Tìm hiểu Servlet

1. Giới thiệu về Servlet

Hiện nay, trong lập trình có một xu hướng rất quan trọng đang được tập trung phát triển ứng dụng, đó là xây dựng các chương trình dịch vụ Java ở phía máy chủ (Server).

Servlet là thành phần chính được sử dụng để phát triển các chương trình dịch vụ Java ở phía máy chủ. Các Servlet là các chương trình Java thực hiện ở các ứng dụng Server (tên gọi "Servlet" cũng gần giống như "Applet" ở phía máy Client) để trả lời cho các yêu cầu của Client. Các Servlet không bị ràng buộc chặt với một giao thức Client-Server cụ thể nào cả, nhưng giao thức thường được sử dụng là HTTP, do vậy, khi nói tới Servlet nghĩa là nói tới HTTP Servlet. Servlet là sự phát triển mở rộng của CGI để đảm bảo Server thực hiện được các chức năng của mình. Ta có thể sử dụng Servlet của Java để tuỳ chỉnh lại một dịch vụ bất kỳ, như Web Server, Mail Server, v.v.

Web Server hiển thị các tư liệu được viết trong HTML và *hồi đáp* cho yêu cầu của người sử dụng qua HTTP. Các tư liệu HTML chứa các văn bản được đánh dấu (định dạng) để các trình duyệt như IE, Netscape đọc được.

Một trình duyệt chấp nhận đầu vào ở dạng HTML, khi người sử dụng nhấn một nút để yêu cầu một số thông tin nào đó, một Servlet đơn giản được gọi để xử lý các yêu cầu đó. Các công việc chính của Servlet được mô tả khái quát trong hình 1, bao gồm:

- Đọc các dữ liệu tường minh được Client gửi đến từ các yêu cầu (dữ liệu theo các khuôn dạng – form data).
- Đọc các dữ liệu không tường minh được Client gửi đến từ các yêu cầu (dữ liệu trong phần đầu của yêu cầu – request headers).
- Xử lý và lưu trữ các dữ liệu được cung cấp dưới dạng HTML.
- Gửi trả lời dữ liệu tường minh cho Client (dạng HTML), cung cấp các nội dung động, ví dụ trả lời yêu cầu Client về các câu truy vấn vào các CSDL.
- Quản lý các thông tin trạng thái và trả lời dữ liệu không tường minh cho Client (các mã trạng thái và các phần đầu của trả lời).

Hình 1 Vai trò của Servlet

Viết một Servlet là tương đối dễ. Ta chỉ cần có Tomcat, nó là tổ hợp của Java Server Pages™ 1.1 và Servlets 2.2.Tomcat có thể nạp miễn phí từ http://java.sun.com/products/jsp/tomcat/, phần cài đặt sẽ được mô tả ở phần sau.

Các Servlet cũng được sử dụng thay cho kịch bản giao diện cổng chung CGI Script. Khi tạo ra một trang Web, ta cũng sẽ tạo ra một ứng dụng Web.

Trước khi sử dụng Servlet để tạo ra các ứng dụng Web, chúng ta đi tìm hiểu xem có những khả năng lựa chọn nào khác để phát triển những ứng dụng Web.

- **CGI**: Theo cách thông thường, để bổ sung các chức năng vào cho một Web Server người ta hay sử dụng Common Gateway Interface (CGI), một giao diện độc lập với ngôn ngữ cho phép một Server khởi động một tiến trình ngoại để nhận thông tin được yêu cầu thông qua các biến môi trường. Mỗi yêu cầu được trả lời bởi một tiến trình riêng thông qua một đại diện riêng của một chương trình CGI hoặc bởi một kịch bản CGI (thường được viết bằng ngôn ngữ thông dịch như Perl).

- **Fast CGI**: Open Marked đã phát triển một chuẩn khác thay cho CGI được gọi là Fast CGI. Fast CGI hành động giống như CGI. Nó khác ở chỗ, Fast CGI tạo ra một tiến trình bền vững cho từng chương trình.

- Một số chương trình ứng dụng khác như ASP và Java Script cũng hỗ trợ để tạo ra các ứng dụng Web. ASP được Microsoft phát triển để tạo ra các nội dung cho các trang Web động. Trong ASP, trang HTML có thể nhúng những phần nhỏ được viết bằng VBScript hoặc JScript. Netscape đưa ra kỹ thuật được gọi là JavaScript, cho phép đưa các phần mã lệnh nhỏ nhúng vào trang HMTL, nhằm tạo ra những nội dung Web động một cách linh hoạt hơn. Ngoài ra, Netscape còn cung cấp NSAPI, Microsoft đưa ra ISAPI cho các Web Server của họ.

Servlet có một số ưu điểm so với CGI:

Một Servlet không làm việc trong một tiến trình riêng. Điều này loại bỏ được việc phải tạo ra quá nhiều tiến trình mới cho mỗi yêu cầu.

Một Servlet sẽ thường trực trong bộ nhớ giữa các yêu cầu, trong khi các chương trình CGI cần phải tải xuống và được khởi động cho từng yêu cầu CGI.

Chỉ cần một Servlet trả lời đồng thời cho tất cả các yêu cầu. Điều này cho phép tiết kiệm được bộ nhớ và đảm bảo nó dễ dàng quản lý được dữ liệu một cách thống nhất.

Một Servlet có thể thực hiện bởi một Servlet Engine trong phạm vi kiểm soát Sandbox để đảm bảo an toàn trong việc sử dụng các Servlet.

Các lớp Servlet của Java có thể được nạp tự động để mở rộng các chức năng của Server. Các Servlet của Java thực hiện bên trong JVM. Chúng được đảm bảo an toàn và chuyển đổi tương thích giữa các hệ điều hành và giữa các Server với nhau. Điều này khác với các Applet, Servlet chỉ thao tác được trong miền của một Server.

Servlet API được phát triển dựa trên những điểm mạnh của Java platform nhằm giải quyết vấn đề của CGI và Server API. Nó là một API đơn giản, hỗ trợ tất cả các Web server và thậm chí cho phép các ứng dụng máy chủ dùng để kiểm tra và quản lý các công việc trên Server. Nó giải quyết vấn đề thực thi bằng việc thực hiện tất cả các yêu cầu như các luồng Thread trong quá trình xử lý, hoặc việc cân bằng tải trên một Server trong các cụm máy tính Cluster. Các Servlet dễ dàng chia sẻ tài nguyên với nhau.

Trong định nghĩa Servlet, vấn đề bảo mật được cải tiến theo nhiều cách. Trước hết, bạn hiếm khi thực thi được các câu lệnh trên Shell với dữ liệu cung cấp bởi người dùng mà Java API đã cung cấp với những khả năng truy cập đến tất cả các hàm thông dụng. Bạn có thể sử dụng Java Mail để đọc và gửi mail, kết nối vào các CSDL (thông qua JDBC), tệp lớp (.class) và những lớp liên quan để truy cập hệ thống tệp, CSDL, RMI, CORBA, Enterprise Java Beans (EJB), …

2. Ưu điểm của Servlet

Servlet được sử dụng để thay thế cho những công nghệ Web động. Việc sử dụng Servlet mang lại những lợi thế:

- *Dễ di chuyển.* Servlet được viết bằng Java nên nó có tính di động cao, thực hiện được trên nhiều hệ điều hành, trên các Web Server khác nhau. Khái niệm "Viết một lần, chạy ở mọi nơi" cũng rất đúng với Servlet.
- *Mạnh mẽ.* Servlet hỗ trợ rất hiệu quả cho việc sử dụng các giao diện lõi API như lập trình mạng, xử lý đa luồng, xử lý ảnh, nén dữ liệu, kết nối các CSDL, bảo mật, xử lý phân tán và triệu gọi từ xa RMI, CORBA, v.v. Nó cũng thích hợp để trao đổi tin, truyền thông giữa Client và Server một cách bình thường.
- *Hiệu quả.* Servlet có tính hiệu quả cao. Một khi được tải về, nó sẽ được lưu lại trong bộ nhớ của máy chủ. Servlet duy trì các trạng thái của nó, do vậy những tài nguyên ngoại như việc kết nối với CSDL cũng sẽ được lưu giữ lại.

