Chương 6
PHÁT TRIỂN CÁC DỊCH VỤ SERVLET VÀ JSP

Chương VI giới thiệu mô hình dịch vụ Web từ phía máy chủ:

- ✓ Giới thiệu về Servlet
- ✓ Chu trình sống của Servlet
- ✓ Sử dụng Servlet để truy tìm thông tin
- ✓ Sử dụng Servlet để gửi tin
- ✓ Sử dụng Servlet để truy cập vào CSDL
- ✓ JSP (Java Server Page).

6.1. GIỚI THIỆU VỀ SERVLET

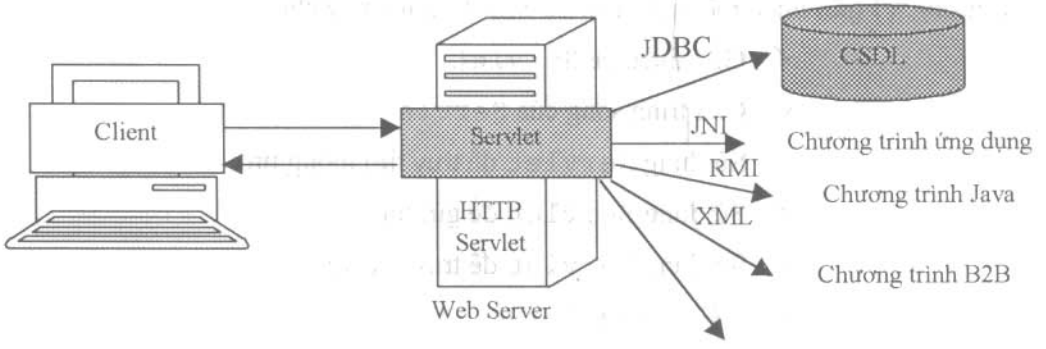
Hiện nay, trong lập trình có một xu hướng rất quan trọng đang được tập trung phát triển ứng dụng, đó là xây dựng các chương trình dịch vụ Java ở phía máy chủ (Server) ([2], [4]).

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 6.1, 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 6.1.* 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, khi cần bổ sung các chức năng mới 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 tồn tại 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 JavaBean (EJB), ... Vấn đề bảo mật thông tin sẽ được đề cập chi tiết ở chương 7.

6.2. 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 số học của Java giúp cho Servlet thoát khỏi nhiều công việc quản lý bộ nhớ phức tạp. 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.

6.3. MÔI TRƯỜNG THỰC HIỆN SERVLET

Các Servlet thường là sự mở rộng (kế thừa) các lớp chuẩn của 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) 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 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 có 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.

6.3.1. 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.

6.3.2. 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/.

6.3.3. 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.

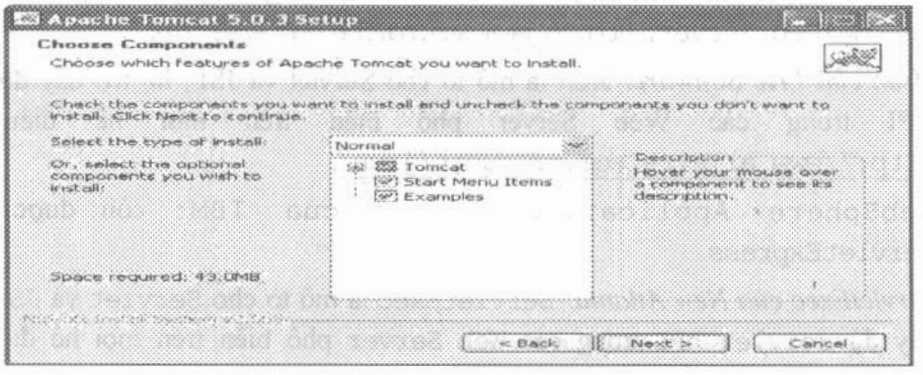
6.3.4. Cài đặt trình chủ Apache Tomcat

Để thực hiện được Servlet ta cần cài đặt một trong các mô tơ Servlet đã nêu. Một trong số đó được sử dụng khá phổ biến là Tomcat. Tomcat, tên một dự án con trong dự án Jakarta được gọi là một Servlet Container, tức là một mô tơ thực thi và triển khai công nghệ Servlet của Sun. Tất nhiên, Tomcat còn chứa một môi trường triển khai cho cả công nghệ JSP. Nói cách khác, Tomcat chính là chương trình dịch và thực thi các tệp .class và .jsp vốn được viết bằng ngôn ngữ Java để tạo ra các trang Web động.

Tomcat có thể xem như là một Web Server đơn giản với các tính năng cơ bản giống như Apache. Nhưng, trong khi Apache không hiểu gì về các trang Web động của công nghệ Java thì Tomcat lại là một phần mềm được xây dựng để chuyên xử lý các chương trình Web làm bằng công nghệ này.

Đĩa cài Tomcat có thể có sẵn ở các hiệu đĩa CD, nhưng để có phiên bản mới nhất hỗ trợ đặc tả Servlet và JSP hiện đại nhất và nhiều tính năng nhất thì tốt nhất là tải trực tiếp trên trang chủ về phiên bản Tomcat 5 (kích thước khoảng 4.5MB). Thông tin trực tuyến về Tomcat có thể tìm tại http://jakarta.apache.org/tomcat/

Hiện tại, Tomcat đã có tới phiên bản 5.5.9 (jakarta-tomcat-5.5.9.exe). Trong tài liệu này chúng ta sẽ dùng phiên bản Tomcat 5.5.9 để cài đặt chương trình ứng dụng. Khi Download bản cài đặt về và chạy chương trình sẽ xuất hiện giao diện sau:



Hình 6.2. **Màn hình giao diện cài đặt Tomcat**

Ở đây, chúng ta phải lựa chọn các thành phần cài đặt cho Tomcat. Nếu mới làm quen với Tomcat thì hãy sử dụng tùy chọn mặc định của Tomcat bằng cách kích vào Next. Thường thì không nên cài đặt Tomcat với một đường dẫn có khoảng trống. Chúng ta hãy lựa chọn một thư mục cài đặt Tomcat khác với mặc định, ví dụ: C:\Tomcat5.

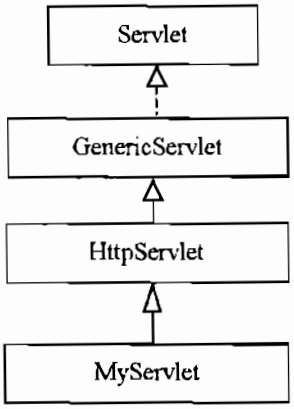
Cũng cần phải chú ý thêm: cổng mặc định của Tomcat là 8080, và ta có thể thay đổi được. Số cổng này trùng với số cổng của Apache 2.0.x nếu như ta đã cài Apache 2 và cho

nó chạy dưới hình thức Manually, tức là khởi động thủ công thay vì cho nó tự động hành động như là một dịch vụ (Service). Khi đó, sẽ có sự xung đột về cổng giữa hai ứng dụng này. Vì thế, ta phải thay đổi số cổng của một trong hai ứng dụng. Ví dụ, với Tomcat ta vẫn để là 8080 nhưng Apache 2 thì chuyển thành 8081, hoặc ngược lại.

Để kiểm tra quá trình cài đặt có thành công hay không, sau khi kích đúp vào tomcat_install_dir\tomcat5.5\bin\tomcat5.exe để khởi động Server, hay có thể truy cập vào địa chỉ http://localhost:8080/, nếu có một trang Web hiện ra thì quá trình cài đặt Tomcat hoàn tất, ngược lại thì phải xem lại các bước đã thực hiện.

6.4. 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**

6.4.1. 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.

6.4.2. 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 gói `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 của HttpServletRequest cung cấp thông tin giống như biến môi trường của CGI nhưng theo một hướng chuẩn hóa. 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 6.1. 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 Cookie). Điều quan trọng nhất là phải nhận được một đối tượng của PrintWriter (ở trong gói java.io) thông qua resp.getWriter() để gửi kết quả trả lại cho Client. Ngoài doGet() và 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.

6.5. DỊCH VỤ CÀI ĐẶT SERVLET

Trước tiên phải lưu ý rằng việc cài đặt chi tiết các Servlet là thay đổi tuỳ theo từng Web Server khác nhau. Ví dụ, khi ta sử dụng Java Web Server (JWS) 2.0 thì các Servlet đòi hỏi phải được đặt ở thư mục được gọi là servlets trong cây thư mục cài đặt của JWS. Tuy nhiên, ta có thể tạo ra một thư mục riêng, ví dụ myservlet và đặt các Servlet vào đó để tránh xung đột với các Servlet khác.

6.5.1. Dịch chương trình Servlet

Dịch các Servlet có thể sử dụng JDK http://www.javasoft.com hoặc các chương trình biên dịch Java khác và thực hiện hoàn toàn giống như đối với các Applet hay

chương trình ứng dụng độc lập. Nếu các gói javax.servlet và javax.servlet.http không có trong chương trình biên dịch và cũng không có trong Servlet Engine thì phải cài đặt chúng riêng bằng cách nạp từ JSDK (Java Servlet Development Kit) và đặt đường dẫn chứa các lớp của JSDK trong biến môi trường CLASSPATH cho phù hợp.

Như vậy, để chạy được các chương trình Servlet như trên, ta có thể thực hiện các bước như sau:

- Cài đặt JSDK (Java Servlet Development Kits)
- Cài đặt một trong các mô tơ Servlet như đã nêu ở mục 6.3
- Dịch chương trình Servlet.

Có hai cách chính để dịch các chương trình Servlet trong các gói.

+ *Cách thứ nhất là đặt* CLASSPATH để chỉ tới thư mục mà ở đó có chứa các Servlet của chúng ta. Ví dụ, JSDK cài đặt ở c:\jsdk2.0, và gói của ta có tên là myservlet (thư mục d:\myservlet) chứa các Servlet cần dịch, ở Window ta thực hiện như sau:

```
DOS> set  CLASSPATH=.;C:\jsdk2.0\lib\jsdk.jar;%CLASSPATH%
DOS> cd  d:\myservlet
DOS> javac FirstServlet.java
```

+ *Cách thứ hai là* dịch các lớp được đặt trong các gói được chỉ rõ trong câu lệnh dịch. Ví dụ, nếu thư mục cơ sở cài đặt JSDK là c:\jsdk2.0 và gói của ta có tên là myservlet (thư mục d:\myservlet) chứa các Servlet cần dịch, ở Window ta thực hiện như sau:

```
DOS> set CLASSPATH =.;C:\jsdk2.0\lib\jsdk.jar;%CLASSPATH%
DOS> javac myservlet\FirstServlet.java
```

Hiển nhiên là trước đó ta phải đặt biến môi trường PATH chỉ tới nơi cài đặt chương trình dịch javac.exe. Giả sử J2DK1.4.2 được cài đặt ở c:\j2sdk1.4.2, thì biến môi trường PATH được đặt như sau

```
DOS> set PATH=.;C:\j2sdk1.4.2\bin
```

Để tiện lợi cho việc dịch các Servlet, ta có thể đặt các biến môi trường như trên vào trong autoexec.bat.

6.5.2. Thực hiện Servlet

1. Với Java Web Server, các Servlet được đặt ở thư mục servlets bên trong cây thư mục cài đặt JWS và để chạy, chúng ta triệu gọi http://host/servlet/ServletName sau khi đã khởi động Server.

Chú ý: Thư mục servlets là số nhiều còn địa chỉ URL gọi tới là servlet, số ít. Nếu Servlet của chúng ta là FirstServlet đặt ở gói myservlet thì phải triệu gọi:

http://host/servlet/myservlet/FirstServlet

Những Web Server khác có thể qui định cách cài đặt Servlet khác nhau. Phần lớn chúng cho phép sử dụng cách đặt biệt danh (aliase) cho các Servlet sao cho có thể triệu gọi một tệp HTML bất kỳ (any-file) ở một thư mục bất kỳ (any-path):

http://host/any-path/any-file.html

2. Với Tomcat 5.5, chúng ta cũng thực hiện tương tự như trên.

- Đặt các chương trình Java Servlet vào một thư mục, ví dụ

  c:\Servlets + JSP

- Khởi động DOS và viết "javac FirstServlet"
- Đặt tệp lớp FirstServlet.class vào thư mục của Servlet, ví dụ

  C:\Tomcat5.5\webapps\ROOT\WEB-INF\classes

- Khởi động Server: kích đúp vào startup.bat hoặc biểu tượng tomcat5.exe.
- Triệu gọi Servlet

  http://localhost/servlet/FirstServlet

  Hoặc

  http://localhost:8080/FirstServlet

  nếu Tomcat được cài đặt ở cổng 8080.

Ví dụ 6.2. Xây dựng một Servlet đơn giản để sinh ra HTML và hiển thị lời chào "Xin chào các bạn" mỗi khi thực hiện.

Phần lớn các Servlet sinh ra các kết quả dạng HTML. Để làm được điều này, chúng ta cần thực hiện theo hai bước:

- Chỉ cho trình duyệt biết là ta sẽ gửi thông tin lại bằng HTML. Điều này được thực hiện bằng cách đặt kiểu nội dung là

  res.setContentType("text/html");

- Thay đổi lệnh println() để tạo ra một trang Web hợp lệ với các đối số là các thẻ HTML.

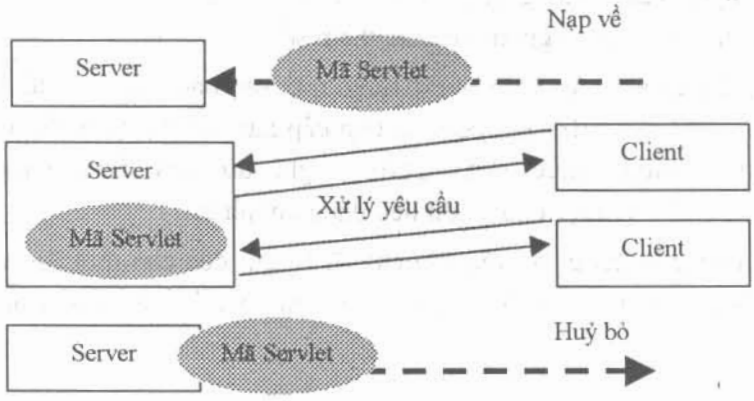
```
package myservlet;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletExample extends HttpServlet{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException{
            res.setContentType("text/html");
```

```
            PrintWriter out = res.getWriter();
            out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
                                    HTML "Transitional//EN\">\n" +
                  "<HTML>\n" +
                  "<HEAD><TITLE>Mời chào</TITLE></HEAD>\n" +
                  "<BODY>\n" +
                  "<H1> Xin chào các bạn</H1>\n" +
                  "</BODY></HTML>");
    }
}
```

6.6. 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.



Hình 6.4. **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).

6.6.1. 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.

6.6.2. 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().

6.6.3. Hủy 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.

6.7. XỬ LÝ CÁC YÊU CẦU

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

6.7.1. 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 6.3. 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));
        }
    }
}
```

6.7.2. 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.
- *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 6.4. 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.setHeader("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 getHeader() 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 chấp nhận.
- 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).

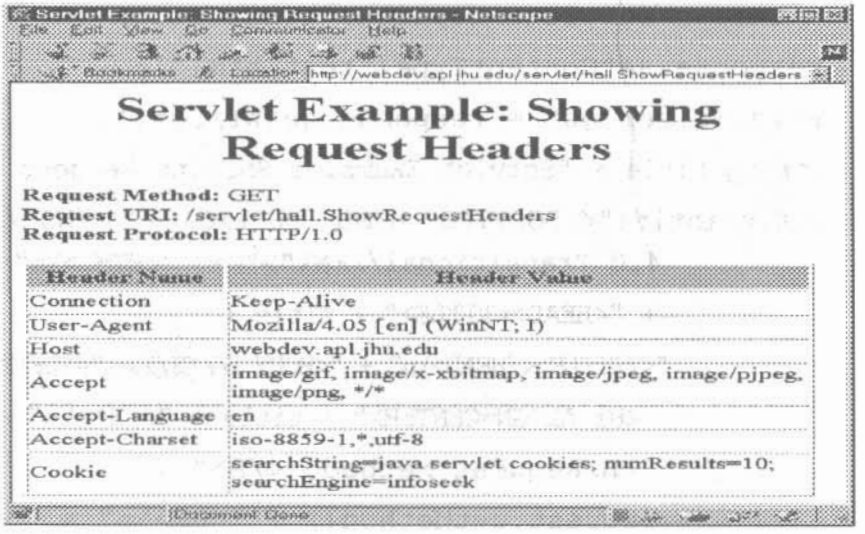
Ví dụ 6.5. Xây dựng một Servlet để đọc, hiển thị một bảng các tiêu đề và nội dung của một yêu cầu.

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

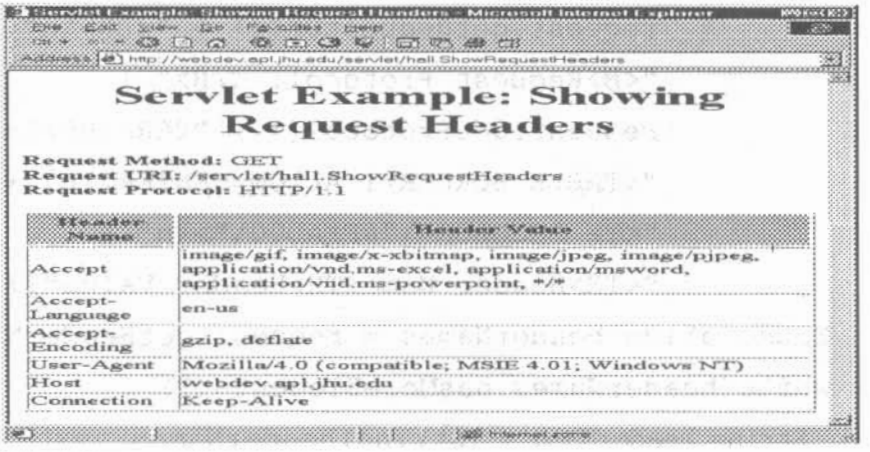
```
import javax.servlet.http.*;
import java.util.*;

/* 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 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**

6.7.3. 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 là 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 6.6. 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();
```

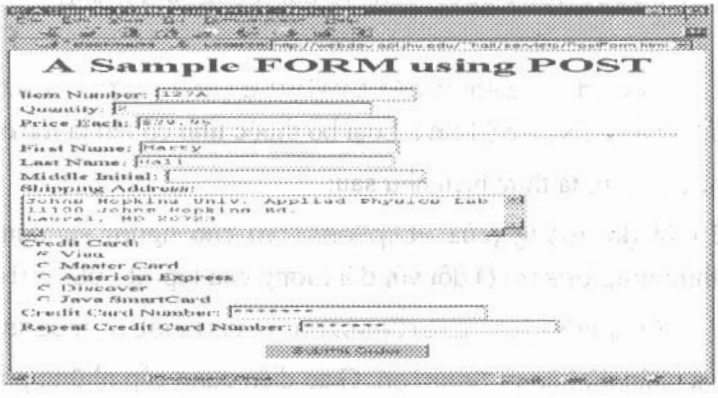
```
        String title = "Reading All Request Parameters";
            out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0
                Transitional//EN\">" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<TABLE 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().

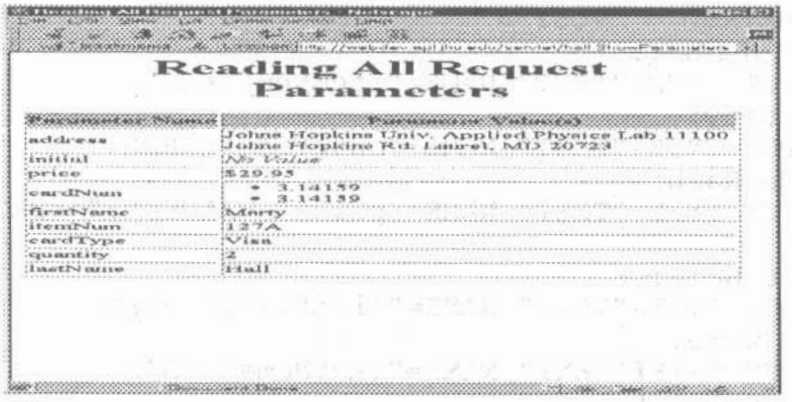
Tệp PostForm.html

```
<!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**

6.8. 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 6.6. Servlet thực hiện lưu trữ và đọc dữ liệu từ một Session.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SessionServlet extends HttpServlet{
    public void doGet(HttpServletRequest re, HttpServletResponse res)
            throws ServletException, IOExceptio{
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        HttpSession session = re.getSession(true);
        // In các thông tin về Session
        Date dateCreated = new Date(session.getCreationTime());
        Date dateAccessed = new Date(session.getLastAccessedTime());
        out.println("ID: " + session.getID());
        out.println("Created: " + dateCreated);
        out.println("Last Accessed: " + dateAccessed);
        // Đặt lại các thông tin cần thiết
        String dataName = re.getParameter("dataName");
        if(dataName != null && dataName.length() > 0){
          String dataValue = re.getParameter("dataValue");
          session.setAttribute(dataName, dataValue);
        }
        // In nội dung nhận được từ Session
        Enumeration e = session.getAttributeName();
        while(e.hasMoreElement()){
          String name = (String)e.nextElement();
          String value = session.getAttribute(name).toString();
          out.println(name + " = " + value);
        }
    }
}
```

6.9. 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 6.7. 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 IOExceptio{
      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);
            }
    }
}
```

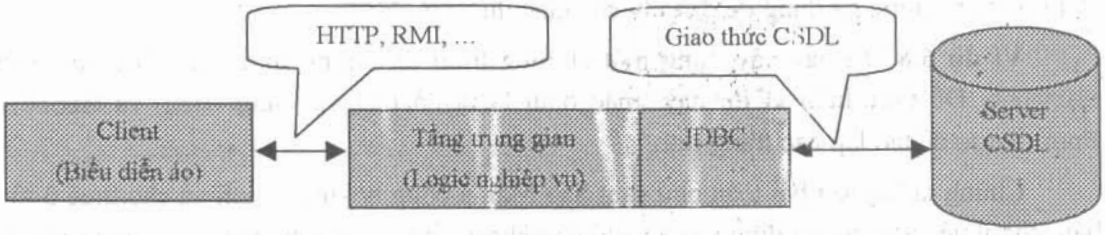
6.10. SERVLET KẾT NỐI CÁC CƠ SỞ DỮ LIỆU

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 [1] là hai công cụ rất hiệu quả cho người lập trình ứng dụng Web kết nối Web với CSDL.

6.10.1. 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.

6.10.2. 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 6.8. 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 kinh doanh 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 dong 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, được 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 hoạt 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 của máy bạn, ví dụ:

http://128.28.10.1:8080/SaoMai_homepage.html

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

6.11. JSP

Java Server Pages (JSP) cho phép tách riêng phần động của những trang Web ra khỏi HTML tĩnh. JSP không chỉ hỗ trợ để tạo ra những trang Web độc lập với các nền, độc lập với các Server, mà còn là công nghệ Java rất hiệu quả để thể hiện nguyên lý WYSIWYG (Những gì bạn nhìn thấy là bạn có được chúng) trên các trang HTML tĩnh ([2], [4]).

Các trang JSP gồm có:

- Các thành phần tĩnh HTML/XML
- Các thẻ đặc biệt của JSP
- Có thể có một số các đoạn mã chương trình viết bằng Java, được gọi là những kịch bản nhỏ Scriptlet.

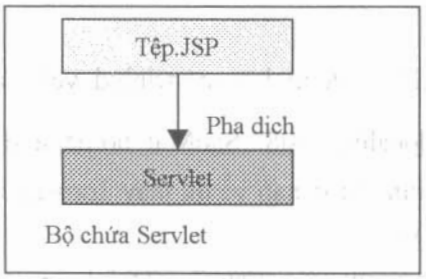
Do vậy, ta có thể tạo lập và duy trì các trang JSP bằng các công cụ truyền thống như đối với HTML/XML.

Một điều quan trọng cần lưu ý: đặc tả JSP là một chuẩn mở rộng, được định nghĩa trên đỉnh của Servlet API. Cả Servlet và JSP cùng có một số đặc tính chung, có thể sử dụng chúng phục vụ cho các nội dung Web động. Tuy nhiên, cũng có một số khác biệt giữa JSP và công nghệ Servlet như đã trình bày ở trên. JSP được sử dụng rộng rãi hơn. Nó không chỉ được sử dụng đối với các nhà phát triển hệ thống, mà cả những người thiết kế các trang Web, những người đóng vai trò rất quan trọng trong quá trình phát triển các trang Web ứng dụng hiện nay.

6.11.1. Kiến trúc của JSP

Mục đích của JSP là cung cấp các phương thức để khai báo, biểu diễn cho sự phát triển của các Servlet.

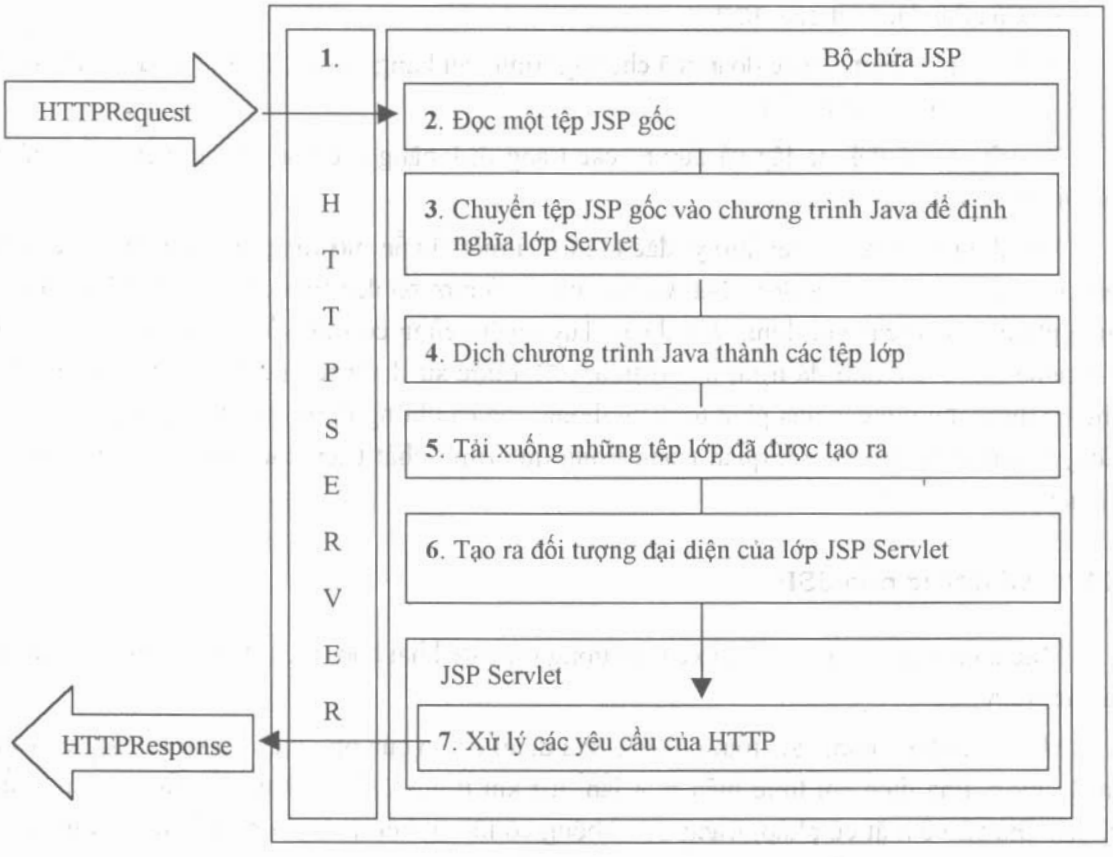
Một cách điển hình, các trang JSP là chủ điểm chính của pha dịch và xử lý các yêu cầu trên Server. Pha dịch chỉ thực hiện một lần, trừ khi trang JSP bị thay đổi thì nó mới phải dịch lại. Giả sử về mặt cú pháp, trang JSP không có lỗi thì kết quả của pha dịch sẽ là tệp lớp cài đặt trang JSP và giao diện Servlet như hình dưới đây.



Hình 6.6. **Cơ chế dịch của JSP**

Pha dịch được thực hiện bởi JSP Engine khi nó nhận được các yêu cầu lần đầu của một trang JSP. Lưu ý rằng, JSP 1.1 cho phép dịch trước (tiền dịch) các trang JSP thành các tệp lớp. Việc dịch trước này đặc biệt hữu dụng trong việc loại bỏ những khởi đầu chậm trễ khi một trang JSP được phân phát từ các nguồn để nhận được các yêu cầu của Client. Nguồn gốc của tệp lớp được tạo ra bởi Tomcat.

Như trên đã nêu, một chương trình JSP có thể chứa các đoạn chương trình Java và chúng kết hợp với nhau để tạo ra các JSP Servlet nhằm xử lý các yêu cầu của Client gửi đến thông qua HTTPRequest. JSP Servlet xử lý các yêu cầu của Client (qua HTTP) và gửi kết quả cho HTTPResponse.

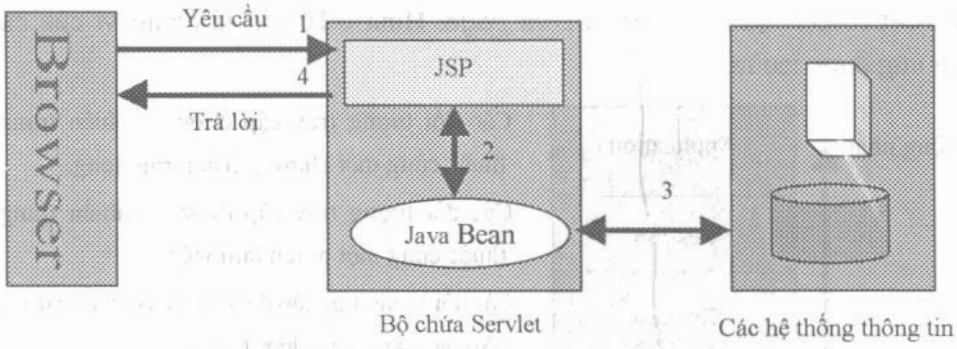


Hình 6.7. **Cơ chế hành động của bộ chứa JSP**

6.11.2. Các mô hình truy cập của JSP

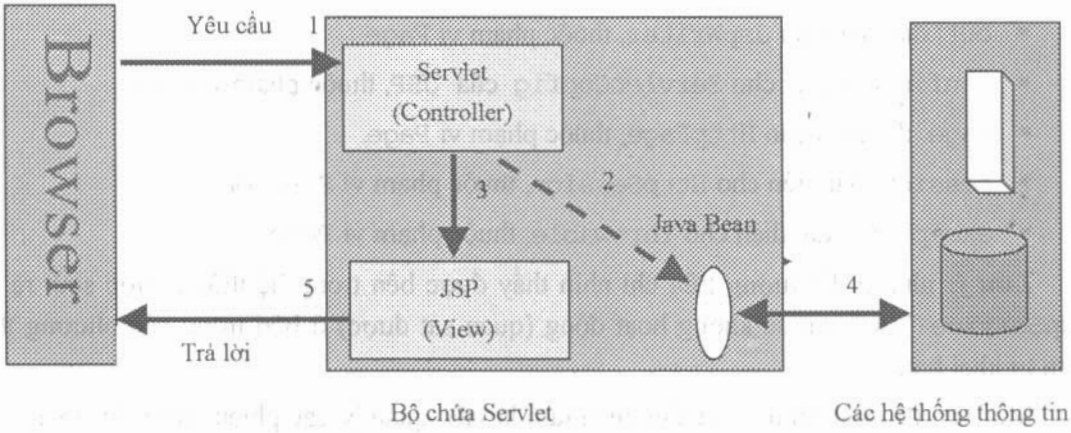
Có hai mô hình truy cập phổ biến để thực hiện truy cập thông tin theo công nghệ JSP.

1. *Mô hình 1*: các yêu cầu đầu vào từ Web Browser được gửi trực tiếp tới trang JSP, ở đó chúng được xử lý và trả lời cho Client. Trong mô hình này, mọi vấn đề truy cập đều được đảm nhận bởi các Java Bean.



Hình 6.8. **Mô hình 1**

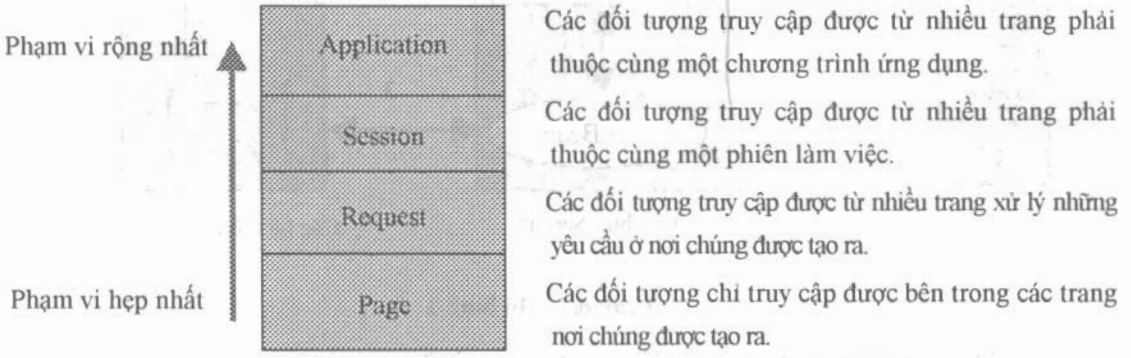
2. *Mô hình 2*: còn được biết đến với tên gọi MVC (Model/View/Controller). Việc xử lý thông tin được tách thành hai phần, thành phần biểu diễn và thành phần điều khiển. Thành phần biểu diễn là những trang JSP làm nhiệm vụ tạo ra câu trả lời dạng HTML/XML và xác định giao diện của người sử dụng mà Web Browser cung cấp. Thành phần điều khiển (còn được gọi là thành phần phía trước), không đảm nhận công việc biểu diễn mà là xử lý tất cả các yêu cầu của HTTP. Các thành phần điều khiển cũng đảm nhận việc tạo ra các Java Bean hoặc các đối tượng phục vụ cho các thành phần biểu diễn sử dụng, đồng thời quyết định các thành phần phụ thuộc vào hành động của người sử dụng.



Hình 6.9. **Mô hình 2**

6.11.3. Các phạm vi đối tượng

Trước khi tìm hiểu về cú pháp và ngữ nghĩa của JSP, điều quan trọng là phải hiểu rõ phạm vi hay khả năng quan sát được của các đối tượng Java xử lý các yêu cầu trong các trang JSP. Các đối tượng có thể được tạo ra một cách tường minh bằng cách sử dụng các thẻ chỉ dẫn JSP, hoặc một cách gián tiếp thông quan các thẻ hành động. Những đối tượng được sinh ra sẽ được gắn tương ứng với những phạm vi hoạt động nhất định. Có bốn phạm vi hoạt động: Application, Session, Request và Page. Hình 6.10 mô tả phạm vi của các đối tượng tương ứng được tạo ra.



Hình 6.10. **Phạn vi hoạt động của các đối tượng**

Để cho thuận tiện, bộ chứa JSP tạo ra một số đối tượng tiềm năng để sử dụng trong các kịch bản và các biểu thức mà không cần phải khởi tạo ra chúng. Có chín đối tượng không tường minh được tạo lập sẵn:

- request: đại diện cho HTTPRequest, thuộc phạm vi Request.
- response: đại diện cho HTTPResponse, thuộc phạm vi Page.
- pageContext: đại diện cho PageContext, thuộc phạm vi Page.
- application: đại diện cho ServletContext, thuộc phạm vi Application.
- out: đại diện cho JspWriter, thuộc phạm vi Page.
- config: đại diện cho ServletConfig của JSP, thuộc phạm vi Page.
- page: đại diện cho HttpPage, thuộc phạm vi Page.
- session: đại diện cho HttpSession, thuộc phạm vi Session.
- exception: đại diện cho Throwable, thuộc phạm vi Page.

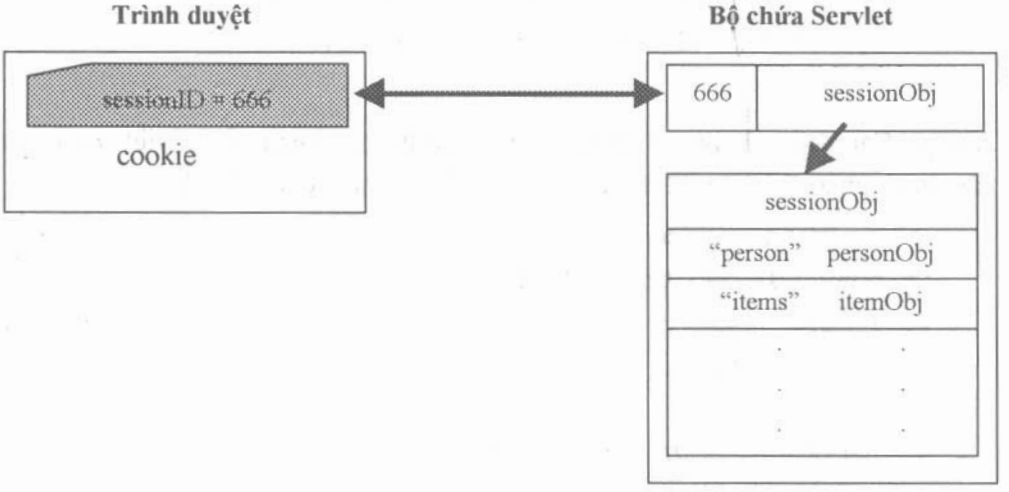
Lưu ý: những đối tượng trên chỉ nhìn thấy được bên trong hệ thống được sinh ra bởi _jspService(). Chúng sẽ không hoạt động (quan sát được) ở bên trong các phương thức mà ta tự khai báo.

Một vấn đề quan trọng nữa cần quan tâm là việc quản lý các phiên làm việc trong JSP. Theo mặc định, tất cả các trang JSP đều tham gia vào một phiên session của HTTP. Các phiên session là những nơi thích hợp để tổ chức lưu trữ các Bean và những đối tượng cần thiết sử dụng chung cho các trang JSP với các Servlet mà người sử dụng cần truy cập. Mỗi

session được xác định thông qua định danh ID và được lưu ở trình duyệt cùng với mẩu tin cookie.

Số lượng các đối tượng trong một phiên session là không giới hạn. Song khi số lượng chúng nhiều quá thì hiệu suất thực hiện sẽ kém đi. Theo mặc định, phần lớn các Server cho phép mỗi đối tượng tham gia vào một phiên session với thời gian tham gia là 30 phút. Nếu muốn thay đổi, bạn có thể đặt lại thời gian hoạt động của đối tượng trong một phiên session thông qua hàm setMaxInvalidationInterval(int secs). Cơ chế quản lý các phiên session được mô tả như hình 6.11.

Mô tơ JSP giữ tham chiếu tới đối tượng được đặt vào trong session cho đến khi hết thời gian có hiệu lực, sau đó những đối tượng đó bị huỷ bỏ để dọn dẹp bộ nhớ.



Hình 6.11. **Cơ chế quản lý các session**

6.11.4. Cơ sở cú pháp của JSP

Ngoài các qui ước của HTML, JSP có ba cấu trúc chính:

- Các phần tử kịch bản (Scripting Elements) cho phép xác định những đoạn mã chương trình Java sẽ trở thành bộ phận của Servlet,
- Các chỉ dẫn (Directives) cho phép điều khiển tất cả các cấu trúc của Servlet,
- Các hành động (Actions) cho phép chỉ ra những thành phần được sử dụng để xử lý và những thành phần còn lại để điều khiển trong mô tơ của JSP.

Lưu ý: Bạn có thể tìm hiểu chi tiết về JSP Tutorial ở địa chỉ:

http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/ .

1. Các phần tử kịch bản

Các phần tử kịch bản được sử dụng để chèn các đoạn mã chương trình Java vào trong một Servlet được sinh ra từ trang JSP hiện thời. Có ba thẻ chính:

(i) *Các thẻ biểu thức* dạng <%= exp %>, được sử dụng để tính giá trị của biểu thức exp và chèn kết quả (sau khi đã được chuyển về dạng xâu, theo các chỉ dẫn) vào trang kết quả. exp là một biểu thức trong Java được tính giá trị lúc thực hiện chương trình (khi trang tin được yêu cầu). Ví dụ, biểu thức JSP sau sẽ xác định ngày/giờ hiện thời của hệ thống khi trang tin yêu cầu:

