chia sẻ** giữa các web components
- Được gọi thông qua các thuộc tính (**attributes**)
- Các thuộc tính được tham chiếu trong các Scope objects thông qua phương thức
 - `getAttribute()` & `setAttribute()`
- 4 loại Scope objects được định nghĩa
 - **Web context, session, request, page**



4 loại Scope Objects: giới hạn truy cập

- Web context (ServletContext)
 - Truy cập từ các Web components trong 1 Web context
- Session
 - Truy cập từ các Web components xử lý request trong 1 session
- Request
 - Truy cập từ các Web components xử lý request đó
- Page
 - Truy cập từ trang JSP tạo ra object đó



4 loại Scope Objects: các Class tương ứng

- Web context
 - `javax.servlet.ServletContext`
- Session
 - `javax.servlet.http.HttpSession`
- Request
 - subtype of `javax.servlet.ServletRequest`:
`javax.servlet.http.HttpServletRequest`
- Page
 - `javax.servlet.jsp.PageContext`

4.1. Web Context (ServletContext)





ServletContext dùng để làm gì?

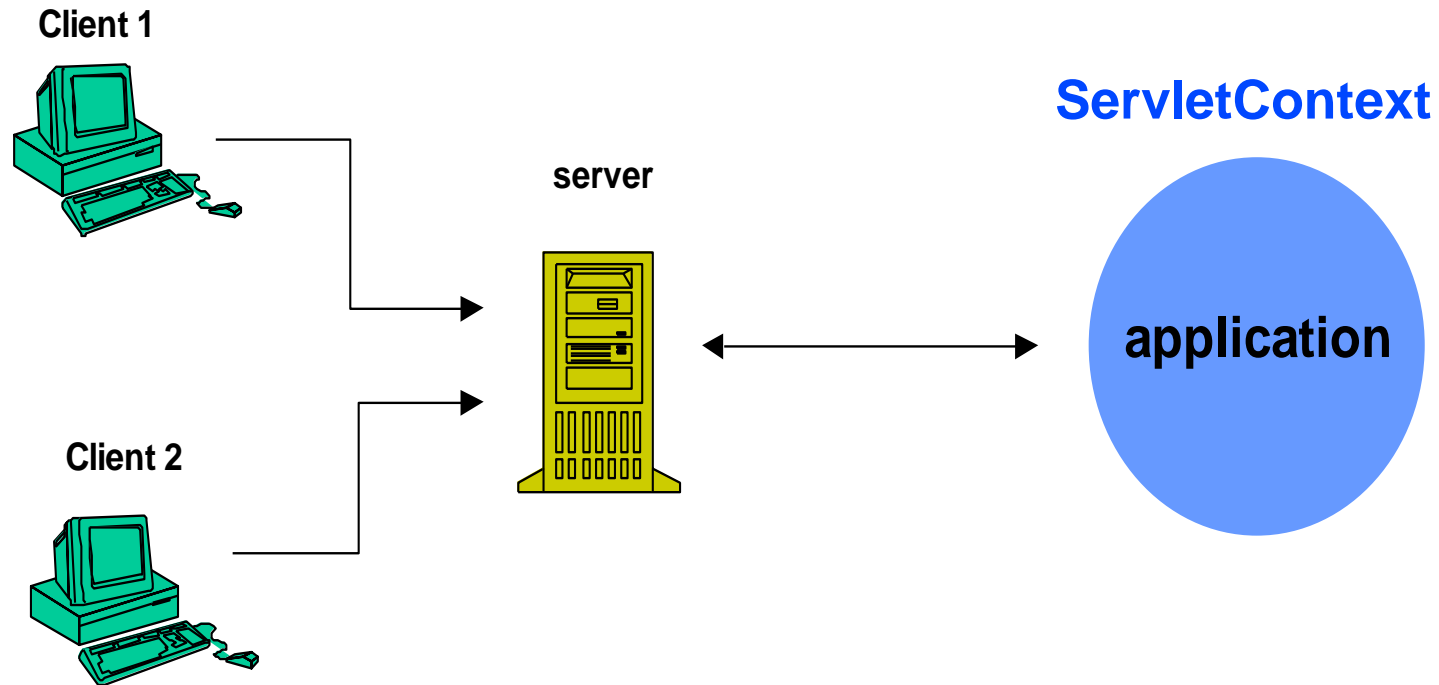
- Được sử dụng bởi Servlet để:
 - Thiết lập các thuộc tính có tầm vực context (trong toàn ứng dụng)
 - Lấy ra đối tượng request dispatcher
 - Forward hoặc include các web component khác
 - Truy cập các tham số khởi tạo tầm vực Web context thiết lập trong file web.xml
 - Truy cập các tài nguyên Web kết hợp với Web context
 - Ghi Log
 - Truy cập các thông tin khác



Tầm vực (**Scope**) của ServletContext

- Có tầm vực context (**Context-wide scope**)
 - Được chia sẻ bởi tất cả các servlets và các trang JSP trong cùng 1 "web application"
 - Vì thế còn gọi là "web application scope"
 - Một "web application" là 1 tập các servlets và các content khác, chung 1 phần URL, và có thể cài đặt qua 1 file *.war
 - Có duy nhất 1 đối tượng ServletContext cho mỗi "web application" trên mỗi Java Virtual Machine

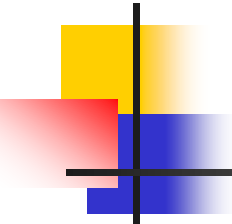
ServletContext: Web Application Scope





Truy cập tới đối tượng ServletContext như thế nào?