- *An toàn.* Bởi vì Servlet được viết bằng Java nên nó kế thừa được tính an toàn của Java. Cơ chế tự động dọn rác và việc không sử dụng con trỏ của Java giúp cho Servlet thoát khỏi nhiều công việc quản lý bộ nhớ. Đồng thời nó xử lý các lỗi rất an toàn theo cơ chế xử lý ngoại lệ của Java.
- *Tính tích hợp.* Các Servlet được tích hợp với các Server. Chúng cộng tác với các Server tốt hơn các chương trình CGI.
- *Tính linh hoạt.* Các Servlet hoàn toàn mềm dẻo. Một HTTP Servlet được sử dụng để tạo ra một trang Web, sau đó ta có thể sử dụng thẻ <Servlet> để đưa nó vào trang Web tĩnh, hoặc sử dụng với các Servlet khác để lọc ra các nội dung cần thiết.

3. Môi trường thực hiện Servlet

Các Servlet thường là sự mở rộng (kế thừa) các lớp chuẩn Java trong gói javax.servlet (chứa các khuôn mẫu cơ bản của Servlet) và javax.servlet.http (mở rộng các khuôn mẫu cơ bản của Servlet và các yêu cầu theo HTTP).

Servlet là một lớp Java và vì thế cần được thực thi trên một máy ảo Java (JVM) và bằng một dịch vụ được gọi là mô tơ Servlet (Servlet Engine). Servlet Engine tải lớp Servlet lần đầu tiên nó được yêu cầu, hoặc ngay khi Servlet Engine được bắt đầu. Servlet ngừng tải để xử lý nhiều yêu cầu khi Servlet Engine bị tắt hoặc nó bị dừng lại.

Như vậy, để dịch và thực hiện các Servlet, việc có các Servlet là chưa đủ, mà cần phải có một mô tơ Servlet để kiểm tra và triển khai chúng. Hiện nay một số mô tơ tương thích với nhiều loại Web Server khác nhau, nhưng nguyên lý hành động tương đối giống nhau. Người ta chia chúng thành ba loại.

- Mô tơ Servlet đơn
- Mô tơ Servlet gộp
- Mô tơ Servlet nhúng.

3.1 Mô tơ Servlet đơn

Đây là loại Server được xây dựng để hỗ trợ cho các Servlet. Ưu điểm của nó là mọi thứ làm việc với các hộp kết quả đầu ra rất phong phú. Tuy nhiên, nó có nhược điểm là ta phải chờ những phiên bản mới của Web Server để nhận được những hỗ trợ mới nhất cho Servlet. Hiện nay có các loại mô tơ đơn sau:

- *Java Server Web Development (JSWDK) của Sun Microsystem:* được viết hoàn toàn bằng Java: http://java.sun.com.products/servlet/. Nó được sử

dụng như là một Server độc lập để kiểm tra các Servlet và các trang JSP trước khi phát triển thành một Web Server thực sự.

- *Jagsaw Server của WWW Consortium,* cũng được viết bằng Java. Chi tiết hơn, xem http://www.w3.org/Jigsaw.
- *Netscape Enterprise Server.* Đây là Web Server rất nổi tiếng, nó hỗ trợ để xây dựng Servlet.
- *Lotus Domino Go WebServer.* Một loại Web Server khác cũng hỗ trợ để xây dựng Servlet.

3.2 Mô tơ Servlet gộp

Servlet gộp là loại mô tơ được viết cho nhiều loại Server khác nhau, kể cả Apache, Fast Track Server, Enterprise Server của Netscape, Personal Web Server, v.v. Hiện nay có các loại sau:

- *Apache Tomcat:* Mô tơ này hỗ trợ thêm cho Apache. Nó được sử dụng như là một Server độc lập để kiểm tra các Servlet và các trang JSP, hoặc được tích hợp vào Apache Web Server, http://java.sun.com.products/servlet/.
- *.Jrun của Live Software:* Jrun là mô tơ cho Servlet và JSP, hỗ trợ đầy đủ Servlet API trong các Web Server phổ biến trên mọi hệ điều hành, http://www.allaire.com.products/jrun/.
- *WebSphere Application Server của IBM:* còn được gọi là ServletExpress.
- *ServletExec của New Atlanta:* ServletExec là mô tơ cho Servlet và JSP, hỗ trợ đầy đủ Servlet API trong các Web Server phổ biến trên mọi hệ điều hành, http://newatlanta.com/.

3.3 Mô tơ Servlet nhúng

Loại mô tơ này có thể nhúng vào phần mềm ứng dụng khác. Hiện nay có các loại sau:

- *Java Server Engine của Sun*. Đây là loại mô tơ được viết hoàn toàn bằng Java và là Web Server đầu tiên hỗ trợ đầy đủ cho các đặc tả của Servlet 2.1 và JSP 1.0, http://www.sun.com.software/jwebserver/try.
- *Nexus Web Server của Anders Kristensen.* Nó có thể dễ dàng nhúng vào các chương trình ứng dụng Java.

4. Kiến trúc của Servlet

Gói javax.servlet cung cấp các giao diện và các lớp để xây dựng các Servlet. Kiến trúc của chúng được mô tả như sau.

Hình 6.3 Kiến trúc của Servlet

4.1 Giao diện Servlet

Giao diện Servlet là một khái niệm trừu tượng trung tâm trong Servlet API. Tất cả các Servlet đều cài đặt trực tiếp hoặc gián tiếp giao diện này hoặc mở rộng (kế thừa) những lớp đã cài đặt nó.

Giao diện này khai báo ba phương thức định nghĩa vòng đời của Servlet.

- public void init(ServletConfig config) throws ServletException

  Phương thức này được gọi một lần khi Servlet được tải vào trong Servlet Engine, trước khi Servlet được yêu cầu để xử lý một yêu cầu nào đó. Phương thức init() có một thuộc tính là đối tượng của ServletConfig, và Servlet có thể đọc các đối số khởi tạo của nó thông qua đối tượng ServletConfig. Chúng thường được định nghĩa trong một tệp cấu hình. Một ví dụ thông thường của một đối số khởi tạo là định danh database cho một CSDL.

```
...
private String databaseURL;
public void init(ServletConfig config) throws ServletException {
  super.init(config);
  databaseURL = config.getInitParameter("database");
 }
```

- public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException

  Phương thức này được gọi để xử lý các yêu cầu. Nó có thể không được gọi, gọi một lần hay nhiều lần cho đến khi Servlet được ngưng tải. Nhiều Thread (mỗi Thread cho một yêu cầu) có thể thực thi phương thức này song song, vì thế nó trở nên an toàn và hiệu quả hơn.

- public void destroy()

  Phương thức này chỉ được gọi một lần trước khi Servlet được ngưng tải và sau khi đã kết thúc các dịch vụ.