```
Thời gian: <%= new java.util.Date() %>
```

(ii) *Các thẻ kịch bản* dạng <% Java code %>, được sử dụng để chèn các phương thức dịch vụ Java code vào trang JSP. Ví dụ:

```
<%
String queryData = request.getQueryString();
out.println("GET Data: " + queryData);
%>
```

Lưu ý: Các đoạn chương trình bên trong các thẻ kịch bản, được viết trước hoặc sau thành phần tĩnh của HTML tạo ra các kịch bản hành động. Ví dụ

```
<% if(Math.random() < 0.5) { %>
Là ngày <B> đẹp trời </B>!
<% } else { %>
Là ngày <B> nhiều mây </B>!
<% } %>
<% for(int i = 0; i <5; i++) {%>
    <H<%= i%>>Xin chào </H<%=i%>>
<%} %>
```

(iii) *Các thẻ khai báo* dạng <%! Java code %>, được sử dụng để chèn vào nội dung của các lớp Servlet, chèn vào bên ngoài của phương thức code. Các thẻ khai báo không sinh ra những kết quả, chúng được sử dụng kết hợp các thẻ biểu thức hoặc các kịch bản của JSP. Ví dụ, đoạn chương trình sau in ra số lần trang Web được truy cập sau khi nó được khởi tạo.

```
<%! private int accessCount = 0; %>
Số lần truy cập:
<%= ++accessCount %>
```

Lưu ý: Cũng giống như các kịch bản, nếu ta sử dụng những ký tự như "%>" thì phải viết thành "%\>".

2. Các chỉ dẫn JSP

Một chỉ dẫn trong JSP được sử dụng để gắn cấu trúc của lớp Servlet. Các thẻ chỉ dẫn JSP là những thông điệp dành cho mô tơ JSP. Thẻ chỉ dẫn được đặt ở đầu của tất cả các trang JSP. Có hai thẻ chỉ dẫn chính: thẻ page, include.

(i) *Chỉ dẫn JSP page.* Thẻ chỉ dẫn page cho phép bạn định nghĩa một trong các thuộc tính sau:

- `<%@ page import = "package.Class" %>`

  Ví dụ: `<%@ page import = "java.util.*" %>`

- `<%@ page contenType = "MINE-Type" %>` hoặc
- <%@ page contenType = "MINE-Type; charset=Character-Set" %>

  Trong đó, MINE là kiểu của kết quả. Ví dụ:

  `<%@ page contenType = "text/plain" %>`

- `<%@ page isThreadSafe = "true | false" %>`

  Một giá trị `true` (mặc định) chỉ ra rằng việc xử lý của `Servlet` là bình thường, trong đó có thể có nhiều yêu cầu được xử lý đồng thời bởi cùng một `Servlet`. Giá trị `false` chỉ ra rằng, `Servlet` cài đặt `SingleThreadModel` để thực hiện các yêu cầu một cách tuần tự hay được thực hiện bởi nhiều `Servlet` cùng một lúc.

- `<%@ page session = "true | false" %>`

  Một giá trị `true` (mặc định) chỉ ra rằng biến được định nghĩa trước `session` (thuộc kiểu `HttpSession`) sẽ được tìm thấy trong `session` nếu nó tồn tại. Giá trị `false` chỉ ra rằng, những `session` như thế không được sử dụng, mọi cố gắng truy cập vào `session` đều thất bại.

- `<%@ page buffer = "sizekb | none" %>`

  Thuộc tính này xác định cỡ của vùng đệm `buffer` được sử dụng bởi đối tượng `out` của `JspWriter`, cỡ nhỏ nhất phải là 8 KB.

- `<%@ page autoflush = "true | false" %>`

  Một giá trị `true` (mặc định) chỉ ra rằng, `buffer` sẽ được làm sạch khi nó bị đầy. Giá trị `false` , ít khi sử dụng, chỉ ra những ngoại lệ, khi `buffer` bị đầy.

- `<%@ page extends = "package.class" %>`

  Thuộc tính này chỉ ra lớp cha của `Servlet` được tạo ra.

- `<%@ page language = "java" %>`

  Thuộc tính này chỉ ra ngôn ngữ được sử dụng.

  `JSP` còn cung cấp một cơ chế để xử lý ngoại lệ (lỗi) khi thực thi chương trình bằng cách sử dụng thẻ page với thuộc tính `errorPage`:

`<%@ page isErrorPage = "false" errorPage = "errorHandler.jsp" %>`

thông báo cho mô tơ `JSP` biết rằng những ngoại lệ được mô tả trong `errorHandler.jsp`.

(ii) *Chỉ dẫn JSP include.* Thẻ chỉ dẫn include cho phép bạn đưa các tệp chương trình, hay tệp tin vào trong trang JSP lúc nó được dịch thành Servlet.

```
<%@ include file="relative url" %>
```

Trong đó, "url" là địa chỉ chứa những tệp cần phải được đưa vào trang JSP.

3. Thẻ chú thích

Thẻ chú thích của HTML có thể sử dụng trong các trang JSP. Tuy nhiên, người sử dụng vẫn có thể nhìn thấy những chú thích đó trong những trang gốc. Nếu bạn muốn những chú thích đó không nhìn thấy được bởi người sử dụng thì phải dùng thẻ chú thích của JSP.

```
<%-- Chú thích chỉ dành cho Server --%>
```

Những chú thích này chỉ có ý nghĩa giải thích dành cho người viết chương trình, sau khi dịch thành tệp lớp thì những người sử dụng không đọc chúng được nữa.

Ví dụ 6.9. Viết chương trình JSP để đọc dữ liệu được nhập vào từ dòng lệnh và hiển thị tên của máy tính chạy chương trình, số lần truy cập và ngày/giờ hiện thời của hệ thống.

```
<%-- jspTest.jsp --%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional //EN>
<HTML>
<HEAD>
<TITLE> Sử dụng Java Server Pages </TITLE>
<LINK REL=STYLESHEET
      HREF="My-Style-Sheet.css"
      TYPE="text/css">
</HEAD>
<BODY BGCOLOR= "#FDF5E6" TEXT="#000000" LINK="0000EE"
           VLINK="#551A8B" ALINK="FF0000">
<CENTER>
<TABLE BORDER=5 BGCOLOR="#EF8429">
      <TR><TH CLASS="TITLE">
           Sử dụng Java Server Pages </TABLE>
</CENTER>
<p>
      Những nội dung được tạo ra trong trang JSP
<UL>
      <LI><B>Thẻ biểu thức. </B><BR>
```

```
            Hostname: <%=request.getRemoteHost() %>.
        <LI><B>Thẻ kịch bản. </B><BR>
            <% out.println("Dữ liệu đọc được: " +
                request.getQueryString()); %>.
        <LI><B>Thẻ khai báo. </B><BR>
            <% private int accessCout = 0; %>
            Số lần truy cập vào Web site:  <%= ++accessCout %>
        <LI><B>Thẻ chỉ dẫn. </B><BR>
            <%@ page import = "java.util.* %>
            Ngày/giờ hiện thời: <%= new Date() %>
        </UL>
    </BODY>
    </HTML>
```

6.11.5. Các hành động của JSP

Các hành động JSP sử dụng các cấu trúc XML để điều khiển hành vi của Servlet Engine. JSP có những hành động sau:

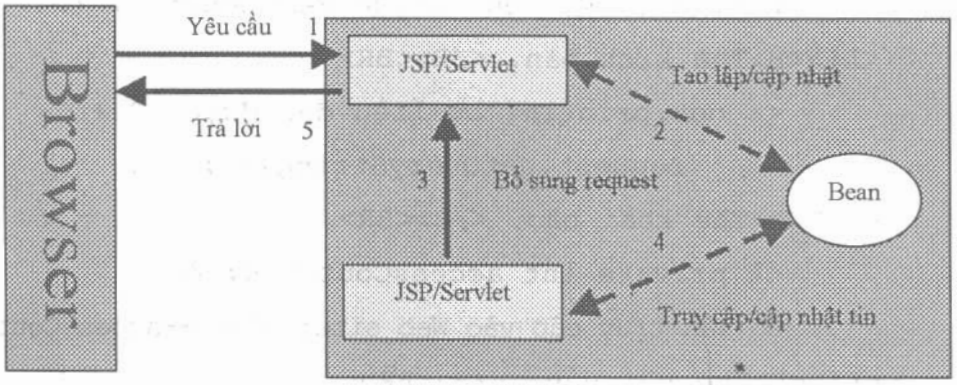
- `jsp:include` – Đưa vào một tệp khi trang tin được yêu cầu.
- `jsp:useBean` – Sử dụng đối tượng JavaBean.
- `jsp:setProperty` – Đặt lại thuộc tính cho JavaBean.
- `jsp:getProperty` – Đọc thuộc tính của JavaBean.
- `jsp:forward` – Hướng tới một trang mới.
- `jsp:plugin` – Cho phép chèn vào trình duyệt được xác định bởi OBJECT hoặc EMBED cần thiết để chạy Applet.

1. Thẻ jsp:include

Thẻ này cho phép bạn chèn các tệp cần thiết vào trang tin được tạo ra. Nó có dạng

```
<jsp:include page = "relative URL" flush = "true" />
```

Nó được dùng để định hướng yêu cầu tới những nguồn tài nguyên tĩnh hoặc động trong cùng ngữ cảnh theo địa chỉ relative URL như khi gọi trang JSP. Cơ chế thực hiện bằng cách tạo ra các tham số là các JavaBean tài nguyên (được giới thiệu ở chương V) và đưa chúng vào yêu cầu, được mô tả như hình 6.12.



Hình 6.12. **Cơ chế xử lý yêu cầu include**

Ví dụ 6.10. Viết chương trình JSP để đọc, hiển thị các tin được đưa vào từ các tệp chứa chúng.

```
<%-- WhatsNew.jsp --%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional //EN>
<HTML>
<HEAD>
<TITLE> Trang tin tức </TITLE>
<LINK REL = STYLESHEET
      HREF = "My-Style-Sheet.css"
      TYPE = "text/css">
</HEAD>
<BODY BGCOLOR = "#FDF5E6" TEXT = "#000000" LINK = "0000EE"
        VLINK = "#551A8B" ALINK = "FF0000">
<CENTER>
<TABLE BORDER = 5 BGCOLOR = "#EF8429">
      <TR><TH CLASS = "TITLE">
         Những tin tới nhận được từ JspNews.com </TABLE>
</CENTER>
<p>
Những tin mới nhận được:
<OL>
      <LI><jsp:include page="news/Item1.html" flush="true"/>
      <LI><jsp:include page="news/Item1.htm2" flush="true"/>
      <LI><jsp:include page="news/Item1.htm3" flush="true"/>
</OL>
</BODY>
</HTML>
```

2. Thẻ jsp:useBean

Thẻ hành động này cho phép tải các JavaBean (được trình bày ở chương V) vào trang JSP để sử dụng chúng. Điều này rất tiện lợi và hiệu quả vì nó cho phép ta khai thác được những khả năng sử dụng lại của các lớp được xây dựng sẵn trong Java mà không đòi hỏi phải thoả mãn những yêu cầu cụ thể, những yêu cầu này sẽ được JSP đưa vào trong Servlet để thực hiện. Nó có cú pháp đơn giản như sau:

```
<jsp:useBean id = "name" class = "package.class" />
```

Trước khi truy cập vào một Bean (JavaBean) trong trang JSP thì phải chỉ rõ nó có tên là gì (định danh), tên lớp tương ứng ở đâu. Khởi tạo một đối tượng của lớp được chỉ ra bởi class và nó liên kết với biến thông qua tên gọi id. Sẽ là hiệu quả hơn khi ta tham chiếu tới những JavaBean đã có sẵn, và jsp:useBean cho biết rằng, một đối tượng mới được khởi tạo chỉ khi chưa có đối tượng nào có cùng id đã được tạo ra trước đó. Khi đã tạo ra một Bean, ta có thể thay đổi các tính chất của nó bằng cách sử dụng thẻ jsp:setProperty, hoặc sử dụng các thẻ kịch bản và gọi những phương thức tường minh của đối tượng đó thông qua định danh id. Ví dụ, hãy xét thẻ sau:

```
<jsp:useBean id = "user" class = "company.Person"
    scope = "session" />
```

Một đối tượng của lớp Person được tạo ra và được đặt vào phiên làm việc session. Nếu sau đó, ở một trang JSP khác gặp phải thẻ jsp:useBean trên thì nó sẽ tham chiếu tới đối tượng ban đầu đã được sinh ra ở trong phiên session.

Khi đã khai báo một thành phần JavaBean, ta có thể truy cập vào các tính chất của nó thông qua thẻ jsp:getProperty và có thể thay đổi những tính chất đó thông qua thẻ jsp:setProperty.

Ví dụ 6.11. Ví dụ đơn giản hướng dẫn cách sử dụng Bean và đặt / xác định các thuộc tính của JavaBean.

```
<%-- BeanTest.jsp --%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional //EN>
<HTML>
<HEAD>
<TITLE> Sử dụng lại JavaBean trong JSP </TITLE>
<LINK REL=STYLESHEET
      HREF="My-Style-Sheet.css"
      TYPE="text/css">
</HEAD>
<CENTER>
<TABLE BORDER=5 BGCOLOR="#EF8429">
```

```
        <TR><TH CLASS="TITLE">
             Sử dụng lại JavaBean trong JSP </TABLE>
</CENTER>
<p>
<jsp:useBean id = "test" class = "hall.SimpleBean" />
<jsp:setProperty name = "test"
             property = "message"
             value = "Xin chào các bạn" />
<H1> Thông báo: <I>
<jsp:getProperty name = "test" property = "message" />
</I>
</BODY>
</HTML>
```

Chương trình của SimpleBean được sử dụng trong BeanTest có thể tải về từ địa chỉ trên mạng. SimpleBean có nội dung đơn giản như sau:

```
package hall;
public class SimpleBean{
    private String message = "Chưa xác định thông báo";
    public String getMessage(){
        return (message);
    }
    public void setMessage(String message){
        this.message = message;
    }
}
```

3. Thẻ jsp:setProperty

Bạn có thể sử dụng thẻ jsp:setProperty để đặt lại các tính chất của các Bean mà trang JSP tham chiếu tới. Có thể thực hiện theo hai cách.

+ *Cách thứ nhất*: sử dụng jsp:setProperty, nhưng bên ngoài của jsp:useBean như sau:

```
<jsp:useBean id="myName" ... />
...
<jsp:setProperty name="myName"
     property="someProperty" ... />
```

Trong trường hợp này, `jsp:setProperty` được thực hiện mà không cần để ý đến việc một đối tượng mới được tạo ra hay đã có sẵn đối tượng đó.

+ *Cách thứ hai*: sử dụng `jsp:setProperty` bên trong của thẻ `jsp:useBean` như sau:

```
<jsp:useBean id="myName" ... >
...
<jsp:setProperty name="myName"
        property="someProperty" ... />
</jsp:useBean>
```

Ở đây, `jsp:setProperty` chỉ thực thi được khi chưa có sẵn đối tượng `Bean` và một đối tượng mới được tạo ra.

`jsp:setProperty` có các thuộc tính sau:

Thuộc tính	Mô tả cách sử dụng
`name`	Thuộc tính yêu cầu chỉ định `Bean` có các tính chất sẽ được đặt lại. Thẻ `jsp:useBean` phải xuất hiện trước `jsp:setProperty`.
`property`	Thuộc tính yêu cầu chỉ ra tính chất mà bạn muốn đặt lại. Song, ở đây có trường hợp đặc biệt: giá trị `"*"` nghĩa là tất cả các tham số yêu cầu của `Bean` có tên sánh được với `id` sẽ truyền tới những phương thức tương ứng.
`value`	Thuộc tính được lựa chọn xác định giá trị của tính chất đó. Các giá trị xâu (String) được tự động chuyển sang các số kiểu `boolean`, `Boolean`, `byte`, `Byte`, `char`, và `Character` thông qua phương thức `valueOf()` của các lớp bao gói. Ví dụ, giá trị `s = "42"` được chuyển sang kiểu `int` hoặc `Integer`, thông qua `Integer.valueOf(s)`.

`param`	Thuộc tính được lựa chọn chỉ định tham số yêu cầu từ tính chất được dẫn xuất ra. Nếu yêu cầu hiện thời không có tham số như trên thì hệ thống không làm gì cả.
	`<jsp:setProperty name="orderBean"`
	`property="numberOfItems"`
	`param="numItems" />`
	Nếu ta bỏ qua cả `value` và `param`, thì nó có tên gọi giống như tên được cung cấp bởi `param` thông qua tên của `property`.

Ví dụ 6.12. Sử dụng `Bean` để tạo ra một bảng các chữ số nguyên tố. Nếu tham số có tên là `numDigits` trong dữ liệu yêu cầu, nó sẽ được truyền vào cho tính chất `numDigits` của `Bean`.

```
<%-- JspPrimes.jsp --%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
<HEAD>
<TITLE>Reusing JavaBeans in JSP</TITLE>
```

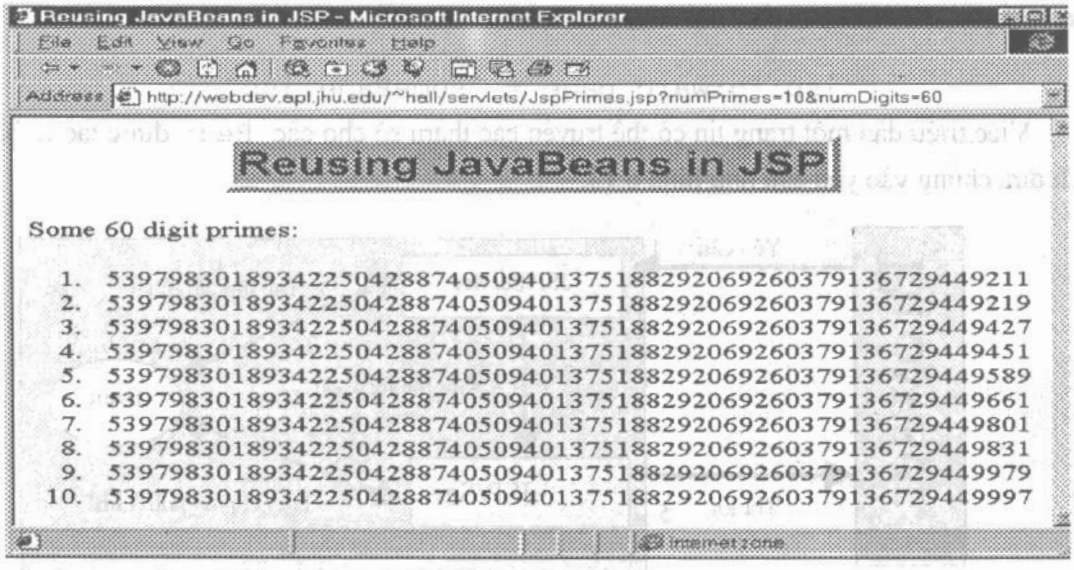
```
<LINK REL=STYLESHEET
       HREF="My-Style-Sheet.css"
       TYPE="text/css">
</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
   <TR><TH CLASS="TITLE">
       Reusing JavaBeans in JSP</TABLE>
</CENTER>
<P>
<jsp:useBean id="primeTable" class="hall.NumberedPrimes" />
<jsp:setProperty name="primeTable" property="numDigits" />
<jsp:setProperty name="primeTable" property="numPrimes" />
Some <jsp:getProperty name="primeTable" property="numDigits" />
digit primes:
<jsp:getProperty name="primeTable" property="numberedList" />

</BODY>
</HTML>
```

Chương trình NumberedPrimes.java để tìm các số nguyên tố được viết như sau:

```
package hall;
import java.util.*;
public class NumberedPrimes {
     private Vector primes;
     private int numPrimes = 15;
     private int numDigits = 50;

     public String getNumberedList() {
       if (primes == null) {
          PrimeList newPrimes =
             new PrimeList(numPrimes, numDigits, false);
          primes = newPrimes.getPrimes();
       }
       StringBuffer buff = new StringBuffer("<OL>\n");
       for(int i=0; i<numPrimes; i++) {
         buff.append("  <LI>");
         buff.append(primes.elementAt(i));
         buff.append("\n");
       }
       buff.append("</OL>");
       return(buff.toString());
     }
```

```
    public int getNumPrimes() {
      return(numPrimes);
    }
    public void setNumPrimes(int numPrimes) {
      if (numPrimes != this.numPrimes)
        primes = null;
      this.numPrimes = numPrimes;
    }
    public int getNumDigits() {
      return(numDigits);
    }
    public void setNumDigits(int numDigits) {
      if (numDigits != this.numDigits)
        primes = null;
      this.numDigits = numDigits;
    }
    public Vector getPrimes() {
      return(primes);
    }
    public void setPrimes(Vector primes) {
      this.primes = primes;
    }
  }
}
```

Khi thực hiện, ta có kết quả như sau:



Hình 6.12. **Chương trình sử dụng JavaBean trong JSP**

4. Thẻ jsp:getProperty

Thẻ này tìm các giá trị thể hiện các tính chất của Bean, được chuyển sang xâu và chèn vào kết quả. Có hai thuộc tính yêu cầu là:

- Tên của Bean được tham chiếu trước đó bởi jsp:useBean,
- Tính chất có giá trị tương ứng được chèn vào.

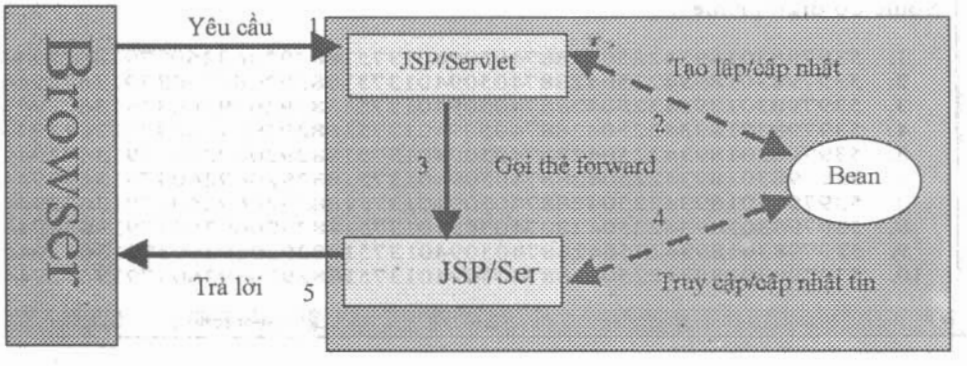
```
<jsp:useBean id="itemBean" ... />
...
<UL>
  <LI>Số các mặt hàng:
      <jsp:getProperty name="itemBean"
       property="numItems"/>
  <LI>Giá bán mỗi mặt hàng:
      <jsp:getProperty name="itemBean" property="unitCost" />
</UL>
```

5. Thẻ jsp:forward

Thẻ này cho phép bạn khai báo trước về yêu cầu tới một trang tin khác. Nó cho phép hướng dẫn để vào một trang JSP, Servlet, hay một trang HTML tĩnh bên trong ngữ cảnh như là cầu khẩn tới những trang đó. Kết quả là mọi công việc xử lý của trang hiện thời sẽ bị dừng lại ở nơi mà thẻ hướng dẫn này xuất hiện, mặc dù những công việc đó vẫn phải tiếp tục thực hiện.