- Trong code **servlet** hoặc code **servlet filter**, gọi hàm `getServletContext()`
- Trong đối tượng `ServletConfig` cũng chứa đối tượng `ServletContext`
 - Web server cung cấp `ServletConfig` cho mỗi `servlet` khi khởi tạo nó: trong giao diện `Servlet`
`init (ServletConfig servletConfig)`



Ví dụ: Lấy giá trị của 1 thuộc tính từ ServletContext

```
public class CatalogServlet extends HttpServlet {  
    private BookDB bookDB;  
    public void init() throws ServletException {  
        // Get context-wide attribute value from  
        // ServletContext object  
        bookDB = (BookDB) getServletContext().  
            getAttribute("bookDB");  
        if (bookDB == null) throw new  
            UnavailableException("Couldn't get database.");  
    }  
}
```



GreetingServlet

```
String username = request.getParameter("username");
if ((username != null) && (username.length() > 0)) {
    RequestDispatcher dispatcher =
        getServletContext()
            .getRequestDispatcher("/response");
    if (dispatcher != null) {
        dispatcher.include(request, response);
    }
}
```

4.2. Session (HttpSession)



”



Tại sao cần HttpSession?

- Cần 1 cơ chế để lưu trữ trạng thái client theo thời gian sau 1 loạt các request từ cùng 1 người dùng (cùng 1 trình duyệt)
 - Ví dụ: Giỏ hàng (**Online shopping cart**)
- HTTP là giao thức phi trạng thái (**stateless**)
- HttpSession lưu trữ (**maintain**) trạng thái client
 - Sử dụng bởi các Servlets để set và get giá trị các thuộc tính có tầm vực session



Lấy ra đối tượng HttpSession?

- Qua phương thức getSession() của 1 đối tượng Request (HttpServletRequest)



Ví dụ: HttpSession

```
public class CashierServlet extends HttpServlet {  
    public void doGet (HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException {  
  
        // Get the user's session and shopping cart  
        HttpSession session = request.getSession();  
        ShoppingCart cart =  
            (ShoppingCart) session.getAttribute("cart");  
  
        ...  
        // Determine the total price of the user's books  
        double total = cart.getTotal();  
    }  
}
```


5. Servlet request (HttpServletRequest)



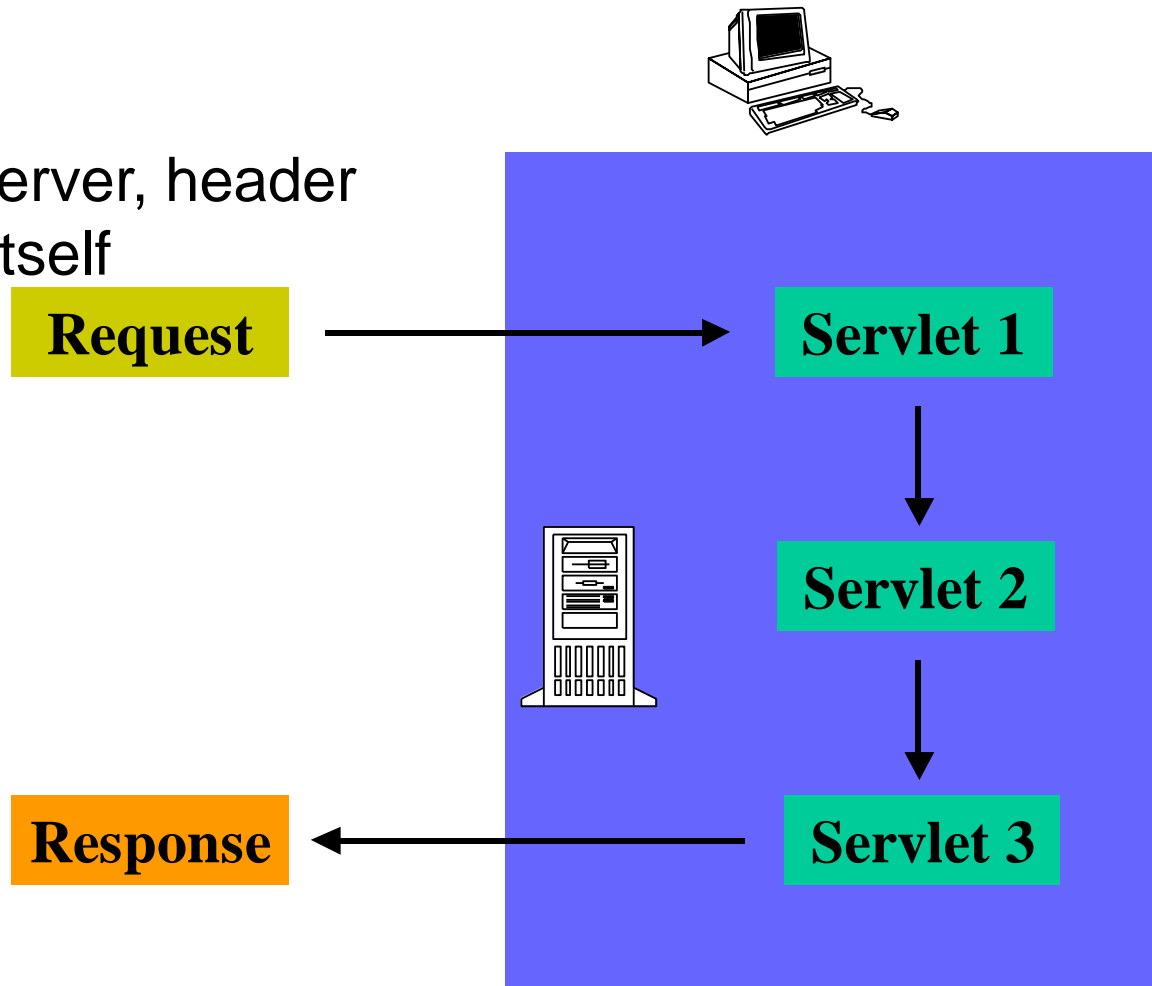


Servlet Request là gì?