Servlet API có cấu trúc để Servlet có thể cho phép bổ sung một giao thức khác với HTTP. Gói javax.servlet chứa các lớp và các giao diện được kế thừa giao diện Servlet một cách độc lập. Gói javax.servlet.http chứa các lớp và giao diện HTTP cụ thể.

4.2. Lớp cơ sở HttpServlet

Như ta đã biết, theo giao thức HTTP, dữ liệu được trao đổi giữa máy chủ Server và các máy Client theo một trong hai phương thức GET hay POST. Java định nghĩa một lớp có tên là HttpServlet ở trong goi javax.servlet để truyền và nhận dữ liệu theo cả hai phương thức trên.

Lớp trừu tượng HttpServlet cung cấp một khung làm việc để xử lý các yêu cầu GET, POST của giao thức HTTP. HTTPServlet kế thừa giao diện Servlet cộng với một số các phương thức hữu dụng khác.

Một tập các phương thức trong HTTPServlet là những phương thức xác định dịch vụ trong giao diện Servlet. Việc bổ sung dịch vụ trong HTTPServlet giống như một kiểu của các yêu cầu được xử lý (GET, POST, HEAD, …) và gọi một phương thức cụ thể cho mỗi kiểu. Bằng việc làm này, các nhà phát triển Servlet sẽ an tâm khi xử lý chi tiết những yêu cầu như HEAD, TRACE, OPTIONS, … và có thể tập trung vào những yêu cầu thông dụng hơn như GET và POST.

HTTP sinh ra các trang HTML và ta có thể nhúng các Servlet vào một trang HTML.

Khi có một yêu cầu được gửi tới, đầu tiên nó phải chỉ ra lệnh cho HTTP bằng cách gọi một phương thức tương ứng. Phương thức này chỉ cho Server biết kiểu hành động mà nó muốn thực hiện.

Khi có một Client gửi tới một yêu cầu, Server sẽ xử lý yêu cầu nhận được và gửi trả lại kết quả cho Client. Hai phương thức doGet() và doPost() được sử dụng chung để nhận và gửi tin trong các Servlet.

Một Servlet bất kỳ, ví dụ MyServlet phải kế thừa HttpServlet và viết đè ít nhất một trong các phương thức doGet() để thực thi thao tác GET của HTTP, hay doPost() để thực thi thao tác POST của HTTP.

Trong ví dụ đầu tiên, chúng ta sẽ viết đè phương thức doGet() dạng

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

Phương thức doGet() có hai tham số đối tượng thuộc hai lớp HttpServletRequest và HttpServletResponse (cả hai lớp này được định nghĩa trong javax.servlet.http). Hai đối tượng này cho phép chúng ta truy cập đầy đủ tất cả các thông tin yêu cầu và cho phép gửi dữ liệu kết quả cho Client để trả lời cho yêu cầu đó.

Với CGI, các biến môi trường và stdin được sử dụng để nhận thông tin về yêu cầu, tuy nhiên việc đặt tên các biến môi trường có thể khác nhau giữa các chương trình CGI, và một vài biến có thể không được cung cấp bởi tất cả các Web Server.

Đối tượng HttpServletRequest cũng cung cấp thông tin giống như biến môi trường của CGI nhưng theo một hướng chuẩn. Nó cũng cung cấp những phương thức để mở ra các tham số HTTP từ dãy các câu truy vấn hoặc từ nội dung của yêu cầu phụ thuộc vào kiểu yêu cầu (GET hay POST).

Ví dụ 1 Chương trình FirstServlet để đếm và hiển thị số lần nó được truy cập.

```
package myservlet;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet{
     int i = 0;
     public void doGet(HttpServletRequest re, HttpServletResponse resp)
                throws ServletException, IOException{
          resp.setContentType("text/html");
          PrintWriter out = resp.getWriter();
          i++;
          out.println("So lan duoc truy cap: " + i);
          out.close();
     }
}
```

doGet(HttpServletRequest req, HttpServletResponse resp) sử dụng đối số thứ nhất req để đọc các phần đầu, tiêu đề (header) của HTTP gửi tới (ví dụ như dữ liệu dạng HTML mà người dùng nhập vào), và sử dụng resp để xác định dòng trả lời cho HTTP (xác định kiểu nội dung trao đổi, đặt Coookie). Điều quan trọng nhất là phải nhận được một đối tượng của PrintWriter (ở java.io) thông qua resp.getWriter()để gửi kết quả trả lại cho Client. Ngoài doGet(), doPost() HttpServlet còn có các phương thức:

- service(): thực hiện khi đối tượng của lớp được tạo lập và triệu gọi doGet() hoặc doPost().
- doPut(): thực hiện thao tác PUT của HTTP.
- doDelete(): thực hiện thao tác DELETE của HTTP.
- init() và destroy(): khởi tạo và huỷ bỏ các Servlet.
- getServletInfo(): nhận các thông tin về Servlet.

Lưu ý: Cả doGet() và doPost() đều có thể phát sinh ra một trong hai ngoại lệ ServletException, hay IOException, do vậy ta phải khai báo chúng như trên. Ngoài ra, một điều nữa chúng ta cũng chú ý là hai phương thức doGet() và doPost() sẽ được phương thức service() gọi để thực hiện và đôi lúc ta có thể viết đè service() thay cho hai phương thức trên.

5. Chu trình sống của các Servlet

Mọi Servlet đều có chu trình sống như sau:

- Server nạp và khởi tạo Servlet
- Servlet xử lý các yêu cầu của các Client
- Server huỷ bỏ Servlet khi không còn cần thiết.

Chu trình sống của Servlet

Như trên đã phân tích, mọi Servlet đều có một chu trình sống nhất định. Nó được khởi tạo (nạp về), xử lý các yêu cầu của Client và sau khi hoàn tất các dịch vụ thì bị Server huỷ bỏ (Hình 6.4).

5.1 Khởi động Servlet

Việc khởi động một Servlet được thực hiện mặc định bởi HttpServlet. Để khởi động một Servlet riêng, ta phải viết đè phương thức init().

```
public class MyServlet …{
     public void init() throws ServletException{
          // Khởi tạo các giá trị ban đầu
     }
     . . .
}
```

Khi một Server nạp xong một Servlet thì nó gọi init() để bắt đầu Servlet đó.

Lưu ý: Server chỉ gọi init() một lần khi nạp Servlet và sau đó sẽ không gọi lại nữa, trừ khi phải nạp lại nó. Server không thể nạp lại nếu Servlet đó chưa bị huỷ bỏ bởi lời gọi destroy(). Phương thức init() có thể được nạp chồng theo mẫu

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

như đã nêu ở trên.

5.2 Tương tác với các Client

Lớp HttpServlet xử lý các yêu cầu của Client thông qua các dịch vụ của nó. Các phương thức trong HttpServlet xử lý yêu cầu của Client đều có hai đối số:

- Một là đối tượng của HttpServletRequest, bao gồm cả dữ liệu từ Client. Nó cho phép nhận được các tham số mà Client gửi đến như một phần của các yêu cầu, thông qua các phương thức getParameterName(), getParameterValue() để xác định tên gọi và giá trị của các tham số.
- Hai là đối tượng của HttpServletResponse, bao gồm cả dữ liệu hồi đáp cho Client. HttpServletResponse cung cấp hai phương thức để trả lại kết quả cho Client. Phương thức getWriter() ghi dữ liệu dưới dạng văn bản còn getOutputStream() cho lại dữ liệu dạng nhị phân.

Ngoài ra, để xử lý được các yêu cầu của HTTP gửi đến cho một Servlet thì phải mở rộng lớp HttpServlet và viết đè các phương thức xử lý các yêu cầu như doGet(), doPost().