```
<jsp:forward page = "somePage.jsp" />
```

Việc triệu dẫn một trang tin có thể truyền các tham số cho các Bean được tạo ra bằng cách đưa chúng vào yêu cầu như hình 6.13.



Hình 6.13. **Cơ chế xử lý yêu cầu forward**

BÀI TẬP

6.1. Xây dựng các lớp và các tệp HTML tương ứng để thực hiện các nhiệm vụ của Công ty `Sao Mai` như đã nêu trong ví dụ 6.8.

6.2. Xây dựng một diễn đàn trao đổi (`forum`) với `JSP` và `Servlet`. Yêu cầu chính của bài toán như sau.

1. Một diễn đàn trao đổi thông tin trên mạng cần cho phép các thành viên đăng ký tham gia, tạo các chủ đề, trả lời các bài viết; còn đối với người quản trị thì được phép tạo ra các hộp (`Box`) chủ đề, phân quyền cho các thành viên, quản lý thông tin diễn đàn ...
2. Thông tin của diễn đàn được thể hiện thông qua các bài viết của thành viên. Các bài viết được phân loại theo từng chủ đề, mỗi chủ đề sẽ đề cập xung quanh một vấn đề, sự kiện, ý kiến bình luận nào đó. Tập hợp các chủ đề liên quan đến một thể loại thông tin nhất định nào đó lại được nhóm vào trong một hộp (`Box`), ví dụ ta có thể có hộp Thể thao, hộp Âm nhạc, hộp Văn học .v.v. Việc phân cấp tạo ra các hộp chủ đề sẽ giúp cho người đọc dễ dàng theo dõi các bài viết cũng như giúp việc quản lý diễn đàn hiệu quả và linh động hơn.
3. Người quản lý cũng là thành viên của diễn đàn có chức năng quản lý diễn đàn. Người quản lý có thể thêm, xóa, sửa ... các thông tin bên trong diễn đàn mà họ trực tiếp quản lý. Việc chỉ định ai là người quản lý của từng hộp chủ đề sẽ do người quản trị (Administrator) quyết định.
4. Bạn có thể chỉnh sửa hay xoá bài viết của bạn bất kỳ lúc nào bằng cách vào diễn đàn có chứa bài viết, hoặc nhấn vào biểu tượng sửa hoặc xoá (). Không ai khác có quyền sửa bài viết của bạn, ngoại trừ người quản lý hay quản trị diễn đàn. Khi bài viết được chỉnh sửa, một dòng ghi chú sẽ xuất hiện ở cuối bài viết để thông báo về thời gian sửa và người sửa bài viết đó. Bạn cũng có thể xóa bài viết do mình đã tạo ra trước đó. Nếu quản trị mạng hay quản lý các diễn đàn chọn chức năng này, bài viết cũng sẽ bị xoá.
5. Bạn có thể chỉnh sửa thông tin cá nhân của mình dễ dàng bằng cách nhấn vào "Thông tin cá nhân" trên mỗi trang, điền các thông tin về bạn (Ngoại trừ tên đăng nhập không được sửa).

Chương 7
VẤN ĐỀ BẢO MẬT VÀ AN NINH THÔNG TIN

Chương VII trình bày những kỹ thuật bảo mật thông tin và đảm bảo an toàn hệ thống thông tin bằng Java.

- ✓ Những vấn đề quan trọng của bảo mật hệ thống thông tin
- ✓ Vai trò của các bộ nạp lớp `ClassLoader` và kiểm định byte code
- ✓ Bộ quản lý bảo mật và các lớp được cấp đặc quyền
- ✓ Những vấn đề bảo mật và chữ ký số
- ✓ Vấn đề chứng thực: chứng chỉ số, chữ ký số và xác thực danh tính người chứng thực
- ✓ Mật mã khóa công khai RSA và ứng dụng trong lập trình Java.

7.1. GIỚI THIỆU VỀ VẤN ĐỀ BẢO MẬT, AN TOÀN HỆ THỐNG THÔNG TIN

Bảo mật thông tin là vấn đề rất quan trọng, đang thu hút nhiều người tập trung nghiên cứu và tìm mọi giải pháp để đảm bảo an toàn, an ninh cho hệ thống phần mềm, đặc biệt là các hệ thống thông tin trên mạng. Việc bảo mật cho hệ thống có thể thực hiện theo nhiều phương diện, ở nhiều tầng khác nhau, bao gồm từ phương diện kiểm soát truy nhập vật lý vào hệ thống; thực hiện sửa chữa, cập nhật, nâng cấp hệ điều hành cũng như vá mọi lỗ hổng về an ninh tìm thấy; quản lý các hoạt động trao đổi thông tin trên mạng (giám sát qua bức tường lửa, các bộ định vị `Router`, phát hiện và phòng ngừa sự xâm nhập, v.v.); xây dựng các giải pháp bảo mật ở mỗi phần mềm đến quản lý người dùng thông qua việc cấp quyền sử dụng, mật khẩu, mật mã, v.v.

Song, cũng cần phải hiểu rằng, không có một hệ thống thông tin nào được bảo mật 100%. Bất kỳ một hệ điều hành nào mà bạn đang sử dụng cũng có thể có những lỗ hổng về an ninh và chưa phát hiện ra. Bộ phận quản trị mạng cũng có thể không phát hiện nhanh được những xâm nhập trái phép có thể làm hư hỏng, thậm chí phá huỷ thông tin của bạn. Ngay cả những thuật toán mật mã nổi tiếng như `MD5`, `SHA-1` cũng không thể khẳng định 100% mức độ an toàn, bởi vì có thể đến thời điểm này chưa phát hiện ra người có thể bẻ gãy được những thuật toán đó, nhưng cũng không có gì đảm bảo sẽ không có ai làm được điều đó trong tương lai.

Chính vì thế, ta cần phải làm hai việc. Việc thứ nhất là cần phải xác định *chính sách bảo mật* phù hợp cho hệ thống của mình. Chính sách bảo mật cho phép người dùng truy cập vào hệ thống theo những quyền đã được cấp và theo những thủ tục nhất định. Tuy nhiên, khi đưa ra một chính sách bảo mật thì ta lại phải trả giá cho những phí tổn về thời gian và tài nguyên vì những qui định bắt buộc (thường là bất tiện) đối với người sử dụng và những tính phiền phức của các thủ tục bảo mật. Việc thứ hai là phải xây dựng các chiến lược *phát hiện và phòng chống* các cuộc tấn công vào hệ thống.

Việc đầu tiên cần phải thực hiện khi lập kế hoạch bảo mật hệ thống là bạn phải xác định rõ các yêu cầu cần bảo mật, những gì bạn cần phải bảo vệ trước các cuộc tấn công, chính sách bảo mật phải dựa trên những cơ sở nào?. Thông thường có hai loại yêu cầu bảo mật chung cho các hệ thống phần mềm trao đổi giữa các doanh nghiệp (B2B) [2].

- *Vấn đề bảo mật trong truyền thông.* Phần lớn các hệ thống của chúng ta hiện nay được kết nối mạng. Khi thông tin được gửi đi trên mạng (Internet) thì nó phải được bảo vệ để chống lại những kẻ nghe (xem) trộm. Bất kỳ một người nào đó cũng có thể truy cập vào các cáp mạng hoặc qua bộ định vị Router để làm thay đổi nội dung trao đổi trên đường truyền. Ta không thể biết được liệu đã có một sự thay đổi nào đã được thực hiện đối với các nội dung trao đổi kể từ lúc chúng được gửi đi hay chưa. Ngay cả khi bạn tin rằng những nội dung thông tin trao đổi được bảo vệ chống những kẻ nghe trộm và những người lạ thì những giao thức truyền thông vẫn có thể còn có những lỗ hổng mà ta chưa phát hiện ra. Trong giao thức TCP/IP, người gửi gói tin được xác định thông qua địa chỉ IP và cổng kết nối ở phần đầu của gói (Package Header) và chính các trường thông tin này có thể dễ dàng thay đổi được. Điều này dẫn tới vấn đề quan trọng là cần chứng thực, xác thực người gửi. Nói chung, có bốn yêu cầu cơ bản về bảo mật truyền thông.

  (i) *Đảm bảo tin cậy.* Các nội dung thông tin không bị theo dõi hoặc sao chép bởi những thực thể không được uỷ thác.

  (ii) *Đảm bảo toàn vẹn.* Các nội dung thông tin không bị thay đổi bởi những thực thể không được uỷ thác.

  (iii) *Chứng minh xác thực.* Không ai có thể tự trá hình như là một bên hợp pháp trong quá trình trao đổi tin.

  (iv) *Không thể thoái thác trách nhiệm.* Người gửi tin không thể thoái thác về những sự việc và những nội dung thông tin mà thực tế họ đã gửi đi.

  Sự phát triển của công nghệ mật mã cho phép ta thực hiện được ba yêu cầu đầu tiên. Ví dụ, bạn có thể sử dụng SSL/TLS [4], được định nghĩa bởi Netscape Communication Corporation cho việc bảo mật kết nối theo HTTP, để bảo mật chương trình ứng dụng dựa vào Java. Yêu cầu thứ tư có thể thực hiện được bằng chữ ký số.

- *Kiểm soát truy cập*. Khi nhận được một thông tin từ một kênh truyền được bảo mật, ta cần phải dựa vào chính sách bảo mật để biết được những thao tác nào là được phép thực hiện đối với những thông tin đó. Việc kiểm soát truy cập đôi khi cũng nhầm lẫn với việc xác thực. Trong nhiều trường hợp, cơ chế kiểm soát truy cập và cơ chế xác thực được tích hợp vào trong một cơ chế chung. Ví dụ, bạn có thể truy cập vào các trang Web của Apache Web Server thông qua các mật khẩu chỉ khi bạn đã được xác thực. Nhưng, nói chung ta cần phân biệt cơ chế kiểm soát truy cập với cơ chế xác thực. Xác thực được sử dụng để khẳng định sự đồng nhất của người tham gia trao đổi tin. Trong những hệ thống lớn, thường có một cơ sở dữ liệu xác thực chứa các định danh và mật khẩu của nhiều người dùng, nhưng chỉ tập con trong số họ được quyền truy cập vào những thành phần đặc biệt của hệ thống.

Như chúng ta đã biết, mục đích của công nghệ Java là "*Viết chương trình một lần và chạy ở mọi nơi*". Tất nhiên, việc phát tán các applet là thực sự có ý nghĩa trong thực tế chỉ khi những người sử dụng được đảm bảo rằng các mã lệnh đó không bị thay đổi và không phá huỷ trên máy của họ. Vì vậy, vấn đề bảo mật thông tin được xem là vấn đề chính được đề cập bởi cả những người thiết kế lẫn người sử dụng công nghệ Java để phát triển các phần mềm ứng dụng phân tán.

Trong công nghệ Java có ba cơ chế đảm bảo an toàn dữ liệu [2].

- *Dựa vào những đặc trưng của ngôn ngữ như*: kiểm soát giới hạn của các cấu trúc dữ liệu như cấu trúc mảng, kiểm tra chặt chẽ sự chuyển đổi giữa các kiểu đối tượng, không sử dụng con trỏ số học, v.v.
- *Cơ chế điều khiển việc thực hiện của các mã lệnh* truy cập vào tệp, truy cập mạng, v.v.
- *Cơ chế hỗ trợ ký, xác nhận mã lệnh*. Người viết chương trình có thể sử dụng những thuật toán mã hoá chuẩn để xác thực các mã lệnh trong chương trình. Người dùng những chương trình này có thể xác thực được danh tính người tạo ra chúng và kiểm tra xem liệu nó đã bị thay đổi (lần cuối cùng) sau khi đã được ký xác nhận hay không.

Máy ảo Java JVM sẽ giúp chúng ta thực hiện được cơ chế thứ nhất, nghĩa là kiểm tra chỉ số các phần tử của mảng, sự tương thích về kiểu giữa các lớp đối tượng, .v.v.

Một khi các tệp lớp (.class) được nạp vào JVM, chúng sẽ được kiểm tra xem có nguyên vẹn hay không. Điều quan trọng là, ngoài bộ nạp lớp mặc định, ta có thể tạo ra các bộ nạp lớp Loader riêng để kiểm soát các hoạt động trong JVM giúp ta thực hiện được cơ chế thứ hai.

Để đảm bảo an toàn cao hơn, ta nên sử dụng cả những bộ nạp ClassLoader chung của hệ thống (mặc định) và những bộ nạp lớp riêng, kết hợp với lớp quản trị an ninh SecurityManager để kiểm soát sự hoạt động của các đoạn mã lệnh. Nói chung ta nên xây dựng những lớp quản trị an ninh riêng cho từng ứng dụng.

Cơ chế thứ ba có thể dễ dàng thực hiện bằng cách sử dụng các thuật toán mật mã được cung cấp bởi các lớp trong gói `java.security` hoặc xây dựng những thuật toán mật mã mới để ký nhận và xác thực chương trình. Cơ chế này sẽ được đề cập cụ thể ở phần cuối chương.

7.2. BỘ NẠP LỚP VÀ KIỂM TRA BYTE CODE

Chương trình dịch của `Java` chuyển đổi mã nguồn sang ngôn ngữ máy giả thuyết, máy ảo JVM. Mỗi lớp trong chương trình `Java` được tạo ra trong JVM là một tệp lớp có đuôi `.class`. Các tệp lớp này muốn thực hiện lại phải được thông dịch để chuyển các lệnh trong JVM sang mã máy của những máy đích [1].

Lưu ý rằng, chương trình thông dịch của JVM chỉ nạp những lớp cần thiết tại mỗi thời điểm để thực hiện chương trình. Ví dụ, ta xét quá trình bắt đầu thực hiện chương trình ứng dụng độc lập `MyProgram.class`. JVM sẽ thực hiện các bước như sau:

1. JVM có cơ chế để nạp các tệp lớp liên quan, đọc từ đĩa hay tải xuống từ những `Website` chứa các lớp đó. Nó sử dụng cơ chế này để nạp `MyProgram.class`.
2. Nếu lớp `MyProgram` có các trường dữ liệu hoặc kế thừa từ một lớp cha khác thì những lớp đó cũng được nạp về.
3. JVM sẽ thực hiện phương thức `main()` trong `MyProgram`, vì phương thức này là tĩnh nên không cần phải tạo ra đối tượng để gọi nó.
4. Nếu trong `main()` lại yêu cầu những đối tượng lớp khác thì chúng sẽ được nạp bổ sung theo yêu cầu.

Nói chung, ta không cần can thiệp vào quá trình trên, hệ thống tự động nạp và kiểm soát các lớp được tải xuống. Tuy nhiên, để hệ thống an toàn hơn, ta nên xây dựng bộ kiểm soát nạp lớp riêng. Nó cho phép ta kiểm tra tính toàn vẹn của các mã byte code trước khi đưa vào JVM

7.2.1. Viết bộ nạp lớp ClassLoader riêng

Mọi bộ nạp lớp đều cài đặt `ClassLoader`. Phương thức `loadClass()` ở lớp này sẽ xác định cách nạp lớp ở mức đỉnh. Một khi có một lớp được nạp thì tất cả các lớp khác mà nó tham chiếu tới đều được nạp bởi cùng một bộ nạp.

Để tạo ra được một bộ nạp, ví dụ `MyClassLoader`, ta phải viết đè phương thức

```
loadClass(String className, booolean resolve)
```

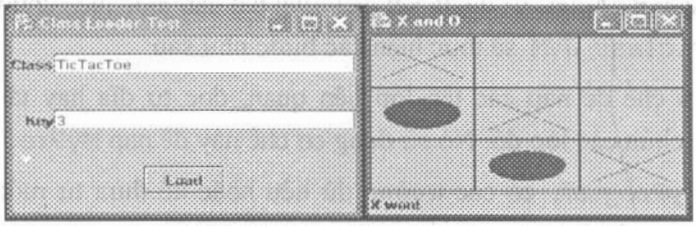
theo các bước như sau.

1. Kiểm tra xem bộ nạp lớp này đã được nạp hay chưa. Mục đích là bộ nạp lớp của ta chỉ phải lưu lại một bản ghi về những lớp mà nó đã nạp.

2. Nếu là lớp mới thì kiểm tra xem nó có phải là lớp hệ thống hay không. Trường hợp không phải là lớp hệ thống, các mã bytecode của lớp đó được nạp từ các hệ thống tệp hoặc từ những nguồn khác.
3. Gọi phương thức defineClass() của ClassLoader để giới thiệu các bytecode cho JVM.

Thông thường, bộ nạp lớp sử dụng một phép ánh xạ (thường là bảng băm) để lưu giữ các tham chiếu tới các lớp đã được nạp.

Ví dụ 7.1. Xây dựng bộ nạp lớp để nạp những tệp lớp đã được mã hoá. Người sử dụng phải nhập vào tên của lớp chính (lớp chứa hàm main()), một ứng dụng cần thực hiện và khóa mật mã. Chương trình sử dụng bộ nạp lớp riêng để nạp các lớp chỉ định và gọi hàm main(). Bộ nạp lớp sẽ giải mã các lớp được tải về.



Hình 7.1. **Chương trình nạp và kiểm soát lớp**

Người sử dụng cho biết tên tệp lớp chính, ví dụ chương trình TicTacToe.class, xác định lại khóa (mặc định là 3) rồi nhấn nút Load. Chương trình được nạp theo bộ nạp được xây dựng và hai người chơi có thể đánh cờ ca rô như hình 7.1.

```
// MyClassLoaderTest.java
import java.util.*;
import java.io.*;
import java.lang.reflect.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MyClassLoaderTest{
   public static void main(String[] args){
      JFrame f = new MyClassLoaderFrame();
      f.show();
   }
}
class MyClassLoaderFrame extends JFrame{
   public MyClassLoaderFrame(){
```

```
        setTitle("Class Loader Test");
        setSize(300, 200);
        addWindowListener(new WindowAdapter(){
              public void windowClosing(WindowEvent e){
                    System.exit(0);
              }
        });
        getContentPane().setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.weightx = 0;
        gbc.weighty = 100;
        gbc.fill = GridBagConstraints.NONE;
        gbc.anchor = GridBagConstraints.EAST;
        add(new JLabel("Class"), gbc, 0, 0, 1, 1);
        add(new JLabel("Key"), gbc, 0, 1, 1, 1);
        gbc.weightx = 100;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.anchor = GridBagConstraints.WEST;
        add(nameField, gbc, 1, 0, 1, 1);
        add(keyField, gbc, 1, 1, 1, 1);
        gbc.fill = GridBagConstraints.NONE;
        gbc.anchor = GridBagConstraints.CENTER;
        JButton loadButton = new JButton("Load");
        add(loadButton, gbc, 0, 2, 2, 1);
        loadButton.addActionListener(
              new ActionListener(){
                 public void actionPerformed(ActionEvent event){
                    runClass(nameField.getText(), keyField.getText());
                    }
              });
    }
    public void add(Component c, GridBagConstraints gbc, int x, int y,
                                 int w, int h) {
        gbc.gridx = x;
```

```
        gbc.gridy = y;
        gbc.gridwidth = w;
        gbc.gridheight = h;
        getContentPane().add(c, gbc);
    }

    public void runClass(String name, String key){
        try{
            ClassLoader loader =
                new CryptoClassLoader(Integer.parseInt(key));
            Class c = loader.loadClass(name);
            String[] args = new String[]{};
            Method m = c.getMethod("main", new Class[]{args.getClass()});
            m.invoke(null, new Object[]{args});
        }catch(Throwable e){
            JOptionPane.showMessageDialog(this, e);
        }
    }
    private JTextField nameField = new JTextField(30);
    private JTextField keyField = new JTextField("3", 4);
}

// Xây dựng lớp nạp riêng
class CryptoClassLoader extends ClassLoader{
    private Map classes = new HashMap();
    private int key;
    public CryptoClassLoader(int k){
        key = k;
    }
    protected synchronized Class loadClass(String name, boolean resolve)
            throws ClassNotFoundException{
        // Kiểm tra xem lớp đã được nạp chưa?
        Class c1 = (Class)classes.get(name);
        if(c1 == null){
```

```
            // Lớp mới
            try{
                  // Kiểm tra xem có phải lớp hệ thống không?
                  return findSystemClass(name);
            }catch(ClassNotFoundException e){}
            catch(NoClassDefFoundError e){}
            // Nạp lớp theo từng byte, tuỳ vào từng bộ nạp lớp
            byte[] classBytes = loadClassBytes(name);
            if(classBytes == null) throw new ClassNotFoundException(name);
            c1 = defineClass(name, classBytes, 0, classBytes.length);
            if(c1 == null) throw new ClassNotFoundException(name);
            classes.put(name, c1);      // Ghi nhớ lớp được nạp
      }
      if(resolve) resolveClass(c1);
      return c1;
}
private byte[] loadClassBytes(String name){
      // Nạp từng byte và mã hoá chúng theo thuật toán "caesar"
      String cname = name.replace('.', '/') + ".caesar";
      FileInputStream in = null;
      try{
            in = new FileInputStream(cname);
            ByteArrayOutputStream buff =
                  new ByteArrayOutputStream();
            int ch;
            while((ch = in.read()) != -1){
                  byte b = (byte)(ch - key);
                  buff.write(b);
            }
            in.close();
            return buff.toByteArray( ;
      }catch(IOException e){
            if(in != null){
```

```
                try{
                    in.close();
                }catch(IOException e1){
                }
            }
            return null;
            }
        }
    }
```

java.lang.ClassLoader API

- `Class defineClass(String name, byte data[], int offset, int length)` : Đưa thêm một lớp mới vào JVM, trong đó có các tham số:
  - name: là tên của một lớp có thể chứa cả tên của gói chứa nó, nhưng không cần chỉ rõ cả .class.
  - data: một mảng để chứa các byte code của lớp đó
  - offset: byte code bắt đầu trên mảng
  - length: độ dài của mảng.
- `void loadClass(String name, boolean resolve)`: Được cài đặt để nhận được các `byte code` của lớp cần nạp vào JVM, nếu `resolve` là `true`. Trong đó, các tham số:
  - name: là tên của một lớp có thể chứa cả tên của gói chứa nó, nhưng không cần chỉ rõ cả .class.
  - resolve: có giá trị true nếu `resolveClass()` cần gọi để kiểm tra sau khi lớp được nạp.
- `Class findSystemClass(String name)` : Tìm lớp hệ thống được nạp vào.
- `void resolveClass(Class c)`: Được gọi thực hiện khi cờ `resolve` là true.

7.2.2. Kiểm tra byte code

Khi một bộ nạp lớp giới thiệu các byte code cho JVM thì trước tiên những byte code này phải được kiểm định bởi một *bộ kiểm tra*. Bộ kiểm tra đảm bảo rằng những lệnh được nạp vào khi thực hiện sẽ không gây ra thiệt hại nào cả. Tất cả các ngoại lệ của các lớp hệ thống sẽ được kiểm tra và được thông báo khi chúng xuất hiện.

Bộ kiểm tra có thể thẩm định:

- Các biến phải được khởi tạo giá trị trước khi chúng được sử dụng.
- Trong các lời gọi hàm, các kiểu tham chiếu phải phù hợp, tương thích với các tham số hình thức.
- Đảm bảo các luật truy cập vào các thành phần riêng không bị vi phạm.
- Đảm bảo chương trình khi thực hiện không bị tràn bộ nhớ, v v.

Một khi phát hiện ra một sai sót bất kỳ thì lớp đó được xem là không hợp lệ và sẽ không được nạp vào JVM

Quan trọng hơn, trong thế giới mở với Internet, ta phải tìm cách bảo vệ để chống lại những người khác cố tình phá hoại. Ví dụ, họ có thể thay đổi kết quả tính toán của chương trình, thay đổi các trường dữ liệu riêng của các đối tượng trong hệ thống, hay một chương trình có thể bị ngắt bởi hệ thống *bảo mật* của một trình duyệt

Song, bạn có thể phân vân tại sao không có một bộ kiểm tra đặc biệt để kiểm định tất cả những điều nêu trên?. Trước hết phải biết rằng, trong Java chương trình biên dịch luôn kiểm duyệt và không cho phép tạo ra những tệp lớp, trong đó có những biến được sử dụng nhưng chưa khởi tạo giá trị hay những biến riêng (private) mà lại bị các đối tượng của lớp khác truy nhập, v v. Nhân đây cũng cần lưu ý là những vấn đề kiểm soát này không được đảm bảo ở những ngôn ngữ khác như C/C++, hay Pascal. Tuy nhiên, dạng byte code sử dụng trong tệp lớp là dạng tài liệu hoá và việc sửa nó không có gì khó khăn đối với những người lập trình Assembly có kinh nghiệm, họ có thể sử dụng những hệ soạn thảo hexa và sửa bằng tay để tạo ra các tệp lớp hợp lệ nhưng có những lệnh không an toàn trong JVM

Ví dụ, sau đây chỉ ra khả năng thay đổi tệp lớp và có thể dẫn đến những kết quả không mong muốn.

Ví dụ 7.2. Chương trình đơn giản chạy được cả độc lập lẫn Applet.

```
// VerifierTest.java
import java.awt.*;
import java.applet.*;
public class VerifierTest extends Applet{
   public static void main(String args[]){
      System.out.println("1 + 2 = " + fun());
   }

   static int fun(){
      int n, m;
      m = 1; n = 2;
      // Sử dụng hệ soạn thảo như Hex Workshop thay đổi "n = 2" bằng "m = 2"
      int r = m + n;
```

```
        return r;
    }
    public void paint(Graphics g){
        g.drawString("1 + 2 = " + fun(), 20, 20);
    }
}
```

Chương trình thực hiện sẽ hiển thị kết quả lên màm hình

```
1 + 2 = 3
```

Nếu ta thay đổi hàm `fun()` ở ví dụ trên thành:

```
static int fun(){
        int n, m;
        m = 1;
        m = 2;
        int r = m + n;
        return r;
}
```

Trong trường hợp này, biến n chưa được khởi tạo giá trị, và do vậy nó có thể nhận giá trị ngẫu nhiên trong bộ nhớ mà những ngôn ngữ lập trình khác như C/C++, Pascal không kiểm soát được. Riêng đối với Java, chương trình dịch phát hiện ra vấn đề này và nó thông báo là có lỗi.

Để tạo ra một tệp lớp nhập nhằng, ta phải can thiệp vào các câu lệnh ở mức sâu hơn, mức mã lệnh. Trước tiên, sử dụng javap để dịch chương trình `VerifierTest.java`. Câu lệnh

```
javap -c VerifierTest
```

cho các byte code trong tệp lớp dưới dạng các câu lệnh ký hiệu gợi nhớ.

Phương thức `int fun()` đã được dịch

```
0 iconst_1
1 istore_0
2 iconst_2
3 istore_1
4 iload_0
5 iload_1
6 iadd
7 istore_2
8 iload_2
9 ireturn
```

Tiếp theo, sử dụng hệ soạn thảo hexa (như Hex Workshop) để thay đổi lệnh thứ ba là istore_1 bằng istore_0. Bởi vì biến cục bộ istore_0 ứng với m, do vậy nó được khởi tạo hai lần và biến n không được khởi tạo giá trị. Ghi lại tệp vừa sửa vào tên cũ VerifierTest.class.

Thử chạy chương trình VerifierTest với chế độ kiểm tra mặc định ta nhận được thông báo lỗi:

```
Exception in thread "main" java.lang.VerifyError: (class:
VerifierTest, method: fun signature: ()I) Accessing
value from unitialized register 1
```

Nhưng, nếu ta chạy chương trình trên với tuỳ chọn –noverify,

```
java –noverify VerifierTest
```

nghĩa là không cần kiểm tra thì chương trình thực hiện và cho một kết quả ngẫu nhiên, ví dụ

```
1 + 2 = 15102330
```

vì n chưa được khởi tạo nên nó có thể nhận kết quả ngẫu nhiên trong bộ nhớ.

Lưu ý: Bộ kiểm tra luôn canh trừng để chống lại sự thay đổi ác ý đối với những tệp lớp, nhưng không kiểm tra những tệp được sinh bởi chương trình dịch.

7.3. LỚP SECURITYMANAGER VÀ PERMISSION

Như ở trên đã nêu, một lớp được nạp vào JVM bằng một bộ nạp lớp và được kiểm định bởi bộ kiểm tra. Cơ chế đảm bảo an ninh thứ ba trong môi trường Java được thực hiện bởi bộ quản lý *bảo mật* thông qua lớp SecurityManager. Nó kiểm tra xem những thao tác chỉ định có được phép hay không. Những thao tác cần kiểm tra tính an ninh bao gồm:

- Luồng hiện thời có thể tạo ra một bộ nạp lớp mới;
- Luồng hiện thời có thể tạo ra một tiến trình con;
- Luồng hiện thời có thể dừng JVM;
- Luồng hiện thời có thể truy cập vào các thành phần của lớp khác;
- Luồng hiện thời có thể truy cập hoặc làm thay đổi các thuộc tính của hệ thống;
- Luồng hiện thời có thể đọc hoặc ghi vào một tệp;
- Luồng hiện thời có thể xoá một tệp;
- Luồng hiện thời có thể kết nối với một Socket từ một máy xác định thông qua số hiệu của cổng;
- Luồng hiện thời có thể phải chờ, hay mở kết nối với một Socket từ một máy xác định thông qua số hiệu của cổng;
- Luồng hiện thời có thể gọi các phương thức: stop(), suspend(), resum(), setPriority(), setName(), setDaemon() của một luồng hoặc của một nhóm luồng;

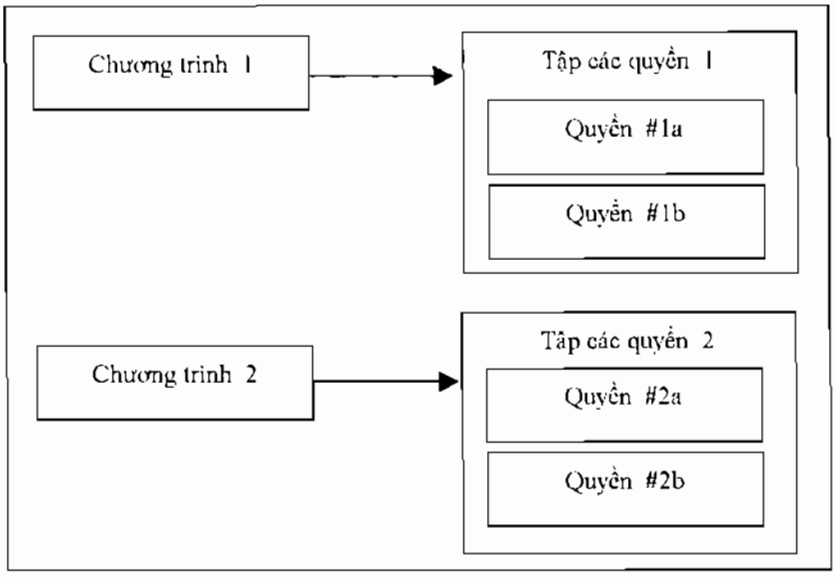
- Luồng hiện thời có thể bắt đầu in một tệp;
- Luồng hiện thời có thể truy cập vào vùng nhớ giành riêng của hệ thống;
- v.v.

Khi chạy chương trình ứng dụng Java, mặc định nó được thực hiện không có quản lý an ninh. Muốn cài đặt nó trong chương trình ứng dụng phải gọi phương thức setcurityManager() của lớp System.

7.3.1. Vấn đề bảo mật trong Java 2 Platform

Các JDK 1.0, 1.1 có mô hình bảo mật khá đơn giản: các đối tượng trong chương trình ứng dụng độc lập có toàn quyền truy cập tới các tài nguyên cục bộ, còn bộ quản lý an ninh của các applet từ chối mọi truy cập tới các tài nguyên cục bộ.

Java 2 Platform có cơ chế quản lý an ninh linh hoạt hơn. Nó sử dụng một *chính sách bảo mật* để cấp quyền (giấp phép) được phép thực hiện cho từng chương trình khác nhau.



***Hình 7.2.* Chính sách cấp quyền được thực hiện cho các chương trình**

Trong đó, *quyền* là một đặc tính được phép thực hiện và được kiểm soát bởi bộ quản lý an ninh, còn được gọi là *đặc quyền*. JDK1.2 hỗ trợ tạo ra những lớp để tạo ra các quyền cho chương trình. Ví dụ

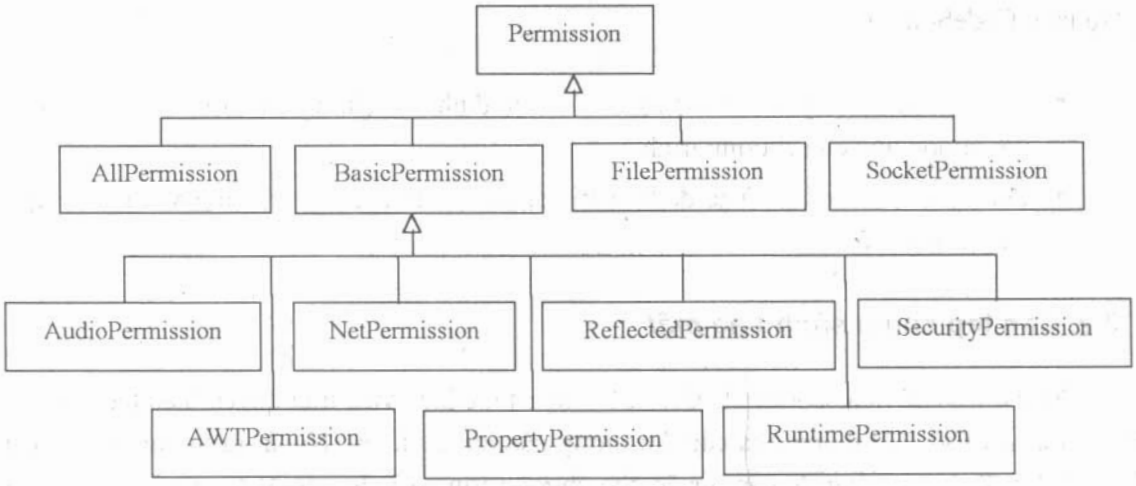
```
FilePermission p = new  FilePermission("/tmp/*", "read, write");
```

cho phép đọc, ghi một tệp bất kỳ ở thư mục /tmp.

Các lớp quản lý quyền thao tác trong JDK 1.2 tạo ra một cấu trúc phân cấp như hình 7.3.

JDK cung cấp mô hình chuẩn để kiểm tra quyền thao tác dựa trên hai lớp:

```
java.security.SecurityClassLoader
java.lang.SecurityManager
```



Hình 7.3. **Cấu trúc phân cấp của các lớp `Permision` trong `JDK 1.2`**

Ngoài ra, mô hình chuẩn còn dựa vào đối tượng của lớp `Policy` để cấp quyền thao tác cho các chương trình nguồn. Tại mỗi thời điểm chỉ có một đối tượng của `Policy` được quyền vận dụng. Ta có thể sử dụng

```
Policy curentPolicy = Policy.getPolicy()
```

để xác định chính sách hiện thời được phép thực hiện của chương trình!

Chi tiết hơn về vấn đề *bảo mật* với Java, nhất là mô hình *bảo mật* cơ sở của Java ứng dụng trong việc phát triển những hệ thống nhúng, thẻ thông minh, v.v., bạn có thể đọc ở tài liệu trực tuyến http://www.securingjava.com.

java.lang.SecurityManager API

- `void checkPermission(Permission p)`
- `void checkPermission(Permission p, Object context)`

  Kiểm tra xem chính sách *bảo mật* hiện thời có cho phép hay không.

java.lang.SecurityManager API

- `static Policy getPolicy()`: Xác định chính sách hiện thời.
- `PermissionCollection getPermissions(CodeSource source)`

  Lấy lại quyền được thao tác đối với source cho trước.

java.lang.PermissionCollection API

- `void add(Permission p)`: Bổ sung thêm quyền `p` vào tuyển tập các quyền cấp trước.
- `Enumeration elements()`: Lấy ra dãy liệt kê các quyền trong tuyển tập.

java.lang.CodeSource API

- `Certificate[] geteCertificate()`: Xác định các chứng chỉ đối với tệp lớp ứng với nguồn gốc của chương trình.
- `URL getLocation()`: Xác định vị trí (địa chỉ) của các tệp lớp ứng với nguồn gốc của chương trình.

7.3.2. Các tệp chính sách *bảo mật*

Như ở trên đã nêu, `SecurityClassLoader` thực hiện việc trao quyền thao tác cho các lớp được nạp vào JVM thông qua các đối tượng của lớp `Policy`. Ngoài ra ta còn có thể cài đặt một lớp `Policy` riêng để cấp quyền cho chương trình. Ở chương V ta cũng đã sử dụng tệp chính sách (`.policy`) để có được quyền truy cập từ xa vào các máy tính trên mạng. Ví dụ, một tệp `MyApp.policy` mẫu có dạng

```
grant codeBase www.horstmann.com/classes{
    permission java.io.FilePermission "/tmp/*", "read,write";
};
```

cấp quyền đọc, ghi các tệp vào thư mục `/tmp` cho tất cả các mã được nạp từ `www.horstmann.com/classes`. Ta có thể sử dụng bất kỳ hệ soạn thảo nào, như `Notepad`, `Office Word`, `Jcreator`, `v.v`, để soạn thảo các tệp chính sách (dưới dạng tệp văn bản).

Lưu ý:

1. Lớp chính sách mặc định được đặt trong tệp `java.security` ở thư mục con `/jre/lib` của JDK. Theo mặc định, tệp này chứa dòng

   ```
   policy.provider=sun.security.provider.policyFile
   ```

2. Ta có thể thay đổi các vị trí của các tệp chính sách trong `java.security`. Tệp chính sách của java: `java.policy` được đặc tả mặc định là:

   ```
   policy.url.1=file:${java.home}/lib/security/java.policy
   policy.url.2=file:${user.home}/java.policy
   ```

   Người quản trị hệ thống có thể thay đổi tệp `java.policy` và chỉ rõ những địa chỉ URL thường trực trên những `Server` khác.

Thực hiện `MyApp` với quyền được cấp trong tệp `MyApp.policy` nêu trên như sau:

```
java -Djava.security.policy = MyApp.policy MyApp
```

Tương tự đối với `Applet`, ta thực hiện

```
appletviewer -J-Djava.security.policy=MyApplet.policy
    MyApplet.html
```

Tệp chính sách chứa một dãy các mục được đảm bảo. Mỗi mục có dạng như sau:

```
grant codesource
{ permission-1;
   permission-2;
      . . .
};
```

Trong đó,

- `codesource` là cơ sở của mã lệnh `codeBase` (có thể không cần khai báo nếu mục đầu vào áp dụng cho tất cả các nguồn) và tên của những người ký giấy chứng nhận (có thể bỏ qua nếu không có yêu cầu về người ký xác nhận).

+ codeBase xác định nơi chứa các lớp sẽ được tải về:

```
codeBase "url"
```

- Nếu URL kết thúc bằng '/', nghĩa là nó chỉ tới một thư mục, ngược lại là tên của một tệp JAR. Ví dụ,

```
grant codeBase "www.horstman.com/classes/" {...}
grant codeBase "www.horstman.com/classes/MyApp.jar" {...}
```

- `codeBase` là một địa chỉ URL và chứa phần tử ngăn cách '/' giữa các tệp, ngay cả đối với URL trong Window, ví dụ

```
grant codeBase "file:d:users/myapps/classes"{...}
```

- Các đặc quyền `permission` có dạng cấu trúc như sau:

```
permission className targetName, actionList
```

+ `className` là tên đầy đủ của lớp đặc quyền thực hiện, ví dụ như java.net.NetPermission, java.net.SocketPermission, java.io.FilePermission, java.security.SecurityPermission, java.util.PropertyPermission, java.awt.AWTPermissionjava.security.AllPermission

+ `targetName` là tên tệp hay thư mục được đặc quyền thực hiện, có dạng:

`fileName`	Một tệp
`directory/`	Một thư mục
`directory/*`	Tất cả các tệp của thư mục `directory`
*	Tất cả các tệp của thư mục hiện thời
`directory/-`	Tất cả các tệp của thư mục `directory` hoặc ở một thư mục con nào đó của nó
-	Tất cả các tệp của thư mục hiện thời hoặc ở một thư mục con nào đó của nó
`<<ALL FILES>>`	Tất cả các tệp trong hệ thống.

+ actionList là danh sách các hành động như: read, write, delete, execute, accept, connect, listen, resolve.

Ví dụ:

```
grant codeBase "file:d:/myapps/classes/"{
        permission java.io.FilePermission "/myapp/-", "read, write";
};
```

Lưu ý: Một số lớp đặc quyền không cần chỉ ra targetName và actionList, ví dụ java.security.AllPermission.

Lớp đặc quyền java.net.NetPermission yêu cầu tên và cổng của máy kết nối dạng:

hostname hoặc IPaddress	Máy chủ
localhost hoặc xâu rỗng	Máy khách
*.domainSuffix	Hậu tố miền của máy thực hiện
*	Cho tất cả các máy.

Các cổng kết nối có thể liệt kê theo dãy:

:n	Một cổng n
:n-	Tất cả được đánh số từ n trở lên
:-n	Tất cả được đánh số từ n trở xuống
:n1-n2	Tất cả được đánh số từ n1 đến n2

Ví dụ,

```
permission java.net.SocketPermission "*.horstman.com:8000-8999", "connect";
```

Lớp đặc quyền java.util.PropertyPermission khai báo về các thuộc tính đích cần thực hiện có hai dạng chính:

property	Chỉ rõ một đặc tính
propertyPrfix.*	Tất cả các đặc tính có cùng tiếp đầu ngữ.

Ví dụ "java.home" hay "java.vm.*".

```
java.util.PropertyPermission "java.vm.*", "read";
```

cho phép chương trình đọc tất cả các thuộc tính bắt đầu bằng java.vm

Ngoài ra, ta có thể sử dụng các thuộc tính của hệ thống trong các tệp chính sách. Dấu hiệu ${property} có thể được thay thế bằng giá trị xác định. Ví dụ, ${user.home} được thay thế bằng thư mục của người sử dụng.