- Chứa dữ liệu gửi từ client đến servlet
- Tất cả các servlet requests đều thực thi giao diện **ServletRequest** định nghĩa các phương thức truy cập tới:
 - Các tham số (parameters) gửi từ clients
 - **Object-valued attributes**
 - Client và server
 - Input stream
 - Thông tin về giao thức (Protocol information)
 - Content type
 - request có được tạo trên 1 kênh truyền secure không (secure channel. Ví dụ: HTTPS)

Requests

data,
client, server, header
servlet itself



Web Server



Lấy các tham số gửi từ Client

- Một request có thể đính kèm số lượng tham số bất kỳ
- Các tham số được gửi từ các forms HTML
 - GET: dưới dạng 1 query string, đính kèm vào URL
 - POST: tham số được mã hóa, không xuất hiện trong URL
- `getParameter("paraName")`
 - Trả về giá trị của tham số `paraName`
 - Trả về null nếu không có tham số tên tương ứng được gọi
 - Làm việc như nhau với GET và POST requests

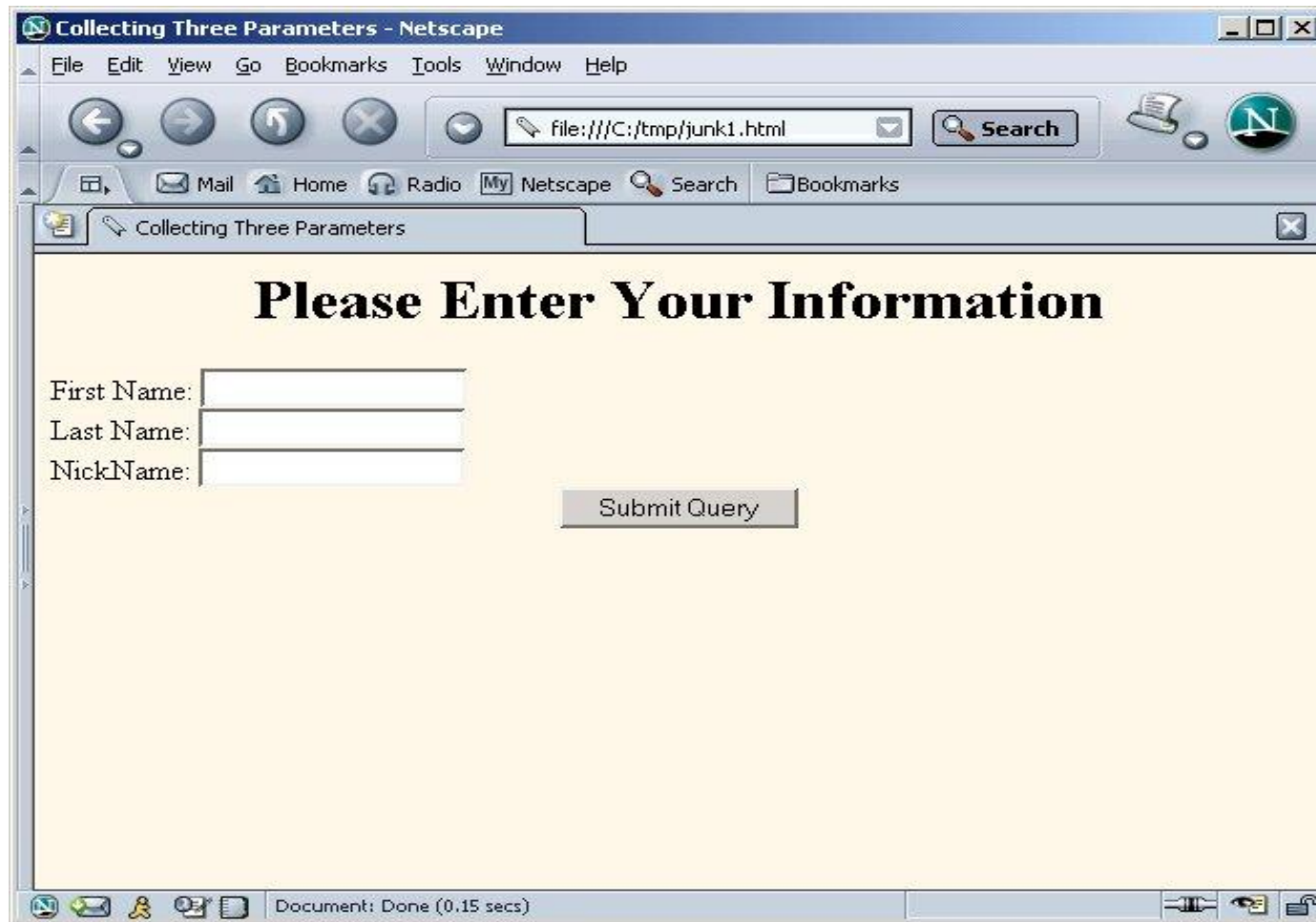
Ví dụ Form gửi theo phương thức GET

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Collecting Three Parameters</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Please Enter Your Information</H1>

<FORM ACTION="/sample/servlet/ThreeParams">
  First Name:  <INPUT TYPE="TEXT" NAME="param1"><BR>
  Last Name:   <INPUT TYPE="TEXT" NAME="param2"><BR>
  Class Name:  <INPUT TYPE="TEXT" NAME="param3"><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT">
  </CENTER>
</FORM>