5.3 Huỷ bỏ Servlet

Sau khi nạp các Servlet và xử lý chúng xong thì cần phải dọn dẹp hệ thống, loại bỏ những Servlet không còn được sử dụng tiếp. Phương thức destroy() của lớp HttpServlet được sử dụng để loại bỏ một đối tượng Servlet khi nó không còn cần thiết. Để loại bỏ một tài nguyên nào đó cụ thể trong một Servlet riêng thì phải viết đè phương thức destroy().

Server sẽ gọi destroy() để huỷ một Servlet, sau khi nó kết thúc tất cả các dịch vụ theo yêu cầu. Ví dụ sau đây mô tả một Servlet mở một kết nối với CSDL thông qua phương thức init() và sau đó nó sẽ bị phá huỷ bởi phương thức destroy().

```
public class DBServlet extends HttpServlet{
  Connection connection = null;
  public void init() throws ServletException{
     // Mở một kết nối với CSDL
     try{
       databaseUrl = getInitParameter("databaseUrl");
       // Đọc tên người sử dụng và mật khẩu
```

```
            connection = DriverManager.getConnection(
               userName, password);
         }catch(Exception e){
            throw new UnavailableException(this,
               "Không mở được CSDL");
         }
      }
      // …
      public void destroy(){
         // Đóng các kết nối để dọn dẹp bộ nhớ
         connection.close();
         connection = null;
      }
   }
```

Server sẽ gọi destroy() sau khi tất cả các dịch vụ đã được kết thúc.

6. Xử lý các yêu cầu

Nhiệm vụ của các Servlet là nhận các yêu cầu, xử lý chúng và trả lời cho các Client.

6.1 Truy tìm thông tin

Để xây dựng thành công một WebSite, ta cần có những thông tin từ Server, nơi thực hiện các Servlet. Servlet có những phương thức sau giúp cho việc nhận được những thông tin yêu cầu.

- int port = req.getServerPort(): Xác định cổng trao đổi tin của máy chủ.
- public String ServletConfig.getInitParameter(String name): Cho lại giá trị ban đầu của tham số có tên name.
- public Enumeration ServletConfig.getInitParameterName(): Xác định dãy liệt kê tên gọi của tất cả các tham số khởi đầu của Servlet như là đối tượng của Enumeration. Lớp GenericServlet sử dụng phương thức này để truy cập đến các Servlet.

Ví dụ Servlet sau in ra tên và giá trị ban đầu của tất cả các tham số.

```
import java.io.*;
import javax.servlet.*;
import java.util.*;
public class ReadServlet extends GenericServlet{

      public void service(ServletRequest re, ServletResponse resp)
```

```
            throws IOException{
        resp.setContentType("text/plain");
        PrintWriter out = resp.getWriter();
        out.println("Tham so khoi dau la:");
        Enumeration eno = getInitParameterNames();
        while(eno.hasMoreElements()){
            String nm = (String)eno.nextElement();
            out.println(nm + " : " + getInitParameter(nm));
        }
    }
}
```

6.2 Gửi thông tin

Các Servlet cần phải trả lời cho Client về các thông tin dưới dạng HTML bao gồm:

- *Các mã hiện trạng*. Mã hiện trạng là một số nguyên, nó mô tả tình trạng của việc hồi đáp thành công hay thất bại. Phần lớn các mã này đã được định nghĩa trong lớp HttpServletResponse như ở bảng 6.1.
- *Số các phần đầu (tiêu đề) của HTTP*. Phương thức setHeader() của lớp HttpServletResponse cho phép đặt lại các giá trị cho các tiêu đề.
- *Nội dung hồi đáp*. Nội dung *hồi đáp* dưới dạng một trang HTML.

Bảng 6.1

Tên gợi nhớ	Mã số	Thông báo
SC_OK	200	Ok (tốt)
SC_NO_CONTENT	204	Không có nội dung
SC_MOVED_PERMANENTLY	301	Đã chuyển đi vĩnh viễn
SC_MOVED_TEMPORARLY	302	Đã chuyển đi tạm thời
SC_NOT_FOUND	404	Không tìm thấy
SC_UNAUTHORIZED	401	Không được phép

Ví dụ Servlet gửi cho Client một trang ngẫu nhiên trong danh sách các trang Web của nó.

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

```
import java.util.*;

public class RandomSend extends HttpServlet{
     Vector st = new Vector();
     Random rd = new Random();
     public void init(ServletConfig config) throws ServletException{
          super.init(config);
          st.addElement("http://www.yahoo.com");
          st.addElement("http://www.hotmail.com");
          st.addElement("http://www.zednet.com");
          st.addElement("http://www.java.sun.com");
     }
     public void doGet(HttpServletRequest re, HttpServletResponse res)
               throws IOExceptio{
          res.setContentType("text/html");
          PrintWriter out = resp.getWrite();
          int siteIndex = Math.abs(rd.nextInt()) % st.size();
          String st = (String) st.elementAt(siteIndex);
          res.setStatus(res.SC_MOVED_TEMPORARLY);
          res.setHeadr("Location", st);
     }
}
```

Khi một Client dùng HTML để gửi đi một yêu cầu thì nó có thể sẽ gửi đi một số các tiêu đề (Header).

Việc đọc các tiêu đề là tương đối đơn giản, có thể sử dụng hàm getHaeder() của HttpServletRequest để nhận được các tiêu đề (header) yêu cầu. Sau đây là các tiêu đề phổ dụng trong các yêu cầu.

- Accept: Các kiểu tệp được trình duyệt chấp nhận, như các tệp image/gif, image/jpeg, hay application/msword, v.v.
- Accept-Charset: Tập các ký tự mà trình duyệt mong đợi.
- Accept-Encoding: Các kiểu mã hoá dữ liệu (ví dụ gzip) mà trình duyệt biết cách giải mã.
- Accept-Language: Ngôn ngữ mà trình duyệt mong đợi.
- Authorization: Thông tin về giấy phép, uỷ quyền.
- Connection: Khẳng định có kết nối thường xuyên hay không? Nếu một Servlet đặt là Keep-Alive thì nó có thể tận dụng được những ưu thế của cơ chế kết nối thường xuyên.
- Content-Length: Số dữ liệu được đưa vào yêu cầu.

- Cookie: Một trong các tiêu đề quan trọng nhất. Một Cookie là một xâu (dãy ký tự) được gửi tới cho một Client để bắt đầu một phiên làm việc.
- Host: Máy chủ và cổng được chỉ ra trong URL.
- User-Agent: Loại trình duyệt.

Các tiêu đề trên là tuỳ chọn, trừ Content-Length là được yêu cầu chỉ đối với yêu cầu POST.

Để tìm một tiêu đề cụ thể, trước tiên ta cần sử dụng getHaederNames() để có được một dãy (đối tượng của Enumeration) tất cả các tiêu đề nhận được từ một yêu cầu cụ thể, sau đó tìm lần lượt tiêu đề đó.