```
permission java.io.FilePermission "${user.home}", "read, write";
```

Để tạo ra các tệp chính sách độc lập với các nền thì phải sử dụng thuộc tính file.separator thay thế cho '/' hoặc "\\" . Người ta sử dụng ký hiệu rút gọn ${/} thay thế cho ${file.separator}. Ví dụ

```
permission java.io.FilePermission "${user.home}${/}-", "read, write";
```

cho phép đọc, ghi tất cả các tệp của thư mục của người sử dụng và bất kỳ một thư mục con của thư mục đó.

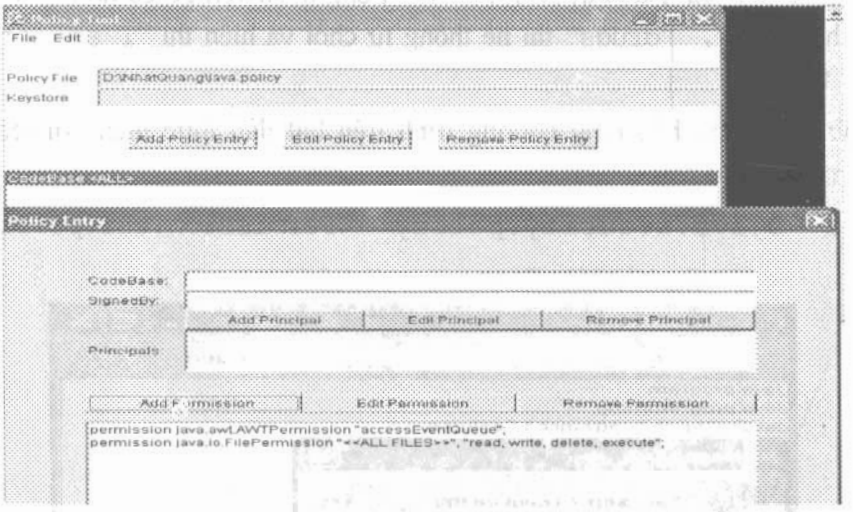
- Ta phải sử dụng "\\" thay cho "\" trong `Window`.

```
permission java.io.FilePermission "d:\\myapps\\-", "read, write, delete";
```

JDK cung cấp công cụ `policytool` để ta sử dụng mà soạn thảo các tệp chính sách. Khi bắt đầu sử dụng `policytool` ta có thể tạo ra một tệp chính sách mới, ví dụ tệp `java.policy` ở thư mục NhatQuang.

```
d:\NhatQuang\>policytool java.policy
```

`policytool` sẽ hiển thị tất cả các chức năng cho phép ta lựa chọn các đặc quyền, rồi gán cho chúng các đặc tính, hành động như trên đã nêu. Ta có thể bổ sung thêm những đặc quyền mới, sửa chữa chúng hoặc loại bỏ đi những đặc tính không có nhu cầu, v.v., như hình 7. 4.



Hình 7.4. **Soạn thảo tệp chính sách với `policytool`**

Ví dụ 7.4. Xây dựng một lớp để kiểm soát việc ghi chèn thêm các từ vào vùng văn bản. Chương trình phải đảm bảo rằng những từ "xấu", như các từ "sex", "drugs", v.v., không được phép ghi chèn vào văn bản. Để thực hiện được yêu cầu này, ta có thể xây dựng lớp cấp quyền tuỳ biến, lớp `WordCheckTextArea` như sau.

```
class WordCheckTextArea extends JTextArea{
   public void append(String text){
      WordCheckPermission p =
           new WordCheckPermission(text, "insert");
      SecurityManager manager = System.getSecurityManager();
      if (manager != null) manager.checkPermission(p);
      super.append(text);
   }
}
```

Nếu bộ quản lý đảm bảo được an ninh cho `WordCheckTextArea` thì những từ được chấp nhận sẽ được phép chèn vào văn bản.

Việc kiểm tra các từ với hai hành động cụ thể: `insert` (được phép chèn) và `avoid` (cho phép bổ sung một văn bản bất kỳ nhưng loại bỏ những từ "xấu", bị cấm). Do vậy, chương trình trên phải thực hiện với tệp `.policy` sau:

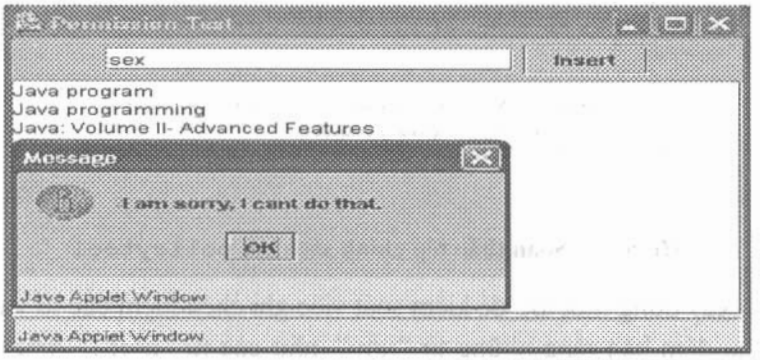
```
grant{
  permission WordCheckPermission "sex, drugs", "avoid";
};
```

cho phép chèn thêm các từ khác với những từ được liệt kê: `"sex", "drugs"`.

Như vậy, giao diện của chương trình trên có dạng như hình 7.5. Người sử dụng nhập vào các dòng văn bản và nhấn nút `Insert`. Nếu trong đoạn văn đó có những từ không được phép nhập như `"sex", "drugs"` thì hệ thống từ chối và hiển thị `"I am sorry, but I cannot do that."`.

Lưu ý: Phải đảm bảo rằng chương trình trên bắt đầu thực hiện với tệp chính sách `PermissionTest.policy`.

```
java -Djava.security.policy = PermissionTest.policy
     PermissionTest
```



Hình 7.5. **Chương trình kiểm duyệt các từ được chèn vào văn bản**

```
// PermissionTest.java
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.net.*;
import java.security.*;
import javax.swing.*;
```

```
public class PermissionTest{
   public static void main (String args[]){
      System.setSecurityManager(new SecurityManager());
      JFrame frame = new PermissionTestFrame();
      frame.show();
   }
}
class PermissionTestFrame extends JFrame{
   public PermissionTestFrame(){
      setTitle("Permission Test");
      setSize(400, 300);
      addWindowListener(new WindowAdapter(){
            public void windowClosing (WindowEvent e){
                  System.exit(0);
            }
      });
      textField = new JTextField(20);
      JPanel panel = new JPanel();
      panel.add(textField);
      JButton openButton = new JButton("Insert");
      panel.add(openButton);
      openButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                  insertWords(textField.getText());
            }
      });
      Container contentPane = getContentPane();
      contentPane.add(panel, "North");
      textArea = new WordCheckTextArea();
      contentPane.add(new JScrollPane(textArea), "Center");
   }
   private JTextField textField;
   private WordCheckTextArea textArea;
```

```
    public void insertWords (String words){
        try{
            textArea.append(words + "\n");
        }
        catch (SecurityException e){
          JOptionPane.showMessageDialog(this, "I am sorry, I cant do that.");
        }
  }
}

class WordCheckTextArea extends JTextArea{
 public void append(String text){
        WordCheckPermission p = new WordCheckPermission(text, "insert");
        SecurityManager manager = System.getSecurityManager();
        if (manager != null) manager.checkPermission(p);
        super.append(text);

  }
}
// WordCheckPermission.java
import java.security.*;
import java.util.*;

public class WordCheckPermission extends Permission{
 private String action;
 public WordCheckPermission(String target, String anAction){
        super(target);
        action = anAction;
 }

 public String getActions(){
        return action;
 }
```

```
public boolean equals(Object other){
    if (other == null) return false;
    if (!getClass().equals(other.getClass())) return false;
    WordCheckPermission b = (WordCheckPermission)other;
    if (!action.equals(b.action)) return false;
    if (action.equals("insert")) return getName().equals(b.getName());
    else if (action.equals("avoid"))
          return badWordSet().equals(b.badWordSet());
    else return false;
}
public int hashCode(){
    return getName().hashCode() + action.hashCode();
}

public boolean implies(Permission other){
  if (!(other instanceof WordCheckPermission)) return false;
  WordCheckPermission b = (WordCheckPermission)other;
  if (action.equals("insert")){
          return b.action.equals("insert")
                && getName().indexOf(b.getName()) >=0;
  }
  else if (action.equals("avoid")){
          if (b.action.equals("avoid"))
          return b.badWordSet().containsAll(badWordSet());
          else if (b.action.equals("insert")){
                Iterator iter = badWordSet().iterator();
                while (iter.hasNext()){
                  String badWord = (String)iter.next();
                  if (b.getName().indexOf(badWord) >= 0)
                      return false;
                }
                return true;
          }
          else return false;
```

```
        }
        else return false;
    }
    public Set badWordSet(){
        StringTokenizer token =
            new StringTokenizer(getName(), ",");
        Set set = new HashSet();
        while (token.hasMoreTokens())
            set.add(token.nextToken());
        return set;
    }
}
```

7.3.3. Bộ quản lý *bảo mật* tùy biến

Trong phần này ta xây dựng bộ quản lý *bảo mật* tuỳ biến đơn giản, đó là lớp `CheckSecurityManager`. Nó kiểm soát việc truy cập vào tệp, đảm bảo rằng những tệp văn bản (`.text`) mà nội dung của nó có các từ bị cấm thì không được phép truy cập.

Ta kiểm soát truy cập vào tệp bằng cách viết đè phương thức `checkPermission()` của lớp `SecurityManager`.

Ví dụ 7.5. Chương trình kiểm duyệt đọc các nội dung của các tệp. Mở một tệp `.text` và đọc nội dung của nó. Nếu nội dung của tệp văn bản không chứa các từ bị cấm thì cho phép truy cập.

```
// CheckSecurityManager.java
import java.security.*;
import java.io.*;
public class CheckSecurityManager extends SecurityManager{
    private String[] badWords = {"sex", "drugs", "C++"};

    public void checkPermission(Permission p){
        if(p instanceof FilePermission
                && p.getActions().equals("read")){
            if(inSameManager()) return;
            String fileName = p.getName();
            if(containsBadWords(fileName)) throw
```

```
                new SecurityException("Bad words in " + fileName);
        }
        else super.checkPermission(p);
    }
    boolean inSameManager(){
        Class[] cc = getClassContext();
        // Bỏ qua tập các lời gọi tới bộ quản lý
        int i = 0;
        while(i < cc.length && cc[0] == cc[i])
                i++;
        // Kiểm tra xem có lời gọi khác tới bộ quản lý không
        while(i < cc.length){
                if(cc[0] == cc[i]) return true;
                i++;
        }
        return false;
    }

    boolean containsBadWords(String fileName){
        if (!fileName.toLowerCase().endsWith("text")) return false;
        // Chỉ kiểm tra tệp .text
        BufferedReader in = null;
        try{
                in = new BufferedReader(new FileReader(fileName));
                String s;
                while((s = in.readLine()) != null){
                        for(int i = 0; i < badWords.length; i++)
                         if(s.toLowerCase().indexOf(badWords[i]) != -1)
                                return true;
                }
                in.close();
                return false;
        }catch(IOException e){
                return true;
```

```
        }finally{
            if(in != null)
              try{in.close();
              }catch(IOException e){
              }
        }
    }
}
// SecurityManagerTest.java
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.net.*;
import javax.swing.*;

public class SecurityManagerTest{
    public static void main (String args[]){
        System.setSecurityManager(new CheckSecurityManager());
        JFrame frame = new SecurityManagerTestFrame();
        frame.show();
    }
}
class SecurityManagerTestFrame extends JFrame{
    private JTextField fileNameField;
    private JTextArea fileText;
    public SecurityManagerTestFrame(){
        setTitle("Security Manager Test");
        setSize(400, 300);
        addWindowListener(new WindowAdapter(){
            public void windowClosing (WindowEvent e){
                System.exit(0);
            }
        });
```

```
        fileNameField = new JTextField(20);
        JPanel panel = new JPanel();
        panel.add(new JLabel("Text File: "));
        panel.add(fileNameField);
        JButton openButton = new JButton("Open");
        panel.add(openButton);
        openButton.addActionListener(new ActionListener(){
              public void actionPerformed(ActionEvent e){
                    loadFile(fileNameField.getText());
              }
        });
        Container contentPane = getContentPane();
        contentPane.add(panel, "North");
        fileText = new JTextArea();
        contentPane.add(new JScrollPane(fileText), "Center");
    }

    public void loadFile(String fileName){
        try{
              fileText.setText("");
              BufferedReader in =
                 new BufferedReader(new FileReader(fileName));
              String s;
              while((s = in.readLine()) != null)
                    fileText.append(s + "\n");
              in.close();
        }catch(IOException e){
             fileText.append("I am sorry, I cant do that.");
        }
    }
}
```

Để tách tệp `CheckSecurityManager.class` ra khỏi các tệp lớp khác, ta có thể tạo ra một tệp `.jar` để lưu giữ tệp đó.

```
jar cvf  FileCheck.jar CheckSecurityManager.class
```

Lệnh này đồng thời xóa `CheckSecurityManager.class` khỏi thư mục hiện thời.

Tương tự như ví dụ trước, để chạy được chương trình trên thì phải tạo ra tệp chính sách, tệp `CheckSecurity.policy` dạng:

```
grant codeBase "file:FileCheck.jar"{
  permission java.security.AllPermission;
};
```

Chính sách này đảm bảo các lớp trong `FileCheck.jar` được thực hiện hầu như tất cả mọi đặc quyền.

Sau cùng, ta thực hiện chương trình nêu trên như sau:

```
java -Djava.security.policy= CheckSecurity.policy
  -classpath FileCheck.jar;. CheckSecurityManager.java
```

java.lang.SecurityManager **API**

- `Class[] getClassContext()`

  Trả lại một mảng các lớp cho các phương thức đang thực hiện.

- `void checkCreateClassLoader()`

  Kiểm tra xem luồng hiện thời có tạo ra một bộ nạp lớp hay không.

- `void checkAccess(Thread g)`

  Kiểm tra xem luồng hiện thời có thể gọi các phương thức `stop()`, `suspend()`, `resume()`, `setName()`, `setDaemon()` của luồng `g`.

- `void checkExit(int status)`

  Kiểm tra xem luồng hiện thời có thể thoát khỏi JVM với trạng thái `status`.

- `void checkExec(String cmd)`

  Kiểm tra xem luồng hiện thời có thể thoát khỏi JVM với trạng thái `status`.

- `void checkRead(FileDescriptor fd)`
- `void checkRead(String file)`
- `void checkWrite(FileDescriptor fd)`
- `void checkWrite(String file)`
- `void checkDelete(String file)`

  Kiểm tra xem luồng hiện thời có thể đọc, ghi, xoá tệp.

7.4. VẤN ĐỀ *BẢO MẬT* TRONG GÓI JAVA.SECURITY

Trong hơn 50 năm qua, các nhà toán học, tin học đã phát triển nhiều thuật toán rất tinh tế đảm bảo tính toàn vẹn của dữ liệu và chữ ký điện tử. Gói `java.security` đã cài đặt khá nhiều trong số các thuật toán đó. Rất may là ta không cần phải hiểu tường tận cơ sở toán học mà vẫn có thể sử dụng được chúng. Sau đây chúng ta tìm hiểu cách để phát hiện ra những sự thay đổi trong các tệp dữ liệu và cách xác thực chữ ký số.

7.4.1. Dấu vết thông điệp

Tóm tắt đặc trưng của thông điệp là một dạng dấu vết tóm tắt của một khối dữ liệu, một thông điệp, gọi tắt là bản dấu vết thông điệp (`message digest`). Mỗi bản dấu vết thông điệp sẽ có hai đặc tính:

1. Khi bản lưu trữ dữ liệu có một hay một số `bit` bị thay đổi thì bản tóm tắt dấu vết cũng sẽ bị thay đổi,
2. Rất khó (hầu như không thể) kiến thiết được một thông điệp mới (một văn bản) có cùng một bản tóm tắt dấu vết như bản gốc.

Đặc tính thứ hai có thể thực hiện được một cách ngẫu nhiên, nhưng với một xác suất rất thấp.

Hiện nay có một số thuật toán nén dữ liệu để tạo ra bản dấu vết thông điệp. Hai thuật toán nổi tiếng nhất là `SHA-1` (do National Insstitute of Standard and Tectnology đề xuất) nén một một khối dữ liệu bất kỳ thành dãy 160 `bit` (20 `byte`) và `MD5` (của Ronald Rivest ở MIT) [2].

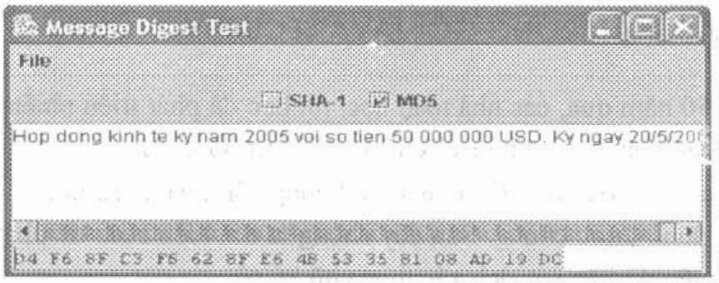
Java cài đặt cả hai thuật toán trên. Lớp `MessageDigest` được xem như một cái máy sản xuất ra các đối tượng thực hiện các thuật toán xử lý các tóm tắt dấu vết. Ví dụ, để nhận được đối tượng thực hiện thuật toán `SHA-1`, ta viết

```
MessageDigest alg = MessageDigest.getInstance("SHA-1");
```

Sau đó truyền các `byte` của một thông điệp cho đối tượng `alg` để nó tạo ra bản tóm tắt dấu vết. Ví dụ, muốn tạo ra bản tóm tắt dấu vết cho tệp có tên là `fileName`, ta thực hiện như sau.

```
FileInputStream in = new  FileInputStream(fileName);
int ch;
while((ch = in.read() != -1)
      alg.update((byte)ch);
```

Ví dụ 7.6. Chương trình sử dụng thuật toán `SHA-1` hoặc `MD5` để tạo ra bản tóm tắt dấu vết của một tệp bất kỳ hoặc một thông điệp được gõ trực tiếp từ bàn phím.



Hình 7.6. **Chương trình sinh bản tóm tắt dấu vết thông điệp**

Người sử dụng có thể nhập vào một đoạn văn bản như: "Hop dong kinh te ky nam 2005 voi so tien 500 000 000 USD. Ky ngay 20/5/2005", chọn thuật toán cần thực hiện, sau đó chọn File rồi Text area digest thì bản tóm tắt dấu vết tương ứng sẽ được sinh ra ở hàng dưới của khung cửa sổ là: D4 F6 8F C3 F6 62 8F E6 4B 53 35 81 08 AD 19 DC (hình 7.6).

Nếu ta sửa đi một chữ nào, ví dụ chữa số 50 000 000 thành 20 000 000 thì bản dấu vết sẽ cho là: 39 17 7D 82 8F DD FD 94 54 72 64 1B F3 2F 51 89. Hai bản dấu vết là khác nhau, ta phát hiện ra đoạn văn bản trên đã bị sửa.

Tương tự, ta có thể chọn một tệp bất kỳ trong mục chọn File\File Digest để sinh ra bản dấu vết nhằm phát hiện ra những dữ liệu đã bị sửa.

```
// MessageDigestTest.java
import java.awt.*;
import java.awt.event.*;
import java.security.*;
import java.io.*;
import javax.swing.*;

public class MessageDigestTest{
   public static void main (String args[]){
      System.setSecurityManager(new CheckSecurityManager());
      JFrame frame = new MessageDigestFrame();
      frame.show();
   }
}
class MessageDigestFrame extends JFrame{
   private JTextField digest = new JTextField();
   private JTextArea message = new JTextArea();
   private MessageDigest currentAlgorithm;
```

```
public MessageDigestFrame(){
    setTitle("Message Digest Test");
    setSize(400, 200);
    addWindowListener(new WindowAdapter(){
        public void windowClosing (WindowEvent e){
            System.exit(0);
        }
    });

    JPanel panel = new JPanel();
    ButtonGroup group = new ButtonGroup();
    ActionListener listener =
        new ActionListener(){
            public void actionPerformed(ActionEvent e){
            JCheckBox b = (JCheckBox)e.getSource();
            setAlgorithm(b.getText());
        }
    };
    addCheckBox(panel, "SHA-1", group, true, listener);
    addCheckBox(panel, "MD5", group, false, listener);

    Container contentPane = getContentPane();

    contentPane.add(panel, "North");
    contentPane.add(new JScrollPane(message), "Center");
    contentPane.add(digest, "South");
    digest.setFont(new Font("Monospaced", Font.PLAIN, 12));

    setAlgorithm("SHA-1");      // Đặt mặc định, chọn SHA-1

    JMenuBar menuBar = new JMenuBar();
    JMenu menu = new JMenu("File");
    JMenuItem fileItem = new JMenuItem("File digest");
    fileItem.addActionListener(
        new ActionListener(){
```

```
            public void actionPerformed(ActionEvent e){
               loadFile();
            }
      });
      menu.add(fileItem);
      JMenuItem text = new JMenuItem("Text area digest");
      text.addActionListener(
            new ActionListener(){
               public void actionPerformed(ActionEvent e){
                  String m = message.getText();
                  computeDigest(m.getBytes());// Tính bản dấu vết
               }
      });
      menu.add(text);
      menuBar.add(menu);
      setJMenuBar(menuBar);
   }
   // Đọc nội dung của tệp và tính bản dấu vết
   public void loadFile(){
      JFileChooser chooser = new JFileChooser();
      chooser.setCurrentDirectory(new File("."));

      int r = chooser.showOpenDialog(this);
      if(r == JFileChooser.APPROVE_OPTION){
            String name =
              chooser.getSelectedFile().getAbsolutePath();
            computeDigest(loadBytes(name));
      }
   }
   public void addCheckBox(Container c, String name,
      ButtonGroup g, boolean selected, ActionListener listener){
            JCheckBox b = new JCheckBox(name, selected);
            c.add(b);
```

```
            g.add(b);
            b.addActionListener(listener);
    }
    // Xác định thuật toán theo lựa chọn của người sử dụng
    public void setAlgorithm(String alg){
        try{
            currentAlgorithm = MessageDigest.getInstance(alg);
            digest.setText("");
        }catch(NoSuchAlgorithmException e){
            digest.setText("" + e);
        }
    }
    public void computeDigest(byte[] b){
        currentAlgorithm.reset();
        currentAlgorithm.update(b);
        byte[] hash = currentAlgorithm.digest();
        String d = "";
        for(int i = 0; i < hash.length; i++){
            int v = hash[i] & 0xFF;
            if(v < 16) d += "0";
            d += Integer.toString(v, 16).toUpperCase() + " ";
        }
        digest.setText(d);
    }

    public byte[] loadBytes(String name){
        FileInputStream in = null;
        try{
            in = new FileInputStream(name);
            ByteArrayOutputStream buff = new ByteArrayOutputStream();
            int ch;
            while((ch = in.read()) != -1)
                buff.write(ch);
            return buff.toByteArray();
```

```
        }catch(IOException e){
            if(in != null){
                try{in.close();
                }catch(IOException e1){
                }
            }
            return null;
        }
    }
}
// MessageDigest.policy
grant{
    permission java.security.AllPermission;
};
```

Sau khi dịch và chạy chương trình:

```
java -Djava.security.policy=MessageDigest.policy MessageDigestTest
```

Lưu ý: Giống như dấu vân tay, hy vọng rằng hầu như sẽ không có hai thông điệp có cùng một bản tóm tắt dấu vết. Tuy nhiên, hy vọng đó không hiện thực về mặt lý thuyết, vì như trong thuật toán `SHA-1` "chỉ có" khoảng 2^{160} bộ tóm tắt dấu vết có thể tạo ra, nên vẫn có thể có một số thông điệp khác nhau cho cùng một bản tóm tắt dấu vết. Tuy nhiên, xác suất trùng lặp ở đây là rất thấp, nên ta có thể yên tâm sử dụng những thuật toán trên để phát hiện những thông tin bị thay đổi trong các thông điệp.

java.security.MessagDigest **API**

- `static MessageDigest getInstance(String alg)`
  Cho lại đối tượng của `MessageDigest` cài đặt thuật toán `alg`.
- `void update(byte input)`
- `void update(byte[] input)`
- `void update(byte[] input, int offset, int len)`
  Cập nhật lại bản dấu vết đối với các byte đầu vào.
- `void reset()`: Tính lại bản dấu vết.

7.4.2. Chữ ký số

Trong phần trước ta đã tìm hiểu cách sinh ra bản tóm tắt dấu vết cho một thông điệp. Khi một thông điệp bị thay đổi thì bản tóm tắt dấu vết được sinh ra sẽ không tương thích với

bản gốc, do vậy dễ dàng phát hiện ra là nó đã bị thay đổi. Nếu ta gửi đi một bản tin và bản tóm tắt dấu vết của nó một cách riêng biệt, người nhận có thể sử dụng bản dấu vết đó để kiểm tra xem bản tin mình nhận có phải là bản gốc hay không. Nhưng nếu những kẻ gian, bằng một cách nào đó có được cả bản tin và bản dấu vết, họ có thể dễ dàng sửa đổi bản tin, tính lại bản dấu vết tương ứng. Điều này hoàn toàn thực hiện được bởi vì các thuật toán tính bản dấu vết là công khai, hoàn toàn không có gì bí mật cả. Trong trường hợp này, người nhận bản tin đã bị sửa và khi tính bản dấu vết (cũng bị sửa theo) có thể sẽ không phát hiện được là bản tin đó đã bị sửa.

Để chống lại những sự tấn công nêu trên, ta có thể sử dụng thêm một số phương pháp bảo mật, xác thực khác, ví dụ sử dụng chữ ký số.

Trước tiên ta cần tìm hiểu về việc xác thực được thực hiện như thế nào. *Khi một thông điệp được xác thực*, nghĩa là

- Thông điệp đó không bị thay đổi
- Thông điệp này là đúng của người gửi.

Nếu như cả bên gửi và bên nhận không có sự không thống nhất nào về xuất xứ cũng như nội dung của thông điệp, thì việc trao đổi như vậy được xác nhận là hoàn tất. Cả hai bên đều tin rằng, không có một người thứ ba can thiệp vào quá trình trao đổi tin này.

Tuy nhiên, có những đoạn tin gian lận xuất phát từ bên gửi hoặc do bên nhận tự tạo ra trong các giao dịch thương mại, thanh toán, trao đổi trên mạng, v.v. Các tranh chấp có thể xảy ra và cũng có nhiều trường hợp người bị lừa khó mà nhận biết được, nếu không có biện pháp phòng ngừa và phát hiện hữu hiệu. Ngay cả khi dùng mật mã khóa bí mật mà cả hai bên gửi và nhận đều biết, người gửi có thể tự tạo thêm một mã để kiểm tra xem thông điệp nhận được có xác thực hay không.

Trong thực tế, các hoạt động thương mại, quản lý hành chính, hoạt động nghiệp vụ, các tài liệu trên giấy có giá trị cam kết giao hẹn với nhau (như ngân phiếu, hợp đồng) thì bên gửi là bên có khả năng làm giả nhiều nhất. Ngược lại, cũng có khi một số trường hợp phía bên nhận lại chối bỏ trách nhiệm của mình vì thấy những điều đó bất lợi cho mình. Trong các trường hợp đó, việc xác thực thường được dựa vào chữ ký của hai bên để xác nhận các điều khoản đã cam kết, giao kèo với nhau trên "giấy trắng mực đen", và đó cũng là cơ sở pháp lý để giải quyết khi có tranh chấp.

Nhưng nếu các hoạt động trên thực hiện trao đổi với nhau trên mạng truyền số liệu thì vấn đề phức tạp hơn nhiều. Ví dụ, nếu bên `B` mang đến Tòa án một tài liệu nhận được qua mạng truyền số liệu (`Internet`) và bên `A` lại chối bỏ trách nhiệm gửi của mình thì Tòa án cũng rất khó phân xử rạch ròi. Bởi vì cũng có khả năng bên `B` làm giả đoạn tin và cũng có khi bên `A` có gửi thật nhưng lại chối bỏ trách nhiệm.

Vấn đề đặt ra là làm thế nào để phân xử được trong những trường hợp như trên. Muốn giải quyết được vấn đề xác thực thì cần phải có một cơ chế nào đó giống như chữ ký tay để cả

hai bên gửi và nhận cùng kiểm tra và không thể tạo giả mạo chữ ký đó. Một trong các biện pháp để thực hiện xác thực là sử dụng *chữ ký số*.

Để hiểu rõ về chữ ký số, ta cần nói rõ hơn về một số khái niệm liên quan đến *mật mã khóa công khai*. Mật mã khóa công khai dựa trên hai khái niệm: *khóa công khai và khóa riêng*. Khóa công khai thì có thể công bố cho mọi người biết. Khóa riêng thì chỉ người giữ nó được biết và điều quan trọng là không để người khác biết. Hai khóa này có mối quan hệ với nhau theo các quan hệ toán học, nhưng có thể tin rằng, thực tế không thể từ một khóa này tìm ra được khóa kia. Mặc dù đã biết khóa công khai, nhưng cả đời người cũng không tìm ra được khóa riêng, cho dù ta có dùng bao nhiêu máy tính, với loại máy tính nào đi nữa cũng không thực hiện được [2]. Điều này có thể khó tin, song cho đến hiện nay chưa ai tìm được thuật toán để tính được khóa riêng từ khóa công khai trong thời gian chấp nhận được.

Đã có khá nhiều thuật toán được sử dụng để mã hoá và giải mã thông tin. Trong số đó thuật toán được nói tới nhiều là thuật toán RSA của Rivest, Shamir và Adleman. Thuật toán này được xây dựng dựa trên độ khó của việc phân tích các số lớn thành các thừa số, nhất là các thừa số nguyên tố. Phần lớn các nhà mật mã tin rằng những khóa với "Modulus" của số gồm 2000 bit hoặc lớn hơn thì hoàn toàn yên tâm đối với mọi sự tấn công từ bên ngoài.

Có hai cặp khóa công khai và khóa riêng được sử dụng để mã hoá và chứng thực. Nếu một người gửi cho bạn một bản tin được mã hoá bằng khóa công khai thì bạn có thể giải mã bằng khóa riêng để đọc bản tin đó mà những người khác không làm được. Ngược lại, bạn có thể "ký" (đánh dấu) vào một bản tin bằng khóa chứng thực riêng, sau đó người nhận có thể kiểm tra tính xác thực của chữ ký bằng khóa công khai mà bạn cung cấp. Việc kiểm chứng này chỉ cần thực hiện được đối với những bản tin đã được đánh dấu (được ký) và nó sẽ thất bại nếu ai đó sử dụng khóa của họ để đánh dấu vào bản tin đó, nghĩa là làm thay đổi bản tin đã được ký.

Ta hãy tìm hiểu cách hoạt động của thuật toán DSA để sinh ra cặp khóa công khai/riêng và cách sử dụng chúng như thế nào.

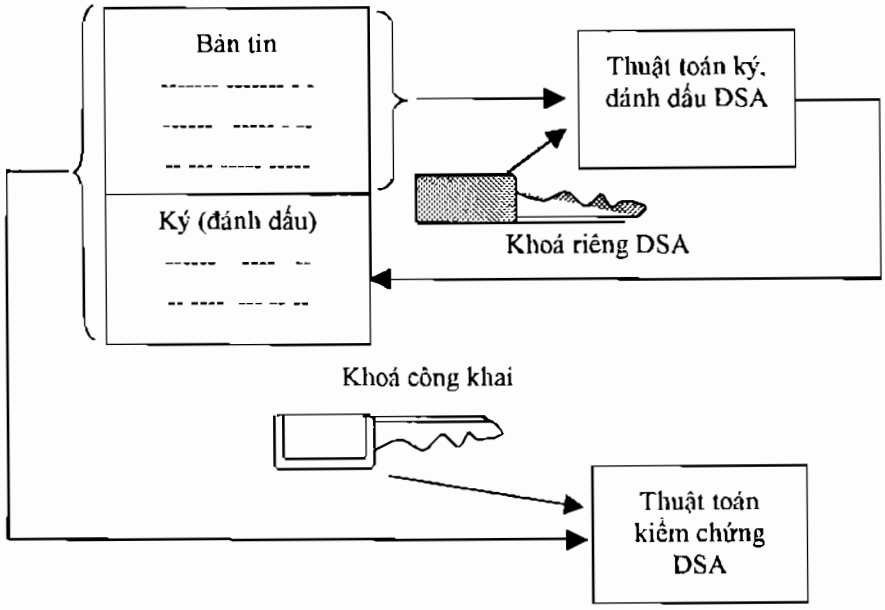
Giả sử rằng Nam muốn gửi cho Hoa một bản tin và Hoa muốn biết xem bản tin này có phải đúng do Nam gửi hay không. Nam viết bản tin rồi đánh dấu (tạo ra bản dấu vết) bằng khóa riêng của mình. Hoa nhận được bản copy của khóa công khai và sử dụng nó để kiểm tra tính xác thực của bản tin. Nếu việc kiểm chứng này thành công thì Hoa tin rằng:

1. Bản tin gốc không bị thay đổi.

2. Bản tin được ký bởi Nam.

Một chữ ký số cần phải phụ thuộc vào tất cả các bit của đoạn tin với mục đích là giữ sự bền vững của đoạn tin đó. Điều quan trọng ở đây là, phải đảm bảo không một ai có thể làm thay đổi nội dung của bản tin rõ trong lúc phải giữ nguyên chữ ký số.

Quá trình trao đổi và xác thực theo thuật toán DSA được mô tả như sau:



Hình 7.7. **Sự trao đổi giữa các khóa của thuật toán DSA**

Tuy nhiên, về mặt lý thuyết sẽ không có một giải pháp nào là tuyệt đối an toàn. Do vậy, chữ ký số cũng không phải là giải pháp toàn năng để chống lại sự giả mạo và có thể phân xử được mọi tranh chấp, cho nên cần có cơ chế trọng tài. Chữ ký số có thể kết hợp với mật mã để tăng tính hiệu quả của vấn đề xác thực.

Gói *bảo mật* của Java đã cài đặt hai thuật toán DSA, SHA-1 nêu trên. Nếu bạn muốn sử dụng RSA thì phải mua các lớp của RSA (www.rsa.com).

Phần tiếp theo chúng ta tập trung khai thác DSA. Có ba thuật toán được sử dụng.

1. Thuật toán sinh ra cặp khóa
2. Thuật toán ký (đánh dấu) thông điệp
3. Thuật toán xác thực chữ ký.

1. Tạo ra cặp khóa

Để sinh ra những cặp khóa một cách hoàn toàn ngẫu nhiên, người ta thường sử dụng bộ sinh số ngẫu nhiên của lớp SecureRandom. Lớp này phát sinh các số ngẫu nhiên hơn (đảm bảo an ninh hơn) các số sinh bởi Random. Ví dụ,

```
SecureRandom secrand = new SecureRandom();
byte[] b = new byte[20];
secrand.setSeed(b);                    // Điền các bít một cách ngẫu nhiên
```

Trong Java, để tính một cặp khóa theo thuật toán DSA mới, ta sử dụng lớp KeyPairGenerator như sau:

```
KeyPairGenerator keygen = KeyPairGenerator.getInstance("DSA");
```

Nếu muốn sinh ra một khóa theo modulus của 512 bit thì hãy viết

```
keygen.initialize(512, secrand);
```

Sử dụng tiếp các lớp KeyPair, PublicKey, PrivateKey để sinh ra cặp khóa công khai và khóa riêng.