</BODY>
</HTML>
```

Ví dụ Form gửi theo phương thức GET



The screenshot shows a Netscape browser window titled "Collecting Three Parameters - Netscape". The address bar displays "file:///C:/tmp/junk1.html". The browser's menu bar includes File, Edit, View, Go, Bookmarks, Tools, Window, and Help. The toolbar contains navigation buttons (back, forward, home, stop), a search button, and icons for Mail, Home, Radio, My Netscape, Search, and Bookmarks. The main content area has a yellow background and a title "Please Enter Your Information". Below the title are three input fields labeled "First Name:", "Last Name:", and "NickName:". A "Submit Query" button is positioned to the right of the input fields. The status bar at the bottom shows "Document: Done (0.15 secs)".

Please Enter Your Information

First Name:

Last Name:

NickName:

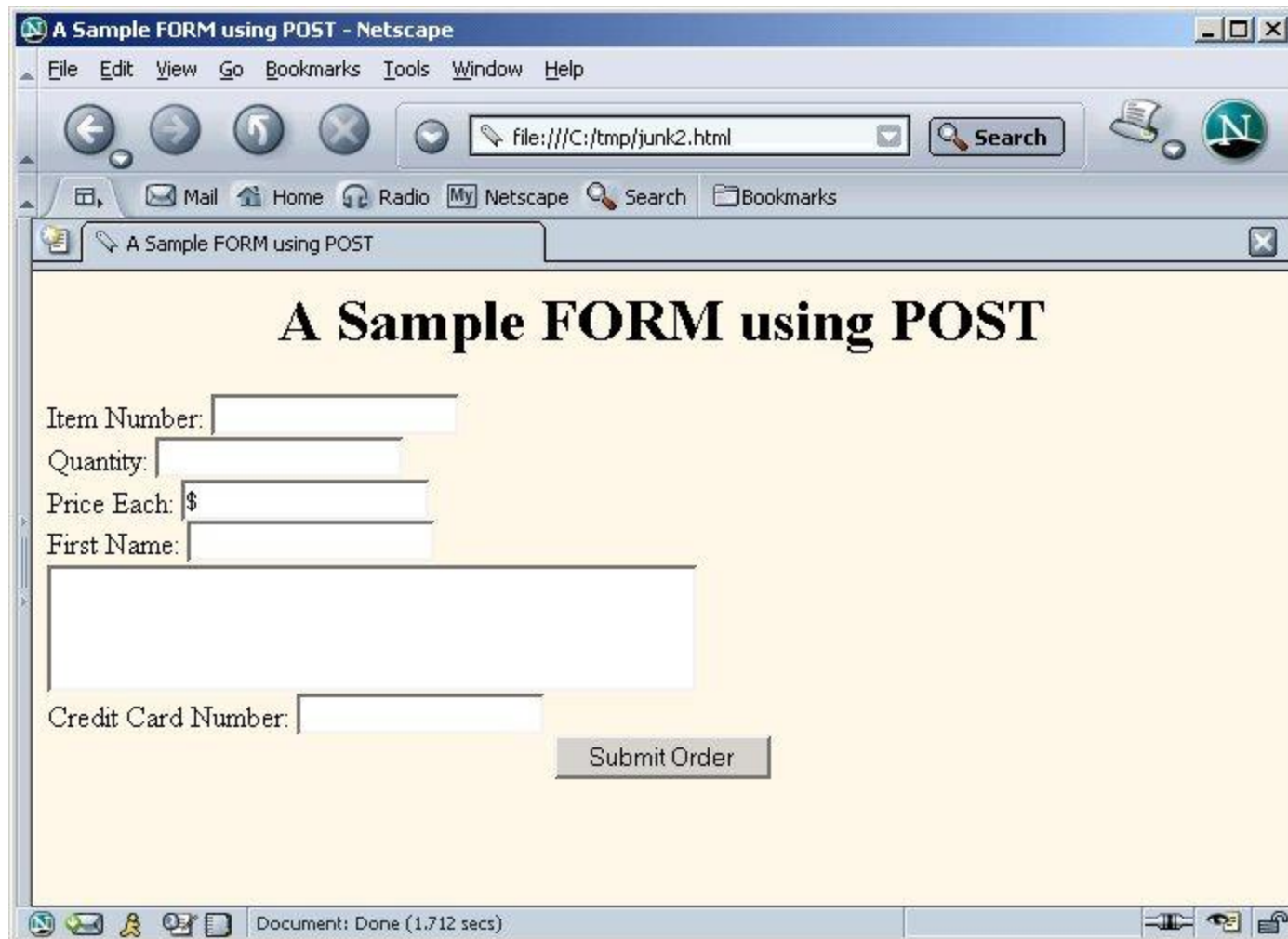
Xử lý trong Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ThreeParams extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Your Information";
        out.println("<HTML>" + "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
            "<UL>\n" + "    <LI><B>First Name in Response</B>:"
            + request.getParameter("param1") + "\n" +
            "    <LI><B>Last Name in Response</B>: "
            + request.getParameter("param2") + "\n" +
            "    <LI><B>NickName in Response</B>: "
            + request.getParameter("param3") + "\n" +
            "</UL>\n" +
            "</BODY></HTML>");
    }
}
```

Ví dụ Form gửi theo phương thức POST

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>A Sample FORM using POST</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">A Sample FORM using POST</H1>
<FORM ACTION="/sample/servlet/ShowParameters" METHOD="POST">
  Item Number: <INPUT TYPE="TEXT" NAME="itemNum"><BR>
  Quantity: <INPUT TYPE="TEXT" NAME="quantity"><BR>
  Price Each: <INPUT TYPE="TEXT" NAME="price" VALUE="$"><BR>
  First Name: <INPUT TYPE="TEXT" NAME="firstName"><BR>
  <TEXTAREA NAME="address" ROWS=3 COLS=40></TEXTAREA><BR>
  Credit Card Number:
  <INPUT TYPE="PASSWORD" NAME="cardNum"><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT" VALUE="Submit Order">
  </CENTER>
</FORM>
</BODY>
</HTML>
```


Ví dụ Form gửi theo phương thức POST



The screenshot shows a Netscape browser window with the title "A Sample FORM using POST - Netscape". The address bar displays "file:///C:/tmp/junk2.html". The browser's menu bar includes File, Edit, View, Go, Bookmarks, Tools, Window, and Help. The toolbar contains navigation buttons (back, forward, home, stop), a search button, and a Netscape logo. The main content area has a title "A Sample FORM using POST" and a form with the following fields:

- Item Number:
- Quantity:
- Price Each: \$
- First Name:
-
- Credit Card Number:
-

The status bar at the bottom indicates "Document: Done (1.712 secs)".



Xử lý trong Servlet

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

public class ShowParameters extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {

        ...
    }
    public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {
        doGet(request, response);
    }
}
```