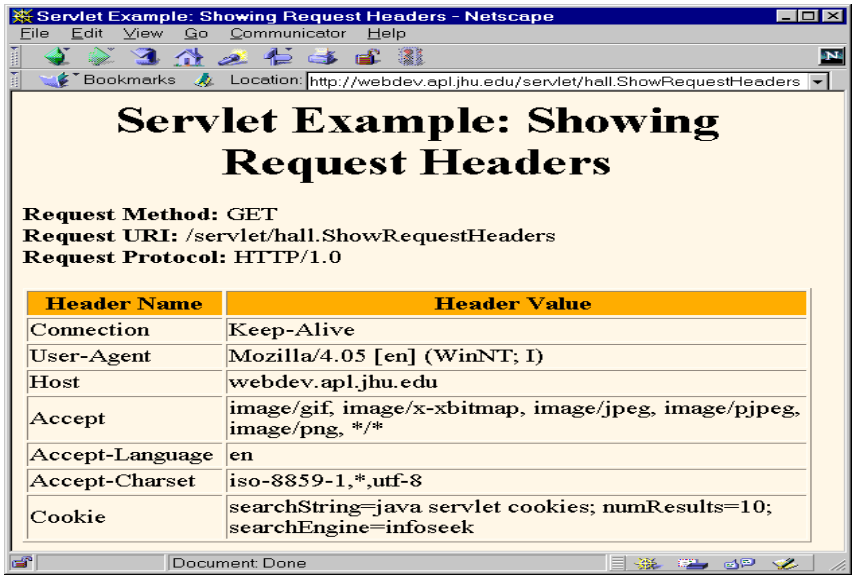
Một vấn đề nữa cần biết khi phải tìm các tiêu đề của một yêu cầu là phải biết thông tin về phương thức chính được sử dụng. Phương thức getMethod() sẽ cho ta biết về phương thức chính của mỗi yêu cầu (thường là GET, POST, hoặc cũng có thể là HEAD, PUT và DELETE). Phương thức getRequestURI() sẽ cho lại URI (một phần của URL, phần đứng sau tên của địa chỉ máy chủ với cổng kết nối và trước phần dữ liệu mẫu).

Ví dụ Xây dựng một Servlet để đọc và hiển thị một bảng các tiêu đề và nội dung của một yêu cầu.

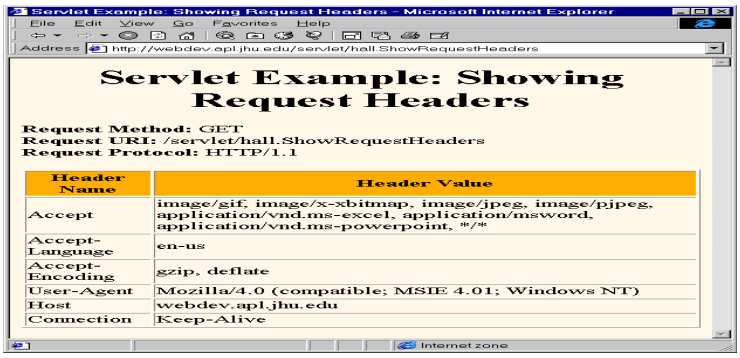
```
// ShowRequestHeaders.java
package myservlet;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

/* Chỉ ra tất cả các tiêu đề của một yêu cầu cụ thể dưới dạng một bảng bao gồm tiêu đề và
nội dung
*/
 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("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\">"
                 +  "<HTML>\n" + "<HEAD><TITLE>" + title +
                 "</TITLE></HEAD>\n" + "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                 "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                 "<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" +
                "<TABLE BORDER=1 ALIGN=CENTER>\n" +
                "<TR BGCOLOR=\"#FFAD00\">\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));
    }
    out.println("</TABLE>\n</BODY></HTML>");
  }
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    doGet(request, response);
  }
}
```

Sau đây là hai kết quả hiển thị đối với hai loại trình duyệt Netscape và Internet Explore.



Hình 6.5 Hiển thị các tiêu đề yêu cầu với Netscape



Hình 6.6 Hiển thị các tiêu đề yêu cầu với IE

6.3 Xử lý các dữ liệu mẫu

Khi sử dụng máy tìm kiếm Web Search Engine, bạn có thể truy cập dạng

```
http://host/path?user=Marty+Hall&orgin=bwi&dest=lax
```

Phần tin sau dấu '?' (user=Marty+Hall&orgin=bwi&dest=lax) được xem như là *dữ liệu mẫu*, đây là cách chung thường được sử dụng để chương trình Server nhận dữ liệu vào từ trang Web.

Việc tách những thông tin cần thiết từ dữ liệu dạng trên là một phần việc mà các chương trình CGI thường làm.

- Trước tiên, đọc dữ liệu vào cho các tham số yêu cầu của GET hoặc POST
- Sau đó, chặt ra từng cặp ở dấu '&' rồi lại tách tiếp để xác định tên của các tham biến (phần bên trái dấu '=') và giá trị của những tham biến đó (phần bên phải của dấu '=').
- Thực hiện giải mã URL theo những giá trị đó.

Lưu ý: Tất cả các chữ cái, chữ số được gửi đi và không thay đổi, nhưng dấu cách ' ' được chuyển thành dấu '+'. Những ký tự khác được chuyển thành %XX, trong đó XX là giá trị (hex) của ký tự ASCII (khác với chữ cái, chữ số). Ví dụ, nếu bạn muốn nhập vào dữ liệu mẫu "~hall, ~gates" vào trường văn bản có tên "user" ở dạng HTML thì dữ liệu gửi đi phải là "user=%7Ehall%2C+%7Egates".

Những công việc nhàm chán trên khi lập trình với CGI đã được Java hỗ trợ để xử lý các dữ liệu mẫu một cách tự động. Đơn giản là bạn gọi getParameter() của HttpServletRequest để nhận được tên gọi của tham số như là một đối số. Dữ liệu mẫu được gửi đi trong GET và POST là giống hệt nhau. Khi một tham số có nhiều hơn một giá trị thì gọi getParameterValues() thay vì gọi getParameter(), kết quả trả lại là mảng các xâu. Tương tự, sử dụng getParameterName() để xác định tập các tên gọi của các tham số, kết quả của nó là danh sách các tên gọi thuộc lớp Enumeration.

Ví dụ Chương trình đọc và hiển thị các tham số được gửi đến cho Servlet qua GET hay POST.

// ShowParameters.java

```
package hall;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
/** Hiển thị tất cả các tham số được gửi đến cho Servlet qua GET hoặc POST. Các tham
  số đặc biệt có thể * không có giá trị hoặc có nhiều giá trị.
 */
public class ShowParameters extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Reading All Request Parameters";
    out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0
              Transitional//EN\">" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<TABLE BORDER=1 ALIGN=CENTER>\n" +
                "<TR BGCOLOR=\"#FFAD00\">\n" +
                "<TH>Parameter Name<TH>Parameter Value(s)");
    Enumeration paramNames = request.getParameterNames();
    while(paramNames.hasMoreElements()) {
      String paramName = (String)paramNames.nextElement();
      out.println("<TR><TD>" + paramName + "\n<TD>");
      String[] paramValues = request.getParameterValues(paramName);
      if (paramValues.length == 1) {
        String paramValue = paramValues[0];
        if (paramValue.length() == 0)
          out.print("<I>No Value</I>");
        else
          out.print(paramValue);
      } else {
        out.println("<UL>");
        for(int i=0; i<paramValues.length; i++) {
          out.println("<LI>" + paramValues[i]);
        }
```