```
KeyPair keys = keygen.generateKeyPair();
PublicKey pubKey = keys.getPublic();
PrivateKey privKey = keys.getPrivate();
```

2. Đánh dấu (ký xác nhận) thông điệp

Muốn đánh dấu (ký xác nhận) một thông điệp (đoạn tin), ta phải tạo ra đối tượng của lớp Signature.

```
Signature sigalg = Signature.getInstance("DSA");
```

Đối tượng này được sử dụng cho cả việc ký xác nhận lẫn việc xác thực thông điệp. Trước khi ký chứng thực, ta sử dụng phương thức initSign() và truyền khóa riêng cho nó.

```
sigalg.initSign(privKey);
```

Tiếp theo, sử dụng phương thức update() để đưa các byte của thông điệp (đọc theo từng byte) vào đối tượng đánh dấu.

```
while((ch = in.read() != -1)
    sigalg.update((byte)ch);
```

Cuối cùng là chữ ký số.

```
byte[] signature = sigalg.sign();
```

3. Xác thực chữ ký

Người nhận được thông điệp cũng sẽ nhận được khóa công khai và cần phải kiểm tra xem tài liệu nhận được có đúng bản gốc của người gửi hay không.

```
Signature verifyAlg = Signature.getInstance("DSA");
verifyAlg.initVerify(pubKey);
while((ch = in.read()) != -1) //Truyền thông điệp vào đối tượng xác thực
    verifyAlg.update((byte)ch);
boolean check = verifyAlg.verify(signature); // Xác thực chữ ký
```

Nếu check là true thì chữ ký là xác thực, thông điệp nhận là đoạn tin đã được ký nhận bởi một khóa riêng tương ứng. Nghĩa là, cả người gửi và nội dung thông điệp được xác thực.

Ví dụ 7.6

/* SinatureTest.java: Chương trình sinh ra hai cặp khóa, sử dụng khóa riêng thứ nhất để ký nhận thông điệp gửi đi. Sau đó sử dụng hai khóa công khai để kiểm tra xem thông điệp nhận được có đúng được xác thực hay không?

*/

```
import java.security.*;
public class SignatureTest{
   public static void main (String args[]){
      try{
            // Tạo ra bộ sinh khóa ngẫu nhiên theo thuật toán DSA
            KeyPairGenerator keygen
                        = KeyPairGenerator.getInstance("DSA");
            SecureRandom secrand = new SecureRandom();
            keygen.initialize(512, secrand);
            // Sinh ra cặp khóa thứ nhất
            KeyPair keys1 = keygen.generateKeyPair();
            PublicKey pubkey1 = keys1.getPublic();
            PrivateKey privkey1 = keys1.getPrivate();
            // Sinh ra cặp khóa thứ hai
            KeyPair keys2 = keygen.generateKeyPair();
            PublicKey pubkey2 = keys2.getPublic();
            PrivateKey privkey2 = keys2.getPrivate();
            // Tạo ra đối tượng để ký xác nhận thông điệp gửi đi theo thuật toán DSA
            Signature signalg = Signature.getInstance("DSA");
            signalg.initSign(privkey1);
            String message = "Trả tác giả tiền thưởng là 100 000.";
            System.out.println("Thông điệp gửi đi: " + message);
            signalg.update(message.getBytes());
            byte[] signature = signalg.sign();
            // Kiểm tra tính xác thực theo khóa công khai thứ nhất
            Signature verifyalg = Signature.getInstance("DSA");
            verifyalg.initVerify(pubkey1);
            verifyalg.update(message.getBytes());
            System.out.println("Thông điệp nhận được: " + message);
```

```
            if(!verifyalg.verify(signature))
                System.out.println("Not ");
            System.out.println("Signed with private key 1");
            // Kiểm tra tính xác thực theo khóa công khai thứ hai
            verifyalg.initVerify(pubkey2);
            verifyalg.update(message.getBytes());
            if(!verifyalg.verify(signature))
                System.out.print("Not ");
            System.out.println("Signed with private key 2");
        }
        catch(Exception e){
            System.out.println("Error: " + e);
        }
    }
}
```

Chương trình trên chạy và cho kết quả:

```
Thông điệp gửi đi: Trả tác giả tiền thưởng là 100 000.
Thông điệp nhận được: Trả tác giả tiền thưởng là 100 000.
Signed with private key 1
No Signed with private key 2
```

java.lang.KeyPairGenerator API

- `static KeyPairGenerator getInstance(String alg)`

  Trả lại một đối tượng của `KeyPairGenerator` để sinh ra các cặp khóa theo thuật toán `alg`.

- `void initialize(int strleng, SecureRandom random)`

  Khởi tạo đối tượng hiện thời (đối tượng sinh cặp khóa) với số bit là `strleng` và đối tượng ngẫu nhiên `random`.

- `KeyPair generateKeyPair()`

  Sinh ra cặp khóa mới.

java.lang.KeyPair API

- `PrivateKey getPrivate()`

  Nhận lại khóa riêng của cặp khóa được sinh ra.

- `PublicKey getPublic()`

  Nhận lại khóa công khai của cặp khóa được sinh ra.

java.lang.Signature API

- `static Signature getInstance(String alg)`

  Nhận đối tượng chữ ký của `Signature` được cài đặt theo thuật toán `alg`.
- `void initSign(PrivateKey privKey)`

  Khởi động việc ký thông điệp theo khóa riêng.
- `void update(byte input)`
- `void update(byte[] input)`
- `void update(byte[] input, int offset, int len)`

  Cập nhật bộ đệm thông điệp theo các byte.
- `byte[] sign()`

  Hoàn tất việc ký xác nhận và trả lại đối tượng chữ ký.
- `void initVerify(PublicKey publicKey)`

  Khởi động đối tượng kiểm tra theo khóa công khai.
- `booolean verify(byte[] signature)`

  Kiểm tra tính xác thực của chữ ký.

7.5. VẤN ĐỀ CHỨNG THỰC

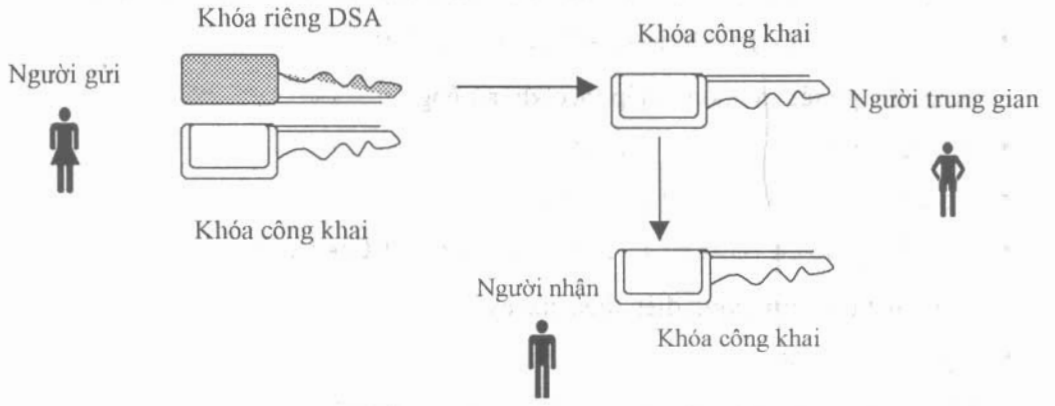
Như trên đã đề cập, nếu ta nhận một thông điệp được ký xác nhận bởi một người quen biết (người bạn) thì ta dễ dàng sử dụng khóa công khai được cung cấp để kiểm tra tính xác thực của thông điệp đó. Bây giờ ta xét tiếp trường hợp nhận được một tài liệu của một người không quen biết, người lạ. Tương tự như trên, nếu ta nhận được khóa cộng khai (vấn đề này không khó) thì vẫn kiểm chứng được xem tài liệu đó có sánh được với tài liệu được ký bằng khóa riêng tương ứng hay không.

Tuy nhiên, ở đây có vấn đề ta cần phải thận trọng vì ta *không có một khái niệm gì về người gửi*. Bất kỳ một người nào đó có thể tạo ra một cặp khóa, ký nhận vào tài liệu và gửi nó cho chúng ta. Vấn đề xuất hiện ở đây là, việc *xác định danh tính của người gửi hay còn gọi vấn đề chứng thực* được thực hiện như thế nào?.

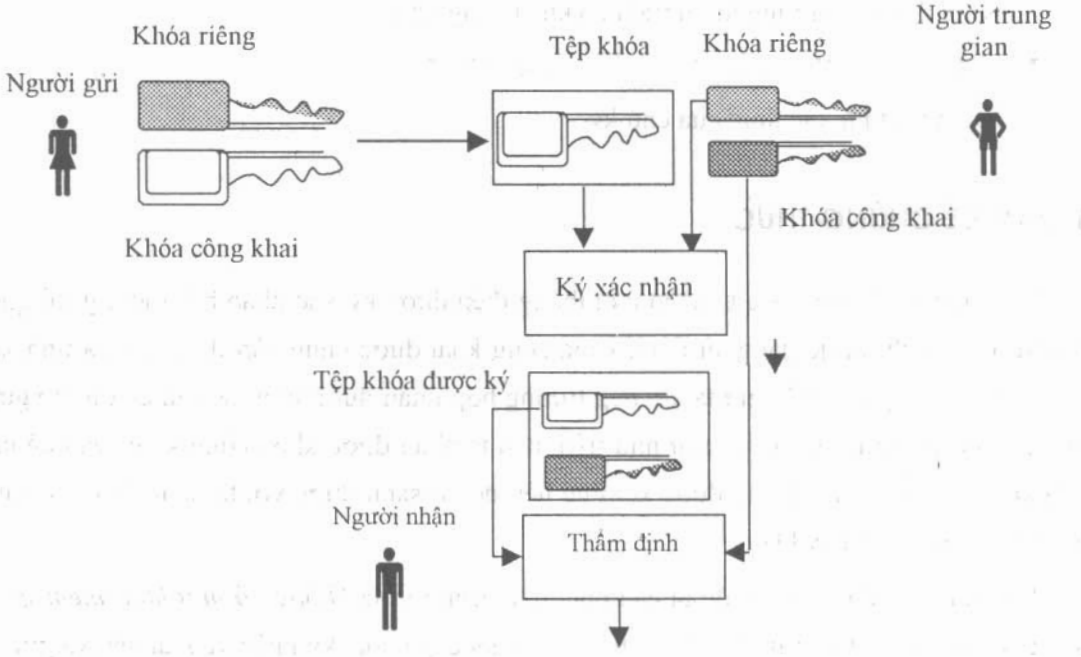
Cách giải quyết vấn đề trên trong thực tế thường là thông qua một đối tượng trung gian. Giả sử cả người gửi (là người không quen biết trước) và bạn có một người "quen" trung gian

(người, hay một tổ chức môi giới, một cơ quan có thẩm quyền). Người gửi gặp người trung gian trao tài liệu và khóa công khai, sau đó anh ta trao lại cho bạn (xem hình 7.8).

Trong thực tế, người trung gian cũng không cần gặp cả người gửi lẫn người nhận tài liệu. Anh ta có thể sử dụng một khóa riêng của mình để ký tiếp vào tài liệu của người gửi và gửi nó với khóa công khai cho người nhận (xem hình 7.9). Người nhận sẽ sử dụng cả hai khoá công khai nhận được xác thực tính trung thực của tài liệu nhận được.



Hình 7.8. **Vấn đề chứng thực thông qua trung gian**



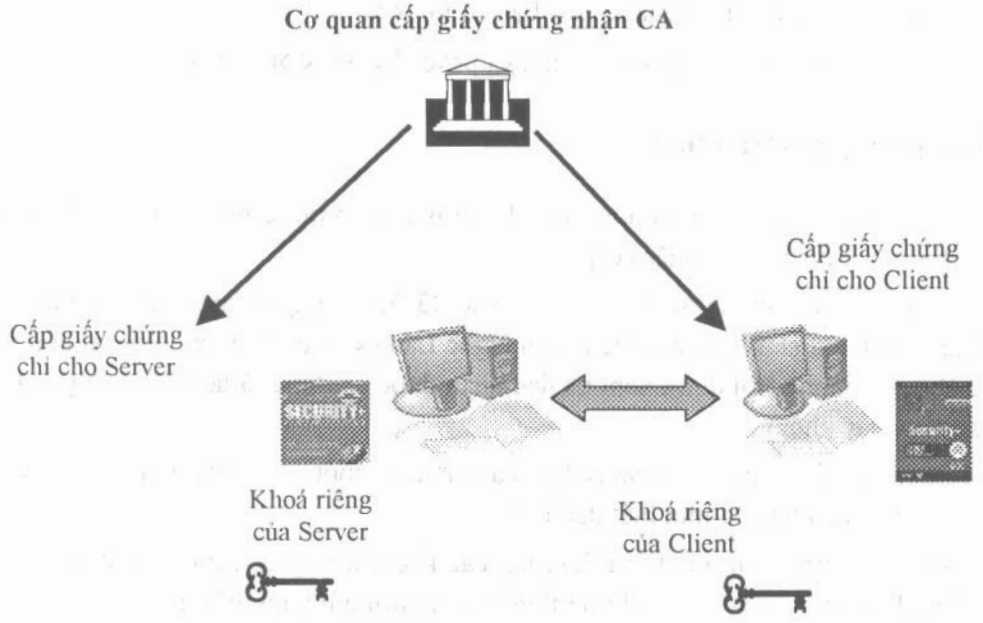
Hình 7.9. **Vấn đề xác thực thông qua chữ ký của người trung gian**

Song, vấn đề giải quyết như thế nào khi không có người môi giới trung gian. Khi đó ta có thể sử dụng giấy chứng nhận để chứng thực danh tính của người gửi.

7.5.1. Giấy chứng nhận Certificate X.509

Tương tự như trong thực tế, mỗi xí nghiệp được cấp một giấy phép sản xuất hay kinh doanh, chúng ta cũng có thể tạo ra các giấy chứng nhận để xác thực trong quá trình trao đổi tin. Giấy chứng nhận được một cơ quan, một tổ chức có thẩm quyền (được luật pháp thừa nhận) CA cấp, những người được cấp sẽ được xác thực và dựa vào giấy chứng nhận đó để bảo mật tin (mã hoá tin), rồi gửi đi. Những người nhận cũng được cấp một giấy chứng nhận cho phép họ xác thực và giải mã những thông tin nhận được. Một trong các mẫu ký xác nhận được sử dụng phổ biến là mẫu X.509 được định nghĩa bởi ITU-T.

Khi một máy trạm (Client) kết nối với một máy chủ (Server) sử dụng SSL/TLS [4], thì cả máy Client và Server phải thoả thuận với nhau về thuật toán mật mã chung sẽ được sử dụng để mã hoá tin cần trao đổi. Quá trình xác thực trong SSL/TLS sử dụng X.509 được thực hiện như sau.



Hình 7.10. **Qui trình xác thực sử dụng X.509**

1. Server gửi chứng chỉ X.509 chứa khoá công khai cho các máy trạm cần trao đổi tin.
2. Máy trạm (Client) tạo ra các số bí mật ngẫu nhiên chiếm khoảng 48 byte và sử dụng khoá công khai của Server để mật mã số bí mật đó, rồi gửi nó cho Server.
3. Server sử dụng khoá riêng để giải mã số bí mật đã được mã hoá nhận được.
4. Cả Server và Client chia sẻ với nhau số bí mật đó và người khác không biết được. Từ số bí mật đó phát sinh ra các khoá đối xứng và bắt đầu trao đổi tin với nhau.

Ngoài ra giấy chứng nhận X.509 được sử dụng rộng rãi bởi Microsoft, Netscape và nhiều hãng khác để ký xác nhận các thư điện tử E-mail, chứng thực mã chương trình và nhiều loại dữ liệu khác. Ở dạng đơn giản nhất, X.509 chứa các thông tin sau:

- Phiên bản (Version) của bằng chứng thực,
- Số hiệu của bằng chứng thực,
- Định danh của thuật toán ký xác nhận (tên gọi và các tham số),
- Tên người ký,
- Thời gian có hiệu lực (ngày bắt đầu và kết thúc),
- Định danh của người được chứng thực,
- Khóa công khai của giấy chứng nhận,
- Chữ ký (mã băm cho các trường dữ liệu, mã hoá chữ ký bằng khóa riêng).

Chi tiết hơn về cấu trúc của X.509 bạn có thể xem

http://www.ietf.cnri.reston.va.us/ids.by.wg/x.509.html

7.5.2. Lớp giấy chứng nhận

Thông thường các giấy chứng chỉ (chứng nhận) số được một tổ chức CA có đủ thẩm quyền (được luật pháp công nhận) cấp.

Trong JDK có chương trình keytool.exe để lập ra giấy chứng nhận và quản lý chúng. Các phần tiếp theo, chúng ta tìm hiểu cách Ngọc Lan ký xác nhận vào một tài liệu và gửi nó cho Lê Ba; người nhận kiểm tra xem tài liệu nhận được có đúng là tài liệu do Ngọc Lan ký và có bị sửa đổi hay không?.

Chương trình keytool quản lý kho các khóa và một CSDL các giấy chứng nhận. Mỗi mục trong kho các khóa có một biệt danh (alias).

Sau đây ta tìm hiểu quá trình thiết lập các kho khóa, giấy chứng nhận và trao đổi các thông điệp đã được ký xác nhận, chứng thực giữa người gửi và người nhận.

Ví dụ 7.7. Ngọc Lan tạo ra một kho các khóa riêng ngoclan.store và sinh ra một cặp khóa với biệt danh là ngoclan như sau:

```
keytool -genkey -keystore ngoclan.store -alias ngoclan
```

Khi sinh ra một khóa, hệ thống nhắc ta nhập các thông tin:

```
Enter keystore password: password
What is your first and last name?
      [Unknown]: Ngoc Lan
What is the name of your organization unit?
      [Unknown]: Engineering Department
What is the name of your organization?
      [Unknown]: Information Technology Institute
```

```
What is the name of your city or Locality?
    [Unknown]: Ha Noi
What is the name of your state or province?
    [Unknown]: Ha Noi
What is the two-letter country code for this unit?
[Unknown]: VN
Is <CN = Ngoc Lan, OU = Engineering Department, O = Information
Technology Institute, L = Ha Noi, ST = Ha Noi, C = VN> correct?
[no]: Y
```

keytool sử dụng các tên gọi phân biệt của X.509: CN (Common Name), OU (Organization Unit), O (Organization), L (Location), ST (State), và C (Country) để xác định khóa và cấp giấy chứng nhận. Lưu ý mật khẩu phải chứa ít nhất là 6 ký tự và ta phải nhớ để khai báo khi cần thiết về sau.

Giả sử Ngọc Lan muốn đưa khóa công khai cho Lê Ba thì phải đưa ra dưới dạng tệp:

```
keytool -export -keystore ngoclan.store -alias ngoclan -file ngoclan.cert
```

Hệ thống sẽ hỏi mật khẩu:

```
Enter keystore password: password
```

Nếu nhập đúng mật khẩu thì hệ thống sẽ tạo ra tệp ngonlan.cert chứa bằng chứng nhận của Ngọc Lan.

Sau đó Ngọc Lan có thể gửi tệp ngoclan.cert cho Lê Ba. Khi Lê Ba nhận được có thể in ra để biết được các thông tin cần thiết:

```
keytool -printcert -file ngoclan.cert
```

Kết quả nhận được trên màn hình:

```
Owner: CN = Ngoc Lan, OU = Engineering Department, O = Information
Technology Institute, L = Ha Noi, ST = Ha Noi, C = VN
Issuer: CN = Ngoc Lan, OU = Engineering Department, O = Information
Technology Institute, L = Ha Noi, ST = Ha Noi, C = VN
Serial number: 38107867
Valid from: Fri Aug 22 07:44:55 ITC 2005 until: Thu Nov 20
06:44:55 ITC 2005
Certicate fingerprints:
 MD5: 5D:00:0F:95:01:30:B4:FE:24:CE:98:34:F0:C8:90:BB
 SHA1:F4:10:2F:34:01:AC:B8:ED:15:E6:A8:01:89:75:88:CB:09:B2:A1:65
```

Giấy chứng nhận này nó tự xác nhận, vì vậy Lê Ba không thể dùng giấy chứng nhận khác để kiểm tra tính hợp lệ của bằng chứng nhận này được. Tuy nhiên, Lê Ba có thể hỏi

Ngọc Lan (qua điện thoại chẳng hạn) để biết được dấu vết số (fingerprint) của giấy chứng nhận để đối sánh.

Sau khi nhận được giấy chứng nhận, Lê Ba có thể chuyển vào kho riêng của mình, ví dụ leba.store như sau:

```
keytool -import -keystore leba.store -alias ngoclan -file ngoclan.cert
```

Hệ thống hỏi Lê Ba về mật khẩu nhập vào và sau đó hiển thị lại những thông tin về giấy chứng nhận đã nêu trên, nếu thấy đúng thì nhấn Y hoặc y để khẳng định.

Sau khi hoàn tất, Ngọc Lan gửi các tài liệu đã được ký xác nhận cho Lê Ba. Bộ công cụ jarsigner của Java thực hiện ký và kiểm định các tệp JAR. Ngọc Lan chỉ cần đưa tài liệu cần ký xác nhận document.txt vào tệp JAR: document.jar.

```
jar  cvf document.jar  document.txt
```

Sau đó Ngọc Lan sử dụng jarsigner để đưa chữ ký vào tệp document.jar.

```
jarsigner -keystore ngoclan.store document.jar ngoclan
```

Ngọc Lan phải nhập đúng mật khẩu thì tài liệu document.txt sẽ được ký xác nhận.

Khi nhận được tệp tài liệu document.jar, Lê Ba có thể sử dụng tuỳ chọn -verify của chương trình jarsigner.

```
jarsigner -verify -keystore leba.store document.jar
```

Nếu tệp document.jar không bị xâm phạm và chữ ký được đối sánh thì jarsigner sẽ in ra kết quả khẳng định tệp JAR được xác thực.

```
jar verified
```

Ngược lại sẽ thông báo lỗi.

Cơ chế đóng gói và lưu trữ các tệp JAR sẽ được đề cập chi tiết ở phần sau.

Để tách tất cả các tệp được nén trong document.jar, Lê Ba có thể sử dụng

```
jar xf document.jar
```

trong đó có tệp document.txt nhận được từ Ngọc Lan.

7.5.3. Ký giấy chứng nhận

Phần trên ta đã xem xét cách sử dụng giấy chứng nhận "tự ký" để gửi khóa công khai cho các bộ phận khác. Song, người nhận phải được được đảm bảo rằng giấy chứng nhận này là hợp lệ.

JDK không cung cấp công cụ để ký giấy chứng nhận nêu trên. Phần tiếp theo chúng ta tìm hiểu cách viết chương trình Java để ký nhận giấy chứng nhận bằng khóa riêng từ kho các khóa.

Ví dụ 7.8. Viết chương trình ký giấy chứng nhận bằng khóa riêng lấy ra từ kho các kho đã được thiết lập từ trước (như ở ví dụ 7.7).

Trước tiên ta phải thiết lập đối tượng của KeyStore. Kho các khóa được tạo ra bởi keytool có kiểu là "JKS" và nguồn cung cấp là "SUN".

```
KeyStore store = KeyStore.getInstance("JKS", "SUN");
```

Tiếp theo là phải tải dữ liệu của kho store vào chương trình, tất nhiên là phải cho biết tên của luồng vào và mật khẩu (password). Lưu ý, mật khẩu thường (nên) tổ chức dưới dạng mảng (Array) các ký tự chứ không phải là xâu (String). JVM có thể lưu giữ các xâu một thời gian dài trước khi chúng bị dọn dẹp và do vậy các harker có thể xâm nhập vào các tệp trao đổi để phát hiện ra mật khẩu còn các đối tượng của Array sẽ bị xóa bỏ ngay sau khi chúng không cần để sử dụng tiếp ở giai đoạn sau.

```
InputStream in = new FileInputStream(ksName);
char[] password = readPassword(console, "Keystore password");
store.load(in, password);
Arrays.fill(password, ' ');      // Xoá mật khẩu
in.close();
```

Ta sử dụng getKey() tìm khóa riêng để ký. Hàm getKey() yêu cầu truyền alias và mật khẩu.

```
char[] keyPassword
      = readPassword(console, "Keystore password for " + alias);
PrivateKey privKey
      = (PrivateKey)store.getKey(alias, keyPassword);
Arrays.fill(keyPassword, ' ');
```

Chúng ta đã sẵn sàng để đọc giấy chứng nhận và ký xác nhận. Lớp CertificateFactory đọc giấy chứng nhận từ tệp đầu vào.

```
in = new FileInputStream(inName);
CertificateFactory factory
      = CertificateFactory.getInstance("X.509");
```

Sau đó gọi hàm generateCertificate() để sinh ra giấy chứng nhận tương ứng.

```
X509Certificate inCert
      = (X509Certificate)factory.generateCertificate(in);
in.close();
```

Mục đích của chương trình là ký các byte trong giấy chứng nhận. Ta tìm các byte bằng hàm getTBSCertificate().

```
byte[] inCertBytes = inCert.getTBSCertificate();
```

Giấy chứng nhận đã được ký cần lưu vào tệp

```
X509CertInfo info = new X509CertInfo(inCertBytes);
info.set(X509CertInfo.ISSUER,
      new CertificateIssuerName((X500Name)issuer));
X509CertImpl outCert = new X509CertImpl(info);
outCert.sign(privKey, issuerAlg);
outCert.derEncode(out);
```

Lưu ý, ta sử dụng gói thư viện sun.security.x509 để sinh ra các giấy chứng nhận, cài đặt các thuật toán đảm bảo an ninh dữ liệu như RSA, v.v.

```
// CertificateSigner.java
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import sun.security.x509.X509CertImpl;
import sun.security.x509.X509CertInfo;
import sun.security.x509.X500Name;
import sun.security.x509.CertificateIssuerName;

public class CertificateSigner{ ,
   public static void main (String args[]){
      String ksName = null;       // Tên của kho
      String alias = null;        // Bí danh của khóa riêng
      String inName = null;       // Tên của tệp đầu vào
      String outName = null;      // Tên của tệp đầu ra
      for(int i = 0; i < args.length; i += 2){
           if(args[i].equals("-keystore"))
                ksName = args[i+1];
           else if(args[i].equals("-alias"))
                alias = args[i+1];
           else if(args[i].equals("-infile"))
                inName = args[i+1];
           else if(args[i].equals("-outfile"))
                outName = args[i+1];
```

```
            else usage();
        }
        if(ksName == null || alias == null || inName == null
                || outName == null) usage();

        try{
            PushbackReader console =
             new PushbackReader(new InputStreamReader(System.in));

            KeyStore store = KeyStore.getInstance("JKS", "SUN");
            InputStream in = new FileInputStream(ksName);
            char[] password = readPassword(console, "Keystore password");

            store.load(in, password);
            Arrays.fill(password, ' ');      // Xoá mật khẩu
            in.close();

            char[] keyPassword
             = readPassword(console, "Keystore password for " + alias);
            PrivateKey privKey
                 = (PrivateKey)store.getKey(alias, keyPassword);
            Arrays.fill(keyPassword, ' ');

            if(privKey == null)
                 error("No such private key");

            in = new FileInputStream(inName);
            CertificateFactory factory
               = CertificateFactory.getInstance("X.509");

            X509Certificate inCert
               = (X509Certificate)factory.generateCertificate(in);
            in.close();
            byte[] inCertBytes = inCert.getTBSCertificate();

            X509Certificate issuerCert
                 = (X509Certificate)store.getCertificate(alias);
```

```
            Principal issuer = issuerCert.getSubjectDN();
            String issuerAlg = issuerCert.getSigAlgName();

            FileOutputStream out = new FileOutputStream(outName);

            X509CertInfo info = new X509CertInfo(inCertBytes);
            info.set(X509CertInfo.ISSUER,
                  new CertificateIssuerName((X500Name)issuer));

            X509CertImpl outCert = new X509CertImpl(info);
            outCert.sign(privKey, issuerAlg);
            outCert.derEncode(out);

            out.close();
      }
      catch(Exception e){
            System.out.println("Error: " + e);
      }
}

public static char[] readPassword(PushbackReader in,
            String promp) throws IOException{
      System.out.print(promp + ": ");
      System.out.flush();
      final int MAX_PASSWORD_LENGTH = 100;
      int leng = 0;
      char[] buff = new char[MAX_PASSWORD_LENGTH];

      while(true){
            int ch = in.read();
            if(ch == '\r' || ch == '\n' || ch == -1 ||
                        ch == MAX_PASSWORD_LENGTH){
                  if(ch == '\r')       // DOS xem như kết thúc dòng "\r\n"
                  {
                        ch = in.read();
```

```
                    if(ch == '\n' && ch != -1) in.unread(ch);
                }
                char[] password = new char[leng];
                System.arraycopy(buff, 0, password, 0, leng);
                Arrays.fill(buff, ' ');
                return password;
            } else{
                buff[leng] = (char)ch;
                leng++;
            }
        }
    }

    public static void usage(){
        System.out.println("Usage: java CertificateSigner"
            + " -keystore keyStore - alias issuerKeyAlias"
            + " -infile inputFile -outfile outputFile");
        System.exit(1);
    }

    public static void error(String message){
        System.out.println(message);
        System.exit(1);
    }
}
```

Để sử dụng được chương trình trên, ta phải sử dụng một kho các khóa đã được thiết lập. Giả sử chúng ta tạo ra một kho tên là engsoft.store với biệt danh là engroot. Giấy phép được tạo ra và đưa vào tệp engroot.cert.

```
keytool -genkey -keystore engsoft.store -alias engroot
keytool -export -alias engroot -keystore engsoft.store
                -file engroot.cert
```

Tiếp theo, đưa vào kho các khóa của người nhận, ví dụ kho có tên lenguyen.store

```
keytool -import -alias engroot -keystore lenguyen.store
              -file engroot.cert
```

Chương trình đọc giấy chứng nhận từ tệp ngoclan.cert lấy khóa riêng từ kho engsoft.store và ký xác nhận rồi ghi lên tệp ngoclan_engisoft.cert.

```
java CertificateSigner -keystore engsoft.store -alias engroot
    -infile ngoclan.cert -outfile ngoclan_engisoft.cert
```

Ta phải nhập mật khẩu tương ứng để chương trình ký xác nhận vào giấy chứng nhận.

Ngọc Lan muốn gửi tài liệu cho những nhân viên trong phòng Engineering Department, ví dụ gửi tệp ngoclan_engisoft.cert cho Lê Nguyên.

java.security.KeyStore API

- `static getInstance(String type)`
- `static getInstance(String type, String provider)`

  Xác định đối tượng của kho các khóa. Nếu sử dụng `keytool` của Java thì `type` là "JKS" và `provider` là "SUN".
- `void load(InputStream in, char[] password)`

  Nhập một kho các khóa từ luồng vào `in` khi mật khẩu được chấp nhận.
- `Key getKey(String alias, char[] password)`

  Nhận lại một khóa riêng với biệt danh `alias` được lưu trong kho hiện thời.
- `Certificate getCertificate(String alias)`

  Nhận lại một giấy chứng nhận với biệt danh `alias` được lưu trong kho hiện thời.

java.security.cert.CertificateFactory API

- `CertificateFactory getInstance(String type)`

  Tạo ra một đối tượng để làm ra các giấy chứng nhận kiểu `type`. Kiểu sử dụng ở đây là "X509".
- `Certificate generateCertificate(InputStream in)`

  Sinh ra giấy chứng nhận từ luồng vào `in`.

java.security.cert.Certificate API

- `PublicKey getPublic()`

  Nhận lại khóa công khai được đính kèm với giấy phép.
- `byte[] getEncoded()`

  Đọc mã của giấy chứng nhận.
- `String getType()`

  Đọc kiểu của giấy chứng nhận, ví dụ "X509".

java.security.cert.X509Certificate API

- `Principal getSubjectDN()`
- `Principal getIssuerDN()`

  Xác định tên của người chứng nhận và người được cấp giấy chứng nhận.
- `Date getNotBefore()`
- `Date getNotAfter()`

  Xác định khoảng thời hợp lệ của giấy phép.
- `String getSigAlgName()`
- `byte[] getSignature()`

  Xác định thuật toán và chữ ký của giấy chứng nhận.
- `byte[] getTBCertificate()`

  Đọc thông tin được mã hoá cần thiết để ký giấy chứng nhận.

7.5.4. Ký nhận các tệp JAR

Trước tiên chúng ta đi tìm hiểu cách đóng gói và tổ chức các tệp lưu trữ JAR trong Java. Cơ chế gói gọn (package) là rất tiện lợi để tổ chức những lớp có liên quan với nhau thành các nhóm và nhất là trong quá trình phát triển phần mềm, ta cần phải đóng gói các tệp chương trình ứng dụng thành các gói để tiện lợi cho việc di chuyển và cài đặt ứng dụng. Chính các gói giúp chúng ta dễ dàng định vị, sử dụng các tệp lớp và hỗ trợ để điều khiển hoạt động truy cập vào dữ liệu của lớp lúc thực thi chương trình.

Khi chương trình đã được hoàn tất và chuyển sang giai đoạn thẩm định, kiểm tra để chuẩn bị triển khai, thì nên tổ chức thành các *tệp JAR*. Tệp *JAR* là dạng nén theo tệp `ZIP` và đóng gói những tệp thực thi (tệp lớp), cùng với những tệp ứng dụng liên quan sao cho chúng có thể được triển khai như là một đơn thể.

Các ưu điểm của tệp dạng JAR là:

- *Bảo mật*: Bạn có thể ký số vào nội dung của một tệp JAR. Người sử dụng nếu nhận dạng được chữ ký của bạn sẽ có cơ hội được đảm bảo an toàn hơn đối với những người khác.
- *Giảm thiểu thời gian chuyển tải*: Nếu các tệp của một chương trình được tổ chức thành một tệp JAR thì chúng sẽ được tải xuống trình duyệt trong cùng một giao dịch HTTP mà không cần tạo ra các kết nối cho từng tệp trong đó.
- *Được nén*: Dạng JAR cho phép nén các tệp để cho việc lưu trữ hiệu quả và sau đó việc tải các tệp nén là khá tiện lợi.

- *Đóng dấu xi vào gói (Version 1.2)*: Tất cả các lớp được định nghĩa trong một gói được tìm thấy trong cùng một tệp JAR.
- *Tương thích*: Cơ chế xử lý các tệp JAR là một bộ phận được chuẩn hoá của API với Java platform.

Để nén và gói những tệp liên quan với nhau vào một gói tệp JAR, sử dụng chương trình *jar* của JDK và viết một dòng lệnh DOS command dạng:

```
jar cf  jarFileName.jar fileList
```

Trong đó, jarFileName là tên của tệp JAR, fileList là danh sách các tệp lớp hoặc những tệp khác có liên quan, ngăn cách với nhau bằng dấu cách.

Ví dụ, nếu muốn tạo ra một gói applet.jar để gói các lớp RMIApp.class, Send.class, index.html, java.policy, MessagesBundle_en_US.properties thì thực hiện như sau:

```
jar cf applet.jar
        RMIApp.class Send.class  index.html  java.policy
    MessagesBundle_en_US.properties
```

Khi muốn tời nén và mở gói applet.jar, ta sử dụng chương trình jar để tách ra với tuỳ chọn xf:

```
jar xf applet.jar
```

Tiếp theo, chúng ta đi tìm hiểu cách ký chứng nhận một applet để sử dụng trong Java Plug-in. Có hai kịch bản chính:

1. Các applet được ký xác nhận trên mạng nội hạt (Intranet)
2. Các applet được ký xác nhận trên mạng quốc tế (Internet).

Trong kịch bản thứ nhất, bộ phận quản trị hệ thống có thể cài đặt các giấy chứng nhận và các tệp chính sách trên các máy địa phương. Khi Java Plug-in nạp về một applet được ký xác nhận, nó sẽ thông qua kho chứa các khóa và tệp chính sách để kiểm duyệt các quyền được thực hiện của applet đó.

Ở kịch bản thứ hai, các hãng bán phần mềm nhận được giấy chứng nhận được ký bởi các tổ chức có thẩm quyền như Thrawte hay Verign. Khi người sử dụng đầu cuối truy cập vào một trang Web có chứa một applet đã được ký xác nhận, hệ thống xác định được danh tính của người cung cấp applet này và cho phép nó thực hiện theo một số giới hạn nào đó hoặc toàn quyền sử dụng.

Phần này chúng ta chủ yếu tìm hiểu cách xây dựng các tệp chính sách để cấp quyền cho các applet từ các nguồn đã biết. Việc xây dựng, triển khai các tệp chính sách không phải là những công việc dành cho những người sử dụng đầu cuối, mà chính là những người quản trị hệ thống phải đảm nhiệm trong mạng nội hạt.

Ví dụ 7.9. Chúng ta xét tiếp ví dụ về yêu cầu trao đổi thông tin của Engineering Department. Phòng này muốn nhân viên của mình chạy một số chương trình applet và chúng được phép truy cập được các tệp cục bộ tại các máy đó. Bởi vì những applet này không thể chạy được ở bên trong hộp cát Sandbox [2] nên họ phải cài đặt các tệp trên các máy đó. Điều này có nghĩa là họ sẽ phải cập nhật lại các tệp chính sách mỗi khi mã chương trình applet phải chuyển sang Web Server khác. Để khắc phục nhược điểm đó, họ quyết định ký xác nhận các tệp JAR chứa mã lệnh của các applet. Quá trình thực hiện như sau:

1. Tạo ra tệp MyApplet.jar để chứa các tệp mã chương trình applet

```
jar cvf  MyApplet.jar *.class
```

2. Chạy jarsigner để ký xác nhận tệp MyApplet.jar theo khóa riêng từ kho các khóa engsoft.store với biệt danh là engroot.

