Nguyễn Phương Lan
Hoàng Đức Hải

005.2
J111
E

CÁC KỸ XẢO LẬP TRÌNH VỚI JAVA

JAVA lập trình mạng

TRƯỜNG THKT VÀ NV PHÚ LÂM
THƯ VIỆN
1758
2003
TRƯỜNG CAO ĐẲNG KTKT PHÚ LÂM
THƯ VIỆN
M0378.

NHÀ XUẤT BẢN GIÁO DỤC
NĂM 2001

6T7
GD – 00 425/297-00 PIK13n0

Lời mở đầu

Java là một ngôn ngữ lập trình không đơn giản. Nếu bạn yêu thích một ngôn ngữ lập trình thuần hướng đối tượng và rõ ràng thì Java là một lựa chọn tốt hơn C++. Tuy nhiên ngoài sự nổi tiếng về bản thân ngôn ngữ, nền tảng của Java là hướng đến các ứng dụng mạng. So với lập trình thông thường, lập trình mạng đòi hỏi bạn những hiểu biết và kỹ năng chuyên sâu hơn để tạo giao tiếp và trao đổi dữ liệu giữa các máy tính với nhau. Hầu như ngày nay việc viết một ứng dụng để chạy trên máy đơn cục bộ không còn được ưa chuộng và thích hợp nữa. Các chương trình và ứng dụng hiện đại phải tích hợp và triệu gọi được lẫn nhau trên mạng Intranet (mạng cục bộ), mạng Internet (mạng toàn cầu)...

Để giúp bạn tiến sâu hơn vào thế giới lập trình Java, giáo trình này sẽ giới thiệu và hướng dẫn bạn đầy đủ cách xây dựng và triển khai một ứng dụng mạng Java bằng những công nghệ lập trình hiện đại. Bạn sẽ được làm quen với các khái niệm mới như RMI, CORBA, XML, Servlet, JSP, SSL, Java Mail, EJB từ lý thuyết cho đến cài đặt thực tế... Bạn cần hiểu biết đôi chút về ngôn ngữ Java trước khi đi vào viết các ứng dụng mạng thuần Java mà chúng tôi trình bày trong giáo trình này (Nếu bạn là người mới đến với Java chúng tôi khuyên bạn trước tiên nên đọc "Giáo trình lý thuyết và Bài tập Java" của cùng tủ sách Minh Khai mà chúng tôi đã xuất bản).

Giáo trình này được chia làm 3 phần: phần một (chương 1) bao gồm các kiến thức cơ bản về lập trình mạng. Phần hai (từ chương 2 đến chương 6) là các kỹ thuật lập trình mạng thú vị như lập trình phân tán, đối tượng RMI, CORBA, tìm hiểu Servlet, thiết kế ứng dụng Servlet chạy trên máy chủ Java Web Server từ cơ bản đến nâng cao, xây dựng chương trình Web với JSP (Java Server Page), cơ chế bảo mật SSL. Phần ba (từ chương 7 đến chương 9) giúp bạn xây dựng ứng dụng thư điện tử với Java Mail, tìm hiểu XML, XSL,

khai triển các ứng dụng đa tầng với kiến trúc Enterprise Java Bean (EJB) mới trong Java ...

Tìm hiểu về mạng tốt nhất là bạn cần lập trình trên hệ thống mạng thực sự với nhiều máy kết nối vật lý. Tuy nhiên hầu hết ví dụ và kiến thức trong giáo trình này nếu bạn không có điều kiện chúng tôi vẫn cố gắng trình bày để bạn có thể dễ dàng nghiên cứu và học tập trên một máy đơn duy nhất. Chúng sẽ giúp ích rất nhiều cho bạn khi tiếp xúc với mô hình mạng ở qui mô xí nghiệp hay công ty lớn thực sự.

Trong lần xuất bản đầu tiên này mặc dù đã có nhiều cố gắng để trình bày giáo trình một cách ngắn gọn, dễ hiểu và sát với thực tế, chúng tôi cũng không tránh khỏi một số thiếu sót nhất định. Rất mong sự phê bình và góp ý chân thành của các bạn đọc giả. Mọi thắc mắc và góp ý các bạn có thể gửi về theo địa chỉ e-mail:

mk.pub@cinet.vnnews.com

Cuối cùng để hoàn thành giáo trình này tác giả đã nhận được sự giúp đỡ từ rất nhiều bạn bè, đồng nghiệp và người thân. Xin gửi đến Mẹ, Chị và người Bạn đời yêu quý của tôi vì những yêu thương chân thành và sự hi sinh vô vàn của họ đã dành cho tôi trong những năm tháng qua. Cảm ơn Anh Trần Tiến Dũng, đồng nghiệp đã cung cấp tài liệu và có những góp ý quý báu cho tôi trong quá trình xây dựng giáo trình. Gửi đến bạn Cao Hiệp lời cảm ơn vì sự giúp đỡ quý báu của bạn.

Chúc các bạn một chuyến du lịch thành công vào thế giới lập trình mạng và thưởng thức hương vị thật đậm đà của Java.

MK.PUB
www.minhkhai.com.vn

MỤC LỤC

LỜI MỞ ĐẦU 3
CHƯƠNG 1 : Java ngôn ngữ lập trình mạng 7
1. Liên lạc trên internet 7
2. Trang web 12
3. Dịch vụ từ phía máy chủ và khái niệm cổng (port) 14
4. Giao tiếp trên mạng theo mô hình khách/chủ (client/server) và khái niệm socket . 14
5. Tìm hiểu một số lớp cần thiết của gói thư viện java.net 16
6. Sử dụng TCP socket 25
7. Sử dụng UDP socket 35
8. Xây dựng các chương trình ở máy khách (client) 47
9. Xây dựng các chương trình ở máy chủ (server) 51
10. Kết chương 80
CHƯƠNG 2 : RMI và kỹ thuật lập trình phân tán đối tượng trong java 81
1. Rmi và lập trình phân tán đối tượng 81
2. Gọi phương thức từ xa và các vấn đề phát sinh 83
3. Vai trò các lớp trung gian (stub và skeletion) 83
4. Cài đặt ứng dụng phân tán RMI 85
5. Bộ đăng ký (registry), classpath và codebase 95
6. Chương trình helloworld 106
7. Chuyển tham số trong các lời gọi phương thức từ xa 112
8. Tuần tự hoá (serializable) đối tượng 129
9. Tìm hiểu rmiregistry và các cách đăng ký đối tượng 133
10. Dùng một đối tượng sản sinh nhiều đối tượng (factory object) 138
11. Kỹ thuật gọi đối tượng ở xa bằng phương thức động (dynamic method invoke) .. 146
12. Tự động kích hoạt đối tượng từ xa (activation) 151
13. Trình mồi nạp ứng dụng từ xa 161
14. Ứng dụng RMI 169
15. Rmi registry và các vấn đề về firewall (tường lửa) 196
16. Kết chương 199
CHƯƠNG 3 : CORBA 201
1. Corba và các khái niệm mở đầu 201
2. Ánh xạ giữa ngôn ngữ IDL và java (IDL mapping to java) 206
3. Corba và mô hình phân tán 209
4. Xây dựng ứng dụng corba khách/chủ trong môi trường java 213
5. Tìm hiểu jndi và dịch vụ quản lý tên cos naming của corba 239
6. Lưu địa chỉ tham chiếu của đối tượng corba ở dạng chuỗi (stringified) 255
7. Tham chiếu ngược (callback) của các đối tượng corba 258
8. Gọi các phương thức động (dynamic method call) của đối tượng corba 263
9. Rmi trên IIOP (Rmi-Over-IIOP) 270
10. Corba và trình môi giới orb visibroker 279
11. Kết chương 290
CHƯƠNG 4 : Java servlet và lập trình web 291
1. Giới thiệu các khái niệm lập trình web 291
2. Cài đặt java web server 300

3. Cấu hình java web server 303
4. Xây dựng servlet đầu tay 312
5. Servlet liên hoàn (chaining servlet) 323
6. Include Servlet 338
4. Session - Khái niệm và cách kiểm soát 346
8. Tìm hiểu cookies 357
9. Sử dụng JDBC truy xuất cơ sở dữ liệu với servlet 364
10. Kết chương 375
CHƯƠNG 5 : Servlet và các tính năng nâng cao 377
1. Kỹ thuật triệu gọi servlet không cần trình duyệt 377
2. Kỹ thuật xuyên hầm (tunneling) với giao thức HTTP 385
3. Giao tiếp giữa applet và servlet 401
4. Tương tác khách/chủ và vấn đề bảo mật (web security) 403
5. Kết chương 435
CHƯƠNG 6 : JSP (java server page) 437
1. JSP thay cho servlet 437
2. Tìm hiểu các thẻ lệnh JSP 448
3. Các đối tượng mặc định của trang JSP 457
4. Truy xuất cơ sở dữ liệu trong trang JSP 458
5. Trang JSP triệu gọi đối tượng RMI, CORBA 462
6. Kết chương 470
CHƯƠNG 7 : java mail 471
1. Gửi mail với smtpclient 471
2. Gửi/Nhận mail bằng java mail API 480
3. Kết chương 524
CHƯƠNG 8 : XML 525
1. Khái niệm cơ bản về XML 525
2. Cú pháp XML 528
3. Xem dữ liệu XML bằng trình duyệt IE 530
4. XML và cơ sở dữ liệu 532
5. XML thay thế cho các tập tin cấu hình 533
6. Đọc và xử lý dữ liệu XML 536
7. HTML, XML và XSL 541
8. Kết chương 544
CHƯƠNG 9 : ENTERPRISE JAVA BEAN (EJB) 545
1. Mô hình lập trình đa tầng (multi-tier) 545
2. Tìm hiểu đối tượng EJB 553
3. Xây dựng đối tượng bean EJB 556
4. Tìm hiểu cách xây dựng EJB của trình đóng gói 573
5. Thiết kết bean thao tác lưu vết trạng thái (statefull session bean) 579
6. So sánh thành phần EJB phi trạng thái và EJB lưu vết trạng thái 589
7. Xây dựng bean thực thể tự quản lý (entity bean-managed) 590
8. Xây dựng bean thực thể do trình chứa quản lý (entity bean container-mainaged) 615
9. EJB và cách trình chứa quản lý bean thông qua XML 630
10. Enterprise java bean và sự hỗ trợ của các trình chủ khác 636
11. Kết chương 637

Chương 1

JAVA NGÔN NGỮ LẬP TRÌNH MẠNG

(Network Programming)

Tóm tắt chương 1:

Chương mở đầu này chúng ta sẽ làm quen và tìm hiểu các khái niệm về lập trình mạng bằng ngôn ngữ Java. Java cung cấp các lớp Socket, ServerSocket, DatagramSocket ... đủ mạnh để thiết kế bất kỳ ứng dụng nào theo mô hình khách/chủ (client/server). Kết thúc chương này ta sẽ tự xây dựng được những ứng dụng cơ bản và thường sử dụng nhất hiện nay như trao đổi số liệu, truy cập các máy chủ ở xa, cách đọc và xử lý các trang Web trên mạng Internet ...

Những vấn đề chính sẽ được đề cập trong chương này:

- ✓ *Liên lạc trên Internet và giao tiếp trên mạng.*
- ✓ *Giới thiệu về mô hình ứng dụng khách chủ.*
- ✓ *Tìm hiểu gói thư viện hỗ trợ lập trình mạng 0java.net*
- ✓ *Sử dụng các giao thức TCP và UDP*
- ✓ *Xây dựng các ứng dụng chạy trên máy khách (client application)*
- ✓ *Xây dựng các ứng dụng chạy trên máy chủ (servert application)*

1. LIÊN LẠC TRÊN INTERNET

1.1. Họ giao thức TCP/IP

Để hai hay nhiều máy tính nói chuyện được với nhau, chúng phải dùng một ngôn ngữ chung: chẳng hạn máy này phải gửi những tín hiệu gì đến máy kia và máy kia phải gửi trả lại những tín hiệu nào để nhận biết. Trên Internet ngày nay việc hai máy có thể trao đổi được với nhau đa số đều dựa theo quy ước hay giao thức cốt lõi là TCP/IP (Transmision Control Protocol/Internet Protocol). Theo giao

thức này mỗi máy sẽ được đặt cho một số riêng biệt gọi là địa chỉ IP (IP address) có vai trò tương tự số điện thoại, chẳng hạn máy tính được đặt tên là *www.microsoft.com* sẽ có địa chỉ IP là *207.46.230.219*. Các số IP này là duy nhất và không máy nào được trùng nhau (trên toàn thế giới). Khi bạn muốn máy của mình có địa chỉ IP để tham gia vào hệ thống Internet toàn cầu như là một máy chủ(host hay server) ta phải đăng ký với tổ chức quốc tế InterNIC (Internet Network Information Center) để nhận được một số IP riêng biệt.

Vậy tại sao vừa có địa chỉ IP lại vừa có tên riêng cho từng máy? Dùng cái nào để xác định liên lạc với một máy chủ (như ở trên, ta nên dùng *www.microsoft.com* hay số *207.46.230.219* để kết nối với máy chủ của công ty Microsoft?). Thật ra tên và địa chỉ IP là một, nhưng địa chỉ IP được ghi bằng số, còn tên cho máy chủ lại được biểu diễn bằng chữ có ý nghĩa và gần gũi hơn với con người. Với mỗi hệ thống đều có sự chuyển đổi trực tiếp từ tên vùng thành địa chỉ IP thích hợp trước khi dữ liệu được gửi đi.

Ví dụ đối với Windows có thể tham khảo hai tập tin HOSTS và LMHOSTS đây là hai tập tin văn bản (được coi như một cơ sở dữ liệu) để lưu trữ tập hợp các số IP cùng với tên tương ứng. Ta có thể tự thêm vào địa chỉ IP và tên máy chủ cách nhau bằng khoảng trắng. Khi có nhu cầu truy cập đến một máy ở xa qua giao thức mạng TCP/IP nếu gõ vào tên máy chủ thì hệ thống sẽ tự động tìm địa chỉ IP tương ứng trong tập tin này:

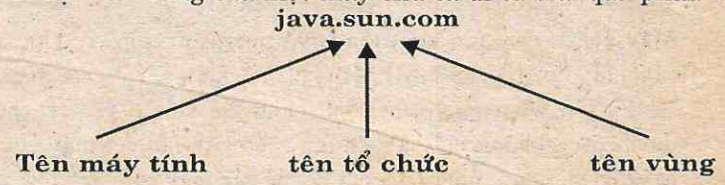
Tập tin HOSTS

```
216.32.74.52      www.myyahoo.com
164.71.2.70       www.fujitsu.com
```

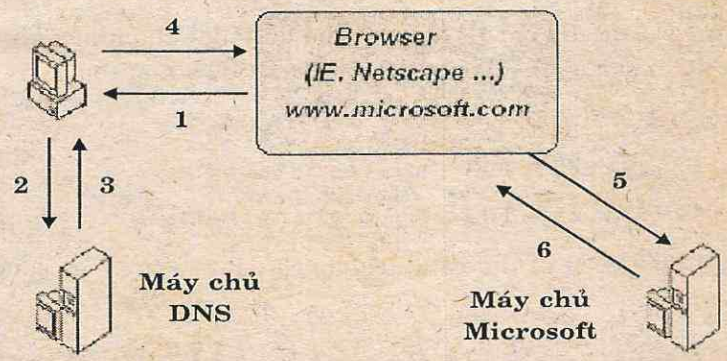
Tên của máy chủ còn được gọi là tên vùng (domain name) bởi vì chúng được đặt theo thứ tự phân cấp của tên lãnh thổ, vùng, tổ chức, hay tên cá nhân ... Mỗi nhóm phân cấp cách nhau bằng một dấu chấm (.). Công việc theo dõi sự thay đổi tên được phân phối qua

Internet nhờ các máy chủ lớn DSN (Domain Name System) theo dõi các máy chủ khác trong vùng con của chúng. Trước đây mỗi máy có thể tự mình lưu trữ một tập tin chứa phần lớn các tên và địa chỉ của những máy chủ thông dụng (như trong Windows là tập tin HOSTS hay LMHOSTS), nhưng ngày nay ta không làm vậy nữa mà đa số các tên vùng cũng như địa chỉ IP được lấy xuống từ các máy chủ DNS.

Khi đọc tên vùng của một máy chủ ta đi từ trái qua phải. Ví dụ:



Nói chung là theo quy ước từ phần riêng biệt nhất đến phần chung nhất. Tuy nhiên không bắt buộc là như vậy, ta vẫn có thể đặt tên theo cách khác không nhất thiết chỉ gồm 3 hay 4 nhóm (ví dụ : `here.is.along.name.address.co.vn` là hợp lệ) bởi vì cuối cùng thì tên vùng cũng được hệ thống DNS chuyển thành địa chỉ IP mà thôi.



Hình 1-1. Cơ chế truy tìm địa chỉ IP thông qua dịch vụ DNS

Hình 1-1 là cơ chế truy tìm địa chỉ IP của máy tính khi kết nối với máy chủ. Giả sử bạn muốn trình duyệt (browser) như Internet Explorer hay Netscape truy tìm một tập tin hay trang Web của máy

chủ Microsoft. Hãng Microsoft cung cấp cho bạn địa chỉ của máy chủ là www.microsoft.com. Đây chỉ là một chuỗi tên bình thường, muốn kết nối được với máy chủ trước tiên trình duyệt phải chuyển đổi được tên www.microsoft.com thành địa chỉ IP tương ứng. Quá trình truy tìm địa chỉ IP diễn ra như sau:

1. Trình duyệt yêu cầu hệ điều hành trên máy khách chuyển chuỗi www.microsoft.com thành địa chỉ IP.
2. Máy khách truy tìm xem tên www.microsoft.com có được ánh xạ trong tập tin HOSTS hay LMHOST hay không. Nếu có, máy khách sẽ đổi tên www.microsoft.com thành địa chỉ IP gửi về cho trình duyệt. Nếu không máy khách tìm cách liên lạc với máy chủ DNS. Máy DNS lưu rất nhiều địa chỉ IP của các máy chủ theo tên.
3. Nếu tìm thấy địa chỉ IP của tên www.microsoft.com máy DNS sẽ gửi địa chỉ IP về cho máy khách.
4. Máy khách chuyển địa chỉ IP cho trình duyệt.
5. Trình duyệt sử dụng địa chỉ IP liên lạc với máy chủ của Microsoft.
6. Quá trình kết nối thành công. Máy chủ Microsoft sẽ trả thông tin yêu cầu về cho máy khách.

Lưu ý: Một địa chỉ IP cũng có thể được đặt cho nhiều tên khác nhau. Ví dụ bạn có thể đặt trong tập tin HOSTS như sau:

HOSTS

```
129.74.250.103   my.favorit.home
129.74.250.103   my.company.net
```

Một máy cũng có thể có nhiều địa chỉ IP nếu có nhiều card giao tiếp mạng.

1.2. Cách chuyển dữ liệu trên mạng, giao thức TCP và UDP

Quá trình chuyển dữ liệu trên mạng diễn ra khá phức tạp. Chi tiết quá trình này diễn ra tương tự như trong thực tế ta gửi thư hay bưu phẩm, trước hết phải ghi rõ địa chỉ nơi đến (trường hợp này là

địa chỉ IP của máy chủ), sau đó có thể gửi thông thường hay gửi bảo đảm (tùy theo cách gửi mà thư hay bưu phẩm có chắc chắn đến được tay người nhận hay không), người nhận sau khi nhận được có thể hồi âm trả lời là đã nhận đủ hoặc bị mất mát gì đó trong quá trình chuyển tải. Người gửi có thể gửi tiếp những phần bị mất (hoặc không cần gửi nữa).

Cách chuyển dữ liệu bảo đảm dựa vào giao thức TCP (Transmission Control Protocol) còn cách chuyển không bảo đảm dựa vào giao thức UDP (User Datagram Protocol).

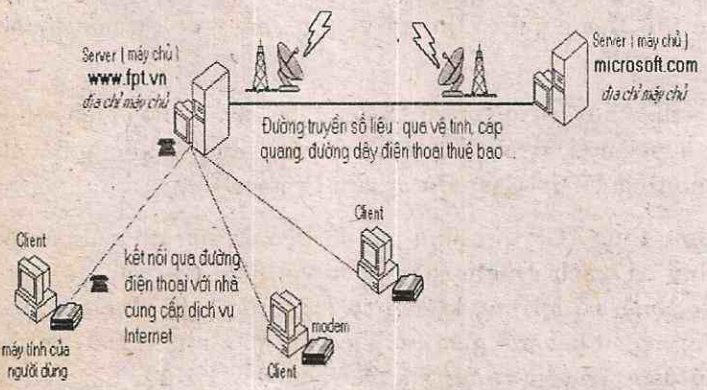
Giao thức TCP gửi từng gói dữ liệu đi, nơi nhận theo giao thức này phải có trách nhiệm thông báo và kiểm tra xem dữ liệu đã đến đủ hay chưa, có lỗi hay không có lỗi. Trước khi chuyển dữ liệu bao giờ cũng có sự kết nối giữa máy gửi và máy nhận. Do phải bảo đảm dữ liệu được truyền chính xác và luôn duy trì kết nối nên sử dụng giao thức TCP cần chiếm thêm một số tài nguyên của hệ thống và cách lập trình cho giao thức này hơi khó (phải thực hiện các bước kiểm tra dữ liệu theo yêu cầu của TCP). Truyền dữ liệu theo TCP thường áp dụng cho các dịch vụ như truyền tập tin, các dịch vụ trực tuyến trên Internet đòi hỏi độ tin cậy cao...

Giao thức UDP ngược lại không đáng tin cậy lắm, không có sự kết nối trước nào giữa nơi gửi và nơi nhận, dữ liệu gửi đi mặc định rằng máy tính ở đầu nhận luôn ở trạng thái sẵn sàng để tiếp đón dữ liệu gửi đến. Nếu dữ liệu gửi đến bị lỗi trong quá trình truyền hay không nhận được đầy đủ giao thức UDP cũng không có thông tin phản hồi gì lại cho nơi gửi. Tuy nhiên UDP không đòi hỏi nhiều tài nguyên của hệ thống và cách lập trình lại tỏ ra đơn giản. Truyền dữ liệu theo giao thức UDP thường được dùng trong những ứng dụng không đòi hỏi sự chính xác cao như dịch vụ thông báo giờ, tỉ giá hay các dịch vụ gửi nhắn tin ...

1.3. Kết nối với Internet

Ta có thể kết nối vào Internet thông qua dịch vụ của nhà cung cấp còn gọi là ISP (như FPT, Cnet ...) bằng đường điện thoại thông

qua modem. Các nhà dịch vụ này đóng vai trò như những máy chủ (server) giúp dễ dàng truy cập dữ liệu từ những vùng khác nhau trên mạng.



2. TRANG WEB

Một khi đã tham gia vào Internet, thông tin từ Internet có thể được lấy về bằng nhiều hình thức khác nhau , chỉ cần cho biết địa chỉ của máy chủ (nơi mà ta muốn truy cập thông tin) và loại yêu cầu mà máy chủ cần phải thực hiện, nếu thỏa mãn máy chủ sẽ cung cấp ngay những thông tin cần thiết.

Chẳng hạn nếu muốn xem một tài liệu về tin học và tài liệu này được lưu tại máy tính có tên (hay địa chỉ) www.itknowledge.com trong tập tin `doc.htm`, ta sẽ yêu cầu một chương trình được gọi là "chương trình phục vụ đọc thông tin trên mạng" của mình như sau: http://www.itknowledge.com/doc.htm. Chương trình này được biết đến như là trình duyệt (browser), điển hình có Internet Explorer của Microsoft, Netscape Navigator của công ty Netscape Communication, HotJava của Sun Microsystem ... Nó có nhiệm vụ phân tích, gửi đi những yêu cầu từ phía người dùng đến máy chủ sau đó nhận lại, trình bày kết quả trả về từ phía máy chủ cho người dùng.

Một chuỗi yêu cầu như trên được gọi là chuỗi định vị tài nguyên URL (Uniform Resource Locator).

Khi ta đánh vào

http://www.itknowledge.com/doc.htm

Chuỗi URL này sẽ được trình duyệt tách làm ba phần:

- Phần yêu cầu (hay còn gọi là phần nghi thức) : ***http*** là loại yêu cầu, cụ thể ở đây ta muốn máy chủ gửi về một tài liệu được định dạng theo kiểu HTML (hypertext markup laguage) còn được gọi là trang Web hay Web page (HTML là một tập tin văn bản bình thường chỉ cần bỏ một chút công sức là ta hoàn toàn có thể tạo ra được nó).
- Phần địa chỉ : ***www.itknowledge.com*** là địa chỉ của máy chủ.
- Phần bổ sung : ***doc.htm*** là tên của tập tin tài liệu mà ta muốn lấy về xem (còn được gọi là trang Web).

Nếu dịch vụ đòi hỏi phải chỉ định số cổng (port) kết nối thì chuỗi URL có thể được định vị thêm một thành phần mang giá trị port. Ví dụ:

http://www.itknowledge.com:80/doc.htm

80 là giá trị cổng dành cho dịch vụ `http`. Dưới đây ta sẽ tìm hiểu cách dùng cổng (port) cho nhiều dịch vụ khác nhau.

3. DỊCH VỤ TỪ PHÍA MÁY CHỦ VÀ KHÁI NIỆM CỔNG (PORT)

Khi kết nối vào máy chủ ta có thể yêu cầu máy chủ nhiều dịch vụ khác nhau, như dịch vụ truy tìm và đọc các trang Web trên Internet, dịch vụ gửi nhận e-mail, dịch vụ dò tìm hệ thống tên vùng DNS ... Mỗi dịch vụ đều có cách gửi nhận dữ liệu theo quy ước riêng. TCP và UDP chỉ chịu trách nhiệm đưa dữ liệu từ một máy tính này đến một máy tính khác, còn dữ liệu đó sẽ được gửi cho dịch vụ nào thì phải thông qua một quy định nữa là cổng (hay port). Mỗi chương trình dịch vụ sẽ sử dụng một cổng khác nhau để truy xuất thông tin. Cổng là một số nguyên dương có giá trị từ 1 đến 16383

Máy chủ (server) sẽ qui định cổng được sử dụng cho mỗi loại dịch vụ. Thông tin giữa máy khách (client) và máy chủ (server) phải sử dụng cổng tương ứng nhau thì mới trao đổi với nhau được. Tuy nhiên, hầu hết các chương trình dịch vụ nổi tiếng hiện nay đều có qui định chuẩn cổng dành riêng cho mình như:

Dịch vụ	*cổng (port)*
FTP	21
HTTP	80
Telnet	23
Finger	79
SMTP	25

Nếu tự xây dựng một ứng dụng làm dịch vụ trên máy chủ ta phải chọn cho mình một số cổng có các giá trị khác với những giá trị cổng mà những dịch vụ nổi tiếng đã sử dụng.

Khi nghiên cứu về RMI và CORBA trong chương sau chúng ta sẽ gặp lại khái niệm cổng này nhiều hơn.

4. GIAO TIẾP TRÊN MẠNG THEO MÔ HÌNH KHÁCH/CHỦ (CLIENT/SERVER) VÀ KHÁI NIỆM SOCKET

4.1. Giao tiếp theo mô hình khách chủ (Client/Server)

Có rất nhiều dịch vụ hỗ trợ trên Internet như e-mail, nhóm tin (newsgroup), chuyển tập tin (file transfer), đăng nhập từ xa (remote login), truy tìm các trang Web ... Những dịch vụ này được tổ chức và kiến trúc theo mô hình khách/chủ (mô hình client/server). Các chương trình ở máy khách (client) như trình duyệt (web browser) hay chương trình gửi nhận e-mail sẽ tạo ra kết nối (connection) với một máy chủ ở xa (server) sau đó gửi các yêu cầu đến máy chủ, các chương trình dịch vụ trên máy chủ như Web server hay Mail server ... sẽ xử lý những yêu cầu này và gửi kết quả ngược về cho máy khách (chẳng hạn Web server sẽ truy tìm và trả về cho máy khách các trang Web theo địa chỉ mà máy khách đưa đến còn Mail server thì lưu giữ và gửi về cho máy khách những bức e-mail mới). Thông thường một dịch vụ trên máy chủ phục vụ cho rất nhiều máy khách.

4.2. Lập trình mạng thông qua Socket

Như vậy trước khi yêu cầu một dịch vụ trên máy chủ thực hiện điều gì đó, máy khách (client) phải có khả năng kết nối được với máy chủ. Quá trình kết nối này được Java thực hiện thông qua một cơ chế trừu tượng hóa gọi là Socket (tạm dịch là "cơ chế ổ cắm"). Kết nối giữa máy khách và máy chủ tương tự như việc cắm phích điện vào ổ cắm điện. Máy khách thường được coi như phích cắm điện còn máy chủ được coi như ổ cắm điện, một ổ cắm có thể cắm vào đó nhiều phích điện khác nhau cũng như một máy chủ có thể kết nối và phục vụ cho rất nhiều máy khách.

Nếu kết nối socket thành công thì máy khách và máy chủ có thể trao đổi dữ liệu với nhau thực hiện các yêu cầu về dịch vụ trên máy chủ. Việc kết nối theo cơ chế socket cần biết hai thông tin chủ yếu đó là địa chỉ của máy cần kết nối và số hiệu cổng của chương trình dịch vụ. Java cung cấp lớp `Socket` (thường được dùng như "phích cắm điện" cho máy khách) và lớp `ServerSocket` (thường được dùng như "ổ cắm điện" đặt trên máy chủ). Hai lớp này được đặt trong gói thư viện `Java.net`.

Lưu ý: Lớp `Socket` có thể được dùng kết nối chung cho cả máy khách và máy chủ nhưng ta vẫn thường dùng lớp `ServerSocket` để thực hiện việc quản lý kết nối đặt riêng ở máy chủ hơn.

Đối với cách truyền dữ liệu không yêu cầu kết nối trước như giao thức UDP thì cơ chế kết nối socket vẫn được thực hiện nhưng thông qua lớp `DatagramSocket`.

5. TÌM HIỂU MỘT SỐ LỚP CẦN THIẾT CỦA GÓI THƯ VIỆN JAVA.NET

5.1. Lớp InetAddress

Vì địa chỉ Internet theo số IP và theo tên rất thường dùng khi kết nối vào mạng cho nên Java xây dựng hẳn một lớp `InetAddress` dành riêng cho việc quản lý địa chỉ theo tên và theo số. Lớp `InetAddress` cung cấp các phương thức `static` thông dụng nhất dùng để chuyển đổi và truy xuất địa chỉ IP (không có phương thức khởi dựng cho lớp này). Thường ta sẽ quan tâm đến các phương thức sau:

- `public static InetAddress getLocalHost()`
  `throws UnknownHostExceptiongetByName`

Trả về đối tượng InetAddress là địa chỉ của máy cục bộ (local host).

- `public static InetAddress getByName(String host)`
  `throws UnknownHostException`

Phương thức này nhận địa chỉ của một máy bằng kiểu chuỗi String và trả về đối tượng kiểu InetAddress thay mặt cho địa chỉ máy này.

- `public static InetAddress[] getAllByName(String host)`
  `throws UnknownHostException`

Phương thức này nhận địa chỉ của một máy bằng kiểu chuỗi và trả về tất cả các đối tượng InetAddress thay mặt cho địa chỉ máy này.

- `public byte[] getAddress()`

Trả về địa chỉ IP của đối tượng `InetAddress` dưới dạng một dãy các byte. Vị trí byte cao nhất nằm ở byte 0.

- `public String getHostAddress()`

Trả về địa chỉ IP của đối tượng InetAddress dưới dạng một chuỗi được định dạng phân làm 4 nhóm %d.%d.%d.%d (ví dụ "172.16.11.12").

Dưới đây là ví dụ cho thấy cách dùng lớp `InetAddress` để lấy về các thông tin của địa chỉ máy chủ:

Ví dụ 1-0 AddrLookupApp.java

```
import java.net.*;
public class AddrLookupApp {
public static void main(String args[]) {
 try {
  if(args.length!=1){
   System.out.println("Usage: java AddrLookupApp
                        <HostName>");
   return;
  }
// Lấy về đối tượng InetAddress chứa thông tin địa chỉ máy chủ
  InetAddress host = InetAddress.getByName(args[0]);
// Lấy về tên máy chủ nếu có
  String hostName = host.getHostName();
// In các thông tin ra màn hình
  System.out.println("Host name: "+hostName);
  System.out.println("IP address:
                "+host.getHostAddress());
 }catch(UnknownHostException e) {
```

M0378
TRƯỜNG THKT VÀ NV PHÚ LÂM
THƯ VIỆN

```
        System.out.println("Address not found");
        return;
    }
  }
}
```

Biên dịch

```
C:\Network>javac AddrLookupApp.java
```

Chạy chương trình

- Nếu kết nối trực tiếp với Internet ta có thể chạy chương trình này để lấy về địa chỉ của những máy chủ nổi tiếng như:

```
C:\ Network >java AddrLookupApp java.sun.com
                Host name : java.sun.com
                IP address: 192.9.9.100
```

- Nếu không kết nối được trực tiếp với Internet ta có thể dùng ngay máy cục bộ để thử nghiệm. Số IP 127.0.0.1 được các hệ thống dùng mặc định cho máy cục bộ. Ta có thể dùng địa chỉ này để đặt tên cho máy của mình trong tập tin HOSTS như sau (tập tin này nằm trong thư mục Windows):

```
127.0.0.1 my.testing.server
```

Chương trình AddrLookupApp khi chạy sẽ cho kết quả là:

```
C:\Network>java AddrLookupApp my.testing.server
Host name : my.testing.server
IP address: 127.0.0.1
```

hoặc:

```
C:\Network>java AddrLookupApp 127.0.0.1
Host name : my.testing.server
IP address: 127.0.0.1
```

Lưu ý: Việc kết nối với Internet không phải ai cũng có đủ điều kiện (nhất là đối với Việt nam ta thì chi phí này khá cao), mặc dù vậy các chương trình và ví dụ cho chương này hoàn toàn có thể chạy trên một máy mà không cần phải truy cập vào hệ thống mạng thật sự. Để mô phỏng cho cách làm việc trên mạng thường ta sẽ mở hai cửa sổ dòng lệnh, một cửa sổ được dùng làm máy chủ và một cửa sổ khác dùng làm máy khách. Hãy dùng số 127.0.0.1 để làm địa chỉ IP cho máy chủ và mở tập tin HOSTS trong thư mục WINDOWS đặt tên cho máy chủ là `my.testing.server` (tên này có thể tùy ý đặt) như sau:

```
127.0.0.1  my.testing.server
```

5.2. Lớp Socket

Lớp `Socket` dùng tạo kết nối từ phía máy khách với máy chủ thường được khởi dựng bằng các phương thức sau:

- `public Socket(String host,int port)`
  `throws UnknownHostException, IOException`

Tạo ra một socket kết nối máy có tên theo địa chỉ `host` và số cổng `port`.

- `public Socket(InetAddress address,int port)`
  `throws IOException`

Tạo ra một socket kết nối từ địa chỉ là đối tượng `InetAddress` và số cổng `port`.

- `public Socket(String host,int port, boolean stream)`
  `throws IOException`

Tạo ra một socket kết nối theo địa chỉ `host` và số cổng `port`, tham số `stream` cuối cùng để quy định kết nối theo TCP (`stream=true`) hay UDP (`stream =false`). Tuy nhiên nếu áp dụng để tạo socket cho giao thức UDP nên sử dụng lớp thay thế là `DatagramSocket`.

Các phương thức khác hỗ trợ cho lớp `Socket` từ phía máy khách bao gồm:

- `InputStream getInputStream() throws IOException`

  Lấy về luồng nhập để máy khách có thể đọc dữ liệu trả về từ phía máy chủ.

- `OutputStream getOutputStream() throws IOException`

  Lấy về luồng xuất để máy khách có thể ghi dữ liệu gửi đến máy chủ.

- `InetAddress getInetAddress()`

  Lấy địa chỉ kết nối socket của máy chủ

- `int getPort()`

  Lấy về số cổng dùng kết nối của máy chủ.

- `synchronized void close() throws IOException`

  Cắt đứt kết nối với máy chủ.

Ví dụ đoạn mã sau sẽ thực hiện kết nối với máy chủ có địa chỉ `"my.testing.server"` và mở ra hai luồng xuất nhập để đọc và gửi thông tin đến máy chủ có theo số cổng 1234:

```
    try{
   Socket me=new Socket("my.testing.server",
   1234);
//Luồng nhập để đọc thông tin trả về từ máy chủ kết nối
   DataInputStream in =
        new DataInputStream(me.getInputStream());
//Luồng xuất để ghi thông tin gửi đến máy chủ
   DataOutputStream out=
        new
   DataOutputStream(me.getOutputStream());}
catch (Exception e){
System.out.println(e);
}
```

5.3. Lớp ServerSocket

Lớp ServerSocket dùng tạo kết nối từ phía máy chủ với các máy khách. Đối tượng ServerSocket được tạo ra trên máy chủ và lắng nghe những kết nối từ phía máy khách gửi đến theo một số cổng định trước. Đối tượng ServerSocket được khởi dựng từ phương thức sau:

- `public ServerSocket(int port) throws IOException`

port là số hiệu cổng mà đối tượng ServerSocket phải lắng nghe để nhận biết những kết nối từ phía máy khách gửi đến.

Để chờ đợi kết nối từ các máy khách gửi đến đối tượng ServerSocket thường nhờ đến phương thức accept như sau:

- `Socket accept() throws IOException`

Phương thức này thực sự dừng lại chờ đợi cho đến khi nhận được thông tin kết nối sẽ trả về đối tượng socket của máy khách nơi có yêu cầu nối vào máy chủ.

Cuối cùng máy chủ có thể cắt đứt mọi kết nối bằng cách gọi phương thức close của đối tượng ServerSocket:

- `public void close() throws IOException`

Ví dụ đoạn mã sau sẽ tạo một đối tượng ServerSocket trên máy chủ luôn lắng nghe kết nối từ máy khách gửi đến qua số cổng 1234

```
try{
   ServerSocket server=new ServerSocket(1234);
   Socket client;
//Chương trình server sẽ dừng lại ở đây để chờ đợi sự kết nối
   client=server.accept()
//Có một kết nối gửi đến từ phía máy khách
   System.out.println("Accept connect");
//Xử lý các yêu cầu về dịch vụ
```

```
//...
//Cắt đứt các kết nối
        client.close();
        server.close();
    catch (Exception e){
    System.out.println(e);
        }
```

5.4. Lớp DatagramSocket

Lớp này được dùng để chuyển đi một gói dữ liệu (biểu diễn bằng đối tượng `DatagramPackage`) theo giao thức UDP. Dữ liệu được gửi đi không bảo đảm được nhận đầy đủ và có thể bị lỗi trên đường truyền (cơ chế dùng lớp `DatagramSocket` không an toàn bằng lớp Socket). Dưới đây là một số phương thức thường dùng của lớp `DatagramSocket`:

- `public DatagramSocket() throws SocketException`

  Phương thức khởi dựng để tạo kết nối UDP.

- `public DatagramSocket(int port) throws SocketException`

  Phương thức khởi dựng để tạo kết nối UDP với số hiệu cổng `port`.

- `public void synchronized send(DatagramPackage p) throws IOException`

  Gửi gói dữ liệu đi.

- `public void synchronized receive(DatagramPackage p) throws IOException`

  Nhận gói dữ liệu về.

- `public void synchronized close()`

  Đóng kết nối.

5.5. Lớp DatagramPackage

Lớp này dùng cho một gói chứa dữ liệu gửi đi trên mạng theo kết nối `DatagramSocket`. Một gói có thể chứa các thông tin như dữ liệu, chiều dài gói, các địa chỉ IP và số cổng mà từ đó gói dữ liệu được gửi đi. Dưới đây là một số phương thức hữu dụng của lớp `DatagramPackage` này:

- `public DatagramePackage(byte buf[], int len)`

  Phương thức khởi dựng tạo ra gói có dữ liệu chứa trong bộ đệm `buff[]` và chiều dài gói dữ liệu là `len`.

- `public DatagramePackage(byte buf[], int len, InetAddress iaddr, int port)`

  Phương thức khởi dựng tạo ra gói có dữ liệu chứa trong bộ đệm `buff[]` cùng với chiều dài vùng đệm muốn lấy, địa chỉ máy đích và số hiệu cổng

- `public InetAddress getAddress()`

  Trả về địa chỉ chứa trong gói dữ liệu

- `public byte[] getData()`

  Trả về dữ liệu thật sự chứa trong gói.

- `public int getLength()`

  Trả về kích thước hay chiều dài gói dữ liệu

- `public int getPort()`

  Trả về số hiệu cổng chứa trong gói dữ liệu.

5.6. Lớp URL

URL (Uniform Resource Locator) là địa chỉ định vị tài nguyên trên mạng, thường một URL như đã đề cập bao gồm 3 phần : phần nghi thức (protocol), phần địa chỉ hay tên máy chủ (host name) và phần chỉ định tên tập tin hay tài liệu muốn lấy từ máy chủ về.

Java đóng gói tất cả những đặc điểm này vào một lớp URL. Đối tượng URL được tạo ra bằng một trong những phương thức khởi dựng sau:

- `public URL(String spec) throws MalformedURLException`

  Tạo một đối tượng URL từ địa chỉ định vị là một chuỗi.

- `public URL(String protocol, String host, int port, String file) throws MalformedURLException`

Tạo một địa chỉ định vị tuyệt đối với đầy đủ các thành phần: nghi thức (protocol), máy chủ (server), cổng (port) và đường dẫn (file) đến tập tin trên máy chủ.

- `public URL(String protocol, String host, String file) throws MalformedURLException`

Tạo một địa chỉ định vị tuyệt đối với đầy đủ các thành phần: nghi thức (protocol), máy chủ (server), đường dẫn (file) đến tập tin trên máy chủ (bỏ qua thành phần định vị cổng giao tiếp, nếu truy tìm trang Web theo nghi thức http thì cổng mặc định của chương trình Web server là 80).

Các phương thức hỗ trợ khác dùng cho lớp URL là:

- `public final Object getContent() throws IOException`

  Lấy về nội dung mà kết nối theo địa chỉ URL có được.

- `String getFile()`

Lấy về thành phần tên tập tin hay tài liệu nằm trong chuỗi địa chỉ URL.

- `String getHost()`

  Lấy tên máy chủ (thường là thành phần thứ hai của chuỗi URL)

- `String getProtocol()`

  Lấy về tên giao thức (thành phần đầu tiên của chuỗi URL)

- `String getPort()`

  Lấy về số hiệu cổng

- `String getRef()`

Lấy về nội dung tham khảo thêm trong chuỗi URL (được đặt sau dấu # của chuỗi)

- `public final InputStream openStream()`
  `throws IOException`

Mở một luồng nhập để đọc thông tin trả về từ máy chủ.

Ví dụ đoạn mã sau dùng để lấy về nội dung trang Web `index.htm` từ máy chủ có địa chỉ là `java.sun.com`:

```
    try{
//Mở kết nối đến trang Web theo địa chỉ định vị URL
    URL o=new URL("http://java.sun.com/index.htm");
//Tạo luồng nhập để đọc nội dung trang Web trả về từ máy chủ
    BufferedReader inStream = new BufferedReader(
            new InputStreamReader(o.openStream()));
//In nội dung trang Web index.htm ra màn hình
    String line;
    while((line = in.readLine()) != null) {
            System.out.println(line);
      }
    }
  catch (Exception e){
//Quá trình mở và kết nối với trang Web bị lỗi
      System.out.println(e);
  }
```

6. SỬ DỤNG TCP SOCKET

Kết nối và gửi dữ liệu theo giao thức TCP được dùng rất nhiều trong các ứng dụng mạng do tính an toàn và bảo đảm của nó.

Dưới đây ta sẽ xây dựng một mô hình ứng dụng khách/chủ (client/server) sử dụng lớp `Socket` và `ServerSocket` để kết nối và trao đổi thông tin giữ máy khách và máy chủ. Máy khách (client)

sẽ gửi một chuỗi đến máy chủ (server), máy chủ sẽ đảo ngược thứ tự của chuỗi và gửi trả về cho máy khách.

Để mô hình này hoạt động ta phải xây dựng hai chương trình: một chương trình là `EchoServer` chạy trên máy chủ dùng để lắng nghe kết nối từ phía máy khách và xử lý yêu cầu do máy khách đưa đến, một chương trình khác là `EchoClient` chạy trên máy khách có nhiệm vụ nhận dữ liệu nhập vào từ bàn phím sau đó gửi đến cho máy chủ xử lý. Khi máy chủ xử lý xong chương trình `EchoClient` sẽ nhận về và đưa kết quả ra màn hình.

6.1. Xây dựng chương trình EchoServer

`EchoServer` chạy trên máy chủ và đón nhận dữ liệu do máy khách gửi đến cổng 3456. Khi nhận được dữ liệu máy chủ sẽ tạo ra đối tượng `EchoString` để thực hiện việc đảo ngược chuỗi sau đó trả dữ liệu về cho máy khách.

Ví dụ 1-1 Server.java

```
import java.net.*;
import java.io.*;
public class EchoServer {
public static void main(String args[]){
 try{
 /*
Tạo đối tượng ServerSocket dùng để lắng nghe kết nối từ các máy
khách gửi đến cổng 345
*/
  ServerSocket server = new ServerSocket(3456);
  int localPort = server.getLocalPort();
  System.out.println("Echo Server is listening on
port
```

```
                    "+localPort+".");
//Chờ đợi kết nối từ máy khách.
/*
    Lưu ý : đến đây chương trình sẽ dừng lại và chờ cho đến khi có
kết nối xảy ra
*/
  Socket client = server.accept();
/*
    Có kết nối xảy ra. Lấy các thông tin từ máy khách và cho in ra
màn hình
*/
  String destName =
          client.getInetAddress().getHostName();
  int destPort = client.getPort();
  System.out.println("Accepted connection to
"+destName
                  +" on port "+destPort+".");
//Lấy về luồng nhập để đọc dữ liệu từ máy khách gửi đến.
/*
    Lưu ý : Ta đưa luồng nhập vào bộ lọc luồng để chuyển đổi
thành luồng
BufferReader có khả năng đọc được một chuỗi  ký tự bằng phương
thức readLine()
*/
  BufferedReader inStream = new BufferedReader(
   new InputStreamReader(client.getInputStream()));
  //Lấy về luồng xuất để ghi dữ liệu gửi đến máy khách
  DataOutputStream outStream = new
```

```
        DataOutputStream(client.getOutputStream());
//Chờ đọc và xử lý dữ liệu do máy khách gửi đến
 boolean finished = false;
 do {
      //Đọc một chuỗi ký tự do máy khách gửi đến
  String inLine = inStream.readLine();
//In chuỗi nhận được ra màn hình
  System.out.println("Received: "+inLine);
//So sánh chuỗi gửi đến. Nếu là "quit" thì chấm dứt
  if(inLine.equalsIgnoreCase("quit"))
finished=true;
//Xử lý đảo ngược chuỗi
  String outLine=new
        EchoString(inLine.trim()).getString();
//Gửi trả kết quả về cho máy khách
  for(int i=0;i<outLine.length();++i)
    outStream.write((byte)outLine.charAt(i));
  outStream.write(13);
  outStream.write(10);
  outStream.flush();
  System.out.println("Sent: "+outLine);
 } while(!finished);
//Đóng luồng và các kết nối
 inStream.close();
 outStream.close();
 client.close();
 server.close();
```

```
 }catch (IOException e){
//Nếu có lỗi thì in ra màn hình
   System.out.println(e);
 }
}
}
//Lớp này dùng để xử lý và đảo ngược nội dung của một chuỗi
class EchoString {
String s;
public EchoString(String in){
 int len = in.length();
 char outChars[] = new char[len];
//Đảo ngược các ký tự từ trước ra sau
 for(int i=0;i<len;++i)
   outChars[len-1-i]=in.charAt(i);
 s = String.valueOf(outChars);
}
public String getString(){
 return s;
}
}
```

Biên dịch chương trình

```
C:\Network>javac EchoServer.java
```

Hãy sử dụng chương trình EchoServer này sau khi ta xây dựng xong chương trình EchoClient.

6.2. Xây dựng chương trình EchoCient

EchoClient chạy trên máy khách nhận dữ liệu nhập vào từ bàn phím và gửi đến máy chủ để xử lý. Cổng kết nối mà chương trình EchoServer qui định là 3456. Máy khách sau đó sẽ nhận dữ liệu trả về từ máy chủ và in ra màn hình.

Ví dụ 1-2 EchoClient.java

```
import java.net.*;
import java.io.*;
public class Client {
 public static void main(String args[]){
  if (args.length !=1){
    System.out.println("Usage: java EchoClient
                        <serverAddr>");
    return;
  }
//Tạo đối tượng liên kết với máy chủ
  ClientConnect client = new
                    ClientConnect(args[0],3456);
//Nhận dữ liệu từ bàn phím và gửi đến máy chủ xử lý
  client.requestServer();
//Cắt đứt kết nối của đối tượng liên kết
  client.shutdown();
 }
}
//Thiết kế lớp liên kết với máy chủ và nhận dữ liệu từ bàn phím
class ClientConnect {
 Socket connection;        //Lưu giữ đối tượng kết nối
 DataOutputStream out;  //Luồng dùng gửi dữ liệu đến máy chủ
 BufferedReader in; //Luồng dùng đọc dữ liệu do máy chủ gửi về
/*
Phương thức khởi dựng tạo kết nối với máy chủ theo địa chỉ
destination và cổng port
```

```
*/
 public ClientConnect(String destination,int port){
  try{
// Tạo socket kết nối
   connection = new Socket(destination,port);
// Lấy về luồng nhập dùng đọc dữ liệu do máy chủ gửi về
   in = new BufferedReader(new
    InputStreamReader(connection.getInputStream()));
// Lấy về luồng xuất dùng gửi dữ liệu đến máy chủ
   out= new DataOutputStream(
                 connection.getOutputStream());
   System.out.println("Connected to server at port
                              3456.");
  } catch (Exception e) {
// Nếu gặp lỗi thì in lỗi ra màn hình
   System.out.println(e);
  }
 }
/*
Phương thức này sẽ nhận chuỗi dữ liệu từ bàn phím và gửi đến máy
chủ xử lý
*/
 public void requestServer(){
//Tạo luồng để đọc dữ liệu từ bàn phím
  BufferedReader keyboardInput = new BufferedReader(
            new InputStreamReader(System.in));
```

```
 boolean finished=false;
 do {
  try{
//Tạo menu lựa chọn. S: gửi dữ liệu, R: nhận dữ liệu, Q: chấm dứt
   System.out.print("Send, receive, or quit (S/R/Q):
");
   System.out.flush();
   String line = keyboardInput.readLine();
   if(line.length()>0){
    line=line.toUpperCase();
    switch (line.charAt(0)){
//Nếu người dùng chọn S, dữ liệu nhập vào sẽ được gửi đến máy
chủ
    case 'S':
     String sendLine = keyboardInput.readLine();
     out.writeBytes(sendLine);
     out.write(13);
     out.write(10);
     out.flush();
     break;
/*
Nếu người dùng chọn R, dữ liệu do máy chủ xử lý sẽ được nhận về
và in ra
*/
    case 'R':
     int inByte;
```

```
    System.out.print(">>>>");
    while ((inByte = in.read()) != '\n')
     System.out.write(inByte);
     System.out.println();
     break;
  //Nếu người dùng chọn Q, chương trình sẽ chấm dứt
    case 'Q':
     finished=true;
     break;
    default:
     break;
    }
   }
  }catch (Exception e){
  //Nếu có lỗi xuất nhập thì in lỗi ra màn hình
    System.out.println(e);
   }
 } while(!finished);
 }
/*
Phương thức này được dùng để đóng kết nối từ máy khách với máy
chủ
```

```
*/
public void shutdown(){
 try{
  connection.close();
 }catch (IOException ex){
  System.out.println("IO error closing socket");
 }
}
}
```

Biên dịch chương trình

```
C:\Network>javac EchoClient.java
```

Hãy chạy chương trình EchoServer trên một máy và EchoClient trên một máy khác nếu có hệ thống mạng (cả hai máy đều phải cài đặt máy ảo Java). Nếu thử nghiệm trên một máy đơn ta có thể mô phỏng mô hình khách/ chủ bằng cách mở hai cửa sổ dòng lệnh của DOS (DOS Prompt). EchoClient sẽ chạy trên một cửa sổ còn EchoServer chạy trên cửa sổ khác, quan sát kết quả trả về để thấy được cách mà mô hình khách/chủ làm việc (nhớ là hãy chạy EchoServer trước rồi mới đến EchoClient). Dưới đây là kết quả thử nghiệm:

Trên cửa sổ máy chủ :

```
C:\Network>java EchoServer
Echo Server is listening on port 3456.
```

Trên cửa sổ máy khách:

```
C:\Network>java EchoClient my.testing.server
    Connected to server at port 3456.
```

```
Send, receive, or quit (S/R/Q):
```

Và sau đây là quá trình trao đổi giữa máy khách và máy chủ:

Máy khách	Máy chủ
Send, receive, or quit (S/R/Q): s	Received: Hello
Hello	Sent: olleH
Send, receive, or quit (S/R/Q): r	
>>>> olleH	
	Received: 123456789
	Sent: 987654321
Send, receive, or quit (S/R/Q): s	
123456789	
Send, receive, or quit (S/R/Q): r	
>>>> 987654321	Received: How are you
	Sent: ?uoy era woH
Send, receive, or quit (S/R/Q): s	
How are you?	
Send, receive, or quit (S/R/Q): r	
>>>> ?uoy era who	
Send, receive, or quit (S/R/Q): q	

7. SỬ DỤNG UDP SOCKET

Trên đây là cách kết nối và truyền dữ liệu trên mạng theo giao thức TCP. Như đã đề cập, mặc dù truyền dữ liệu theo giao thức UDP không chính xác và bảo đảm bằng TCP nhưng bù lại nó không đòi hỏi nhiều tài nguyên của hệ thống, quá trình xử lý và

tiếp nhận dữ liệu cũng đơn giản hơn nhiều. Dữ liệu gửi đi theo giao thức UDP thường được đóng trong một gói (data package) bao gồm địa chỉ IP của máy nhận, số cổng cùng với dữ liệu thật sự. UDP được dùng trong các ứng dụng mang tính chất thông báo như về giá cả, thời tiết ...