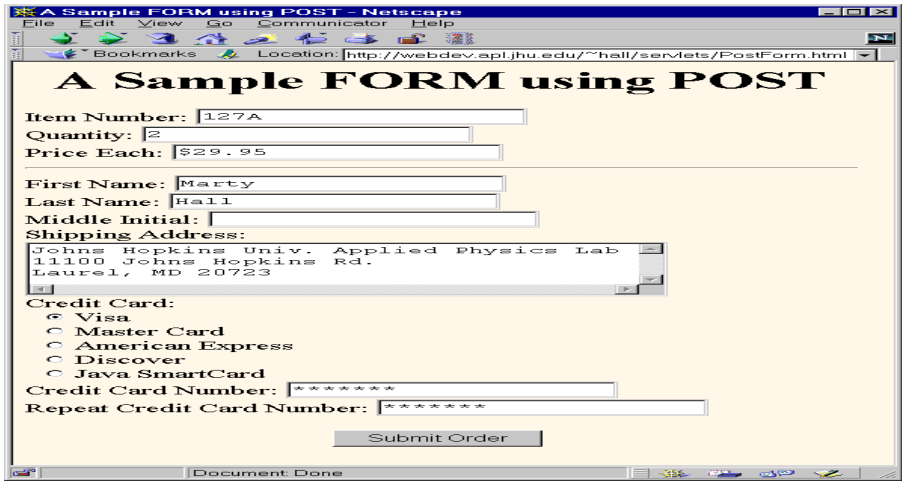
```
        out.println("</UL>");
      }
    }
    out.println("</TABLE>\n</BODY></HTML>");
  }
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    doGet(request, response);
  }
}
```

Để chương trình Servlet trên nhận được các tham số thì cần phải có một trang HTML gửi chúng. Trang HTML (PostForm.html) sau đây sử dụng POST để gửi dữ liệu (theo các mẫu forms có các mục đầu vào), thể hiện các giá trị mà Servlet nhận được theo cả hai phương thức doGet() và doPost().

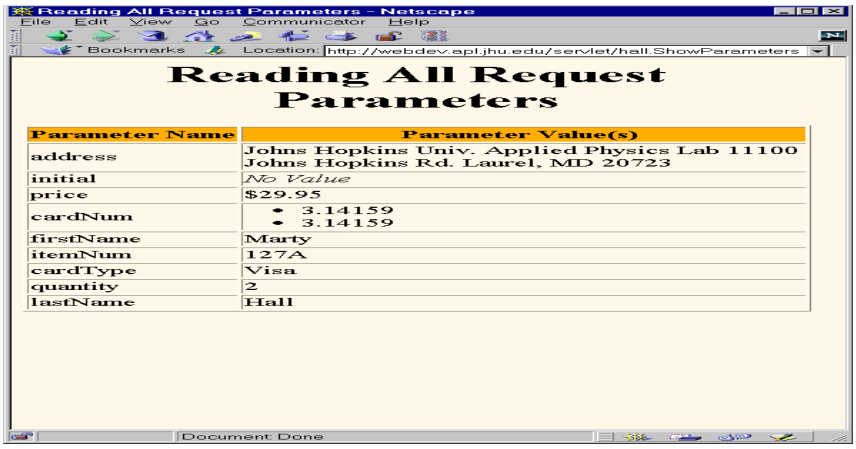
Tệp PostForm.html

```
<!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="/servlet/hall.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>
  <HR>
  First Name:
  <INPUT TYPE="TEXT" NAME="firstName"><BR>
  Last Name:
  <INPUT TYPE="TEXT" NAME="lastName"><BR>
  Middle Initial:
  <INPUT TYPE="TEXT" NAME="initial"><BR>
  Shipping Address:
  <TEXTAREA NAME="address" ROWS=3 COLS=40></TEXTAREA><BR>
  Credit Card:<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
```

```
                    VALUE="Visa">Visa<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
                    VALUE="Master Card">Master Card<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
                    VALUE="Amex">American Express<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
                    VALUE="Discover">Discover<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
                    VALUE="Java SmartCard">Java SmartCard<BR>
  Credit Card Number:
  <INPUT TYPE="PASSWORD" NAME="cardNum"><BR>
  Repeat Credit Card Number:
  <INPUT TYPE="PASSWORD" NAME="cardNum"><BR><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT" VALUE="Submit Order">
  </CENTER>
</FORM>
</BODY>
</HTML>
```



Hình 6.7 Trang PostForm.html



Hình 6.8 Chương trình đọc các tham số ShowParameters.java

7. Các phiên làm việc Session

Session được sử dụng để duy trì sự kết nối giữa Client và Server. Khi trình duyệt yêu cầu Web Server cung cấp một trang tài liệu, nó thiết lập một kết nối, lấy về nội dung của trang yêu cầu và sau đó huỷ bỏ kết nối đó ngay lập tức. Để lưu lại dấu vết các trạng thái mà trình duyệt yêu cầu thực hiện trước đó, Web Server cài đặt sẵn đối tượng của HttpSession. HttpSession dựa vào khái niệm Cookie qui định giữa Server và Client. *Cookie là một mẩu tin được gửi cho trình duyệt ở Client khi có yêu cầu về một trang tin*. Mỗi khi trình duyệt gửi một yêu cầu cho Server, nó lại chuyển mẩu tin Cookie trả lại cho Server. Dựa vào các Cookie mà chương trình Server và Client có được những thông tin trạng thái thông báo cho nhau.

Giao diện HttpSession có 3 phương thức thường xuyên sử dụng:

- public void setAttribute(String name, Object value) throws IllegalStateException: Kết hợp một tên khoá với giá trị của biến.
- public Object setAttribute(String name) throws IllegalStateException: Trả về đối tượng tương ứng với thuộc tính có tên là name.
- public void removeAttribute(String name) throws IllegalStateException: Loại bỏ thuộc tính có tên là name.

Để sử dụng Session, ta thực hiện như sau:

- Nhận về một đối tượng (của HttpSession) cho người sử dụng bằng cách gọi phương thức getSession() đối với đối tượng của lớp HttpServletRequest.

```
HttpSession userSession = request.getSession();
```

- Lưu trữ và nhận dữ liệu từ Session. Giao diện cung cấp những phương thức để đặt lại giá trị cho các thuộc tính hoặc xác định chúng như đã nêu ở trên.

Ví dụ 7.1 Servlet thực hiện lưu trữ và đọc dữ liệu từ một Session.

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class SessionServlet extends HttpServlet {


    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body bgcolor=\"white\">");
        out.println("<head>");

        String title = "Session Example";
        out.println("<title>" + title + "</title>");
        out.println("</head>");
        out.println("<body>");

        // img stuff not req'd for source code html showing
     // relative links everywhere!

        // XXX
        // making these absolute till we work out the
        // addition of a PathInfo issue

        out.println("<a href=\"../sessions.html\">");
         out.println("<img src=\"../images/code.gif\" height=24 "
+
                        "width=24 align=right border=0 alt=\"view
code\"></a>");
        out.println("<a href=\"../index.html\">");
         out.println("<img src=\"../images/return.gif\" height=24
" +
                                  "width=24  align=right  border=0
alt=\"return\"></a>");
```

```
        out.println("<h3>" + title + "</h3>");

        HttpSession session = request.getSession(true);
        out.println("Session ID:" + " " + session.getId());
        out.println("<br>");
        out.println("Created:" + " ");
              out.println(new  Date(session.getCreationTime())  +
"<br>");
        out.println("Last Accessed:" + " ");
        out.println(new Date(session.getLastAccessedTime()));