```
jarsigner -keystore engsoft.store MyApplet.jar engroot
```

Tất nhiên, kho chứa khóa gốc cần phải để ở vị trí an toàn. Muốn thế, ta hãy lập kho thứ hai là certs.store để cho các giấy chứng nhận.

```
keytool -genkey -keystore engsoft.store -alias engroot
keytool -export -keystore engsoft.store -alias engroot -file  engroot.cert
keytool -import -keystore certs.store -alias engroot -file  engroot.cert
```

Lệnh đầu tiên hệ thống sẽ hỏi chúng ta về mật khẩu và các thông tin cần thiết để cấp giấy chứng nhận và thiết lập kho chứa các cặp khóa cùng các giấy chứng nhận. Lưu ý: cần nhớ mật khẩu đó để báo cho hệ thống thẩm định ở các câu lệnh sau đó.

Tiếp theo ta phải tạo ra tệp chính sách để cấp phép cho tất cả các applet được ký xác nhận trong kho vừa được thiết lập.

Ta phải đưa thêm vị trí lưu kho và kiểu kho vào đầu tệp chính sách.

```
keystore "keystoreURL", "keystoreType";
```

Ví dụ, "JKS" là kiểu kho được sinh bởi keytool, dòng

```
keystore "file:certs.store", "JKS";
```

sẽ được đưa vào đầu tệp chính sách.

Ngoài ra cũng cần bổ sung signedBy "alias" vào sau một hoặc nhiều mệnh đề grant trong tệp chính sách.

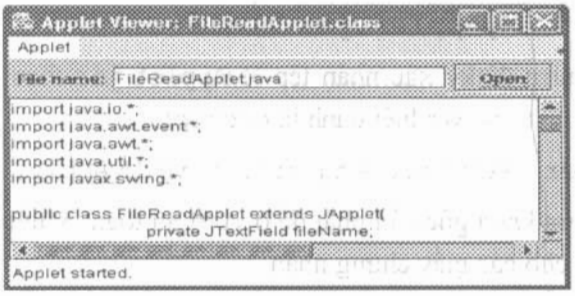
Như vậy, tệp chính sách trong ví dụ của chúng ta: applets.policy sẽ phải là

```
keystore "file:certs.store", "JKS";
grant signedBy "engroot"
{  permission java.io.FilePermission "<<ALL FILES>>", "read";
};
```

Hãy xây dựng một chương trình `FileReadApplet` để đọc các tệp ở thư mục cơ sở và tất cả các thư mục con của nó.

Sau khi đã hoàn tất các bước nêu trên, `engsoft.store`, `certs.store` với bí danh là `engroot`, `applets.policy` và `FileReadApplet.class` đã được tạo ra, ta có thể sử dụng `appletviewer` có sử dụng tệp chính sách như sau:

```
appletviewer -J-Djava.security.policy=applets.policy
        FileReadApplet.html
```



Hình 7.11. **Chương trình `FileReadApplet` được phép đọc các tệp cục bộ**

```
// FileReadApplet.java
import java.io.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;
import javax.swing.*;

public class FileReadApplet extends JApplet{
   private JTextField fileName;
   private JTextArea fileText;
   public FileReadApplet()
   {
      fileName = new JTextField(20);
      JPanel panel = new JPanel();
      panel.add(new JLabel("File name:"));
      panel.add(fileName);
      JButton openButton = new JButton("Open");
      panel.add(openButton);
      openButton.addActionListener(
```

```
                new ActionListener(){
                    public void actionPerformed(ActionEvent evt){
                        loadFile(fileName.getText());
                    }
            });
            Container contentPane = getContentPane();
            contentPane.add(panel, "North");

            fileText = new JTextArea();
            contentPane.add(new JScrollPane(fileText), "Center");
        }

        public void loadFile(String fileName){
            try{
                fileText.setText("");
                BufferedReader in = new BufferedReader
                    (new FileReader(fileName));
                String s;
                while((s = in.readLine()) != null)
                    fileText.append(s + "\n");
                in.close();
            }catch(IOException e){
                fileText.append(e + "\n");
            }catch(SecurityException e){
                fileText.append("I am sorry, but I cannot do that!\n");
            }
        }
    }
```

Để chạy được trong chế độ Java Plug-in, ta phải thay đổi tệp java.security ở thư mục {java.home}/jre/lib/security.

```
# The default is to have a single system wide policy file
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/java.policy
```

Và bổ sung thêm file URL cho tệp chính sách như sau:

```
policy.url.3=file:///home}/applets.policy
```

Nếu ta sử dụng Plug-in trong Netscape 4 hoặc Internet Explore thì cần phải thay đổi tệp HTML để nạp Plug-in cùng với các thẻ EMBED hoặc OBJECT. Netscape 5 sẽ tự động nạp Plug-in khi ta sử dụng thẻ OBJECT.

BÀI TẬP

7.1. Cho trước một danh sách các từ, các thư mục cấm truy cập. Xây dựng một applet để kiểm soát việc xem (đọc) nội dung của các tệp tin, ở những thư mục được phép theo tệp chính sách.

7.2. Một đơn vị muốn trao đổi thông tin và cung cấp các phầm mềm cho nhân viên của mình trên mạng nội hạt. Hãy xây dựng giấy chứng nhận cho người đứng đầu đơn vị. Mỗi thông điệp hay phần mềm cần đưa lên mạng sẽ được đưa vào tệp nén JAR cùng với giấy chứng nhận. Người nhận sẽ kiểm tra lại tính xác thực của thông tin, của phần mềm căn cứ vào tính xác thực của giấy chứng nhận.

7.3. Xây dựng hệ thống bán hàng qua mạng của Công ty Sao Mai.

+ Khách hàng có thể đặt mua hàng bằng cách gửi đơn đặt hàng, trong đó lựa chọn những mặt hàng, số lượng cần mua và thông báo số tài khoản. Đơn đặt mua hàng được ký xác nhận bởi khách hàng bằng khóa riêng. Khách mua hàng gửi đơn mua hàng và khóa công khai cho Công ty Sao Mai.
+ Công ty Sao Mai (người bán hàng) sẽ kiểm tra tính xác thực đơn đặt mua hàng của khách bằng khóa công khai. Nếu đúng thì lập hoá đơn bán hàng, ký xác nhận rồi gửi hàng cùng hoá đơn cho khách hàng.

Chương 8
LẬP TRÌNH THEO CHUẨN QUỐC TẾ VÀ BẢN ĐỊA HÓA PHẦN MỀM

Chương VIII giới thiệu những kỹ thuật xây dựng phần mềm đáp ứng các qui định chuẩn của mỗi quốc gia (bản địa hoá) gồm những nội dung chính:

- ✓ Vai trò của các lớp `Locale` và các dạng biểu diễn dữ liệu
- ✓ Lớp xử lý ngày tháng `Date`, thời gian `Time` và tiền tệ
- ✓ Những vấn đề quốc tế hoá phần mềm
- ✓ Bản địa hoá giao diện đồ hoạ.

8.1. VẤN ĐỀ BIỂU DIỄN DỮ LIỆU PHỤ THUỘC VÀO VĂN HÓA CỦA TỪNG DÂN TỘC

Trong thời đại hội nhập kinh tế thế giới hiện nay, sự trao đổi giữa các công ty dù lớn hay nhỏ với nhau phần lớn đều sử dụng nhiều ngôn ngữ khác nhau. Vấn đề trao đổi thông tin giữa các dân tộc, giữa nhiều cộng đồng trên thế giới với nhau luôn gặp phải những khó khăn, những trở ngại do việc qui định về cách viết và hiểu các thông điệp, số liệu là rất khác nhau. Ví dụ đơn giản như dấu '.' được sử dụng để phân biệt các đơn vị hàng ngàn, triệu, tỉ, v.v. của các số nguyên ở Việt Nam, trong khi người Mỹ lại sử dụng dấu ',' để phân biệt các đơn vị đó. Như vậy, *dữ liệu phụ thuộc vào văn hoá* là dữ liệu có thể thay đổi (cách biểu diễn) theo văn hoá, dân tộc, theo từng chuẩn hoá của từng quốc gia, từng khu vực [2].

Thực tế hiện nay có nhiều người trên thế giới có thể đọc và hiểu được tiếng Anh. Song, người sử dụng sẽ cảm thấy thoải mái, tự tin hơn khi sử dụng những `applet` hay chương trình ứng dụng hiển thị các thông tin được viết bằng tiếng dân tộc của họ và biểu diễn dữ liệu theo những khuôn dạng mà họ quen thuộc. Hãy tưởng tượng, một `AppletCalculator` (được giới thiệu ở cuối chương) sẽ được nhiều người sử dụng hiệu quả hơn nhiều nếu nó hiển thị được các kết quả tính toán theo khu vực của họ.

`Java 2 Platform` cung cấp các đặc trưng để thực hiện việc quốc tế hoá, cho phép tách các dữ liệu phụ thuộc vào các nền văn hoá từ những phần mềm và làm cho thích ứng với nhiều nền văn hoá theo nhu cầu (địa phương hoá hay còn gọi là bản địa hoá). Chúng ta hy

vọng phần lớn những vấn đề về thông tin, dữ liệu được cung cấp cho dân cư trú ở những vùng khác nhau được biểu diễn phù hợp với nền văn hoá của họ trên các phần mềm ứng dụng, hay trên các applet.

Nhiều người lập trình tin rằng, họ cần quốc tế hoá những phần mềm của mình bằng cách sử dụng Unicode và dịch các thông báo sang giao diện của người sử dụng. Tuy nhiên, như ở chương này chúng ta sẽ thấy, còn nhiều vấn đề liên quan đến việc quốc tế hoá trong lập trình hơn là sự hỗ trợ của Unicode. Nhiều hệ điều hành, thậm chí nhiều trình duyệt có thể không cần thiết phải hỗ trợ Unicode. Vấn đề quan trọng là phải dịch, chuyển đổi giữa tập các ký tự và font chữ của máy người sử dụng với JVM có sự hỗ trợ Unicode.

Dễ dàng nhận thấy, *ngày tháng, thời gian, tiền tệ lưu hành và các số liệu được biểu diễn khác nhau theo những vùng khác nhau trên thế giới*. Do vậy, chúng ta cần sử dụng một cách nào đó để định dạng lại các thực đơn, tên của các nút nhấn, các xâu thông điệp, các phím nóng, v.v. trong các phần mềm ứng dụng theo những ngôn ngữ khác nhau để đáp ứng những yêu cầu nêu trên.

8.2. LỚP LOCALE

Khi cần tìm một phần mềm phù hợp cho thị trường quốc tế thì vấn đề khác biệt hiển nhiên cần chú ý đầu tiên là ngôn ngữ. Vấn đề ngôn ngữ luôn là trở ngại lớn nhất trong giao lưu quốc tế, trong đó có cả vấn đề quốc tế hoá phần mềm. Ngay cả khi một số nước sử dụng chung một ngôn ngữ, nhưng từng quốc gia, dân tộc có thể có những qui định khác nhau, do vậy vẫn phải tạo ra những phần mềm sao cho phù hợp với họ nhất có thể.

Trong tất cả các trường hợp, các thực đơn, tên các nút nhấn, các thông báo của chương trình sẽ phải được dịch sang ngôn ngữ bản địa. Ở đây, các thuật ngữ, các qui định biểu diễn dữ liệu luôn có khá nhiều sự khác biệt rất tinh vi. Ví dụ, các số được biểu diễn theo những khuôn dạng khác nhau giữa Anh (Mỹ) và Đức. Người Đức sử dụng dấu chấm '.' để phân biệt các đơn vị hàng ngàn, triệu và dấu phẩy ',' thập phân để phân biệt phần số nguyên với phần thập phân, như số 124.456,45, nhưng người Anh thì sử dụng ngược lại. Vấn đề tương tự đối với việc hiển thị ngày tháng, thời gian, tiền tệ, v.v. Ngày tháng ở Mỹ được hiển thị theo mẫu tháng/ngày/năm, người Đức sử dụng theo thứ tự ngày/tháng/năm, trong khi người Trung Quốc lại sử dụng năm/tháng/ngày.

Để giải quyết những vấn đề nêu trên, Java cung cấp lớp Locate. Lớp này tập trung mô tả:

- language: xác định ngôn ngữ, ví dụ English, German, Chinese, v.v.
- location: tên quốc gia, bản địa sử dụng ngôn ngữ nêu trên, ví dụ United State (Hoa Kỳ), Germany (Đức), China (Trung Quốc), v.v.
- variant: (Tuỳ chọn) ít sử dụng, thường sử dụng trong những trường hợp ngoại lệ hoặc những tình huống phụ thuộc vào hệ thống qui định. Ví dụ, người Na-Uy sử

dụng hai tập qui tắc viết vần: qui tắc truyền thống là `Bokmål` và qui tắc mới được gọi là `Nynorsk`. Khi muốn chọn `Bokmål` thì khai báo

```
language = Norwegian, location = Norway, variant = Bokmål
```

Các ngôn ngữ bản địa được mã hoá trong `ISO-639` (Bảng 8.1) bằng hai chữ cái thường cho ngôn ngữ và hai chữ cái hoa trong `ISO-3166` (Bảng 8.2) cho tên của các quốc gia. Ta có thể xem danh sách đầy đủ các ngôn ngữ trong bảng mã ISO-639 ở http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt và danh sách đầy đủ các quốc gia trong bảng mã ISO-3166 ở http://www.niso.org/3166.html.

Bảng 8.1. **Bảng ISO-639 cho tên các ngôn ngữ**

Ngôn ngữ	Mã
Chinese (Trung Quốc)	zh
Danish (Đan Mạch)	da
Dutch (Hà Lan)	nl
English (Anh)	en
French (Pháp)	fr
Finnish (Phần Lan)	fi
German (Đức)	de
Greek (Hy Lạp)	el
Italian (Ý)	it
Japanese (Nhật bản)	ja
Korean (Triều Tiên)	ko
Norwegian (Na Uy)	no
Portuguese (Bồ Đào Nha)	pt
Spanish (Tây Ba Nha)	sp
Swedish (Thuỵ Điển)	sv
Turkish (Thổ Nhĩ Kỳ)	tr

Để xác định một vùng bản địa, ta phải khai báo mã ngôn ngữ, mã vùng (quốc gia) và biến thể nếu cần. Ví dụ:

```
Locale germanGermany = new Locate("de", "DE");
Locale uSA = new Locate("en", "US");
Locale norwegianNorwayBokmål = new Locate("no", "NO", "B");
```

Để tiện lợi, JDK định nghĩa trong `Locale` một số ngôn ngữ phổ biến như:

```
Locale.CHINESE                  Locale.ENGLISH
Locale.FRENCH                   Locale.GERMAN
Locale.ITALIAN                  Locale.JAPANESE, v.v.
```

Tương tự, một số tên nước được định nghĩa trong `Locale`.

`Locale.CANADA`	`Locale.CHINA`
`Locale.FRANCE`	`Locale.GERMANY`
`Locale.ITALY`	`Locale.JAPAN`
`Locale.UK`	`Locale.US`, v.v.

Bảng 8.2. **Bảng ISO-31666 cho tên các quốc gia**

Quốc gia	Mã
Austria (Áo)	AT
Belgium (Bỉ)	BE
Canada (Ca – na – đa)	CA
China (Trung Quốc)	CN
Denmark (Đan Mạch)	DK
Finland (Phần Lan)	FI
Germany (Đức)	DE
Great Britain (Anh)	GB
Greece (Hy Lạp)	GR
Ireland (Ai – Len)	IE
Italy (Ý)	IT
France (Pháp)	FR
Japan (Nhật bản)	JP
Korea (Triều Tiên)	KR
The Netherlands (Hà Lan)	NL
Norway (Na Uy)	NO
Portugal (Bồ Đào Nha)	PT
Spain (Tây Ba Nha)	ES
Sweden (Thuỵ Điển)	SE
Switzerland (Thuỵ Sĩ)	CH
Taiwan (Đài Loan)	TW
Turkey (Thổ Nhĩ Kỳ)	TR
United States (Hoa Kỳ)	US

java.util.Locale API

- `static Locale getDefault()`

  Nhận lại một đối tượng bản địa mặc định.

- `static void setDefault(Locale l)`

  Đặt lại đối tượng bản địa mặc định là `l`.

- `String getDisplayName()`

  Nhận lại tên của đối tượng hiển thị hiện thời.

- `String getDisplayName(Locale l)`

  Nhận lai tên của đối tượng hiển thị được xác định bởi `l` cho trước.

- `String getLanguage()`

  Nhận lại mã của ngôn ngữ theo bảng mã ISO-639.

- `String getDisplayLanguage()`

  Nhận lại tên của ngôn ngữ được xác định bởi đối tượng hiện thời.

- `String getDisplayLanguage(Locale l)`

  Nhận lại tên của ngôn ngữ được xác định bởi `l` cho trước.

- `String getCountry()`

  Nhận lại mã của nước được xác định trong bảng mã ISO-3166.

- `String getDisplayCountry()`

  Nhận lại tên của nước được xác định bởi đối tượng hiện thời.

- `String getDisplayCountry(Locale l)`

  Nhận lại tên của nước được xác định bởi `l` cho trước.

- `String getVariant()`

  Nhận lại tên của biến thể.

- `String getDisplayVariant()`

  Nhận lại tên của biến thể được xác định bởi đối tượng hiện thời.

- `String getDisplayVariant(Locate l)`

  Nhận lại tên của biến thể được xác định bởi `l` cho trước.

- `String toString()`

  Nhận lại mô tả về đối tượng hiện thời bao gồm mã tiếng, nước, biến thể được nối với nhau bằng dấu '_', ví dụ "en_US".

8.3. CÁC DỮ LIỆU SỐ VÀ ĐƠN VỊ TIỀN TỆ

Như trên đã lưu ý, các đại lượng số và giá trị tiền tệ được biểu diễn theo những khuôn dạng rất khác nhau, tuỳ thuộc vào từng vùng trên thế giới. Java cung cấp một tuyển tập các

lớp giúp cho việc định dạng và phân loại các giá trị số sang lớp tương ứng trong `java.text`. Để định dạng khuôn mẫu các con số ta có thể thực hiện các bước như sau:

1. Xác định đối tượng bản địa hiện thời
2. Sử dụng phương thức như `getNumberInstance()`, `getCurrencyInstance()`, v.v. để xác định đối tượng định dạng hiện thời.
3. Sử dụng đối tượng định dạng hiện thời để định dạng và phân loại theo yêu cầu.

Ví dụ, ta muốn hiển thị số tiền 123456.78 theo đơn vị tiền và qui định của người Đức.

```
Locale loc = new Locale("de", "DE");

NumberFormat currFmt = NumberFormat.getCurrencyInstance(loc)
double val = 123456.78;
System.out.println(currFmt.format(val));
```

Kết quả in ra màn hình sẽ là

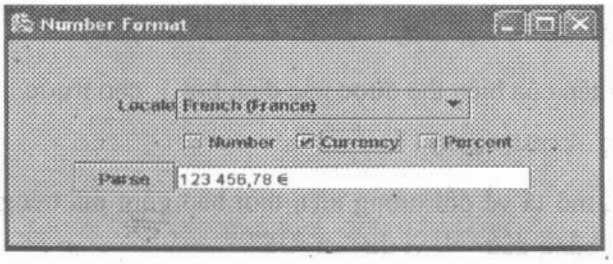
```
123.456,78 DM
```

Lưu ý, `DM` là đơn vị tiền tệ của Đức, được viết ở sau số tiền.

Ngược lại, nếu ta muốn đọc các đại lượng số đã được nhập vào, lưu trữ theo qui ước của một vùng nào đó thì phải sử dụng phương thức `parse()` để phân loại các đại lượng đó. Ví dụ, đọc một giá trị số nhập vào từ `inputField`, chuyển nó về khuôn dạng của vùng bản địa hiện thời trên máy tính.

```
TextField inputField;

NumberFormat fmt = NumberFormat.getNumberInstance();
Number input = fmt.parse(inputFiled.getText().trim());
double val = input.doubleValue();
```

Ví dụ 8.1. Chương trình hiển giá trị 123456.78 như là đại lượng số, tiền tệ hay tỷ lệ % theo các nước được lựa chọn. Ví dụ, chọn `French(France)` và chọn đơn vị tiền tệ, chương trình sẽ hiển thị như sau:



Hình 8.1. **Chương trình hiển thị số, tiền tệ và tỷ lệ % theo từng vùng**

```
// NumberFormatTest.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

```
import java.text.*;
import java.util.*;
public class NumberFormatTest{
   public static void main(String args[]){
      JFrame fr = new NumberFormatFrame();
      fr.show();
   }
}
class NumberFormatFrame extends JFrame{

   private Locale[] locales;
   private double currentNumber;

   private JComboBox combo = new JComboBox();
   private JButton parseButton = new JButton("Parse");
   private JTextField numText = new JTextField(30);
   private JCheckBox numberCheckBox = new JCheckBox("Number");
   private JCheckBox currencyCheckBox = new JCheckBox("Currency");
   private JCheckBox percentCheckBox = new JCheckBox("Percent");
   private ButtonGroup group = new ButtonGroup();
   private NumberFormat currentNumberFormat;

   public NumberFormatFrame(){
      setSize(400, 200);
      setTitle("Number Format");
      addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                  System.exit(0);
            }
       });
      getContentPane().setLayout(new GridBagLayout());
      ActionListener listener = new ActionListener(){
            public void actionPerformed(ActionEvent evt){
                  updateDisplay();
```

```
JPanel p = new JPanel();
addCheckBox(p, numberCheckBox, group, listener, false);
addCheckBox(p, currencyCheckBox, group, listener, true);
addCheckBox(p, percentCheckBox, group, listener, false);

GridBagConstraints gbc = new GridBagConstraints();
gbc.fill = GridBagConstraints.NONE;
gbc.anchor = GridBagConstraints.EAST;
add(new JLabel("Locale"), gbc, 0, 0, 1, 1);
add(p, gbc, 1, 1, 1, 1);
add(parseButton, gbc, 0, 2, 1, 1);
gbc.anchor = GridBagConstraints.WEST;
add(combo, gbc, 1, 0, 1, 1);
gbc.fill = GridBagConstraints.HORIZONTAL;
add(numText, gbc, 1, 2, 1, 1);

locales = NumberFormat.getAvailableLocales();
for(int i = 0; i < locales.length; i++)
     combo.addItem(locales[i].getDisplayName());
combo.setSelectedItem(Locale.getDefault().getDisplayName());
currentNumber = 123456.78;
updateDisplay();

combo.addActionListener(listener);

parseButton.addActionListener(
     new ActionListener(){
          public void actionPerformed(ActionEvent evt){
              String s = numText.getText();
              try{
                  Number n = currentNumberFormat.parse(s);
                  if(n != null){
                      currentNumber = n.doubleValue();
                      updateDisplay();
                  }else{
```

```
                        numText.setText("Parse error: " + s);
                        }
                    }catch(ParseException e){
                        numText.setText("Parse error: " + s);
                        }
                }
            });
    }

    public void add(Component c, GridBagConstraints gbc,
        int x, int y, int w, int h){
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridheight = h;
        gbc.gridwidth = w;
        getContentPane().add(c, gbc);
    }

    public void addCheckBox(Container p, JCheckBox checkBox,
        ButtonGroup g, ActionListener listener, boolean v){
        checkBox.setSelected(v);
        checkBox.addActionListener(listener);
        g.add(checkBox);
        p.add(checkBox);
    }

    public void updateDisplay(){
        Locale currentLocale = locales[combo.getSelectedIndex()];
        currentNumberFormat = null;
        if(numberCheckBox.isSelected())
            currentNumberFormat =
                NumberFormat.getNumberInstance(currentLocale);
        else if(currencyCheckBox.isSelected())
            currentNumberFormat =
                NumberFormat.getCurrencyInstance(currentLocale);
```

```
        else if(percentCheckBox.isSelected())
            currentNumberFormat =
                NumberFormat.getPercentInstance(currentLocale);
        String n = currentNumberFormat.format(currentNumber);
        numText.setText(n);
    }
}
```

java.text.NumberFormat API

- `static Locale[] getAvailableLocales()`

  Nhận lại danh sách các đối tượng khu vực mà `NumberFormat` hỗ trợ.
- `static NumberFormat getNumberInstance()`
- `static NumberFormat getNumberInstance(Locale l)`
- `static NumberFormat getCurencyInstance()`
- `static NumberFormat getCurencyInstance(Locale l)`
- `static NumberFormat getPercentInstance()`
- `static NumberFormat getPercentInstance(Locale l)`

  Nhận lại định dạng `formatter` của đại lượng số, tiền tệ và tỷ lệ % của đối tượng bản địa hiện thời, hoặc đối tượng `l` cho trước.
- `String format (double x)`
- `String format (long x)`

  Nhận lại các số `double`, `long` theo định dạng hiện thời dưới dạng xâu.
- `Number parse (String s)`

  Chuyển đổi xâu `s` sang giá trị số tương ứng, là `Double` nếu đầu vào là số thực, ngược là `Long`.
- `void setMinimumIntegerDigits(int n)`
- `void getMinimumIntegerDigits()`
- `void setMaximumIntegerDigits(int n)`
- `void getMaxnimumIntegerDigits()`
- `void setMinimumFractionDigits(int n)`
- `void getMinimumFractionDigits()`
- `void setMaximumFractionDigits(int n)`
- `void getMaximumFractionDigits()`

  Các phương thức trên đặt lại, đọc ra số các chữ số phần nguyên và phần thập phân của đại lượng số.

8.4. NGÀY THÁNG VÀ THỜI GIAN

Khi cần định dạng ngày tháng và thời gian thì chúng ta phải quan tâm đến bốn vấn đề chính phụ thuộc vào từng khu vực trên thế giới.

- Tên của các tháng trong năm, tên của ngày trong tuần theo từng ngôn ngữ.
- Thứ tự viết ngày, tháng, năm.
- Lịch `Gregorian` có phù hợp với khu vực của người sử dụng hay không.
- Múi giờ của mỗi vùng.

Rất may là lớp `DateFormat` nhận xử lý tất cả những vấn đề nêu trên. Việc sử dụng nó cũng rất giống như `NumberFormat`. Trước tiên, ta cần xác định đối tượng khu vực của máy tính. Ta có thể sử dụng đối tượng mặc định hoặc thông qua `getAvailableLocales()` để nhận được danh sách các đối tượng khu vực mà `NumberFormat` hỗ trợ. Sau đó gọi một trong ba phương thức sau:

```
fmt = DateFormat.getDateInstance(dateStyle, loc);
fmt = DateFormat.getTimeInstance(timeStyle, loc);
fmt = DateFormat.getDateTimeInstance(dateStyle, timeStyle, loc);
```

Trong đó `dateStyle`, `timeStyle` có thể là một trong các hằng sau `DateFormat.DEFAULT`, `DateFormat.FULL`, `DateFormat.LONG`, `DateFormat.MEDIUM`, `DateFormat.SHORT`, còn `loc` là đối tượng khu vực.

Ta có thể sử dụng phương thức `parse()` của `NumberFormat` để phân tích `Date` và `Time`. Ví dụ, việc nhập vào ngày tháng và chuyển đổi sang dạng qui định của máy mặc định được thực hiện như sau.

```
TextField inputField;
DateFormat fmt =
        NumberFormat.getDateInstance(DateFormat.MEDIUM);
Date input = fmt.parse(inputFiled.getText().trim());
```

Nếu dữ liệu nhập vào không tương thích thì sẽ phát sinh ngoại lệ thuộc loại `ParseException`. Ví dụ, nếu `format` nhập là `MEDIUM` theo qui ước của Mỹ thì dữ liệu ngày tháng nhập vào phải có dạng

Sep 18, 2005

Nếu người sử dụng nhập vào

Sep 18 2005

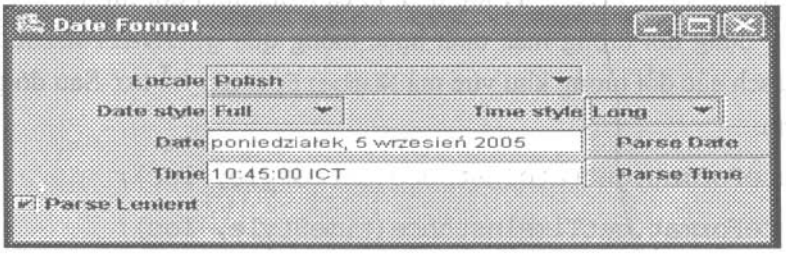
không có dấu ',' giữa ngày và năm, hoặc dạng

18/9/2005

thì chương trình sẽ phát sinh lỗi.

Cờ lenient báo cho chương trình thông dịch chỉnh lại ngày tháng cho phù hợp với lịch Dương, nghĩa là tha thứ một số lỗi trong nhập liệu. Ví dụ, nếu cờ này được bật (nhận giá trị true) thì hệ thống sẽ tự động chuyển ngày tháng nhập vào không thật chính xác theo lịch, ví dụ February 30, 1999 thành March 2, 1999. Điều này có vẻ nguy hiểm, song hệ thống Java vẫn chuyển theo mặc định, nếu không đặt lại cờ lenient.