Dưới đây ta sẽ xây dựng một mô hình ứng dụng khách/chủ (client/server) sử dụng lớp `DatagramSocket` và `DatagramPackage` để kết nối và trao đổi thông tin giữ máy khách và máy chủ bằng giao thức UDP. Chương trình `ExchangeRateServer` tại máy chủ sẽ cung cấp tỉ giá hối đoái của 3 thị trường chứng khoán Tokyo, NewYork và HongKong. Chương trình tại máy khách `ExchangeRateTable` sẽ cập nhật liên tục những thông tin này sau mỗi giây.

7.1. Xây dựng chương trình ExchangeRateServer

`ExchageRateServer` chạy trên máy chủ và đón nhận dữ liệu do máy khách gửi đến cổng 2345. Khi nhận được yêu cầu máy chủ sẽ gửi trả các thông báo về tỉ giá kèm theo ngày giờ mới nhất về cho máy khách. Ta sử dụng hàm `random` của lớp `Math` để lấy về tỉ giá mang tính chất ngẫu nhiên của 3 thị trường là Tokyo, NewYork và HongKong. Dữ liệu được nhận và gửi theo từng gói dựa vào lớp `DatagramPackage`.

Chương trình chạy trên máy chủ cung cấp tỉ giá của các thị trường chứng khoán.

Ví dụ 1-3 ExchangeRateServer.java

```
import java.net.*;
import java.io.*;
import java.util.*;
public class ExchangeRateServer {
 public static void main(String args[]){
```

```
try{
  //Tạo đối tượng kết nối socket theo giao thức UDP tại cổng 2345
 DatagramSocket socket = new DatagramSocket(2345);
  //Lấy về địa chỉ và số cổng kết nối của máy chủ
 String localAddress =
  InetAddress.getLocalHost().getHostName().trim();
 int localPort = socket.getLocalPort();
  //In các thông tin ra màn hình
 System.out.print(localAddress+": ");
 System.out.println("Exchange rate server is
listening on port "+localPort+".");
  //Tạo bộ đệm dùng để gửi và nhận dữ liệu do máy khách gửi đến
 int bufferLength = 256;
 byte outBuffer[];
 byte inBuffer[] = new byte[bufferLength];
  //Tạo gói dữ liệu để gửi đi
 DatagramPacket outDatagram;
  //Tạo gói dữ liệu để nhận về
 DatagramPacket inDatagram = new
          DatagramPacket(inBuffer,inBuffer.length);
  //Vòng lặp chờ nhận dữ liệu do máy khách gửi đến
 boolean finished = false;
 [redacted]
```

[redacted]ẽ được đưa vào gói inDatagram bao gồm
[redacted]a máy khách, số cổng mà tại đó dữ liệu

đã được gửi đi. Phương thức receive này thực sự dừng lại chờ cho đến khi nhậnđược gói dữ liệu mới chạy tiếp các lệnh kế.

```
  */
  socket.receive(inDatagram);
/*
```

Đã nhận được một gói dữ liệu từ phía máy khách. Lấy nội dung của gói dữ liệu, địa chỉ, số cổng và in ra màn hình

```
*/
  InetAddress destAddress =
inDatagram.getAddress();
  String destHost =
destAddress.getHostName().trim();
  int destPort = inDatagram.getPort();
  System.out.println("\nReceived a datagram from
          "+destHost+" at port "+destPort+".");
  String data = new
  String(inDatagram.getData()).trim();
System.out.println("It contained the data: "+data);
//Nếu nội dung gửi đến là chuỗi "quit" thì chấm dứt kết nối
  if(data.equalsIgnoreCase("quit")) finished=true;
//Lấy các thông tin về tỉ giá
  String s = new Date().toString();
  s=s+"\n NewYork  :"+getNewYorkRate();
  s=s+"\n Tokyo    :"+getTokyoRate();
  s=s+"\n HongKong :"+getHongKongRate();
//Gửi thông báo tỉ giá đến máy khách
  outBuffer=s.getBytes();
  outDatagram = new Datagram
```

```
            outBuffer.length,destAddress,destPort);
  socket.send(outDatagram);
  System.out.println("Sent "+s+" to "+destHost+
                    " at port"+destPort+".");
 } while(!finished);
 }catch (IOException ex){
  System.out.println("IOException occurred.");
 }
}
/*
Phương thức này trả về tỉ giá ngẫu nhiên của thị trường chứng
khoán NewYork
*/
private static String getNewYorkRate(){
    return Double.toString(Math.random()*135);
}
/*
Phương thức này trả về tỉ giá ngẫu nhiên của thị trường chứng
khoán Tokyo
*/
private static String getTokyoRate(){
    return Double.toString(Math.random()*135);
}
/*
Phương thức này trả về tỉ giá ngẫu nhiên của thị trường chứng
khoán HongKong
*/
private static String getHongKongRate(){
```

```
    return Double.toString(Math.random()*135);
 }
}
```

Biên dịch chương trình:

```
C:\Network>javac ExchangeRateServer.java
```

Chương trình `ExchangeRateServer` sẽ được sử dụng khi ta thiết kế xong chương trình `ExchangeRateTable` dùng cho máy khách.

7.2. Xây dựng chương trình ExchangeRateTable

`ExchangeRateTable` chạy trên máy khách, nó chịu tránh nhiệm mỗi giây gửi yêu cầu đến máy chủ để cập nhật thông tin về tỉ giá của thị trường chứng khoáng. Để đạt được mục tiêu này chương trình của ta phải xây dựng 3 lớp:

- Lớp `ExchangeRateTable`
  Là lớp chương trình chính dùng tạo một cửa sổ trình bày tỉ giá của các thị trường chứng khoán trong một vùng văn bản `TextArea`.
- Lớp `ExchangeThread`
  Đây là một phân tuyến chạy song song với chương trình chịu trách nhiệm mỗi giây sẽ gửi yêu cầu đến máy chủ và lấy số liệu về cập nhật.
- Lớp `ExchangeData`
  Là lớp thực hiện kết nối với máy chủ, lớp này sẽ thực sự cài đặt phương thức `getRates` dùng để gửi yêu cầu và nhận dữ liệu trả về từ máy chủ.

Dưới đây là toàn bộ chương trình `ExchangeRateTable`:

Ví dụ 1-4 ExchangeRateTable.java

```
import java.awt.*;
```

```
import java.awt.event.*;
import java.net.*;
import java.io.*;
public class ExchangeRateTable{
  public static void main(String args[]){
 //Tạo cửa sổ cho chương trình chính
     Frame myWindow =  new
            Frame("Stock Exchange Application");
 //Vùng văn bản dùng để thông báo tỉ giá
     TextArea rateTable = new TextArea("Wait ...");
     Label rateLabel = new
                Label("Exchange Rate Table");
//Đặt vị trí cho các đối tượng
     rateTable.setBounds(new
                Rectangle(16, 33, 240, 100));
     rateLabel.setBounds(new
                Rectangle(16, 6, 158, 21));
      myWindow.setLayout(null);
     myWindow.add(rateTable, null);
     myWindow.add(rateLabel, null);
//Xử lý tình huống cho cửa sổ chương trình chính
     myWindow.addWindowListener(new
WindowAdapter(){
        public void windowClosing(WindowEvent event){
            System.exit(0);
        }
     });
```

```
        myWindow.setSize(new Dimension(300, 150));
        myWindow.show();
//Tạo phân tuyến dùng để cập nhật tỉ giá sau mỗi giây
        ExchangeThread exRate = new
                    ExchangeThread(rateTable);
        exRate.start();
    }
}
/*
Lớp này cài đặt phân tuyến chạy song song dùng để cập nhật bảng tỉ
giá
*/
class ExchangeThread extends Thread{
 TextArea rateTable;
//Tạo đối tượng dùng trao đổi tỉ giá với máy chủ
 ExchangeData rate=new ExchangeData();
/*
Phương thức khởi dựng sẽ lưu giữ lại bảng tỉ giá do chương trình
chính đưa sang
*/
 public ExchangeThread(TextArea rateTable){
    this.rateTable=rateTable;
 }
 public void run() {
   while (true) {
   String data=rate.getRates();//Lấy tỉ giá mới từ máy chủ
      rateTable.setText(data);    //Cập nhật bảng tỉ giá
```

```
      delay(1000);
    }
  }
//Phương thức này dùng  trì hoãn một khoảng thời gian của phân tuyến
  private void delay(int miliSeconds) {
      try {
        this.sleep(miliSeconds);
      } catch (Exception e) {
        System.out.println("Sleep error !");
      }
  }
}
/*
Thiết kế lớp ExchangeData chịu trách nhiệm gửi và nhận số liệu trả về từ máy chủ
*/
class ExchangeData {
  DatagramSocket socket;        //Đối tượng lưu giữ kết nối
  InetAddress serverAddress;
  String localHost;
//Tạo bộ đệm dùng để gửi và nhận dữ liệu do máy chủ gửi đến
  int bufferLength = 256;
  byte inBuffer[] = new byte[bufferLength];
  byte outBuffer[];
  DatagramPacket outDatagram;  //Gói dữ liệu dùng để gửi đi
  DatagramPacket inDatagram;  //Gói dữ liệu dùng để nhận về
```

```
 public ExchangeData(){
  try{
//Tạo đối tượng kết nối socket theo giao thức UDP
   socket      = new DatagramSocket();
   //Tạo gói dữ liệu dùng để nhận về
   inDatagram  = new
         DatagramPacket(inBuffer,inBuffer.length);
//Lấy về đối tượng InetAddress cho biết chi tiết địa chỉ máy chủ
   serverAddress =
         InetAddress.getByName("my.testing.server");
  } catch (Exception e){
   System.out.println("connect error!");
  }
 }
/*
Phương thức này được dùng để chuyển yêu cầu và nhận dữ liệu trả
về từ máy chủ
*/
 public String getRates(){
  String data="";
  try{
//Tạo vùng đệm và gửi yêu cầu cập nhật tỉ giá đến máy chủ
   outBuffer = new byte[bufferLength];
   outBuffer = "rate".getBytes();
   outDatagram = new DatagramPacket(outBuffer,
            outBuffer.length,serverAddress,2345);
   socket.send(outDatagram);
```

```
//System.out.println("\nSent server at port 2345.");
 //Chờ nhận kết quả trả về từ máy chủ
  socket.receive(inDatagram);
//Lấy thông tin từ gói dữ liệu vừa nhận được
  InetAddress destAddress = inDatagram.getAddress();
 String destHost = destAddress.getHostName().trim();
  int destPort = inDatagram.getPort();
 //System.out.println("Received a data package from "
  //          +destHost+" at port "+destPort+".");
//Lấy dữ liệu thật sự chứa trong gói
  data = new String(inDatagram.getData());
  data=data.trim();
  //System.out.println("It contained data:"+data);
 }catch (IOException ex){
  System.out.println("IOException occurred.");
 }
 return data;
}
}
```

Biên dịch chương trình:

```
C:\Network>java ExchangeRateTable.java
```

Tương tự như chương trình EchoServer và EchoClient, hãy chạy chương trình ExchangeRateServer trên một máy và ExchangeRateTable trên một máy khác nếu có hệ thống mạng (cả hai máy đều phải cài đặt máy ảo Java). Nếu thử nghiệm trên một máy đơn ta có thể mô phỏng mô hình khách/chủ bằng cách mở hai cửa sổ dòng lệnh của DOS (DOS Prompt). ExchangeRateTable sẽ chạy trên một cửa sổ còn ExchangeRateServer chạy trên cửa sổ khác, quan sát kết quả trả về để thấy được bảng tỉ giá được gửi

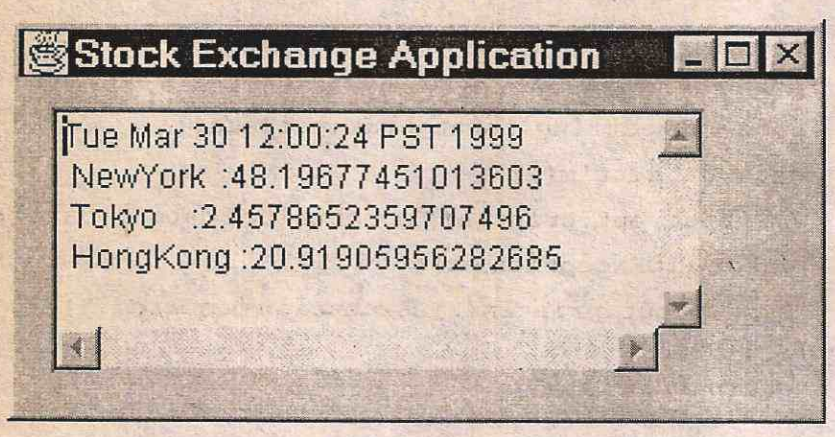
đi và cập nhật như thế nào ở cả hai phía máy khách và máy chủ (nhớ là hãy chạy `ExchangeRateServer` trước rồi mới đến `ExchangeRateTable`). Dưới đây là kết quả thử nghiệm:

Trên cửa sổ máy chủ:

```
C:\Network>java ExchangeRateServer
Echo Server is listening on port 3456.
```

Trên cửa sổ máy khách:

```
C:\Network>java ExchageRateTable
```



Khi chương trình `ExchangeRateTable` vừa chạy cứ cách khoảng 1 giây lại gửi yêu cầu đến máy chủ yêu cầu cập nhật số liệu về tỉ giá. Trên cửa sổ máy chủ (chương trình `ExchangeRateServer` đang chạy) sẽ xuất hiện các số liệu tỉ giá được sản sinh ngẫu nhiên và gửi đến máy khách như sau:

```
Received a datagram from my.testing.server at port 1222.
It contained the data: rate
Sent Tue Mar 30 12:00:24 PST 1999
 NewYork :48.19677451013603
 Tokyo   :2.4578652359707496
```

```
HongKong :20.91905956282685 to my.testing.server at
port 1222.
Received a datagram from my.testing.server at port 1221.
It contained the data: rate
Sent Tue Mar 30 12:00:26 PST 1999
NewYork  :92.64784702758493
Tokyo    :85.51780576017764
HongKong :82.58911770230986 to my.testing.server at
port 1221.
Received a datagram from my.testing.server at port 1222.
It contained the data: rate
Sent Tue Mar 30 12:00:30 PST 1999
NewYork  :103.77963834724935
Tokyo    :7.036632375300591
HongKong :32.88167554241727 to my.testing.server at
port 1222.
```

Trên đây là cách xây dựng các ứng dụng mạng dựa trên giao thức truyền dữ liệu TCP và UDP. Các ứng dụng mạng đa số đều dựa trên mô hình khách/chủ: một chương trình chạy trên máy chủ cung cấp các dịch vụ, một chương trình chạy trên máy khách gửi các yêu cầu đến máy chủ để nhận được sự hỗ trợ về các dịch vụ này. Dưới đây ta sẽ đi vào chi tiết cũng như thiết kế các ứng dụng đặc trưng cho từng phía.

8. XÂY DỰNG CÁC CHƯƠNG TRÌNH Ở MÁY KHÁCH (CLIENT)

8.1. Các chương trình chạy trên máy khách (client)

Khi thiết kế chương trình theo mô hình khách/chủ thông thường ta phân ra làm hai phần, một phần là chương trình viết chạy

trên máy chủ (như `EchoServer` hay `ExchangeRateServer`) còn một phần là chương trình viết chạy trên máy khách (như `EchoClient` hay `ExchangeRateTable`). Các chương trình ở máy khách nhìn chung thường thực hiện những nhiệm vụ sau:

- Kết nối với máy chủ
- Nhận dữ liệu hay yêu cầu do người dùng nhập vào từ bên ngoài.
- Gửi dữ liệu hay yêu cầu đến máy chủ để xử lý.
- Trình bày kết quả trả về từ máy chủ cho người dùng xem.

Các chương trình thực hiện những tác vụ này ở máy khách rất đa dạng. Có thể kể một số loại ứng dụng tiêu biểu sau:

- *Chương trình đọc và gửi E-mail:* Tạo điều kiện cho người dùng gửi và nhận E-mail. Soạn thảo E-mail trước khi gửi đi, lấy các bức e-mail về từ máy chủ và cho xem nội dung của nó, quản lý các bức e-mail theo ngày tháng ... Các chương trình gửi nhận mail nổi tiếng điển hình như : trên Windows có Edora, Outlook, trên hệ UNIX có Pine, Elm, mh ...
- *Trình duyệt (Web browser)* : Dùng truy tìm và đọc các trang Web trên mạng. Địa chỉ trang Web sẽ được các trình duyệt gửi đến máy chủ và máy chủ sẽ gửi trả về nội dung trang Web bao gồm các dữ liệu như văn bản, hình ảnh, âm thanh ...Trình duyệt sẽ chịu trách nhiệm trình bày kết quả trang Web ra màn hình với những loại dữ liệu nhận được này. Các trình duyệt điển hình như Internet Explorer, Netscape Navigator đều được hỗ trợ chạy trên Windows, UNIX, Macintosh và các hệ thống khác.
- *Chương trình truyền tập tin FTP:* Hỗ trợ việc sao chép tập tin trên mạng, tập tin có thể được chép về hay chuyển lên một máy chủ nào đó nhờ vào chương trình giao tiếp với máy chủ FTP. Thường thì những chương trình này sử dụng các lệnh cơ bản như `get` để lấy tập tin về và `send` để gửi tập tin đi. Chương trình FTP theo giao diện Windows nổi tiếng hiện nay có WS_FTP.
- *Chương trình tổ chức nhóm tin (News group):* Hỗ trợ cho việc thảo luận, trao đổi thảo luận và giải quyết về một vấn đề nào đó trong nhóm. Hiện có rất nhiều chương trình phục vụ cho mục

đích này dành cho đủ loại hệ điều hành như Windows, UNIX, Macintosh ...

- *Chương trình Telnet dùng đăng nhập và truy xuất từ xa*: Một vài hệ thống cho phép đăng nhập và xử lý tập tin từ xa như hệ UNIX chẳng hạn. Hệ điều hành UNIX hỗ trợ một chương trình Telnet chạy trên máy chủ có thể kết nối với máy khách để xử lý và chạy các lệnh UNIX y như đang ngồi trên máy chủ vậy. Phần sau ta sẽ thiết kế một chương trình Telnet ở máy khách thực hiện được khả năng này.

Một vài ứng dụng gộp chung các chứng năng này lại như Netscape Comminicator tích hợp vừa trình duyệt, gửi nhận E-mail, nhóm tin ...

Dưới đây ta sẽ xây dựng một số chương trình Java minh họa cho cách làm việc của các ứng dụng trên máy khách.

8.2. Xây dựng chương trình ReadWebPage

`ReadWebPage` tương tự như một trình duyệt (browser) thô sơ và không có gì hấp dẫn cả, nó chỉ gửi yêu cầu cần đọc một trang Web đến địa chỉ máy chủ sau đó nhận nội dung trang Web do máy chủ gửi về và cho in ra màn hình.

Lưu ý : Đối với một trình browser thực thụ thì sau khi tiếp nhận được nội dung, trang Web sẽ tiến hành bước phân tích các thẻ (tag) HTML của trang Web để trình bày nội dung trang Web theo một định dạng nào đó biểu diễn được văn bản, hình ảnh âm thanh thật sống động.

Chương trình đọc trang Web từ máy chủ.

Ví dụ 1-5 ReadWebPage.java

```
import java.net.*;
import java.io.*;

public class ReadWebPage {
 public static void main(String args[]){
```

```
  try{
   if(args.length!=1) {
   System.out.println("Usage  : java ReadWebPageApp
                                <URL>");
   System.out.println("Example: java ReadWebPageApp
                    http://java.sun.com");
    return;
   }
//Tạo đối tượng URL định vị địa chỉ trang Web
    URL u=new URL(args[0]);
//Mở luồng nhập để đọc nội dung trang Web trả về từ máy chủ
   BufferedReader in = new BufferedReader(
        new InputStreamReader(u.openStream()));
//Đọc nội dung trang Web
   String line;
    while((line = in.readLine()) != null) {
        System.out.println(line);
    }
   }
   catch (MalformedURLException e){
//Lỗi do sai địa chỉ hay không tìm thấy địa chỉ trang Web
      System.out.println("wrong address");
   }
   catch(IOException e){
//Lỗi do kết nối và truyền dữ liệu
      System.out.println("open stream Error!");
   }
```

```
}
}
```

Biên dịch chương trình:

```
C:\Network>javac ReadWebPage.java
```

Nếu kết nối được với Internet, chương trình `ReadWebPage` của ta sẽ đọc được bất cứ địa chỉ trang Web Internet nào, ví dụ sau sẽ đọc về nội dung trang chủ tại địa chỉ `http://java.sun.com`:

```
C:\Network>java ReadWebPage http://java.sun.com
```

Nếu không kết nối được với Internet hãy chờ cho đến khi thiết kế xong chương trình Webserver, sau này ta có thể thử nghiệm mô hình khách/chủ và cách truy cập trang Web trên một máy duy nhất.

9. XÂY DỰNG CÁC CHƯƠNG TRÌNH Ở MÁY CHỦ (SERVER)

Các chương trình thiết kế chạy trên máy chủ thường phải lắng nghe các kết nối từ máy khách gửi đến. Tiếp nhận những yêu cầu từ phía máy khách sau đó xử lý và gửi trả kết quả về cho máy khách. Như vậy chương trình ở máy chủ đóng vai trò như một dịch vụ (service) phục vụ cho máy khách. Có rất nhiều loại dịch vụ trên máy chủ điển hình gồm có:

- *E-mail server*: Cung cấp dịch vụ E-mail, các bức E-mail do máy khách gửi đi sẽ được chương trình mail server ở máy chủ quản lý lưu giữ tạm thời và dò tìm địa chỉ để gửi được E-mail đến đích.
- *Web server*: Cung cấp dịch vụ dò tìm trang Web trên máy chủ. Khi máy khách yêu cầu lấy về một trang Web (thường yêu cầu này do các trình duyệt browser gửi đến) chương trình Web server sẽ đọc nội dung trang Web và gửi toàn bộ về cho máy khách (trước khi gửi đi có thể thực hiện một số thao tác biến đổi khác như thông qua các chương trình CGI hay Active Server Page ...).

- *FTP server*: Tương ứng với chương trình FTP ở máy khách, máy chủ cũng phải có một chương trình FTP server để kết nối và nhận hay truyền tập tin về cho máy khách.

- *DNS server (Domain Name System server):* Máy chủ quản lý tên vùng. Thường muốn truy cập các máy chủ khác ta phải biết địa chỉ IP của nó hoặc tên mà máy chủ được đặt. Nếu có khả năng lưu và quản lý nổi ta có thể chứa tất cả địa chỉ IP và tên tương ứng của nó trên máy mình (chẳng hạn lưu trong tập tin hosts). Thế nhưng nếu có quá nhiều địa chỉ ta có thể nhờ một máy chủ khác quản lý tất cả những địa chỉ này. Như vậy chỉ cần nhớ địa chỉ của máy chủ DNS, khi có một tên máy cần xác minh địa chỉ IP hệ thống sẽ tự động tìm đến máy chủ DNS để lấy về địa chỉ IP thực sự đi kèm với tên.

- *Telnet server*: Thường chạy trên các hệ hỗ trợ đa người dùng như UNIX, VMS... (Windows không hỗ trợ cách đăng nhập và xử lý từ xa kiểu Telnet). Telnet server được dùng để trao đổi với chương trình Telnet chạy ở máy khách. Nó khiến người dùng phía máy khách có cảm tưởng như đang tương tác và xử lý trực tiếp trên hệ thống máy chủ.

Dưới đây ta sẽ xây dựng một chương trình khá nổi tiếng có tên là Webserver. Chương trình này chạy trên máy chủ và có thể trả về bất cứ trang Web nào mà máy khách yêu cầu (tương tự như chương trình Personal Webserver của Windows 95 hay NT Webserver của Windows NT). Một trình duyệt browser thực thụ như Internet Explorer chẳng hạn sẽ liên lạc được với chương trình Webserver của ta và lấy về trang Web với đầy đủ hình ảnh, âm thanh, định dạng văn bản ... (Chương trình Webserver hoàn toàn có thể thử nghiệm trên một máy hoặc nhiều máy trong hệ thống mạng cục bộ)

9.1. Thiết kế chương trình Webserver

Việc xây dựng chương trình Webserver cũng dựa trên mô hình khách/chủ mà ta đã làm quen trước đây. Webserver là chương trình

chạy trên máy chủ cho nên nó cũng cần thực hiện một số bước cơ bản như là: tạo socket và lắng nghe các kết nối xuất phát từ phía máy khách, nếu có kết nối thì xử lý bằng cách đọc tập tin .html của trang Web và gửi nội dung trang Web về cho máy khách sau đó thì đóng kết nối lại. Toàn bộ chương trình Webserver của ta sẽ được xây dựng từ những lớp sau:

- `JavaWebServer`
  Lớp chương trình chính.
- `JavaWebServerThread`
  Lớp xử lý từng kết nối riêng biệt từ phía máy khách.
- `RequestData`
  Lớp tiếp nhận và xử lý các yêu cầu do máy khách đưa đến
- `ServerInputStream`
  Lớp xử lý luồng nhập dùng để đọc dữ liệu do máy khách gửi đến
- `ServerOutputStream`
  Lớp xử lý luồng xuất dùng để gửi dữ liệu (hay nội dung trang Web) về cho máy khách.
- `ServerConfiguration`
  Lớp này đọc và xử lý các thiết lập cấu hình cần thiết cho chương trình Webserver.
- `Logger`
  Lớp này dùng để kết xuất thông báo ra màn hình hoặc ra máy tập tin.

Dưới đây ta sẽ xây dựng dần dần các lớp cần thiết cho chương trình `JavaWebserver` và sao đó sẽ kết hợp chúng lại như một chương trình duy nhất.

Để dễ quản lý, hãy tạo thêm thư mục con `C:\Network\Webserver`, tất cả các lớp được thiết kế dưới đây sẽ được lưu trữ và biên dịch trong thư mục này.

9.1.1. Xây dựng lớp Logger

Trong quá trình giao tiếp với máy khách chương trình Webserver của ta có nhu cầu thường xuyên phải xuất ra các thông báo để theo

dõi kết quả và ngày giờ thực hiện những giao dịch với máy khách. Lớp `Logger` cung cấp hai phương thức chính là `writeMsg` dùng để xuất ra chuỗi thông báo và `writeDateMsg` dùng để xuất ra chuỗi thông báo kèm theo ngày tháng. Thông báo có thể xuất ra tập tin, màn hình hay cả hai tùy cấu hình mà ta thiết lập trong lúc khởi tạo. (trong thực tế các chương trình Webserver thật sự thường xuất ra rất nhiều thông tin để theo dõi như ngày giờ kết nối, ngày giờ chấm dứt, lỗi trong quá trình truy xuất ... tất cả những thông tin này cực kỳ quan trọng để các nhà quản trị mạng biết cách theo dõi và chẩn đoán lỗi.)

Ví dụ 1-6 Logger.java

```
public class Logger {
  public String logFile;
  public boolean EchoLogToConsole = true;
  public BufferedWriter writer   = null;
 //Phương thức khởi dựng dùng các thông số mặc định
  public Logger() {
  }
/*
Phương thức khởi dựng cho biết tên tập tin để đưa các thông báo ra,
Echo cho biết sau khi đã đưa thông báo vào tập tin xong có cần hiển
thị ra màn hình hay không
*/
  public Logger(String fname, boolean Echo) {
    logFile = fname;
    EchoLogToConsole = Echo;
    try {
      writer = new BufferedWriter(new
                    FileWriter(fname,true));
```

```
    }catch(IOException ex){}
  }
//Phương thức này dùng để ghi ra một thông báo
  public synchronized void writeMsg(String msg) {
    writeLog(" "+msg);
  }
//Phương thức này dùng để ghi ra một thông báo kèm theo ngày giờ
  public synchronized void writeDateMsg(String msg) {
    writeLog((new Date()).toString()+" "+msg);
  }
//Phương thức này sẽ ghi thông báo ra tập tin hoặc cả màn hình
  void writeLog(String msg) {
    if(writer!=null) {
      try {
        writer.write(msg);
        writer.newLine();
        writer.flush();
      }catch(IOException ex){}
    }
    if(EchoLogToConsole) System.out.println(msg);
  }
}
```

Biên dịch chương trình:

```
C:\Network\Webserver>javac Logger.java
```

9.1.2. Xây dựng lớp ServerConfiguration

Thiết lập cấu hình ban đầu cho máy chủ cũng là bước khá quan trọng. Để linh động ta cho phép đặt các thông số cấu hình ban đầu cho chương trình Webserver trong một tập tin (tương tự các tập tin .ini của Windows). Lớp `ServerConfiguaration` này sẽ chịu trách nhiệm đọc và phân tích nội dung của tập tin cấu hình. Cấu hình tối thiểu mà ta cần thiết lập bao gồm những thông số sau:

- `serverPort:` số hiệu cổng để các máy khách kết nối vào.

- `logFile`: tên tập tin để xuất ra các thông báo theo dõi.
- `EchoLogToConsole`: cờ trạng thái cho phép in các thông báo ra màn hình hay không.
- `defaultFile`: tên tập tin html được dùng làm trang chủ mặc định.
- `serverRoot`: đường dẫn được coi là thư mục gốc trên máy chủ khi các máy khách truy cập vào.

Khi yêu cầu một trang Web từ máy chủ ta thường đánh vào địa chỉ định vị URL chẳng hạn:

`http://my.testing.server/hotnews.htm`

khi đó tập tin `hotnews.htm` trên máy chủ sẽ được đặt trong một thư mục ánh xạ được coi là thư mục gốc của máy chủ khi các máy khách truy cập vào.

Ví dụ trên máy chủ `my.testing.server` ta ánh xạ thư mục `C:\Webserver\Webroot` làm thư mục gốc và nội dung thư mục này như sau:

```
C:\Webserver\Webroot
        index.htm
        hotnews.htm
        BEANS          <DIR>
                  newBeans.htm
                  JFC.htm
```

Thư mục `C:\Webserver\webroot` bao gồm 2 tập tin `.htm` là `index.htm` và `hotnews.htm` (còn được coi là 2 trang Web) cùng với một thư mục con `BEANS` bao gồm 2 tập tin `.htm` khác là `newBeans.htm` và `JFC.htm`. Nếu đặt tập tin mặc định (defaulFile) làm trang chủ là `index.htm` thì máy khách có thể yêu cầu các trang Web từ máy chủ và nhận về các kết quả như sau:

- `http://my.testing.server`

Trả về trang `index.htm` (tập tin `index.htm` được chỉ định là trang chủ)

- http://my.testing.server/index.htm

  Trả về trang index.htm
- http://my.testing.server/hotnews.htm

  Trả về trang hotnews.htm
- http://my.testing.server/Beans/JFC.htm

  Trả về trang JFC.htm
- http://my.testing.server/Beans/newBeans.htm

  Trả về trang newBeans.htm

Tập tin cấu hình của ta là một tập tin văn bản có tên là webserver.ini với nội dung như sau:

```
# JavaWebserver Configuration File
# Set default port to 6789 instead of using port 80
serverPort 6789
# Set the server root to C:\Network\webserver\
serverRoot C:\Network\webserver\
# Set the default file to index.htm
defaulFile index.htm
# Set log file to log.txt
logFile log.txt
#Disable Echo to consol
EchoLogToConsole false
```

Dưới đây là lớp ServerConfiguration dùng để đọc và phân tích nội dung tập tin cấu hình dùng cho chương trình Webserver của ta sau này:

Ví dụ 1-7 ServerConfiguration.java

```
class ServerConfiguration {
  static final char CONFIG_COMMENT_CHAR = '#';
```

//Cổng kết nối

```
  int serverPort        = 80;
//Đường dẫn đến thư mục được coi là thư mục gốc của máy chủ
  String serverRoot     = "";
//Trang chủ mặc định của máy chủ
  String defaultFile    = "index.htm";
//Tập tin log để ghi ra các thông tin theo dõi
  String logFile        = "";
//Mặc định là in các thông báo ra màn hình.
  boolean EchoLogToConsole = true;
  public ServerConfiguration() {
  }
//Trả về ký tự dùng làm chú thích trong tập tin cấu hình
  public char getCommentChar() {
    return CONFIG_COMMENT_CHAR;
  }
//Trả về số hiệu cổng kết nối
  public int getServerPort() {
    return serverPort;
  }
//Trả về thư mục được chỉ định là thư mục web gốc trên máy chủ
  public String getServerRoot() {
    return serverRoot;
  }
//Trả về tên tập tin html được dùng làm trang chủ
  public String getDefaultFile() {
    return defaultFile;
  }
```

```
//Trả về tên tập tin dùng thiết lập cấu hình cho chương trình
JavaWebServer
  public String getLogFile() {
    return logFile;
  }
//Trả về trạng thái được xuất các thông báo ra màn hình hay không
  public boolean EchoToConsole() {
    return EchoLogToConsole;
  }
//Phương thức này dùng trình bày nội dung cấu hình của chương
trình ra màn hình
  public void display() {
  System.out.println(" serverPort: "+serverPort);
  System.out.println(" serverRoot: "+serverRoot);
  System.out.println(" defaultFile: "+defaultFile);
  System.out.println(" logFile: "+logFile);
  System.out.println(" EchoLogToConsole: "
                                  +EchoLogToConsole);
  }
//Phương thức này đọc và xử lý tập tin cấu hình nếu có
  public void processConfigurationFile(String
fname) {
    try {
      File file = new File(fname);
      if(file.exists()) {
        BufferedReader reader = new
          BufferedReader(new FileReader(file));
```

```
        String line;
        while((line=reader.readLine())!=null)
          processConfigurationLine(line);
        reader.close();
      }
    }catch(Exception ex) {
      System.out.println("Unable to process
                    configuration file.");
    }
  }
/*
Phương thức này xử lý nội dung tập tin cấu hình theo từng dòng:
Các giá trị như cổng kết nối, tên tập tin html làm trang chủ, đường
dẫn được coi là thư mục gốc trên máy chủ .. sẽ được đọc vào nếu có.
*/
  public void processConfigurationLine(String line)
                    throws NumberFormatException {
    line = removeLeadingWhiteSpace(line);
    if(line.length()==0) return;
    int n;
    int n1 = line.indexOf(' ');
    int n2 = line.indexOf('\t');
    if(n1 == -1) n = n2;
    else if(n2 == -1) n = n1;
    else if(n1 < n2) n = n1;
    else n = n2;
    if(n==-1 || n==line.length()-1) return;
```

```
    String param = line.substring(0,n);
    String value = line.substring(n+1);
  /*
  Lấy về đường dẫn thư mục được coi là thư mục gốc trên máy chủ
  nếu có
  */
    if(param.equals("serverPort"))
      serverPort = (new Integer(value)).intValue();
    else if(param.equals("serverRoot")){
      serverRoot = value;
      if(!serverRoot.equals("")){
        char ch =
          serverRoot.charAt(serverRoot.length()-1);
        if(ch!='/' && ch!='\\') serverRoot+="/";
      }
     //Lấy về tên tập tin html được dùng làm  trang chủ nếu có
    }else if(param.equals("defaultFile"))
              defaultFile = value;
     //Lấy về tên tập tin được dùng làm tập tin log nếu có
     else if(param.equals("logFile"))
              logFile = value;
//Lấy về trạng thái cho phép xuất thông báo ra màn hình hay không
    else if(param.equals("EchoLogToConsole"))
        EchoLogToConsole = (new
              Boolean(value)).booleanValue();
  }
/*
```

Phương thức này dùng xóa các khoảng trắng đầu chuỗi và loại bỏ ký tự chú thích

```
*/
private String removeLeadingWhiteSpace(String line) {
    boolean finished = false;
    do {
      if(line.length()==0) return "";
      char ch = line.charAt(0);
      if(ch==CONFIG_COMMENT_CHAR) return "";
      if(ch!=' ' && ch!='\t') return line;
      line=line.substring(1);
    } while (!finished);
    return "";
 }
}
```

Biên dịch chương trình:

```
C:\Network\Webserver>javac
ServerConfiguration.java
```

9.1.3. Xây dựng lớp RequestData

Đối tượng `RequestData` được gọi bởi lớp `ServerInputStream` (ta sẽ thiết kế lớp này sau) dùng để lưu các yêu cầu mà máy khách gửi đến theo định dạng chuỗi. `RequestData` cung cấp ba phương thức:

- `isGetMethod` để xác định xem yêu cầu mà máy khách gửi đến có phải là muốn lấy về một trang Web hay không (nếu là yêu cầu truy xuất trang Web thì dòng đầu tiên của dữ liệu yêu cầu gửi đến phải chứa chuỗi "GET")
- `getFileName` dùng để lấy về tên và đường dẫn đầy đủ trỏ đến tập tin trang Web mà máy khách yêu cầu.
- `writeToLog` dùng để xuất tất cả các yêu cầu của máy khách ra tập tin log dùng để theo dõi (sử dụng đối tượng `Logger` đã thiết kế trước đây).

Ví dụ 1-8 RequestData.java

```
class RequestData {
```

```
//Đối tượng vector dùng lưu trữ các chuỗi
  Vector lines = new Vector();
  public RequestData() {
  }
//Phương thức này dùng để đưa một chuỗi vào đối tượng Vector lines
  public void addLine(String line) {
    lines.addElement(line);
  }
/*
Phương thức này xác nhận xem yêu cầu từ máy khách gửi đến có
phải là muốn lấy về một trang Web hay không.
*/
  boolean isGetMethod() {
    if(lines.size() > 0) {
      String firstLine =
(String)lines.elementAt(0);
      if(firstLine.length() > 0)
        if(firstLine.substring(0,3).
              equalsIgnoreCase("GET"))
          return true;
    }
    return false;
  }
/*
Phương thức này sẽ trả về tên tập tin mà máy khách yêu cầu lấy về.
Nếu không tìm thấy tên tập tin trong yêu cầu mà máy khách gửi
đến, tên tập tin của trang chủ mặc định sẽ được trả về.
*/
 private String getFileName(ServerConfiguration
config) {
```

```
    if(lines.size()>0) {
   //Phân tích và tìm tên tập tin do máy khách gửi đến
    String firstLine = (String)lines.elementAt(0);
    String fileName =
    firstLine.substring(firstLine.indexOf(" ")+1);
    int n = fileName.indexOf(" ");
    if(n!=-1) fileName = fileName.substring(0,n);
   try {
        if(fileName.charAt(0) == '/')
            fileName = fileName.substring(1);
      } catch(StringIndexOutOfBoundsException ex) {
      }
//Nếu không thấy thì trả về tên của trang chủ mặc định
     if(fileName.equals(""))
            fileName = config.getDefaultFile();
     if(fileName.charAt(fileName.length()-1)=='/')
         fileName+=config.getDefaultFile();
      return config.getServerRoot()+fileName;
    }else return "";
  }
/*
Phương thức này dùng để ghi tất cả các yêu cầu nhận được từ máy
khách ra màn hình hay ra tập tin log (tùy theo chọn lựa thiết lập
trong tập tin cấu hình)
*/
  void writeToLog(Logger log) {
 log.writeDateMsg("Received the following request:");
```

```
    for(int i=0;i<lines.size();++i)
      log.writeMsg((String) lines.elementAt(i));
  }
}
```

Biên dịch chương trình:

```
C:\Network\Webserver>javac RequestData.java
```

9.1.4. Xây dựng lớp ServerInputStream

ServerInputStream là một bộ lọc dùng tiếp nhận tất cả dữ liệu (hay yêu cầu) do máy khách gửi đến và đưa vào đối tượng RequestData. Sau này ta sẽ sử dụng phương thức getRequestData của lớp ServerInputStream để lấy về đối tượng RequestData thể hiện các yêu cầu do máy khách gửi đến.

Ví dụ 1-9 ServerInputStream.java

```
class ServerInputStream extends FilterInputStream {
  public ServerInputStream(InputStream in) {
    super(in);
  }
//Phương thức này để đọc ra một chuỗi từ luồng
  public String readLine() throws IOException {
    StringBuffer result=new StringBuffer();
    boolean finished = false;
    boolean cr = false;
    do {
      int ch = -1;
      ch = read();
      if(ch==-1) return result.toString();
```

```
        result.append((char) ch);
        if(cr && ch==10){
          result.setLength(result.length()-2);
          return result.toString();
        }
        if(ch==13) cr = true;
        else cr=false;
      } while (!finished);
      return result.toString();
    }
/*
Phương thức này trả về đối tượng RequestData chứa dữ liệu mà máy
khách yêu cầu dưới dạng các chuỗi.
*/
public RequestData getRequest() throws IOException {
      RequestData request = new RequestData();
      String line;
      do {
        line = readLine();
        if(line.length()>0) request.addLine(line);
        else break;
      }while(true);
      return request;
    }
}
```

Biên dịch chương trình:

```
C:\Network\Webserver>javac ServerInputStream.java
```

9.1.5. Xây dựng lớp ServerOutputStream

`ServerOutStream` là một bộ lọc dùng xử lý dữ liệu (hay nội dung trang Web) trước khi gửi trả kết quả về cho máy khách. Chương trình Webserver của ta không làm gì cả ngoài việc hỗ phương thức `println` để có thể ghi dữ liệu gửi đi dưới dạng từng chuỗi.

Lưu ý : Các chương trình Webserver rất đa dạng và thật ra cũng không hẳn là máy khách yêu cầu một tập tin html thì bắt buộc chương trình Webserver phải trả về tập tin html theo yêu cầu. Những chương trình trên máy chủ hiện nay như Active Server Page của Microsoft chẳng hạn cung cấp thêm khả năng xử lý trước khi chuyển thành tập tin html thật sự, các nhà thiết kế có thể tạo nên những tập tin văn bản `.asp` là hình thức mở rộng của tập tin html ở dạng lập trình được, khi trình duyệt (browser) của máy khách yêu cầu chuyển về nội dung một tập tin .asp, chương trình webserver ở máy chủ sẽ đọc và xử lý các lệnh trong tập tin .asp, sau đó gửi trả về máy khách tập tin html là kết quả sau quá trình xử lý. Trình duyệt ở máy khách sẽ nhận được tập tin html như là một trang Web bình thường (chứ không phải tập tin `.asp` như đã yêu cầu). Ở các chương sau chúng ta sẽ cùng nghiên cứu về JSP (Java Active Server Page) với những đặc điểm thú vị này.

Ví dụ 1-10 ServerOutputStream.java

```
class ServerOutputStream extends FilterOutputStream {
  public ServerOutputStream (OutputStream out) {
    super(out);
  }
  public void println() throws IOException{
    write(13);
```

```
    write(10);
  }
  public void println(String s) throws IOException{
  for(int i=0;i<s.length();++i) write(s.charAt(i));
    println();
  }
}
```

Biên dịch chương trình:

```
C:\Network\Webserver>javac ServerInputStream.java
```

9.1.6. Xây dựng lớp JavaWebServerThread

Chương trình Webserver của ta phải có khả năng truy tìm và trả về trang Web cùng một lúc cho nhiều máy khách khác nhau kết nối vào. Để đạt được yêu cầu này ta xây dựng lớp `JavaWebServerThread` như là một phân tuyến (thread) có khả năng xử lý yêu cầu của từng máy khách riêng rẽ. Khi nhận được kết nối và yêu cầu truy tìm trang Web từ một máy khách, chương trình chính sẽ giao công việc này cho một phân tuyến `JavaWebServerThread` chịu trách nhiệm xử lý và gửi kết quả về cho máy khách đã yêu cầu.

Ví dụ 1-11 JavaWebServerThread.java.

```
class JavaWebServerThread extends Thread {
Socket client; //Đối tượng socket lưu lại kết nối từ máy khách
ServerConfiguration config; //Đọc các thông tin cấu hình
Logger log; //Đối tượng Logger dùng để xuất các thông báo theo dõi
  public JavaWebServerThread(Socket client,
      ServerConfiguration config, Logger log) {
    this.client = client;
```

```
    this.config = config;
    this.log    = log;
  }
  public void run() {
   try{
//In ra màn hình các thông tin về máy khách
    describeConnection(client);
//Tạo các luồng xuất nhập để nhận và gửi dữ liệu đến máy khách
    ServerOutputStream outStream = new
    ServerOutputStream(client.getOutputStream());
    ServerInputStream inStream = new
    ServerInputStream(client.getInputStream());
//Lấy về yêu cầu từ máy khách
    RequestData request = inStream.getRequest();
//Xuất các yêu cầu này ra màn hình hoặc tập tin log
    request.writeToLog(log);
//Xác định và xử lý yêu cầu truy xuất trang Web từ máy khách
    if(request.isGetMethod())
        processGetMethod(request,outStream);
    log.writeDateMsg("Request completed. Closing
                        connection.");
   }catch(IOException ex) {
     log.writeDateMsg("IOException occured when
                    processing request.");
   }
//Đóng kết nối với máy khách
```

```
    try {
      client.close();
    }catch(IOException ex) {
      log.writeDateMsg("IOException occured when
                        closing socket.");
    }
  }
//Phương thức này dùng xuất ra các thông tin kết nối từ máy khách
  void describeConnection(Socket client) {
    String destName =
          client.getInetAddress().getHostName();
    String destAddr =
          client.getInetAddress().getHostAddress();
    int destPort = client.getPort();
    log.writeDateMsg("Accepted connection to
        "+destName+" ("+destAddr+")"+" on port "+
                        destPort+".");
  }
/*
Phương thức này dùng truy tìm và gửi về trang Web mà máy khách
yêu cầu
*/
  void processGetMethod(RequestData
            request,ServerOutputStream outStream)
        throws IOException {
//Lấy về tên và đường dẫn của trang Web mà máy khách yêu cầu
    String fileName = request.getFileName(config);
```

```
//Đọc nội dung trang Web
      File file = new File(fileName);
      if(file.exists()) {
        String fullPath = file.getCanonicalPath();
        if(inServerRoot(fullPath)) {
          int len = (int) file.length();
//Gửi trang Web đi
          sendFile(outStream,file);
        }else log.writeDateMsg("File is not in server
                                root.");
      }else log.writeDateMsg("File"+
          file.getCanonicalPath()+" does not exist.");
    }
/*
Phương thức này xác nhận xem trang Web truy tìm có nằm trong thư
mục ánh xạ được coi là thư mục gốc trên máy chủ hay không.
*/
    public boolean inServerRoot(String fileName) {
      String serverRoot = config.getServerRoot();
      int fileLength = fileName.length();
      int rootLength = serverRoot.length();
      if(fileLength<rootLength) return false;
      if(serverRoot.equals(
       fileName.substring(0,rootLength))) return true;
       return false;
    }
/*
Phương thức này gửi trả nội dung trang Web (hay tập tin html) về
cho máy khách
*/
```

```
  void sendFile(ServerOutputStream out,File file) {
    try {
      DataInputStream in = new DataInputStream(new
                    FileInputStream(file));
      int len = (int) file.length();
      byte buffer[] = new byte[len];
      in.readFully(buffer);
      out.write(buffer);
      out.flush();
      in.close();
      out.close();
      log.writeDateMsg("File sent:
                "+file.getCanonicalPath());
      log.writeMsg("Number of bytes: "+len);
    }catch(Exception ex){
      log.writeDateMsg("Error retrieving "+file);
    }
  }
}
```

Biên dịch chương trình:

```
C:\Network\Webserver>javac JavaWebServerThread.java
```

9.1.7. Xây dựng chương trình JavaWebServer

Chương trình Webserver coi như đã hoàn tất với các lớp cần thiết. Dưới đây ta sẽ kết hợp các đối tượng cụ thể từ những lớp trên lại với nhau thành chương trình mang tên `JavaWebServer` chạy được thực thụ.

Chương trình chính JavaWebServer sẽ thực hiện những công việc sau:

- Đọc và xử lý tập tin cấu hình nếu có (còn không sẽ dùng các giá trị mặc định của đối tượng `ServerConfiguration`).
- Khởi tạo đối tượng ServerSocket để lắng nghe các kết nối từ phía máy khách gửi đến.
- Dùng vòng lặp liên tục lắng nghe các kết nối từ máy khách. Nếu có kết nối xảy ra thì tạo một phân tuyến `JavaWebServerThread` và giao cho nó xử lý yêu cầu của máy khách.
- Tiếp tục lắng nghe các kết nối khác.

Dưới đây là toàn bộ chương trình `JavaWebServer`: Chương trình Webserver chạy trên máy chủ giúp các máy khách truy tìm trang Web.

Ví dụ 1-12 JavaWebServer.java

```
import java.net.*;
import java.io.*;
import java.util.*;
public class JavaWebServer {
   static final String NAME = "JavaWebServer";
   static final String VERSION = "1.0";
//Tạo đối tượng xử lý và lưu giữ thiết lập cấu hình cho máy chủ
   ServerConfiguration config = new
                        ServerConfiguration();
//Tạo đối tượng dùng để xuất ra các thông báo
   Logger log = new Logger();
//Lớp chương trình chính
```

```
  public static void main(String args[]){
   //Tạo đối tượng Webserver
     JavaWebServer server = new JavaWebServer();
  /*
Xử lý tập tin cấu hình nếu có. Tập tin cấu hình sẽ được đưa vào từ
dòng lệnh như sau : JavaWebServer  -CONFIG  webserver.ini
*/
     if(args.length>0)
server.processCommandLine(args);
     server.run();
  }
//Phương thức khởi dựng
  public JavaWebServer() {
  }
/*
Phương thức này xử lý tên tập tin cấu hình được truyền trên dòng
lệnh nếu có
*/
  public void processCommandLine(String[] args) {
    for(int i=0;i<args.length;++i) {
      if(args[i].equals("-CONFIG")) {
       if(i+1<args.length)
  config.processConfigurationFile(args[i+1]);
       else System.out.println("Configuration file
                argument is missing.");
       break;
```

```
        }
      }
      String logFile = config.getLogFile();
      if(logFile!="") log = new
            Logger(logFile,config.EchoToConsole());
    }
//Cho biết thông tin về chương trình
    public void displayVersionInfo(){
      System.out.println("JavaWebServer version "+
                              VERSION);
    }
/*
Phương thức này thật sự khởi động và chạy chương trình
JavaWebServer
*/
    public void run() {
      displayVersionInfo();
      config.display();
      try {
//Tạo đối tượng socket tại máy chủ
        ServerSocket server = new
              ServerSocket(config.getServerPort());
        int localPort = server.getLocalPort();
        log.writeDateMsg(NAME+" is listening on port"
                        +localPort+".");
```

```
//Vòng lặp liên tục lắng nghe các kết nối từ phía máy khách
        do {
//Chờ đợi kết nối
            Socket client = server.accept();
//Có kết nối xảy ra, chuyển giao cho một phân tuyến xử lý
            (new JavaWebServerThread(
                    client,config,log)).start();
        } while(true);
    } catch(IOException e) {
        log.writeDateMsg("Unable to listen on "+
                    config.getServerPort()+".");
        System.exit(0);
    }
  }
}
```

Biên dịch chương trình:

```
C:\Network\Webserver>javac JavaWebServer.java
```

9.2. Chạy và thử nghiệm chương trình JavaWebServer

Như vậy là chương trình Webserver của ta đã hoàn tất. Trước khi chạy chương trình hãy tạo tập tin `index.htm` (lưu vào thư mục `C:\Network\Webserver`) làm trang chủ với nội dung như sau:

```
<html>
<body>
  <p> Hello from Java WebServer </p>
</body>
```

```
</html>
```

Để thử nghiệm hãy tạo thêm một trang Web khác là `welcome.htm` (cũng lưu vào thư mục `C:\Network\Webserver`) với nội dung là:

```
<html>
<body>
 <B>Welcome to JavaWebServer ! </B><br>
 Thanks for using my Website.
</body>
</html>
```

Chạy chương trình Webserver như sau:

```
C:\Network\Webserver>java JavaWebServer
JavaWebServer version 1.0
 serverPort: 80
 serverRoot:
 defaultFile: index.htm
 logFile:
 EchoLogToConsole: true
Tue Mar 30 09:15:42 PST 1999 JavaWebServer is
listening
on port 80.
```

Chương trình của ta xuất hiện một chuỗi các thông báo và dừng lại chờ các kết nối từ phía máy khách.

Đọc trang Web bằng chương trình Java

Hãy mở một cửa sổ dòng lệnh khác của DOS và dùng lại chương trình `ReadWebPage` mà ta đã thiết kế ở phần đầu, gõ vào dòng lệnh như sau (có thể thay tên `my.testing.server` bằng địa chỉ máy cục bộ là `http://127.0.0.1`):

```
C:\Network>java ReadWebPage
http://my.testing.server
</html>
```

```
<body>
<p>Hello from Java WebServer </p>
</body>
</html>
```

Kết quả là nội dung trang chủ index.htm được trả về. Lúc này nếu quan sát trên cửa sổ chương trình JavaWebServer sẽ thấy các thông tin xuất ra như sau:

```
Tue Mar 30 09:58:16 PST 1999 Accepted connection to
  my.testing.server (127.0.0.1) on port 1191.
Tue Mar 30 09:58:17 PST 1999 Received the following
request:
 GET / HTTP/1.0
 User-Agent: JavaJDK1.2
 Host: my.testing.server
 Accept: text/html, image/gif, image/jpeg, *;
                                    q=.2, */*; q=.2
 Connection: keep-alive
Tue Mar 30 09:58:17 PST 1999 File sent:
C:\Network\index.htm
 Number of bytes: 202
Tue Mar 30 09:58:17 PST 1999 Request completed.
Closing connection.
```

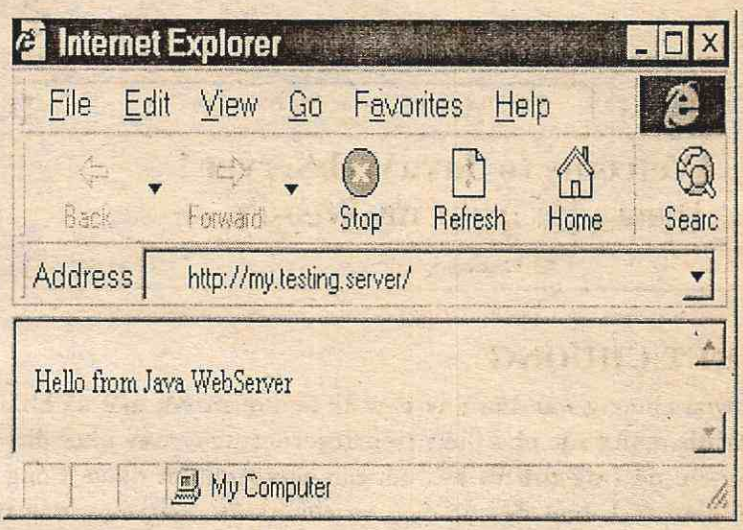
Hãy thử dùng thêm các địa chỉ định vị URL sau để lấy về nội dung các trang Web khác:

```
C:\Network>java ReadWebPage
      http://my.testing.server/index.htm
C:\Network>java ReadWebPage
                http://my.testing.server/welcome.htm
```

Đọc trang Web bằng trình duyệt Internet Explorer

Bộ các thẻ (tag) .html chỉ đưa ra kết quả sau khi đã định dạng. Chương trình Webserver của ta có thể trả về trang Web mà trình duyệt có thể đọc được. Thay vì đánh vào địa chỉ trang Web trên

dòng lệnh như đối với chương trình ReadWebPage hãy thử dùng chương trình Internet Explorer và gõ vào địa chỉ máy chủ của ta như hình sau:



Kết quả trả về là lời chào gọn gàng nằm trong trang Web của máy chủ. Không như chương trình thô sơ ReadWebPage trước đây trình bày toàn bộ nội dung tập tin index.html nhận về từ máy chủ, trình duyệt Internet Explore biết cách phân tích và loại welcome.htm từ máy chủ được trình duyệt Internet Explorer lấy về và trình bày nội dung trang Web trong cửa sổ.