Thiết lập thuộc tính tầm vực Request

- Các thuộc tính tầm vực Request có thể được thiết lập theo 2 cách
 - Servlet container có thể tự thiết lập 1 thuộc tính trong 1 request
 - Ví dụ: thuộc tính `javax.servlet.request.X509Certificate` cho HTTPS
 - Servlet cũng có thể thiết lập thuộc tính:
 - `void setAttribute(java.lang.String name, java.lang.Object o)`



Lấy thông tin client

- Servlet có thể lấy thông tin về client từ request
 - `String request.getRemoteAddr()`
 - Lấy ra địa chỉ IP của client
 - `String request.getRemoteHost()`
 - Lấy ra tên host của client



Lấy thông tin Server

- Servlet có thể lấy các thông tin về server:
 - `String request.getServerName()`
 - Ví dụ: "www.sun.com"
 - `int request.getServerPort()`
 - Ví dụ: Port number "8080"



Lấy ra các thông tin khác

- Input stream
 - `ServletInputStream getInputStream()`
 - `java.io.BufferedReader getReader()`
- Protocol
 - `java.lang.String getProtocol()`
- Content type
 - `java.lang.String getContentType()`
- Là secure hay không (là HTTPS hay không)
 - `boolean isSecure()`



HttpServletRequest



HTTP Servlet Request là gì?

- Chứa dữ liệu truyền từ HTTP client tới HTTP servlet
- Được tạo bởi servlet container và được truyền cho servlet như 1 tham số của phương thức doGet() hoặc doPost()
- HttpServletRequest mở rộng interface ServletRequest và cung cấp thêm các phương thức cho phép truy cập
 - HTTP request URL
 - Context, servlet, path, query information
 - Các thông tin về HTTP Request header
 - Thông tin về loại Authentication và User security
 - Cookies
 - Session



HTTP Request URL

- Chứa các phần như sau:
 - `http://[host]:[port]/[request path]?[query string]`



HTTP Request URL: [request path]

- `http://[host]:[port]/[request path]?[query string]`
- [request path] bao gồm
 - Context: /<context of web app>
 - Servlet name: /<component alias>
 - Path information: phần còn lại
- Ví dụ
 - `http://localhost:8080/hello1/greeting`
 - `http://localhost:8080/hello1/greeting.jsp`
 - `http://daydreamer/catalog/lawn/index.html`



HTTP Request URL: [query string]

- `http://[host]:[port]/[request path]?[query string]`
- [query string] bao gồm tập các tham số và giá trị người dùng nhập vào
- 2 cách sinh ra query strings
 - Một query string có thể xuất hiện ngay trong 1 trang web
 - `Add To Cart`
 - `String bookId = request.getParameter("Add");`
 - Một query string sẽ được gắn vào 1 URL khi submit 1 form qua phương thức **GET HTTP**
 - `http://localhost/hello1/greeting?username=Monica+Clinton`
 - `String userName=request.getParameter("username")`



Context, Path, Query, Parameter Methods

- `String getContextPath()`
- `String getQueryString()`
- `String getPathInfo()`
- `String getPathTranslated()`



HTTP Request Headers

- HTTP requests chứa nhiều **request headers** cung cấp các thông tin phụ về request
- Ví dụ HTTP 1.1 Request:

GET /search? keywords= servlets+ jsp HTTP/ 1.1

Accept: image/ gif, image/ jpg, */*

Accept-Encoding: gzip

Connection: Keep- Alive

Cookie: userID= id456578

Host: **www.sun.com**

Referer: http://www.sun.com/codecamp.html

User-Agent: Mozilla/ 4.7 [en] (Win98; U)



HTTP Request Headers

- Accept

- Chỉ ra những loại MIME trình duyệt có thể xử lý

- Accept-Encoding

- Chỉ ra loại mã hóa (Ví dụ gzip hoặc compress) trình duyệt có thể xử lý

- Authorization

- Nhận dạng người dùng cho các trang bảo mật
 - Thay vì HTTP authorization, sử dụng HTML forms để gửi username/password và lưu trữ thông tin trong session object



HTTP Request Headers

■ Connection

- Trong HTTP 1.1, mặc định là kết nối persistent (**persistent connection**)
- Servlets nên thiết lập Content-Length bằng phương thức setContentLength (sử dụng ByteArrayOutputStream chỉ định độ dài của output) để hỗ trợ kết nối persistent.

■ Cookie

- cookies server gửi cho client trước đó.

■ Host

- Chỉ định host từ URL gốc
- Được yêu cầu trong HTTP 1.1.