Ví dụ 8.2. Viết chương trình hiển thị ngày giờ của máy tính hiện thời theo các khu vực và theo dạng: DEFAULT, LONG, MEDIUM, SHORT được người sử dụng lựa chọn ở các ComboBox: Locale, Date style, Time style. Ví dụ, khi người sử dụng chọn vùng Ba Lan (Polish) và dạng hiển thị ngày là FULL, dạng thời gian là LONG thì hệ thống sẽ hiển thị



Hình 8.2. **Chương trình hiển thị ngày giờ theo qui ước của Ba Lan**

trong đó Poniedzałek là thứ Hai, Wrzesień là tháng Chín.

```
// DateFormatTest.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.util.*;

public class DateFormatTest{
   public static void main(String args[]){
      JFrame fr = new DateFormatFrame();
      fr.show();
   }
}

class DateFormatFrame extends JFrame{

   private Locale[] locales;
   private Date currentDate;
   private Date currentTime;
```

```
private DateFormat currentDateFormat;
private DateFormat currentTimeFormat;

private JComboBox combo = new JComboBox();
private EnumCombo dateStyleCombo
    = new EnumCombo(DateFormat.class,
        new String[]{"Default", "Full", "Long", "Medium", "Short"});
private EnumCombo timeStyleCombo
    = new EnumCombo(DateFormat.class,
        new String[]{"Default", "Full", "Long", "Medium", "Short"});

private JButton dateButton = new JButton("Parse Date");
private JButton timeButton = new JButton("Parse Time");
private JTextField dateText = new JTextField(30);
private JTextField timeText = new JTextField(30);
private JTextField parseText = new JTextField(30);
private JCheckBox lenientCheckBox = new JCheckBox("Parse Lenient", true);

public DateFormatFrame(){
    setSize(400, 200);
    setTitle("Date Format");
    addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e){
              System.exit(0);
         }
     });

    getContentPane().setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.fill = GridBagConstraints.NONE;
    gbc.anchor = GridBagConstraints.EAST;
    add(new JLabel("Locale"), gbc, 0, 0, 1, 1);
    add(new JLabel("Date style"), gbc, 0, 1, 1, 1);
    add(new JLabel("Time style"), gbc, 2, 1, 1, 1);
```

```
add(new JLabel("Date"), gbc, 0, 2, 1, 1);
add(new JLabel("Time"), gbc, 0, 3, 1, 1);

gbc.anchor = GridBagConstraints.WEST;
add(combo, gbc, 1, 0, 2, 1);
add(dateStyleCombo, gbc, 1, 1, 1, 1);
add(timeStyleCombo, gbc, 3, 1, 1, 1);
add(dateButton, gbc, 3, 2, 1, 1);
add(timeButton, gbc, 3, 3, 1, 1);
add(lenientCheckBox, gbc, 0, 4, 1, 1);
gbc.fill =  GridBagConstraints.HORIZONTAL;
add(dateText, gbc, 1, 2, 2, 1);
add(timeText, gbc, 1, 3, 2, 1);

locales = DateFormat.getAvailableLocales();
for(int i = 0; i < locales.length; i++)
     combo.addItem(locales[i].getDisplayName());
combo.setSelectedItem(Locale.getDefault().getDisplayName());
currentDate = new Date();
currentTime = new Date();
updateDisplay();

ActionListener listener =
     new ActionListener(){
          public void actionPerformed(ActionEvent evt){
               updateDisplay();
          }
};
combo.addActionListener(listener);
dateStyleCombo.addActionListener(listener);
timeStyleCombo.addActionListener(listener);

dateButton.addActionListener
     new ActionListener(){
          public void actionPerformed(ActionEvent evt){
```

```
                String d = dateText.getText();
                try{
                  currentDateFormat.setLenient(
                           lenientCheckBox.isSelected()
                      );
                  Date date = currentDateFormat.parse(d);
                  currentDate = date;
                  updateDisplay();
                }catch(ParseException e){
                  dateText.setText("Parse error: " + d);
                }
                catch(IllegalArgumentException e){
                  dateText.setText("Argument error: " + d);
              }
            }
        });
        timeButton.addActionListener(
             new ActionListener(){
                  public void actionPerformed(ActionEvent evt){
                      String t = timeText.getText();
                      try{
                        currentDateFormat.setLenient(
                           lenientCheckBox.isSelected());
                        Date date = currentTimeFormat.parse(t);
                        currentTime = date;
                        updateDisplay();
                      }catch(ParseException e){
                        timeText.setText("Parse error: " + t);
                      }
                      catch(IllegalArgumentException e){
                        timeText.setText("Argument error: " + t);
                    }
                  }
        });
    }
```

```
    public void add(Component c, GridBagConstraints gbc,
        int x, int y, int w, int h){
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridheight = h;
        gbc.gridwidth = w;
        getContentPane().add(c, gbc);
    }

    public void updateDisplay(){
        Locale currentLocale =
            locales[combo.getSelectedIndex()];
        int dateStyle = dateStyleCombo.getValue();
        currentDateFormat =
            DateFormat.getDateInstance(dateStyle, currentLocale);
        String s = currentDateFormat.format(currentDate);
        dateText.setText(s);

        int timeStyle = timeStyleCombo.getValue();
        currentTimeFormat =
            DateFormat.getTimeInstance(timeStyle, currentLocale);
        String t = currentTimeFormat.format(currentTime);
        timeText.setText(t);
    }
}
class EnumCombo extends JComboBox {
    public EnumCombo(Class cl, String[] labels){
        for(int i = 0; i < labels.length; i++){
            String label = labels[i];
            String name = label.toUpperCase().replace(' ', '_');
            int value = 0;
            try{
                java.lang.reflect.Field f = cl.getField(name);
                value = f.getInt(cl);
```

```
            }catch(Exception e){
                label = "(" + label + ")";
            }
            table.put(label, new Integer(value));
            addItem(label);
        }
        setSelectedItem(labels[0]);
    }

    public int getValue(){
        return ((Integer)table.get(getSelectedItem())).intValue();
    }
    private Map table = new HashMap();
}
```

java.text.DateFormat **API**

- `static Locale[] getAvailableLocales()`

  Nhận lại một danh sách các đối tượng khu vực mà DateFormat hỗ trợ.
- `static DateFormat getDateInstance(int dateStyle)`
- `static DateFormat getDateInstance(int dateStyle, Locale l)`
- `static DateFormat getTimeInstance(int timeStyle)`
- `static DateFormat getDateTimeInstance(int timeStyle, Locale l)`
- `static DateFormat getDateTimeInstance(int dateStyle, int timeStyle)`
- `static DateFormat getDateTimeInstance(int dateStyle, int timeStyle, Locale l)`

  Nhận lại khuôn dạng cho ngày, giờ, ngày và giờ theo khu vực của máy mặc định hoặc khu vực cho trước bởi l.
- `String format(String s)`

  Nhận lại xâu kết quả của ngày / giờ theo khuôn dạng qui định.
- `Date parse(String s)`

  Chuyển xâu s sang ngày / giờ theo khuôn dạng qui định.
- `void setLenient(boolean b)`
- `boolean isLenient()`

  Đặt cờ và kiểm tra trạng thái của cờ lenient phục vụ cho việc chỉnh sửa ngày theo lịch khi dữ liệu đầu vào không tương thích.

- `void setCalendar(Calendar cal)`
- `Calendar getCalendar()`

  Đặt và đọc đối tượng lịch biểu để tách ra các thông tin về ngày, tháng, năm, giờ, phút, giây. Lịch biểu mặc định là `Gregorian` còn được gọi là Lịch Dương.
- `void setTimeZone(TimeZone f)`
- `TimeZone getTimeZone()`

  Đặt và đọc đối tượng múi giờ phục vụ cho việc định dạng cho đúng với khu vực hiện thời và đúng lịch biểu.
- `void setNumberFormat(NumberFormat f)`
- `NumberFormat getNumberFormat()`

  Đặt và đọc đối tượng khuôn số liệu sử dụng để định dạng ngày, tháng, năm, giờ, phút, giây.

8.5. VĂN BẢN TEXT

Việc hiển thị các văn bản cũng rất nhiều vấn đề cần phải chú ý. Trong phần này chúng ta tìm hiểu cách biểu diễn và thao tác trên các xâu văn bản.

8.5.1. Sắp xếp văn bản

Sắp xếp các xâu văn bản theo một thứ tự nào đó là vấn đề tương đối đơn giản, nếu chúng được xây dựng từ các ký tự trong bản mã ASCII (tiếng Anh). Thường ta có thể sử dụng

```
s1.compareTo(s2)
```

để so sánh xâu `s1` với `s1` để sắp xếp chúng theo thứ tự từ điển.

Nhưng nếu giữa các xâu có một số chữ thường, chữ in hoa lẫn lộn thì dãy các xâu có thể không được sắp theo thứ tự mà ta mong muốn. Biết rằng, phương thức `compareTo()` trong Java sử dụng bộ mã `Unicode` để sắp xếp. Trong bảng mã `Unicode`, mã của chữ cái thường lớn hơn mã chữ cái in hoa, các chữ có dấu thậm chí còn có mã cao hơn. Ví dụ, dãy gồm năm xâu: "An", "Áng", "anna", "Phe", "phan" nếu sắp theo thứ tự với phương thức `compareTo()` sẽ được dãy:

"An", "Phe", "anna", "phan", "Áng"

Nhưng theo cách sắp xếp của nhiều nước, trong đó có Việt Nam thì chữ Á phải đứng sau A, nghĩa là dãy trên cần phải xếp theo thứ tự như sau:

"An", "Áng", "anna", "phan", "Phe"

Ngược lại, người sử dụng một số nước khác lại không chấp nhận thứ tự nêu trên, ví dụ Đan Mạch. Theo qui định của họ, chữ cái có dấu (có trọng âm) khác với chữ không có dấu,

nghĩa là chữ 'Á' phải xếp sau 'Z'. Do vậy, theo yêu cầu của người Đan Mạch, dãy trên phải được xếp theo thứ tự như sau:

"An", "anna", "phan", "Phe", "Áng"

Những vấn đề nêu trên có thể giải quyết được bằng cách sử dụng lớp `Collator` để đối chiếu. Như thường lệ, ta phải xác định đối tượng khu vực của `Locale`. Sau đó gọi `getInstance()` để nhận được đối tượng của `Collator`. Cuối cùng để so sánh ta sử dụng `compare()` của `Collator`.

```
Locale loc = ...
Collator coll = Collator.getInstance(loc);
if(coll.compare(s1, s2) < 0)   // s1 đứng trước s2
```

Điều quan trọng hơn, lớp `Collator` cài đặt giao diện `Comparator`. Vì thế ta có thể sử dụng `Collections.sort(strings, coll)` để sắp xếp `strings` theo thứ tự của đối tượng `coll`.

Hơn nữa, trong lý thuyết sắp xếp từ vựng, người ta còn phân chia sự khác nhau giữa các ký tự thành ba mức: *cơ sở, thứ cấp và tam cấp*. Ví dụ, trong tiếng Anh, sự khác nhau giữa 'A' và 'Z' là cơ sở (`PRIMARY`), sự khác nhau giữa 'A' và 'Á' là thứ cấp (`SECONDARY`) và giữa 'A' và 'a' là tam cấp (`TERTIARY`). Hiển nhiên ta có thể đặt lại mức độ cho công việc sắp xếp.

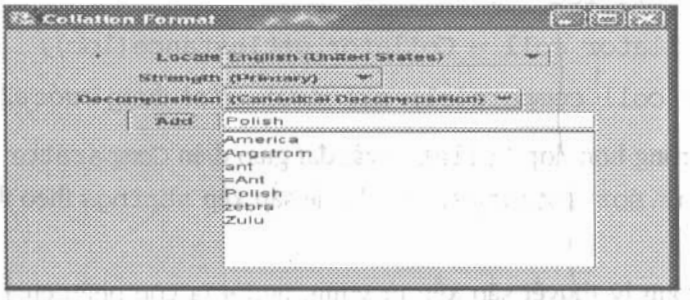
Việc chọn mức độ `Collator.PRIMARY` là muốn báo cho hệ thống biết rằng chỉ cần chú ý vào sự khác nhau cơ sở. Chúng ta hãy xét ví dụ sau.

```
// Giả sử dụng khu vực của những người nói tiếng Anh
String s1 = "Angstrom";
String s1 = "Ángstrom";
coll.setStrength(Collator.PRIMARY)
if(coll.compare(s1, s2) == 0)
      System.out.println("Giống nhau");     // in ra "Giống nhau"
else System.out.println("Khác nhau");
coll.setStrength(Collator.SECONDARY)
if(coll.compare(s1, s2) == 0)
      System.out.println("Giống nhau");
else System.out.println("Khác nhau");              // in ra "Khác nhau"
```

Vấn đề cuối cùng liên quan đến việc sắp xếp là các cách thức "phân rã" (`decomposition mode`). Cách thức hay sử dụng nhất là phân rã chuẩn tắc (`canonical decomposition`), nó thường được đặt mặc định. Nếu ta chọn cách không phân rã (`no decomposition`) thì các chữ có dấu sẽ không được tách ra khỏi dạng cơ sở và dấu của

chúng. Sự lựa chọn này sẽ giúp cho việc sắp xếp nhanh hơn, song chỉ cho kết quả đúng nếu dãy cần sắp xếp không có các chữ có dấu. Khả năng lựa chọn thứ ba là phân rã hoàn toàn (`full decomposition`), được sử dụng để phân rã các biến thể của `Unicode`.

Ví dụ 8.3. Chương trình thực hiện sắp xếp một dãy các xâu được nhập vào theo từng khu vực (`Locate`), mức độ (`Strength`) và cách thức (`Decomposition`) được lựa chọn trong các hộp `ComboBox`. Mỗi lần nhập một xâu, hay thay đổi một trong các lựa chọn trên thì dãy các xâu đó sẽ được sắp xếp lại. Dấu '=' chỉ ra rằng những mục đó được xem là đồng nhất với mục trước đó trong thứ tự sắp xếp (xem hình 8.3).



Hình 8.3. **Chương trình sắp xếp dãy các xâu theo các tuỳ chọn**

```
// CollationTest.java
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.util.*;
import java.util.List;-++++

public class CollationTest{
   public static void main(String args[]){
      JFrame fr = new CollationFrame();
      fr.show();
   }
}

class CollationFrame extends JFrame{

   String[] d = {"America", "ant", "Zulu", "zebra", "Ant", "Angstrom"};

   private Locale[] locales;
```

```
private List strings = new ArrayList();
private Collator currentCollator;

private JComboBox combo = new JComboBox();
private EnumCombo strengthCombo
   = new EnumCombo(DateFormat.class,
        new String[]{"Primary", "Secondary", "Tertiary"});
private EnumCombo decompositionCombo
   = new EnumCombo(DateFormat.class,
        new String[]{"Canonical Decomposition",
          "No Decomposition", "Full Decomposition"});

private JButton addButton = new JButton("Add");
private JTextField newWord = new JTextField(20);
private JTextArea sortedWords = new JTextArea(10, 20);

public CollationFrame(){
   setSize(400, 400);
   setTitle("Collation Format");
   addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
             System.exit(0);
        }
    });

   getContentPane().setLayout(new GridBagLayout());
   GridBagConstraints gbc = new GridBagConstraints();
   gbc.fill = GridBagConstraints.NONE;
   gbc.anchor = GridBagConstraints.EAST;
   add(new JLabel("Locale"), gbc, 0, 0, 1, 1);
   add(new JLabel("Strength"), gbc, 0, 1, 1, 1);
   add(new JLabel("Decomposition"), gbc, 0, 2, 1, 1);
   add(addButton, gbc, 0, 3, 1, 1);

   gbc.anchor = GridBagConstraints.WEST;
   add(combo, gbc, 1, 0, 1, 1);
```

```
        add(strengthCombo, gbc, 1, 1, 1, 1);
        add(decompositionCombo, gbc, 1, 2, 1, 1);

        gbc.fill =  GridBagConstraints.HORIZONTAL;
        add(newWord, gbc, 1, 3, 1, 1);
        gbc.fill =  GridBagConstraints.BOTH;
        add(new JScrollPane(sortedWords), gbc, 1, 4, 1, 1);

        locales = Collator.getAvailableLocales();
        for(int i = 0; i < locales.length; i++)
             combo.addItem(locales[i].getDisplayName());
        combo.setSelectedItem(Locale.getDefault().getDisplayName());

        for(int i = 0; i < d.length; i++)
             strings.add(d[i]);

        updateDisplay();

        addButton.addActionListener(
             new ActionListener(){
                  public void actionPerformed(ActionEvent evt){
                       strings.add(newWord.getText());
                       updateDisplay();
                  }
             });

        ActionListener listener =
             new ActionListener(){
                  public void actionPerformed(ActionEvent evt){
                       updateDisplay();
                  }
        };
        combo.addActionListener(listener);
        strengthCombo.addActionListener(listener);
        decompositionCombo.addActionListener(listener);
   }
```

```
    public void add(Component c, GridBagConstraints gbc,
        int x, int y, int w, int h){
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridheight = h;
        gbc.gridwidth = w;
        getContentPane().add(c, gbc);
    }
    public void updateDisplay(){
        Locale currentLocale = locales[combo.getSelectedIndex()];
        currentCollator =
              Collator.getInstance(currentLocale);
    currentCollator.setStrength(strengthCombo.getValue());
    currentCollator.setDecomposition(decompositionCombo.getValue());
    Collections.sort(strings, currentCollator);

    sortedWords.setText("");
    for(int i = 0; i < strings.size(); i++){
        String s = (String)strings.get(i);
        if(i > 0 && currentCollator.compare(s, strings.get(i-1)) == 0 )
                  sortedWords.append("=");
        sortedWords.append(s + "\n");
    }
    }
}

class EnumCombo extends JComboBox {
    public EnumCombo(Class cl, String[] labels){
        for(int i = 0; i < labels.length; i++){
              String label = labels[i];
              String name = label.toUpperCase().replace(' ', '_');
              int value = 0;
              try{
                    java.lang.reflect.Field f = cl.getField(name);
```

```
                value = f.getInt(cl);
            }catch(Exception e){
                label = "(" + label + ")";
            }
            table.put(label, new Integer(value));
            addItem(label);
        }
        setSelectedItem(labels[0]);
    }

    public int getValue(){
        return ((Integer)table.get(getSelectedItem())).intValue();
    }
    private Map table = new HashMap();
}
```

java.text.Collator API

- `static Locale[] getAvailableLocales()`

  Nhận lại một mảng các đối tượng của `Locale` mà `Collator` hỗ trợ.
- `static Collator getInstance()`
- `static Collator getInstance(Locale l)`

  Đưa lại đối tượng đối sánh `Collator` sử dụng cho khu vực mặc định hay khu vực được cho bởi `l`.
- `int compare(String a, String b)`

  So sánh hai xâu, cho lại giá trị âm nếu `a` đứng trước `b`, bằng 0 nếu `a` và `b` đồng nhất với nhau, giá trị dương trong trường hợp ngược lại.
- `boolean equals(String a, String b)`

  Cho lại giá trị `true` nếu hai xâu `a`, `b` đồng nhất với nhau, ngược lại là `false`.
- `void setStrength(int strength)`
- `int getStrength()`

  Đặt và nhận lại mức độ của bộ đối sánh hiện thời. Tham số `strength` có thể là một trong các hằng `Collator.PRIMARY`, `Collator.SECONDARY`, `Collator.TERTIARY`.
- `void setdDecomposition(int decomp)`

- `int getdDecomposition()`

  Đặt và lấy ra đối tượng phân tích của bộ đối sánh. decomp có thể là `Collator.NO_DECOMPOSITION`, `Collator.CANONICAL_DECOMPOSITION`, `Collator.FULL_DECOMPOSITION`.

8.5.2. Ranh giới của các văn bản Text

Một vấn đề rất quan trọng trong xử lý văn bản là phải xác định được các ranh giới (phần tử ngắt) giữa các ký tự, từ, câu và các dòng .

Ngắt từ là ký tự bắt đầu và kết thúc của các từ. Trong tiếng Anh, người ta định nghĩa *một từ* là một dãy các ký tự, trong đó không có dấu cách ' '. Ví dụ, câu

```
The quick, brown fox jump-ed over the lazy dog.
```

được ngắt bởi các dấu cách ' ', dấu ',' và dấu '.' thành các từ như sau:

```
The| |quick|,| |brown| |fox| |jump-ed| |over| |the| |lazy| |dog|.|
```

Ngắt dòng là các vị trí mà ở đó một dòng bị ngắt ở trên màn hình hoặc ở trên văn bản được in ra. Trong văn bản tiếng Anh, các dòng được ngắt bởi một từ trước đó hoặc bởi một dấu gạch nối. Lưu ý, các điểm ngắt dòng là những vị trí mà ở đó một dòng có thể bị ngắt chứ không nhất thiết phải là những điểm mà nó bị ngắt thực sự. Ví dụ, dòng sau có các điểm ngắt là:

```
The |quick, |brown |fox |jump-|ed |over |the |lazy |dog.|
```

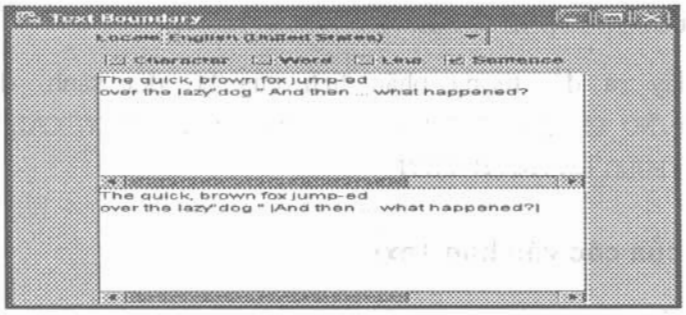
Việc xác định ranh giới giữa các ký tự, các từ và các dòng là tương đối đơn giản đối với ngôn ngữ tiếng Anh, tiếng Việt và nhiều ngôn ngữ của châu Âu, Á. Tuy nhiên, nó có thể rất phức tạp đối với một số ngôn ngữ khác, ví dụ như tiếng Hindi.

Tiếp theo chúng ta tìm hiểu về việc *ngắt các câu văn bản*. Trong tiếng Anh (tiếng Việt), một câu bị ngắt bởi các dấu chấm câu '.', các dấu cảm thán '!' và dấu '?'.

Ta có thể sử dụng lớp `BreakIterator` để tìm các chỗ ngắt của ký tự, từ, dòng và các câu trong một văn bản. Ta cũng có thể sử dụng chúng để soạn thảo, hiển thị và in các tài liệu văn bản.

```
Locale loc = ...;                    // Xác định khu vực của hệ thống
// Xác định đối tượng tách từ
BreakIterator wordIter =  BreakIterator.getWordInstance(loc);
String msg = "The quick, brown fox";
wordIter.setText(msg);   // Đặt msg vào bộ lặp để tách các từ
```

Ví dụ 8.4. Chương trình cho phép nhập vào một văn bản ở phía trên và lựa chọn `Character`, `Word`, `Line`, `Sentence` để xác định các ranh giới giữa các ký tự, từ, dòng hay câu tương ứng theo khu vực đã chọn ở `ComboBox` (Xem hình 8.4).



Hình 8.4. **Chương trình xác định các ranh giới trong văn bản**

```
// TextBoundaryTest.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.util.*;

public class TextBoundaryTest{
   public static void main(String args[]){
      JFrame fr = new TextBoundaryFrame();
      fr.show();
   }
}

class TextBoundaryFrame extends JFrame{

   private Locale[] locales;
   private BreakIterator currentBreak;

   private JComboBox combo = new JComboBox();
   private JTextArea inputText = new JTextArea(6, 40);
   private JTextArea outputText = new JTextArea(6, 40);
   private ButtonGroup cbButton = new ButtonGroup();
   private JCheckBox charBox = new JCheckBox("Character");
   private JCheckBox wordBox = new JCheckBox("Word");
   private JCheckBox lineBox = new JCheckBox("Line");
   private JCheckBox sentenceBox = new JCheckBox("Sentence");
```

```
public TextBoundaryFrame(){
    setSize(400, 400);
    setTitle("Text Boundary");
    addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e){
              System.exit(0);
         }
     });

    ActionListener listener =
         new ActionListener(){
              public void actionPerformed(ActionEvent evt){
                   updateDisplay();
              }
    };

    JPanel p = new JPanel();
    addCheckBox(p, charBox, cbButton, listener, false);
    addCheckBox(p, wordBox, cbButton, listener, false);
    addCheckBox(p, lineBox, cbButton, listener, false);
    addCheckBox(p, sentenceBox, cbButton, listener, true);

    getContentPane().setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.fill = GridBagConstraints.NONE;
    gbc.anchor = GridBagConstraints.EAST;
    add(new JLabel("Locale"), gbc, 0, 0, 1, 1);
    gbc.anchor = GridBagConstraints.WEST;
    add(combo, gbc, 1, 0, 1, 1);
    add(p, gbc, 0, 1, 2, 1);
    gbc.fill = GridBagConstraints.BOTH;
    gbc.weighty = 100;
    add(new JScrollPane(inputText), gbc, 0, 2, 2, 1);
    add(new JScrollPane(outputText), gbc, 0, 3, 2, 1);
```

```
        locales = BreakIterator.getAvailableLocales();
        for(int i = 0; i < locales.length; i++)
            combo.addItem(locales[i].getDisplayName());
        combo.setSelectedItem(Locale.getDefault().getDisplayName());

        combo.addActionListener(listener);

        inputText.setText("The quick, brown fox jump-ed\n" +
            "over the lazy\"dog.\" And then ... what happened?");
        updateDisplay();
    }

    public void addCheckBox(Container p, JCheckBox checkBox,
            ButtonGroup g, ActionListener listener, boolean v)
        {
        checkBox.setSelected(v);
        checkBox.addActionListener(listener);
        g.add(checkBox);
        p.add(checkBox);
    }

    public void add(Component c, GridBagConstraints gbc,
        int x, int y, int w, int h){
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridheight = h;
        gbc.gridwidth = w;
        getContentPane().add(c, gbc);
    }
    public void updateDisplay(){
        Locale currentLocale = locales[combo.getSelectedIndex()];
        BreakIterator currentBreak = null;
     if(charBox.isSelected())
        currentBreak = BreakIterator.getCharacterInstance(currentLocale);
```

```
        else if(wordBox.isSelected())
            currentBreak = BreakIterator.getWordInstance(currentLocale);
        else if(lineBox.isSelected())
            currentBreak = BreakIterator.getLineInstance(currentLocale);
        else if(sentenceBox.isSelected())
            currentBreak = BreakIterator.getSentenceInstance(currentLocale);
        String text = inputText.getText();
        currentBreak.setText(text);
        outputText.setText("");
        int from = currentBreak.first();
        int to;
        while((to = currentBreak.next()) != BreakIterator.DONE){
            outputText.append(text.substring(from, to) + "|");
            from = to;
        }
    }
}
```

java.text.BreakIterator API

- `static Locale[] getAvailableLocales()`

  Nhận lại một mảng các đối tượng của `Locale` mà `BreakIterator` hỗ trợ.
- `static BreakIterator getCharacterInstance()`
- `static BreakIterator getCharacterInstance(Locale l)`
- `static BreakIterator getWordInstance()`
- `static BreakIterator getWordInstance(Locale l)`
- `static BreakIterator getLineInstance()`
- `static BreakIterator getLineInstance(Locale l)`
- `static BreakIterator getSentenceInstance()`
- `static BreakIterator getSentenceInstance(Locale l)`

  Nhận lại bộ lặp để tách các ký tự, từ, dòng và câu theo khu vực mặc định hay khu vực được cho bởi `l`.
- `void setText(String text)`
- `void setText(CharacterIterator text)`

- `CharacterIterator getText()`

  Đặt và nhận lại văn bản `text` để phân tích.

- `int first()`

  Chuyển về vị trí của ranh giới đầu tiên của xâu đang đọc và đưa lại chỉ số của nó.

- `int next()`

  Chuyển về ranh giới tiếp theo và đưa lại chỉ số của vị trí đó. Nếu đạt đến cuối xâu thì trả lại `BreakIterator.DONE`.

- `int previous()`

  Chuyển về ranh giới phía trước và đưa lại chỉ số của vị trí đó. Nếu đạt đến đầu xâu thì trả lại `BreakIterator.DONE`.

- `int last()`

  Chuyển về ranh giới cuối cùng và đưa lại chỉ số của vị trí đó.

- `int current()`

  Cho lại vị trí của ranh giới hiện thời.

- `int next(int n)`

  Chuyển đến ranh giới thứ `n` kể từ vị trí hiện thời và trả lại chỉ số của vị trí đó. Nếu `n` là số âm thì chuyển về phía trước, ngược lại chuyển về phía sau. Nếu đạt đến đầu xâu thì trả lại `BreakIterator.DONE`.

8.6. BỌC TÀI NGUYÊN

Khi thực hiện bản địa hoá một phần mềm, ta thường gặp phải vấn đề gây nhiều phiền phức là phải dịch rất nhiều thông báo, tên gọi của các nút, nhãn, các mục, v.v. sang tiếng bản địa. Để cho tiện lợi hơn, ta có thể định nghĩa các xâu ở bên ngoài và đặt vào các *bọc tài nguyên*, độc lập với chương trình, được gọi là *tệp tài nguyên*. Các tệp tài nguyên được biên soạn, dịch sang tiếng bản địa mà không ảnh hưởng gì đến mã nguồn của chương trình.

8.6.1. Bản địa hóa tài nguyên

Công nghệ `Java` cung cấp lớp `ResourceBundle` hỗ trợ để ta thực hiện được các yêu cầu trên. Ta có thể tạo ra một lớp khác cho mỗi khu vực, sau đó gọi phương thức `getBundle()` của `ResourceBundle` để xác định tự động lớp theo yêu cầu. Các lớp đó phải kế thừa từ lớp `ResourceBundle`.

Để thực hiện được các yêu cầu trên, ta phải tuân theo qui ước đặt tên cho các lớp này tương ứng với từng khu vực. Ví dụ, tài nguyên dành riêng người Đức thì lớp đó là

ProgramResourcesde_de, còn tài nguyên cho khu vực những người nói tiếng Đức là ProgramResourcesde_de_DE. Ta sử dụng qui ước

```
ProgramResources_language_country
```

để đặt tên các tệp tài nguyên cho tất cả các nước theo country cụ thể, và

```
ProgramResources_language
```

để đặt tên các tệp tài nguyên cho tất cả các khu vực nói tiếng language.

Khi đã có một lớp bọc tài nguyên, ta có thể tải nó xuống bằng lệnh

```
ResourceBundle currentResource =
    ResourceBundle.getBundle("ProgramResources", currentLocale);
```

Hàm getBundle() sẽ thử tải xuống một trong các lớp tài nguyên sau:

```
ProgramResources_language_country_variant
ProgramResources_language_country
ProgramResources_language
ProgramResources
```

Nếu không tải xuống được thì nó thử làm một lần nữa, nhưng lần này không phải áp dụng với đối tượng khu vực hiện thời mà là với khu vực mặc định. Trường hợp không thể tải xuống được thì nó phát sinh ra ngoại lệ MissingResourceException.

8.6.2. Đặt tài nguyên vào các bọc

Trong Java, ta có thể đặt tài nguyên vào trong các lớp được mở rộng từ lớp ResourceBundle. Mỗi bọc tài nguyên cài đặt như là một bảng tra cứu tìm các từ tương ứng theo các từ khoá.

Khi thiết kế chương trình, ta phải cung cấp xâu chính (từ khoá) cho mỗi lần thiết lập vùng bản địa và sau này sử dụng nó để truy tìm đối tượng khu vực.

```
String computeButtonLabel
    = resources.getString("computeButton");
```

Ta cũng có thể sử dụng hàm getObject() để xác định đối tượng bất kỳ từ bọc tài nguyên.

```
Color backgroundColor
    = (Color)resources.getObject("backgroundColor");
double[] paperSize
    = (double[])resources.getObject("defaultPaperSize");
```

Để cài đặt lớp bọc tài nguyên riêng, ta phải cài đặt hai hàm

```
Enumeration getKeys()
Object handleGetObject(String key)
```

Ví dụ, lớp sau đây cung cấp tài nguyên để dịch cho khu vực nói Đức và người Mỹ nói tiếng Anh.

```
public class ProgramResources extends ResourceBundle{
    // Đặt hàm getKeys() vào lớp cơ sở
    public Enumeration getKeys(){
      return Collections.enumeration(Arrays.asList(keys));
    }
    private String[] keys = {"computeButton",
            "backgroundColor", "defaultPaperSize"};
    }
}
public class ProgramResources_de extends ProgramResources{
    // Lớp tài nguyên cho khu vực nói tiếng Đức
    public Object handleGetObject(String key){
        if(key.equals("computeButton"))
            return "Rechmen";
        else if(key.equals("backgroundColor"))
            return Color.black;
        else if(key.equals("defaultPaperSize"))
            return new double[]{210, 297};
    }
}
public class ProgramResources_en_US extends ProgramResources{
    // Lớp tài nguyên cho khu vực người Mỹ nói tiếng Anh
    public Object handleGetObject(String key){
        if(key.equals("computeButton"))
            return "Compute";
        else if(key.equals("backgroundColor"))
            return Color.blue;
        else if(key.equals("defaultPaperSize"))
            return new double[]{216, 279};
    }
}
```

Hiển nhiên, công việc viết các mã lệnh như trên là khá buồn chán. Thư viện chuẩn của Java cung cấp hai loại lớp tài nguyên khác là `ListResourceBundle` và `PropertyResourceBundle` giúp cho việc tổ chức chúng dễ dàng hơn.

a) `ListResourceBundle` cho phép đặt tài nguyên vào một mảng các đối tượng.

```
public class ProgramResources_language_country
        extends ListResourceBundle{
  public Object[] getContents(){
        return contents;
  }
  static final Object[][] contents = {
        // Các thông tin được bản địa hoá
        };
}
```

Ví dụ trên có thể viết lại như sau:

```
public class ProgramResources_de extends ListResourceBundle{
   public Object[] getContents(){
      return contents;
   }
   static final Object[][] contents = {
      {"computeButton", "Rechmen"},
      {"backgroundColor", Color.black}
      {"defaultPaperSize", new double[]{210, 297}}
   };
}

public class ProgramResources_en_US extends ListResourceBundle{
   public Object[] getContents(){
      return contents;
   }
   static final Object[][] contents = {
      {"computeButton", "Compute"},
      {"backgroundColor", Color.blue}
      {"defaultPaperSize", new double[]{216, 279}}
   };
}
```

b) PropertyResourceBundle cho phép đặt các tài nguyên một cách linh hoạt hơn. Ta có thể tạo ra *tệp tài sản* của tài nguyên đơn giản *là tệp văn bản bao gồm từng cặp giá trị trên mỗi dòng*, viết dưới dạng

```
computeButton=Compute
backgroundColor=blue
defaultPaperSize=216x279
```

Sau đó ta mở tệp tài sản này và truyền nó vào cho toán tử tạo lập của PropertyResourceBundle.

```
InputStream in = ...; // Mở tệp tài sản
PropertyResourceBundle currentResources
    = new PropertyResourceBundle(in);
```

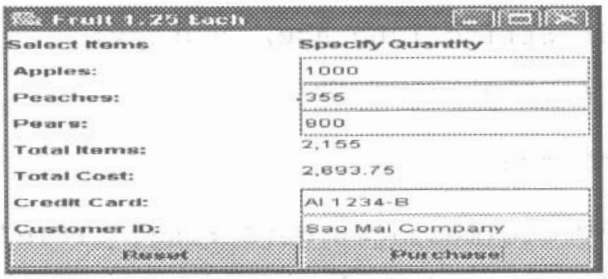
Các tệp tài sản thường có tên trùng với tên của các tài nguyên và có phần mở rộng là .property, ví dụ ProgramResources_en_US.property là tệp tài sản dùng cho những người Mỹ nói tiếng Anh. Ta có thể mở tệp tài sản thông qua hàm getResourceAsStream() của lớp Class như sau:

```
in = Program.class.getResourceAsStream(
    "ProgramResources_en_US.propety");
PropertyResourceBundle currentResources
    = new PropertyResourceBundle(in);
```

Lưu ý: Như vậy đối với những chương trình cần được bản địa hoá thì cần phải tạo ra hai loại tệp: các tệp lớp (.class) và các tệp tài sản (.property). Tệp tài sản là các tệp văn bản gồm các cặp giá trị từ khoá / giá trị (key/value), trong đó giá trị value chính là tên được bản địa hoá của từ khoá key. Cả hai loại tệp này phải được đặt vào cùng một thư mục hoặc gộp vào trong một tệp JAR.

Khi cần tìm hiểu sâu về vấn đề quốc tế hoá và bản địa hoá phần mềm, bạn có thể sử dụng công cụ jikit (http://java.sun.com/products/jikit) để quản lý việc tạo lập và duy trì các bọc tài nguyên.

Ví dụ 8.5. Chương trình hiển thị các mục trong đơn đặt mua hoa quả (táo, đào, mận) theo ngôn ngữ và các đại lượng số theo khuôn mẫu qui định của từng khu vực. Người sử dụng nhập vào số lượng các loại hoa quả cần mua, và số thẻ tín dụng, mã khách hàng (tuỳ chọn), rồi nhấn nút "Đặt mua" ("Purchase" nếu là khu vực nói tiếng Anh) thì tổng số hoa quả đặt mua và số tiền phải trả sẽ được hiển thị theo đúng qui định của khu vực đã lựa chọn, như hình 8.5 và 8.6. Mặc định là khu vực nói tiếng Anh như hình 8.5. Khi cần nhập lại, hay đặt đơn hàng mới thì nhấn nút "Nhập lại" (đối với người Việt Nam) hay ("Reset" nếu là khu vực nói tiếng Anh).