Lưu ý : Cửa sổ chương trình JavaWebServer vẫn đang mở. Hãy theo dõi trạng thái mỗi lần Internet Explorer yêu cầu truy cập trang Web từ máy chủ my.testing.server (hoặc dùng địa chỉ IP 127.0.0.1) sẽ thấy được cách mà các chương trình Webserver làm việc.

Nếu đang trên hệ thống mạng cục bộ (Intranet) hãy dùng địa chỉ IP thật sự thay cho địa chỉ IP 127.0.0.1 (các nhà quản trị mạng sẽ cung cấp cho ta địa chỉ này).

10. KẾT CHƯƠNG

Qua chương mở đầu này bạn đã làm quen với một số khái niệm lập trình mạng rất phổ biến trên Internet ngày nay như: địa chỉ IP là địa chỉ để truy tìm và kết nối các máy tính với nhau. Cổng (port) là số hiệu tách biệt để các dịch vụ chạy trên cùng một máy tính có thể liên lạc với các trình khách trên máy khác. DNS là dịch vụ truy tìm địa chỉ IP theo tên. Socket là cơ chế ổ cắm để kết nối giữa ứng dụng trình khách và trình chủ theo giao thức TCP/IP. Bạn cũng đã biết giao thức TCP được dùng chuyển dữ liệu an toàn hơn giao thức UDP thông qua các chương trình ví dụ như EchoServer, ExchangeRate. Chúng ta cũng đã xây dựng một trình chủ WebServer cùng với trình khách ReadWebpage để đọc và trả về trang Web theo giao thức http phổ biến.

Những khái niệm này rất cần thiết và căn bản để bạn cùng chúng tôi tiếp tục tiến vào thế giới lập trình mạng của Java trong các chương sau. Ở chương sau chúng ta bắt đầu đi vào tìm hiểu một đề tài lập trình mới: Lập trình phân tán đối tượng với kỹ thuật RMI của Java.

Chương 2

RMI VÀ KỸ THUẬT LẬP TRÌNH PHÂN TÁN ĐỐI TƯỢNG TRONG JAVA

Tóm tắt chương 2

Lập trình phân tán đối tượng là một trong những vấn đề nóng bỏng của công nghệ phần mềm ngày nay. Chương này sẽ giới thiệu với bạn khái niệm và cách cài đặt các đối tượng phân tán bằng kỹ thuật RMI trong Java.

Các vấn đề chính sẽ được đề cập đến:

- ✓ *Khái niệm về lập trình phân tán đối tượng và kỹ thuật RMI*
- ✓ *Thiết kế ứng dụng phân tán RMI*
- ✓ *Chuyển dữ liệu và tham chiếu đối tượng trên mạng*
- ✓ *Kết nối mạng và vấn đề tường lửa (firewall)*

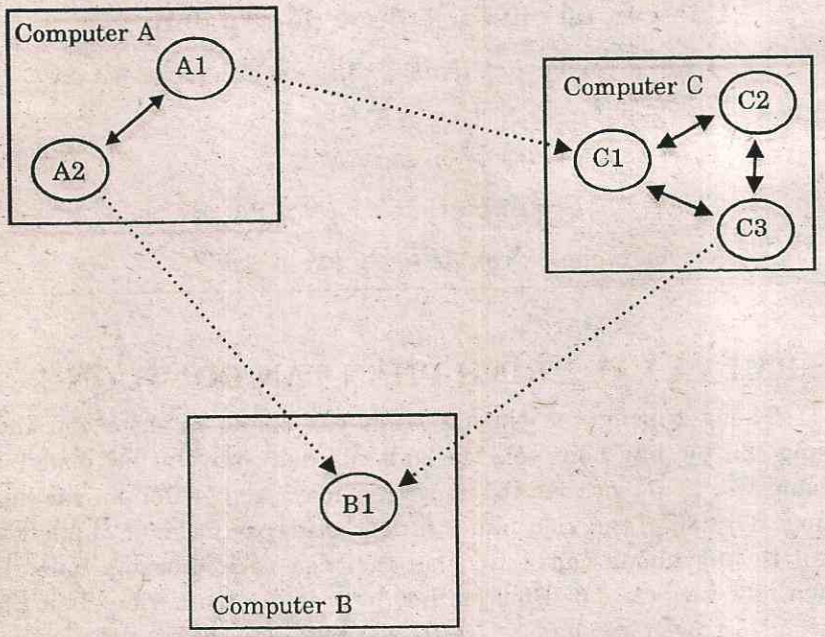
1. RMI VÀ LẬP TRÌNH PHÂN TÁN ĐỐI TƯỢNG

Thông thường các chương trình của chúng ta được viết dưới dạng thủ tục hay hàm gọi. Mã lệnh của hàm hay thủ tục được nạp thẳng vào ký ức và thực thi ngay trên máy cục bộ. Đối với các hàm (hoặc đối tượng) thư viện như `System.out.println()` chẳng hạn bạn thường không cần phải quan tâm đến cách thức mà hàm thư viện này được cài đặt. Điều mà hầu hết các lập trình viên quan tâm là đối số truyền cho hàm và kết quả trả về. Vậy thì có cách nào nạp nội dung của hàm hay đối tượng ở một máy nào đó và gọi chúng từ một máy khác hay không? Đây chính là nội dung của lập trình phân tán mã lệnh. RMI (Remote Method Invoke) - mang ý nghĩa triệu gọi phương thức từ xa - là cách thức giao tiếp giữa các đối tượng Java có

mã lệnh cài đặt (bao gồm phương thức và thuộc tính) nằm trên các máy khác nhau có thể triệu gọi lẫn nhau.

Hình 2-1 cho thấy mô hình triệu gọi đối tượng phân tán. Trên máy A các đối tượng A1, A2 gọi phương thức của nhau được gọi là triệu gọi phương thức cục bộ (local method Invoke) đây là cách mà lập trình hướng đối tượng truyền thống vẫn sử dụng. Tương tự, tương tác giữa C1, C2, C3 là tương tác cục bộ.

Tuy nhiên, đối tượng Java có thể gọi phương thức của đối tượng nằm trên một máy khác dựa vào giao thức triệu gọi từ xa RMI (như A1 gọi C1, A2 gọi B1, C3 gọi B1).



Hình 2-1 Mô hình triệu gọi các đối tượng từ xa

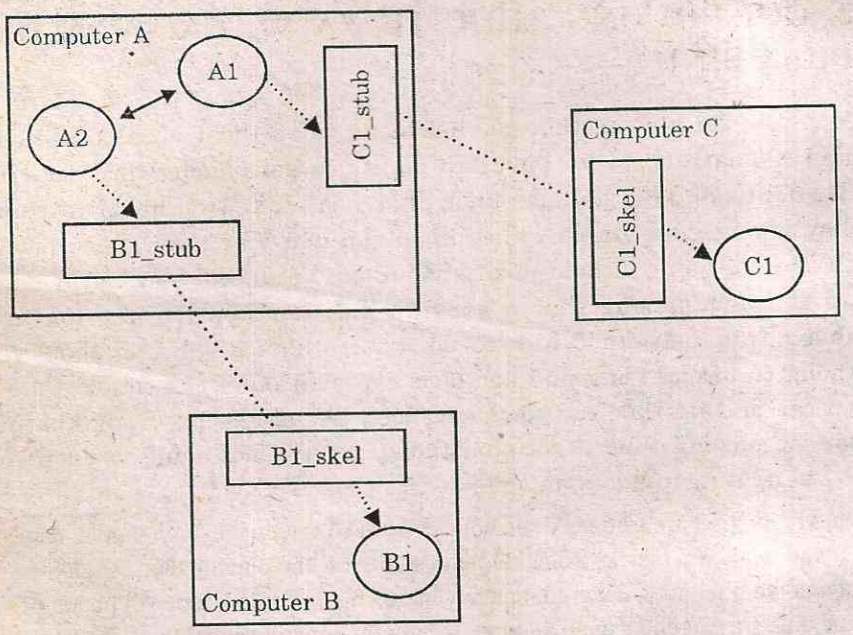
2. GỌI PHƯƠNG THỨC TỪ XA VÀ CÁC VẤN ĐỀ PHÁT SINH

Việc gọi phương thức của đối tượng từ xa thoạt nhìn có vẻ đơn giản nhưng thực tế lại phức tạp hơn triệu gọi phương thức cục bộ. Các đối tượng trên hai máy khác nhau hoạt động trên hai tiến trình (hay không gian địa chỉ) khác nhau cho nên việc tham chiếu biến, địa chỉ đối tượng là hoàn toàn khác nhau. Ví dụ, khi bạn truyền con trỏ cho một phương thức ở xa, trên máy bạn con trỏ này tồn tại nhưng trên máy khác (nơi có đối tượng thực thi phương thức) sẽ không có bất kỳ vùng nhớ nào được cấp phát dành cho con trỏ. Lời gọi các phương thức cục bộ thường luôn trả về kết quả trong khi lời gọi các phương thức từ xa phải thông qua kết nối mạng và luôn có thể bị ngắt ngang do mạng gặp sự cố.

Đối với lời gọi hàm trên máy cục bộ, các tham số truyền cho hàm thường được đặt vào ngăn xếp (stack) trong khi tham số truyền cho phương thức của đối tượng ở xa phải được đóng gói và chuyển qua mạng để đến được với phương thức thực sự. Cơ chế truyền và quản lý biến dữ liệu giữa các tiến trình trên hai máy khác nhau thường được gọi là mashalling (Chúng ta sẽ nghiên cứu về cơ chế mashalling ở mục 7).

3. VAI TRÒ CÁC LỚP TRUNG GIAN (STUB VÀ SKELETION)

Để giải quyết các vấn đề trên, đối tượng Java trên hai máy khác nhau không gọi nhau trực tiếp mà thông qua lớp trung gian. Lớp trung gian này tồn tại ở cả hai phía: máy khách (nơi gọi phương thức của đối tượng ở xa) và máy chủ (nơi đối tượng thực sự được cài đặt để thực thi mã lệnh của phương thức). Phía máy khách lớp trung gian này được gọi là stub (lớp móc), phía máy chủ lớp trung gian này được gọi là skeletion (lớp nối). Bạn hình dung stub và skeletion là hai người môi giới giúp các đối tượng ở xa giao dịch với nhau.



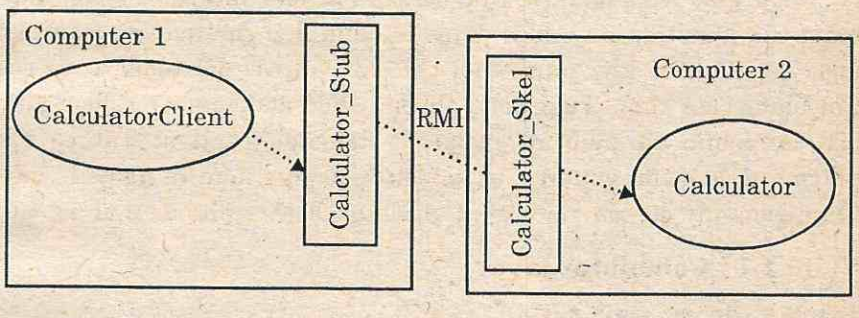
Hình 2-2 Gọi phương thức của các đối tượng thông qua lớp trung gian

Ví dụ bạn có đối tượng C1 được cài đặt chạy trên máy C. Trình biên dịch Java sẽ giúp bạn tạo ra hai lớp trung gian C1_Skel và C1_Stub. Lớp C1_Stub sẽ được đem về máy Computer A. Khi A1 trên máy computer A gọi C1 nó sẽ chuyển lời gọi đến lớp C1_Stub. C1_Stub chịu trách nhiệm đóng gói tham số, chuyển tham số qua mạng đến máy Computer C. Trên máy Computer C lớp nối C1_Skel sẽ nhận tham số chuyển vào vùng không gian địa chỉ tương thích với đối tượng C1 sau đó gọi phương thức tương ứng. Kết quả nếu có do phương thức của đối tượng C1 trả về sẽ được lớp C1_Skel đóng gói trả ngược về cho C1_Stub. C1_Stub chuyển giao kết quả cuối cùng lại cho A1. Bằng cơ chế này, A1 luôn nghĩ rằng nó đang hoạt động trực tiếp với đối tượng C1 ngay trên máy cục bộ. Hơn nữa, nhờ sự giúp đỡ của lớp trung gian C1_Stub, khi kết nối mạng gặp sự cố lớp trung gian stub sẽ luôn biết cách thông báo lỗi đến đối tượng A1.

Thực tế làm cách nào A1 tham chiếu được đến C1, một khi không có lớp C1 nào được cài đặt ở máy A? C1_Stub trên máy A chỉ làm lớp nền trung gian chuyển đổi tham số và thực hiện các giao thức mạng, nó không phải là hình ảnh của đối tượng C1. Để làm được điều này, đối tượng C1 cần cung cấp một giao tiếp interface tương ứng với các phương thức cho phép đối tượng A1 gọi nó trên máy A. Chúng ta sẽ tiếp tục làm sáng tỏ vấn đề hơn trong quá trình xây dựng một ứng dụng RMI thật sự sau đây.

4. CÀI ĐẶT ỨNG DỤNG PHÂN TÁN RMI

Dưới đây là một ví dụ đơn giản giúp bạn dễ dàng hình dung cách làm việc của các đối tượng RMI. Chương trình của chúng ta là một hệ đối tượng phân tán (hình 2-3).



Hình 2-3 Triệu gọi đối tượng RMI giữa trình khách và đối tượng chủ ở xa

Đối tượng Calculator chạy trên máy tính Computer 2 sẽ được gọi bởi đối tượng CalculatorClient chạy trên máy tính Computer 1

Chúng ta xây dựng lớp đối tượng Calculator với đặc tả interface như sau:

```
public interface Calculator {
  public int addNumber(int x, int y);
}
```

Phương thức `addNum()` được dùng để cộng hai tham số x, y với nhau và trả về kết quả cho đối tượng nào gọi nó. Chúng ta muốn Calculator có khả năng giao tiếp được với các đối tượng ở xa. Muốn vậy bạn phải báo cho Java biết Calculator là một đối tượng có khả năng Remote. Hơn nữa tất cả các phương thức trong giao tiếp Calculator muốn gọi được từ xa đều phải có khả năng ném ra ngoại lệ RemoteException. Đặc tả giao tiếp interface của đối tượng Calculator được khai báo đầy đủ như sau

```
public interface Calculator extends Remote{
   public int addNumber(int x, int y) throws
                                        RemoteException;
}
```

Lớp giao tiếp Remote thật ra chỉ là một interface rỗng, chức năng của giao tiếp này được dùng để đánh dấu cho máy ảo java biết được đặc tính có khả năng "nói chuyện" với các đối tượng ở xa của Calculator mà thôi. Tiếp đến, phương thức addNumber() được khai báo là public và ném ra ngoại lệ RemoteException. Tất cả các phương thức muốn gọi được từ xa đều buộc phải ném ra ngoại lệ này. Mã nguồn đầy đủ của giao tiếp Calculator được trình bày ở ví dụ sau:

Ví dụ 2-1 Calculator.java

```
import java.rmi.*;
public interface Calculator extends Remote{
   public int addNum(int x, int y) throws
                                        RemoteException;
}
```

Biên dịch từ dòng lệnh :

```
C:\network>javac Calculator.java
```

bạn sẽ được tập tin Calculator.class

Lớp Calculator chỉ là lớp giao tiếp đặc tả các phương thức của đối tượng. Từ lớp giao tiếp, đối tượng thực sự được cài đặt như sau:

Ví dụ 2-2 CalculatorImpl.java

```
import java.rmi.*;
public class CalculatorImpl implements Calculator {
public int addNum(int x, int y) throws
                                        RemoteException{
    System.out.println("Client request to
                                        calculate");
    return (x+y);
  }
}
```

Biên dịch từ dòng lệnh chúng ta sẽ thu được tập tin CalculatorImpl.class

```
C:\network>javac Calculator Impl.java
```

Như vậy bạn đã có được đối tượng Calculator chứa phương thức addNum() gọi được từ một máy khác. Đối tượng Calculator bao gồm 2 lớp, lớp CalculatorImpl.class chứa cài đặt của đối tượng. Và lớp Calculator.class chứa đặc tả để giao tiếp với lớp đối tượng thật sự CalculatorImpl.class.

Vấn đề đặt ra là làm thế nào để cài đặt đối tượng Calculator lên một máy (máy Computer 2) và gọi phương thức addNum() của Calculator từ máy khác (máy Computer 1)? Như đã nêu, chúng ta không gọi được phương thức addNum() của Calculator trực tiếp mà cần có thêm hai lớp giao dịch trung gian là Stub và Skel. Dựa vào lớp cài đặt CalculatorImpl.class, trình biên dịch `rmic.exe` của Java sẽ cung cấp cho bạn hai lớp trung gian Stub và Skel. Bạn mở cửa sổ DOS-Prompt và đánh lệnh biên dịch sau:

```
C:\network>javac CalculatorImpl.class
```

Kết quả bạn sẽ thu được hai tập tin CalculatorImpl_Stub.class và CalculatorImpl_Skel.class

Bây giờ là bước phải viết một chương trình để cài đặt đối tượng Calculator trên một máy - ta gọi chương trình này là CalculatorServer. Mã nguồn của CalculatorServer được trình bày trong ví dụ sau:

Ví dụ 2-3 CalculatorServer.java

```
import java.rmi.server.*;
import java.rmi.*;
public class CalculatorServer{
  public static void main(String args[]) {
    try{
      //Tạo đối tượng Calculator thật sự
      CalculatorImpl c=new CalculatorImpl();
     System.out.println("Exporting Calculator ...");
      //Thông báo sự hiện diện của đối tượng cho máy ảo Java
      UnicastRemoteObject.exportObject(c);
      //đăng ký đối tượng với bộ quản lý rmi
    Naming.bind("rmi://localhost/MyCalculator", c);
    System.out.println("Register Calculator!");
    }catch (Exception e){
       System.out.println(e);
    }
  }
}
```

Biên dịch chương trình từ dòng lệnh

```
C:\network>javac CalculatorServer.java
```

Chúng ta hãy tìm hiểu cách chương trình CalculatorServer làm việc. Trước hết đối tượng CalculatorImpl được tạo ra

```
CalculatorImpl c=new CalculatorImpl();
```

Bạn có thể dùng chính lớp giao tiếp Calculator để nhận đối tượng trả về từ CalculatorImpl. Lệnh sau là hoàn toàn tương đương

```
Calculator c=new CalculatorImpl();
```

Tiếp đến CalculatorServer gọi phương thức tĩnh exportObject() của lớp UnicastRemoteObject để máy ảo Java nhận diện được đối tượng c (Calculator) là đối tượng có khả năng truy xuất từ xa.

```
UnicastRemoteObject.exportObject(c);
```

Lưu ý để sử dụng được lớp UnicastRemoteObject bạn phải khai báo gói java.rmi.server ở đầu chương trình như sau

```
import java.rmi.server.*;
```

Bước cuối cùng là đặt cho đối tượng Calculator một tên gợi nhớ và đăng ký tên này với bộ quản lý rmi. Phương thức tĩnh bind() của lớp Naming sẽ thực hiện điều này

```
Naming.bind("rmi://localhost/MyCalculator", c);
```

Phương thức bind() nhận hai đối số: đối số thứ nhất là chuỗi định vị URL. Đối số thứ hai là bản thân đối tượng cần đăng ký. Chuỗi định vị URL có định dạng như sau

```
rmi://hostname:port/ObjectName
```

rmi là tên giao thức dùng để đăng ký. hostname, port là địa chỉ IP và số hiệu cổng của máy chủ nơi bộ đăng ký đối tượng rmi đang chạy. ObjectName là tên gợi nhớ bất kỳ mà bạn muốn đặt cho đối tượng. Các chương trình phía máy khách sẽ dựa vào tên này để truy tìm tham chiếu đến đối tượng cần dùng.

Bạn có thể tự hỏi bộ đăng ký này nằm ở đâu? Vâng, bạn hãy mở một cửa sổ DOS-Promt khác chạy chương trình rmiregistry từ dòng lệnh như sau:

C:\Network>rmiregistry.exe

Chương trình này có thể không phản hồi cho bạn thông tin gì cả bởi vì đây là chương trình chạy ở dạng service. Nó có nhiệm vụ đón nhận các kết nối chứa thông tin về đối tượng do phương thức Naming.bind() gửi đến. Tốt nhất là sau khi khởi động rmiregistry.exe trong cửa sổ DOS-promt bạn hãy thu nhỏ cửa sổ này trên thanh taskbar của Windows. Tuy không nói năng gì nhưng bộ quản lý đối tượng này luôn trong trạng thái sẵn sàng phục vụ bạn.

Mặc định rmiregistry.exe lắng nghe các kết nối gửi đến cổng 1099. Bạn có thể chỉ định số hiệu cổng khác cho rmiregistry. Ví dụ lệnh sau sẽ chạy rmiregistry.exe trên cổng 2001

C:\Network>rmiregistry.exe 2001

Tùy theo phiên bản Java Bạn có thể tìm thấy chương trình rmiregistry.exe trong thư mục \jdk1.3\bin

Chúng ta đặt cho đối tượng Calculator tên gợi nhớ là MyCalculator. Địa chỉ IP được chỉ định là máy cục bộ localhost (hay 127.0.0.1). Nếu không chỉ định số hiệu cổng, mặc định Naming.bind() sẽ tìm cách liên lạc với bộ đăng ký rmiregistry thông qua cổng 1099. Ví dụ, nếu bạn đã chạy chương trình rmiregistry trên cổng 2001, Naming.bind() sẽ đăng ký đối tượng như sau:

```
Naming.bind("rmi://localhost:2001/MyCalculator", c);
```

Chạy chương trình CalculatorServer từ cửa sổ DOS-Prompt như sau:

```
C:\Network>java CalculatorServer
Exporting Calculator ...
Register Calculator!
```

Cũng như rmiregistry.exe sau khi CalculatorServer gọi hàm Naming.bind() nó sẽ đi vào vòng lặp vô tận chờ nhận yêu cầu gửi đến từ phía máy khách. Hiện giờ bạn đã có hai chương trình đang hoạt động phía hậu trường: một là chương trình rmiregistry và một

là CalculatorServer. Mỗi chương trình chạy trên một cửa sổ DOS-Prompt độc lập.

Đã đến lúc thiết kế chương trình phía máy khách có khả năng gọi và sử dụng đối tượng Calculator trên máy chủ. Chương trình CalculatorClient của ta phía máy khách được cài đặt như sau:

Ví dụ 2-4 CalculatorClient.java

```
import java.rmi.*;
public class CalculatorClient{
  public static void main(String args[]){
   try{
     //Tìm đối tượng cần truy xuất theo tên
    System.out.println("Finding object ...");
    Calculator c=(Calculator)Naming.lookup(
                    "rmi://localhost/MyCalculator");
     //Gọi phương thức của đối tượng
    System.out.println(c.addNum(12,1));
  }catch (Exception e){
    System.out.println(e);
  }
 }
}
```

Để truy tìm đối tượng ở xa, chương trình máy khách gọi phương thức tĩnh Naming.lookup(). Phương thức này chỉ yêu cầu đối số là chuỗi định dạng cho biết địa chỉ máy chủ và tên đăng ký của đối tượng.

```
Calculator c=(Calculator)Naming.lookup(
                    "rmi://localhost/MyCalculator");
```

Ở đây địa chỉ máy chủ là localhost và tên của đối tượng là MyCalculator. Trường hợp một nhà cung cấp nào đó cho phép bạn sử dụng đối tượng của họ, ho cũng phải cho bạn biết tên đối tượng và địa chỉ máy chủ nơi đối tượng tồn tại. Ví dụ:

```
VendorObject =(VendorObject)Naming.lookup(
            "rmi://www.vendorAddress.com/SomeObject");
```

Bạn có thể hình dung lớp Naming đóng vai trò như một dịch vụ tìm kiếm. Đối tượng ở máy chủ nhờ dịch vụ Naming đăng ký tên giao dịch bằng phương thức `Naming.bind()`. Tiếp đến bất kỳ khách hàng nào muốn quan hệ với đối tượng chỉ cần cung cấp tên đối tượng và địa chỉ nơi đối tượng hiện hữu bằng cách gọi hàm `Naming.lookup()`. Dịch vụ Naming sẽ tìm cách liên lạc với đối tượng chủ và trả về tham chiếu của đối tượng để bạn sử dụng.

Điểm lưu ý đó là, bản thân đối tượng trên máy chủ được cài đặt bằng lớp CalculatorImpl.class trong khi để sử dụng đối tượng trên máy khách bạn chỉ cần dùng lớp giao tiếp Calculator.class mà thôi. Calculator.class chính là hình ảnh của CalculatorImpl.class. Và như bạn biết do Calculator.class là một interface cho nên nó chỉ chứa đựng các đặc tả của phương thức gọi, không có bất kỳ cài đặt nào bên trong Calculator.class. Lớp Calculator.class rất nhỏ gọn trong khi lớp CalculatorImpl.class có thể có kích thước rất lớn. Chính vì lý do này mà CalculatorImpl.class mới cần phải đặt chạy trên máy chủ (server) có tốc độ cao và khả năng xử lý mạnh. Bạn có thể tham chiếu đến đối tượng ở xa theo cách

```
CalculatorImpl c=(CalculatorImpl)Naming.lookup(
            "rmi://localhost/MyCalculator");
```

Tuy nhiên thực tế cách này sẽ không bao giờ được áp dụng bởi vì nó yêu cầu máy khách phải chứa lớp CalculatorImpl.class mà như vậy có nghĩa là tất cả mã lệnh của đối tượng đều tập trung trên một máy cục bộ (máy khách), trái với mục đích của ý tưởng lập trình phân tán.

Nếu bạn thử nghiệm trên một máy bằng địa chỉ cục bộ localhost bạn có thể có cả hai lớp Calculator.class và CalculatorImpl cùng tồn tại. Nhưng thực tế nếu bạn thử nghiệm trên mô hình mạng 2 máy client/server thật sự, máy server sẽ cần đến lớp CalculatorImpl.class, CalculatorImp_Skel.class để cài đặt đối tượng. Máy khách client chỉ cần dùng đến lớp Calculator.class và CalculatorImpl_Stub.class để gọi phương thức của đối tượng.

Biên dịch chương trình từ dòng lệnh

```
C:\Network>javac CalculatorClient.java
```

Tất cả đều sẵn sàng, bạn hãy mở một cửa sổ DOS-Prompt khác, chạy chương trình CalculatorClient như sau

```
C:\Network>java CalculatorClient
Finding object ...
13
```

Kết quả trả về là tổng của 12+1 chính là lời gọi đến phương thức `addNum()` của đối tượng CalculatorImpl. Theo dõi cửa sổ DOS-Prompt nơi chương trình CalcultorServer đang chạy bạn sẽ thấy kết xuất sau:

```
Client request to calculate
```

Thông điệp này cho thấy hàm `addNum()` được gọi thực thi từ một máy ảo Java trên máy chủ trước khi trả kết quả tính toán về cho máy khách (đang chạy trên một máy ảo Java khác).

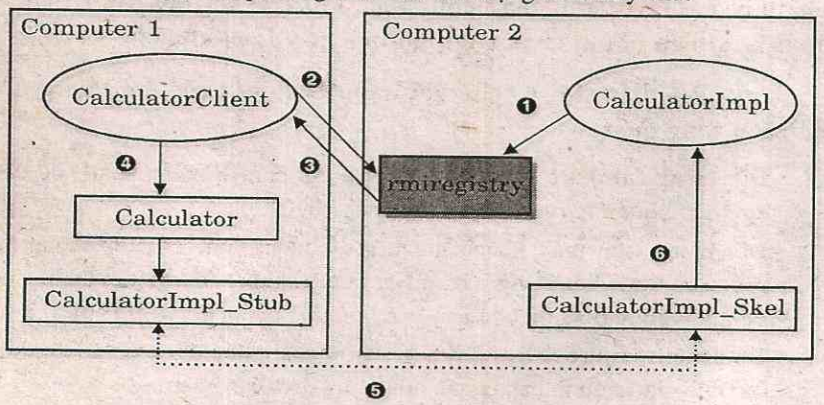
Hình 2-4 diễn đạt cơ chế gọi hàm từ xa của các đối tượng RMI một cách tổng quát:

1. Đối tượng cài đặt các phương thức và gọi hàm Naming.bind() để đăng ký với bộ quản lý rmiregistry trên máy chủ.
2. Đối tượng trên máy khách muốn gọi phương thức của đối tượng trên máy chủ trước hết cần gọi hàm Naming.lookup() để truy tìm tham chiếu đến đối tượng ở xa theo tên.
3. Bộ đăng ký rmiregistry sẽ trả về tham chiếu đến đối tượng ở xa thông qua lớp giao tiếp interface mà đối tượng ở xa cung cấp.

4. Dựa vào lớp giao tiếp interface đối tượng ở máy khách sẽ gọi các phương thức của đối tượng trên máy chủ
5. Khi một phương thức được gọi, lời gọi sẽ được chuyển tiếp đến lớp trung gian _Stub. Xử lý chuyển tham số của phương thức gọi đến lớp _Skel trên máy chủ.
6. Lớp trung gian _Skel trên máy chủ sẽ trực tiếp yêu cầu đối tượng thực thi phương thức và chuyển trả kết quả về cho máy khách.

Các bước cài đặt và thiết kế đối tượng RMI có thể được tóm tắt như sau:

7. Đặc tả lớp giao tiếp của đối tượng(Calculator.java)
8. Dựa vào lớp đặc tả cài đặt chi tiết đối tượng (CalculatorImpl.java)
9. Biên dịch đối tượng (CalculatorImpl.class). Dựa vào đối tượng vừa cài đặt dùng trình rmic.exe (chương trình này nằm trong thư mục \jdk1.3\bin) tạo ra hai lớp trung gian CalculatorImpl_Stub và CalculatorImpl_Skel.
10. Viết chương trình (CalculatorServer.class) cài đặt đối tượng trên máy chủ. Lưu ý bạn phải gọi phương thức UnicastRemoteObject.exportObject() để thông báo cho Java nhận dạng được sự tồn tại của đối tượng ở dạng truy xuất được từ xa bởi các đối tượng khác.
11. Sử dụng lớp giao tiếp (Calculator.class) và CalculatorImpl_Stub từ máy khách để gọi các phương thức của đối tượng trên máy chủ.



Hình 2-4 Cơ chế RMI gọi phương thức của đối tượng từ xa

5. BỘ ĐĂNG KÝ (REGISTRY), CLASSPATH VÀ CODEBASE

Với ví dụ trên bạn đã biết đến bộ đăng ký rmiregistry dùng để tiếp nhận đăng ký tên của đối tượng. Hầu như trước khi máy khách muốn sử dụng được các phương thức và gọi được đối tượng ở xa nó phải liên lạc với rmiregistry để lấy về tham chiếu đến đối tượng trên máy chủ. Dịch vụ đăng ký và truy tìm tên đối tượng (các bước ❶, ❷, ❸ ở hình 2-4) được Java quản lý bằng các hàm giao tiếp API JNDI (Java Naming Directory Interface). Các hàm JNDI ở máy khách (Naming.lookup()) sẽ liên lạc với rmiregistry để nhận về tham chiếu của đối tượng trong khi các hàm JNDI ở máy chủ (Naming.bind()) có nhiệm vụ đăng ký tên của đối tượng với rmiregistry.

Lưu ý: hàm Naming.bind() chỉ đăng ký tên của đối tượng với bộ quản lý rmiregistry đúng một lần. Một khi tên đăng ký đã tồn tại trước đó bạn sẽ nhận được thông báo lỗi AlreadyBoundException. Trường hợp bạn sửa đổi và muốn đăng ký lại đối tượng bạn cần chấm dứt chương trình rmiregistry từ cửa sổ DOS-Promt (nhấn CTRL-C) và chạy lại chương trình này để rmiregistry tiếp nhận lại thông tin mới. Tuy nhiên bạn có thể thay thế hàm Naming.bind() bằng hàm Naming.rebind(). Khi dùng hàm Naming.rebind() bạn không cần phải chấm dứt và khởi động lại rmiregistry, hàm này có nhiệm vụ yêu cầu đăng ký mới đối tượng RMI với rmiregistry nếu đối tượng này chưa có và đăng ký lại nếu đối tượng này đã được đăng ký trước đó.

Mục đích của các ứng dụng RMI là lập trình phân tán đối tượng trên mô hình kết nối mạng. Cách tốt nhất là bạn chạy chương trình CalculatorServer để đăng ký đối tượng CalculatorImpl trên một máy và chạy chương trình CalculatorClient trên một máy khác. Tuy nhiên không phải lúc nào bạn cũng có được điều kiện sở hữu cùng lúc hai hay nhiều máy để kiểm tra. Mặc dù vậy, máy ảo Java chính là mô hình lý tưởng để bạn mô phỏng cách làm việc của mạng.

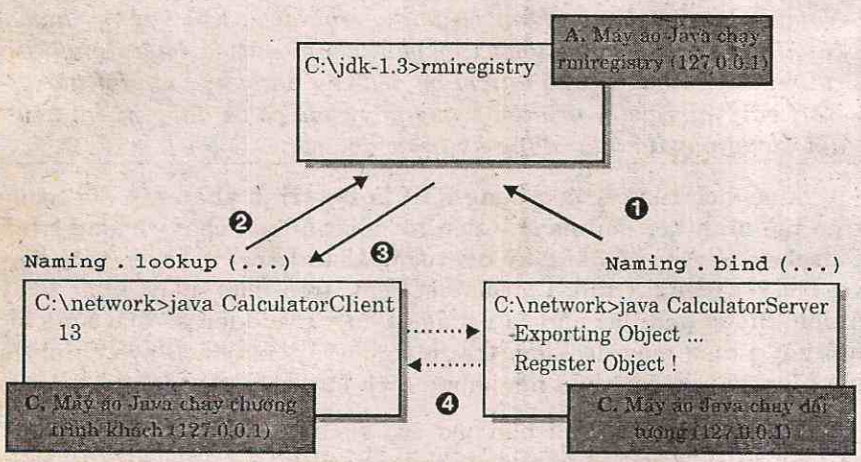
Trong Windows khi bạn mở cửa sổ DOS-Prompt để chạy một chương trình Java, về lý thuyết chương trình này được xem là chạy độc lập trên một máy (máy ảo). Do đó nếu 2 chương trình Java chạy trên 2 cửa sổ DOS-Prompt khác nhau bạn có thể xem đây là kết quả

kết xuất từ hai máy tính khác nhau. Hay nói cách khác dựa vào máy ảo Java bạn có thể tùy thích tạo ra các máy tính ảo để giao tiếp với nhau giả lập mạng. Dữ liệu chuyển từ máy ảo Java này sang máy ảo Java khác hoàn toàn tương tự quá trình giao tiếp và chuyển dữ liệu giữa hai máy tính kết nối vật lý thật sự trên mạng.

Tuy nhiên cũng có một số hạn chế đó là tất cả các máy ảo Java đều dùng chung một địa chỉ IP vật lý trong khi hai máy ảo Java chạy trên hai máy tính vật lý khác nhau sẽ có địa chỉ IP khác nhau. Các máy ảo Java chạy trên cùng một máy vật lý sẽ sử dụng cùng thư viện Java, cùng một chính sách bảo mật, điều này làm bạn rất dễ nhầm tưởng là chương trình chạy tốt trên cả hai phía máy khách và máy chủ. Trong khi thực tế nếu máy khách và máy chủ dùng hai thư viện Java (cũng như biến môi trường CLASSPATH) khác nhau hoặc chính sách bảo mật khác nhau sẽ làm cho kết nối từ phía máy khách và máy chủ không thực hiện được. Bạn nên lưu ý đến vấn đề này khi cài đặt các ứng dụng Java trên các máy mạng vật lý thật sự.

5.1. Đăng ký rmiregistry

Hình 2-5 trình bày cơ chế làm việc của trình khách và trình chủ trên máy ảo Java khi đăng ký và truy xuất rmiregistry.



Hình 2-5 Cơ chế làm việc của các máy ảo Java khi đăng ký và gọi đối tượng giữa trình khách và trình chủ

❶ Đối tượng trên máy ảo Java B dùng hàm Naming.bind() hoặc Naming.rebind() để đăng ký đối tượng với chương trình rmiregistry đang chạy trên máy ảo Java A.

❷ Chương trình khách chạy trên máy ảo Java C dùng hàm Naming.lookup() để yêu cầu bộ quản lý rmiregistry trên máy ảo A trả về tham chiếu đến đối tượng.

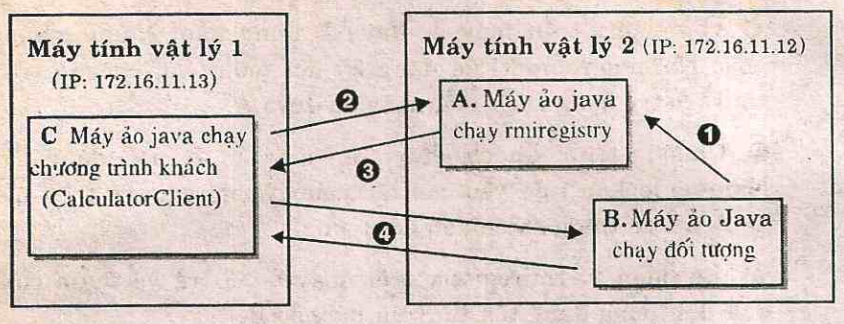
❸ Bộ quản lý rmiregistry trên máy ảo A trả về tham chiếu đến đối tượng đang tồn tại trên máy ảo B.

❹ Chương trình khách chạy trên máy ảo Java C sử dụng tham chiếu do hàm Naming.lookup() trả về để truy và gọi phương thức của đối tượng đang chạy trên máy ảo B từ xa.

Như đã nêu, cả 3 máy ảo này đều có thể xem như 3 máy vật lý kết nối trực tiếp với nhau qua mô hình mạng. Tuy nhiên, nếu 3 máy ảo này chạy trên 3 máy khác nhau chúng sẽ khác nhau số địa chỉ IP. Do chạy trên cùng một máy vật lý 3 nên cả máy ảo này sẽ mang cùng một địa chỉ IP cục bộ 127.0.0.1 (hay localhost).

Thực tế với mô hình trên, nếu hoạt động thật sự qua mạng thì chỉ có thể tách ra làm hai máy vật lý chứ không phải là 3. Máy ảo A chạy chương trình rmiregistry và máy ảo B chạy phần cài đặt của đối tượng phải chạy trên cùng một máy vật lý. Tức là máy ảo A và B phải trùng địa chỉ IP của nhau. Hiện tại Java không cho phép rmiregistry chạy trên máy vật lý khác với nơi mà đối tượng rmi đang hoạt động.

Hình 2-6 là cơ chế hoạt động của các máy ảo Java trên mô hình máy mạng vật lý thật sự. Máy khách có địa chỉ mạng IP là 172.16.11.13 truy xuất đến bộ đăng ký rmiregistry của máy chủ có địa chỉ IP là 172.16.11.12. Trên máy chủ 172.16.11.12 chúng ta có cả 2 máy ảo Java cùng hoạt động. Máy ảo Java A chạy chương trình rmiregistry trong khi đó máy ảo Java B chạy phần cài đặt của đối tượng.



Hình 2-6 Cơ chế hoạt động của các máy ảo Java trên mô hình máy mạng vật lý thật sự

Với địa chỉ IP thật sự (khác địa chỉ cục bộ localhost 127.0.0.1) hàm đăng ký đối tượng với rmiregistry từ máy chủ phải được gọi như sau:

```
Naming.bind("rmi://172.16.11.12/MyCalculator", c);
```

Địa chỉ máy chủ nơi rmiregistry đang chạy

Và hàm truy tìm đối tượng từ máy khách sẽ được gọi như sau

```
Calculator c=(Calculator)Naming.lookup("
        rmi://172.16.11.12/MyCalculator");
```

Địa chỉ máy chủ nơi rmiregistry đang chạy

5.2. Biến CLASSPATH và tùy chọn CODEBASE

Bạn đã biết khi chạy chương trình Java dựa vào biến môi trường CLASSPATH để truy tìm các tập tin .class. Chương trình minh hoạ Calculator của ta chứa trong thư mục C:\Network bao gồm các tập tin .class sau :

Tên lớp	Mục đích	Sử dụng trên
Calculator.class	Lớp đặc tả các phương thức giao tiếp của đối tượng.	Máy khách và máy chủ
CalculatorImpl.class	Phần cài đặt của bản thân đối tượng.	Máy chủ
CalculatorImpl_Skel.class	Lớp giao tiếp trung gian do trình biên dịch rmic.exe tạo ra	Máy chủ
CalculatorImpl_Stub.class	Lớp giao tiếp trung gian do trình biên dịch rmic tạo ra	Máy khách và máy chủ
CalculatorServer.class	Chương trình dùng chạy trên máy chủ dùng tạo ra đối tượng và đăng ký đối tượng với rmiregistry.	Máy chủ
CalculatorClient.class	Chương trình chạy trên máy khách gọi các phương thức của đối tượng trên máy chủ.	Máy khách

Như vậy trên máy chủ server bạn cần phải có các lớp sau:

Calculator.class

CalculatorImpl.class

CalculatorImpl_Skel.class

CalculatorImpl_Stub.class

CalculatorServer.class

Khi CalculatorServer yêu cầu rmiregistry đăng ký tên đối tượng rmiregistry. sẽ phải đi truy tìm lớp trung gian CalculatorImpl_Stub.class. Trên máy ảo nơi rmiregistry đang chạy nếu biến CLASSPATH không trỏ đến đường dẫn nơi chứa tập tin này bạn sẽ nhận được thông báo lỗi trả về trên máy ảo nơi CalculatorServer yêu cầu đăng ký đối tượng.

```
java.lang.ClassNotFoundException:
CalculatorImpl_Stub
```

Trên máy khách client bạn chỉ cần các lớp:

Calculator.class

CalculatorImpl_Stub.class

CalculatorClient.class

trong đó CalculatorClient.class là do bạn viết, còn Calculator.class và CalculatorImpl_Stub.class là do nhà cung cấp (hay người thiết kế ra đối tượng) gửi đến bạn.

Thật sự thì khi viết chương trình từ phía máy khách lập trình viên chỉ cần đến lớp giao tiếp Calculator.class. Lớp trung gian CalculatorImpl_Stub.class không có ý nghĩa đối với các nhà phát triển ứng dụng. Nó chỉ cần thiết cho cơ chế RMI của Java. Chính vì điều này Java cung cấp cho bạn cách thức tự động nạp lớp CalculatorImpl_Stub.class từ xa thông qua tùy chọn CODEBASE khi đăng ký đối tượng với rmiregistry trên máy chủ. Bạn chạy chương trình CalculatorServer vào lúc đăng ký như sau:

```
C:\Network>java -Djava.rmi.server.codebase="
        http://172.16.11.12/myclass/" CaculatorServer
```

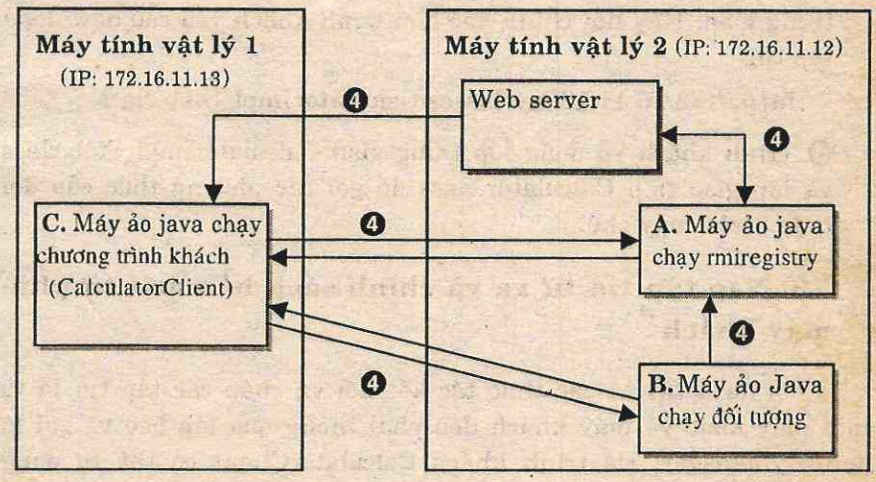
Máy chủ của bạn phải hỗ trợ thêm dịch vụ web server hay nói cách khác bạn phải có chương trình web server chạy chung trên máy vật lý nơi rmiregistry đang chạy.

Bạn chép tập tin CalculatorImpl_Stub.class vào thư mục /myclass của trình chủ web server và lưu ý địa chỉ codebase phải kết thúc bằng dấu sổ trái ("/").

Khi bạn sử dụng tùy chọn CODEBASE thì biến CLASSPATH trên máy ảo Java nơi rmiregistry đang chạy không được phép trỏ đến cùng thư mục chứa CalculatorImpl_Stub.class. Nếu không rmiregistry sẽ luôn ưu tiên lấy lớp CalculatorImpl_Stub.class từ đường dẫn CLASSPATH mà bỏ qua giá trị tùy chọn CODEBASE do trình đăng ký đối tượng CalculatorServer yêu cầu. Đây là một trong những lỗi mà lập trình viên Java vẫn thường gặp phải.

Khi máy khách có yêu cầu rmiregistry trả về tham chiếu của đối tượng, nếu máy khách chưa có lớp CalculatorImpl_Stub.class, rmiregistry sẽ hướng dẫn máy khách tự động nạp lớp này từ địa chỉ codebase http://172.16.11.12/myclasses/.

Bằng cách này nhà cung cấp đối tượng chỉ cần chuyển giao cho các nhà phát triển ứng dụng phía máy khách lớp giao tiếp Calculator.class là đủ. Lớp CalculatorImpl_Stub.class sẽ được tự động nạp về máy khách khi trình khách yêu cầu rmi tham chiếu đến đối tượng.



Hình 2-7. Cơ chế nạp tự động lớp trung gian Calculator_Stub.class xuống máy khách

❶ Trình chủ CalculatorServer đăng ký đối tượng với rmiregistry. CalculatorServer yêu cầu bộ quản lý rmiregistry truy tìm lớp CalculatorImpl_Stub.class ở địa chỉ URL http://172.16.11.12/myclasses/

```
C:\Network>java -Djava.rmi.server.codebase="
    http://172.16.11.12/myclass/" CaculatorServer
```

❷ rmiregistry nhận được yêu cầu đăng ký đối tượng nó sẽ truy tìm lớp Calculator_Stub.class trong biến CLASSPATH. Nếu không thấy, rmiregistry sẽ dựa vào chuỗi URL do trình CalculatorServer cung cấp yêu cầu Web server trả về lớp trung gian stub này. Rmiregistry ghi nhớ lại địa chỉ URL nơi lớp trung gian CalculatorImpl_Stub.class đang được lưu trữ.

❸ Chương trình ở máy khách yêu cầu rmiregistry trả về tham chiếu của đối tượng. Nếu phía máy khách chưa tồn tại lớp CalculatorImpl_Stub.class bộ quản lý rmiregistry sẽ cho trình khách biết địa chỉ URL để chép về lớp trung gian này.

❹ Chương trình ở máy khách yêu cầu Web server cung cấp lớp trung gian. Hay nói chính xác hơn trình khách yêu cầu download tập tin

http://172.16.11.12/myclasses/CalculatorImpl_Stub.class

❺ Trình khách sử dụng lớp trung gian CalculatorImpl_Stub.class và lớp giao tiếp Calculator.class để gọi các phương thức của đối tượng trên máy chủ.

5.3. Nạp tập tin từ xa và chính sách bảo mật từ phía máy khách

Với Java tất cả các thao tác kết nối và chép các tập tin lạ từ một máy khác về máy khách đều phải thông qua lớp bảo vệ gọi là securityManager. Để trình khách CalculatorClient có thể tự động nạp lớp CalculatorImpl_Stub.class từ webserver về bạn phải thêm vào dòng lệnh kiến tạo lớp bảo vệ từ phía máy khách như sau:

```
System.setSecurityManager(new RMISecurityManager());
```

Mã nguồn của chương trình CalculatorClient trong trường hợp tự động nạp lớp CalculatorImpl_Stub từ máy chủ sẽ được thiết kế lại như ví dụ 2-5 sau:

Ví dụ 2-5 CalculatorClient.java

```
import java.rmi.*;
public class CalculatorClient{
  public static void main(String args[]){
   try{
     //Thiết lập lớp phòng vệ từ xa
     System.setSecurityManager(new
                                RMISecurityManager());
     //Tìm đối tượng cần truy xuất theo tên
     System.out.println("Finding object ...");
     Calculator c=(Calculator)Naming.lookup(
              "rmi://172.16.11.12/MyCalculator");
     //Gọi phương thức của đối tượng
     System.out.println(c.addNum(12,1));
   }catch (Exception e){
     System.out.println(e);
   }
  }
}
```

Hệ thống phòng vệ RMISecurityManager sẽ sử dụng các thiết lập trong tập tin D:\jdk1.3\jre\lib\security\java.policy để cho phép hoặc cấm các kết nối từ xa. Bạn hãy mở tập tin này (đây chỉ là tập tin văn bản) xóa trắng và đánh vào nội dung sau:

```
grant {
 permission java.security.AllPermission;
};
```

điều này có nghĩa là ... không phòng vệ gì cả. Tuy nhiên bạn sẽ không gặp phải những phiền toái do chính sách bảo mật nghiêm ngặt của Java gây nên. Chương sau chúng ta sẽ trở lại với vấn đề bảo mật và tìm hiểu tập tin java.policy này.

Còn bây giờ bạn hãy biên dịch tập tin CalculatorClient.java. Chép tập tin đã biên dịch xong (CalculatorClient.class) sang một thư mục khác (thư mục C:\TESTING) cùng với lớp giao tiếp Calculator.class để giao tiếp với đối tượng. Chúng ta thiết lập 3 máy ảo java chạy trên 3 cửa sổ DOS-Prompt khác nhau với ngữ cảnh và trình tự thực hiện như sau:

- Máy ảo Java chạy chương trình rmiregistry

  Biến môi trường CLASSPATH=.;

  Chạy chương trình C:\jdk1.3\bin\>rmiregistry.exe

- Máy ảo Java chạy chương trình CalculatorServer dùng đăng ký đối tượng

  Biến môi trường CLASSPATH=.;C:\Network

  Chạy chương trình

```
C:\Network>java -Djava.rmi.server.codebase="
   http://172.16.11.12/myclass/" CalculatorServer
```

- Dịch vụ Web server trong thư mục gốc \myclass chứa tập tin CalculatorImpl_Stub.class

- Máy ảo Java chạy chương trình khách

  Biến môi trường CLASSPATH=.;

  Chạy chương trình C:\TESTING>java CalculatorClient

Chương trình rmiregistry khởi động từ thư mục C:\Jdk-1.3\bin trên máy ảo có biến môi trường CLASSPATH=.; nghĩa là rmiregistry

luôn truy tìm các tập tin .class trong thư mục hiện hành C:\jdk-1.3\bin. Thư mục hiện hành này không hề có các tập tin .class của đối tượng Calculator.

Chương trình CalculatorServer chạy trên máy ảo có biến môi trường CLASSPATH=.;C:\Network. Thư mục C:\Network của ta chứa đầy đủ các tập tin .class của chương trình. Khi Calculator đăng ký với rmiregistry, nó yêu cầu rmiregistry truy tìm các tập tin .class ở địa chỉ web URL http://172.16.11.12/myclasses/. Bạn hãy thử tắt dịch vụ web server trên máy chủ, một khi trình rmiregistry không liên lạc được với web server để lấy về tập tin Calculator_Stub.class bạn sẽ nhận được thông báo lỗi.

Chương trình CalculatorClient khởi động từ thư mục C:\TESTING\ trên máy ảo có biến môi trường CLASSPATH=.; Thư mục hiện hành C:\TESTING chỉ chứa lớp chương trình CalculatorClient.class và lớp giao tiếp Calculator.class. Bạn sẽ thấy CalculatorClient tự động nạp CalculatorImpl_Stub.class từ xa về và chương trình của bạn vẫn chạy cho kết quả tương tự như khi bạn đặt CalculatorImpl_Stub.class trên máy cục bộ.

Để thiết lập biến môi trường CLASSPATH khác nhau cho từng máy ảo, bạn chỉ cần đánh lệnh SET CLASSPATH= từ cửa sổ DOS-Prompt của máy ảo. Ví dụ:

SET CLASSPATH=.;C:\Network

Biến môi trường thiết lập từ một cửa sổ máy ảo sẽ không ảnh hưởng đến cửa sổ máy ảo khác.