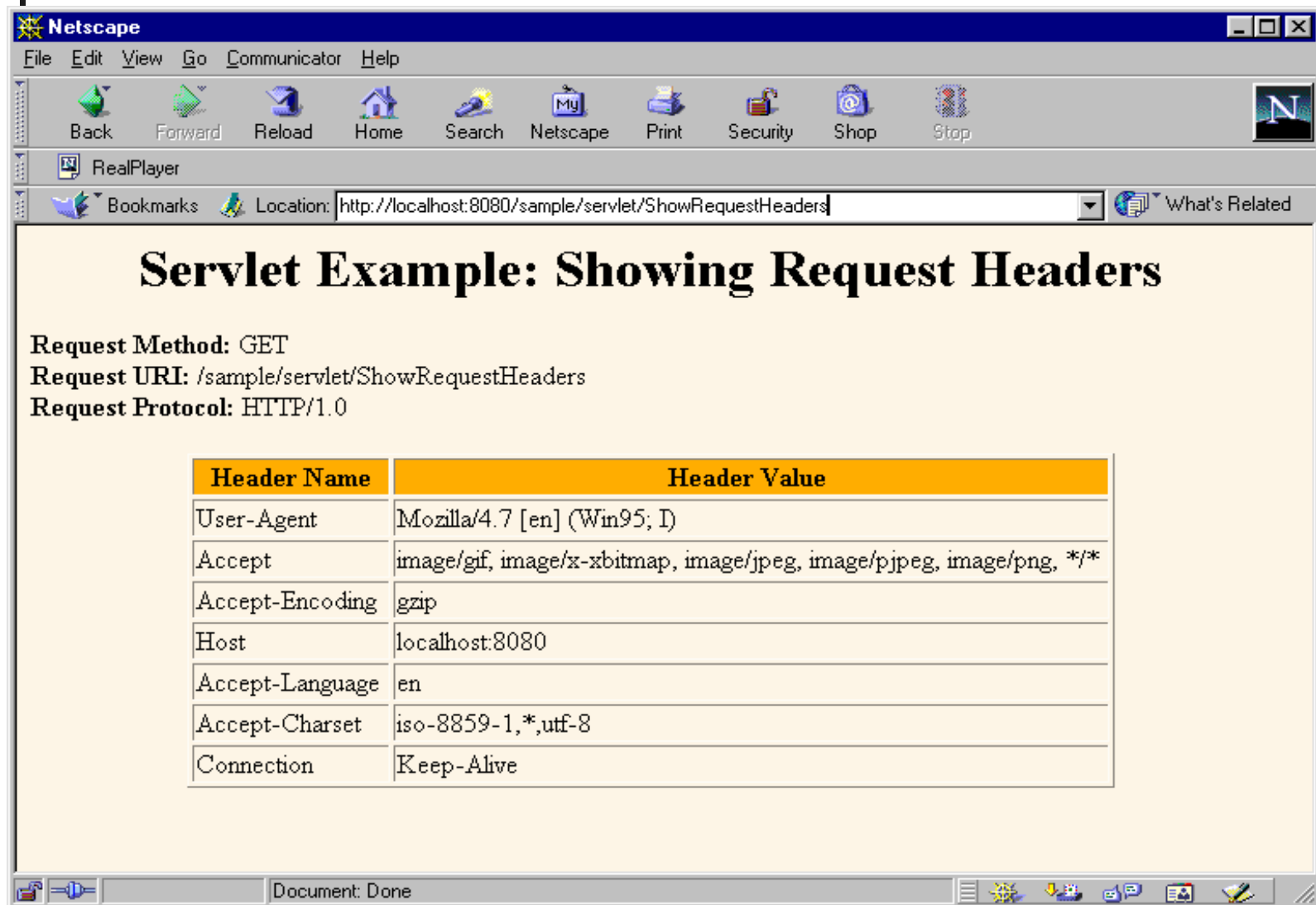
Các phương thức HTTP Header

- `String getHeader(java.lang.String name)`
 - Giá trị String của 1 **request header** cụ thể
- `java.util.Enumeration`
`getHeaders(java.lang.String name)`
 - Giá trị Enum của 1 **request header**
- `java.util.Enumeration getHeaderNames()`
- `int getIntHeader(java.lang.String name)`

Code lay ra cac Request Headers

```
//Shows all the request headers sent on this particular request.
public class ShowRequestHeaders extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Servlet Example: Showing Request Headers";
        out.println("<HTML>" + ...
                    "<B>Request Method: </B>" +
                    request.getMethod() + "<BR>\n" +
                    "<B>Request URI: </B>" +
                    request.getRequestURI() + "<BR>\n" +
                    "<B>Request Protocol: </B>" +
                    request.getProtocol() + "<BR><BR>\n" +
                    ...
                    "<TH>Header Name<TH>Header Value");
        Enumeration headerNames = request.getHeaderNames();
        while(headerNames.hasMoreElements()) {
            String headerName = (String)headerNames.nextElement();
            out.println("<TR><TD>" + headerName);
            out.println("    <TD>" + request.getHeader(headerName));
        }
        ...
    }
}
```

Kết quả lấy ra các Request Headers



The screenshot shows a Netscape browser window with the title "Netscape". The address bar displays "http://localhost:8080/sample/servlet/ShowRequestHeaders". The main content area has a heading "Servlet Example: Showing Request Headers" and lists the following request details:

- Request Method:** GET
- Request URI:** /sample/servlet/ShowRequestHeaders
- Request Protocol:** HTTP/1.0

Below this information is a table with two columns: "Header Name" and "Header Value".

Header Name	Header Value
User-Agent	Mozilla/4.7 [en] (Win95; I)
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding	gzip
Host	localhost:8080
Accept-Language	en
Accept-Charset	iso-8859-1,*,utf-8
Connection	Keep-Alive

The status bar at the bottom indicates "Document: Done".



Các phương thức Authentication và thông tin người dùng

- `String getRemoteUser()`
 - Định danh (name) của **client user** nếu servlet là **password protected**, null nếu ngược lại
- `String getAuthType()`
 - Loại kỹ thuật authentication sử dụng để bảo vệ servlet
- `boolean isUserInRole(java.lang.String role)`
 - User có “role” hay không?