        String dataName = request.getParameter("dataname");
        String dataValue = request.getParameter("datavalue");
        if (dataName != null && dataValue != null) {
            session.setAttribute(dataName, dataValue);
        }

        out.println("<P>");
          out.println("The following data is in your session:" +
"<br>");
        Enumeration names = session.getAttributeNames();
        while (names.hasMoreElements()) {
            String name = (String) names.nextElement();
            String value = session.getAttribute(name).toString();
            out.println(HTMLFilter.filter(name) + " = "
                        + HTMLFilter.filter(value) + "<br>");
        }

        out.println("<P>");
        out.print("<form action=\"");
     out.print(response.encodeURL("SessionServlet"));
        out.print("\" ");
        out.println("method=POST>");
        out.println("Name of Session Attribute:");
        out.println("<input type=text size=20 name=dataname>");
        out.println("<br>");
        out.println("Value of Session Attribute:");
        out.println("<input type=text size=20 name=datavalue>");
        out.println("<br>");
        out.println("<input type=submit>");
        out.println("</form>");

        out.println("<P>GET based form:<br>");
        out.print("<form action=\"");
     out.print(response.encodeURL("SessionServlet"));
        out.print("\" ");
```

```
        out.println("method=GET>");
        out.println("Name of Session Attribute:");
        out.println("<input type=text size=20 name=dataname>");
        out.println("<br>");
        out.println("Value of Session Attribute:");
        out.println("<input type=text size=20 name=datavalue>");
        out.println("<br>");
        out.println("<input type=submit>");
        out.println("</form>");

        out.print("<p><a href=\"");
     out.print(response.encodeURL("SessionExample?
dataname=foo&datavalue=bar"));
     out.println("\" >URL encoded </a>");

        out.println("</body>");
        out.println("</html>");

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

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

Lớp HTMLFilter

```
public final class HTMLFilter {


    /**
      * Filter the specified message string for characters that
are sensitive
        * in HTML.   This avoids potential attacks caused by
including JavaScript
      * codes in the request URL that is often reported in error
messages.
     *
     * @param message The message string to be filtered
     */
    public static String filter(String message) {
```

```
        if (message == null)
            return (null);

        char content[] = new char[message.length()];
        message.getChars(0, message.length(), content, 0);
         StringBuffer result = new StringBuffer(content.length +
50);
        for (int i = 0; i < content.length; i++) {
            switch (content[i]) {
            case '<':
                result.append("&lt;");
                break;
            case '>':
                result.append("&gt;");
                break;
            case '&':
                result.append("&amp;");
                break;
            case '"':
                result.append("&quot;");
                break;
            default:
                result.append(content[i]);
            }
        }
        return (result.toString());

    }


}
```

8. Sự truyền thông giữa các Servlet

Các Servlet có nhiều cách trao đổi tin với nhau. Chúng cần trao đổi với nhau vì:

- Một Servlet có thể truy cập trực tiếp đến các Servlet được nạp về và cần chúng thực hiện một số công việc nào đó. Một Servlet sử dụng đối tượng của ServletContext để nhận tin từ các Servlet khác.

  ```
  public Servlet ServletContext.getServlet(String name)
              throws ServletException
  ```

- Một Servlet có thể sử dụng lại những khả năng của Servlet khác để thực hiện một nhiệm vụ nào đó.
- Hai hoặc nhiều Servlet cần chia sẻ thông tin với nhau.

Ta có thể sử dụng phương thức getServlets():

```
public Enumeration ServletContext.getServlets()
```

để lấy về các đối tượng Servlet được nạp về từ ngữ cảnh Servlet hiện thời ServletContext.

Ví dụ Chương trình sau mô tả sự trao đổi tin giữa các Servlet với nhau.