Bạn đừng thất vọng khi có quá nhiều điều phải làm cho một chương trình đơn giản như Calculator. Thật sự lập trình phân tán là vấn đề khá hóc búa và luôn gây đau đầu cho các lập trình viên khi gỡ lỗi, kể cả các lập trình viên chuyên nghiệp. Để tóm tắt chúng ta có thể xây dựng một chương trình phân tán khác cô động hơn, chương trình HelloWorld.

6. CHƯƠNG TRÌNH HELLOWORLD

6.1. Cài đặt chương trình

Trên đây là các bước chi tiết để cài đặt một ứng dụng phân tán bằng kỹ thuật RMI của Java. Để bạn cảm thấy đơn giản hơn chúng tôi trở lại trình bày chương trình HelloWorld kinh điển được thiết kế ở dạng đối tượng phân tán gọi lời chào sayHello() từ xa. Chương trình được hình thành qua các bước sau:

Bước 1: Đặc tả giao tiếp interface cho lớp đối tượng ở xa mang tên Hello. Mã nguồn chương trình được trình bày ở ví dụ 2-6

Ví dụ 2-6 Hello.java

```
import java.rmi.*;
public interface Hello extends Remote{
  public String sayHello()throws RemoteException;
}
```

Bước 2: Thực hiện cài đặt cho lớp giao tiếp Hello

Ví dụ 2-7 HelloImpl.java

```
import java.rmi.*;
public class HelloImpl implements Hello{
  public String sayHello()throws RemoteException {
    System.out.println("Client request") ;
    return "Hello World";
  }
}
```

Bước 3: Viết chương trình cài đặt (hay khởi tạo đối tượng trên máy chủ)

Ví dụ 2-8 Setup.java

```
import java.rmi.*;
```

```
import java.rmi.server.*;
public class Setup{
  public static void main(String args[])
                                    throws Exception{
    //Kiến tạo đối tượng
    HelloImpl hello=new HelloImpl();
    //Yêu cầu máy ảo nhận dạng đối tượng
    UnicastRemoteObject.exportObject(hello);
    //Đăng ký (hay ràng buộc) đối tượng với dịch vụ tìm kiếm
Naming
    System.out.println("Registering object ...");
    Naming.bind("rmi://localhost/myhello",hello);
    System.out.println("Waiting for
                                    client request...");
  }
}
```

Bước 4: Viết chương trình kiểm tra hay triệu gọi đối tượng từ máy khách ở xa

Ví dụ 2-9 Client.java

```
import java.rmi.*;
public class Client{
  public static void main(String args[])
                                    throws Exception{
    //Truy tìm tham chiếu của đối tượng dựa vào dịch vụ Naming
    System.out.println("Finding object ...");
    Hello obj=(Hello)Naming.lookup("
                                    rmi://localhost/myhello");
```

```
    //Triệu gọi đối tượng
    System.out.println(obj.sayHello());
  }
}
```

Bước 5: Biên dịch, bạn lưu tất cả mã nguồn của chương trình vào thư mục C:\Network\Hello. Chuyển vào thư mục này biên dịch tất cả các tập tin nguồn như sau:

```
C:\Network\Hello>javac *.java
```

Bước 6: Tạo các tập tin trung gian _Stub và _Skel cho đối tượng.

```
C:\Network\Hello>rmic HelloImpl
```

Qua 6 bước trên bạn sẽ thu được các tập tin lớp .class trong thư mục C:\Network\Hello như sau:

```
Hello.class
HelloImpl.class
HelloImpl_Stub.class
HelloImpl_Skel.class
Setup.class
Client.class
```

Bước 7: Chạy chương trình theo các thứ tự sau:

Khởi động bộ đăng ký rmiregistry

```
C:\Network\Hello>start rmiregistry
```

Khởi động trình chủ đăng ký đối tượng

```
C:\Network\Hello>start java Setup
Binding object ...
Waiting for client request...
```

Chạy chương trình gọi đối tượng từ máy khách

```
C:\Network\Hello>java Client
Finding object ...
Hello World
```

6.2. Các biến tấu khi cài đặt đối tượng

Ở ví dụ Hello World trên, như bạn thấy quá trình đặc tả và cài đặt đối tượng Hello chỉ thực sự diễn ra ở bước 1 và bước 2 (thiết kế lớp Hello và HelloImpl) Chức năng chính của đối tượng là cung cấp phương thức sayHello() cho các máy khách ở xa gọi nó. Khả năng thao tác của một đối tượng thường được gọi là logic nghiệp vụ (business logic) của đối tượng đó.

Quá trình cài đặt đối tượng (chương trình Setup.java) trông có vẻ phức tạp nhưng sau này sẽ được các chương trình khác như Web Server hay Application Server thay thế. Bạn chỉ việc cấu hình cho các trình chủ này cách thức đưa đối tượng của bạn vào môi trường hoạt động của chúng là đủ. Các chương sau sẽ giới thiệu với bạn những trình chủ nổi tiếng hiện có thể thực hiện công việc này. Còn bây giờ chúng ta hoàn toàn có thể sử dụng Setup.java như cách cài đặt thô sơ và hữu hiệu nhất.

Trở lại với Setup.java, lệnh UnicastRemoteObject.exportObject() là một lệnh bắt buộc mà Java qui định bạn phải gọi để máy ảo Java nhận dạng ra được đối tượng của bạn có khả năng triệu gọi từ xa. Nếu không muốn gọi phương thức này thì đối tượng của bạn phải cài đặt kế thừa lớp UnicastRemoteObject. Ví dụ HelloImpl có thể tự động thông báo cho máy ảo Java khả năng trả lời từ xa của nó như sau:

```
import java.rmi.*;
import java.rmi.server.*;
public class HelloImpl extends
            UnicastRemoteObject implements Hello{
 public HelloImpl() throws RemoteException{
   super();
 }
```

```
  public String sayHello()throws RemoteException {
    System.out.println("Client request") ;
    return "Hello World";
  }
}
```

Phương thức khởi dựng HelloImpl() của đối tượng gọi lại lớp cha super() tức lớp UnicastRemoteObject. Và nếu khám phá mã nguồn của Java bạn cũng sẽ thấy phương thức exportObject được gọi. Bằng cách này trình Setup.java của bạn có thể bỏ đi lệnh UnicastRemoteObject.exportObject và viết lại đơn giản hơn như sau:

```
import java.rmi.*;
public class Setup{
  public static void main(String args[])
                                    throws Exception{
    HelloImpl hello=new HelloImpl();
    Naming.bind("rmi://localhost/myhello",hello);
  }
}
```

Khi tìm hiểu phong cách thiết kế đối tượng RMI ở một số ví dụ mẫu của Sun bạn sẽ gặp cách thức cài đặt này. Chúng tôi không chọn phương án kế thừa lớp UnicastRemoteObject cho đối tượng bởi sau này bạn sẽ thấy lớp UnicastRemoteObject không khả chuyển. Lớp này chỉ thực sự thích hợp cho các ứng dụng RMI. Khi làm quen với CORBA (một khái niệm lập trình phân tán tương tự RMI nhưng phổ biến và đa dụng hơn) lớp UnicastObject sẽ thay bằng lớp PortableRemoteObject.

Đôi lúc, trình cài đặt Setup.java được viết gộp chung với phần cài đặt đối tượng HelloImpl.java như sau:

```
import java.rmi.*;
```

```
import java.rmi.server.*;
public class HelloImpl extends
                UnicastRemoteObject implements Hello{
 public HelloImpl() throws RemoteException{
   super();
 }
  public String sayHello()throws RemoteException {
   System.out.println("Client request") ;
   return "Hello World";
  }
  public static void main() throws Exception{
    HelloImpl hello=new HelloImpl();
    System.out.println("Binding object ...");
    Naming.bind("rmi://localhost/myhello",hello);
    System.out.println("Waiting request...");
  }
}
```

Bằng cách này bạn không phải viết tập tin Setup.java, thay cho lời gọi `start java Setup` bạn có thể gọi `start java HelloImpl`. Cách cài đặt này chỉ dành cho các lập trình viên thành thạo với java nhưng chúng tôi cũng khuyên bạn không nên sử dụng. Bởi vì mã nguồn của đối tượng và phần thiết lập đối tượng trộn lẫn với nhau.

Vì lý do đơn giản hoá chúng ta đã loại bỏ phần thiết lập cơ chế bảo vệ ở phía máy chủ và máy khách. Nếu muốn bạn có thể thêm vào lớp phòng vệ ở trình khách Client và trình chủ Server như sau:

```
public class Client{
 public static void main(String args[])
```

```
                                        throws Exception{
    System.setSecurityManager(new
                                    RMISecurityManager());
    ...
  }
}
public class Setup{
  public static void main(String args[])
                                        throws Exception{
    System.setSecurityManager(new
                                    RMISecurityManager());
    ...
  }
}
```

Nhớ rằng lệnh `System.setSecurityManager` phải luôn được đặt ngay phần bắt đầu của hàm main(). Bạn tham khảo lại ví dụ Calculator ở ví dụ 2-5 về cơ chế phòng vệ.

7. CHUYỂN THAM SỐ TRONG CÁC LỜI GỌI PHƯƠNG THỨC TỪ XA

7.1. Chuyển tham số theo tham trị và tham biến

Đối với java, trên máy cục bộ hầu hết biến kiểu đối tượng đều truyền theo tham chiếu trong các lời gọi hàm. Nghĩa là một khi biến đối tượng được truyền vào phương thức nếu bên trong phương phức thay đổi giá trị của đối tượng thì khi lời gọi phương thức chấm dứt giá trị của đối tượng cũng thay đổi theo. Ví dụ:

```
class Number{
```

```
 public int value=0;
 public Number(int v){
   value=v;
 }
}
public class Program{
  public static void main(String args[]){
    Number num= new Number(12);
       doIncrease(num);
    System.out.println(num.value);
  }
  public static void doIncrease(Number n){
    n.value++;
  }
}
```

Ở ví dụ trên ta tạo ra đối tượng Number lưu một giá trị nguyên integer. Phương thức doIncrease() tiếp nhận tham số num có kiểu đối tượng là Number giá trị khởi đầu là 12. Bên trong phương thức doIncrease() biến đối tượng được tăng giá trị lên 1. Kết quả khi lời gọi phương thức chấm dứt đối tượng num bị tăng trị lên 13.

Tuy nhiên các kiểu dữ liệu nguyên tố như int, float, double, char, byte, long ... lại được truyền theo trị (tham trị). Giá trị tham số mà hàm hay phương thức xử lý chỉ là bản sao của biến truyền từ ngoài vào. Ví dụ sau sẽ cho kết quả là 12 sau khi gọi phương thức doIncrease().

```
public class Program{
  public static void main(String args[]){
    int num= 12;
```

```
    doIncrease(num);
    System.out.println(num);
  }
  public static void doIncrease(int n){
    n++;
  }
}
```

Tóm lại trong Java đối tượng được truyền theo tham chiếu, còn các kiểu dữ liệu đơn giản như int, char ... được truyền theo tham trị. Thế còn truyền tham số qua mạng theo cơ chế RMI thì sao? Thật sự truyền tham số cho các lời gọi phương thức từ xa trong RMI có hơi khác với nguyên tắc truyền tham số thông thường. Bạn hãy nhớ hai nguyên tắc sau:

- Tất cả các kiểu dữ liệu đơn giản như int, char ... đều được truyền theo tham trị.
- Tất cả các dữ liệu kiểu đối tượng muốn truyền qua mạng đều buộc phải cài đặt 1 trong hai giao tiếp Remote hoặc Serializable. Các đối tượng cài đặt giao tiếp Remote sẽ được truyền theo tham chiếu còn các đối tượng cài đặt giao tiếp Serializable sẽ được truyền theo tham trị.

7.2. Chuyển đối tượng đến trình chủ theo tham trị

Hầu như các kiểu dữ liệu đối tượng cơ bản như String, Date, Time ... trong java đều cài đặt giao tiếp Serializable, cho nên chúng được chuyển cho các lời gọi hàm hay phương thức ở xa theo trị.

Tương tự lớp Remote, lớp Serializable chỉ là một lớp rỗng dùng cho mục đích báo hiệu. Tất cả các lớp đối tượng cài đặt giao tiếp Serializable đều được java xem như có khả năng tuần tự hóa (serialize). Nghĩa là bạn có thể yêu cầu Java lưu toàn bộ đối tượng lên đĩa cứng thành một tập tin nào đó, sau đó đem tập tin này đến máy khác, bạn có thể khôi phục đối tượng về trạng thái ban đầu mà không cần phải khởi tạo đối tượng mới bằng lệnh new. Bạn xem thêm khả năng tuần tự hoá của đối tượng ở mục 8

Với cơ chế truyền tham số đối tượng theo trị, khi bạn gọi một phương thức của đối tượng ở xa, nếu trong lời gọi phương thức này có yêu cầu tham số là kiểu đối tượng, đối tượng sẽ được đóng gói và chuyển toàn bộ đến máy chủ (nơi tiếp nhận tham số và thực thi phương thức). Tại máy chủ đối tượng sẽ được bung ra lại trạng thái ban đầu và đưa vào sử dụng. Quá trình đóng gói tham số trên máy khách để chuyển đi được thực hiện bởi lớp trung gian _Stub, ngược lại quá trình mở gói dữ liệu để khôi phục tham số diễn ra trên máy chủ được thực hiện bởi lớp trung gian _Skel. Qui trình chuyển tham số và dữ liệu qua lại giữa hai lớp trung gian _Stub và _Skel trong thuật ngữ lập trình phân tán gọi là mashaling data.

Trong các lời gọi phương thức RMI, kiểu dữ liệu đối tượng nếu không cài đặt một trong hai giao tiếp Serializable hoặc Remote thì sẽ không thể dùng làm tham số chuyển qua mạng được.

7.3. Xây dựng chương trình chuyển tham số đối tượng qua mạng

Dưới đây là một chương trình ví dụ cho thấy cách chuyển một đối tượng từ máy khách đến máy chủ. Chương trình của chúng ta tựa trò chơi ném bóng, trình khách sẽ ném một quả banh (đối tượng Ball) lên máy chủ, trình chủ sẽ tiếp nhận quả banh và trả về cho trình khách.

Bước 1: Thiết kế lớp đối Ball.class được dùng làm tham số chuyển qua mạng giữa trình khách và trình chủ. Mã nguồn Ball.java được cài đặt như sau:

Ví dụ 2-10 Ball.java

```
import java.io.*;
public class Ball implements Serializable {
  int weight=0;
```

```
  public Ball(int w){
    weight=w;
  }
  public int getWeight(){
    return weight;
  }
  public void setWeight(int w){
    weight=w;
  }
}
```

Lớp Ball chỉ dùng để biểu diễn một quả banh có trọng lượng weight. Bạn có thể gọi hàm `getWeight()`, `setWeight()` để lấy về hoặc thiết lập trọng lượng cho quả banh. Mục đích của ta là sử dụng đối tượng Ball để chuyển đi trên mạng giữa trình khách và trình chủ, cho nên lớp Ball được khai báo là có khả năng tuần tự hoá (thông qua việc cài đặt giao tiếp Serializable).

```
public class Ball implements Serializable{
  ...
}
```

Nếu bạn cảm thấy từ ngữ "tuần tự hoá" hay "Serializable" nghe có vẻ khó hiểu thì cũng không nên bận tâm, mục sau chúng ta sẽ tìm hiểu về nó kỹ hơn. Đơn giản bạn hãy hình dung, muốn khai báo biến nguyên thì phải sử dụng kiểu int, còn muốn đối tượng chuyển được qua mạng thì bạn phải áp đặt Serializable lên đối tượng là đủ.

Bước 2: Đặc tả giao tiếp interface cho đối tượng trên máy chủ.

Ví dụ 2-11 PingServer.java

```
import java.rmi.*;
public interface PingServer extends Remote {
```

```
  public Ball ping(Ball b) throws RemoteException;
}
```

PingServer được đặt trên trình chủ. Đối tượng PingServer của ta có một phương thức ping(). Phương thức này sẽ được gọi bởi trình khách. Trình khách sử dụng phương thức ping() để ném quả bóng lên trình chủ thông qua tham số có kiểu đối tượng là Ball. Phương thức ping() sau khi tiếp nhận sẽ ném trả về đối tượng Ball cho trình khách thông qua trị trả về của phương thức.

Bước 3: Cài đặt chi tiết cho đối tượng PingServer thông qua lớp PingServerImpl với mã nguồn như sau:

Ví dụ 2-12 PingServerImpl.java

```
import java.rmi.*;
public class PingServerImpl implements PingServer {
  public Ball ping(Ball b) throws RemoteException{
 //Tiếp nhận đối tượng từ trình khách
    System.out.println(" Client send a Ball object
                                weight "+b.getWeight());
 //Thay đổi nội dung đối tượng – hay nói cách khác tăng trọng lượng
 //của quả banh lên
    b.setWeight(b.getWeight()+15);
    //Ném đối tượng về cho trình khách
    return b;
  }
}
```

Trình chủ của ta tiếp nhận quả banh từ trình khách, tăng trọng lượng của quả banh lên và ném trả về cho trình khách.

Tiếp theo là phần cài đặt để sử dụng chương trình ở hai phía khách chủ.

Bước 4: Thiết kế chương trình Setup cài đặt đối tượng PingServerImpl trên máy chủ.

Ví dụ 2-13 Setup.java

```
import java.rmi.*;
import java.rmi.server.*;
public class Setup {
  public static void main(String args[])
                                    throws Exception{
    //Kiến tạo đối tượng
    PingServer server=new PingServerImpl();
    //Thông báo khả năng giao tiếp được từ xa của đối tượng với
máy ảo Java
    UnicastRemoteObject.exportObject(server);
    //Đăng ký đối tượng với dịch vụ rmiregistry
    //(thay localhost bằng địa chỉ IP thật sự nếu bạn làm việc trên
mạng)
Naming.bind("rmi://localhost/pingobject",server);
    System.out.println("Waiting for
                                    client request ...");
  }
}
```

Bước 5: Thiết kế chương trình Client gọi phương thức của đối tượng PingServer.

Ví dụ 2-14 Client.java

```
import java.rmi.*;
public class Client {
  public static void main(String args[])
                                    throws Exception{
```

```
    //Tạo quả banh có trọng lượng là 12
    Ball ball=new Ball(12);
    //Yêu cầu rmiregistry truy tìm đối tượng PingServer
    PingServer server=(PingServer)Naming.lookup(
                        "rmi://localhost/pingobject");
    System.out.println("Ball weight before send to
                        server "+ball.getWeight());
    Ball anotherBall=server.ping(ball);
    System.out.println("Ball weight after send to
                        server "+ball.getWeight());
    System.out.println("Ball weight return by
                        server "+anotherBall.getWeight());
  }
}
```

Bước 6: Biên dịch chương trình, bạn lưu tất cả mã nguồn vào thư mục C:\Network\ByValue. Chuyển vào thư mục này thực hiện lệnh biên dịch.

```
C:\Network\ByValue\>javac *.java
```

tiếp đến tạo các tập tin trung gian _Stub và _Skel bằng lệnh

```
C:\Network\ByValue\>rmic PingServerImpl
```

kết quả ta thu được các tập tin sau:

```
Ball.class
PingServer.class
PingServerImpl.class
PingServerImpl_Stub.class
PingServerImpl_Skel.class
```

```
Setup.class
Client.class
```

Bước 7: Cài đặt đối tượng và thực thi chương trình. Bạn gõ từ cửa sổ DOS-Prompt các lệnh sau:

- Khởi động bộ đăng ký rmiregistry

```
C:\Network\ByValue\>start  rmiregistry
```

- Gọi trình Setup để đăng ký đối tượng PingServer

```
C:\Network\ByValue\>start java Setup
Waiting for client request ...
Client send a Ball object weight 12
```

- Chạy trình khách Client

```
C:\Network\ByValue\>java Client
Ball weight before send to server 12
Ball weight after send to server 12
Ball weight return by server 27
```

Như bạn thấy, trong trình khách Client.java, đối tượng ball trước khi gọi phương thức ping() có giá trị là 12. Sau khi gọi phương thức ping() vẫn giữ nguyên trị là 12. Trong khi đó đối tượng ball đưa lên trình chủ được tăng giá trị lên 15. Kết quả trả về từ trình chủ là một đối tượng anotherBall khác với đối tượng ball đã chuyển đi từ trình khách (trọng lượng trả về của anotherBall là 27=15+12). Rõ ràng đối tượng ball được phương thức ping() chuyển đi theo tham trị.

Hãy thử bỏ đi đặc tính Serializable của lớp Ball. Quá trình biên dịch vẫn thành công nhưng khi chương trình thực thi bạn sẽ gặp phải lỗi ngoại lệ do java ném ra là:

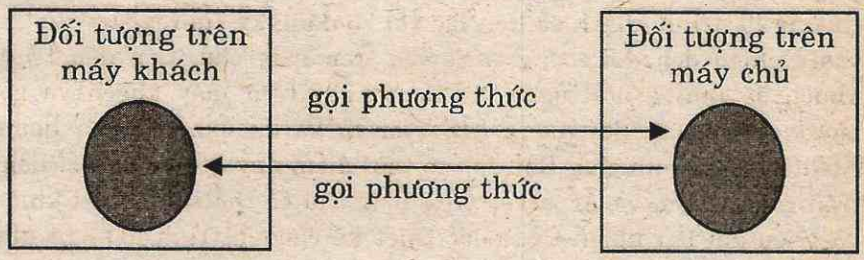
```
java.io.NotSerializableException: Ball
```

Java yêu cầu các đối tượng chuyển theo tham trị khi truyền vào phương thức gọi từ xa bắt buộc phải được đánh dấu là Serializable.

7.4. Chuyển đối tượng đến trình chủ theo tham chiếu

Nếu đối tượng Ball quá lớn thì sao? Đóng gói toàn bộ đối tượng chuyển đi chuyển lại trên mạng sẽ ảnh hưởng đến tốc độ thực thi của chương trình. Có cách nào mà trình chủ có thể tham chiếu và xử lý được trực tiếp đối tượng đang nằm trên máy khách hay không? Có nghĩa là nếu trình chủ được trình khách truy xuất từ xa thì chính trình khách ngược lại cũng có thể được gọi từ xa bởi trình chủ. Hay nói khác hơn, trình khách không cần chuyển đối tượng cho trình chủ theo trị mà có thể chuyển theo tham chiếu.

Bằng cách này bạn có thể cho đối tượng trên trình khách và trình chủ triệu gọi lẫn nhau (trước giờ ta vẫn theo cơ chế "thụ động" là trình khách gọi phương thức của trình chủ)



Hình 2-8 Đối tượng giữa trình khách và trình chủ triệu gọi lẫn nhau

Bạn hãy nhớ lại làm cách nào trình khách có thể tham chiếu đến được đối tượng trên máy chủ? Đối tượng trên máy chủ yêu cầu phải cài đặt lớp giao tiếp Remote, tiếp đến là sinh ra các lớp trung gian _Stub và _Skel, sau cùng là đăng với rmiregistry để trình khách tham chiếu đến. Chúng ta sẽ làm điều này hoàn toàn tương tự với các đối tượng trên trình khách. Với Java các đối tượng nếu cài đặt giao tiếp Remote sẽ được xem như có khả năng chuyển qua mạng thông qua tham chiếu chứ không cần phải đóng gói chuyển đi theo tham trị như trường hợp cài đặt lớp Serializable.

Cơ chế gọi ngược từ xa của trình chủ đến trình khách thông qua tham chiếu còn được gọi là cơ chế callback. Kỹ thuật gọi ngược callback thường rất hữu hiệu cho trình khách thiết lập các yêu cầu tùy biến không phải lệ thuộc nhiều vào trình chủ.

7.5. Xây dựng chương trình chuyển đối tượng qua mạng theo tham chiếu

Dưới đây là một chương trình ví dụ cho thấy cách cài đặt kỹ thuật callback triệu gọi lẫn nhau giữa đối tượng trên trình khách và đối tượng trên trình chủ thông qua cơ chế tham chiếu.

Chúng ta sẽ tạo ra hai đối tượng, đối tượng AtClient chạy trên máy khách và đối tượng AtServer chạy trên máy chủ. Đầu tiên như thường lệ, trình khách sẽ liên lạc với bộ đăng ký rmiregistry để tìm tham chiếu đến đối tượng AtServer trên máy chủ, tiếp đến trình khách sẽ tạo ra đối tượng AtClient ngay trên máy khách và gọi phương thức của đối tượng AtServer từ xa để đăng ký đối tượng AtClient với trình chủ. Bắt đầu từ thời điểm này đối tượng AtClient trên máy khách và AtServer trên máy chủ có thể tự do điều khiển và triệu gọi lẫn nhau. Các bước thiết kế chương trình diễn ra như sau:

Bước 1: Đặc tả giao tiếp interface cho đối tượng AtClient

Ví dụ 2-15 AtClient.java

```
import java.rmi.*;
public interface AtClient extends Remote {
  public void callClientMethod(String message)
                                  throws RemoteException;
}
```

Đối tượng AtClient của ta đặt trên máy khách và muốn có khả năng gọi được từ máy chủ, java yêu cầu bạn phải áp đặt giao tiếp

Remote cho lớp đối tượng này. Và tương tự như cách cài đặt cho các đối tượng trên máy chủ trước đây, phương thức được triệu gọi từ xa callClientMethod() phải có khả năng ném ra ngoại lệ RemoteException. Sau này bạn sẽ thấy callClientMethod() được gọi bởi đối tượng AtServer trên máy chủ.

Bước 2: Đặc tả giao tiếp interface cho đối tượng AtServer

Ví dụ 2-16 AtSrver.java

```
import java.rmi.*;
public interface AtServer extends Remote {
  public void registerClient(AtClient c)
                                    throws RemoteException;
  public void callServerMethod(String message)
                                    throws RemoteException;
}
```

Đối tượng AtServer của ta cung cấp hai phương thức, phương thức thứ nhất registerClient() dùng để tiếp nhận tham chiếu đến đối tượng AtClient trên trình khách. Phương thức thứ hai callServerMethod() được dùng để cung cấp các dịch vụ cho trình khách.

Phần cài đặt chi tiết cho nội dung đối tượng AtClient và AtServer được thực hiện như sau:

Bước 3: Cài đặt chi tiết cho đối tượng AtClient thông qua lớp AtClientImpl

Ví dụ 2-17 AtClientImpl.java

```
import java.rmi.*;
public class AtClientImpl implements AtClient {
  public void callClientMethod(String message)
                                    throws RemoteException{
```

```
    System.out.println(message);
  }
}
```

Bước 4: Cài đặt chi tiết cho đối tượng AtServer thông qua lớp AtServerImpl

Ví dụ 2-18 AtServerImpl.java

```
import java.rmi.*;
public class AtServerImpl implements AtServer {
AtClient client;//Lưu giữ tham chiếu đến đối tượng trên máy khách
  //Phương thức này được trình khách triệu gọi để đăng ký
  //đối tượng AtClient với trình chủ
  public void registerClient(AtClient c)
                                  throws RemoteException{
    client=c;
  }
  public void callServerMethod(String message)
                                  throws RemoteException{
    System.out.println(message);
    for (int i=1; i<10; i++){
  String msg= "Server response "+ Math.random()*1000;
      //Triệu gọi phương thức của đối tượng chạy trên máy khách
      client.callClientMethod(msg);
    }
  }
}
```

Bước 5: Xây dựng chương trình cài đặt đối tượng AtServer trên máy chủ và đăng ký đối tượng AtServer với rmiregistry.

Ví dụ 2-19 Setup.java

```
import java.rmi.*;
import java.rmi.server.*;
public class Setup {
  public static void main(String args[])
                                    throws Exception{
    //Kiến tạo đối tượng chủ
    AtServer server=new AtServerImpl();
    //Thông báo khả năng giao tiếp được từ xa của đối tượng với
máy ảo Java
    UnicastRemoteObject.exportObject(server);
    //Đăng ký đối tượng với rmiregistry
    Naming.bind("rmi://localhost/serverobject",
                                          server);
    System.out.println("Waiting for client
                                      request ...");
  }
}
```

Bước 6: Thiết kế chương trình Client gọi phương thức của đối tượng AtServer.

Ví dụ 2-20 Client.java

```
import java.rmi.*;
import java.rmi.server.*;
public class Client {
  public static void main(String args[])
                                    throws Exception{
```

```
    //Khởi tạo đối tượng của trình khách
    AtClient client=new AtClientImpl();
    UnicastRemoteObject.exportObject(client);
    AtServer server=(AtServer)Naming.lookup(
                        "rmi://localhost/serverobject");
  server.registerClient(client);
  server.callServerMethod("Client Contact Server");
 }
}
```

Bởi vì AtClient là một đối tượng có khả năng tham chiếu xa cho nên tương tự như đối tượng trên máy chủ, chúng ta phải gọi UnicastRemoteObject.exportObject() để thông báo sự hiện diện và khả năng giao tiếp của đối tượng AtClient với máy ảo java. Sau khi lấy về tham chiếu đến đối tượng trên máy chủ AtServer bằng phương thức Naming.lookup() quen thuộc, trình khách triệu gọi phương thức registerClient() của đối tượng chủ. Phương thức này mục đích là truyền tham chiếu của AtClient từ trình khách lên trình chủ. Bằng cách này trình chủ không còn phải dùng đến hàm Naming.lookup() truy tìm tham chiếu đến đối tượng AtClient của khách trên rmiregistry nữa.

Bước 7: Biên dịch và chạy chương trình, bạn lưu tất cả mã nguồn của chương trình vào thư mục C:\Network\ByRef. Chuyển vào thư mục này để biên dịch mã nguồn.

```
C:\Network\ByRef\>javac *.java
```

Tạo tập tin trung gian _Stub và _Skel cho hai lớp đối tượng AtServer và AtClient.

```
C:\Network\ByRef\>rmic AtServerImpl
C:\Network\ByRef\>rmic AtClientImpl
```

- Khởi động rmiregistry

```
C:\Network\ByRef\>start rmiregistry
```

- Cài đặt đối tượng trên máy chủ

```
C:\Network\ByRef\>start java Setup
```

- Chạy trình khách

```
C:\Network\ByRef\>java Client
```

Kết xuất từ phía máy chủ	Kết xuất từ phía máy khách
Waiting for client request ...	Server response 82.21
Client Contact Server	Server response 611.31
	Server response 298.24
	Server response 295.01
	Server response 233.26
	Server response 784.93
	Server response 982.08
	Server response 124.91
	Server response 334.83

Khi trình khách gọi phương thức server.callServerMethod() của đối tượng chủ, phương thức này trên máy chủ lại sử dụng và gọi ngược lại phương thức client.callClientMethod() của đối tượng đang chạy trên trình khách. Trước khi gọi callClientMethod() trình chủ thực hiện một số tính toán

```
String msg= "Server response "+ Math.random()*1000;
```

kết quả tính toán sau đó được chuyển về cho trình khách và trình khách tự động in ra kết quả. Như bạn thấy, sau khi trình chủ tính toán xong nó chủ động yêu cầu trình khách tiếp nhận kết quả. Nếu không dùng kỹ thuật gọi ngược callback, trình chủ sẽ không có cách nào liên lạc được với trình khách để trả về kết quả mà khi đó trình khách phải tự canh thời gian (như thiết lập timer chẳng hạn) để gọi phương thức trên trình chủ lấy kết quả về.

Điều này tương tự như dịch vụ thông báo giá chẳng hạn. Thông thường cứ 5 phút bạn lại phải gọi điện lên dịch vụ để cập nhật thông tin về giá cả. Cơ chế này chính là lời gọi hàm từ trình khách đến

trình chủ. Ngược lại, bạn có thể đăng ký tên (hay số điện thoại) của mình với dịch vụ giá cả và cứ 5 phút dịch vụ cung cấp giá sẽ gọi đến bạn để cập nhận giùm bạn thông tin mới. Đây chính là cơ chế gọi ngược callback từ trình chủ về trình khách thông qua tham chiếu.

Tất cả các đối tượng muốn gọi được hay tham chiếu từ xa đều phải cài đặt giao tiếp java.rmi.Remote và tạo hai lớp giao tiếp trung gian _Stub, _Skel thông qua trình biên dịch rmic

Như vậy bạn đã biết trong các lời gọi phương thức ở xa nếu một đối tượng được cài đặt giao tiếp java.io.Serializable thì nó sẽ được truyền theo tham trị, còn nếu cài đặt giao tiếp java.rmi.Remote thì sẽ được truyền theo tham chiếu. Vì Java cho phép một đối tượng có thể cài đặt cùng lúc nhiều giao tiếp interface khác nhau, cho nên nếu đối tượng cài đặt cùng lúc cả hai giao tiếp Serializable và Remote thì sao? Ví dụ:

```
public interface MyObject extends Remote,
                                  Serializable{

    ...

}
```

và

```
public class MyObjectImpl implements MyObject{

    ...

}
```

Đối tượng MyObject được truyền theo tham chiếu hay tham trị? Một bài tập dành cho bạn. Với ý tưởng trên bạn hãy thử cài đặt và chạy chương trình xem Java xử lý ra sao?

Mách nhỏ: Nếu bạn cài đặt cả hai giao tiếp Remote và Serializable thì vẫn không sao cả. Đối tượng của bạn sẽ có cùng lúc hai khả năng vừa có thể dùng làm tham chiếu vừa có thể dùng làm tham trị. Lúc này lệnh UnicastRemoteObject.exportObject() sẽ thể hiện rõ vai trò của nó. Nếu bạn thông báo cho Java sự hiện diện và

khả năng nói chuyện được từ xa của đối tượng bằng hàm UnicastRemoteObject.exportObject() thì đối tượng của bạn sẽ được truyền đi theo tham chiếu. Nếu không có lệnh này đối tượng của bạn sẽ được truyền theo tham trị.

8. TUẦN TỰ HOÁ (SERIALIZABLE) ĐỐI TƯỢNG

Java cung cấp cho bạn khả năng ghi toàn bộ đối tượng xuống một tập tin (đối tượng có thể đang ở một trạng thái nào đó). Sau đó bạn có thể đem tập tin chứa đối tượng đi khắp nơi nếu cần thì khôi phục đối tượng về trạng thái ban đầu, đưa đối tượng vào bộ nhớ sử dụng đúng y như tình trạng trước khi nó được lưu xuống đĩa. Các đối tượng có khả năng này được gọi là tuần tự hoá (serializable). Lý do có thuật ngữ trên xuất phát từ việc lưu đối tượng lên đĩa, dữ liệu lưu trên tập tin là loại dữ liệu tuần tự. Đối tượng trong bộ nhớ có thể tham chiếu quan hệ chằng chịt "giây mơ rễ má" với nhau nhưng khi được lưu xuống tập tin thì bắt buộc phải trải phẳng đối tượng ra theo thứ tự.

Trong java các đối tượng muốn có khả năng ghi và lưu trữ được thành tập tin thì phải cài đặt giao tiếp (interface) java.io.Serializable. Lớp giao tiếp này chỉ là một lớp rỗng thực hiện chức năng đánh dấu khả năng tuần tự hóa của đối tượng mà thôi. Việc đọc ghi các đối tượng không phải lúc nào cũng đơn giản, có những đối tượng mà bản thân nó bao hàm nhiều đối tượng con khác thì sao? Bạn không cần phải lo điều đó. Một khi đối tượng đã có khả năng tuần tự hoá và yêu cầu Java ghi thành tập tin, Java sẽ "túm" lấy tất cả các đối tượng con chứa bên trong đối tượng cần lưu đưa luôn xuống đĩa. Quá trình khôi phục đối tượng luôn bảo đảm rằng trạng thái và nội dung của đối tượng trước khi lưu được khôi phục nguyên vẹn. Đối tượng có thể trở về hoạt động bình thường sau khi khôi phục.

Dưới đây là một ví dụ cho thấy khả năng tuần tự hoá của một đối tượng phức hợp.

Ví dụ 2-21 WriteObjectApp.java

```
import java.io.*;
/* Lớp đối tượng có khả năng lưu thành tập tin trên đĩa */
class LinkObject implements Serializable{
  static int count=1;
  LinkObject next;
  int id;
  public LinkObject(int id){
     this.id=id;
      if (count<50) {
           count++;
           next=new LinkObject(count);
      } else next=null;
  }
  public String toString(){
      return ("Object  "+id);
  }
}
public class WriteObjectApp{
   public static void main(String args[]){
       //Tạo file để lưu đối tượng
      ObjectOutputStream out =new
ObjectOutputStream(
            new FileOutputStream("myobject.ser"));
       //Khởi tạo đối tượng trong bộ nhớ
      LinkObject X=new LinkObject(1);
      System.out.println("X is "+X);
       //Ghi đối tượng xuống tập tin
```

```
        out.writeObject(X);
        //In nội dung đối tượng ra màn hình
        System.out.println("Here is object content");
        while ((X=X.next)!=null)
                System.out.println(X);
    }
}
```

Biên dịch

```
javac WriteObjectApp.java
```

Chạy chương trình

```
java WriteObjectApp
```

Kết quả kết xuất

```
X is Object 1
Here is object content
Object 2
Object 3
...
Object 49
Object 50
```

Như bạn thấy khi khởi tạo đối tượng LinkObject, phương thức khởi dựng của LinkObject tự động tạo ra 50 đối tượng con khác liên kết với nhau thành một xâu (danh sách liên kết). Do LinkObject cài đặt giao tiếp Serializable nên nó có khả năng ghi được xuống thành tập tin trên đĩa. Chúng ta sử dụng lớp xuất nhập ObjectOutputStream để ghi toàn bộ đối tượng X thành tập tin mang tên myobject.ser. Thật sự thì 50 đối tượng con liên kết với X đều được lưu vào tập tin myobject.ser.

Giả sử 3 ngày sau bạn cần sử dụng lại đối tượng X. Bạn có thể đọc và khôi phục lại đối tượng từ tập tin myobject.ser như sau:

Ví dụ 2-22 ReadObjectApp.java

```
import java.io.*;
public class ReadObjectApp{
   public static void main(String args[]){
      //Mở tập tin chứa đối tượng
      ObjectInputStream in =new ObjectInputStream(
            new FileInputStream("myobject.ser"));
      //Đọc đối tượng từ tập tin
      LinkObject X=(LinkObject) in.readObject();
      //In nội dung của đối tượng ra màn hình
      System.out.println("Here is X "+X);
      while ((X=X.next)=!null)
                  System.out.println(X);
   }
}
```

Kết quả

```
Here is X Object 1
          Object 2
          Object 3
          ...
          Object 49
          Object 50
```

Như bạn thấy chương trình ReadObjectApp không hề sử dụng lệnh new để tạo đối tượng. ReadObjectApp sử dụng lớp luồng ObjectInputStream để đọc đối tượng lên từ đĩa.

Khả năng tuần tự hoá của đối tượng rất hữu dụng. Hầu như nó làm nền tảng và được dùng xuyên suốt trong cơ chế hoạt động, truyền dữ liệu kiểu đối tượng trong máy ảo Java và giữa những máy ảo Java với nhau trên mạng. Ở mục 7.1 như bạn thấy, để truyền đối tượng đi trên mạng, bắt buộc chúng ta phải xây dựng đối tượng với đặc tính Serializable. Bạn hãy tham khảo giáo trình "Lý thuyết và bài tập Java" của cùng tủ sách Minh Khai (xuất bản 1999) chúng tôi đề cập đến vấn đề tuần tự hóa của đối tượng rất chi tiết.

9. TÌM HIỂU RMIREGISTRY VÀ CÁC CÁCH ĐĂNG KÝ ĐỐI TƯỢNG

Trước giờ chúng ta vẫn sử dụng bộ đăng ký rmiregistry như một dịch vụ tìm kiếm. Các đối tượng chủ muốn trình khách truy tìm được từ xa thì trước hết phải đăng ký với rmiregistry. Bộ đăng ký này là một chương trình dịch vụ chạy ở hậu trường (chương trình rmiregistry.exe) mở ổ cắm socket và lắng nghe các yêu cầu gửi đến cổng mặc định 1099. Bạn có thể chỉ định một số cổng khác với cổng mặc định. Ví dụ:

```
C:\jdk1.3\bin>Rmiregistry.exe  2002
```

sẽ chạy dịch vụ đăng ký rmiregistry trên cổng 2002. Đối tượng trên máy chủ có thể đăng ký đến dịch vụ này bằng cách gọi Naming.bind() theo cách

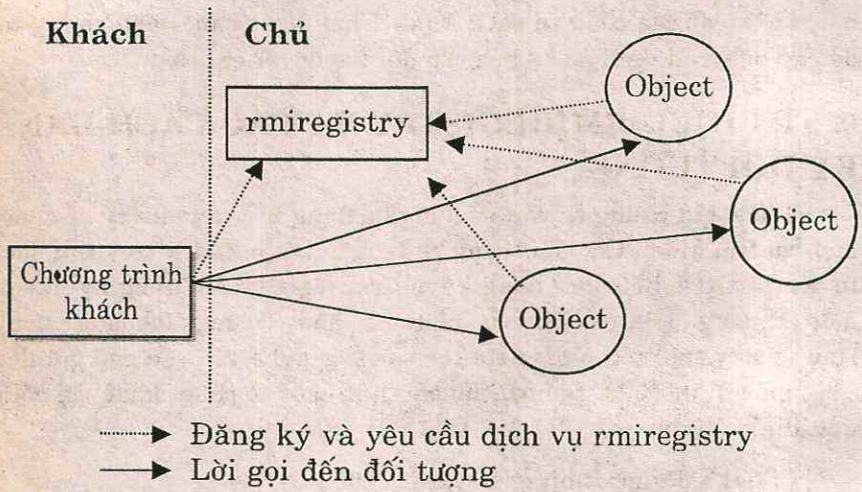
```
MyObject o=new MyObjectImpl();
Naming.bind("rmi://localhost:2002/myobjectname", o);
```

Tiếp đến chương trình trên máy khách sẽ yêu cầu dịch vụ đăng ký trả về tham chiếu đến đối tượng trên máy chủ

```
MyObject o=(MyObject)Naming.lookup(
                "rmi://localhost:2002/myobjectname");
```

Một dịch vụ đăng ký rmiregistry có thể tiếp nhận và quản lý cùng lúc nhiều đối tượng chủ khác nhau (hình 2-9).

Java cho phép cơ chế liên lạc với bộ đăng ký rmiregistry để lấy về danh sách các đối tượng hiện rmiregistry đang nắm giữ để bạn tùy nghi sử dụng. Điều này tương tự một cửa hàng có khả năng cung cấp cho bạn nhiều dịch vụ. Muốn biết mình cần thứ gì bạn chỉ việc yêu cầu cửa hàng cung cấp danh sách các dịch vụ hiện có. Chọn ra một dịch vụ sau đó yêu cầu dịch vụ đó thực hiện chức năng của mình.



Hình 2-9 Cơ chế đăng ký nhiều đối tượng với rmiregistry

9.1. Chương trình duyệt các đối tượng có trong rmiregistry

Dưới đây là chương trình tìm hiểu danh sách tên các đối tượng do rmiregistry nằm giữ. Bạn xây dựng chương trình qua các bước sau:

Bước 1: Xây dựng chương trình duyệt rmiregistry

Ví dụ 2-23 Traverse.java

```
import java.rmi.registry.*;
public class Traverse{
```

```
  public static void main(String args[])
                                        throws Exception{
    //Thay địa chỉ localhost bằng địa chỉ IP (nếu có) nơi
rmiregistry đang chạy
    String hostAddr="127.0.0.1"; //hoặc localhost
    System.out.println("Connecting registry...");
    //Kết nối với bộ đăng ký
    Registry registry=
            LocateRegistry.getRegistry(hostAddr);
    //Lấy về danh sách các đối tượng do rmiregistry nắm giữ
    String objectAvailable[]=registry.list();
    //Liệt kê danh sách các đối tượng
    System.out.println("Registry object: ");
    for (int i=0; i<objectAvailable.length; i++){
      System.out.println(objectAvailable[i]);
    }
  }
}
```

Biên dịch chương trình

```
javac  Traverse.java
```

Bước 2: Khởi động và đăng ký một số đối tượng với rmiregistry.

Khởi động bộ đăng ký

```
start rmiregistry
```

Đăng ký đối tượng Calculator với bộ đăng ký

```
C:\Network\Calculator>java Setup
```

Đăng ký đối tượng Hello với bộ đăng ký

```
C:\Network\Hello>java Setup
```

Đăng ký đối tượng PingServer với bộ đăng ký

```
C:\Network\ByValue>java Setup
```

Bước 3: Chạy chương trình Traverse

```
C:\Network\Traverse> java Traverse
MyCalculator
Myhello
Pingobject
```

Bạn sẽ liệt kê được tổng cộng 3 đối tượng đã đăng ký với rmiregistry. Chương trình của ta sử dụng lớp `LocateRegistry` chứa trong gói `java.rmi.registry`. Bạn cung cấp địa chỉ IP của máy ảo nơi rmiregistry đang chạy, phương thức tĩnh `getRegistry()` của lớp này sẽ trả về đối tượng Registry.

```
Registry registry=
LocateRegistry.getRegistry(hostAddr);
```

Gọi phương thức registry.list() bạn sẽ có trong tay một mảng chứa danh sách tên các đối tượng đã đăng ký với rmiregistry. Công việc còn lại chỉ là in tên của các đối tượng trong danh sách ra màn hình.

9.2. Tự tạo bộ đăng ký cục bộ

Java cho phép bạn tự tạo bộ đăng ký cho riêng mình mà không cần dùng đến chương trình rmiregistry.exe. Để tạo bộ đăng ký và tự đăng ký đối tượng bạn gọi phương thức tĩnh createRegistry() của lớp LocateRegistry. Chẳng hạn chương trình Setup.java ở ví dụ 2-8 có thể tự đăng ký đối tượng Hello theo cách sau:

Ví dụ 2-24 Setup.java

```
import java.rmi.*;
```

```
import java.rmi.server.*;
java.rmi.registry.*
public class Setup{
  public static void main(String args[])
                                    throws Exception{
    //Tự tạo bộ đăng ký registry
    LocateRegistry.createRegistry(1099);
    //Kiến tạo đối tượng
    HelloImpl hello=new HelloImpl();
    //Yêu cầu máy ảo nhận dạng đối tượng
    UnicastRemoteObject.exportObject(hello);
    //Đăng ký (hay ràng buộc) đối tượng với dịch vụ tìm kiếm
Naming
    System.out.println("Registering object ...");
    Naming.bind("rmi://localhost/myhello",hello);
    System.out.println("Waiting for
                                  client request...");
  }
}
```

Bạn chỉ cần biên dịch và gọi chương trình `java Setup` của ví dụ trên từ máy chủ. `Setup` sẽ tự động tạo bộ đăng ký rmiregistry ở cổng mặc định 1099, sau đó tự đăng ký đối tượng với rmiregistry do mình tạo ra. Tuy nhiên cách này chỉ có thể áp dụng cho một đối tượng. Nếu một đối tượng nào đó đã tạo ra bộ đăng ký rồi thì bạn không được gọi `LocateRegistry.createRegistry()` lần thứ hai trên cùng một cổng của máy chủ. Bạn phải tạo một cổng khác cho bộ đăng ký, ví dụ:

```
LocateRegistry.createRegistry(1234);
...
Naming.bind("rmi://localhost:1234/myhello",hello);
```

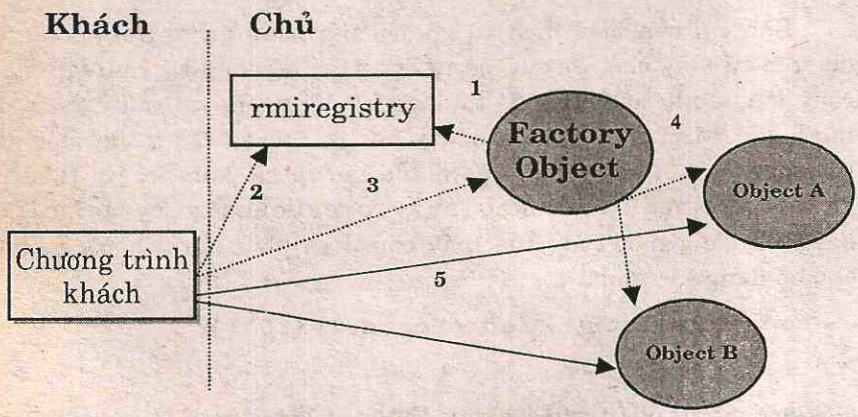
10. DÙNG MỘT ĐỐI TƯỢNG SẢN SINH NHIỀU ĐỐI TƯỢNG (FACTORY OBJECT)

10.1. Mô hình xưởng chế tác đối tượng

Cho đến bây giờ mỗi khi xây dựng đối tượng chủ chúng ta đều phải đặt cho đối tượng một tên. Đăng ký tên đối tượng với rmiregistry. Điểm bất tiện đó là rmiregistry phải quản lý quá nhiều tên đối tượng và bạn với tư cách là nhà phát triển ứng dụng trên máy khách cũng phải nhớ được tên của những đối tượng này. Bạn có thể xây dựng một đối tượng duy nhất và đăng ký đối tượng này với rmiregistry. Nhiệm vụ của đối tượng trên chỉ dùng để tạo ra các đối tượng con khác. Một đối tượng như vậy trong mô hình hướng đối tượng gọi là factory object.

Có thể coi factory object là "xưởng" chế tác và cung cấp các đối tượng con cho máy khách.

Khi muốn yêu cầu tham chiếu hay triệu gọi một đối tượng con nào đó từ máy chủ, bạn chỉ cần nhớ tên của đối tượng factory. Sau khi lấy về tham chiếu của đối tượng factory bạn dùng nó để tạo ra các đối tượng con cần thiết. Hình 2-10 là mô hình hoạt động của đối tượng factory



Hình 2-10 Mô hình hoạt động của đối tượng Factory

1. Đối tượng Factory Object đăng ký tên mình với rmiregistry.
2. Trình khách muốn gọi đối tượng Object A. Trước hết trình khách liên hệ với rmiregistry để lấy về tham chiếu đến đối tượng Factory Object.
3. Sau khi có tham chiếu của Factory Object, trình khách triệu gọi đối tượng Factory Object yêu cầu tạo ra và cho phép tham chiếu đến Object A (hoặc Object B).
4. Đối tượng Factory Object chủ động tạo ra và trả về tham chiếu của Object A, Object B theo yêu cầu của trình khách.
5. Dựa vào tham chiếu do Factory Object trả về trình khách thực hiện lời gọi đến Object A (hoặc Object B).

10.2. Cài đặt ứng dụng Factory

Bạn đã có một khái niệm tổng quát về cách làm việc của đối tượng factory. Để hiểu rõ hơn chúng ta sẽ xây dựng một ứng dụng thật sự theo mô hình này.

Trước hết ứng dụng của ta trên máy chủ bao gồm hai đối tượng: News dùng cung cấp thông tin cho máy khách và Timer dùng để cung cấp thời gian hiện hành. Chúng ta không muốn đăng ký hai đối tượng này với rmiregistry mà muốn một đối tượng FactoryService chịu trách nhiệm đứng trung gian giao tiếp với máy khách để cung cấp đối tượng News và Timer. Với yêu cầu trên, chương trình được xây dựng lần lượt qua các bước sau:

Bước 1: Đặc tả giao tiếp và cài đặt đối tượng Timer

Ví dụ 2-25 Timer.java

```
import java.rmi.*;
import java.util.*;
public interface Timer extends Remote{
 public Date getCurrentTime() throws RemoteException;
```

```
}
```

Ví dụ 2-26 Timerimpl.java

```
import java.rmi.*;
import java.util.*;
public class TimerImpl implements Timer{
 public Date getCurrentTime() throws RemoteException{
    System.out.println("Client request time");
    return new Date();
  }
}
```

Timer của ta là một đối tượng RMI, cung cấp phương thức `getCurrentTime()` có thể được gọi từ xa bởi máy khách. `getCurrentTime()` trả về ngày giờ hiện hành trên máy chủ.

Bước 2: Đặc tả giao tiếp và cài đặt đối tượng News

Ví dụ 2-27 News.java

```
import java.rmi.*;
public interface News extends Remote{
 public String getTodayNews() throws RemoteException;
}
```

Ví dụ 2-28 NewsImpl.java

```
import java.rmi.*;
public class NewsImpl implements News{
  //Các thông tin mà dịch vụ News cung cấp
  String newslist[]={
            "Sun released new Java version",
            "Microsoft announce Windows 2003",
```

```
                "Fujitsu setup new factory in Vietnam"};
  static int index=0;
 public String getTodayNews() throws RemoteException{
    System.out.println("Client request news");
    index++;
    if (index>2) index=0;
     // Cung cấp thông tin cho trình khách
    return newslist[index];
  }
}
```

News của ta là một đối tượng RMI, cung cấp phương thức `getTodayNews()` trả về thông tin cập nhật mới nhất. getTodayNews() lấy thông tin từ mảng `newslist[]`. Trong ứng dụng thực tế, bạn có thể cung cấp thông tin cho trình khách từ cơ sở dữ liệu hay một nguồn tin nào đó. Đối tượng News của ta như bạn thấy chỉ có khả năng cung cấp 3 tin mà thôi.

Do không muốn đăng ký News và Timer với rmiregistry, chúng ta tạo ra đối tượng FactoryService quản lý News và Timer như sau

Bước 3: Đặc tả giao tiếp và cài đặt đối tượng FactoryService

Ví dụ 2-29 FactoryService.java

```
import java.rmi.*;
public interface FactoryService extends Remote{
  //Được trình khách gọi để tạo ra đối tượng News
  public News  createNews() throws RemoteException;
  //Được trình khách gọi để tạo ra đối tượng Timer
  public Timer createTimer() throws RemoteException;
}
```

Ví dụ 2-30 FactoryServiceImpl.java

```
import java.rmi.*;
import java.rmi.server.*;
public class FactoryServiceImpl
                                implements FactoryService{
  //Tạo thể hiện của đối tượng News và Timer
  News  newsobject =new NewsImpl();
  Timer timerobject= new TimerImpl();
  public FactoryServiceImpl(){
     try {
        //Thông báo khả năng triệu gọi từ xa của đối tượng news
và timer
UnicastRemoteObject.exportObject(newsobject);
UnicastRemoteObject.exportObject(timerobject);
     } catch (Exception e){
       System.out.println(e);
     }
  }
  public News  createNews() throws RemoteException{
      return newsobject;
  }
  public Timer createTimer() throws RemoteException{
      return timerobject;
  }
}
```

Trình khách khi giao tiếp với FactoryService sẽ gọi `createNews()` và `createTimer()` để tạo ra đối tượng News và Timer. Phương thức khởi dựng của FactoryService có nhiệm vụ kiến tạo thể hiện của hai đối tượng này đồng thời gọi `UnicastRemoteObject.exportObject()` để thông báo khả năng triệu gọi từ xa của News và Timer cho máy ảo Java.