Các phương thức Cookie (trong HttpServletRequest)

- `Cookie[] getCookies()`
 - Một mảng chứa tất cả các đối tượng Cookie client gửi trong request

6. Servlet Response (HttpServletResponse)

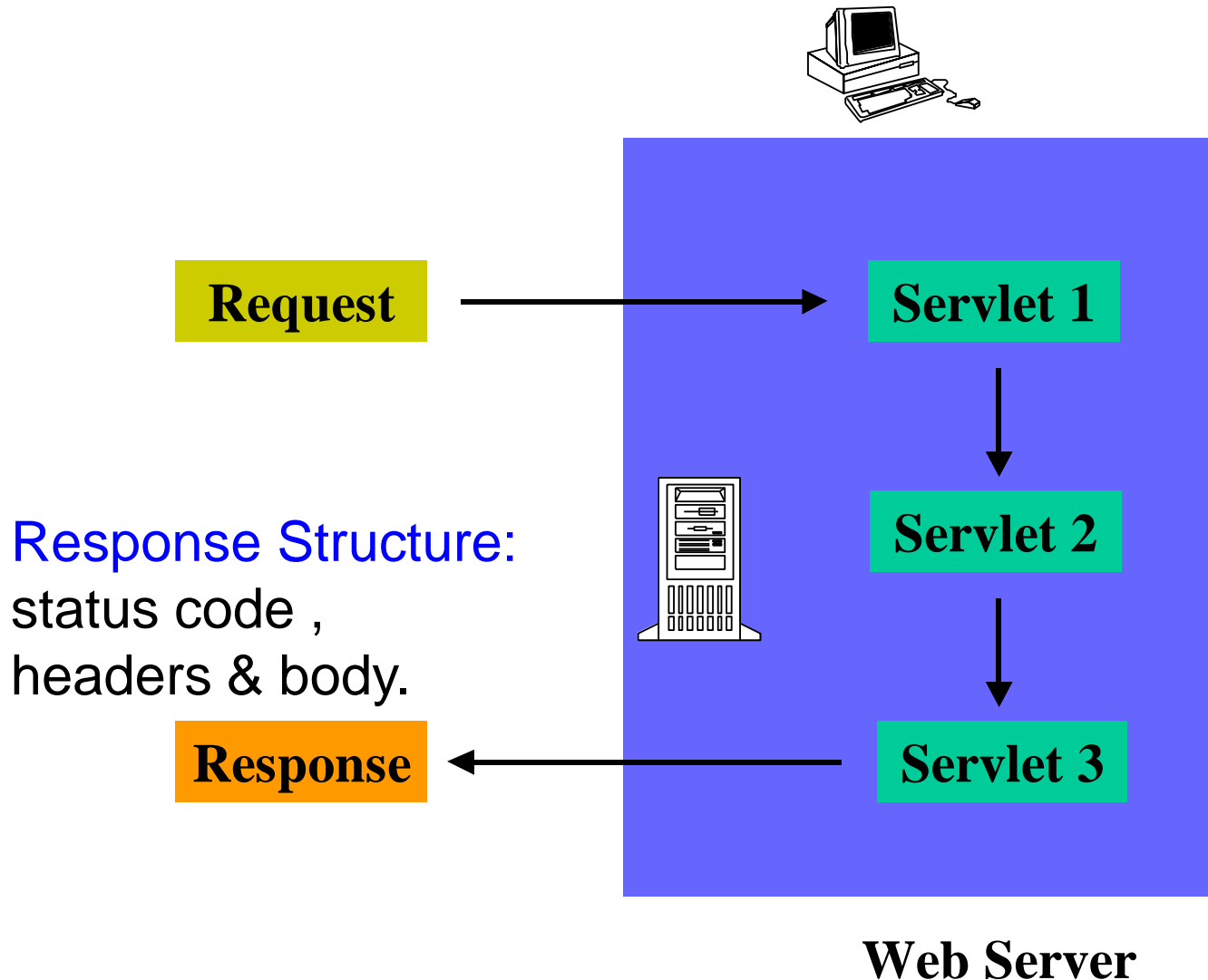




Servlet Response là gì?

- Chứa dữ liệu truyền từ servlet về client
- Tất cả các các servlet responses thực thi giao diện `ServletResponse`
 - Lấy 1 **output stream**
 - Chỉ định **content type**
 - Có thiết lập buffer đầu ra không
 - Thiết lập **localization information**
- `HttpServletResponse` kế thừa giao diện `ServletResponse`
 - Mã trạng thái HTTP trả về (**HTTP response status code**)
 - Cookies

Responses





Cấu trúc Response

Status Code

Response Headers

Response Body



6.1. Mã trạng thái (Status Code) trong Http Response



HTTP Response Status Codes

- Tại sao cần **HTTP response status code**?
 - Giúp trình duyệt forward đến 1 trang khác
 - Chỉ ra được có resource bị thiếu
 - Hướng dẫn browser sử dụng bản sao được cache của dữ liệu



Phương thức thiết lập HTTP Response Status Codes

- `public void setStatus(int statusCode)`
 - Mã trạng thái được định nghĩa trong `HttpServletResponse`
 - Các mã trạng thái chia làm 5 nhóm:
 - 100-199 Informational
 - 200-299 Successful
 - 300-399 Redirection
 - 400-499 Incomplete
 - 500-599 Server Error
 - Mã trạng thái mặc định là 200 (OK)



Ví dụ HTTP Response Status

```
HTTP/ 1.1 200 OK
Content-Type: text/ html
<! DOCTYPE ...>
<HTML
...
</ HTML>
```



Các mã trạng thái phổ biến

- 200 (SC_OK)
 - Mã thành công, kèm nội dung gửi theo
 - Mặc định cho servlet
- 204 (SC_No_CONTENT)
 - Mã thành công nhưng không có nội dung gửi theo
 - Trình duyệt sẽ hiển thị nội dung nhận lần trước
- 301 (SC_MOVED_PERMANENTLY)
 - Tài liệu yêu cầu đã bị loại bỏ, Trình duyệt tự động request đến địa chỉ mới