// InterServlet.java

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class InterServlet extends HttpServlet{
     public void doGet(HttpServletRequest re, HttpServletResponse res)
               throws IOException{
          res.setContentType("text/plain");
          PrintWriter out = res.getWriter();
          ServletContext ct = getServletContext();
          Enumeration num = ct.getServletNames();
          out.println("<HTML>");
          out.println("<HEAD><TITLE>Demo Servlet </TITLE></EAD>");
          out.println("<h3>!!Warning </h3>");
          out.println("getServletNames() ----");
          try{
            while(num.hasMoreElement()){
               String nm = (String)num.nextElement();
               out.println(nm);
               Servlet s = ct.getServlet(nm);
               out.println("Servlet Name: " + nm);
               out.println("Servlet classe: " + s.getClass().getName());
               out.println("Servlet Information: " + s.getServletInfo());
               out.println();
             }
          }catch(Exception e){
               System.out.println("Error: " + e);
            }
     }
}
```

9. Servlet kết nối các cơ sở dữ liệu

Các WebSite ngày nay không chỉ hiển thị những thông tin của các trang HTML tĩnh. Ví dụ, trong thương mại điện tử, một lĩnh vực đang thịnh hành trên thế giới, khách hàng có thể vào một trang Web bán hàng, chọn những mặt hàng cần mua, điền vào phiếu mua hàng sau đó thanh toán (bằng các phương thức trả tín phiếu, check, chuyển

khoản, v.v.) thì sẽ nhận được các mặt hàng theo yêu cầu. Như vậy, các thông tin chi tiết về khách hàng phải được lưu trữ trong CSDL ở máy dịch vụ để thực hiện được các dịch vụ thương mại điện tử. Trong những kịch bản như thế, các Website phải được kết nối với các CSDL khác.

Servlet và JDBC là hai công cụ rất hiệu quả cho người lập trình ứng dụng Web kết nối Web với CSDL.

9.1 Vai trò của Servlet trong mô hình kết nối CSDL

Các chương trình ứng dụng trên mạng hiện nay đang chuyển từ mô hình Client/Server sang *mô hình ba tầng*, hoặc n-tầng. Trong mô hình ba tầng, Client không thực hiện truy vấn trực tiếp các CSDL mà thông qua tầng trung gian ở Server để truy vấn vào các CSDL.

Hình 6.5 Mô hình chương trình ứng dụng ba tầng

Mô hình ba tầng có ưu điểm là nó tách riêng phần biểu diễn ảo (ở phía Client) ra khỏi phần logic nghiệp vụ (tầng trung gian) và dữ liệu thô ở Server. Do vậy, nó cho phép nhiều Client có thể truy cập vào cùng một dữ liệu, cùng một nghiệp vụ và nhiều loại chương trình Java thực hiện như chương trình ứng dụng độc lập, chương trình Applet hay chương trình Web. Servlet có một vị trí quan trọng trong việc truy cập vào CSDL ở tầng trung gian. Ví dụ, hãy tưởng tượng có một chương trình đặt mua hàng trên mạng. Nếu không có tầng trung gian thì chương trình này phải kết nối trực tiếp với CSDL trên máy chủ và máy chủ phải ghi lại đơn hàng, cập nhật lại các trường dữ liệu trong CSDL (trừ đi những mặt hàng đã bán theo đơn đặt mua). Chương trình của khách có thể bị ngắt nếu Server của CSDL bị thay đổi theo một cách nào đó. Hoặc có thể một ai đó can thiệp vào chương trình của khách, Server nhận được đơn đặt hàng nhưng không nhận được sự thanh toán của khách, mặc dù khách hàng đã thực hiện thanh toán theo đơn đặt hàng, v.v.

Tầng trung gian sử dụng logic nghiệp vụ để trừu tượng hoá quá trình xử lý đặt hàng. Nó nhận thông tin từ đơn đặt hàng, kiểm tra tính xác thực của thẻ tín dụng, tài khoản, v.v, và chuyển những thông tin đó cho hệ CSDL. Khi hệ CSDL thay đổi thì tầng trung gian sẽ được cập nhật mà không làm ảnh hưởng đến chương trình của khách. Ngoài ra, tầng trung gian còn giúp ta tăng hiệu quả xử lý công việc vì nó hỗ trợ để kết nối chéo với nhiều hệ CSDL khác nhau.

9.2 Xử lý giao dịch với JDBC

Sự truyền thông giữa Client và tầng trung gian thường sử dụng HTTP (khi người sử dụng dùng Web Browser), RMI (khi người sử dụng dùng phần mềm ứng dụng hay

Applet triệu gọi từ xa như đã đề cập ở chương II). Bộ điều khiển kết nối JDBC được sử dụng để quản lý sự trao đổi thông tin giữa tầng trung gian và CSDL.

JDBC là giao diện với SQL, một giao diện với hầu như tất cả các CSDL mô hình dữ liệu quan hệ hiện đại.

Ta hãy xét tiếp chương trình đặt mua hàng trực tuyến. Khách hàng điền vào đơn đặt mua và gửi đến cho Cửa hàng (hệ thống bán hàng trực tuyến). CSDL của hệ thống phải được cập nhật và chèn thêm bản ghi thông tin về khách hàng đó. Tương tự, sau khi giao dịch mua/bán kết thúc thì một số bảng trong hệ CSDL cũng sẽ phải được cập nhật.

Những vấn đề trên được các câu lệnh của SQL thực hiện. Đối tượng của lớp Connection được sử dụng để quản lý các giao dịch với JDBC.

Ví dụ Ta hãy xây dựng một chương trình ứng dụng thời gian thực có sử dụng Servlet. Để thực hiện ví dụ này, mặc định là ta đã biết sử dụng HTML và Microsoft FrontPage để tạo lập các trang Web.

Chúng ta hãy xét bài toán như sau: Sao Mai là công ty cho bán và cho thuê ô tô. Họ bán, cho thuê các ô tô gia đình và các ô tô chở khách cho các tua du lịch. Công ty muốn thành lập một câu lạc bộ cùng với một Website: CarSaoMai.com để thực hiện các dịch vụ trực tuyến nêu trên.

Để tham gia được vào CarSaoMai.com thì bạn phải điền vào một thẻ đăng ký gia nhập câu lạc bộ và chỉ những người đã đăng ký mới được cung cấp các dịch vụ trực tuyến để mua và sử dụng ô tô. Những người chưa đăng ký thì chỉ được quyền truy cập để biết được những thông tin về Công ty và những thông tin tóm tắt về các dịch vụ mà Công ty cung cấp.

Trước tiên, ta hãy thiết kế quá trình đăng ký như sau.

- Tạo ra trang HTML chứa mẫu đăng ký.
- Cập nhật lại CSDL về các Servlet làm việc ở tầng trung gian khi có một người mới đăng ký.
- Tạo ra một CSDL chính để lưu trữ thông tin về các thành viên của câu lạc bộ.
- Tạo ra Servlet khác để cho phép tất cả các thành viên câu lạc bộ kết nối được vào Website và được phục vụ.

1. Thiết lập CSDL

Đầu tiên, ta có thể sử dụng Microsoft Access để tạo ra một CSDl nhỏ, ví dụ carsSaoMai.mdb. Để sử dụng CSDL này, ta sử dụng 32-bit ODBC (chọn trong Control Panel\Administrative Tools\Data Sources (ODBC)). Sau khi mở được hộp thoại ODBC Data Sources Administrator, hãy chọn System DSN và

nhấn vào nút Add để đưa thêm CSDL mới vào nguồn dữ liệu. Nhập tên CSDL carsSaoMai vào trường Data Source Name và nhấn vào Selection để thiết lập đường dẫn tới thư mục chứa tệp carsSaoMai.mdb.

carsSaoMai.mdb có hai bảng được thiết kế như sau.

(i) Bảng Login

Tên trường	Kiểu dữ liệu	Các ràng buộc
LogName	Text(20)	Khoá nguyên thuỷ
Passwd	Text(20)	Không rỗng

(ii) Bảng Memebers: lưu trữ các thành viên câu lạc bộ

Tên trường	Kiểu dữ liệu	Các ràng buộc
LogName	Text(20)	Khoá tham chiếu
FName	Text(30)	Không rỗng
LName	Text(30)	Không rỗng
Age	Number(2)	Không rỗng
Gender	Text(10)	Không rỗng
Address	Text(30)	Không rỗng
City	Text(30)	Không rỗng
PIN	Number(7)	Không rỗng
EMail	Text(30)	Không rỗng
Phone	Number(10)	Không rỗng
Salary	Number(20)	Không rỗng

2. Tạo lập trang chủ và mẫu đăng ký câu lạc bộ

Ta có thể tạo ra các trang HTML cho trang chủ và các mẫu đăng ký đã được thiết kế

Lưu lại trang Web trên vào tệp carsSaoMai_HomePage.shtm. Tệp này có thêm dòng lệnh:

```
<servlet code = http://128.28.10.1:8080/servlet/pageValidator.class>
</servlet>
```

Servlet pageValidator.class cùng với một phần mã HTML sẽ tạo ra Servlet ở phía máy chủ. Lưu ý là cần phải thay đổi địa chỉ URL tới URL của Web Server của bạn.

Tệp carsSaoMai_HomePage.shtm có các điểm liên kết tới các trang HTML bao gồm:

- SaoMai.html: trang chứa các thông tin về câu lạc bộ.
- member.html: ghi nhận các thành viên đăng ký vào câu lạc bộ
- getRegisteredNow.html và registrationContD.html: cho phép các thành viên mới đăng ký vào câu lạc bộ.

3. Viết các Servlet cho tầng trung gian

carsSaoMai.com sử dụng các Servlet để nhận được các thông tin từ Client Browser, thẩm định thông tin này, tương tác với CSDL để chèn các bản ghi mới, tìm trong CSDL những thành viên đã đăng ký và thực hiện các dịch vụ mà các thành viên câu lạc bộ yêu cầu. Đó là các Servlet:

- NewRegistry và MamberDetailEntry: ghi nhận những thành viên mới.
- RegisteredMemeber: tìm trong CSDL những thành viên đã đăng ký và truy cập để xác định những thông tin cần thiết.

4. Dịch và thực hiện chương trình ứng dụng

Tóm lại, việc tạo lập một trang Web (diễn đàn) đáp ứng các yêu cầu trên, ta thực hiện các bước như sau:

- Tạo ra một CSDL carsSaoMai.mdb, và để truy cập được vào nó thì phải cho biết tên và mật khẩu của người đã đăng ký tham gia Câu lạc bộ. CSDL này có hai bảng Login và Memebers.
- Chuyển tất cả các tệp HTML mà Website yêu cầu vào thư mục, ví dụ public_html của JavaWebServer. Đồng thời chuyển các tệp ảnh vào thư mục này. Nhớ là phải thay đổi URL cho thích hợp như đã nêu ở trên.
- Tiếp theo, chuyển các tệp lớp đã được dịch (.class) của các chương trình Servlet để nạp vào thư mục Servlet của JavaWebServer. Ở đây cũng cần nhớ là phải thay đổi URL đã chỉ ra trong các tệp .java trước khi dịch và chuyển chúng về thư mục \JavaWebServer2.0\servlets.
- Khởi động Java Web Server bằng cách thực hiện httpd.exe ở thư mục Web server\bin.
- Khi Web Server đã hành động, hãy mở trình duyệt của bạn và gõ vào

  http://localhost:8080/SaoMai_homepage.shtml

- Nếu trang Web này chưa được nạp về thì thay localhost trong URL ở trên bằng địa chỉ IP máy của bạn, ví dụ:

  http://128.28.10.1:8080/SaoMai_homepage.shtml

Tương tự như trên nếu ta muốn thực hiện trên Web Server.