Bước 4: Xây dựng trình cài đặt đối tượng và đăng ký FactoryService với rmiregistry.

Ví dụ 2-31 Setup.java

```
import java.rmi.*;
import java.rmi.server.*;
public class Setup{
  public static void main(String args[])
                                    throws Exception{
    System.out.println("Factory object created ");
    //Tạo đối tượng FactoryService
    FactoryService obj=new FactoryServiceImpl();
    UnicastRemoteObject.exportObject(obj);
    //Đăng ký FactoryService với rmiregistry
    Naming.bind("rmi://localhost/myfactory",obj);
  }
}
```

Trình Setup.java vẫn như thường lệ chịu trách nhiệm tạo thể hiện của đối tượng FactoryService. Đăng ký đối tượng với rmiregistry.

Trình khách muốn gọi News và Timer chỉ cần nhớ tên đăng ký của FactoryService với rmiregistry là myfactory. Liên hệ với FactoryService trình khách sẽ có được tham chiếu để gọi đến News và Timer. Trình khách Client.java được xây dựng ở bước tiếp theo sau

Bước 5: Xây dựng trình khách triệu gọi đối tượng trên máy chủ

Ví dụ 2-32 Client.java

```
import java.rmi.*;
```

```
import java.rmi.server.*;
public class Client{
  public static void main(String args[])
                                    throws Exception{
    //Liên hệ với đối tượng FactoryService
    FactoryService service=(FactoryService)
Naming.lookup("rmi://localhost/myfactory");
    //Yêu cầu FactoryService cung cấp đối tượng News và Timer
    News news=service.createNews();
    Timer timer=service.createTimer();
    //Triệu gọi phương thức của các đối tượng news và timer
    System.out.println("Current "+
                          timer.getCurrentTime());
    System.out.println("We have news "+
                            news.getTodayNews());
    //Dừng 1 giây
    Thread.sleep(1000);
    //Triệu gọi phương thức của các đối tượng news và timer
    System.out.println("Current "+
                          timer.getCurrentTime());
    System.out.println("We have news "+
                            news.getTodayNews());
  }
}
```

Như bạn thấy trình khách của ta chỉ việc tham chiếu đến FactoryService bằng hàm

```
    FactoryService service=(FactoryService)
```

```
Naming.lookup("rmi://localhost/myfactory");
```

Có đối tượng service trình khách yêu cầu service cung cấp cho mình tham chiếu đến hai đối tượng con News và Timer như sau

```
News news=service.createNews();
Timer timer=service.createTimer();
```

Việc truy xuất news và timer diễn ra không có gì khó khăn lắm. Chương trình Client của ta gọi `getCurrentTime()`, `getTodaysNews()` để lấy về giờ hiện hành và thông tin trên máy chủ.

Bước 6: Biên dịch và chạy chương trình

Lưu tất cả mã nguồn vào thư mục C:\Network\Factory. Chuyển vào thư mục này đánh lệnh biên dịch sau

```
C:\Network\Factory>javac *.java
```

Do News, Timer và FactoryService là các đối tượng phân tán nên ta cần phải tạo ra các lớp trung gian _Stub và Skel cho chúng. Bạn đánh lệnh biên dịch tiếp theo sau:

```
C:\Network\Factory>rmic TimerImpl
C:\Network\Factory>rmic NewsImpl
C:\Network\Factory>rmic FactoryServiceImpl
```

Khởi động bộ đăng ký rmiregistry

```
C:\Network\Factory>start rmiregistry
```

Khởi động trình đăng ký đối tượng FactoryService

```
C:\Network\Factory>start java Setup
```

Chạy trình khách

```
C:\Network\Factory>java Client
```

Kết quả kết xuất trên máy khách

```
Current Mon Jun 27 21:53:56 PDT 1999
```

```
We have news: Microsoft announce Windows 2003
Current Mon Jun 27 21:53:57 PDT 1999
We have news: Fujitsu setup new factory in
Vietnam
```

Kết quả kết xuất trên máy chủ

```
Client request time
Client request news
Client request time
Client request news
```

Mô hình xưởng chế tác đối tượng là cách thức giao tiếp khá thông dụng mà sau này bạn sẽ thấy Java đưa vào mô hình EJB (Enterprise Java Bean) rất nổi tiếng trong các phiên bản Java 1.2 trở đi. Chúng ta sẽ nghiên cứu về EJB trong chương sau. Lúc đó bạn sẽ có dịp tham khảo lại kỹ thuật này.

11. KỸ THUẬT GỌI ĐỐI TƯỢNG Ở XA BẰNG PHƯƠNG THỨC ĐỘNG (DYNAMIC METHOD INVOKE)

Qua các chương trình ví dụ trên ta thấy muốn trình khách gọi được các phương thức của đối tượng chủ thì đối tượng ở xa ít nhất phải cung cấp lớp giao tiếp interface của bản thân đối tượng. Trình khách sẽ dựa vào lớp giao tiếp này để chuyển kiểu đối tượng về dạng tường minh sau khi nhận được tham chiếu đến đối tượng bằng hàm Naming.lookup().

Ví dụ:

```
(Hello) obj = (Hello)Naming.lookup("
                        rmi://localhost/myhello");
```

hay:

```
(PingServer) obj = (PingServer)Naming.lookup("
                                rmi://localhost/myhello");
```

Hello và PingServer là các lớp giao tiếp của đối tượng mà nhà cung cấp phải gửi đến bạn. Cách gọi hàm này được xem là cách gọi hàm tĩnh. Bởi trình khách sau khi chuyển kiểu đối tượng về dạng tường minh thì có thể truy xuất đối tượng ở xa theo cách thông thường. Tuy nhiên nếu bạn không có lớp giao tiếp interface của đối tượng trong tay thì sao? Nghĩa là bạn biết cách lấy về tham chiếu của đối tượng nhưng không có lớp giao tiếp interface của đối tượng để chuyển kiểu. Trường hợp này bạn có thể sử dụng kỹ thuật phản chiếu (reflect) của Java để thực hiện triệu gọi phương thức động. Bạn lấy về tham chiếu của đối tượng bằng hàm Naming.lookup() ở dạng tổng quát Object. Sử dụng đối tượng Object và gọi phương thức getMethod() bạn sẽ biết được các phương thức mà đối tượng hiện đang nằm giữ. Hàm invoke() sẽ giúp bạn thực thi phương thức bên trong đối tượng. Dưới đây là chương trình ví dụ được thực hiện qua các bước sau

Bước 1: Đặc tả giao tiếp cho lớp đối tượng Friend

Ví dụ 2-33 Friend.java

```
import java.rmi.*;
public interface Friend  extends  Remote{
   public String greeting(String name)
                                  throws RemoteException;
}
```

Bước 2: Cài đặt chi tiết cho đối tượng

Ví dụ 2-34 FriendImpl.java

```
import java.rmi.*;
public class FriendImpl implements Friend{
  public String greeting(String name)
                                  throws RemoteException{
```

```
    return ("Hi "+name+ "! Nice to meet you");
  }
}
```

Bước 3: Thiết lập và đăng ký đối tượng trên máy chủ

Ví dụ 2-35 Setup.java

```
import java.rmi.*;
import java.rmi.server.*;
public class Setup {
  public static void main(String args[])
                                    throws Exception{
    Friend obj=new FriendImpl();
    UnicastRemoteObject.exportObject(obj);
    System.out.println("Friend waiting
                              for client request...");
    Naming.bind("rmi://localhost/myfriend", obj);
  }
}
```

Bước 4: Viết chương trình khách

Ví dụ 2-36 Client.java

```
import java.rmi.*;
import java.lang.reflect.*;
public class Client {
  public static void main(String args[])
                                    throws Exception{
    //Tìm tham chiếu của đối tượng
    Object o=Naming.lookup("
                        rmi://localhost/myfriend");
```

```
    //Truy tìm xuất xứ lớp của đối tượng
    Class c = o.getClass();
    //Định kiểu tham số của phương thức cần gọi
Class[] parameterTypes = new Class[] {String.class};
    //Truy tìm phương thức cần gọi
    Method theMethod;
    theMethod = c.getMethod("greeting",
                                        parameterTypes);
    //Định đối số truyền vào phương thức cần gọi
    Object[] arguments = new Object[] {"Johny"};
    //Gọi phương thức và nhận kết quả trả về
    String result = (String) theMethod.invoke(o,
                                        arguments);
    System.out.println(result);
  }
}
```

Bước 5: Biên dịch và chạy chương trình. Bạn lưu tất cả mã nguồn vào thư mục C:\Network\Dynamic. Chuyển vào thư mục này để biên dịch mã nguồn

```
C:\Network\Dynamic>javac *.java
```

- *Biên dịch các lớp trung gian*

```
C:\Network\Dynamic>rmic FriendImpl
```

- *Khởi động bộ đăng ký rmiregistry*

```
C:\Network\Dynamic>start rmiregistry
```

- *Cài đặt đối tượng*

```
C:\Network\Dynamic>start java Setup
```

```
Friend waiting for client request ...
```

- *Gọi đối tượng từ trình khách*

```
C:\Network\Dynamic>start java Setup
Hi Johny, Nice to meet you !
```

Chương trình khách Client.java của ta lấy về tham chiếu đến đối tượng myfriend trên máy chủ chỉ ở dạng tổng quát Object. Chúng ta không sử dụng lớp giao tiếp Friend.class để chuyển kiểu như thường lệ, thay vào đó ta tìm cách "khảo sát" nội dung bên trong lớp đối tượng này. Trước hết bạn dùng lớp Class tổng quát để lấy về xuất xứ của lớp mà đối tượng thể hiện

```
Class c = o.getClass();
```

Tiếp đến, bạn biết rằng đối tượng của ta có chứa phương thức mang tên "greeting". Phương thức này yêu cầu một tham số có kiểu String. Chúng ta cung cấp thông tin này để Java lấy về phương thức cần gọi như sau:

```
//Định kiểu tham số của phương thức cần gọi
Class[] parameterTypes = new Class[] {String.class};
//Yêu cầu Java truy tìm phương thức cần gọi
Method theMethod;
theMethod = c.getMethod("greeting",
                                parameterTypes);
```

Công việc tiếp theo là gọi phương thức. Đối tượng theMethod cung cấp cho bạn phương thức invoke() để gọi phương thức mà nó tham trỏ đến. Trước khi gọi phương thức, chúng ta cần khởi tạo danh sách đối số để truyền vào phương thức như sau:

```
//Định đối số truyền vào phương thức cần gọi
Object[] arguments = new Object[] {"Johny"};
```

Bước cuối cùng là triệu gọi phương thức gián tiếp

```
//Gọi phương thức và nhận kết quả trả về
String result = (String) theMethod.invoke(o,
                                    arguments);
```

phương thức invoke() yêu cầu bạn 2 đối số, đối số thứ nhất là thể hiện của đối tượng chứa phương thức. Đối số thứ hai là danh sách các đối số cần truyền vào phương thức gọi.

Trên đây là cách gọi phương thức động nghĩa là không cần đến lớp giao tiếp của đối tượng từ máy chủ. Kỹ thuật này không những chỉ áp dụng cho đối tượng ở xa mà còn cho mọi đối tượng trong Java. Bạn hãy tham khảo thêm chủ đề "Reflection" trong các tài liệu lập trình của Java về đề tài hấp dẫn này.

Ở ví dụ trên thay cho lời gọi động bạn có thể thu được cùng một kết quả bằng cách gọi tĩnh ngắn gọn sau:

```
Friend o=(Friend) Naming.lookup("
                        rmi://localhost/myfriend");
String result= o.greeting("Johny");
```

Cách gọi động tuy phức tạp nhưng lại uyển chuyển và linh động hơn cách gọi tĩnh.

12. TỰ ĐỘNG KÍCH HOẠT ĐỐI TƯỢNG TỪ XA (ACTIVATION)

Qui trình mà ta vẫn thực hiện từ trước đến giờ khi cài đặt đối tượng trên máy chủ vẫn là:

- Khởi động bộ đăng ký rmiregistry bằng cách gọi chương trình rmiregistry.exe hay gọi hàm LocateRegistry.createRegistry().
- Tạo ra đối tượng trên máy chủ và đăng ký nó với rmiregistry.
- Lắng nghe các yêu cầu hay lời triệu gọi phương thức từ phía máy khách.

Hãy hình dung trên máy chủ bạn có 10,000 đối tượng đều có khả năng phục vụ cho máy khách. Vậy thì để sử dụng được 10,000 đối tượng này, trước tiên bạn phải xin cấp phát bộ nhớ cho tất cả đối tượng. Sau đó mỗi đối tượng đều đặt trong trạng thái sẵn sàng chờ máy khách. Điều này sẽ gây hao tốn tài nguyên của máy chủ bởi vì không phải tất cả các đối tượng đều được máy khách yêu cầu sử dụng vào cùng một thời điểm. Java cung cấp cơ chế tự động kích hoạt (activation) đối tượng trên máy chủ chỉ khi máy khách có yêu cầu sử dụng. Cách hoạt động của cơ chế này là bạn chỉ cần đăng ký sự hiện diện của đối tượng trên máy chủ (không cần phải cấp phát bộ nhớ và đặt đối tượng trong trạng thái sẵn sàng). Java cung cấp chương trình rmid.exe đóng nhiệm vụ tiếp nhận sự hiện diện của đối tượng và tự động khởi tạo đối tượng khi máy khách có yêu cầu. Dưới đây là chương trình ví dụ. Bạn hãy thực hiện cài đặt chương trình qua các bước sau:

Bước 1: Xây dựng lớp giao tiếp của đối tượng mang tên MyAutoObject

Ví dụ 2-37 MyAutoObject.java

```
import java.rmi.*;
public interface MyAutoObject  extends Remote {
   public String callMeRemotely()
                                  throws RemoteException;
}
```

Đối tượng MyAutoObject của ta đơn giản chỉ cung cấp một phương thức callMeRemotely() cho trình khách gọi từ xa.

Bước 2: Cài đặt chi tiết cho đối tượng MyAutoObject thông qua lớp MyAutoObjectImpl.

Ví dụ 2-38 MyAutoObjectImpl.java

```
import java.rmi.*;
```

```
import java.rmi.activation.*;
public class MyAutoObjectImpl
                    extends Activatable
                    implements MyAutoObject {
// Phương thức khởi dựng dùng kích hoạt đối tượng.
//Khi dịch vụ rmid kích hoạt đối tượng nó sẽ gọi phương thức
//ActivationInstantiator.newInstance và gián tiếp gọi đến phương thức
//khởi dựng này.
// id      là định danh của đối tượng cần kích hoạt.
//data     là dữ liệu bên ngoài cần truyền vào khi đối tượng được
kích hoạt
  public MyAutoObjectImpl(ActivationID id,
                          MarshalledObject data)
                          throws RemoteException
{
// Đăng ký và thông báo sự hiện diện của đối tượng có khả năng tự
// kích hoạt với hệ thống
        super(id, 0);
  }
  // Cài đặt phương thức của đối tượng trong giao tiếp
MyAutoObject
  //
  public String callMeRemotely()
                          throws RemoteException {
  return "Success Server Call";
  }
}
```

Khác với cách cài đặt thông thường, đối tượng muốn tự kích hoạt được bởi dịch vụ rmid phải dẫn xuất thêm từ lớp cha Activatable (các lớp hỗ trợ cho kỹ thuật tự kích hoạt nằm trong gói thư viện `java.rmi.acivation.*`) . Phương thức khởi dựng của đối tượng phải được thiết lập như sau:

```
public MyAutoObjectImpl(ActivationID id,
                        MarshalledObject data)
                    throws RemoteException {
    super(id, 0);
}
```

Phương thức này nhận 2 đối số, đối số thứ nhất cho biết định danh kích hoạt (ActivationID) do dịch vụ kích hoạt rmid truyền vào. Đối số thứ hai (data) có kiểu đối tượng MarshalledObject là đối tượng tổng quát có thể truyền qua lại trên mạng (thật sự thì MarshalledObject là đối tượng được cài đặt giao tiếp Serializable) dùng cho mục đích chung khi bạn muốn truyền thông tin từ bên ngoài vào trong quá trình đối tượng khởi động. Bạn phải chuyển những thông tin này lên cho lớp cha bằng cách gọi `super(id,0)`. Lớp cha là lớp Activatable sẽ thông báo và đăng ký sự hiện diện của đối tượng với dịch vụ rmid và máy ảo Java. Bạn sẽ nhận rõ cơ chế hoạt động này trong bước cài đặt trình Setup.java tiếp theo sau đây.

Bước 3: Viết trình cài đặt và đăng ký khả năng kích hoạt của đối tượng.

Ví dụ 2-39 Setup

```
import java.rmi.*;
import java.rmi.activation.*;
import java.util.Properties;
public class Setup {
```

```
// Chương trình đăng ký đối tượng MyAutoObject với
// rmid và rmiregistry
//
public static void main(String[] args)
                                    throws Exception {
//Thiết lập lớp bảo vệ cho trình chủ
System.setSecurityManager(
                        new RMISecurityManager());
// Với mô hình bảo mật của Java 1.2 bạn bắt buộc phải chỉ định
// tập tin chính sách policy quy định quyền cho máy ảo chạy đối
// tượng kích hoạt
// Properties là bảng thuộc tính tổng quát chứa các giá trị cần
thiết lập
Properties props = new Properties();
//Chỉ định tập tin chứa các thiết lập bảo mật
props.put("java.security.policy",
            "file:/C:/network/activate/policy");
//Tạo môi trường chứa các đối tượng kích hoạt
ActivationGroupDesc.CommandEnvironment ace = null;
//Tạo thông tin về nhóm chứa đối tượng kích hoạt
ActivationGroupDesc exampleGroup =
                new ActivationGroupDesc(props, ace);
    //Dựa vào thông tin của nhóm, chúng ta yêu cầu hệ thống
    // cung cấp một định danh kích hoạt cho nhóm
    //
ActivationGroupID agi =
      ActivationGroup.getSystem().registerGroup(
```

```
                                        exampleGroup);
// Sau khi có định danh của nhóm, chương trình yêu cầu hệ thống
//tạo ra nhóm để chứa các đối tượng kích hoạt.
ActivationGroup.createGroup(agi, exampleGroup, 0);
// Chuỗi "location" là địa chỉ URL nơi lớp đối tượng tồn tại
// Khi nhận được yêu cầu từ máy khác. Dịch vụ rmid sẽ dựa vào
// chuỗi này để đi tìm lớp đối tượng trên máy chủ và kích hoạt nó
// Ở đây chúng ta dùng giao thức file. Nếu có web server bạn có thể
// dùng giao thức http. Tuy nhiên nên nhớ kết thúc chuỗi URL bằng
// ký tự xổ trái (/)
   String location = "file:/D:/network/activate/";
// Tạo tham số cần truyền cho đối tượng khi kích hoạt
// Bạn có thể dùng MarshalledObject làm đối tượng mang thông tin
// từ bên ngoài vào đối tượng kích hoạt. Ở đây ta không cần truyền
// thông tin gì cả nên data được đặt trị bằng null
   MarshalledObject data = null;
   // Tạo mô tả về đối tượng MyAutoObjectImpl cần kích hoạt
   ActivationDesc desc = new ActivationDesc (
                    "MyAutoObjectImpl", location, data);
      // Đăng ký đối tượng với dịch vụ kích hoạt rmid
      //
   MyAutoObject mri = (MyAutoObject)
                         Activatable.register(desc);
// Đăng ký tên của đối tượng với rmiregistry để máy khách truy tìm
//
   System.out.println("Register object
                              with rmiregistry");
```

```
Naming.rebind("rmi://localhost/MyAutoObjectImpl",
                                                mri);
System.out.println("Available for client request");
   System.exit(0);
   }
}
```

Bước đăng ký đối tượng với dịch vụ kích hoạt rmid trông có vẻ phức tạp nhưng đó là những bước Java yêu cầu. Các đối tượng cần kích hoạt được quản lý theo từng nhóm. Trước tiên bạn cần chỉ định môi trường mà nhóm sẽ hoạt động.

```
ActivationGroupDesc.CommandEnvironment ace = null;
```

Ở đây ta sử dụng các giá trị mặc định của môi trường Java nên biến ace được đặt trị null.

Điều kế tiếp cần làm là tạo ra các mô tả nhóm

```
ActivationGroupDesc exampleGroup =
                    new ActivationGroupDesc(props, ace);
```

kế đến dựa vào mô tả nhóm bạn yêu cầu hệ thống cấp cho một định danh id dành cho nhóm

```
    ActivationGroupID agi =
        ActivationGroup.getSystem().registerGroup(
                                        exampleGroup);
```

tiếp theo là tạo nhóm (dựa vào định danh nhóm do hệ thống cung cấp)

```
ActivationGroup.createGroup(agi, exampleGroup, 0);
```

sau đó là tạo mô tả về đối tượng cần kích hoạt

```
   ActivationDesc desc = new ActivationDesc (
                    "MyAutoObjectImpl", location, data);
```

bước cuối cùng là đăng ký đối tượng cần kích hoạt

```
MyAutoObject mri = (MyAutoObject)
                    Activatable.register(desc);
```

Quá trình trên sẽ cung cấp thông tin cho chương trình dịch vụ rmid (còn gọi là rmi diamond). Với những thông tin này khi nhận được yêu cầu từ máy khách, rmid sẽ tự động yêu cầu máy chủ tạo và kích hoạt đối tượng để máy khách tham chiếu đến.

Chúng ta vẫn phải thực hiện bước cuối cùng là đăng ký tên đối tượng với rmiregistry

```
Naming.rebind("rmi://localhost/MyAutoObjectImpl",
                                        mri);
```

Như bạn thấy đối tượng mri trong phương thức `Naming.rebind()` không cần tạo ra từ lệnh new mà có được là do đăng ký với dịch vụ rmid.

```
MyAutoObject mri = (MyAutoObject)
                    Activatable.register(desc);
```

Khi bạn chạy chương trình Setup này, lệnh Naming.rebind() không còn dừng lại để chờ yêu cầu từ máy khách gửi đến. Nó chỉ thông báo đến rmid là đối tượng đã sẵn sàng, sau đó thi hành lệnh kế tiếp và chấm dứt chương trình. Khi rmid cần dùng đến đối tượng, rmid sẽ tự động kích hoạt và tạo đối tượng bằng lệnh new, sau đó đưa đối tượng vào sử dụng.

Bước 4: Viết chương trình khách triệu gọi đối tượng từ xa

Ví dụ 2-40 Client.java

```
import java.rmi.*;
public class Client {
  public static void main(String args[]) {
  //Thay localhos bằng địa chỉ IP thích hợp của bạn nếu có
```

```
    String server = "localhost";
    if (args.length > 1) {
      server = args[0];
    }
        // Thiết lập lớp phòng vệ cho trình khách
        //
    System.setSecurityManager(
                                new RMISecurityManager());
    try {
      String location = "rmi://" + server +
                                "/MyAutoObjectImpl";
          // Truy tìm đối tượng
        //
      MyAutoObject mri = (MyAutoObject)
                                Naming.lookup(location);
    //Triệu gọi phương thức của đối tượng và in kết quả trả về
    result = (String)mri.callMeRemotely();
    System.out.println("Returned from remote call");
    System.out.println("Result: " + result);
    } catch (Exception e) {
      e.printStackTrace();
    }
    }
}
```

Thiết kế chương trình phía máy khách không có gì khác so với khi không dùng chế độ kích hoạt tự động. Trình khách hoàn toàn không biết được cơ chế tự động kích hoạt của trình chủ diễn ra như

thế nào. Theo cách thông thường, trình khách gọi Naming.lookup() để lấy về tham chiếu và gọi trực tiếp đến phương thức callMeRemotely() của đối tượng chủ. Trước khi thực hiện biên dịch, bạn cần tạo tập tin policy chứa các qui định bảo mật áp dụng khi chạy chương trình như sau:

Ví dụ 2-41 Nội dung tập tin policy

```
grant {
        // Không hạn chế quyền truy xuất
   permission java.security.AllPermission;
};
```

Đây chỉ là tập tin văn bản bình thường. Bạn có thể dùng trình soạn thảo notepad của Windows để chỉnh sửa và lưu tập tin dưới tên policy.

Bước 5: Biên dịch và chạy chương trình. Bạn lưu tất cả mã nguồn vào thư mục C:\Network\activate. Chuyển vào thư mục này, để biên dịch và chạy chương trình theo thứ tự sau:

Biên dịch các tập tin .class

```
C:\Network\activate>javac *.java
```

Tạo lớp trung gian

```
C:\Network\activate>rmic MyAutoObjectImpl
```

Khởi động bộ đăng ký registry

```
C:\Network\activate>start rmiregistry
```

Khởi động bộ quản lý kích hoạt đối tượng

```
C:\Network\activate>start rmid
        -J-Djava.security.policy=
            file:/C:/network/activate/policy
```

(với tham số -J trình rmid sẽ chuyển toàn bộ tham số -D qua cho máy ảo java tiếp nhận. Ở đây ta yêu cầu rmid khởi động dựa vào chính sách bảo mật quy định trong tập tin policy)

Khởi động trình cài đặt Setup đăng ký đối tượng

```
java -Djava.security.policy=
        file:/C:/network/activate/policy
    -Djava.rmi.server.codebase=
        file:/C:/network/activate/
    Setup
```

(Ở đây trình Setup khởi động dựa vào chế độ bảo mật qui định trong tập tin policy. Tiếp đến là thuộc tính -Djava.rmi.codebase yêu cầu java truy tìm các tập tin cần thiết ở thư mục C:/Network/activate/)

Khởi động trình khách để triệu gọi đối tượng

```
java -Djava.security.policy=
    file:/C:/network/activation/policy Client
```

Tuy cơ chế tự động kích hoạt đối tượng hơi phức tạp nhưng hầu hết các hệ thống lập trình phân tán đối tượng đều hỗ trợ cơ chế này. Chúng tiết kiệm tài nguyên của hệ thống và làm cho máy chủ hoạt động hiệu quả hơn.

13. TRÌNH MỒI NẠP ỨNG DỤNG TỪ XA

Điều thú vị của hệ thống mạng là cho phép bạn kết nối giao tiếp và điều khiển từ xa. Một câu hỏi đặt ra là có cách nào viết một ứng dụng hoàn chỉnh đặt trên máy chủ sau đó gọi ứng dụng chạy từ máy khách hay không? Nói chung là phía máy khách của bạn không cần phải cài đặt gì cả!.

Với cơ chế RMI gọi hàm từ xa mà Java cung cấp, bạn hoàn toàn có thể thực hiện được ý tưởng này. Trong Java bộ nạp lớp

RMIClassLoader cho phép lập trình viên nạp về một lớp chương trình trên máy chủ theo giao thức http. RMIClassLoader cũng chịu trách nhiệm đem về tất cả các lớp con hay lớp thư viện mà lớp chương trình cần đến. Điều này tương tự cơ chế làm việc của Applet. Trong trường hợp Applet, trình duyệt Browser chính là bộ nạp lớp dễ thấy nhất. Tất cả những lớp thư viện hay lớp con mà Applet dùng đến đều được đặt trên máy chủ và được trình duyệt tự động download về đầy đủ trước khi gọi Applet thực thi.

Dưới đây là một chương trình ví dụ triệu gọi ứng dụng từ xa. Ứng dụng của ta được thiết kế theo mô hình phân tán, ứng dụng sẽ gọi phương thức của một đối tượng ở xa. Đối tượng này là một từ điển có trách nhiệm dò tìm và trả về ý nghĩa của một từ do ứng dụng yêu cầu. Toàn bộ ứng dụng được đặt trên web server của một máy chủ. Chương trình khách chỉ cần dùng lớp RMIClassLoader để triệu gọi ứng dụng từ web server.

13.1.1. Thiết kế chương trình trên máy chủ:

Giả sử địa chỉ IP của máy chủ là 172.16.11.12 (bạn có thể thay bằng 127.0.0.1 nếu thử nghiệm trên một máy đơn) hoặc địa chỉ IP do người quản trị mạng cấp cho bạn.

Thư mục gốc của webserver: C:\jswdk-1.0.1\webpages\ bạn vào thư mục này tạo thêm thư mục con DictApp để chứa chương trình ứng dụng. Đường dẫn đầy đủ là C:\jswdk-1.0.1\webpages\DictApp. Đường dẫn URL tương ứng là http://172.16.11.12/DictApp/ (hoặc http://127.0.0.1/DictApp/ nếu bạn thử nghiệm trên máy cục bộ)

Dưới đây là mã nguồn của các lớp cài đặt trên máy chủ.

Ví dụ 2-42 Dictionary.java

```
import java.rmi.*;

/* Lớp giao tiếp đặc tả các phương thức của đối tượng Dictionary */
public interface Dictionary extends Remote{
```

```
  public String findWord(String keyword)
                                    throws RemoteException;
}
```

Ví dụ 2-43 DictionaryImpl.java

```
import java.rmi.*;
/* Lớp cài đặt đối tượng Dictionary */
public class DictionaryImpl implements Dictionary{
  String wordlist[];    //Danh sách các từ cần tra
  String meaning[];     //Danh sách chứa ý nghĩa tương ứng của từ
  public DictionaryImpl(){
     wordlist=new String[3];
     meaning =new String[3];
     //Khởi tạo danh sách từ điển
     wordlist[0]="one";  meaning[0]="mot";
     wordlist[1]="two";  meaning[1]="hai";
     wordlist[2]="three"; meaning[2]="ba";
     // ... Thêm dữ liệu vào danh sách khi bạn xây dựng một từ
điển thật sự
  }
  //Phương thức này được triệu gọi từ xa để tra từ
  public String findWord(String keyword)
                                    throws RemoteException{
   for (int i=0; i<wordlist.length; i++) {
     System.out.println("Lookup word "+wordlist[i]);
       if (keyword.equals(wordlist[i]))
                                    return meaning[i];
```

```
    }
    return "word not found";
  }
}
```

Ví dụ 2-44 DictionaryServer.java

```
import java.rmi.server.*;
import java.rmi.*;
import java.rmi.registry.*;
/* Lớp kiến tạo và đăng ký đối tượng Dictionary với rmiregistry */
public class DictionaryServer{
  public static void main(String args[]) {
    try{
      Dictionary d=new DictionaryImpl();
      UnicastRemoteObject.exportObject(d);
      //Thay địa chỉ IP của máy chủ bằng địa chỉ của máy bạn
nếu có
      Naming.bind("//172.16.11.12/mydictionary",d);
      System.out.println("Registered Dictionary");
      //Nếu bạn chạy trên máy đơn cục bộ thì có thể dùng lệnh sau
      //Naming.bind("//127.0.0.1/mydictionary",d);
    }catch (Exception e){
      System.out.println(e);
    }
  }
}
```

Ví dụ 2-45 DictionaryClient.java

```
import java.rmi.*;
```

```
/* Lớp chương trình gọi và sử dụng đối tượng Dictionary */
public class DictionaryClient implements Runnable{
  public static void main(String args[])
                                        throws Exception{
    System.setSecurityManager(new
                                  RMISecurityManager());
    //Truy tìm đối tượng từ rmiregistry
    Dictionary dict=(Dictionary)Naming.lookup(
                  "rmi://172.16.11.12/mydictionary");
    //Triệu gọi phương thức của đối tượng
    System.out.println(dict.findWord("three"));
  }
  //Phương thức này sẽ giúp cho các máy ở xa có thể nạp chương
trình về chạy
  public void run(){
    main(null);
  }
}
```

Biên dịch 4 tập tin này bạn sẽ thu được 4 lớp tương ứng

```
Dictionary.class
DictionaryImpl.class
DictionaryServer.class
DictionaryClient.class
```

Sử dụng trình biên dịch rmic để tạo ra 2 lớp trung gian _Stub và _Skel cho đối tượng Dictionary như sau:

```
rmic DictionaryImpl
```

bạn sẽ thu thêm được 2 lớp

```
DictionaryImpl_Stub.class
DictionaryImpl_Skel.class
```

Bạn thực hiện theo các bước thứ tự sau

Bước 1: Đặt tất cả 6 lớp này vào thư mục

C:\jswdk-1.0.1\webpages\DictApp

với C:\jswdk-1.0.1\webpages được coi là đường dẫn bắt đầu (webroot) của webserver. Khởi động webserver JSWDK từ dòng lệnh:

```
C:\jswdk-1.0.1>startserver.bat
```

Bước 2: Khởi động rmiregistry từ dòng lệnh

```
C:\jswdk-1.0.1>rmiregistry.exe
```

Bước 3: Khởi động chương trình đăng ký đối tượng Dictionary

```
C:\jswdk-1.0.1\webpages\DictApp>java
  -Djava.rmi.server.codebase="http://203.162.42.189/
                                DictApp/"DictionaryServer
```

Quá trình chuẩn bị phía máy chủ đã hoàn tất. Bạn có thể gọi chương trình DictionaryClient

```
C:\jswdk-1.0.1\webpages\DictApp>java DictionaryClient
```

để kiểm tra kết quả kết xuất. Tuy nhiên DictionaryClient không nhằm chạy trên máy chủ. Nó có thể được triệu gọi từ máy khách ở xa. Có một điểm bạn cần lưu ý đó là DictionaryClient cài đặt giao tiếp Runnable. Vâng, chính giao tiếp này sẽ cho phép bạn gọi được phương thức run() nhằm khởi động chương trình từ máy khách ở xa. Cách thức triệu gọi DictionaryClient ở máy khách sẽ nhờ vào bộ nạp lớp RMIClassLoader được đề cập ngay sau đây.

13.1.2. Thiết kế chương trình gọi ứng dụng phía máy khách

Trên máy khách bạn không cần bất kỳ lớp hỗ trợ nào cả. Điều bạn cần biết là địa chỉ URL trỏ đến nơi lớp ứng dụng DictionaryClient

đang nằm trên máy chủ webserver. Lưu mã nguồn của ví dụ 2-46 dưới đây vào thư mục bất kỳ.

Ví dụ 2-46 BootApp.java

```
import java.rmi.*;
import java.rmi.server.*;
public class BootApp{
  public static void main(String args[])
                                  throws Exception{
    //Thiết lập lớp phòng vệ cho phía máy khách
    System.setSecurityManager(new
                              RMISecurityManager());
    //Nạp lớp ứng dụng DictionaryClient từ máy chủ
    Class clientclass=RMIClassLoader.loadClass(
        "http://172.16.11.12/DictApp/",
                                  "DictionaryClient");
    //Khởi tạo thể hiện của ứng dụng thông qua giao tiếp
Runnable
   Runnable c= (Runnable)clientclass.newInstance();
    //Chạy ứng dụng ngay trên máy khách
    c.run();
  }
```

Biên dịch chương trình

```
C:\TESTING>javac BootApp.java
```

Triệu gọi ứng dụng

```
C:\TESTING> java BootApp
```

Thông thường BootApp dùng cho mục đích chung là khởi động ứng dụng từ xa. Java gọi cơ chế này là cơ chế mồi (bootstrap). Chương trình mồi có thể được viết lại tổng quát hơn như sau:

Ví dụ 2-47 BootstrapApp.java

```
import java.rmi.*;
import java.rmi.server.*;
public class BootstrapApp{
  public static void main(String args[])
                                  throws Exception{
    if (args.length!=2){
      System.out.println("Syntax:");
      System.out.println("java
              BoottrapApp <URL>  <Appclassname>");
      return;
    }
    //Thiết lập lớp phòng vệ cho phía máy khách
    System.setSecurityManager(new
                              RMISecurityManager());
    //Nạp lớp ứng dụng từ máy chủ
    Class clientclass=RMIClassLoader.loadClass(
                                    args[0],args[1]);
    //Khởi tạo thể hiện của ứng dụng thông qua giao tiếp Runnable
   Runnable c= (Runnable)clientclass.newInstance();
    //Chạy ứng dụng ngay trên máy khách
    c.run();
  }
```

Bạn có thể sử dụng BootstrapApp để gọi DictionaryClient từ xa như sau:

```
C:\TESTING>java BootstrapApp
```

```
http://172.16.11.12/DictApp/    DictionaryClient
```

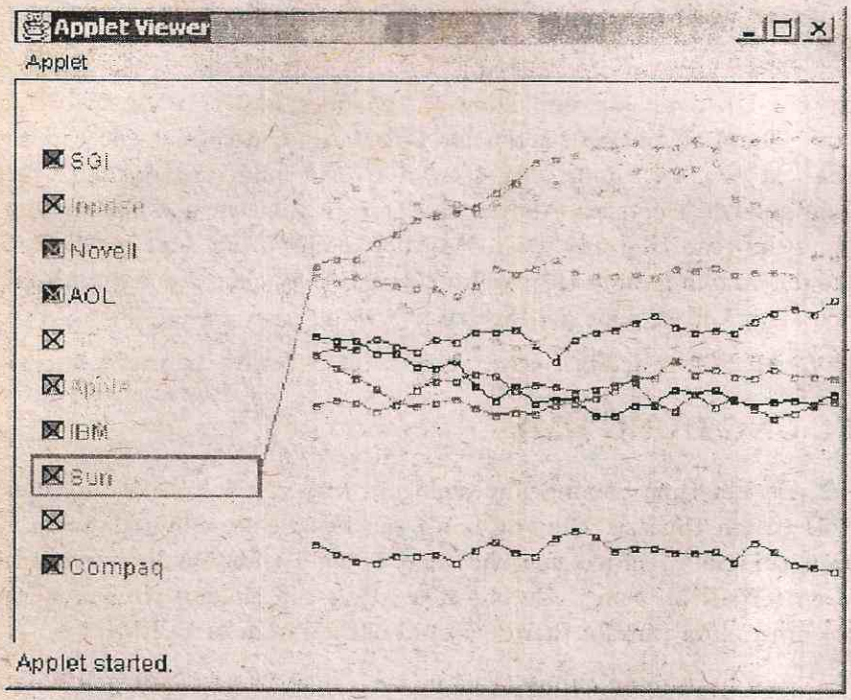
BootstrapApp cần 2 tham số dòng lệnh. Tham số thứ nhất là địa chỉ URL nơi lớp ứng dụng có thể được nạp về bằng giao thức http. Tham số thứ hai là tên lớp ứng dụng. Chỉ có một yêu cầu duy nhất là lớp ứng dụng đặt trên máy chủ phải cài đặt interface Runnable để BootstrapApp có thể chuyển kiểu và gọi phương thức run() của giao tiếp này khởi động đối tượng. Tổng quát hơn bạn có thể đặt mọi ứng dụng trên webserver của máy chủ, chương trình mồi BootstrapApp sẽ được gọi theo cú pháp sau:

```
java BootstrapApp <đường dẫn URL> <tên lớp ứng dụng>
```

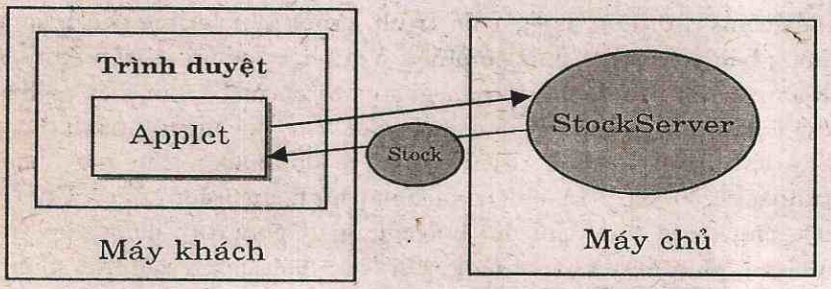
14. ỨNG DỤNG RMI

Để kết thúc chương này chúng ta hãy xây dựng một ứng dụng RMI có ích thực sự. Chương trình của ta có chức năng cung cấp và hiển thị giá cổ phiếu của các công ty máy tính lớn hiện nay như Microsoft, IBM, Sun ... Cơ chế hoạt động của chương trình dựa vào mô hình phân tán đối tượng và kỹ thuật gọi hàm từ xa RMI.

Bạn hình dung chúng ta có một đối tượng ở xa chịu trách nhiệm cung cấp giá cổ phiếu hiện hành của tất cả công ty trên thế giới. Máy khách là một chương trình Applet chạy trên trình duyệt Browse. Applet sẽ gửi yêu cầu đến đối tượng chủ yêu cầu mỗi giây phải cung cập nhật lại giá để trình khách (applet) có thể hiển thị thông báo đến người mua cổ phiếu. Với phác thảo như trên bạn thấy chương trình của chúng ta gần giống với chương trình mẫu callback ở ví dụ 2-15. Vâng, đúng là chúng ta sẽ thiết kế chương trình ở dạng máy chủ có thể tham chiếu ngược về đối tượng trên máy khách (callback). Ví dụ 2-15 chỉ là một mô hình mẫu đơn giản. Thiết kế một ứng dụng thực thụ đòi hỏi chúng ta phải xây dựng giao diện applet ở phía máy khách đồng thời tính toán chính xác thời gian để gửi dữ liệu từ máy chủ về cho máy khách ở xa. Hình 2-11 là giao diện Applet của chương trình phía máy khách còn hình 2-12 là cơ chế giao tiếp giữa trình khách và đối tượng trên máy chủ.



Hình 2-11 Biểu đồ cập nhật giá cổ phiếu của các công ty trên applet



Hình 2-12 Cơ chế truyền thông tin về giá cổ phiếu giữa trình khách và chủ

Chúng ta sẽ lần lượt xây dựng chương trình qua các bước sau:

Bước 1: Đầu tiên bạn hình dung chúng ta cần phải tạo một đối tượng có khả năng chuyển thông tin qua lại giữa trình khách và trình chủ. Chúng ta gọi đối tượng này là Stock.

Đối tượng Stock chứa thông tin về giá cổ phiếu và tên của công ty bán ra cổ phiếu đó. Stock được cài đặt như sau:

Ví dụ 2-48 Stock.java

```
import java.util.*;
/**
 * Đối tượng Stock dùng để chứa thông tin cập nhật từ máy chủ gửi về
 * Do phải di chuyển  trên mạng nên lớp Stock cần cài đặt
 * giao tiếp  java.io.Serializable. Khi đó trong các lời gọi hàm RMI
 * Stock sẽ được truyền theo trị.
 */
public class Stock implements java.io.Serializable {
   String symbol;
   float current;
   private static Random random = new Random();
   private final static float MAX_VALUE = 67;
   /**
    * Tạo đối tượng Stock dựa vào tên của công ty và chọn ngẫu nhiên
    * giá cổ phiếu cho công ty đó
    */
   public Stock(String name) {
   symbol = name;
         // Tạo giá cổ phiếu ngẫu nhiên trong khoảng từ 20-60
   current = (float)(Math.abs(random.nextInt()) % 40
                                                    + 20);
```

```
  }
  /**
   * Cập nhật lại thay đổi của giá cổ phiếu.
   */
  public float update() {
float change = ((float)(random.nextGaussian() * 1.0));
  change = Math.abs(change); // Thay đổi này tùy bạn quyết định
  float newCurrent = current + change;
      // Không cho phép giá cổ phiếu vượt quá mức trần
  if (newCurrent < 0 || newCurrent > MAX_VALUE)
    change = 0;
  current += change;
  return change;
  }
}
```

Chúng ta sử dụng đối tượng Random của gói java.util để tạo số ngẫu nhiên mô phỏng sự biến động của giá cổ phiếu. Phương thức update() trong Stock dùng để lấy về thay đổi giá cổ phiếu của một công ty. Trên thực tế bạn có thể thiết kế hàm update() dựa vào cơ sở dữ liệu hoặc lấy thông tin từ thị trường chứng khoán.

Bước 2: Đặc tả giao tiếp của đối tượng StockNotify phía máy khách. Đối tượng StockNotify là một đối tượng có khả năng tham chiếu xa. Nó giúp cho trình chủ có thể chủ động gọi ngược và truyền thông tin về trình khách (bạn xem lại kỹ thuật callback ở mục 7-4).

Ví dụ 2-49 StockNotify.java

```
import java.rmi.*;
import java.util.*;
/**
```

```
 * StockNotify dùng để nhận các thông tin cập nhật về giá cổ phiếu
do đối tượng
 * trên máy chủ tạo ra
 */
public interface StockNotify extends java.rmi.Remote {
  /**
 * Trình chủ sẽ gọi phương thức update() này ngay trên máy chủ để
 * truyền thông tin về cho trình khách
 * date    Thời gian mới nhất giá cổ phiếu được cập nhật
 * stocks  Danh sách cổ phiếu của các công ty đã đăng ký với trình
 * chủ yêu cầu cập nhật.
 */
   void update (Date date, Stock[] stocks)
                                throws RemoteException;
}
```

Máy khách sẽ thực hiện đăng ký danh sách các cổ phiếu cần cập nhật. Đối tượng chủ sẽ dựa vào danh sách này để gửi giá cổ phiếu về cho máy khách cùng với thời gian cập nhật mới nhất. (Chúng ta sẽ thực hiện cài đặt cho đối tượng StockNotify chung với trình khách Applet ở bước 5).

Bước 3: Đặc tả giao tiếp của đối tượng chủ

Ví dụ 2-50 StockWach.java

```
import java.rmi.*;
/**
 * Đối tượng StockWatch dùng để tiếp nhận đăng ký đối tượng StockNotify
 * từ trình khách. Đồng thời cập nhật giá cổ phiếu cho trình khách định kỳ
 */
public interface StockWatch extends java.rmi.Remote {
```

```
/**
 * Tiếp nhận tham chiếu đến đối tượng StockNotify trên máy khách
 * stock    Tên cổ phiếu
 * obj      đối tượng StockNotify cập nhật trình khách
 * Trả về giá cổ phiếu mới nhất
 */
public Stock watch(String stock, StockNotify obj)
        throws RemoteException;
/**
 * Hủy bỏ yêu cầu cập nhật giá cổ phiếu của một công ty nào đó.
 * stock    Tên cổ phiếu cần hủy bỏ
 * obj      Đối tượng trình khách yêu cầu hủy bỏ cổ phiếu
 */
public void cancel(String stock, StockNotify obj)
                                    throws RemoteException;
/**
 * Trả về danh sách các cổ phiếu đã đăng ký được cập nhật giá
 * obj   Đối tượng trình khách yêu cầu cập nhật giá
 */
public Stock[] list(StockNotify obj)
                                    throws RemoteException;
/**
 * Hủy bỏ tất cả yêu cầu cập nhật giá
 * obj   Đối tượng trình khách yêu cầu hủy bỏ quá trình cập nhật
 */
public void cancelAll(StockNotify obj)
                                    throws RemoteException;
}
```

Bước 4: Cài đặt chi tiết cho đối tượng chủ. Để rút gọn chúng ta cài đặt giao tiếp StockWatch chung với trình thiết lập và đăng ký đối tượng gọi tên là StockServer.