Các mã trạng thái phổ biến

- 302 (SC_MOVED_TEMPORARILY)
 - Chú ý thông điệp ở đây là “**Đã tìm thấy**” (Found)
 - Tài liệu được yêu cầu tạm thời chuyển sang nơi khác (được chỉ ra trong Location header)
 - Browsers tự động chuyển request đến vị trí mới
 - Servlets nên sử dụng phương thức sendRedirect, thay vì setStatus, khi thiết lập header này
- 401 (SC_UNAUTHORIZED)
 - Trình duyệt cố gắng truy cập trang có yêu cầu password mà không có **Authorization header**
- 404 (SC_NOT_FOUND)
 - Không có trang yêu cầu



Các phương thức gửi lỗi (Error)

- Các mã trạng thái lỗi (400-599) có thể được sử dụng trong phương thức `sendError`.
- `public void sendError(int sc)`
- `public void sendError(int code, String message)`
 - Đóng gói thông điệp trong 1 HTML document nhỏ



setStatus() & sendError()

```
try {  
    returnAFile(fileName, out)  
}  
catch (FileNotFoundException e) {  
    response.setStatus(response.SC_NOT_FOUND);  
    out.println("Response body");  
}
```

GIỐNG NHƯ:

```
try {  
    returnAFile(fileName, out)  
}  
catch (FileNotFoundException e) {  
    response.sendError(response.SC_NOT_FOUND);  
}
```


6.2. Header trong Http Response



Tại sao cần HTTP Response Headers?

- Forward đến địa chỉ mới nào
- Sửa cookies
- Cung cấp thông tin thời gian chỉnh sửa page.
- Hướng dẫn trình duyệt load lại trang sau 1 khoảng thời gian nhất định
- Đưa ra kích thước file được sử dụng trong HTTP connections loại persistent
- Chỉ định loại document sinh ra & trả về client
- ...



Các phương thức thiết lập Response Headers

- `public void setHeader(String headerName, String headerValue)`
 - Thiết lập 1 header bất kỳ
- `public void setDateHeader(String name, long millisecs)`
- `public void setIntHeader(String name, int headerValue)`
- `addHeader, addDateHeader, addIntHeader`
 - Thêm mới header



Các phương thức thiết lập các Response Headers phổ biến

- `setContentType`
 - Thiết lập **Content-Type** header. Servlets gần như luôn sử dụng phương thức này.
- `setContentLength`
 - Thiết lập **Content-Length** header. Được sử dụng cho HTTP connections loại persistent .
- `addCookie`
 - Thêm 1 giá trị trong **Set-Cookie** header.
- `sendRedirect`
 - Thiết lập **Location** header và thay đổi mã trạng thái



HTTP 1.1 Response Headers phổ biến

■ Location

- Chỉ ra địa chỉ mới của 1 document
- Nên sử dụng phương thức sendRedirect, thay vì thiết lập trực tiếp

■ Refresh

- Chỉ ra khoảng thời gian định kỳ trình duyệt tự động load lại trang

■ Set-Cookie

- Cookies mà trình duyệt phải lưu trữ.
- Không thiết lập trực tiếp header này, mà sử dụng phương thức addCookie



HTTP 1.1 Response Headers phổ biến (2)

- Cache-Control (1.1) và Pragma (1.0)
 - Giá trị no-cache ngăn trình duyệt **caching page**.
Gửi cả 2 loại headers hoặc check phiên bản HTTP
- Content-Encoding
 - Cách thức mã hóa document.
 - Browser giải mã trước khi xử lý document
- Content-Length
 - Số byte trong response. Được sử dụng cho HTTP connections loại persistent .



HTTP 1.1 Response Headers phổ biến (3)

■ Content-Type

- Loại MIME của document được trả về.
- Sử dụng phương thức **setContentTypes** để thiết lập header này.

■ Last-Modified

- Thời điểm thay đổi cuối cùng của document
- Cung cấp phương thức **getLastModified** thay vì thiết lập trực tiếp header này,.



Ví dụ thiết lập Refresh header

```
public class DateRefresh extends HttpServlet {  
    public void doGet(HttpServletRequest req,  
        HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/plain");  
        PrintWriter out = res.getWriter();  
        res.setHeader("Refresh", "5");  
        out.println(new Date().toString());  
    }  
}
```


6.3. Body trong Http Response



Tạo Response Body

- Một servlet gần như luôn trả về 1 response body
- Response body có thể là một **PrintWriter** hoặc một **ServletOutputStream**
- **PrintWriter**
 - Sử dụng phương thức **response.getWriter()**
 - Cho output loại ký tự (**character-based**)
- **ServletOutputStream**
 - Sử dụng phương thức **response.getOutputStream()**
 - Cho dữ liệu dạng binary (ví dụ: image)



7. Xử lý lỗi (Errors)



Xử lý lỗi

- Web container sinh ra trang hiển thị lỗi (error page) mặc định
 - LTV có thể thay bằng trang mới
- Các bước xử lý lỗi:
 - Tạo các trang html tương ứng với các loại lỗi khác nhau
 - Chỉnh sửa file web.xml



Ví dụ: Thiết lập các trang hiển thị lỗi trong Pages in web.xml

```
<error-page>
```

```
  <exception-type>
```

```
    exception.BookNotFoundException
```

```
  </exception-type>
```

```
  <location>/errorpage1.html</location>
```

```
</error-page>
```

```
<error-page>
```

```
  <exception-type>
```

```
    exception.BooksNotFoundException
```

```
  </exception-type>
```

```
  <location>/errorpage2.html</location>
```

```
</error-page>
```

```
<error-page>
```

```
  <exception-type>exception.OrderException</exception-type>
```

```
  <location>/errorpage3.html</location>
```

```
</error-page>
```