Hình 8.5. **Đơn đặt hàng bằng tiếng Anh**

Hình 8.6. **Đơn đặt hàng bằng tiếng Việt**

// ClientOrder.java

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import java.util.*;
import java.text.*;

public class ClientOrder extends JFrame implements ActionListener {
      JLabel col1, col2;
      JLabel totalItems, totalCost;
      JLabel cardNum, custID;
      JLabel appleLabel, pearLabel, peachLabel;
      JButton purchase, reset;
      JPanel panel;
      JTextField appleNum, pearNum, peachNum;
      JTextField creditCard, customer;
      JTextArea items, cost;

      static Locale currentLocale;
      static ResourceBundle messages;
```

```
static String language, country;
NumberFormat numFormat;

ClientOrder(){

  setTitle(messages.getString("title"));
  col1 = new JLabel(messages.getString("1col"));
  col2 = new JLabel(messages.getString("2col"));

  appleLabel = new JLabel(" " + messages.getString("apples"));
  appleNum = new JTextField();

  pearLabel = new JLabel(" " + messages.getString("pears"));
  pearNum = new JTextField();

  peachLabel = new JLabel(" " + messages.getString("peaches"));
  peachNum = new JTextField();

  cardNum = new JLabel(" " + messages.getString("card"));
  creditCard = new JTextField();

  customer = new JTextField();
  custID = new JLabel(" " + messages.getString("customer"));

  totalItems = new JLabel(" " + messages.getString("items"));
  totalCost = new JLabel(" " + messages.getString("cost"));

  items = new JTextArea();
  cost = new JTextArea();

  purchase = new JButton(messages.getString("purchase"));
  purchase.addActionListener(this);

  reset = new JButton(messages.getString("reset"));
  reset.addActionListener(this);

  panel = new JPanel();
  panel.setLayout(new GridLayout(0,2));
```

```
        panel.setBackground(Color.white);
        getContentPane().add(panel);
        panel.add(col1);
        panel.add(col2);

        panel.add(appleLabel);
        panel.add(appleNum);

        panel.add(peachLabel);
        panel.add(peachNum);

        panel.add(pearLabel);
        panel.add(pearNum);

        panel.add(totalItems);
        panel.add(items);

        panel.add(totalCost);
        panel.add(cost);

        panel.add(cardNum);
        panel.add(creditCard);

        panel.add(custID);
        panel.add(customer);

        panel.add(reset);
        panel.add(purchase);
    }

    public void actionPerformed(ActionEvent event){
       Object source = event.getSource();
       if (source == reset){
            creditCard.setText("");
            appleNum.setText("");
            peachNum.setText("");
            pearNum.setText("");
```

```
            customer.setText("");
            items.setText("");
            cost.setText("");
        }else if(source == purchase){
            int totalNum;
            double totalCost;
            totalNum = Integer.valueOf(appleNum.getText()).intValue()
                  + Integer.valueOf(pearNum.getText()).intValue()
                  + Integer.valueOf(peachNum.getText()).intValue();
            totalCost = totalNum * 1.25;
            numFormat = NumberFormat.getInstance(currentLocale);
            String text = numFormat.format(totalNum);
            items.setText(text);
            text = numFormat.format(totalCost);
            cost.setText(text);
        }
    }

    public static void main (String args[]){
        if (args.length != 2){ // Mặc định
            language = new String("en");
            country = new String("US");
            System.out.println("English");
        }
        else{// Đặt lại theo khu vực được đưa vào dưới dạng tham số của chương trình
            language = new String(args[0]);
            country = new String(args[1]);
            System.out.println(language + " " + country);
        }

        currentLocale = new Locale (language, country);
        // Đọc tệp bọc tài nguyên theo khu vực (tên quốc gia và ngôn ngữ)
        messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
```

```
        WindowListener l = new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        };
        ClientOrder frame = new ClientOrder();
        frame.addWindowListener(l);
        frame.setSize(300, 300);
        frame.setVisible(true);
    }
}
```

Như trên đã phân tích, để chương trình trên hiển thị đơn hàng theo khu vực thì ta phải tạo ra các tệp tài sản tương ứng. Ví dụ, muốn hiển thị đơn đặt mua hàng bằng tiếng Việt thì phải sử dụng một hệ soạn thảo nào đó để tạo ra tệp dạng văn bản (text) là MessagesBundle_vn_VN.properties (xem hình 8.6) có nội dung như sau.

```
apples = Táo:
peaches = Đào
pears = Mận:
items = Số hàng mua:
cost = Số tiền trả:
card = Credit Card:
customer = Mã hiệu KH:
title = Hoa Quả 2500đ/quả
1col = Chọn hàng
2col = Số lượng
reset = Nhập mới
view = Xem
purchase = Đặt mua
```

Hoặc, để hiển thị đơn đặt mua hàng bằng tiếng Pháp thì phải sử dụng một hệ soạn thảo nào đó để tạo ra tệp dạng văn bản (text) là:

MessagesBundle_fr_FR.properties có nội dung như sau.

```
apples=Pommes:
peaches=Peches:
pears=Poires:
items=Partial total:
cost=Cost total:
card=Carde de Credit:
customer=Numero de Client:
title=Fruit 1,25 pièce
1col=Choisissez les éléments
2col=Indiquez la quantite
reset=Réinitialisez
view=Visualisez
purchase=Achetez
```

Tương tự, ta có thể tạo ra các tệp tài sản cho tất cả các khu vực mà ta muốn.

Sau khi dịch thành công tệp ClientOrder.java, ta thực hiện chương trình này với tham số mặc định sẽ được như hình 8.5.

```
>java ClientOrder
```

Để chương trình hiển thị đơn đặt hàng bằng tiếng Việt Nam như hình 8.6, ta thực hiện chương trình với tệp tài nguyên như sau:

```
>java ClientOrder vn VN
```

java.util.ResourceBundle **API**

- `static ResourceBundle getBundle(String baseName, LOcale l)`
- `static ResourceBundle getBundle(String baseName)`

  Nạp lớp bọc tài nguyên có tên baseName và dịch sang khu vực l hoặc khu vực mặc định .
- `OBject getObject(String name)`

  Tìm đối tượng từ bọc tài nguyên hoặc từ các lớp cha của nó.
- `String getString(String name)`

  Tìm đối tượng từ bọc tài nguyên hoặc từ các lớp cha của nó và ép kiểu về String.

- `String[] getStringArray(String name)`

  Tìm đối tượng từ bọc tài nguyên hoặc từ các lớp cha của nó và ép kiểu về mảng của các xâu.

- `Enumeration getKeys()`

  Lấy ra dãy các từ khoá theo thứ tự được liệt kê.

8.7. BẢN ĐỊA HÓA CÁC GIAO DIỆN ĐỒ HỌA

Ở trên chúng ta đã đề cập đến những vấn đề chính của công việc bản địa hoá phần mềm. Phần này, chúng ta giải thích cách địa phương hoá các yêu cầu để thay đổi các mã lệnh mà bạn đã viết. Chúng ta xét phương thức lập trình sau:

```
public class MyApplet extends JApplet implements ActionListener{
    public void init(){
        JButton cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(this);
        ...
    }
    public void actionPerformed(ActionEvent evt){
        String arg = evt.getActioncommand();
        if(arg.equals("Cancel"))
            doCancel();
        else ...
        }
}
```

Dễ nhận thấy, tên gọi của nút nhấn "`Cancel`" sử dụng tốt cho những người biết tiếng Anh, nhưng khi chương trình sử dụng cho người Việt thì phải dịch thành "`Huỷ bỏ`". Do vậy, các tên gọi, thông báo phải được cập nhật tự động cả ở phương thức `init()` và `actionPerformed()`. Nhưng nếu vì một lý do nào đấy mà ta quên không cập nhật một trong các hàm đó thì những nút này sẽ không thực hiện được như mong muốn. Để khắc phục vấn đề nêu trên, ta có thể thực hiện theo các chiến lược:

1. Sử dụng các lớp bên trong thay vì sử dụng các phương thức riêng `actionPerformed()`.

2. Xác định các thành phần thông qua các tham chiếu của chúng chứ không phải bằng các tên của các nhãn.
3. Sử dụng thuộc tính name của lớp Component để xác định các thành phần.

Chúng ta lần lượt xét các cách nêu trên.

1. Sử dụng lớp bên trong

Thay vì tạo ra một bộ xử lý để xử lý cho nhiều hành động, ta có thể định nghĩa một bộ xử lý riêng cho mỗi thành phần. Ví dụ, đoạn chương trình trên được viết thành:

```
public class MyApplet extends JApplet implements ActionListener{
  public void init(){
     JButton cancelButton = new JButton("Cancel");
     cancelButton.addActionListener(
     new ActionListener(){
        public void actionPerformed(ActionEvent evt){
          doCancel();
     }
    });
   ...
   }
}
```

Chương trình tạo ra một lớp bên trong để lắng nghe sự kiện khi nút nhấn Cancel. Do vậy, chỉ có một chỗ xuất hiện tên của thành phần cần được dịch sang tiếng bản địa.

2. Xác định các thành phần thông qua các tham chiếu

Nhiều khi ta không muốn sử dụng những lớp bên trong một lớp khác, bởi vì như thế có thể dẫn đến sự nhập nhằng trong một số thao tác, như đọc dữ liệu chẳng hạn. Chúng ta có cách thực hiện khác là đặt các thành phần vào các biến thể hiện và so sánh chúng với câu lệnh tương ứng để xử lý các sự kiện.

```
public class MyApplet extends JApplet implements ActionListener{
   JButton cancelButton;
      public void init(){
      cancelButton = new JButton("Cancel");
      cancelButton.addActionListener(this);
      ...
   }
```

```
    public void actionPerformed(ActionEvent evt){
       Object source = evt.getSource();
       if(source == cancelButton)
            doCancel();
       else ...
       }
}
```

3. Sử dụng thuộc tính name của lớp Component để xác định các thành phần

Các thành phần mà chúng ta xây dựng đều là lớp con của `Component`, vì thế có thể sử dụng thuộc tính `name` thông qua hàm `getName()` để xác định thành phần tương ứng. Đặc trưng tên gọi của thành phần là bất biến đối với mọi sự thay đổi tên (nhãn) của chúng khi cần bản địa hoá chương trình. Ví dụ, nếu ta gắn cho nút "Huỷ bỏ" với tên gọi là "`Cancel1`" thông qua `name` bằng cách sử dụng `setName("Cancel1")` trong chương trình thì nó không phải là nhãn được hiển thị của nút đó, mà là xâu tương ứng.

```
public class MyApplet extends JApplet implements ActionListener{
   public void init(){
      JButton cancelButton = new JButton("Cancel");
      cancelButton.setName("Cancel1");
      cancelButton.addActionListener(this);
      ...
   }
   public void actionPerformed(ActionEvent evt){
      Component source = evt.getSource();
      if(source.getName().equals("Cancel1")
           doCancel();
      else ...
      }
}
```

Như vậy, nếu nhãn của nút huỷ bỏ được hiển thị là "`Cancel`" (trong tiếng Anh), và khi cần được dịch sang một ngôn ngữ khác, ví dụ được dịch sang tiếng Việt Nam là "`Huỷ bỏ`" thì tên đặt trong chương trình của nút đó vẫn không bị thay đổi, nó vẫn là "`Cancel1`".

Ví dụ 8.6. Chương trình `applet` tính toán và hiển thị biểu đồ số tiền tiết kiệm và khả năng thu nhập khi nghỉ hưu (nghỉ làm việc) ở nhiều nước khác nhau như hình 8.7, 8.8.

Người sử dụng có thể chọn khu vực (`English, China, Germany`) và nhập vào:

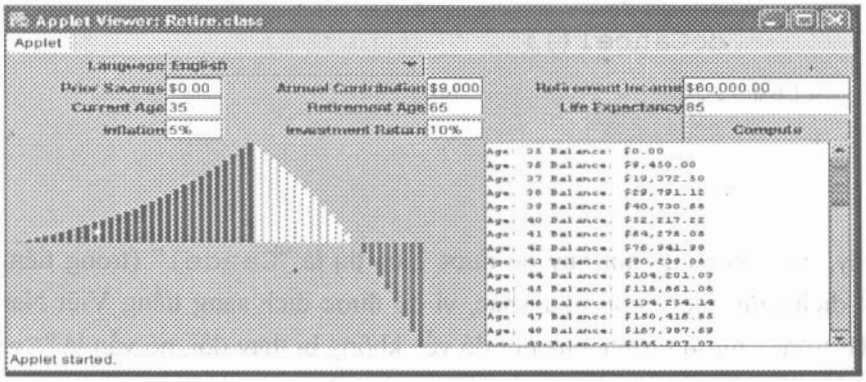
- Số tiền tiết kiệm được từ trước (`Prior Savings`),
- Số tiền tiết kiệm hàng năm (`Annual Contribution`),
- Thu nhập khi nghỉ việc (`Retirement Income`),
- Tuổi hiện nay (`Current Age`),
- Tuổi nghỉ hưu (`Retirement Age`),
- Dự kiến tuổi thọ (`Life Expectancy`),
- Tỷ lệ lạm phát (`Inflation`),
- Tỷ lệ tăng trưởng (`Investment Return`).

Sau khi nhập các số liệu nêu trên, nhấn nút "`Compute`" để tính và vẽ lại biểu đồ về thu nhập theo các độ tuổi về hưu.

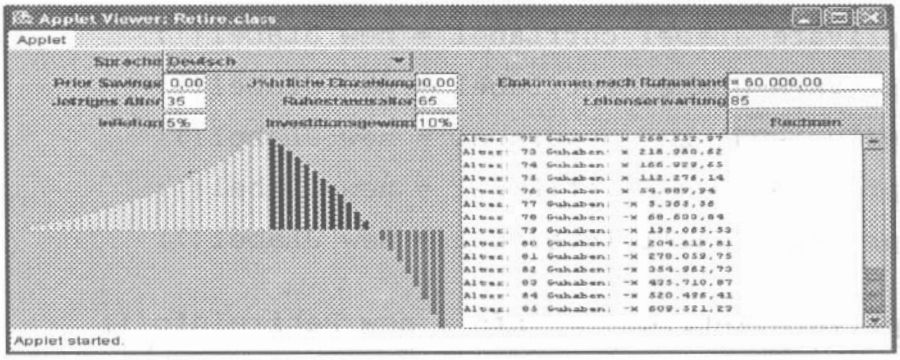
Vùng văn bản hiển thị số tiền dư hàng năm từ năm hiện thời cho đến tuổi thọ dự kiến và tương ứng là biểu đồ mô tả ở vùng bên trái.

Chương trình tính toán tiền tiết kiệm và khả năng thu nhập khi nghỉ làm việc chạy thử cho ba nước: Anh, Đức và Việt Nam. Sau đây là một số điểm cần chú ý trong chương trình.

- Các nhãn, tên nút và các thông báo được dịch sang tiếng Đức, tiếng Việt tương ứng có thể tìm thấy trong các lớp: `RetireResources`, `RetireResources_de`, `RetireResources_zh`.
- Các trường văn bản xử lý các dữ liệu số, tiền tệ và tỷ lệ % theo khuôn dạng của từng vùng bản địa.
- Trường tính toán sử dụng `MessageFormat`.
- Ta có thể chọn màu khác nhau để hiển thị biểu đồ cho từng nước theo yêu cầu của người sử dụng.



Hình 8.7. **Tính toán tiền được lĩnh khi nghỉ việc (về hưu) ở Anh (Mỹ)**



Hình 8.8. **Tính toán tiền được lĩnh khi nghỉ việc (về hưu) ở Đức**

```
// Retire.java
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.util.*;
import java.applet.*;

public class Retire extends JApplet{
   private JTextField savingsField = new JTextField(10);
   private JTextField contribField = new JTextField(10);
   private JTextField incomeField = new JTextField(10);
   private JTextField currentAgeField = new JTextField(4);
   private JTextField retireAgeField = new JTextField(4);
   private JTextField deathAgeField = new JTextField(4);
   private JTextField inflationPercentField = new JTextField(6);
   private JTextField investPercentField = new JTextField(6);
   private JTextArea retireText = new JTextArea(10, 25);
   private RetireCanvas retireCanvas = new RetireCanvas();
   private JButton computeButton = new JButton();
   private JLabel langLabel = new JLabel();
   private JLabel saviLabel = new JLabel();
   private JLabel contLabel = new JLabel();
   private JLabel incoLabel = new JLabel();
```

```
private JLabel currLabel = new JLabel();
private JLabel retiLabel = new JLabel();
private JLabel deadLabel = new JLabel();
private JLabel inflLabel = new JLabel();
private JLabel inveLabel = new JLabel();

private RetireInfo info = new RetireInfo();

private Locale[] locales;
private Locale currentLocale;
private JComboBox comboBox = new JComboBox();
private ResourceBundle res;
private NumberFormat currencyFmt;
private NumberFormat numberFmt;
private NumberFormat percentFmt;

public void init(){
    GridBagLayout gbl = new GridBagLayout();
    getContentPane().setLayout(gbl);

    GridBagConstraints gbc = new GridBagConstraints();
    gbc.weightx = 100;
    gbc.weighty = 0;

    gbc.fill= GridBagConstraints.NONE;
    gbc.anchor = GridBagConstraints.EAST;
    add(langLabel, gbc, 0, 0, 1, 1);
    add(saviLabel, gbc, 0, 1, 1, 1);
    add(contLabel, gbc, 2, 1, 1, 1);
    add(incoLabel, gbc, 4, 1, 1, 1);
    add(currLabel, gbc, 0, 2, 1, 1);
    add(retiLabel, gbc, 2, 2, 1, 1);
    add(deadLabel, gbc, 4, 2, 1, 1);
    add(inflLabel, gbc, 0, 3, 1, 1);
    add(inveLabel, gbc, 2, 3, 1, 1);
```

```
gbc.fill= GridBagConstraints.HORIZONTAL;
gbc.anchor = GridBagConstraints.WEST;
add(comboBox, gbc, 1, 0, 2, 1);
add(savingsField, gbc, 1, 1, 1, 1);
add(contribField, gbc, 3, 1, 1, 1);
add(incomeField, gbc, 5, 1, 1, 1);
add(currentAgeField, gbc, 1, 2, 1, 1);
add(retireAgeField, gbc, 3, 2, 1, 1);
add(deathAgeField, gbc, 5, 2, 1, 1);
add(inflationPercentField, gbc, 1, 3, 1, 1);
add(investPercentField, gbc, 3, 3, 1, 1);

computeButton.setName("computeButton");
computeButton.addActionListener(
  new ActionListener(){
    public void actionPerformed(ActionEvent evt){
                getInfo();
                updateData();
                updateGraph();
        }
});
add(computeButton, gbc, 5, 3, 1, 1);

gbc.weighty = 100;
gbc.fill= GridBagConstraints.BOTH;
add(retireCanvas, gbc, 0, 4, 4, 1);
add(new JScrollPane(retireText), gbc, 4, 4, 2, 1);
retireText.setEditable(false);
retireText.setFont(new Font("Monospaced", Font.PLAIN, 10));

info.setSavings(0);
info.setContrib(9000);
info.setIncome(60000);
info.setCurrentAge(35);
```

```
    info.setRetireAge(65);
    info.setDeathAge(85);
    info.setInvestPercent(0.1);
    info.setInflationPercent(0.05);

    comboBox.addActionListener(
      new ActionListener(){
          public void actionPerformed(ActionEvent evt){
                setCurrentLocale(comboBox.getSelectedIndex());
                }
    });

    locales = new Locale[]
                {Locale.US, Locale.CHINA, Locale.GERMAN};

    int localeIndex = 0; // Mặc định là US
    for(int i = 0; i < locales.length; i++)
          if(getLocale().equals(locales[i])) localeIndex = i;

    setCurrentLocale(localeIndex);
}
public void add(Component c, GridBagConstraints gbc,
    int x, int y, int w, int h){
    gbc.gridx = x;
    gbc.gridy = y;
    gbc.gridheight = h;
    gbc.gridwidth = w;
    getContentPane().add(c, gbc);
}

public void setCurrentLocale(int localeIndex){
    currentLocale = locales[localeIndex];

    comboBox.removeAllItems();
    String language;
```

```
        for(int i = 0; i < locales.length; i++){
          language = locales[i].getDisplayLanguage(currentLocale);
          comboBox.addItem(language);
        }

        comboBox.setSelectedIndex(localeIndex);
        res = ResourceBundle.getBundle("RetireResources", currentLocale);

        currencyFmt = NumberFormat.getCurrencyInstance(currentLocale);
        numberFmt = NumberFormat.getNumberInstance(currentLocale);
        percentFmt = NumberFormat.getPercentInstance(currentLocale);

        updateDisplay();
        updateInfo();
        updateData();
        updateGraph();
    }

    public void updateDisplay(){
        langLabel.setText(res.getString("language"));
        saviLabel.setText(res.getString("savings"));
        contLabel.setText(res.getString("contrib"));
        incoLabel.setText(res.getString("income"));
        currLabel.setText(res.getString("currentAge"));
        retiLabel.setText(res.getString("retireAge"));
        deadLabel.setText(res.getString("deathAge"));
    inflLabel.setText(res.getString("inflationPercent"));
        inveLabel.setText(res.getString("investPercent"));
    computeButton.setText(res.getString("computeButton"));

        validate();
    }

    public void updateInfo(){
     savingsField.setText(currencyFmt.format(info.getSavings()));
     contribField.setText(currencyFmt.format(info.getContrib()));
```

```
    incomeField.setText(currencyFmt.format(info.getIncome()));
    currentAgeField.setText(numberFmt.format(info.getCurrentAge()));
    retireAgeField.setText(numberFmt.format(info.getRetireAge()));
    deathAgeField.setText(numberFmt.format(info.getDeathAge()));
    inflationPercentField.setText(percentFmt.format(info.getInflationPercent()));
    investPercentField.setText(percentFmt.format(info.getInvestPercent()));
  }

  public void updateData(){
    retireText.setText("");
    MessageFormat retireMsg = new MessageFormat("");
    retireMsg.setLocale(currentLocale);
    retireMsg.applyPattern(res.getString("retire"));

    for(int i = info.getCurrentAge(); i <= info.getDeathAge(); i++){
      Object[] args = {new Integer(i), new Double(info.getBalance(i))};
      retireText.append(retireMsg.format(args) + "\n");
    }
  }

  public void updateGraph(){

      retireCanvas.setColorGain((Color)res.getObject("colorGain"));
      retireCanvas.setColorPre((Color)res.getObject("colorPre"));
      retireCanvas.setColorLoss((Color)res.getObject("colorLoss"));

      retireCanvas.setInfo(info);
      repaint();
  }
  public void getInfo(){
    try{
      info.setSavings(currencyFmt.parse(
            savingsField.getText()).doubleValue());
      info.setContrib(currencyFmt.parse(
            contribField.getText()).doubleValue());
```

```
            info.setIncome(currencyFmt.parse(
                  incomeField.getText()).doubleValue());
            info.setCurrentAge(numberFmt.parse(
                  currentAgeField.getText()).intValue());
            info.setRetireAge(numberFmt.parse(
                  retireAgeField.getText()).intValue());
            info.setDeathAge(numberFmt.parse(
                  deathAgeField.getText()).intValue());
            info.setInvestPercent(percentFmt.parse(
                  investPercentField.getText()).doubleValue());
            info.setInflationPercent(percentFmt.parse(
                inflationPercentField.getText()).doubleValue());
          }catch(ParseException e){
          }
       }
    }

    class RetireInfo{
       public double getBalance(int year){
         if(year < currentAge) return 0;
         else if(year == currentAge){
           age = year;
           balance = savings;
           return balance;
         }else if(year == age)
           return balance;
         else if(year != age + 1)
           getBalance(year - 1);
         age = year;
         if(age < retireAge)
           balance += contrib;
         else
           balance -= income;
        balance = balance * (1 + (investPercent - inflationPercent));
```

```
 return balance;
}
public double getSavings(){
   return savings;
}
public double getContrib(){
   return contrib;
}
public int getCurrentAge(){
   return currentAge;
}
public int getRetireAge(){
   return retireAge;
}
public double getIncome(){
   return income;
}
public int getDeathAge(){
   return deathAge;
}
public double getInflationPercent(){
   return inflationPercent;
}
public double getInvestPercent(){
   return investPercent;
}

public void setSavings(double x){
   savings = x;
}
public void setContrib(double x){
   contrib = x;
}
public void setIncome(double x){
```

```
        income = x;
    }
    public void setCurrentAge(int x){
        currentAge = x;
    }
    public void setRetireAge(int x){
        retireAge = x;
    }
    public void setDeathAge(int x){
        deathAge = x;
    }
    public void setInflationPercent(double x){
        inflationPercent = x;
    }
    public void setInvestPercent(double x){
        investPercent = x;
    }

    private double savings;
    private double contrib;
    private double income;
    private double inflationPercent;
    private double investPercent;
    private int currentAge;
    private int retireAge;
    private int deathAge;

    private int age;
    private double balance;
}

class RetireCanvas extends JPanel{
    public RetireCanvas(){
        setSize(300, 300);
    }
```

```
public void setInfo(RetireInfo newInfo){
    info = newInfo;
}
public void paint(Graphics g){
    if(info == null) return;
    double minValue = 0;
    double maxValue = 0;
    int i;
    for(i = info.getCurrentAge(); i <= info.getDeathAge(); i++){
        double v = info.getBalance(i);
        if(minValue > v) minValue = v;
        if(maxValue < v) maxValue = v;
    }
    if(maxValue == minValue) return;

    int barW = getWidth() / (info.getDeathAge() - info.getCurrentAge() + 1);
    double scale = getHeight() / (maxValue - minValue);
    for(i = info.getCurrentAge(); i <= info.getDeathAge(); i++){
        int x1 = (i - info.getCurrentAge()) * barW + 1;
        int y1;
        double v = info.getBalance(i);
        int h;
        int y = (int) (maxValue * scale);

        if(v >= 0){
            y1 = (int)((maxValue - v) * scale);
            h = y - y1;
        } else{
            y1 = y;
            h = (int) (-v * scale);
        }
        if(i < info.getRetireAge())
            g.setColor(colorPre);
```

```
                else if( v >= 0)
                    g.setColor(colorGain);
                else
                    g.setColor(colorLoss);
                g.fillRect(x1, y1, barW - 2, h);
                g.fillRect(x1, y1, barW - 2, h);
            }
        }

        public void setColorPre(Color c){
            colorPre = c;
        }
        public void setColorGain(Color c){
            colorGain = c;
        }
        public void setColorLoss(Color c){
            colorLoss = c;
        }

        private RetireInfo info = null;
        private Color colorPre;
        private Color colorGain;
        private Color colorLoss;
    }
```

// RetireResources.java: Lớp hiển thị tiếng Anh

```
import java.util.*;
import java.awt.*;
public class RetireResources extends ListResourceBundle{
    public Object[][] getContents(){
        return contents;
    }
    static final Object[][] contents ={
        {"language", "Language"},
        {"computeButton", "Compute"},
```

```
        {"savings", "Prior Savings"},
        {"contrib", "Annual Contribution"},
        {"income", "Retirement Income"},
        {"currentAge", "Current Age"},
        {"retireAge", "Retirement Age"},
        {"deathAge", "Life Expectancy"},
        {"inflationPercent", "Inflation"},
        {"investPercent", "Investment Return"},
        {"retire", "Age: {0, number} Balance: {1, number, currency}" },
        {"colorPre", Color.blue},
        {"colorGain", Color.white},
        {"colorLoss", Color.red}
    };
}
```

// RetireResources_de.java: Lớp hiển thị tiếng Đức

```
import java.util.*;
import java.awt.*;
public class RetireResources_de extends ListResourceBundle{
    public Object[][] getContents(){
        return contents;
    }
    static final Object[][] contents ={
        {"language", "Sprache"},
        {"computeButton", "Rechnen"},
        {"savings", "Prior Savings"},
        {"contrib", "J%hrliche Einzahlung"},
        {"income", "Einkommen nach Ruhestand"},
        {"currentAge", "Jetziges Alter"},
        {"retireAge", "Ruhestandsalter"},
        {"deathAge", "Lebenserwartung"},
        {"inflationPercent", "Inflation"},
```

```
        {"investPercent", "Investitionsgewinn"},
        {"retire", "Alter: {0, number} Guhaben: {1, number, currency}"},
        {"colorPre", Color.yellow},
        {"colorGain", Color.black},
        {"colorLoss", Color.red}
    };
}
```

// RetireResources_zh.java: Lớp hiển thị tiếng Việt

```
import java.util.*;
import java.awt.*;
public class RetireResources_zh extends ListResourceBundle{
    public Object[][] getContents(){
        return contents;
    }
    static final Object[][] contents ={
        {"language", "Ngon ngu"},
        {"computeButton", "Tinh"},
        {"savings", "Tich luy tu truoc"},
        {"contrib", "Thu nhap hang nam"},
        {"income", "Thu nhap khi nghi huu"},
        {"currentAge", "Tuoi hien nay"},
        {"retireAge", "Tuoi nghi huu"},
        {"deathAge", "Du kien tuoi tho"},
        {"inflationPercent", "Ty le lam phat"},
        {"investPercent", "Ty le tang truong"},
        {"retire", "Tuoi: {0, number} So du: {1, number, currency}"},
        {"colorPre", Color.yellow},
        {"colorGain", Color.black},
        {"colorLoss", Color.red}
    };
}
```

BÀI TẬP

8.1. Viết chương trình đặt mua và bán hàng qua mạng sử dụng `RMI`. Đơn đặt hàng được hiển thị theo khu vực của khách hàng (ví dụ bằng tiếng Việt) tương tự như ở ví dụ 8.5. Khi người mua nhấn nút "Đặt mua" thì đơn hàng này được gửi cho người bán và được lại được hiển thị bằng ngôn ngữ của người bán, ví dụ bằng tiếng Anh.

8.2. Viết chương trình đặt mua và bán hàng qua mạng sử dụng `Servlet`. Đơn đặt hàng được hiển thị theo khu vực của khách hàng (ví dụ bằng tiếng Việt) tương tự như ở ví dụ 8.5 . Khi người mua nhấn nút "Đặt mua" thì đơn hàng này được gửi cho người bán và được lại được hiển thị bằng ngôn ngữ của người bán, ví dụ bằng tiếng Anh.

DANH SÁCH CÁC THUẬT NGỮ ANH - VIỆT VÀ TỪ VIẾT TẮT

Account	Tài khoản
Application Programming Interface	Giao diện lập trình ứng dụng
Authenticated	Được xác thực, được chứng thực
Bean	Hạt nhân, thành phần hạt nhân
Bean Context	Ngữ cảnh của Bean
Blocked	Bị chặn
Bound property	Thuộc tính biên
Browser	Trình duyệt
Certificate	Chứng chỉ, giấy chứng nhận
Certificate Authority	Bộ phận, cơ quan chứng thực
Collection	Tuyển tập
Common Object Broker Request Architecture	Kiến trúc môi giới yêu cầu đối tượng chung: CORBA
Constrained property	Thuộc tính bị khống chế
Constructor	Toán tử tạo lập, tạo dựng, cấu tử
Datagram	Đồ hình dữ liệu
Deadlock	Chết tắc, tắc nghẽn
Destructor	Huỷ tử, toán tử huỷ bỏ
Digital Signature	Chữ ký số
Digital Certificate	Chứng chỉ số, chứng nhận số
Directory	Thư mục
Distributed	Phân tán
Encapsulation	Bao gói, gói gọn
Engine	Mô tơ, động cơ

Extend	Kế thừa, mở rộng
File	Tệp
Filter Model	Mô hình bộ lọc
Firewall	Bức tường lửa
Hierarchical file system	Hệ thống tệp phân cấp
Holder class	Lớp cất giữ
Holder object	Đối tượng cất giữ
Implement	Cài đặt
Inherit	Kế thừa
Intranet	Mạng nội hạt, nội bộ
Interface Definition Language	Ngôn ngữ định nghĩa giao diện: IDL
Internationalization	Quốc tế hoá
Introspection Report	Báo cáo thanh tra
Interrupted	Bị ngắt
Java Runtime Environment	Môi trường thực thi Java
Java Virtual Machine	Máy Java ảo
Mail Standard Format	Định dạng chuẩn thư điện tử
Message	Thông điệp, thỉnh cầu, thông báo
Message Digest	Bản tóm tắt thông điệp, dấu vết thông điệp
Method	Phương thức, hàm
Multitasking	Đa nhiệm
MultiThreading	Xử lý đa luồng
Localization	Bản địa hoá, địa phương hoá
Lool & feel	Thấy và cảm nhận được
Object lock	Khoá đối tượng
Object Request Broker	Bộ môi giới yêu cầu
Object Oriented	Hướng đối tượng
Option	Tuỳ chọn, tuỳ chỉnh
Overflow	Tràn bộ nhớ

Reference	Tham chiếu
Register	Thanh ghi
Remove interface	Giao diện từ xa
Remove Mothod Invocation	Triệu gọi phương thức từ xa: RMI
Remove object	Đối tượng từ xa
Remove Reference	Tham chiếu từ xa
Resource Bundle	Bọc tài nguyên
Package	Gói
Private key	Khoá riêng
Process	Tiến trình
Progress Meter	Thước đo tiến độ
Protocol	Giao thức
Public key cryptography	Mật mã khoá công khai
Safety	An toàn (sự an toàn)
Security	Bảo mật, an ninh
Security policy	Chính sách bảo mật, an ninh
Server's Internet Host	Máy chủ (Server) trên Intermet
Slider	Thanh trượt, thước đo
Socket	Cơ chế ổ cắm, bảng cắm
Standard Protocol	Giao thức chuẩn
Stub	Đại diện trên máy khách
Synchronization	Đồng bộ hoá
Thread	Luồng
Time-Sharing	Chia sẻ thời gian
Three-tier model	Mô hình ba tầng
Transport Layer	Tầng giao vận
Tree	Cây, cấu trúc cây
Verifier	Bộ kiểm tra

API	Application Programming Interface
ASP	Active Server Page
BDK	Bean Development Kit
B2B	Business-to-Business
CA	Certificate Authority
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DSA	Digital Signature Algorithm
EJB	Enterprise Java Bean
FTP	File Transfer Protocol
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
IE	Internet Explore
IMAP	Internet Message Access Protocol
IP	Internet Protocol
ITU-T	International Telecommunication Union-Telecommunication Standardization Sector
JDBC	Java DataBase Connectivity
JDK	Java Development Kit
JNDI	Java Naming Directory Interface
JSP	Java Server Page
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extension
ODBC	Open Data Base Connectivity
COM	Common Object Model
OMG	Object Management Group
ORB	Object Request Broker
POP3	Post Office Protocol
RFC	Requests For Comments
RMI	Remote Method Invocation
ROA	Remote Object Activation

RSA	Rivest – Shamir - Adleman
TCP	Transmission Control Protocol
SDK	Servlet Development Kit
SHA1	Security Hash Algorithm # 1
SMTP	Simple Mail Server Transport Protocol
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
XML	Extensible Markup Language
WYSIWYG	What you see is what you get

TÀI LIỆU THAM KHẢO

[1] Đoàn Văn Ban, *Lập trình hướng đối tượng với Java*, Nhà xuất bản Khoa học và Kỹ thuật, Hà Nội, 2005 (tái bản).

[2] *Core Java, Volume I, II- Advanced Features,* Cay S. Horstmann, Gary Cornell, Sun Microsystems Press, 2000.

[3] Daniel Y. Liang, *Introduction to Java Programming,* Prentice Hall, 2000.

[4] Hiroshi Maruyama, et all., *XML and Java™ Developing Web Application*, Second Eddition, Addison-Wesley, 2002.

[5] Joseph JáJá, *An Introduction to Parallel Algorithms*, Addison-wesley, 1992.

[6] Khalid A.Mughal & Rolf W.Rasmussen, *A Programmer's Guide to Java ™* Certification, Addison-Wesley, 2000.

http://java.sun.com/j2se/1.4/docs/guide/rmi

http://www.securingjava.com

http://achives.java.sun.com

http://www.apache.org

http://www.oreilly.com/catalog/javasec2/

CHỈ MỤC

206192

Giá 68.000