Ví dụ 2-51 StockServer.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.LocateRegistry;
import java.util.*;
public class StockServer
            implements StockWatch, Runnable
{
  /** Tạo bảng lưu giữ các đối tượng StockNotify của máy khách */
  private Hashtable notifyTable = new Hashtable();
  /** Tạo bảng lưu trữ cổ phiếu của các công ty */
  private Hashtable stockTable = new Hashtable();
  /** Tuyến chịu trách nhiệm cập nhật giá trị cổ phiếu sau một
thời gian chỉ định */
  private Thread notifier = null;
  /* Danh sách các công ty bán cổ phiếu */
  private static String name[] =
      { "Sun", "HP", "Microsoft", "Compaq", "Novell",
       "IBM", "Apple", "AOL", "Inprise", "SGI"};
  /**
   * Phương thức khởi dựng của StockServer
   */
  public StockServer() throws RemoteException
  {
```

```
for (int i=0; i<name.length; i++) {
    // Tạo danh sách các cổ phiếu tương ứng với danh sách các
công ty
    stockTable.put(name[i], new Stock(name[i]));
}
}
/**
 * Trả về đối tượng cổ phiếu Stock cho máy khách với giá cập
nhật mới nhất.
 * stock    Tên cổ phiếu
 * obj      Đối tượng trình khách yêu cầu cập nhật giá.
 */
public Stock watch(String stock,StockNotify obj)
                        throws RemoteException
{
System.out.println("StockServer.watch: " + stock )
  //Lấy về danh sách các cổ phiếu
  Vector stocks = (Vector)notifyTable.get(obj);
    // Nếu chưa có thì tạo danh sách cổ phiếu mới
  if (stocks == null) {
    stocks = new Vector();
    notifyTable.put(obj, stocks);
  }
// Nếu chưa có cổ phiếu mà trình khách yêu cầu thì thêm cổ phiếu
// mới vào danh sách
  if (!stocks.contains(stock)) {
    stocks.addElement(stock);
  }
```

```
        //Khởi động tuyến theo dõi và cập nhật giá cổ phiếu.
    if (notifier == null) {
      notifier = new Thread(this, "StockNotifier");
      notifier.start();
    }
    return (Stock)stockTable.get(stock);
    }
    /**
    * Hủy bỏ yêu cầu cập nhật giá cổ phiếu
    * stock   Tên cổ phiếu
    * obj     Đối tượng trình khách yêu cầu cập nhật giá.
    */
    public void cancel(String stock, StockNotify obj)
    {
     Vector stocks = (Vector)notifyTable.get(obj);
     stocks.removeElement(stock);
    }
    /**
    * Trả về danh sách các cổ phiếu trong bảng giá cổ phiếu trên máy chủ
    * obj   Đối tượng trình khách yêu cầu cập nhật giá.
    */
    public Stock[] list(StockNotify obj)
    {
     Vector stocks = (Vector)notifyTable.get(obj);
     Stock[] stockList = null;
       if (stocks != null) {
       Enumeration enum = stocks.elements();
```

```
    stockList = new Stock[stocks.size()];
    int i=0;
        //Trích ra danh sách các cổ phiếu từ bảng cổ phiếu
while (enum.hasMoreElements()) {
String stockname = (String)enum.nextElement();
stockList[i++] = (Stock)stockTable.get(stockname);
    }
  }
  //Trả về danh sách cổ phiếu
  return stockList;
  }
  /**
   * Hủy bỏ việc cập nhật của tất cả các cổ phiếu
   */
 public synchronized void cancelAll(StockNotify obj)
  {
      notifyTable.remove(obj);
  }
  /**
   * Sinh giá ngẫu nhiên cho từng cổ phiếu
   */
  private void generateUpdates()
  {
  Enumeration enum = stockTable.elements();
  while (enum.hasMoreElements()) {
    Stock stock = (Stock)enum.nextElement();
    stock.update();
```

```
}
}
/**
 * Tạo tuyến định kỳ cập nhật giá cổ phiếu cho những công ty
 * được phía máy khách yêu cầu
 */
public void run()
{
boolean done = false;
do {
  try {
          // Cập nhật dữ liệu 2 giây 1 lần
  Thread.currentThread().sleep(2000);
  } catch (InterruptedException e) {
  }
  //Ghi nhận ngày giờ giá cổ phiếu thay đổi
  Date date = new Date();
      // Tạo giá mới cho cổ phiếu
  generateUpdates();
      //Yêu cầu trình khách nhận dữ liệu đã cập nhật
Enumeration enum = notifyTable.keys();
while (enum.hasMoreElements()) {
StockNotify obj = (StockNotify)enum.nextElement();
    Stock[] stockList = list(obj);
    if (stockList != null) {
          // Gửi số liệu đã cập nhật
      try {
```

```
        System.out.println("StockServer.run:
                                sending update " +date);
        obj.update(date, stockList);
        } catch (RemoteException e) {
          e.printStackTrace();
          cancelAll(obj);
        }
      }
      }
    // Kiểm tra xem còn giá cổ phiếu trong danh sách
    // cần cập nhật hay không
      if (notifyTable.isEmpty()) {
        notifier = null;
        done = true;
      }
    } while (!done);
    }
  /**
  * Chương trình chính: kiến tạo đối tượng chủ; Tự tạo bộ đăng ký registry
  * Đăng ký và ràng buộc đối tượng chủ với registry
  */
    public static void main(String args[])
    {
    // Cài đặt lớp bảo vệ cho máy chủ
    System.setSecurityManager(new
                                RMISecurityManager());
    try {
```

```
        // Tự tạo bộ đăng ký registry hoạt động trên cổng 1234
        System.out.println("StockServer.main:
                                        creating registry");
        LocateRegistry.createRegistry(1234);
        //Tạo đối tượng chủ
        System.out.println("StockServer.main:
                                        creating server");
        StockServer server = new StockServer();
        //Thông báo khả năng gọi được từ xa của đối tượng chủ với
máy ảo java
        UnicastRemoteObject.exportObject(server);
        //Ràng buộc đối tượng chủ với registry ở cổng 1234
        System.out.println("StockServer.main:
                                        binding server ");
        Naming.rebind("//localhost:1234/StockServer",
                                        server);
        System.out.println("StockServer.main: done");
    } catch (Exception e) {
        System.out.println("StockServer.main:
                                an exception occurred: " +
                                e.getMessage());
        e.printStackTrace();
    }
    }
}
```

Ngoài việc cài đặt giao tiếp StockWatch, StockServer cài đặt thêm giao tiếp Runnable. Giao tiếp Runnable được dùng với mục đích

tạo ra tuyến (thread) chạy độc lập định kỳ 2 giây 1 lần tính toán giá cổ phiếu mới và yêu cầu đối tượng StockNotify ở trình khách cập nhật lại thông tin.

Quá trình cài đặt đối tượng ở máy chủ đã hoàn tất. Chúng ta sẽ thiết kế một Applet ở máy khách (hay trình khách) có khả năng triệu gọi phương thức của StockServer trên máy chủ.

Bước 5: Xây dựng Applet và cài đặt đối tượng StockNotify.

Ví dụ 2-52 StockApplet.java

```
import java.applet.Applet;
import java.awt.*;
import java.net.URL;
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
/**
 * Bạn lưu ý StockApplet của ta vừa là Applet vừa là StockNotify. Vì
 * StockNotify được tham chiếu bởi đối tượng StockServer ở xa nên
 * ta cần gọi lệnh UnicastRemoteObject.exportObject() để thông báo
 * với máy ảo Java sự hiện diện của StockNotify
 */
public class StockApplet extends Applet
                                    implements StockNotify {
    /** Tổng số cổ phiếu tối đa cần hiển thị */
    static final int MAX_UPDATES = 34;
    /** Chiều rộng của bảng vẽ biểu đồ báo giá */
    private static final int width = 500;
    /** Chiều cao của bảng vẽ biểu đồ báo giá */
```

```
private static final int height = 350;
/** Mảng chứa các thời điểm cập nhật */
private Vector time = null;
/** Mảng chứa các hộp checkbox đánh dấu giá cổ phiếu có cần
cập nhật hay không */
private Vector checkboxes = null;
/** Bảng ánh xạ tên cổ phiếu với đối tượng Stock */
private Hashtable stockTable = null;
/** Tham chiếu đến đối tượng chủ */
private StockWatch stockWatch = null;
/* Danh sách các công ty bán cổ phiếu cần niêm yết */
private static String name[] =
    { "Sun", "HP", "Microsoft", "Compaq", "Novell",
      "IBM", "Apple", "AOL", "Inprise", "SGI"};
/* Màu sắc để vẽ biểu đồ */
private Color color[]= { new Color(205,92,92),
        Color.orange,
        new Color(220,220,90),
        new Color(85, 107, 47),
        Color.blue,
        new Color(160,32,240),
        new Color(238, 130, 238),
        Color.black,
        new Color(205,120,92),
        new Color(0,100,0) };
/**
* Cài đặt phương thức của đối tượng StockNotify
```

```
 */
 public void update (Date date, Stock[] stock)
 {
System.out.println("StockApplet.update: " + date);
        // Lưu lại ngày giờ cập nhật
  if (time.size() == MAX_UPDATES) {
          time.removeElementAt(0);
  }
   time.addElement(date);
// Lưu lại dữ liệu giá cổ phiếu. Bạn xem phần cài đặt của lớp
// StockData bên dưới
  int numUpdates = time.size();
  for (int i=0; i<stock.length; i++) {
     StockData data = (StockData)stockTable.get(
                                        stock[i].symbol);
     if (data != null) {
     data.update(stock[i], numUpdates);
    }
  }
  //Vẽ lại bảng báo giá cổ phiếu
 repaint();
 }
 /**
  * Khởi tạo Applet
 */
 public void init()
 {
        //Khởi tạo các biến ban đầu
```

```
    time = new Vector(MAX_UPDATES);
    stockTable = new Hashtable();
    stockWatch = null;
        // Thông báo sự hiện diện của đối tượng StockNotify
    System.out.println("StockApplet.init:
                        exporting remote object");
    UnicastRemoteObject.exportObject(this);
        // Tìm đối tượng trên máy chủ
    URL base = getDocumentBase();
    String serverName = "rmi//localhost:1234/
                                        StockServer";
    System.out.println("StockApplet init");
    stockWatch = (StockWatch)
                Naming.lookup(serverName);
        // Đăng ký các cổ phiếu mà Applet muốn trình chủ cập
nhật giá
    for (int i=0; i<name.length; i++) {
    System.out.println("StockApplet.init:
                            watch stock " + name[i]);
    stockWatch.watch(name[i], this);
    stockTable.put(name[i],
                new StockData(name[i], color[i]));
    }
    System.out.println("StockApplet.init:
                            done registering stocks");
  } catch (Exception e) {
    add(new Label("exception occurred
```

```
            during initialization. Check the log"));
        add(new Label(e.getClass().getName()
                        + ": " + e.getMessage()));
        System.out.println("got exception: " +
                                e.getMessage());
        e.printStackTrace();
        return;
    }
    setLayout(null);
// Xóa các dữ liệu trước đó. Bạn xem thêm phần cài đặt của lớp
// SensitiveCheckbox bên dưới
    if ((ee != null) && (checkboxes != null)) {
    Enumeration oldCheckboxes = checkboxes.elements();
        while (oldCheckboxes.hasMoreElements()) {
          SensitiveCheckbox cb = (SensitiveCheckbox)
          oldCheckboxes.nextElement();
          remove(cb);
        }
    }
        // Vẽ các ô checkbox hiển thị tên cổ phiếu của các công ty
    checkboxes = new Vector();
    Enumeration enum = stockTable.elements();
    int i=0;
    while (enum.hasMoreElements()) {
        StockData data = (StockData)enum.nextElement();
        SensitiveCheckbox cb = new
                    SensitiveCheckbox (data, this);
```

```
    data.cb = cb;
    checkboxes.add(cb);
    add(cb);
    cb.setState(data.displayed);
    cb.reshape(10,i++*25+35,110,18);
  }
}
/**
 * Yêu cầu đối tượng chủ ngừng cung cấp thông tin khi Applet
 * chấm dứt hoạt động
 */
public void destroy()
{
    // Yêu cầu đối tượng chủ hủy bỏ cập nhật thông tin
if (stockWatch != null) {
    try {
    stockWatch.cancelAll(this);
    } catch (Exception e) {
    }
}
}
/**
 * Cập nhật và vẽ đồ thị của giá cổ phiếu.
 */
public void paint(Graphics g) {
    g.setColor(Color.black);
    g.drawRect(0,0,width-1,height-1);
```

```
    float miny = 0.0f;
    float maxy = 75.0f;
        // Vẽ đồ thị của tất cả dữ liệu
    Enumeration enum = stockTable.elements();
    while (enum.hasMoreElements()) {
      StockData data =
(StockData)enum.nextElement();
      int size;
      Stock[] updates;
      synchronized (data.updates) {
        size = data.updates.size();
        updates = new Stock[size];
        data.updates.copyInto(updates);
      }
      g.setColor(data.color);
      if (data.displayed) {
  // Vẽ hình chữ nhật bao quanh cổ phiếu nếu chuột di chuyển
  // đến cổ phiếu đó
        if (data.cb != null && data.cb.haveMouse()) {
          Point p = data.cb.location();
          Dimension d = data.cb.size();
          g.drawRect(p.x-1, p.y-1,
                     d.width+4, d.height+4);
          g.drawRect(p.x-2, p.y-2, d.width+4,
                     d.height+4);
      // Vẽ đồ thị
      if (size > 0)
```

```
            g.drawLine(p.x+d.width+2,p.y+10,150,
                        scale(updates[0].current));
        }
        int x = 150, inc = 10;
        for (int i = 0; i < size; i++) {
          if (updates[i] != null) {
            g.drawRect(x-1,
                  scale(updates[i].current)-1,3,3);
          if ((i < size - 1) && updates[i + 1] != null) {
              int x2 = x + inc;
              g.drawLine(x,
                    scale(updates[i].current), x2,
                    scale(updates[i + 1].current));
            }
          }
        x += inc;
    }
    }
    }
}
/**
 * Đổi toạ độ tương đối y
 */
int scale(float y) {
  return height - (int) (y*5+.5);
}
```

```
void setMouseHere(boolean display)
 {
 Enumeration enum = stockTable.elements();
 while (enum.hasMoreElements()) {
   StockData data = (StockData)enum.nextElement();
   data.cb.mouseHere = display;
 }
 }
 /**
  * Lớp StockData chứa cập nhật của giá cổ phiếu và hiển thị
  * thông tin về cổ phiếu
  */
 static class StockData {
 public String name;
 public Vector updates;
 public Color color;
 public boolean displayed;
 public SensitiveCheckbox cb;
 public StockData(String name, Color cl) {
this.name = name;
this.color = cl;
this.updates = new Vector(StockApplet.MAX_UPDATES);
displayed = true;
 }
    /**
      * Cập nhật
      */
```

```
    void update(Stock stock, int numUpdates) {
      synchronized (updates) {
    if (updates.size() == StockApplet.MAX_UPDATES) {
        updates.removeElementAt(0);
      }
if (updates.size() < numUpdates) {
    for (int i=updates.size(); i<numUpdates-1; i++) {
    updates.addElement(stock);
        }
      }
      updates.addElement(stock);
      }
    }
    }
    /**
     * Lớp SensitiveCheckbox dùng xác định chuột di chuyển
     */
   static class SensitiveCheckbox extends Canvas {
    StockData data;
    boolean state = true;
    StockApplet panel;
    boolean mouseHere = false;
    public boolean haveMouse() {
      return mouseHere;
    }
    public SensitiveCheckbox(StockData data,
                                        StockApplet p) {
```

```
    this.data = data;
    panel = p;
  }
  public boolean mouseEnter(Event evt, int x, int y) {
    if (state) {
    panel.setMouseHere(false);
    mouseHere = true;
    panel.repaint();
    }
    return false;
  }
  public boolean mouseExit(Event evt, int x, int y) {
    if (state) {
    mouseHere = false;
    panel.repaint();
    }
    return false;
  }
  public boolean mouseDown(Event evt, int x, int y) {
    if (state)
    state = false;
    else
    state = true;
    mouseHere=state;
    data.displayed = state;
    repaint();
```

```
    panel.repaint();
    return true;
  }
  public void paint(Graphics g) {
    g.setColor(Color.black);
    g.drawLine(4,4,14,4);
    g.drawLine(4,4,4,14);
    g.setColor(Color.gray);
    g.drawLine(5,14,14,14);
    g.drawLine(14,5,14,14);
    g.setColor(data.color);
    g.fillRect(5,5,8,8);
    g.setColor(data.color);
    g.drawString(data.name, 17,15);
    g.setColor(Color.black);
    if (state) {
    if (data.color == Color.black)
      g.setColor(Color.gray);
    if (data.color == Color.blue)
      g.setColor(Color.gray);
    g.drawLine(5,5,13,13);
    g.drawLine(5,6,12,13);
    g.drawLine(13,5,5,13);
    g.drawLine(13,6,6,13);
    }
  }
```

```
  public void setState(boolean s) {
    state = s;
    repaint();
  }
  }
}
```

StockApplet của ta cần được đặt trong một trang web. Bạn tạo trang web StockApp.html có nội dung như sau:

Ví dụ 2-53 StockApp.html

```
<HTML>
 Stock Exchange from Java
<applet codebase="."
  code="StockApplet"
  width=500 height=350>
</applet>
</HTML>
```

Trước khi biên dịch, chúng ta cần tạo tập tin policy qui định cơ chế bảo mật cho applet và trình chủ như sau:

Mở trình soạn thảo notepad trong Windows nhập vào chế độ bảo mật và lưu tập tin xuống đĩa với tên security.policy .

Ví dụ 2-54 Tập tin chính sách security.policy

```
grant {
 permission java.security.AllPermission;
};
```

Bước 6: Cuối cùng là biên dịch và chạy chương trình ở cả hai phía khách chủ. Bạn hãy lưu tất cả mã nguồn và tập tin

security.policy, StockApp.html vào thư mục C:\Network\Stock\ Chuyển vào thư mục này thực hiện các bước biên dịch như sau:

Biên dịch mã nguồn

```
C:\Network\Stock>javac *.java
```

Biên dịch tập tin trung gian cho đối tượng StockServer

```
C:\Network\Stock>rmic StockServer
```

Biên dịch tập tin trung gian cho đối tượng StockNotify

```
C:\Network\Stock>rmic StockApplet
```

(Bạn lưu ý, StockApplet của ta do cài đặt giao tiếp StockNotify nên bản thân nó là một đối tượng có thể phục vụ cho giao tiếp StockNotify thật sự).

Tạo và đăng ký đối tượng trên máy chủ

```
C:\Network\Stock>java Djava.security.policy=
                security.policy  StockServer
```

(Chúng ta không cần khởi động chương trình rmiregistry, StockServer tự tạo bộ đăng ký registry ở cổng 1234 và tự đăng ký với registry chính bản thân nó)

Khởi động Applet trên máy khách

```
C:\Network\Stock> appletviewer StockApp.html
```

Kết quả chương trình sẽ được hiển thị như hình 2-11. Applet liệt kê một bảng giá cổ phiếu cùng với đồ thị biểu diễn biến động của giá cổ phiếu đang tăng hay giảm trong rất sinh động. Bạn có thể di chuyển chuột đến từng loại cổ phiếu của các công ty khác nhau. Kích chuột vào ô checkbox, khi bạn không đánh dấu ô checkbox, đối tượng StockNotify sẽ không còn yêu cầu máy chủ cập nhật giá cổ phiếu của công ty đó nữa.

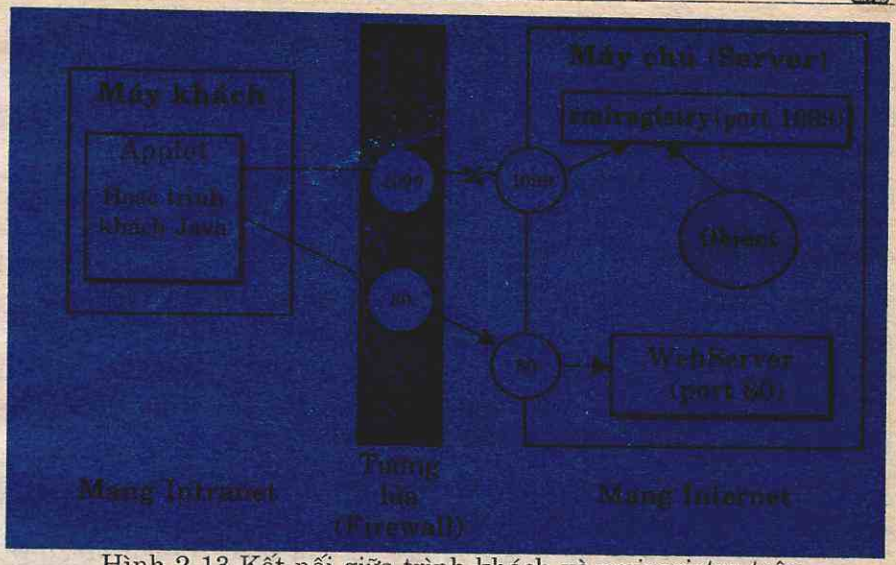
Điểm thú vị là Applet của ta có khả năng chạy trên Browser. Không cần bất kỳ cài đặt nào từ phía máy khách. Những công việc

cần làm của trình khách chỉ là mở trình duyệt (IE hay Netscape) và đánh vào địa chỉ URL dẫn đến trang web StockApp.html do nhà cung cấp chỉ định. Việc cài đặt StockServer, StockNotify ... là do trình chủ đảm nhiệm.

15. RMI REGISTRY VÀ CÁC VẤN ĐỀ VỀ FIREWALL (TƯỜNG LỬA)

Trong cơ chế kết nối theo giao thức TCP/IP bằng khái niệm socket, bắt buộc bạn phải dùng đến khái niệm cổng port. Mỗi trình chủ phải hoạt động trên một cổng chỉ định để lắng nghe các kết nối gửi đến từ máy khách. Khi ứng dụng của bạn hoạt động trên mô hình mạng nội bộ (intranet) thì điều này không thành vấn đề. Bạn có thể tự do chỉ định hay gán số cổng bất kỳ cho socket trên máy chủ (miễn là những cổng này không trùng với các cổng nổi tiếng - là những cổng mặc định dành cho các ứng dụng như Web, Mail, FTP - đã có). Tuy nhiên thực tế mạng diện rộng Internet luôn bị kiểm soát nghiêm ngặt bởi tường lửa (firewal). Tường lửa là phần mềm (có thể được cài đặt hoặc điều khiển bởi phần cứng tùy theo nhà cung cấp Internet) chặn ngay ngõ ra vào của đường truyền dữ liệu kết nối mạng của bạn với bên ngoài (Internet). Chức năng của tường lửa là kiểm soát cách thức truyền dữ liệu và kết nối giữa mạng Intranet với mạng Internet. Mục đích tránh sự đột nhập và tấn công hay ăn cắp dữ liệu của các tay hacker bên ngoài mạng. Tường lửa cũng được dùng để hạn chế sự truy xuất của bạn ra bên ngoài mạng Internet.

Vấn đề phát sinh là việc sử dụng cổng kết nối. Hầu như tất cả các tường lửa đều chỉ cho sử dụng một số rất ít cổng. Cổng kết nối theo giao thức http (port 80) là cổng nổi tiếng và được cho phép sử dụng để kết nối với dịch vụ web server. Một số cổng khác mà bạn có thể được phép sử dụng như: cổng 21 (ftp), 23 (telnet), 110 (POP3 mail). Như vậy các ứng dụng Java hầu như không sử dụng được mô hình khách chủ nếu gặp phải tường lửa.



Hình 2-13 Kết nối giữa trình khách và rmiregistry trên máy chủ bị chặn bởi tường lửa

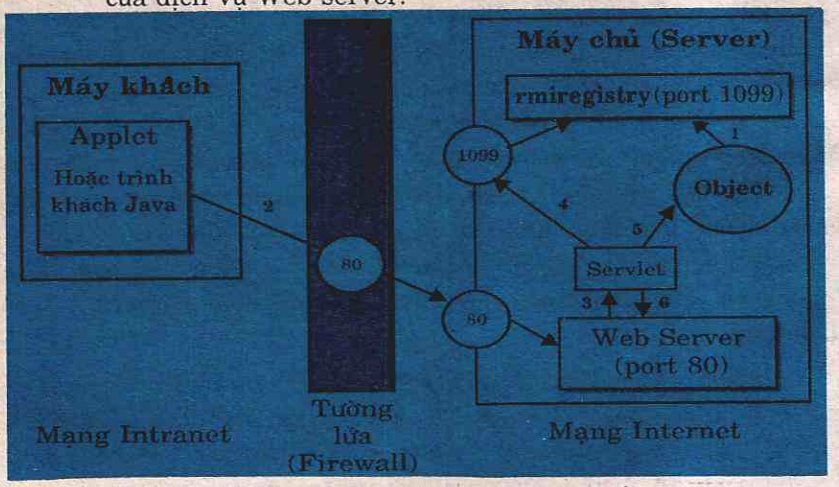
Hình 2-13 là cơ chế kết nối của ứng dụng RMI từ trình khách đến máy chủ bị chặn bởi tường lửa. Như bạn đã biết, một ứng dụng ở trình khách muốn triệu gọi được phương thức của đối tượng Object trên máy chủ thì trước hết phải liên hệ với bộ đăng ký rmiregistry để lấy về tham chiếu của đối tượng. Dịch vụ rmiregistry mở socket mặc định ở cổng 1099 (hoặc một cổng nào đó do bạn chỉ định) lắng nghe kết nối từ trình khách. Tiếc thay những cổng như vậy không được tường lửa chấp nhận cho qua. Liên lạc giữa trình khách và rmiregistry trên máy chủ sẽ không thực hiện được và bạn cũng sẽ không có cách nào tham chiếu được đến đối tượng Object ở xa như mong muốn.

Thực sự thì khi đối tượng Object phục vụ bạn nó sẽ mở thêm một cổng vô danh để thực hiện giao thức kết nối TCP/IP với trình khách và bạn cũng không thể "nói chuyện" được với đối tượng nếu tường lửa ngăn cấm cổng kết nối của đối tượng

Bạn có thể thử nghiệm cơ chế tường lửa của Việt nam bằng cách viết một ứng dụng khách chủ Java RMI. Bạn gửi chương trình đăng ký đối tượng chủ của mình với một người bạn ở nước ngoài và yêu cầu họ cài đặt trên máy chủ host. Sau đó bạn thử chạy kết nối từ trình khách ngay trong nước. Chương trình sẽ không chạy!. Đây cũng chính là lý do mà hầu hết các ứng dụng Chat (chương trình trò chuyện trên mạng) viết bằng Applet đều không chạy ở Việt nam.

Tuy nhiên như bạn thấy cổng 80 là cổng của dịch vụ http hầu như được mọi tường lửa cho qua. Cổng dịch vụ http chuyên cung cấp trang Web cho các trình duyệt browser nên ít khi bị tường lửa ngăn cấm. Chính vì vậy ứng dụng phân tán Java gọi đối tượng từ xa có thể giải quyết vấn đề tường lửa bằng hai cách:

- Thứ nhất, bạn yêu cầu người quản trị tường lửa cung cấp cho bạn một số cổng để sử dụng. Tuy nhiên cách này ít khi được chấp nhận và nếu có bạn phải trả chi phí rất cao cho việc mở cổng.
- Thứ hai bạn sử dụng cơ chế trung gian thông qua cổng 80 của dịch vụ Web server.



Hình 2-14 Gọi đối tượng ở xa thông qua cơ chế trung gian của Web server

Hình 2-14 là cơ chế gọi đối tượng từ xa thông qua dịch vụ Web server bên ngoài tường lửa. Đối tượng Object của ta đăng ký với rmiregistry trên máy chủ (1) lắng nghe ở cổng 1099. Trình khách gửi yêu cầu theo giao thức http đến máy chủ web server yêu cầu truy xuất đến đối tượng Object (2). Web server yêu cầu một đối tượng trung gian trên máy chủ (ở chương sau bạn sẽ biết đến đối tượng này với tên là Servlet) xử lý lời gọi (3). Servlet liên lạc với rmiregistry lấy về tham chiếu của đối tượng Object (4). Servlet triệu gọi Object (5) và trả kết quả về cho web server (6). Sau cùng là Web server trả kết quả về cho trình khách.

Cơ chế truy xuất và gọi đối tượng thông qua cổng 80 của Web Server còn gọi là cơ chế "luồn hầm" (tunneling).

Như bạn thấy cơ chế liên lạc với đối tượng ở xa thông qua tường lửa trên máy chủ khá phức tạp. Tuy nhiên đó cũng chính là vấn đề để chúng ta nghiên cứu những kỹ thuật lập trình mạng mới mẻ trong chương sau. Kỹ thuật lập trình các ứng dụng mạng đa tầng (multi-tier Application)

16. KẾT CHƯƠNG

Chương này chúng ta đã nghiên cứu kỹ thuật lập trình phân tán triệu gọi đối tượng từ xa RMI của Java. Để xây dựng đối tượng ở xa bạn phải thiết lập một giao tiếp (interface) của đối tượng. Cài đặt giao tiếp của đối tượng sau đó tạo ra các lớp trung gian _Stub, _Skel tương ứng. Giao tiếp ở trình khách với đối tượng chủ thông qua lớp _Stub. Giao tiếp của đối tượng chủ với trình khách thông qua lớp _Skel. Trước khi hoạt động đối tượng chủ phải đăng ký tên với trình rmiregistry để trình khách có thể truy tìm tham chiếu của đối tượng từ xa. Truyền dữ liệu giữa trình khách và đối tượng trên máy chủ có thể thông qua tham trị và tham chiếu. Các đối tượng muốn truyền theo tham trị phải cài đặt giao tiếp java.io.Serializable. Các đối tượng muốn truyền theo tham chiếu phải cài đặt giao tiếp java.rmi.Remote. Tham chiếu từ đối tượng chủ đến đối tượng trên trình khách còn gọi

là tham chiếu ngược (callback). Với tham chiếu ngược, trình chủ có thể chủ động gọi và điều khiển trình khách. Chúng ta cũng đã nghiên cứu những giới hạn của ứng dụng phân tán RMI trên mạng diện rộng Internet khi gặp phải tường lửa (firewall). Cách thiết lập và sử dụng trình mồi nạp toàn bộ ứng dụng từ máy chủ về máy khách. Cách tự động kích hoạt đối tượng (activation) ...

Kỹ thuật RMI của Java không phải là kỹ thuật lập trình phân tán đối tượng tối ưu. Chúng ta sẽ tiếp tục nghiên cứu đề tài lập trình và gọi đối tượng ở xa thú vị hơn, CORBA.

Chương 3

CORBA

Tóm tắt chương 3

Ở chương trước bạn đã làm quen với khái niệm lập trình phân tán bằng kỹ thuật RMI của Java. Chương này sẽ cung cấp cho bạn một khái niệm lập trình phân tán mở rộng hơn với các đối tượng được thiết kế trên nền CORBA áp dụng cho mọi ngôn ngữ lập trình.

Các vấn đề chính sẽ được đề cập đến:

- ✓ *CORBA và các khái niệm mở đầu*
- ✓ *Vai trò trình môi giới ORB và mô hình phân tán*
- ✓ *Xây dựng đối tượng CORBA*
- ✓ *Tìm hiểu JNDI và COS Naming*
- ✓ *Tham chiếu ngược trong CORBA*
- ✓ *Tìm hiểu kỹ thuật RMI-over-IIOP*
- ✓ *CORBA và trình môi giới VisiBroker*

1. CORBA VÀ CÁC KHÁI NIỆM MỞ ĐẦU

Hơn ai hết, nếu bạn đã từng là lập trình viên bạn sẽ hiểu được công việc nặng nhọc khi phải viết mã lệnh cho chương trình. Cách thiết kế chương trình theo mô hình thủ tục (top-down procedural hay thủ tục trên-xuống) dần dần được thay thế bằng mô hình hướng đối tượng khoa học và hiệu quả hơn. Lập trình hướng đối tượng đã chứng minh được tính đúng đắn và sức mạnh của nó trong việc thiết kế các hệ thống lớn cũng như dễ dàng trong việc nâng cấp và bảo trì mã lệnh. Ngày nay hầu như mọi ngôn ngữ lập trình đều hỗ trợ và thấm nhuần tư tưởng đối tượng. Mở đầu là Smalltalk, C++, Object

Pascal, Delphi, Visual Basic tiếp đến là các hệ quản trị dữ liệu Visual Foxpro, Access, SQL Server, Oracle ... và như bạn thấy Java là ngôn ngữ ra đời sau nhưng lại thuần đối tượng nhất. Trong Java mọi thứ, mọi nơi, mọi khái niệm đều qui về đối tượng.

Tuy nhiên bạn có nhận ra một điểm bất đồng lớn nhất giữa các ngôn ngữ lập trình? Đó là các đối tượng thiết kế bằng ngôn ngữ nào thì sau khi biên dịch ra dạng nhị phân (binary) chỉ có mã lệnh tương ứng của ngôn ngữ đó mới có khả năng truy xuất được đối tượng. Đối tượng viết bằng C++ không dễ dàng gì truy xuất được từ mã lệnh Delphi hay Visual Basic một cách tự nhiên. Ngay cả đối tượng viết bằng Java cũng khó lòng gọi được từ các ngôn ngữ lập trình khác (mặc dù có một số kỹ thuật như sử dụng thư viện liên kết động DLL , gọi các hàm thư viện native ... nhưng đó không phải là các giải pháp toàn diện).

Java ra đời với mong muốn các ứng dụng hay đối tượng viết bằng Java có thể chạy được mọi lúc mọi nơi trên mọi hệ điều hành (và cho đến nay thì có thể nói Sun Microsystem hầu như đã thực hiện được mong ước của mình cùng với sự nổi tiếng của Java). Tuy nhiên Java không thể thay thế các ngôn ngữ lập trình khác được. Không thể bắt mọi lập trình viên trên thế giới đều chuyển sang viết Java trong khi có hàng trăm triệu dòng mã lệnh của chương trình cũng như thư viện đã được thiết kế và xây dựng từ các ngôn ngữ tiền bối khác (điển hình là C/C++ và Pascal) đang hoạt động rất hiệu quả. Hàng chục ngàn lập trình viên chuyên nghiệp đã quen và thân thuộc với C, Pascal, Delphi, Visual Basic. Java chỉ là một lựa chọn trong số các lựa chọn đã có.

Tuy nhiên các nhà lập trình luôn mong muốn tìm được một tiếng nói chung cho tất cả các ngôn ngữ lập trình hiện có (tương tự con người muốn có một "quốc tế ngữ"). Và thế là CORBA đã ra đời.

CORBA (Common Object Request Broker Architecture) – tạm dịch *Kiến trúc môi giới gọi các đối tượng thông dụng*. CORBA được hình thành từ một tổ chức nghiên cứu quốc tế OMG (Object

Management Group) với sự hợp tác của hơn 800 công ty. Tham vọng của OMG là đưa ra cách để các đối tượng viết bằng những ngôn ngữ khác nhau có thể triệu gọi lẫn nhau theo mô hình đối tượng phân tán.

CORBA không phải là một ngôn ngữ lập trình như C++ hay Java. CORBA là một ngôn ngữ đặc tả (description language). Nếu bạn đã viết qua nhiều ngôn ngữ lập trình (hay ít nhất cũng là Java) bạn có thấy điểm gì chung nhất giữa các ngữ trình hay không? Đó chính là lời gọi thủ tục, hàm, kiểu dữ liệu, tham số truyền vào hàm, trị trả về của hàm. Đối với các ngôn ngữ, điều đầu tiên khi cài đặt hàm hay thủ tục là phải khai báo. Ví dụ:

Trong C/C++

```
int get_price(char* product);
```

Trong Pascal/ Delphi

```
function get_price(product: string):integer;
```

Trong Visual Basic

```
function get_price(product as string) as integer
```

Trong Java

```
public interface SomeLibrary{
    public static int get_price(String product);
}
```

Bước đầu tiên khi thiết kế chương trình là phân tích chương trình ở mức trừu tượng nhất nếu có thể. Giả sử ở bước này bạn không hề muốn áp đặt với một ngôn ngữ lập trình nào cả. Bạn dùng một ngôn ngữ tổng quát để khai báo hàm get_price của riêng mình như sau:

```
thefunction theinteger get_price(
            product as thestring);
```

Đừng ngạc nhiên nếu bạn thấy khai báo trên là sự kết hợp giữa các cách khai báo C++/Delphi/Visual Basic với từ khóa như `thefunction`, `theInteger`, `theString` kỳ lạ. Không có gì khó

hiểu, chúng ta cần một hàm mang tên `get_price`. Hàm này cần một đối số kiểu chuỗi `thestring` mang tên sản phẩm. Sau khi xử lý gì đó hàm sẽ trả về cho bạn một số nguyên `theinteger`. Đây chỉ là đặc tả hàm và như bạn thấy nó đúng cho mọi ngôn ngữ lập trình!

Tiếp theo, bạn muốn cài đặt hàm này bằng ngôn ngữ Pascal? Có cách nào để chuyển đặc tả trên thành khai báo của Pascal không nhỉ? Điều này không khó nếu bạn biết được các khai báo tương ứng giữa đặc tả của bạn và các từ khóa cũng như kiểu dữ liệu trong Pascal. Ví dụ:

Ngôn ngữ đặc tả	Ngôn ngữ Pascal
`thefunction`	`function`
`theinteger`	`integer`
`get_price`	`get_price`
`as`	`:`
`thestring`	`string`

và trong Pascal, hàm của bạn có thể được viết lại như sau:

```
function get_price(product : string):integer;
```

Quá trình trên gọi là ánh xạ (mapping) từ ngôn ngữ đặc tả sang ngôn ngữ cài đặt. Giả sử bạn lưu nội dung đặc tả của bạn thành tập tin getprice.txt. Bây giờ lại giả sử chúng ta có một chương trình translate đủ thông minh biết cách ánh xạ từ ngôn ngữ đặc tả sang các ngôn ngữ lập trình thông dụng. Để cài đặt hàm get_price trong các ngôn ngữ khác bạn có thể dịch tập tin getprice.txt như sau

```
translate -C getprice.txt getprice.c
translate -pascal getprice.txt getprice.pas
translate -vb getprice.txt getprice.bas
translate -java getprice.txt getprice.java
```

Việc cài đặt bên trong thân hàm get_price là nhiệm vụ của mỗi ngôn ngữ bạn không thể biết, nhưng giá trị trả về của hàm và đối số

truyền vào hàm luôn là kiểu số nguyên cũng như kiểu chuỗi biết trước.

Đó là ý tưởng của các nhà thiết kế và xây dựng nên kiến trúc CORBA. CORBA qui định một tập các mô tả hàm, kiểu dữ liệu cách khai báo để bạn đặc tả đối tượng. Đặc tả đối tượng cũng giống như khái niệm giao tiếp interface trong Java. Chỉ là mô tả chức năng của đối tượng thông qua hàm, phương thức, thuộc tính .... Không có bất kỳ cài đặt mã lệnh nào trong ngôn ngữ đặc tả. Chính vì vậy ngôn ngữ CORBA còn được gọi là ngôn ngữ đặc tả giao tiếp (Interface Decsription Language hay IDL). Ví dụ dưới đây là một đặc tả của đối tượng Calculator viết bằng ngôn ngữ IDL của CORBA

Ví dụ 3-1 Calculator.idl

```
interface Calculator{
    long addnumber(in long x, in long y);
};
```

Ở ví dụ về đặc tả hàm get_price chúng ta đã dùng chương trình translate tưởng tượng để dịch tập tin đặc tả getprice.txt thành các ngôn ngữ cụ thể. Tuy nhiên với ngôn ngữ IDL, CORBA cung cấp cho bạn những chương trình thực tế hơn. Ví dụ để chuyển tập tin Calculator.idl sang ngôn ngữ C, Pascal bạn có thể gọi:

```
idl2cpp  Calculator.idl
```

kết quả bạn sẽ thu được tập tin `Calculator`.CPP. Hay:

```
idl2pas  Calculator.idl
```

kết quả bạn sẽ thu được tập tin `Calculator`.pas

Giáo trình này chủ yếu giúp bạn tìm hiểu Java nên tốt nhất là chúng ta hãy chuyển tập tin Calculator.idl sang mã nguồn Java. Sun cung cấp cho bạn chương trình idl2java.exe để thực hiện công việc này (với Java 1.3 trình idl2java được thay bằng idlj)

```
idlj  Calculator.idl
```

Kết quả bạn sẽ thu được một số tập tin .java (chúng ta sẽ nghiên cứu ở phần sau). Trong số các tập tin do trình thông dịch idlj

phát sinh bạn sẽ thấy tập tin CalculatorOperations.java có khai báo đúng như những gì chúng ta đã đặc tả bằng ngôn ngữ IDL của CORBA nhưng được viết bằng ngôn ngữ Java thuần túy:

Ví dụ 3-2 CalculatorOperations.java

```
/**
* CalculatorOperations.java
* Generated by the IDL-to-Java compiler (portable), version "3.0"
* from Calculator.idl
* Monday, July 6, 1998 1:09:55 AM GMT+01:00
*/
public interface CalculatorOperations
{
 int addnumber (int x, int y);
} // interface CalculatorOperations
```

Bước cài đặt hàm addnumber() là của lập trình viên Java. Nhiệm vụ còn lại của CORBA chỉ là làm sao gọi được hàm addnumber() từ đối tượng CalculatorOperations do lập trình viên Java cung cấp. Đó là điều chúng ta sẽ nghiên cứu xuyên suốt toàn bộ chương này.

2. ÁNH XẠ GIỮA NGÔN NGỮ IDL VÀ JAVA (IDL MAPPING TO JAVA)

Dưới đây là tập các ánh xạ của ngôn ngữ IDL trong CORBA tương đương với các kiểu dữ liệu và từ khóa có trong Java.

IDL CORBA	Java
module	*package*
interface	*interface, helper class, holder class*
constant	*public static final*

IDL CORBA	Java
boolean	*boolean*
char, wchar	*char*
octet	*byte*
string, wstring	*java.lang.String*
short, unsigned short	*short*
long, unsigned long	*int*
long long, unsigned long long	*long*
float	*float*
double	*double*
enum, struct, union	*class*
sequence, array	*array*
exception	*class*
operation	*Method*

Như bạn thấy, ngôn ngữ IDL không có các cấu trúc điều khiển như `if, while, do, for` ...

IDL chỉ là ngôn ngữ đặc tả. Trong ngôn ngữ đặc tả mọi việc diễn ra tương tự như khi bạn khai báo biến, mô tả đối tượng, kiểu record ... không cần phải có các cấu trúc lệnh điều khiển.

Java cung cấp cho bạn công cụ dùng để chuyển đổi qua lại giữa khai báo CORBA sang khai báo Java và ngược lại như sau:

- Để chuyển một lớp interface nào đó của java sang đặc tả IDL bạn dùng lệnh
  ```
  rmic -idl YourObject.class
  ```
- Để chuyển một đặc tả IDL sang khai báo interface tương ứng trong Java bạn dùng lệnh
  ```
  idlj YourObject.idl
  ```

rmic và idlj là hai chương trình .exe nằm trong thư mục [JAVA_HOME]\bin.

> *Ở một số phiên bản Java cũ hoặc ở các công cụ phát triển của những nhà cung cấp khác bạn có thể dùng trình java2idl.exe (chuyển java sang idl) và idl2java (chuyển idl sang java) với cùng một mục đích.*

Ví dụ ánh xạ giữa CORBA và Java :

CORBA IDL	JAVA

module{
interface MathLibrary {
long add(
in long x,
in long y);
 	
long subtract(
in long x,
in long y);
 	
long divide(
in long x,
in long y);
 	
long multiple(
in long y,
in long y);
 	
string About(
in string version)
}
};
	package Math;
	public interface
	MathLibrary{
	int add(
	int x,
	int y);
	
	int subtract(
	int x,
	int y);
	
	int divide(
	int x,
	int y);
	
	int multiple(
	int x,
	int y);
	
	String About(
	String version);
	}

Như bạn thấy, khái niệm package của Java tương đương với module trong CORBA. CORBA đưa ra từ khóa in cho các biến

truyền vào phương thức theo trị (tương tự từ khóa out dùng để lấy trị trả về).

Ngôn ngữ CORBA gần giống với cách khai báo của ngôn ngữ C và cũng không khó học. Bạn có thể tham khảo địa chỉ www.omg.org để biết những thay đổi mới nhất về CORBA. Hầu như khi nghiên cứu về CORBA trong Java ở phần sau, chúng ta cũng không cần phải đi chi tiết về ngôn ngữ CORBA, bởi trình chuyển ngữ rmic (java2idl) và idlj (idl2java) đã thực hiện giùm bạn công việc này.

3. CORBA VÀ MÔ HÌNH PHÂN TÁN

3.1. Chương trình môi giới trung gian ORB (Object Request Broker)

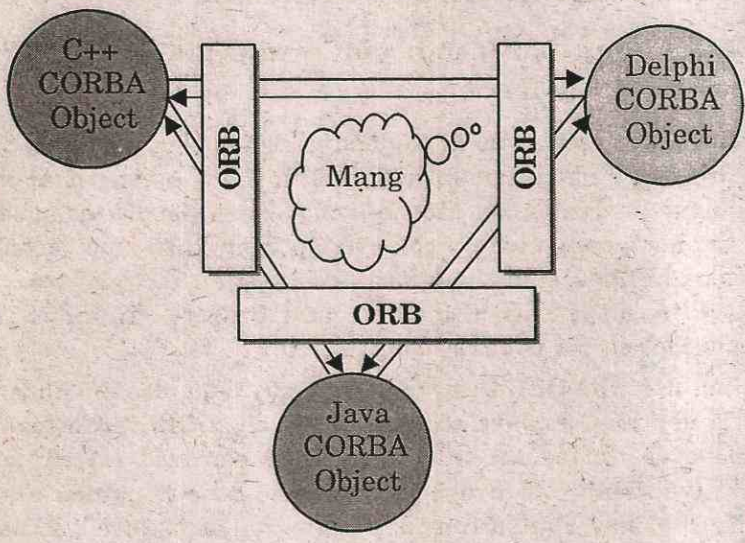
Java dựa vào máy ảo để tạo nên tiếng nói chung giữa tất cả ứng dụng viết bằng ngôn ngữ Java. Tương tự CORBA cũng phải có một nền tảng gì đó để liên kết các đối tượng. Vâng, các đối tượng sau khi được xây dựng bằng những ngôn ngữ lập trình khác nhau muốn triệu gọi được lẫn nhau phải thông qua chương trình môi giới trung gian của CORBA gọi là ORB (Object Request Broker – Tạm dịch *Trình môi giới các yêu cầu của đối tượng*).

Bạn nên nhớ ORB hoàn toàn được xây dựng độc lập không liên quan gì đến máy ảo Java cả. Bạn có thể chạy ORB hoàn toàn tách biệt Java và thường thì ORB do các nhà sản xuất có tầm cỡ như Inprise (với trình Visi Broker nổi tiếng sẽ được ta nghiên cứu sau), Oracle, IBM hay Fujitsu cung cấp ... Tuy nhiên từ Java 1.2, máy ảo Java đã nhúng luôn khả năng ORB và xem nó như một phần của nền Java chuẩn. Bạn có thể thử nghiệm các ứng dụng CORBA ngay trên máy ảo Java mà không cần phải tìm kiếm một chương trình ORB của những nhà cung cấp khác.

Hình 3-1 là cơ chế triệu gọi lẫn nhau của các đối tượng viết bằng C++, Delphi, Java thông qua ORB.

Làm cách nào ORB gọi được phương thức của các đối tượng cài đặt bằng các ngôn ngữ khác nhau? Đó là nhờ ngôn ngữ đặc tả IDL mà ta đã có dịp tìm hiểu ở phần trên. Các đối tượng cho dù cài đặt

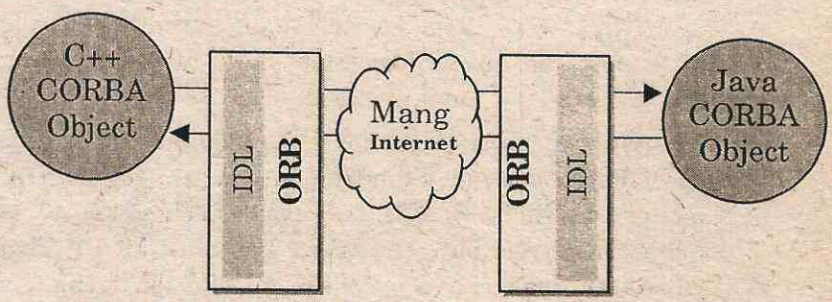
bằng bất kỳ ngôn ngữ nào nếu như tuân theo đặc tả IDL đều được xem là đối tượng CORBA. Tuy không cài đặt bản thân đối tượng nhưng dựa vào đặc tả IDL ORB sẽ biết được tên phương thức cần gọi, kiểu đối số truyền cho phương thức, trị trả về của phương thức gọi. Với những thông tin này, ORB có thể triệu gọi phương thức mà đối tượng cài đặt (hình 3-2). ORB là hoàn toàn trong suốt (transparent) với lập trình viên (hay người sử dụng đối tượng). Bạn gọi đối tượng của Java trong C++ hay Delphi hoàn toàn theo cách tự nhiên như bạn gọi đối tượng được xây dựng từ chính ngôn ngữ gốc.



Hình 3-1. Triệu gọi đối tượng CORBA thông qua trình môi giới ORB

Khả năng "trong suốt" không chỉ giới hạn giữa các đối tượng trên cùng một máy mà còn được mở rộng giữa các đối tượng nằm ở bất kỳ đâu trên thế giới thông qua kết nối mạng. Đối tượng của bạn viết bằng Java đặt trên một máy tại Việt nam có thể được gọi bởi một đối tượng ở Châu Mỹ ("bên kia đại dương") cài đặt bằng C++.

Với khả năng trên, CORBA quả là mang lại điều kỳ diệu cho thế giới lập trình. CORBA mở cửa để các lập trình viên có thể xích lại gần nhau hơn, tạo nên một thể thống nhất của mô hình lập trình hướng đối tượng.



Hình 3-2. ORB giao tiếp với đối tượng thông qua các thông tin đặc tả IDL

3.2. Cơ chế làm việc của ORB

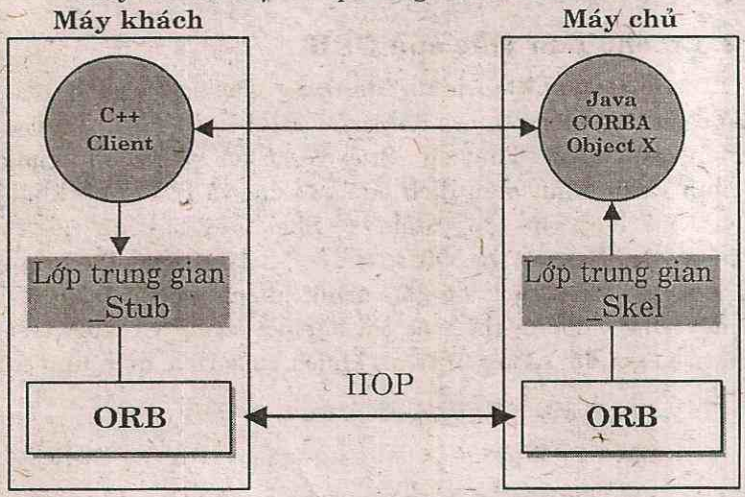
ORB cũng dùng khái niệm lớp trung gian _Stub và _Skel tương tự như cách làm việc của các đối tượng RMI (mà ta đã có dịp nghiên cứu trong chương 2). Thật sự _Stub và _Skel là hai lớp trung gian được sinh ra từ trình biên dịch dựa vào đặc tả IDL. Bạn không cần phải viết mã lệnh cho lớp _Stub và _Skel. Đây là công việc của các nhà cung cấp và thiết kế ORB. Hình 3-3 là cơ chế hoạt động của trình môi giới ORB liên lạc giữa trình khách và đối tượng CORBA trên máy chủ. Thường thì việc cài đặt đối tượng CORBA trên máy chủ và triệu gọi đối tượng từ trình khách được thực hiện như sau:

1) Phần cài đặt đối tượng CORBA trên máy chủ

- Đặc tả đối tượng X bằng ngôn ngữ IDL của CORBA
- Chuyển đặc tả X thành mô tả của một ngôn ngữ cụ thể (chẳng hạn dùng trình biên dịch idlj.exe để chuyển đặc tả của đối tượng X sang ngôn ngữ Java). Dựa vào đặc tả IDL yêu cầu trình biên dịch tạo ra lớp trung gian _Skel.
- Cài đặt đối tượng X bằng ngôn ngữ Java

2) Phần kết nối với đối tượng từ máy khách

- Sử dụng đặc tả IDL của đối tượng X yêu cầu trình biên dịch tạo ra lớp trung gian _Stub - Tùy theo ngôn ngữ muốn viết ở máy khách mà ta sử dụng trình biên dịch thích hợp. Ví dụ nếu bạn muốn viết trình khách bằng ngôn ngữ C++ bạn có thể dùng trình biên dịch idl2cpp.exe để tạo ra lớp _Stub dành cho chương trình C++ triệu gọi đối tượng CORBA.
- Khi trình khách triệu gọi đối tượng chủ, thực tế lời gọi sẽ được chuyển qua lớp _Stub trên máy khách, gửi đến trình môi giới ORB. Trình ORB trên máy khách sẽ kết nối với trình ORB trên máy chủ (theo giao thức IIOP). Trình ORB trên máy chủ chuyển lời gọi cho lớp trung gian _Skel trên máy chủ. Cuối cùng thì đối tượng CORBA trên máy chủ sẽ thực thi phương thức của nó.



Lời gọi đối tượng trên lý thuyết

___ Lời gọi đối tượng thực tế

Hình 3-3. Giao tiếp giữa trình khách và đối tượng CORBA trên máy chủ

3.3. Giao thức IIOP

Giao tiếp giữa ORB trên máy khách và ORB trên máy chủ được xây dựng dựa trên giao thức mang tên IIOP (Interoperate Internet Object Protocol). Giao thức này do tổ chức CORBA đề xướng. Nó làm nền cho tất cả các đối tượng có thể trò chuyện được với nhau trên mạng Internet một cách dễ dàng. Hiện nay giao thức IIOP được dùng khá phổ biến thậm chí còn vượt trội cả giao thức triệu gọi từ xa RMI của Java (Java gọi giao thức RMI mà các máy ảo có thể dùng trò chuyện được với các đối tượng ở xa là JRMP – Java Remote Method Protocol). Ở phần sau ta sẽ học cách kết hợp cả hai giao thức IIOP và JRMP với nhau trong các lời gọi đối tượng CORBA từ Java. Cách kết hợp này còn được gọi bằng một tên khá quen thuộc trong các tài liệu của Java, đó là *RMI-over-IIOP*.

4. XÂY DỰNG ỨNG DỤNG CORBA KHÁCH/CHỦ TRONG MÔI TRƯỜNG JAVA

4.1. Xây dựng đối tượng CORBA

Cách tốt nhất để hiểu về CORBA là chúng ta hãy bắt tay vào xây dựng một chương trình CORBA thực sự. Vâng, theo truyền thống lập trình chúng ta vẫn thường bắt đầu bằng chương trình Hello World đơn giản nhất. Chương trình CORBA trên máy khách (CORBA Client) sẽ gọi đối tượng trên máy chủ. Đối tượng CORBA trên máy chủ sẽ trả về chuỗi mang lời chào "Hello CORBA". Bạn hãy thực hiện qua các bước sau:

Bước 1: Chúng ta cần một đối tượng Hello. Đối tượng này cung cấp phương thức sayHello(). Đặc tả đối tượng bằng ngôn ngữ CORBA IDL được diễn đạt như sau:

Ví dụ 3-1 Hello.idl

```
interface Hello {
  string sayHello();
```

```
};
```

Bạn lưu ví dụ 3-1 thành tập tin Hello.idl trong thư mục C:\Network\CORBA\Hello

Bước 2: Xây dựng đối tượng Hello bằng ngôn ngữ Java.

Với nội dung của tập tin Hello.idl bạn sử dụng chương trình idlj.exe (hay idl2java.exe nếu bạn dùng Jdk 1.2 trở xuống) để chuyển đặc tả của CORBA sang ngôn ngữ Java. Đánh lệnh biên dịch như sau

```
idlj -fserver Hello.idl
```

Kết quả bạn sẽ thu được các tập tin sau:

`HelloOperations.java`	*Tập tin chuyển đặc tả IDL thành đặc tả interface của Java*
`Hello.java`	*Tập tin chuyển lớp giao tiếp đối tượng kiểu Java thành lớp giao tiếp đối tượng đặc thù trong CORBA*
`_HelloImplBase.java`	*Tập tin chứa lớp cài đặt cho đối tượng CORBA.*

Muốn cài đặt đối tượng CORBA, bạn chỉ cần quan tâm đến lớp _HelloImplBase (các tập tin phụ như Hello.java, HelloOperations.java sẽ được ta nghiên cứu sau). Đối tượng CORBA thật sự của lớp Hello sẽ được xây dựng theo bước kế tiếp.

Bước 3: Cài đặt đối tượng CORBA Hello.

Ví dụ 3.2 HelloServant.java

```
public class HelloServant
                    extends _HelloImplBase
{
    public String sayHello(){
        System.out.println("Client contact !");
```

```
        return "Hello CORBA";
    }
}
```

Lớp _HelloImplBase (do trình biên dịch idlj tạo ra) nguyên thủy kế thừa từ lớp CORBA.Object (bạn có thể nghiên cứu tập tin _HelloImplBase.java để thấy rõ điều này) cho nên đối tượng HelloServant của bạn một khi đã kế thừa lớp _HelloImplBase cũng sẽ là một đối tượng CORBA chính thống. Điều này tương tự các đối tượng RMI phải kế thừa từ lớp Remote vậy.

Lưu mã nguồn vào thư mục C:\Network\CORBA\Hello (cùng với thư mục chứa tập tin idl và các tập tin phụ). Chuyển vào thư mục này và đánh lệnh biên dịch

```
C:\Network\CORBA\Hello> javac *.java
```

Tập tin HelloServant.class mà bạn thu được là một đối tượng CORBA thật sự được cài đặt bằng ngôn ngữ Java. Các lớp như Hello.class, HelloOperations.class là những lớp trung gian hỗ trợ cho lời gọi của trình môi giới ORB đến đối tượng CORBA.

Thật sự thì đối tượng CORBA được xây dựng khá đơn giản. Tuy nhiên để các trình khách triệu gọi được đối tượng CORBA chúng ta cần phải có thao tác cài đặt và đăng ký đối tượng CORBA trên máy chủ.

4.2. Viết trình cài đặt đối tượng CORBA trên máy chủ

Bạn đã biết trong chương trước, các đối tượng RMI của Java muốn chạy được trước hết phải đăng ký tên mình với dịch vụ rmiregistry. Với CORBA cơ chế đăng ký tên cũng diễn ra tương tự. Trước khi viết trình đăng ký đối tượng CORBA trên máy chủ, trước hết bạn phải khởi động dịch vụ quản lý tên của CORBA. Dịch vụ này là chương trình tnameserv.exe nằm trong thư mục [JAVA_HOME]\BIN của bộ JDK. Mặc định dịch vụ tnameserv chạy trên cổng 900 và mang tên là NameService.

> *Bạn có thể chỉ định một tên hoặc số hiệu cổng khác cho dịch vụ này. Ví dụ để chạy tnameserv trên cổng 1020 bạn có thể gọi lệnh sau:*
>
> *tnameserv -ORBInitialPort 1020*

Ở phần sau ta sẽ nghiên cứu sâu hơn về dịch vụ tnameserv này. Còn bây giờ trước khi viết chương trình cài đặt đối tượng CORBA HelloServant ở ví dụ 3-2 chúng ta hãy khởi động tnameserv ở chế độ mặc định như sau (bạn mở một cửa sổ DOS-prompt khác bởi vì tương tự rmiregistry, tnameserv là một dịch vụ chạy ở hậu trường - background)

```
C:\Network\CORBA\Hello>tnameserv
Initial Naming Context:
IOR:000000000000002849444c3a6f6d672e6f72672f436f734e61
6d696e672f4e616d696e67436f6e746578743a312e300000000001
000000000000005000010100000000a3132372e302e302e310004
1300000018afabcafe00000002f514f586000000008000000000000
000000000010000000100000014000000000001002000000
TransientNameServer: setting port for initial object
references to: 900
Ready.
```

Bạn đừng cảm thấy hốt hoảng. Một chuỗi ký tự dài ngoằng chẳng qua là tnameserv muốn cho bạn biết dịch vụ đã khởi động thành công. Và như bạn thấy dịch vụ này đang lắng nghe các kết nối hay đăng ký gửi đến cổng 900.

Đã đến lúc xây dựng chương trình cài đặt và đăng ký đối tượng CORBA HelloServant của ta trên máy chủ. Chương trình Setup.java dưới đây sẽ thực hiện nhiệm vụ đó

Ví dụ 3-3 Setup.java

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
```

```
public class Setup
{
  public static void main(String args[])
                                    throws Exception{
    System.out.println("Setup CORBA Hello Object");
    //Khởi động trình môi giới đối tượng ORB
    ORB orb=ORB.init(args,null);
    //Tạo đối tượng CORBA
    HelloServant servant= new HelloServant();
    //Kết nối đối tượng CORBA với trình môi giới ORB
    orb.connect(servant);
    //Lấy tham chiếu đến dịch vụ đăng ký tnameserv
    org.omg.CORBA.Object nameService=
        orb.resolve_initial_references("NameService");
    // Lấy về các thông tin ngữ cảnh hay môi trường mà dịch
    // vụ tnameserv đang hoạt động.
    /* Bạn lưu ý: thật ra phương
       thức narrow() của lớp NamingContextHelper chỉ mang nhiệm
       vụ chuyển kiểu (type cast) đối tượng nameService về kiểu
       NamingContext mà thôi. Tuy nhiên trong CORBA thường bạn
       không được phép chuyển kiểu theo cách
          NamingContext  nsContext=(NamingContext)nameService
       phương thức narrow() được dùng để "chuyển kiểu an toàn".
       Nếu đọc mã nguồn của Java bạn sẽ thấy narrow() có khả năng
       chuyển đổi đối tượng theo nhiều cách tùy vào tình huống của
đối tượng.
    */
    NamingContext nsContext =
        NamingContextHelper.narrow(nameService);
    //Tạo tên của đối tượng
```

```
    NameComponent nc=new NameComponent("Hello","");
    //Tạo đường dẫn để lưu tên đối tượng
    NameComponent path[]={nc};
    //Ràng buộc đối tượng theo tên do dịch vụ tnameserv quản lý
    nsContext.rebind(path,servant);
    System.out.println("Waiting for client ...");
    // Tạo đối tượng Java
    java.lang.Object obj = new java.lang.Object();
    //Vòng lặp vô tận, chờ nhận yêu cầu từ máy khách
    synchronized (obj){
        obj.wait();
    }
  }
}
```

So với các chương trình RMI đăng ký đối tượng chủ trong chương trước bạn sẽ thấy chương trình đăng ký đối tượng CORBA ở ví dụ 3-3 trên đây có nhiều điểm tương đồng và khác biệt.

Đầu tiên do CORBA cần đến trình môi giới trung gian ORB cho nên trình đăng ký phải khởi động môi trường này bằng lệnh

```
ORB orb=ORB.init(args,null);
```

Tiếp đến đối tượng CORBA được tạo ra một cách bình thường

```
HelloServant servant= new HelloServant();
```

Để trình ORB biết đối tượng đang tồn tại bạn gọi phương thức `orb.connect()`. Phương thức này sẽ kết nối đối tượng với trình môi giới

```
orb.connect(servant);
```

Các đối tượng có chức năng phục vụ được cho trình khách thường được gọi với thuật ngữ servant hay "người phục vụ"

Nếu bạn nhớ lại cũng sẽ thấy rằng trong RMI khi muốn máy ảo Java biết một đối tượng có khả năng triệu gọi từ xa ta dùng lệnh

`UnicastRemoteObject.exportObject()`. Trong CORBA điều này được thay thế bằng lệnh `connect()` với ORB.

Dịch vụ tnameserv của CORBA được gọi là COS Naming (Common Object Service Naming). Thư viện chứa các lớp đối tượng truy xuất dịch vụ này nằm trong gói `org.omg.CosNaming.*`. Quá trình đăng ký đối tượng servant được thực hiện qua các bước:

Đầu tiên là phải lấy về nơi chốn mà dịch vụ tnameserv đang hiện hữu. Tiếp đến là tạo đường dẫn và tên của đối tượng servant yêu cầu dịch vụ đăng ký. Cuối cùng là nhờ dịch vụ ràng buộc tên và thể hiện của đối tượng với nhau. Các bước nêu trên được diễn đạt bằng đoạn mã sau:

```
//Lấy về tham chiếu của dịch vụ tên
org.omg.CORBA.Object nameService=
    orb.resolve_initial_references("NameService");
//Chuyển tham chiếu về đối tượng NamingContext
NamingContext nsContext =
    NamingContextHelper.narrow(nameService);
//Tạo tên của đối tượng
NameComponent nc=new NameComponent("Hello","");
//Tạo đường dẫn để lưu tên đối tượng
NameComponent path[]={nc};
//Yêu cầu dịch vụ tnamesvr ràng buộc đối tượng theo tên
nsContext.rebind(path,servant);
```

Thật ra đoạn lệnh trên đây của CORBA tương đương với lệnh sau trong RMI mà bạn đã biết qua ở chương trước:

```
Naming.rebind("//localhost/Hello");
```

Tuy nhiên Naming.rebind() là cách mà RMI quản lý tên còn COS Naming của CORBA lại dùng khái niệm NamingContext và NamingComponent. Bạn đừng lo có quá nhiều điều phải làm. Nếu bạn vẫn chưa hiểu rõ về COS Naming, chúng ta sẽ nghiên cứu sâu hơn về dịch vụ đăng ký tên này ở phần sau.

Không như đối tượng RMI chỉ cần gọi hàm Naming.rebind() là có thể ràng buộc đối tượng và chờ yêu cầu từ máy khách gửi đến. Lệnh

```
nsContext.rebind(path,servant);
```

không ngừng lại để chờ đối tượng được gọi. Chúng ta phải tự động thực hiện quá trình chờ này bằng cách tạo ra đối tượng java.lang.Object thuần túy và gọi thủ tục chờ.

```
java.lang.Object obj = new java.lang.Object();
//Vòng lặp vô tận, chờ nhận yêu cầu từ máy khách
synchronized (obj){
    obj.wait();
}
```

Bạn lưu ý đối tượng obj trên đây không liên quan gì đến đối tượng CORBA HelloServant. Nó chỉ có nhiệm vụ tạo ra vòng lặp while vô tận để tạm không cho chương trình chấm dứt mà thôi (chờ nhận yêu cầu từ trình khách).

Lưu Setup.java vào thư mục C:\Network\CORBA\hello thực hiện lệnh biên dịch

```
C:\Network\CORBA\hello>javac Setup.java
```

Để ý rằng trình tnameserv vẫn còn đang chạy. Bạn chạy và đăng ký đối tượng CORBA từ một cửa sổ DOS-Promt khác như sau:

```
C:\Network\CORBA\hello>java Setup
Setup CORBA Hello Object
Waiting for client ...
```

Như vậy là bạn đã có một máy chủ cài đặt và đăng ký thành công đối tượng CORBA HelloServant viết bằng Java. Ưu điểm của đối tượng CORBA hơn hẳn đối tượng RMI ở chỗ: đối tượng RMI chỉ gọi được bởi các trình khách viết bằng Java trong khi đối tượng CORBA có thể gọi được bởi các trình khách viết bằng nhiều ngôn ngữ khác nhau như C++, Delphi, Java ... Giáo trình này chủ yếu tìm hiểu Java nên tiếp theo đây chúng ta sẽ xây dựng một trình khách

cũng chính bằng Java để truy xuất đối tượng HelloServant vừa đăng ký. Tuy nhiên bạn hãy luôn nhớ rằng có thể dùng một ngôn ngữ khác (như C++) để thiết kế trình khách triệu gọi HelloServant.

4.3. Xây dựng trình khách triệu gọi đối tượng CORBA

Bạn cần những gì khi phải thiết kế trình khách triệu gọi đối tượng CORBA? Tất cả bạn cần chỉ là tập tin đặc tả .idl. Nhà cung cấp sẽ đưa cho bạn tập tin Hello.idl. Bạn dựa vào tập tin này để cài đặt và tìm cách triệu gọi đối tượng CORBA từ xa.

Trước tiên là tạo ra các lớp trung gian. Dùng trình idlj.exe để chuyển đặc tả IDL về các lớp đặc tả bằng ngôn ngữ Java. Để tượng trưng cho quá trình biên dịch phía máy khách độc lập với máy chủ bạn tạo thư mục C:\Network\CORBA\Hello\Client. Lưu tập tin Hello.idl vào thư mục này. Đánh lệnh biên dịch sau:

```
idlj -fclient Hello.idl
```

Lệnh trên yêu cầu chuyển đổi tập tin Hello.idl thành các tập tin thích hợp để thực hiện trên máy khách. Kết quả bạn sẽ thu được các tập tin sau:

```
HelloHelper.java
HelloHolder.java
Hello.java
HelloOperations.java
_HelloStub.java
```

So với quá trình biên dịch tập tin Hello.idl trên máy chủ chúng ta cũng thu được 2 tập tin Hello.java, HelloOperations.java. Tuy nhiên, bạn sẽ nhận thêm được hai tập tin khác dùng để hỗ trợ lời gọi trung gian đến đối tượng CORBA, đó là HelloHelper.java và HelloHolder.java. Thật sự thì bạn không đòi hỏi phải biết nội dung của những tập tin này (tuy vậy phần sau ta sẽ tìm hiểu chúng). _HelloStub.java chính là lớp trung gian thực hiện giao tiếp giữa trình ORB và đối tượng CORBA. Chúng ta cài đặt trình khách như sau.

Ví dụ 3-4 Client.java

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
public class Client
{
  public static void main(String args[])
                                    throws Exception{
    // Khởi động trình môi giới ORB
    ORB orb=ORB.init(args,null);
    // Tham chiếu đến dịch vụ tên tnameserv
    org.omg.CORBA.Object nameService=
        orb.resolve_initial_references("NameService");
    //Chuyển tham chiếu về đối tượng NamingContext
    NamingContext nsContext =
        NamingContextHelper.narrow( nameService);
    //Tạo đường dẫn mang tên đối tượng
    NameComponent nc=new NameComponent("Hello","");
    NameComponent path[]={nc};
    // Lấy về tham chiếu của đối tượng CORBA trên máy chủ dựa
vào tên
    Hello servant=HelloHelper.narrow(
                        nsContext.resolve(path));
    System.out.println( servant.sayHello() );
  }
}
```

Như đã nêu, cả trình khách và đối tượng chủ đều phải giao dịch với nhau thông qua trình môi giới ORB. Vì lý do này trình khách cũng như trình chủ trước tiên phải khởi tạo đối tượng ORB như sau:

```
ORB orb=ORB.init(args,null);
```

Tiếp đến trình khách gọi dịch vụ tnameserv để yêu cầu truy tìm tham chiếu của đối tượng CORBA trên máy chủ. Muốn yêu cầu tnameserv truy tìm một đối tượng nào đó bạn phải biết cách chỉ

đường dẫn đến đối tượng. Đầu tiên trình khách lấy về đường dẫn gốc của dịch vụ tnameserv

```
NamingContext nsContext =
    NamingContextHelper.narrow( nameService);
```

Với đường dẫn gốc nsContext bạn có thể tạo tiếp đường dẫn con đến đối tượng Hello.

```
//Tạo đường dẫn đến đối tượng
NameComponent nc=new NameComponent("Hello","");
NameComponent path[]={nc};
```

Dùng đường dẫn để lấy về tham chiếu của đối tượng.

```
// Lấy về tham chiếu của đối tượng CORBA trên máy chủ
Hello servant=HelloHelper.narrow(
                        nsContext.resolve(path));
```

Phương thức resolve() của nsContext sẽ nhận vào đường dẫn và trả về tham chiếu là một đối tượng CORBA tổng quát. Bạn lưu ý lệnh trên có thể được viết tách biệt như sau:

```
org.omg.CORBA.Object obj=nsContext.resolve(path);
Hello servant=HelloHelper.narrow(obj);
```

Một lần nữa bạn gặp lại phương thức narrow(). Nếu còn nhớ, đối tượng RMI được lấy về và chuyển kiểu theo cách.

```
Hello servant = (Hello)Naming.lookup(...);
```

Với CORBA bạn không thể thực hiện lệnh tương tự

```
Hello servant=(Hello) nsContext.resolve(path);
```

Để chuyển một đối tượng tổng quát về một kiểu đối tượng tường minh thường CORBA cung cấp cho bạn phương thức narrow() tương ứng nằm trong lớp hỗ trợ (lớp này thường mang tên _Helper). Bạn hãy làm quen với phong cách lập trình này trong CORBA và cả những mã lệnh sau này của Java cũng phát triển theo hướng trên. Cụ thể nếu mở mã nguồn (tập tin HelloHelper.java) bạn sẽ thấy phương thức narrow() có nội dung sau:

```
public static Hello narrow (org.omg.CORBA.Object obj)
{
  if (obj == null)
     return null;
  else if (obj instanceof Hello)
    return (Hello)obj;
  else if (!obj._is_a (id ()))
   throw new org.omg.CORBA.BAD_PARAM ();
  else
  {
   org.omg.CORBA.portable.Delegate delegate =
     ((org.omg.CORBA.portable.ObjectImpl)
                          obj)._get_delegate ();
   return new _HelloStub (delegate);
  }
 }
```

Bạn thấy đó, phương thức narrow() của lớp HelloHelper mục đích là chuyển đổi tượng CORBA (obj) tổng quát về kiểu đối tượng Hello. Tuy nhiên quá trình chuyển kiểu này xảy ra khá phức tạp. Điều mà bạn không thể thực hiện nếu chỉ bằng cách ép kiểu thông thường.

Bạn hãy lưu Client.java cùng vào thư mục

```
C:\Network\CORBA\Hello\Client
```

Thực hiện lệnh biên dịch sau

```
C:\Network\CORBA\Hello\Client>javac *.java
```

Bạn mở một cửa sổ DOS-Prompt khác chạy trình khách từ dòng lệnh

```
C:\Network\CORBA\Hello\Client>java Client
Hello CORBA
```

Chúc mừng bạn, chương trình CORBA khách/chủ của bạn đã hoạt động tốt đẹp.

4.4. Các cách khởi động trình môi giới ORB

Ở ví dụ trên chúng ta đã khởi động trình môi giới ở hai phía khách chủ bằng cách gọi lệnh

```
ORB orb=ORB.init(args,null);
```

Phương thức tĩnh init() của lớp ORB cần hai đối số. Đối số thứ nhất dùng để nhận các tùy chọn từ dòng lệnh nếu có. Đối số thứ hai là các tùy chọn do lập trình viên chỉ định. Nếu cả hai đối số này đều mang trị null thì ORB sẽ sử dụng các thông số hệ thống mặc định.

Trong trường hợp Applet, muốn kết nối với trình ORB bạn có thể khởi tạo phương thức ORB.init() theo cách sau:

```
public void init() {
   try {
      ORB orb = ORB.init(this, null);
      ...// code continues
```

Trong trường hợp bạn không dùng các giá trị mặc định của ORB chúng ta cần khởi tạo các đối số cho phương thức ORB.init(). Ví dụ để chương trình tnameserv khởi động trên cổng 1050 tại máy chủ có địa chỉ IPlà 172.16.11.12 bạn khởi động dòng lệnh như sau:

```
C:\jdk1.3\bin> tnameserv -ORBInitialPort 1050
```

Để trình khách từ xa kết nối được với dịch vụ này bạn truyền tham số dòng lệnh khi chạy trình khách theo cách sau:

```
C:\Network\CORBA\Hello\Client>java
        -ORBInittialPort 1050
        -ORBInitialHost 172.16.11.12 Client
```

Bạn có thể chỉ định các tùy chọn để ORB khởi động ngay trong mã lệnh như sau:

```
import org.omg.CORBA.*;
import java.util.Properties;
public class Client {
 public static void main(String args[]){
    Properties props = new Properties();
```

```
        props.put("org.omg.CORBA.ORBInitialPort ",
                                        "1050");
        props.put("org.omg.CORBA.ORBInitialHost ",
                                "172.16.11.12");
        ORB orb = ORB.init(args, props);
        ...
    }
}
```

Trong trường hợp bạn chỉ định cả hai tùy chọn: vừa từ tham số dòng lệnh, vừa trong mã lệnh thì ORB sẽ ưu tiên lấy các tùy chọn do mã lệnh thiết lập.

4.5. Tìm hiểu cách đối tượng CORBA được cài đặt

Trong quá trình xây dựng đối tượng CORBA, để đơn giản hóa chúng ta đã bỏ qua việc tìm hiểu các tập tin trung gian mà trình idlj.exe của java sinh ra.

4.5.1. Các tập tin trung gian cài đặt phía máy chủ

Dưới đây là nội dung của các tập tin .java khi bạn đánh lệnh

```
idlj -fserver Hello.idl
```

Ví dụ 3-5 HelloOperations.java

```
/**
* HelloOperations.java
* Generated by the IDL-to-Java compiler
*                        (portable), version "3.0"
* from Hello.idl
* Friday, July 10, 1998 10:49:18 PM GMT+01:00
*/
public interface HelloOperations
{
    String sayHello ();
} // interface HelloOperations
```

Ví dụ 3-6 Hello.java

```
/**
* Hello.java
* Generated by the IDL-to-Java compiler
*                                (portable), version "3.0"
* from Hello.idl
* Friday, July 10, 1998 10:49:18 PM GMT+01:00
*/
public interface Hello extends HelloOperations,
                        org.omg.CORBA.Object,
                        org.omg.CORBA.portable.IDLEntity
{
} // interface Hello
```

Ví dụ 3-7 _HelloImplBase

```
/**
* _HelloImplBase.java
* Generated by the IDL-to-Java compiler
*                                (portable), version "3.0"
* from Hello.idl
* Monday, June 28, 1999 9:37:31 PM PDT
*/
public abstract class _HelloImplBase
        extends
                org.omg.CORBA.portable.ObjectImpl
        implements
                Hello,
                org.omg.CORBA.portable.InvokeHandler
{

 // Constructors
 public _HelloImplBase ()
 {
 }
```

```
private static java.util.Hashtable _methods =
                        new java.util.Hashtable ();
static
{
_methods.put ("sayHello", new java.lang.Integer (0));
}
public org.omg.CORBA.portable.OutputStream _invoke
        (String method,
         org.omg.CORBA.portable.InputStream in,
         org.omg.CORBA.portable.ResponseHandler rh)
{
 org.omg.CORBA.portable.OutputStream out = null;
 java.lang.Integer __method =
        (java.lang.Integer)_methods.get (method);
 if (__method == null)
  throw new org.omg.CORBA.BAD_OPERATION (0,
     org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);
 switch (__method.intValue ())
 {
  case 0: // Hello/sayHello
  {
   String __result = null;
   __result = this.sayHello ();
   out = rh.createReply();
   out.write_string (__result);
   break;
  }
  default:
   throw new org.omg.CORBA.BAD_OPERATION (0,
    org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);
 }
 return out;
} // _invoke
```

```
 // Type-specific CORBA::Object operations
 private static String[] __ids = { "IDL:Hello:1.0"};
 public String[] _ids ()
 {
  return __ids;
 }

} // class _HelloImplBase
```

Đừng lo lắng, lần đầu tiên đọc phải những dòng mã này ngay cả những lập trình viên kinh nghiệm vẫn phải dành ra ít nhất 1 ngày để hiểu và nghiền ngẫm hương vị café Java nhúng trong CORBA. Chúng ta hãy đi vào những bước cơ bản nhất.

Tập tin Hello.idl của bạn được viết bằng CORBA như sau:

```
interface Hello{
 string sayHello();
};
```

Trình biên dịch idlj tạo ra tập tin HelloOperations.java có khai báo interface bằng ngôn ngữ Java hoàn toàn giống với những gì bạn đặc tả bằng ngôn ngữ IDL.

```
public interface HelloOperations
{
     String sayHello ();
}
```

Bên cạnh đó, idlj cũng sinh ra tập tin Hello.java chứa khai báo giao tiếp interface Hello như sau:

```
public interface Hello extends HelloOperations,
                    org.omg.CORBA.Object,
                    org.omg.CORBA.portable.IDLEntity
{
} // interface Hello
```

Mục đích của interface Hello là chuyển giao tiếp interface HelloOperations từ thuần Java sang CORBA. Như bạn thấy, ngoài

việc dẫn xuất từ interface HelloOperations, interface Hello không đặc tả thêm gì cả (interface rỗng). Tuy nhiên interface Hello thực hiện kế thừa thêm 2 giao tiếp interface khác đó là: org.omg.CORBA.Object và org.omg.CORBA.portable.IDLEntity. Đây chính là 2 interface tạo nên bộ mặt CORBA cho Hello.

Để tạo ra đối tượng CORBA cho Hello bạn không cài đặt trực tiếp lớp giao tiếp này. Thay vào đó idlj cung cấp cho bạn lớp _HelloImplBase như sau:

```
public abstract class _HelloImplBase
        implements
              Hello,
              org.omg.CORBA.portable.InvokeHandler
        extends
              org.omg.CORBA.portable.ObjectImpl
{
  ...
}
```

_HelloImplBase là lớp trừu tượng (abstract class). Trong lớp trừu tượng này, một số đặc tính của CORBA đã được cài đặt nhờ vào việc kế thừa lớp `org.omg.CORBA.portable.ObjectImpl`. Hơn nữa, do được nhập thêm giao tiếp Hello, _HelloImplBase cung cấp khả năng để bạn cài đặt cho interface Hello ở bất kỳ lớp con nào kế thừa _HelloImplBase.

Công việc còn lại của lập trình viên khi xây dựng đối tượng CORBA chỉ là kế thừa lớp _HelloImplBase. Thực hiện cài đặt cho phương thức sayHello() (như đặc tả trong Hello.idl) theo cách sau:

```
public class HelloServant extends _HelloImplBase
{
   public String sayHello(){
    return "Hello CORBA";
   }
}
```

Tóm lại về mặt kỹ thuật, để xây dựng đối tượng CORBA trong Java bạn thực hiện các trình tự sau:

• Viết đặc tả đối tượng bằng ngôn ngữ IDL của CORBA. Ví dụ:

```
interface MyObject{
  method1();
  method2();
};
```

• Dùng trình chuyển ngữ idlj.exe (hoặc idl2java.exe) của Java chuyển đặc tả IDL thành ngôn ngữ Java. Ví dụ :

```
idlj -fserver MyObject.idl
```

• Cài đặt đối tượng CORBA bằng cách kế thừa lớp _MyObjectImplBase. Ví dụ _MyObjectImplBase :

```
public class MyObjectServant
                    extends _MyObjectImplBase{
          method1(){
                    //Cài đặt nội dung cho method1()
          }
          method1(){
                    //Cài đặt nội dung cho method2()
          }
}
```

Để đăng ký đối tượng CORBA với trình ORB trên máy chủ bạn xem lại mục 4.2.

4.5.2. Các tập tin trung gian cài đặt phía máy khách

Khi thiết kế trình khách (ở mục 4.3) triệu gọi đối tượng CORBA, ta dùng idlj.exe dịch tập tin Hello.idl thành các tập tin trung gian cần thiết cho máy khách bằng lệnh:

```
idlj -fclient Hello.idl
```

kết quả idlj sẽ sinh ra các tập tin .java sau đây:

Ví dụ 3-8 HelloOperations.java

```
/**
* HelloOperations.java
* Generated by the IDL-to-Java compiler
*                                (portable), version "3.0"
* from Hello.idl
* Friday, July 10, 1998 11:17:56 PM GMT+01:00
*/
public interface HelloOperations
{
    String sayHello ();
} // interface HelloOperations
```

Ví dụ 3-9 Hello.java

```
/**
* Hello.java
* Generated by the IDL-to-Java compiler
*                                (portable), version "3.0"
* from Hello.idl
* Friday, July 10, 1998 11:17:56 PM GMT+01:00
*/
public interface Hello
        extends HelloOperations,
                org.omg.CORBA.Object,
                org.omg.CORBA.portable.IDLEntity
{
} // interface Hello
```

Ví dụ 3-10 HelloHolder.java

```
/**
* HelloHolder.java
* Generated by the IDL-to-Java compiler
*                                (portable), version "3.0"
* from Hello.idl
* Friday, July 10, 1998 11:17:56 PM GMT+01:00
```

```
*/
public final class HelloHolder  implements
                    org.omg.CORBA.portable.Streamable
{
 public Hello value = null;
 public HelloHolder ()
 {
 }
 public HelloHolder (Hello initialValue)
 {
  value = initialValue;
 }
 public void _read
            (org.omg.CORBA.portable.InputStream i)
 {
  value = HelloHelper.read (i);
 }
 public void _write(
            org.omg.CORBA.portable.OutputStream o)
 {
  HelloHelper.write (o, value);
 }
 public org.omg.CORBA.TypeCode _type ()
 {
  return HelloHelper.type ();
 }
}
```

Ví dụ 3-11 HelloHelper.java

```
/**
* HelloHelper.java
* Generated by the IDL-to-Java compiler
*                          (portable), version "3.0"
* from Hello.idl
* Friday, July 10, 1998 11:17:56 PM GMT+01:00
```

```
*/
abstract public class HelloHelper
{
 private static String _id = "IDL:Hello:1.0";

 public static void insert (
                    org.omg.CORBA.Any a, Hello that)
 {
  org.omg.CORBA.portable.OutputStream
                    out = a.create_output_stream ();
  a.type (type ());
  write (out, that);
 a.read_value (out.create_input_stream (), type ());
 }
 public static Hello extract (org.omg.CORBA.Any a)
 {
  return read (a.create_input_stream ());
 }
 private static org.omg.CORBA.TypeCode
                                  __typeCode = null;
 synchronized public static org.omg.CORBA.TypeCode
                                              type ()
 {
  if (__typeCode == null)
  {
   __typeCode =
     org.omg.CORBA.ORB.init ().create_interface_tc (
                    HelloHelper.id (), "Hello");
  }
  return __typeCode;
 }
 public static String id ()
 {
  return _id;
```

```
}
public static Hello read(
        org.omg.CORBA.portable.InputStream istream)
{
  return narrow (istream.read_Object (
                                _HelloStub.class));
}
public static void write (
    org.omg.CORBA.portable.OutputStream ostream,
     Hello value)
{
ostream.write_Object ((org.omg.CORBA.Object) value);
}
public static Hello narrow (org.omg.CORBA.Object obj)
{
  if (obj == null)
    return null;
  else if (obj instanceof Hello)
    return (Hello)obj;
  else if (!obj._is_a (id ()))
    throw new org.omg.CORBA.BAD_PARAM ();
  else
  {
    org.omg.CORBA.portable.Delegate delegate = (
      (org.omg.CORBA.portable.ObjectImpl)
                                obj)._get_delegate ();
    return new _HelloStub (delegate);
  }
}


}
```

Ví dụ 3-12 _HelloStub.java

```
/**
* _HelloStub.java
```

```
* Generated by the IDL-to-Java compiler (portable),
*                                        version "3.0"
* from Hello.idl
* Wednesday, June 30, 1999 9:46:07 PM PDT
*/
public class _HelloStub      extends
        org.omg.CORBA.portable.ObjectImpl
        implements Hello
{
 // Constructors
 // NOTE: If the default constructor is used, the
 //     object is useless until _set_delegate (...)
 //     is called.
 public _HelloStub ()
 {
  super ();
 }
 public _HelloStub (org.omg.CORBA.portable.Delegate
                                        delegate)
 {
  super ();
  _set_delegate (delegate);
 }
 public String sayHello ()
 {
  org.omg.CORBA.portable.InputStream _in = null;
  try {
   org.omg.CORBA.portable.OutputStream _out =
                    _request ("sayHello", true);
   _in = _invoke (_out);
   String __result = _in.read_string ();
   return __result;
  } catch (org.omg.CORBA.portable.ApplicationException
                                    _ex) {
```

```
    _in = _ex.getInputStream ();
    String _id = _ex.getId ();
    throw new org.omg.CORBA.MARSHAL (_id);
  } catch (org.omg.CORBA.portable.RemarshalException
                                    _rm) {
    return sayHello ();
  } finally {
    _releaseReply (_in);
  }
} // sayHello
// Type-specific CORBA::Object operations
private static String[] __ids = {
 "IDL:Hello:1.0"};
public String[] _ids ()
{
 return (String[])__ids.clone ();
}
private void readObject (java.io.ObjectInputStream s)
{
  try
  {
    String str = s.readUTF ();
    org.omg.CORBA.Object obj =
      org.omg.CORBA.ORB.init ().string_to_object (str);
    org.omg.CORBA.portable.Delegate delegate =
          ((org.omg.CORBA.portable.ObjectImpl)
                                    obj)._get_delegate ();
    _set_delegate (delegate);
  } catch (java.io.IOException e) {}
}
private void writeObject (java.io.ObjectOutputStream
                                                  s)
{
  try
```

```
    {
     String str =
     org.omg.CORBA.ORB.init ().object_to_string (this);
     s.writeUTF (str);
    } catch (java.io.IOException e) {}
  }
} // class _HelloStub
```

Như bạn thấy, idlj sinh ra cho bạn tập tin Hello.java và HelloOperations.java giống như trên máy chủ. Tuy nhiên, phía máy khách idlj tạo thêm tập tin HelloHelper.java và HelloHolder.java. Lớp HelloHelper cung cấp cho bạn một số phương thức tiện ích để xử lý đối tượng Hello (như phương thức narrow() dùng để ép kiểu). Lớp HelloHolder chịu trách niệm xử lý việc chuyển đối tượng Hello qua lại giữa trình trung gian ORB trên máy khách và máy chủ. Cuối cùng trình khách được hỗ trợ thêm lớp _HelloStub dùng làm lớp trung gian để giao tiếp giữa trình khách với trình môi giới trung gian ORB.

Trong số các lớp trên, trình khách của ta chỉ thực sự dùng đến lớp Hello và HelloHelper để tham chiếu đến đối tượng (như bạn thấy ở ví dụ 3-4 mục 4-3) theo cách sau:

```
Hello servant=HelloHelper.narrow(
                        nsContext.resolve(path));
System.out.println( servant.sayHello() );
```

CORBA không yêu cầu bạn quan tâm đến việc cài đặt của các tập tin trung gian như HelloHelper, HelloHolder, _HelloStub. Thật sự thì mỗi nhà cung cấp ORB khác nhau (như Inprise, IBM hay Sun) đều có cách cài đặt bên trong những lớp trung gian này khác nhau. Ở các ví dụ sau này chúng ta sẽ không trình bày lại các tập tin trung gian của cả hai phía máy khách và máy chủ. Chúng sẽ đơn giản hóa cho bạn hơn khi nghiên cứu và cài đặt đối tượng CORBA trong Java.

5. TÌM HIỂU JNDI VÀ DỊCH VỤ QUẢN LÝ TÊN COS NAMING CỦA CORBA

5.1. JNDI và dịch vụ Naming trong RMI

Khi bạn muốn sử dụng một đối tượng nào đó trong cuộc sống điều đầu tiên bạn cần phải biết là nơi mà vật đó đang được cất giữ (hay đặt để). Làm thế nào để tìm ra đối tượng? Làm thế nào để lưu lại đối tượng sau khi sử dụng?... Mỗi đối tượng trong thế giới thực đều được mang một tên gọi nhất định. Ít ra hai vật thể khác nhau cũng có thể phân biệt với nhau theo tên gọi (các vật thể trùng nhau về tính chất cũng có thể phân biệt qua một đặc trưng khác biệt nào đó mà trong môi trường cơ sở dữ liệu ta vẫn quen gọi là khóa – key). Mặc khác, cần phải có một hệ thống lưu trữ những tên gọi này theo cách nào đó để khi cần bạn có thể dựa vào tên để tìm ra đối tượng thật dễ dàng.

Khái niệm trên chúng ta gặp rất nhiều trong cuộc sống. Như danh bạ điện thoại là một hệ thống lưu trữ tên dựa vào số điện thoại của người dùng. Bản đồ là một hệ thống tên gắn liền với các thành phố, đường xá. Hệ thống file là một dịch vụ quản lý tên của thư mục và tập tin trên đĩa, DNS là dịch vụ gắn liền địa chỉ IP của máy tính với một tên gọi nhớ nào đó.

Mỗi dịch vụ quản lý tên đều qui định cách thức để bạn lưu trữ và gắn tên cho đối tượng. Ví dụ trong danh bạ điện thoại bạn lưu tên gắn với số điện thoại theo thứ tự ABC. Trên bản đồ, tên của các thành phố đường xá được phân theo quận huyện. Hệ thống file thường lưu tập tin và thư mục theo cấu trúc hình cây. Với mỗi cách lưu trữ như vậy, dịch vụ quản lý tên luôn cung cấp cho bạn cách thức tìm kiếm và đặt tên cho đối tượng cần quản lý một cách nhất định.

Tổng quát hoá Java đặc tả một tập hợp các hàm API dùng để phục vụ cho việc quản lý và lưu tên đối tượng. Các hàm API này gọi là JNDI (Java Naming Directory Interface). JNDI cung cấp cách đặt

tên, lưu trữ và tìm kiếm đối tượng một cách tổng quát nhất. Với JNDI bạn có thể gọi cùng một phương thức để tìm kiếm tập tin trên hệ thống file, thư mục. tên của địa chỉ mạng Internet, tên của các đối tượng RMI, CORBA ... JNDI chỉ đưa ra đặc tả (ở dạng interface). Các dịch vụ nào muốn sử dụng JNDI thì phải cài đặt giao tiếp do Java qui định. Dịch vụ Naming mà bạn có dịp sử dụng khi nghiên cứu về RMI ở chương trước chính là một tập các cài đặt giao tiếp JNDI dành cho việc quản lý đối tượng RMI trong Java. Với sự hỗ trợ của dịch vụ quản lý tên rmiregistry.exe, bạn có thể gọi Naming.bind() để ràng buộc một đối tượng nào đó theo tên. Ví dụ tên myhello được ràng buộc với đối tượng hello như sau:

```
Naming.bind("rmi://localhost/myhello", hello);
```

Dịch vụ đặt tên Naming chỉ hỗ trợ cho các đối tượng RMI trong Java. Naming có cách truy xuất theo địa chỉ URL dựa vào tên đối tượng khi đăng ký. Ví dụ, để lấy về tham chiếu của một đối tượng RMI, bạn gọi hàm lookup() của dịch vụ RMI Naming theo cách:

```
Object o= Naming.lookup("rmi://localhost/myhello");
```

Quá trình dựa vào tên của đối tượng để tìm lại được tham chiếu của đối tượng (đã được dịch vụ quản lý tên lưu trước đó) gọi là *phân giải đối tượng* (object resolving).

5.2. JNDI và dịch vụ COS Naming của CORBA

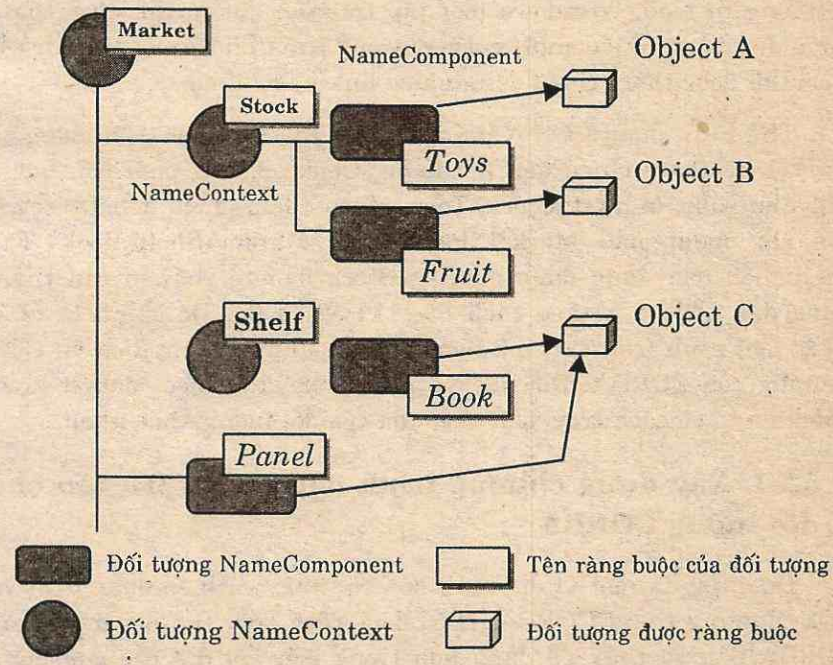
Cũng dựa vào đặc tả JNDI của Java nhưng CORBA xây dựng một cơ chế lưu trữ tên và tham chiếu của đối tượng khác với dịch vụ Naming của RMI.

CORBA phân ra khái niệm NamingContext và NamingComponent tương tự khái niệm file và thư mục mà ta vẫn quen thuộc.

NamingContext là một đối tượng có khả năng dẫn tham chiếu đến một NamingComponent hay NamingContext khác (NamingContex đóng vai trò như một thư mục) NamingContext được đặt tên dựa vào NamingComponent.

NamingComponent là một thành phần chứa tên (tương tự nhãn hoặc tên file) có khả năng ràng buộc tên với một đối tượng bất kỳ nào đó (đối tượng này phải là đối tượng CORBA).

NamingComponent và NamingContext quan hệ với nhau theo cấu trúc hình cây tương tự cấu trúc cây thư mục của hệ thống file trên đĩa. Hình 3-4 là một ví dụ về cách lưu trữ của dịch vụ COS Naming trong CORBA. Dịch vụ này chính là chương trình tnameserv.exe mà ta đã có dịp tìm hiểu qua khi cài đặt chương trình in ra chuỗi "Hello CORBA".



Hình 3-4 Mô hình ràng buộc và lưu trữ đối tượng của dịch vụ *COS Naming*

Như bạn thấy ở hình 3-4 **Market** là một đối tượng NamingContext. Nó cho phép bạn tìm ra 3 thành phần tên: **Stock**, **Shelf** và *Panel*. Bản thân **Stock**, **Shelf** là hai NamingContext. **Stock** chứa 2 thành phần mang tên *Toys* và *Fruit*. **Shelf** chứa một thành phần mang tên *Book*. Đối tượng Object A được ràng buộc với tên *Toys* trong khi đối tượng Object B, Object C được ràng buộc bởi tên *Fruit* và *Book*.

Một đối tượng không chỉ được ràng buộc bởi một tên mà có thể nhiều tên. Ví dụ cả *Panel* và *Book* cùng ám chỉ đến đối tượng Object C (tương tự trong Windows một tập tin trên đĩa có thể được tham chiếu đến bằng nhiều short cut hay trong Unix/Linux một tập tin vật lý có thể được tham chiếu bằng nhiều link khác nhau).

Thử hình dung COS Naming là một cây thư mục. Nếu đứng từ Market để tham chiếu đến đối tượng Object A bạn phải chỉ đường dẫn như sau: Market\Stock\Toys. Muốn tham chiếu đến Object C bạn chỉ đường dẫn Market\Panel hoặc Market\Shelf\Book. Tuy nhiên nếu bạn đang đứng ở vị trí Stock bạn có thể chỉ cần tham chiếu đến Object A theo cách Stock\Toys là đủ. Đó chính là khái niệm ngữ cảnh (context) mà bạn sẽ gặp sau này trong dịch vụ COS Naming của CORBA. Đối với một đối tượng, tùy theo context khác nhau mà ta có cách truy xuất theo tên của đối tượng khác nhau.

5.2.1. Xây dựng chương trình đặt và lưu trữ tên cho đối tượng CORBA

Dưới đây là một ví dụ minh hoạ cho thấy cách sử dụng dịch vụ COS Naming trong CORBA. Hiểu được cách truy tìm đối tượng của CORBA bạn sẽ thấy dễ dàng hơn trong việc cài đặt các ứng dụng CORBA như ta đã làm ở ví dụ 3-3.

Chương trình sử dụng mô hình ở hình 3-4 làm minh họa. Chúng ta lần lượt tạo ra các đối tượng Object A, Object B, Object C đồng thời xây dựng cách đặt tên và lưu trữ những đối tượng này theo

cấu trúc hình cây như đã vẽ. Chương trình được xây dựng qua các bước sau:

Bước 1: Đặc tả một đối tượng CORBA để gắn với các thành phần tên trong cây.

Ví dụ 3-13 Generic.idl

```
interface Generic{
   string getName();
};
```

Đối tượng Generic của ta đơn giản chỉ có nhiệm vụ trả về tên của chính bản thân nó mà thôi. Bạn dịch file Generic.idl bằng trình idlj như sau:

```
idlj -fserver Generic.idl
```

Tiếp đến là cài đặt cho Generic trở thành đối tượng CORBA với mã nguồn GenericServant sau đây:

Ví dụ 3-14 GenericServant.java

```
public class GenericServant extends _GenericImplBase
{
   String name;
   //Phương thức khởi dựng tiếp nhận và đặt định danh cho đối tượng
   public GenericServant (String name){
      this.name =name;
   }
   //Trả về tên của đối tượng
   public String getName (){
      return name;
   }
}
```

Bước 2: Tạo trình khách đăng ký tên của các đối tượng Generic với dịch vụ COS Naming.

Ví dụ 3-15 NamingRegister.java

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
public class NamingRegister
{
  public static void main(String args[])
                                        throws Exception{
    System.out.println("COS Naming Demo Application");
    System.out.println(" Create Corba Object");
    //Tạo các đối tượng CORBA Generic
    GenericServant objectA=new
                    GenericServant("This is Object A");
    GenericServant objectB=new
                    GenericServant("This is Object B");
    GenericServant objectC=new
                    GenericServant("This is Object C");
    //Khởi động trình môi giới ORB
    ORB orb=ORB.init(args,null);
    System.out.println(" Getting Corba root context");
    //Lấy về đối tượng context gốc của dịch vụ COS Naming
    NamingContext ctx = NamingContextHelper.narrow(
          orb.resolve_initial_references(
                                        "NameService"));
    //Tạo thành phần NameComponent mang tên Market
    System.out.println(
                    " Create Market context under root");
    NameComponent market = new NameComponent(
                        "Market", "Market folder");
    //Tạo đường dẫn đến Market
    NameComponent[] path = {market};
    //Yêu cầu context gốc tạo context mới mang tên Market
```

```
NamingContext marketctx=
                    ctx.bind_new_context(path);
System.out.println(
                " Create Stock context under Market");
//Tạo thành phần NameComponent mang tên Stock
NameComponent stock = new NameComponent(
                "Stock", "Stock folder");
//Tạo đường dẫn đến Stock (đường dẫn này được tính từ
context gốc)
NameComponent[] path1 = {market,stock};
//Yêu cầu context gốc tạo context mới mang tên Stock
NamingContext stockctx=
                    ctx.bind_new_context(path1);
System.out.println(" Create and bound object
                        with Stock context");
//Tạo thành phần NameComponent mang tên Toys
NameComponent toys = new NameComponent("Toys",
                        "object id");
//Tạo thành phần NameComponent mang tên Fruit
NameComponent fruit = new NameComponent("Fruit",
                        "object id");
//Tạo đường dẫn đến tên Toys (đường dẫn này tính từ context
gốc)
NameComponent[] path11 = {market,stock,toys};
//Tạo đường dẫn đến tên Fruit (đường dẫn này tính từ context
gốc)
NameComponent[] path12 = {market,stock,fruit};
//Yêu cầu context gốc ràng buộc đối tượng ObjectA
//với thành phần mang tên Toys theo đường dẫn chỉ định
ctx.rebind(path11,objectA);
//Yêu cầu context gốc ràng buộc đối tượng ObjectB
//với thành phần mang tên Fruit theo đường dẫn chỉ định
```

```
ctx.rebind(path12,objectB);
System.out.println(" Create Shelf context
                                under Market");

//Tạo thành phần NameComponent mang tên Shelf
NameComponent shelf = new NameComponent("Shelf",
                                "Shelf folder");

//Tạo đường dẫn đến tên Shelf
// (bạn lưu ý ở đây đường dẫn được tính từ context market)
NameComponent[] path2 = {shelf};

//Yêu cầu context Market tạo context mới mang tên Shelf
NamingContext shelfctx =
            marketctx.bind_new_context(path2);
System.out.println(" Create and bound object
                            with Shelf context");

//Tạo thành phần NameComponent mang tên Book
NameComponent book = new NameComponent("Book",
                                "object id");

//Tạo đường dẫn đến tên Book (tính từ context Shelf)
NameComponent[] path21 = {book};

//Yêu cầu context Shelf ràng buộc đối tượng ObjectC
//với thành phần mang tên Book theo đường dẫn chỉ định
shelfctx.rebind(path21,objectC);
System.out.println(" Create and bound object with
                                Market context");

//Tạo thành phần NameComponent mang tên Panel
NameComponent panel = new NameComponent("Panel",
                                "object id");

//Tạo đường dẫn đến tên Panel (tính từ context gốc)
NameComponent[] path3 = {pane};

//Yêu cầu context Shelf ràng buộc đối tượng ObjectC
//với thành phần mang tên Panel theo đường dẫn chỉ định
```

```
    marketctx.rebind(path3,objectC);
    System.out.println("rebind successfull");
    System.out.println("Waiting for client ...");
    // Tạo đối tượng Java
    java.lang.Object obj = new java.lang.Object();
    //Vòng lặp vô tận, chờ nhận yêu cầu từ máy khách
    synchronized (obj){
        obj.wait();
  }
}
```

Tương tự như cây thư mục của hệ điều hành có thư mục gốc, dịch vụ COS Naming cung cấp cho bạn một context gốc mang tên "NameService". Bạn lấy về context này bằng cách:

- Khởi tạo kết nối với trình môi giới trung gian

```
orb=ORB.init(args,null)
```

- Gọi phương thức `orb.resolve_initial_references()` của trình môi giới ORB. Phương thức `orb.resolve_initial_references()` nhận vào tên của đối tượng context gốc ("NameService) và trả về đối tượng context. Phương thức NamingContextHelper.narrow() có nhiệm vụ chuyển đổi tượng context về đúng kiểu lớp NamingContext để bạn sử dụng.

```
NamingContext ctx = NamingContextHelper.narrow(
        orb.resolve_initial_references(
                                "NameService"));
```

Với đối tượng context gốc (ctx) có trong tay bạn có thể lần lượt tạo ra các tham chiếu đến các context hoặc thành phần ràng buộc tên NamingComponent khác.

Trong ví dụ trên chúng ta muốn tạo một context mới mang tên Market để lưu trữ các thành phần tên. Trước tiên ta dùng đối tượng NamingComponent để tạo tên cho context:

```
NameComponent market = new NameComponent(
                    "Market", "Market folder");
```

phương thức khởi dựng của NamingComponent yêu cầu hai đối số. Đối số thứ nhất là tên mà bạn muốn đặt. Đối số thứ hai là định danh id để nhận dạng đối tượng. Cả hai đối số này đều ở dạng chuỗi và do bạn tự quyết định.

Một đối tượng context thường có hai khả năng. Khả năng tạo ra các context khác và khả năng ràng buộc tên của thành phần NamingComponent với một đối tượng nào đó vào context. Nhưng trước khi muốn context thực hiện được điều này chúng ta phải cung cấp đường dẫn tương ứng. Với thành phần tên Market ta muốn tạo ra đường dẫn được chỉ định từ context gốc như sau:

```
//Tạo đường dẫn đến Market
NameComponent[] path = {market};
```

Sau cùng là tạo mới đối tượng context Market dựa trên đường dẫn path. Bạn gọi phương thức `bind_new_context()` của đối tượng context gốc để thực hiện điều này.

```
//Yêu cầu context gốc tạo context mới mang tên Market
NamingContext marketctx=
                    ctx.bind_new_context(path);
```

Quá trình tạo các context con khác như Stock và Shelf cũng diễn ra tương tự.

Để ràng buộc đối tượng ObjectA vào thành phần NamingComponent của context. Bạn thực hiện như sau

- Đặt tên cho đối tượng thông qua NamingComponent

```
NameComponent toys = new NameComponent("Toys",
                    "object id");
```

- Tạo đường dẫn để ràng buộc NamingComponent vào context

```
NameComponent[] path11 = {market,stock,toys};
```

- Gọi phương thức rebind() của context để ràng buộc tên với đối tượng

```
ctx.rebind(path11,objectA);
```

Lưu ý tùy theo context sử dụng mà bạn tạo đường dẫn thích hợp. Ví dụ hai lệnh sau đây là tương đương nhau (chúng cùng ràng buộc đối tượng ObjectB với tên Fruit và đặt Fruit vào context Stock)

```
NameComponent[] path12 = {market,stock,fruit};
ctx.rebind(path12,objectB);
```

và

```
NameComponent[] path12 = {stock,fruit};
marketctx.rebind(path12,objectB);
```

cặp lệnh đầu tiên dùng context gốc để ràng buộc đối tượng, còn cặp lệnh thứ hai dùng context Market. Vì lý do này mà đường dẫn tương đối (path12) của chúng tuy khác nhau nhưng đều dẫn đến một tên duy nhất là Fruit.

Bước 3: Biên dịch và chạy chương trình. Bạn lưu tất cả mã nguồn vào thư mục C:\Network\corba\naming. Chuyển vào thư mục này đánh lệnh biên dịch sau:

```
C:\Network\corba\naming>javac *.java
```

Khởi động dịch vụ COS Naming từ một cửa sổ DOS-Prompt khác

```
C:\Network\corba\naming>tnameserv
```

Chạy chương trình:

```
C:\Network\corba\naming>java NamingRegister
```

Kết quả kết xuất:

```
C:\Network\corba\Naming>java NamingRegister
COS Naming Demo Application
 Create Corba Object
 Getting Corba root context
 Create Market context under root
 Create Stock context under Market
```

```
Create and bound object with Stock context
Create Shelf context under Market
Create and bound object with Shelf context
Create and bound object with Market context
rebind successful
Waiting for client ...
```

Chương trình NamingRegister của ta in ra thông báo là tên của các context và đối tượng đã được đăng ký với dịch vụ CORBA COS Naming. Sau khi đăng ký xong, ở bước cuối cùng thường các chương trình CORBA không muốn chấm dứt. Bởi nếu chấm dứt thì các đối tượng đã khởi tạo như ObjectA, ObjectB, ObjectC sẽ trở nên không thể gọi được từ trình khách. Các trình cài đặt và đăng ký đối tượng CORBA thường tạo một vòng lặp vô tận để chờ yêu cầu từ phía máy khách như sau

```
System.out.println("Waiting for client ...");
 // Tạo đối tượng Java
java.lang.Object obj = new java.lang.Object();
 //Vòng lặp vô tận, chờ nhận yêu cầu từ máy khách
synchronized (obj){
     obj.wait();
}
```

Nếu bạn loại bỏ đoạn mã này, quá trình đăng ký vẫn diễn ra tốt đẹp. tnameserv vẫn ghi nhớ cấu trúc tên của đối tượng mà bạn tạo ra. Tuy nhiên nếu có một trình khách nào đó yêu cầu dịch vụ tnameserv trả về tham chiếu của đối tượng Book chẳng hạn, tnameserv vẫn tìm thấy tham chiếu của đối tượng. Mặc dù vậy lúc này tham chiếu của đối tượng sẽ không còn hiệu lực nữa bởi trình đăng ký NamingRegiter đã hủy bỏ đối tượng ngay sau khi chương trình chấm dứt. Đó là lý do bạn cần phải xây dựng trạng thái chờ cho trình chủ bằng cách gọi obj.wait() tạo ra vòng lặp vô tận.

Hãy để chương trình NamingRegister chạy ở trạng thái chờ này. Chúng ta sẽ viết một chương trình khách khác duyệt toàn bộ

cấu trúc tên của dịch vụ COS Naming. Chương trình của chúng ta mang tên NamingBrowse.java.

5.2.2. Xây dựng trình duyệt các thành phần tên của COS Naming

Tương tự phép duyệt tập tin và thư mục trên đĩa, chúng ta sử dụng phép đệ quy để thám hiểm toàn bộ nội dung của tnameserv (hay dịch vụ COS Naming của CORBA). Trước hết do cần truy xuất đối tượng Generic, chúng ta sử dụng idlj để biên dịch tập tin Generic.idl ra các lớp hỗ trợ cần thiết cho phía máy khách như sau:

```
idlj -fclient Generic.idl
```

bạn sẽ thu được lớp Generic và GenericHelper viết bằng Java. Ta sử dụng hai lớp này trong quá trình duyệt và tìm lại đối tượng ObjectA, ObjectB, ObjectC (đã đăng ký với tnameserv ở ví dụ 3-15) như sau:

Ví dụ 3-16 NamingBrowse.java

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
public class NamingBrowse
{
  public static void getAllTree(NamingContext nc)
                                        throws Exception{
    //Lấy về danh sách các tên có trong context
    BindingListHolder bl = new BindingListHolder();
    BindingIteratorHolder blIt= new
                          BindingIteratorHolder();
    nc.list(1000, bl, blIt);
    Binding bindings[] = bl.value;
    if (bindings.length == 0) return;
    //Duyệt danh sách tên lấy được
    for (int i=0; i < bindings.length; i++) {
```

```
        // Lấy về tham chiếu của đối tượng dựa vào đường dẫn
      org.omg.CORBA.Object obj =
              nc.resolve(bindings[i].binding_name);
        // Kiểm tra xem có phải là context hay không
      if (bindings[i].binding_type ==
                              BindingType.ncontext) {
        System.out.println( "Context: "
              + bindings[i].binding_name[lastIx].id);
         //Đệ quy tìm sâu xuống các context bên dưới nếu có
        getAllTree((NamingContext)obj);
      } else {
          //Lấy về tên của đối tượng trong đường dẫn
        int lastIx = bindings[i].binding_name.length-1;
        System.out.println("Object: "
              + bindings[i].binding_name[lastIx].id);
          //Lấy về đối tượng (mà tên ràng buộc)
        Generic generic=GenericHelper.narrow(obj);
        System.out.println(generic.getName());
      }
    }
  }
  public static void main(String args[])
                              throws Exception{
    System.out.println("CORBA Browse Application");
     //Khởi tạo trình môi giới ORB
    ORB orb=ORB.init(args,null);
    //Lấy về context gốc của dịch vụ COS Naming
    NamingContext nc = NamingContextHelper.narrow(
       orb.resolve_initial_references("NameService"));
     //Duyệt toàn bộ nội dung cây, bắt đầu từ context gốc
    getAllTree(nc);
    System.out.println("End browse");
```

```
  }
}
```

Phương thức getAllTree() chịu trách nhiệm duyệt nội dung của dịch vụ COS Naming bắt đầu từ một nhánh context nào đó. Sau khi gọi orb.resolve_initital_references() để lấy về context gốc mang tên "NamingService", bạn truyền cho getAllTree() đối tượng context gốc này, nó sẽ thực hiện phép duyệt và in toàn bộ nội dung tên của các đối tượng ra màn hình. Quá trình cài đặt phương thức getAllTree() được thực hiện như sau:

Bạn cần tạo sẵn một danh sách nắm giữ các đối tượng do context trả về. Lớp đối tượng BindingListHolder dùng để nhận về 1000 tên đầu tiên từ context nc. Lớp BindingIteratorHolder được dùng để chứa những phần tử còn lại trong context.

```
BindingListHolder bl = new BindingListHolder();
BindingIteratorHolder blIt= new
                        BindingIteratorHolder();
nc.list(1000, bl, blIt);
```

Sau lời gọi nc.list() bạn lấy về danh sách các đường dẫn chứa trong mảng có kiểu đối tượng Binding:

```
Binding bindings[] = bl.value;
```

Tiếp đến là duyệt toàn bộ mảng đường dẫn Binding để truy tìm đối tượng.

```
for (int i=0; i < bindings.length; i++) {
   ...
}
```

Với mỗi đường dẫn, chúng ta lấy về đối tượng được ràng buộc với nó.

```
org.omg.CORBA.Object obj =
            nc.resolve(bindings[i].binding_name);
```

và kiểm tra xem đang gặp phải thành phần context hay NamingComponent theo cách:

```
if (bindings[i].binding_type ==
                        BindingType.ncontext) {
```

(điều này tương tự việc kiểm tra một đường dẫn hiện là tên của tập tin hay thư mục vậy). Nếu đường dẫn của đối tượng hiện là một context chúng ta gọi đệ quy getAllTree() để tìm sâu xuống dưới.

```
getAllTree((NamingContext)obj);
```

Nếu không phải là context thì đối tượng lấy được chính là một trong những đối tượng kiểu Generic. Chúng ta in ra tên và gọi getName() của đối tượng Generic tìm thấy được:

```
int lastIx = bindings[i].binding_name.length-1;
System.out.println("Object: "
      + bindings[i].binding_name[lastIx].id);
Generic generic=GenericHelper.narrow(obj);
System.out.println(generic.getName());
```

ở đây `binding[i].binding_name` là một mảng đường dẫn kiểu NamingComponent. Do chỉ muốn lấy về tên phần tử cuối cho nên ta sử dụng chỉ số lastIx.

Biên dịch chương trình:

```
C:\Network\corba\Naming>javac *.java
```

Chạy chương trình:

```
C:\Network\corba\Naming>java NamingBrowse
```

Kết quả kết xuất:

```
C:\Network\corba\Naming>java NamingBrowse
CORBA Application
Context: Market
Context: Shelf
Object: Book
This is Object C
Object: Panel
This is Object C
Context: Stock
```

```
Object: Fruit
This is Object B
Object: Toys
This is Object A
End browse
```

6. LƯU ĐỊA CHỈ THAM CHIẾU CỦA ĐỐI TƯỢNG CORBA Ở DẠNG CHUỖI (STRINGIFIED)

Nếu bạn thấy dịch vụ COS Naming của CORBA quá phức tạp thì vẫn còn cách truy xuất khác dễ dàng hơn mà không cần dùng đến COS Naming.

CORBA cho phép bạn chuyển đổi tham chiếu của một đối tượng trên máy chủ thành một chuỗi (String). Bạn lưu chuỗi này thành tập tin và chuyển đến trình khách. Trình khách đọc nội dung của chuỗi sau đó khôi phục chuỗi trở lại thành tham chiếu của đối tượng. Kỹ thuật này trong CORBA gọi là Stringified.

Để ví dụ chúng ta sử dụng lại đối tượng Generic ở mục 5.2.1. Chương trình Setup.java sau đây không dùng đến dịch vụ COS Naming của CORBA mà vẫn có thể hoạt động được.

Ví dụ 3-17 Setup.java

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import java.io.*;
public class Setup
{
  public static void main(String args[])
                                    throws Exception{
    System.out.println("Corba Stringified Demo");
    //Khởi tạo trình môi giới ORB
    ORB orb=ORB.init(args,null);
```

```
    //Tạo đối tượng CORBA
    GenericServant servant= new GenericServant(
                                    "My Generic Object");
    //Kết nối đối tượng CORBA với trình môi giới ORB
    orb.connect(servant);
    //Nhờ trình ORB chuyển tham chiếu của đối tượng thành kiểu
chuỗi
    String genericRef= orb.object_to_string(servant);
    //Tạo tập tin để lưu chuỗi
    FileOutputStream f = new FileOutputStream("t.tmp");
    ObjectOutputStream outstream = new
                                ObjectOutputStream(f);
    outstream.writeObject(genericRef);
    System.out.println("Already save reference
                                        to file");
    System.out.println("Wait for client request ...");
    //Chờ kết nối từ phía máy khách
    java.lang.Object obj = new    java.lang.Object();
    synchronized (obj){
        obj.wait();
    }
  }
}
```

Như bạn thấy, sau khi tạo ra đối tượng CORBA Generic, chương trình gọi phương thức object_to_string() yêu cầu ORB chuyển địa chỉ tham chiếu của đối tượng về dạng chuỗi. Sau đó ta mở tập tin t.tmp để lưu nội dung chuỗi xuống file. Bởi vì kiểu String mang đặc tính Serializable (tuần tự hoá) nên nếu được khôi phục lại từ một máy khác, nó sẽ giữ nguyên nội dung của tham chiếu đối tượng trước khi lưu.

Bạn chuyển tập tin t.tmp xuống máy khách. Trên máy khách sẽ tham chiếu và gọi đối tượng Generic như sau:

Ví dụ 3-18 Client.java

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import java.io.*;
public class Client
{
  public static void main(String args[])
                                    throws Exception{
    //Khởi tạo trình môi giới ORB
    ORB orb=ORB.init(args,null);
    //Mở file đọc nội dung tập tin t.tmp
    FileInputStream f = new FileInputStream("t.tmp");
    ObjectInputStream instream = new
                                ObjectInputStream(f);
    //Đọc đối tượng chuỗi chứa trong tập tin
    String genericRef =(String) instream.readObject();
    //Nhờ ORB chuyển đổi chuỗi trở về tham chiếu của đối tượng
CORBA
    org.omg.CORBA.Object o =
                    orb.string_to_object(genericRef);
    //Chuyển kiểu đối tượng CORBA trở về tham chiếu cụ thể
    Generic generic = GenericHelper.narrow(o);
    //Gọi phương thức của đối tượng CORBA
    System.out.println( generic.getName() );
  }
}
```

Cách tham chiếu bằng kỹ thuật này trong CORBA xem ra tiện lợi và đơn giản hơn tham chiếu bằng dịch vụ COS Naming. Tuy nhiên tập tin t.tmp chỉ có giá trị khi chương trình Setup chưa chấm dứt. Nghĩa là vòng lặp chờ

```
synchronized (obj){
      obj.wait();
```

}

vẫn chưa thoát ra. Nếu bạn chấm dứt chương trình Setup và chạy lại thì nội dung của t.tmp sẽ thay đổi và bạn phải gửi lại tập tin này về máy khách để cập nhật. Đối với dịch vụ COS Naming của CORBA cũng vậy, một khi đã ràng buộc tên cho đối tượng bạn phải duy trì sự hoạt động liên tục của cả hai phía. Nếu trình đăng ký đối tượng (hoặc trình quản lý tên tnameserv.exe) chấm dứt thì tham chiếu của máy khách từ xa đến đối tượng cũng sẽ trở nên mất hiệu lực.

Kỹ thuật lưu tham chiếu của đối tượng thành kiểu chuỗi (stringnified) là một trong những cách khá tiện lợi mà CORBA cung cấp để các lập trình viên có thể tìm ra và gọi được đối tượng mọi lúc mọi nơi không cần phải nhờ đến các dịch vụ phức tạp khác như COS Naming. Trong CORBA kỹ thuật này còn được gọi là IOR (Interoperable Object References)

7. THAM CHIẾU NGƯỢC (CALLBACK) CỦA CÁC ĐỐI TƯỢNG CORBA

Khi nghiên cứu về các đối tượng RMI (ở chương 2 mục 7-4) bạn đã biết đến tính năng tham chiếu ngược (callback), đó là lời triệu gọi của trình chủ ngược về các đối tượng cài đặt trên máy khách. Đối tượng CORBA cũng cho phép bạn thực hiện kỹ thuật này. Chúng ta cài đặt lại đối tượng RMI AtServer và AtClient trong chương trước nhưng được viết trên nền CORBA.

AtServer là đối tượng CORBA cài đặt trên máy chủ. AtClient là đối tượng CORBA chạy trên máy khách. Chương trình khách triệu gọi AtServer trên máy chủ và gửi tham chiếu của AtClient đến trình chủ. AtServer sử dụng tham chiếu của AtClient để gọi ngược về trình khách.

Bước 1: Đặc tả đối tượng AtClient và AtServer bằng ngôn ngữ IDL được khai báo như sau

Ví dụ 3-19 ClientServer.idl

```
interface AtClient{
  void callClientMethod(in string message);
};
interface AtServer{
  void registerClient(in AtClient c);
  void callServerMethod(in string message);
};
```

Tập tin ClientServer.idl đặc tả cả hai interface cho AtClient và AtServer. Bạn dùng trình biên dịch idlj để dịch ClientServer.idl thành các tập tin java phục vụ cho cả hai phía khách, chủ như sau:

```
idlj -fserver ClientServer.idl
idlj -fclient ClientServer.idl
```

Bước 2: Cài đặt AtClient và AtServer. Giao tiếp interface AtClient được cài đặt thành đối tượng CORBA AtClientServant còn giao tiếp interface AtServer được cài đặt thành đối tượng CORBA AtServerServant như sau

Ví dụ 3-20 AtClientServant.java

```
public class AtClientServant extends _AtClientImplBase
{
  public void callClientMethod (String message){
     System.out.println(message);
  }
}
```

Ví dụ 3-21 AtServerServant.java

```
public class AtServerServant extends _AtServerImplBase
{
 AtClient client;
 public void registerClient (AtClient c){
    client=c;
 }
 public void callServerMethod (String message){
```

```
    System.out.println(message);
    for (int i=1; i<10; i++){
      String msg= "Server response "+
                                Math.random()*1000;
        //Triệu gọi phương thức của đối tượng chạy trên máy khách
      client.callClientMethod(msg);
    }
  }
}
```

Bước 3: Xây dựng trình chủ dùng cài đặt và đăng ký đối tượng AtServer trên máy chủ

Ví dụ 3-22 Setup.java

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
public class Setup
{
  public static void main(String args[])
                                  throws Exception{
    System.out.println("CORBA Callback demo");
     //Khởi động trình môi giới
    ORB orb=ORB.init(args,null);
     //Tạo đối tượng AtServer
    AtServer servant= new AtServerServant();
    //Kết nối đối tượng AtServer với trình môi giới
    orb.connect(servant);
     /* Đăng ký tên đối tượng với dịch vụ COS Naming */
    org.omg.CORBA.Object nameService=
        orb.resolve_initial_references("NameService");
    NamingContext nsContext =
        NamingContextHelper.narrow( nameService);
    NameComponent nc=new
```

```
        NameComponent("AtServerObject","");
    NameComponent path[]={nc};
    nsContext.rebind(path,servant);
    //Chờ yêu cầu gửi đến từ máy khách
    java.lang.Object obj = new    java.lang.Object();
    synchronized (obj){
        obj.wait();
    }
  }
}
```

Bước 4: Cài đặt trình khách, triệu gọi đối tượng AtServer trên máy chủ

Ví dụ 3-23 Client.java

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
public class Client
{
  public static void main(String args[])
                                    throws Exception{
    System.out.println("CORBA Client");
    //Khởi động trình môi giới
    ORB orb=ORB.init(args,null);
    //Tạo đối tượng AtClient
    AtClientServant clientservant=
                        new AtClientServant();
    //Kết nối AtClient với trình môi giới
    orb.connect(clientservant);
    /* Yêu cầu dịch vụ COS Naming truy tìm tham chiếu của
AtServer thông qua tên gọi "AtServerObject" */
    org.omg.CORBA.Object nameService=
        orb.resolve_initial_references("NameService");
```

```
    NamingContext nsContext =
        NamingContextHelper.narrow( nameService);
    NameComponent nc=new
        NameComponent("AtServerObject","");
    NameComponent path[]={nc};
    AtServer serverservant=
        AtServerHelper.narrow(nsContext.resolve(path));
    /* Bắt đầu quá trình tương tác với đối tượng AtServer trên máy
chủ*/
    //Đăng ký (hay gửi tham chiếu) của đối tượng AtClient với đối
tượng chủ
    •serverservant.registerClient (clientservant);
    //Triệu gọi đối tượng chủ
    serverservant.callServerMethod ("Client request");
  }
}
```

Bước 5: Biên dịch và chạy chương trình. Bạn tạo thư mục C:\Network\corba\callback. Lưu tất cả mã nguồn vào thư mục này. Chuyển vào thư mục biên dịch và chạy chương trình như sau:

Biên dịch:

```
C:\Network\corba\callback>javac *.java
```

Khởi động dịch vụ tnameserv (ở một cửa sổ DOS-Prompt khác)

```
C:\Network\corba\callback>tnameserv
```

Chạy trình chủ (trên cửa sổ DOS-Prompt khác):

```
C:\Network\corba\callback>java Setup
```

Chạy trình khách:

```
C:\Network\corba\callback>java Client
```

Kết quả kết xuất trên máy khách:

```
CORBA Client
Server response 121.31082068066834
Server response 938.758736612587
```

```
Server response 626.1170145933081
Server response 129.04741023511335
Server response 761.8327395366998
Server response 979.4335449048668
Server response 757.4679933994281
Server response 682.0261146363886
Server response 996.1526944549872
```

Kết quả kết xuất trên máy chủ:

```
CORBA Callback demo
Client request
```

8. GỌI CÁC PHƯƠNG THỨC ĐỘNG (DYNAMIC METHOD CALL) CỦA ĐỐI TƯỢNG CORBA

8.1. Sử dụng kỹ thuật phản chiếu (reflect) của Java

Trong Java, bạn có thể dùng kỹ thuật phản chiếu (reflect) để truy tìm các phương thức có trong đối tượng CORBA. Cơ chế này giúp bạn có thể gọi được phương thức của một đối tượng CORBA bất kỳ mà không cần dùng đến lớp giao tiếp interface do trình chủ cung cấp. Điều này tương tự cơ chế gọi phương thức động của những đối tượng RMI mà ta đã có dịp tìm hiểu qua ở chương 2 (mục 10). Dưới đây là một chương trình ví dụ.

Đối tượng Friend của ta là một đối tượng CORBA cung cấp phương thức greeting() để trình khách triệu gọi. Trình khách không dùng trực tiếp lớp Friend.class do idlj sinh ra để triệu gọi phương thức greeting(). Thay vào đó trình khách sử dụng kỹ thuật reflect của Java để triệu gọi greeting(). Bạn thực hiện chương trình qua các bước sau:

Bước 1: Đặc tả đối tượng Friend bằng ngôn ngữ IDL

Ví dụ 3-24 Friend.idl

```
interface Friend{
```

```
  string greeting(in string name);
};
```

Tiếp đến bạn dùng trình dịch idlj để chuyển đổi đặc tả IDL thành các tập tin cần dùng cho cả hai phía khách chủ.

```
idl -fall Friend.idl
```

Lưu ý: Thay cho tham số dòng lệnh `-fserver` và `-fclient` như chúng ta đã làm trước đây, bạn có thể sử dụng `-fall` để yêu cầu trình idlj sinh ra các tập tin trung gian phục vụ cho cả hai phía khách chủ.

Bước 2: Phần cài đặt cho đối tượng CORBA Friend trong Java được thực hiện thông qua lớp FriendServant như sau

Ví dụ 3-25 FriendServant.java

```
public class FriendServant extends _FriendImplBase
{
  public String greeting (String name){
     return "Hi "+name +" from CORBA !";
  }
}
```

Bước 3: Thiết kế chương trình đăng ký đối tượng Friend với dịch vụ *COS Naming*.

Ví dụ 3-26 Setup.java

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import java.util.Properties;
public class Setup
{
  public static void main(String args[])
                                  throws Exception{
    System.out.println("CORBA Friend Object");
    ORB orb=ORB.init(args,null);
```

```
    Friend servant= new FriendServant();
    orb.connect(servant);
    org.omg.CORBA.Object nameService=
        orb.resolve_initial_references("NameService");
    NamingContext nsContext =
        NamingContextHelper.narrow( nameService);
    NameComponent nc=new
       NameComponent("FriendObject","");
    NameComponent path[]={nc};
    nsContext.rebind(path,servant);
    //Chờ nhận yêu cầu từ máy khách
    java.lang.Object obj = new java.lang.Object();
    synchronized (obj){
         obj.wait();
    }
  }
}
```

Như bạn thấy việc cài đặt và đăng ký đối tượng CORBA không có gì đặc biệt. Nó hoàn toàn tương tự như những ví dụ ta đã làm trước đây. Điểm khác biệt là trình khách mà ta sẽ viết sau đây không dùng lớp Friend.class để gọi đối tượng chủ. Trình khách sử dụng kỹ thuật reflect.

Bước 4: Xây dựng trình khách

Ví dụ 3-27 Client.java

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import java.lang.reflect.*;
public class Client
{
  public static void main(String args[])
                                    throws Exception{
    ORB orb=ORB.init(args,null);
```

```
    org.omg.CORBA.Object nameService=
        orb.resolve_initial_references("NameService");
    NamingContext nsContext =
        NamingContextHelper.narrow( nameService);
    NameComponent nc=new
        NameComponent("FriendObject","");
    NameComponent path[]={nc};
    //Chuyển tham chiếu của đối tượng CORBA thành đối tượng
Java
    java.lang.Object o=
            (java.lang.Object)nsContext.resolve(path);
    //Truy tìm xuất xứ lớp của đối tượng
    Class c = o.getClass();
    //Định kiểu tham số của phương thức cần gọi
    Class[] parameterTypes = new Class[] {String.class};
    //Truy tìm phương thức cần gọi
    Method theMethod;
    theMethod = c.getMethod("greeting", parameterTypes);
    //Định đối số truyền vào phương thức cần gọi
    java.lang.Object[] arguments = new
                        java.lang.Object[] {"Jerry"};
    //Gọi phương thức và nhận kết quả trả về
    String result = (String) theMethod.invoke(o,
                                        arguments);
    System.out.println(result);
  }
}
```

Bước 5: Biên dịch và chạy chương trình. Bạn lưu tất cả mã nguồn vào thư mục C:\Network\corba\dynamic. Biên dịch và chạy chương trình như sau:

Biên dịch mã nguồn:

```
C:\Network\corba\dynamic>javac *.java
```

Khởi động dịch vụ tnameserv trên cửa sổ DOS-Prompt khác:

```
C:\Network\corba\dynamic>javac *.java
```

Chạy trình chủ đăng ký đối tượng:

```
C:\Network\corba\dynamic>java Setup
```

Chạy trình khách:

```
C:\Network\corba\dynamic>java Client
```

Kết quả kết xuất:

```
Hi Jerry from CORBA !
```

8.2. Sử dụng kỹ thuật gọi động (DII - Dynamic Invocation Interface) của CORBA

Kỹ thuật phản chiếu (reflect) là đặc trưng của riêng ngôn ngữ Java. CORBA được xây dựng với mục đích dùng cho đa ngôn ngữ nên bản thân CORBA không thể chỉ dựa vào khả năng phản chiếu mà Java cung cấp. CORBA cung cấp cho bạn cách gọi động các phương thức của một đối tượng CORBA bất kỳ. CORBA gọi khả năng này là "giao tiếp gọi động" (Dynamic Invocation Interface). Với kỹ thuật này trình khách client có thể yêu cầu trình môi giới ORB duyệt qua các phương thức mà đối tượng CORBA nắm giữ. Truyền tham số và triệu gọi phương thức mà không cần đến lớp tham chiếu tường minh của đối tượng. Ví dụ 3-27 viết bằng kỹ thuật reflect trong Java có thể được viết lại theo phong cách CORBA như sau:

Ví dụ 3-28 ClientCorba.java

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
public class ClientCorba
{
  public static void main(String args[])
                                    throws Exception{
    ORB orb=ORB.init(args,null);
    org.omg.CORBA.Object nameService=
```

```
        orb.resolve_initial_references("NameService");
    NamingContext nsContext =
        NamingContextHelper.narrow( nameService);
    NameComponent nc=new
        NameComponent("FriendObject","");
    NameComponent path[]={nc};
    //Lấy về tham chiếu tổng quát của đối tượng CORBA
    org.omg.CORBA.Object o= nsContext.resolve(path);
    //Yêu cầu CORBA truy tìm và trả về tham chiếu của phương thức
    //greeting có trong đối tượng CORBA.
    //Đối tượng Request sẽ chứa các thông tin về phương thức của
    //đối tượng nếu tìm thấy
    org.omg.CORBA.Request request=
                        o._request("greeting");
    //Tạo đối số để truyền vào hàm greeting
    request.add_in_arg().insert_string("Jerry");
    //Định kiểu trả về của phương thức greeting
    request.set_return_type(
            orb.get_primitive_tc(
                org.omg.CORBA.TCKind.tk_string));
    //Triệu gọi phương thức greeting
    request.invoke();
    //Lấy kết quả trả về sau khi gọi hàm
    String result=
            request.result().value().extract_string()
    System.out.println(result);
  }
}
```

Ở ví dụ trên, sau khi lấy về tham chiếu của đối tượng CORBA bằng lệnh:

```
org.omg.CORBA.Object o= nsContext.resolve(path);
```

Biết rằng trong đối tượng CORBA **o** vừa lấy về có chứa phương thức mang tên "greeting" ta yêu cầu **o** trả về tham chiếu của phương thức này như sau:

```
org.omg.CORBA.Request request=
                        o._request("greeting");
```

Dựa vào đối tượng Request bạn có thể truyền đối số và gọi phương thức "greeting" một cách gián tiếp như sau:

```
//Tạo đối số kiểu chuỗi để truyền vào greeting
request.add_in_arg().insert_string("Jerry");
//Định kiểu trả về của phương thức greeting
request.set_return_type(
        orb.get_primitive_tc(
            org.omg.CORBA.TCKind.tk_string));
//Triệu gọi phương thức greeting
request.invoke();
//Lấy kết quả trả về sau khi gọi hàm
String result=
        request.result().value().extract_string()
```

invoke() dùng để triệu gọi phương thức mà đối tượng Request đang tham chiếu đến. Khi bạn triệu gọi phương thức, trị trả về sẽ được đối tượng Request quản lý. Bạn chỉ định kiểu dữ liệu trả về cho phương thức gọi bằng cách gọi set_return_type() của Request. CORBA cung cấp cho bạn rất nhiều hằng định kiểu trả về như TCKind.tk_string, TCKind.tk_float, TCKind.tk_char, TCKind.tk_long ... ở đây "greeting" trả về kiểu chuỗi nên ta sử dụng TCKind.tk_string để định kiểu. Sau lời gọi invoke() bạn có thể gọi phương thức result() để nhận kết quả trả về.

Biên dịch và chạy trình khách ClientCorba (bạn nhớ chạy trình Setup đăng ký đối tượng chủ CORBA Friend ở ví dụ 3-26 trước khi chạy trình khách). Kết quả kết xuất hoàn toàn tương tự như chương trình Client ở ví dụ 3-27 dùng kỹ thuật phản chiếu reflect của Java.

9. RMI TRÊN IIOP (RMI-OVER-IIOP)

9.1. Tại sao lại sử dụng RMI trên IIOP?

RMI và CORBA đều hướng đến mô hình lập trình phân tán đối tượng với cùng một khái niệm. Tuy nhiên RMI của Java có những hạn chế so với CORBA. Kỹ thuật RMI chỉ áp dụng cho những đối tượng thuần Java. Trong khi đó CORBA dung hoà được đối tượng của rất nhiều ngôn ngữ lập trình. Mặc dù vậy, nếu bạn đã nghiên cứu qua chương trước và chương này thì có lẽ sẽ thấy rằng áp dụng kỹ thuật CORBA không đơn giản. RMI của Java lập trình đơn giản và dễ hiểu hơn CORBA. Các đối tượng RMI sử dụng giao thức JRMP (Java Remote Method Protocol) để "nói chuyện" với nhau. Trong khi đó các đối tượng CORBA sử dụng giao thức IIOP được sự chấp nhận và chuẩn hóa của hầu hết các nhà phát triển lớn.

Sun Microsystem muốn có một tiếng nói chung giữa đối tượng CORBA và RMI. Vì lý do này Java cung cấp cho bạn cách cài đặt để các đối tượng RMI của Java có thể sử dụng giao thức IIOP giao tiếp với các đối tượng CORBA. Khả năng này được gọi là RMI-trên-IIOP (RMI-Over-IIOP). Một đối tượng RMI của bạn với kỹ thuật cài đặt RMI-Over-IIOP sẽ vừa có thể giao tiếp được với các trình khách RMI vừa có thể giao tiếp được với các đối tượng CORBA.

Hình 3-5 là các khả năng giao tiếp bằng giao thức RMI-Over-IIOP so với các giao thức JRMP và IIOP giữa các đối tượng.

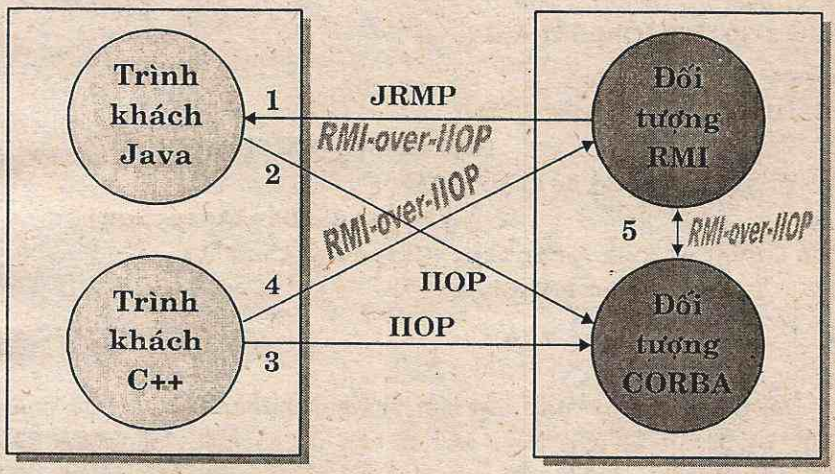
1. Trình khách của bạn viết bằng Java. Bạn có thể giao tiếp được với đối tượng chủ RMI theo giao thức JRMI (thuần Java), hoặc RMI-Over-IIOP.

2. Trình khách của bạn viết bằng Java.Bạn có thể giao tiếp được với đối tượng CORBA theo giao thức IIOP.

3. Trình khách của bạn viết bằng C++. Bạn có thể dùng giao thức IIOP để triệu gọi đối tượng CORBA.

4. Trình khách của bạn viết bằng C++. *Bạn chỉ có thể giao tiếp được với đối tượng RMI thông qua giao thức RMI-Over-IIOP mà thôi.*

5. Đối tượng RMI và CORBA có thể giao tiếp được với nhau dựa trên RMI-over-IIOP.



Hình 3-5 Giao tiếp RMI-Over-IIOP giữa đối tượng trình chủ và trình khách

9.2. Xây dựng đối tượng RMI giao tiếp bằng IIOP

Bạn sẽ thấy khả năng RMI-Over-IIOP bằng ví dụ cụ thể sau đây.

Đối tượng RMIObject của ta là một đối tượng thuần RMI trong Java. Đối tượng này được thiết kế theo kỹ thuật RMI mà bạn đã tìm hiểu qua ở chương trước.

Bước 1: Đặc tả giao tiếp interface cho đối tượng RMIObject.

Ví dụ 3-29 RMIObject.java

```
import java.rmi.*;
public interface RMIObject extends Remote{
   public String tellAbout()
                              throws RemoteException;
}
```

Bước 2: Cài đặt đối tượng RMIObject.

Ví dụ 3-30 RMIObjectImpl.java

```
import java.rmi.*;
public class RMIObjectImpl
                         implements RMIObject{
     public String tellAbout()
                              throws RemoteException{
        System.out.println("Client asking");
        return "I am RMI Object";
     }
}
```

Đối tượng của ta cung cấp một phương thức tellAbout() để "xưng danh" với trình khách. Như vậy là bạn đã có một đối tượng RMI thuần nhất hoàn toàn viết bằng ngôn ngữ Java. Bạn biên dịch chương trình như sau:

```
javac *.java
```

Với lớp đối tượng RMIObjectImpl.class thu được ta muốn đối tượng phải có khả năng giao tiếp bằng giao thức IIOP thay cho giao thức JRMP mà Java qui định đối với RMI. Điều này có thể thực hiện dễ dàng nhờ trình dịch rmic.exe với thông số –iiop như sau:

```
rmic.exe -iiop RMIObjectImpl
```

Kết quả bạn sẽ thu được các tập tin trung gian là _Stub và _Tie đây là những lớp chịu trách nhiệm chuyển đổi lời gọi theo giao thức JRMP thành IIOP và ngược lại.

Trình cài đặt cho đối tượng RMI "giả" đối tượng CORBA được viết như sau:

Ví dụ 3-31 Setup.java

```
import javax.naming.*;
import javax.rmi.*;
public class Setup{
  public static void main(String args[])
                                    throws Exception{
      //Tạo thể hiện của đối tượng Java
    RMIObject obj=new RMIObjectImpl();
      //Yêu cầu máy ảo Java nhận dạng obj là một đối tượng
CORBA
    PortableRemoteObject.exportObject(obj);
    System.out.println("Binding object ...");
      //Lấy về context gốc của dịch vụ đăng ký tên
    Context ctx = new InitialContext();
      //Ràng buộc tên đối tượng với dịch vụ quản lý tên
      //Sau lời gọi bind() chương trình rơi vào trạng thái chờ các yêu
      // cầu cầu gửi đến từ máy khách
    ctx.bind("MyRMIasCORBA",obj);
    System.out.println("Waiting for
                                client request...");
  }
}
```

Khi viết trình cài đặt, có một số điểm bạn cần thay đổi khi chuyển đối tượng RMI thành đối tượng có khả năng sử dụng giao thức IIOP. Thứ nhất chúng ta thay thư viện:

```
import java.naming.*;
import java.rmi.*;
```

thành thư viện:

```
import javax.naming.*;
import javax.rmi.*;
```

Thứ hai, thay vì gọi `UnicastRemoteObject.exportObject()` để thông báo cho máy ảo Java sự hiện diện của đối tượng RMI chúng ta gọi

```
PortableRemoteObject.exportObject(obj);
```

Lệnh trên sẽ thông báo cho máy ảo Java biết rằng đối tượng obj của ta có khả năng giao tiếp từ xa bằng giao thức IIOP của CORBA thay cho JRMP trong RMI.

Các đối tượng CORBA cần đăng ký với dịch vụ quản lý tên tnameserv (COS Naming) trước khi được truy xuất bởi trình khách. Trong CORBA ta vẫn quen gọi lệnh đăng ký sau:

```
org.omg.CORBA.Object nameService=
    orb.resolve_initial_references("NameService");
NamingContext nsContext =
    NamingContextHelper.narrow( nameService);
```

thay cho đoạn lệnh truy tìm context gốc của dịch vụ COS Naming trên, thư viện javax.naming cung cấp cho bạn đối tượng Context tổng quát và gọn hơn để lấy về context gốc như sau:

```
Context ctx = new InitialContext();
```

Với đối tượng context gốc có được chúng ta ràng buộc RMIObject với dịch vụ tên COS Naming như sau:

```
ctx.bind("MyRMIasCORBA",obj);
```

Để chỉ định context gốc lấy về là context của dịch vụ COS Naming bạn chạy trình Setup từ dòng lệnh như sau:

```
java
    -Djava.naming.factory.initial=
            com.sun.jndi.cosnaming.CNCtxFactory
-Djava.naming.provider.url=iiop://localhost:900
```

(bạn lưu ý: dòng lệnh trên được viết trên một dòng liền nhau) với tham số dòng lệnh `-Djava.naming.factory.initial` `java` sẽ sử dụng dịch vụ lớp khởi tạo `com.sun.jndi.cosnaming.CNCtxFactory` làm dịch vụ đăng ký tên (bằng cách này các nhà cung cấp khác nhau có thể chỉ định lớp khởi tạo khác nhau). Tham số

`java.naming.provider.url` chỉ định vị trí mà dịch vụ quản lý tên đang chạy (Trước khi chạy trình Setup bạn nhớ khởi động tnameserv ở một cửa sổ DOS-Prompt khác. Cổng mặc định của tnameserv là 900).

Như vậy là trình Setup đã cài đặt thành công đối tượng RMIObject (đối tượng RMI) có khả năng giao tiếp với trình khách bằng giao thức IIOP của CORBA.

Bước 4: Viết trình khách truy xuất đối tượng RMI theo giao thức IIOP.

Trình khách của ta được viết theo mô hình CORBA thay cho RMI. Vì vậy đối tượng chủ phải có khả năng cung cấp cho trình khách tập tin .idl để trình khách tạo ra các lớp trung gian. Như bạn thấy RMIObject của ta ngay từ đầu đã được thiết kế bằng ngôn ngữ Java. Vì vậy không có tập tin .idl để cung cấp cho trình khách như trong trường hợp xây dựng đối tượng CORBA. Bạn không phải lo lắng, trình rmic.exe sẽ giúp bạn chuyển interface của java sang đặc tả .idl của CORBA. Bạn gọi chương trình rmic.exe với tham số dòng lệnh -idl yêu cầu rmic diễn dịch lớp RMIObject từ Java sang IDL như sau:

```
rmic.exe -idl RMIObject
```

> *Lưu ý: ở các phiên bản cũ của Java hoặc công cụ của những nhà phát triển thứ ba (thirt-party) như Inprise, Oracle chẳng hạn bạn có thể sử dụng trình java2idl.exe thay cho lệnh rmic -idl.*

Kết quả bạn sẽ thu được tập tin RMIObject.idl như trình bày ở ví dụ 3-32 sau:

Ví dụ 3-32 RMIObject.idl

```
/**
 * RMIObject.idl
 * Generated by rmic -idl. Do not edit
 * Friday, July 2, 1999 2:29:00 AM PDT
```

```
*/
#include "orb.idl"
#ifndef __RMIObject__
#define __RMIObject__
  interface RMIObject {
    ::CORBA::WStringValue tellAbout( );
  };
#pragma ID RMIObject "RMI:RMIObject:0000000000000000"
#endif
```

Trình biên dịch sinh thêm một số mã lệnh và thư viện nhúng vào tập tin IDL. Tuy nhiên bạn không cần phải quan tâm chi tiết đến chúng. Cơ bản bạn nhận được một đặc tả của RMIObject với phương thức tellAbout() trả về kiểu chuỗi viết bằng ngôn ngữ IDL của CORBA. Nhiệm vụ của bạn là chuyển tập tin này về trình khách. Trình khách sử dụng tập tin RMIObject.idl này để cài đặt cách triệu gọi đối tượng trên máy chủ. Và đúng như tinh thần CORBA, trình khách có thể được viết bằng hầu hết các ngôn ngữ mà CORBA hỗ trợ (C++, Java ...). Ở đây ta sử dụng chính Java để viết trình khách. Bạn gọi trình idlj để sinh ra các tập tin trung gian cho trình khách theo cách sau:

```
idlj -fclient RMIObject.idl
```

Cài đặt cho trình khách được thực hiện ở ví dụ 3-33.

Ví dụ 3-33 Client.java

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
public class Client
{
  public static void main(String args[])
                                    throws Exception{
    //Khởi động trình môi giới trung gian
    ORB orb=ORB.init(args,null);
    //Tìm tham chiếu của context gốc
```

```
    org.omg.CORBA.Object nameService=
        orb.resolve_initial_references("NameService");
    //Lấy về tham chiếu của đối tượng
    NamingContext nsContext =
        NamingContextHelper.narrow( nameService);
    NameComponent nc=new
        NameComponent("MyRMIasCORBA","");
    NameComponent path[]={nc};
    RMIObject servant=
      RMIObjectHelper.narrow(nsContext.resolve(path));
     //Triệu gọi đối tượng
    System.out.println( servant.tellAbout() );
  }
}
```

Biên dịch và chạy chương trình khách như sau:

Biên dịch:

```
javac *.java
```

Chạy chương trình:

```
java Client
```

Kết quả kết xuất trên trình khách:

```
I am RMI Object
```

Chương trình Client.java trình bày ở ví dụ trên hoàn toàn là một trình khách CORBA bình thường. Bạn có thể sử dụng đối tượng Context để viết trình khách theo phong cách RMI như ví dụ 3-34 sau:

Ví dụ 3-34 ClientRMI.java

```
import javax.rmi.*;
import javax.naming.*;
public class ClientRMI
{
 public static void main(String args[])
                                    throws Exception{
```

```
    Context ctx = new InitialContext();
    RMIObject servant = (RMIObject)
        PortableRemoteObject.narrow(
            ctx.lookup("MyRMIasCORBA"),RMIObject.class);
    System.out.println( servant.tellAbout() );
  }
}
```

Tương tự trình chủ, để chạy trình khách viết theo phong cách RMI truy tìm được dịch vụ COS Naming bạn phải gọi máy ảo Java với tham số dòng lệnh như sau:

```
  java
    -Djava.naming.factory.initial=
            com.sun.jndi.cosnaming.CNCtxFactory
-Djava.naming.provider.url=iiop://localhost:900
```

Kết quả kết xuất từ trình khách vẫn sẽ là lời "tự bạch" của đối tượng chủ trả về từ phương thức tellAbout(): `I am RMI Object`.

9.3. Khởi tạo các thông số ngữ cảnh

Khi xây dựng chương trình khách và chủ dùng đến đối tượng Context như ví dụ 3-31 và 3-33, chúng ta đã phải truyền đối số từ dòng lệnh để khởi tạo dịch vụ `java.naming.factory.initial` khá dài và phức tạp. Thay vì truyền các thiết lập cho đối tượng Context từ dòng lệnh, bạn có thể thực hiện ngay thiết lập này trong mã lệnh như sau:

Ví dụ 3-35 ClientRMIConfig.java

```
import java.util.*;
import javax.rmi.*;
import javax.naming.*;
public class ClientRMIConfig
{
  public static void main(String args[])
                                    throws Exception{
    /* Phần thiết lập khởi tạo cho Context ngay trong mã lệnh */
```

```
    Hashtable env = new Hashtable();
    env.put("java.naming.corba.applet", this);
    env.put("java.naming.factory.initial",
            "com.sun.jndi.cosnaming.CNCtxFactory");
    env.put("java.naming.provider.url",
                    "iiop://localhost:900");
    Context ctx = new InitialContext(env);
    RMIObject servant = (RMIObject)
        PortableRemoteObject.narrow(
          ctx.lookup("MyRMIasCORBA"),RMIObject.class);
    System.out.println( servant.tellAbout() );
  }
}
```

Như bạn thấy, chúng ta sử dụng bảng env kiểu Hashtable để lưu các giá trị thiết lập cho Context. Thay cho lời gọi khởi tạo InitialContext() với trị rỗng, đối tượng Context được gọi khởi động với thông số thiết lập cụ thể từ bảng env như sau:

```
Context ctx = new InitialContext(env);
```

Bằng cách này trình khách có thể chỉ cần gọi đơn giản từ dòng lệnh:

```
java ClientRMIConfig
```

10. CORBA VÀ TRÌNH MÔI GIỚI ORB VISIBROKER

Tổ chức CORBA chỉ đưa ra các đặc tả tổng quát để xây dựng mô hình đối tượng phân tán triệu gọi lẫn nhau. Thực tế để có thể đưa vào ứng dụng, bạn phải sử dụng các trình môi giới trung gian ORB của một nhà cung cấp nào đó. Nhờ trình ORB các đối tượng CORBA mới có khả năng trao đổi được với nhau theo giao thức IIOP.

Java phiên bản JDK 1.2 và 1.3 nhúng luôn trình ORB vào gói thư viện để bạn có thể gọi trực tiếp được ORB ngay trong Java mà không cần phải mua thêm một trình ORB của hãng khác. Tuy nhiên,

do tính chất miễn phí nên ORB do chính hãng Sun cài đặt đi kèm với bộ JDK của Java không mang đầy đủ các chức năng cần thiết để phát triển một ứng dụng thật sự (Thường thì những dịch vụ miễn phí kèm theo JDK chỉ có tính năng minh họa).

Phần này sẽ trình bày với bạn tóm tắt các chức năng của trình ORB VisiBroker nổi tiếng do hãng Inprise (Borland) cài đặt.

10.1. Giới thiệu VisiBroker của Inprise

Inprise là hãng phát triển công cụ lập trình rất nổi tiếng. Bạn đã quen với Inprise qua các sản phẩm như Borland C++, Pascal, Delphi, JBuilder. Hơn ai hết chính Inprise luôn tìm ra cách tích hợp các ngôn ngữ lập trình với nhau. CORBA là một giải pháp khá hoàn hảo để Inprise thực hiện mong muốn của mình. VisiBroker là một sản phẩm dành cho những nhà lập trình theo hướng CORBA. VisiBroker chính là trình môi giới trung gian ORB có khả năng giao tiếp được với hầu hết các đối tượng CORBA cài đặt bằng JBuilder, C++, Delphi (Pascal). Trong hầu hết các sản phẩm của Inprise khi cài đặt bạn đều tìm thấy bộ VisiBroker kèm theo. VisiBroker cho đến thời điểm này cũng đã được nhúng vào sản phẩm của hệ Oracle, cùng với sự chấp nhận của các đối tác khác như Sun, IBM, Cisco Systems, Hewlett Packard ...

Khi cài đặt đối tượng CORBA trong VisiBroker, về nguyên tắc không có gì khác với cách cài đặt các đối tượng CORBA mà ta đã nghiên cứu qua ở mục trên. Tuy nhiên, VisiBroker 4.1 của Inprise đưa ra thêm một số tính năng mới của CORBA mà JDK 1.3 vẫn chưa cập nhật kịp theo các đặc tả CORBA (sản phẩm trả tiền bao giờ cũng tốt hơn sản phẩm miễn phí).

Bạn có thể tham khảo những thông tin về VisiBroker trên trang web của Inprise tại địa chỉ: http://www.inprise.com/visibroker

Để tiện cho bạn học tập chúng tôi đã download sản phẩm VisiBroker 4.1 mới nhất (11Mb). Bạn có thể coi trên CD kèm theo sách. Sản phẩm này chỉ cho phép sử dụng 30 ngày nhưng cũng đầy đủ mọi chức năng và tài liệu để bạn nghiên cứu.

10.2. Xây dựng đối tượng chủ CORBA

Chương trình ví dụ sau đây minh họa giao dịch trong ngân hàng thông qua đối tượng CORBA. Đối tượng Account dùng để chứa tài khoản của khách hàng. Đối tượng AccountManager dùng để quản lý và cấp phát đối tượng Account cho khách hàng nào muốn mở tài khoản.

Bước 1: Đặc tả đối tượng CORBA bằng ngôn ngữ IDL

Ví dụ 3-36 Bank.idl

```
module Bank {
 interface Account {
  float balance();
 };
 interface AccountManager {
  Account open(in string name);
 };
};
```

Bạn tạo thư mục C:\Network\CORBA\BankApp. Lưu tập tin Bank.idl vào thư mục này. Biên dịch Bank.idl sang Java như sau:

```
C:\Network\CORBA\BankApp >Idl2java Bank.idl
```

Chương trình idlj2java.exe nằm trong thư mục cài đặt \VisiBroker\Bin. Chương trình này của Inprise thay cho idlj.exe mà bộ JDK cung cấp. Kết quả bạn sẽ thu được các tập tin sau đây trong thư mục con Bank:

Đối tượng Account:

```
Account.java
AccountHelper.java
AccountHolder.java
AccountOperations.java
AccountPOA.java
AccountPOATie.java
```

Đối tượng AccountManager:

```
AccountManager.java
AccountManagerHelper.java
AccountManagerHolder.java
AccountManagerOperations.java
AccountManagerPOA.java
AccountManagerPOATie.java
```

Tổng quát, tương tự như những ví dụ CORBA trước đây, bạn thu được các tập tin:

```
<interfaceName>.java,
<interfaceName>_Helper.java,
<interfaceName>_Holder.java,
<interfaceName>_Operations.java
```

Đặc tả CORBA mới nhất đưa ra giao tiếp khả chuyển (Portable) cho nên thay vì tạo ra lớp `<interfacename>_ImplBase.java` bạn sẽ thu được lớp `<interfacename>_POA.java`. Lớp `<interfacename>_POATie.java` là lớp trung gian đóng vai trò giao tiếp giữa đối tượng và ORB.

Để cài đặt cho đối tượng CORBA trong VisiBroker bạn kế thừa lớp `<interfacename>_POA` như sau:

Ví dụ 3-37 AccountImpl.java

```
/* Cài đặt đối tượng Account */
public class AccountImpl extends Bank.AccountPOA {
 private float _balance;
 public AccountImpl(float balance) {
        _balance = balance;
 }
 public float balance() {
        return _balance;
 }
}
```

Đối tượng Account của ta cung cấp phương thức balance() trả về số cân đối tài khoản của khách hàng. Account được tạo ra bởi phương thức open() của AccountManager như sau:

Ví dụ 3-38 AccountManagerImpl.java

```
import org.omg.PortableServer.*;
import java.util.*;
public class AccountManagerImpl
                    extends Bank.AccountManagerPOA {
 //Bảng chứa các tài khoản cần quản lý
 private Dictionary _accounts = new Hashtable();
 //Đối tượng tạo số ngẫu nhiên
 private Random _random = new Random();
 /* Phương thức này được triệu gọi để mở tài khoản cho khách hàng
mang tên name */
 public Bank.Account open(String name) {
   // Tìm xem tài khoản đã có trong danh sách hay chưa
   Bank.Account account = (Bank.Account)
                                  _accounts.get(name);
   // Nếu chưa thì tạo tài khoản mới
   if(account == null) {
   //Tạo số tiền ban đầu cho tài khoản
    float balance = Math.abs(
                    _random.nextInt()) % 100000;
   // Tạo đối tượng tài khoản mới
    AccountImpl accountServant =
                    new AccountImpl(balance);
    try {
       //Chuyển đối tượng về kiểu Account
      account = Bank.AccountHelper.narrow(
            _default_POA().servant_to_reference(
                                  accountServant));
    } catch (Exception e) {
       e.printStackTrace();
    }
```

```
    // Tài khoản được tạo thành công
    System.out.println("Created " + name
                        + "'s account: " + account);
    // Lưu tài khoản vào danh sách
    _accounts.put(name, account);
  }
  // Trả về tài khoản cho khách hàng.
  return account;
 }
}
```

Chương trình Server dùng đăng ký đối tượng AccountManager với trình ORB của VisiBroker được thực hiện như sau:

Ví dụ 3-39 Server.java

```
import org.omg.PortableServer.*;
public class Server {
 public static void main(String[] args) {
  try {
   //Khởi tạo trình ORB
   org.omg.CORBA.ORB orb =
          org.omg.CORBA.ORB.init(args,null);
   //Lấy về tham chiếu gốc của đối tượng Context POA
   POA rootPOA = POAHelper.narrow(
          orb.resolve_initial_references("RootPOA"));
   //Tạo chính sách qui định cách đối tượng hoạt động
   org.omg.CORBA.Policy[] policies = {
          rootPOA.create_lifespan_policy(
                  LifespanPolicyValue.PERSISTENT)
   };
   // Tạo đối tượng myPOA mới  thêm vào đối tượng POA gốc
   POA myPOA = rootPOA.create_POA( "bank_agent_poa",
                          rootPOA.the_POAManager(),
                                  policies );
```

```
    //Tạo đối tượng phục vụ Servant
    AccountManagerImpl managerServant =
                                    new AccountManagerImpl();
    // Chỉ định tên (id) cho đối tượng
    byte[] managerId = "BankManager".getBytes();
    // Kích hoạt đối tượng chứa trong POA
    myPOA.activate_object_with_id(managerId,
                                          managerServant);
    // Kích hoạt bộ quản lý POA
    rootPOA.the_POAManager().activate();
    System.out.println(
          myPOA.servant_to_reference(managerServant) +
                                          " is ready.");
    // Chờ yêu cầu triệu gọi từ máy khách
    orb.run();
  }
  catch (Exception e) {
        e.printStackTrace();
  }
 }
}
```

10.3. Xây dựng trình khách

Ví dụ 3-40 Client.java

```
public class Client {
 public static void main(String[] args) {
    // Khởi động trình trung gian ORB.
    org.omg.CORBA.ORB orb =
          org.omg.CORBA.ORB.init(args,null);
   // định tên của đối tượng CORBA cần gọi
  byte[] managerId = "BankManager".getBytes();
   // Truy tìm tham chiếu của đối tượng
```

```
    Bank.AccountManager manager =
    Bank.AccountManagerHelper.bind(orb,
                        "/bank_agent_poa", managerId);
    // Sử dụng đối số args[0] làm tên tài khoản. Nếu args[0] không
    // có tên tài khoản được đặt là Jack
    String name = args.length > 0 ? args[0] : "Jack";
    // Yêu cầu đối tượng AccountManager mở tài khoản
    Bank.Account account = manager.open(name);
    // Lấy về số cân đối của tài khoản
    float balance = account.balance();
    // In kết quả
    System.out.println
     ("The balance in " + name + "'s account is $"
                                        + balance);
  }
}
```

10.4. Biên dịch và triệu gọi đối tượng CORBA trong VisiBroker

Để biên dịch chương trình chúng ta đã xây dựng ở mục 10.1 và 10.2 bạn lưu tất cả mã nguồn vào thư mục C:\Network\corba\BankApp (lưu ý trong thư mục này có thư mục con \Bank do trình idl2java sinh ra để chứa các lớp trung gian). VisiBroker sử dụng trình vbjc.exe để biên dịch và vbj.exe để thực thi chương trình. Thực ra cả hai chương trình vbjc và vbj đều gọi đến javac.exe và java.exe. Tuy nhiên khi gọi vbjc và vbj bạn không cần phải thiết lập đường dẫn CLASSPATH trỏ đến thư viện của VisiBroker. VisiBroker tự động chỉ định thư viện biên dịch cho Java hộ bạn.

Biên dịch mã nguồn:

```
C:\Network\corba\BankApp>vbjc *.java
```

Khởi động trình VisiBroker ORB mang tên *VisiBroker Smart Agent*. Bạn vào thư mục C:\Visibroker\bin\ đánh lệnh sau:

```
C:\Visibroker\bin\osagent.exe
```

Trình Smart Agent của VisiBroker chịu trách nhiệm quản lý tên đăng ký của đối tượng CORBA. Thực hiện kết nối giữa Smart Agent trên máy khách và máy chủ ...

Bạn chạy chương trình đăng ký AccountManager như sau:

```
C:\Network\corba\BankApp>vbj Server
```

Chạy trình khách:

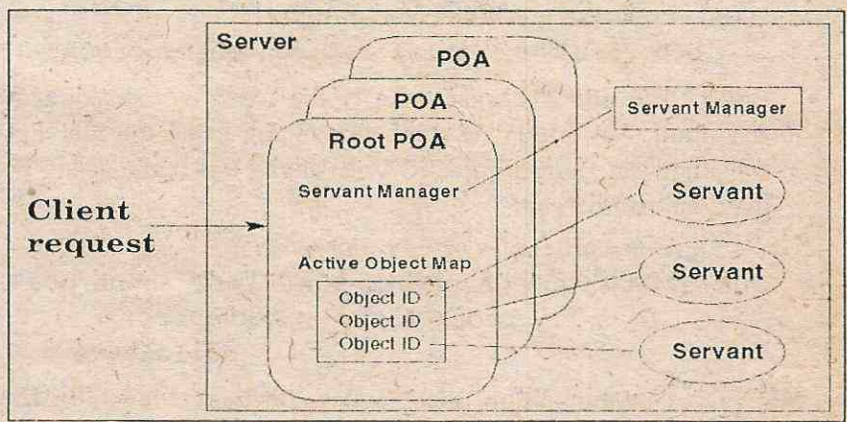
```
C:\Network\corba\BankApp>vbj Client
```

Kết quả kết xuất trên trình khách:

```
The balance in Jack's account is $779.56
```

10.5. Các đặc tả mới của CORBA và Visibroker

CORBA trong đặc tả mới nhất muốn xây dựng mô hình khả chuyển (portable) cho các đối tượng bằng khái niệm POA (Portable Object Adaptor). POA là lớp chịu trách nhiệm quản lý và kích hoạt các đối tượng theo yêu cầu của trình khách. ORB cung cấp cho bạn rất nhiều lớp POA. Tất cả các lớp POA con đều phải được tạo ra từ POA gốc (gọi là Root POA). Mỗi POA chứa một bảng danh sách (Active Object Map) cho biết đối tượng nào đang trong trạng thái phục vụ (đã được kích hoạt). Đối tượng được POA quản lý theo định danh ID chứa trong danh sách kích hoạt (hình 3-6).



Hình 3-6. Cơ chế quản lý đối tượng của CORBA theo đặc tả mới

Khi trình khách yêu cầu triệu gọi một đối tượng từ POA, lớp POA kiểm tra xem định danh ID của đối tượng cần gọi đã có trong bản kích hoạt hay chưa. Nếu có rồi, POA sẽ trả về tham chiếu của đối tượng để trình khách triệu gọi. Nếu chưa có, POA sẽ yêu cầu bộ quản lý đối tượng (Servant Manager) kiến tạo đối tượng (nói chính xác hơn là tạo ra thể hiện instnce của đối tượng bằng lệnh new). Servant Manager sẽ đưa đối tượng vào bảng kích hoạt (Active Object Map) để POA quản lý.

10.5.1. Đăng ký phía máy chủ

Với khái niệm trên, trình tự của quá trình đăng ký đối tượng AccountManager ở ví dụ 3-39 (Server.java) diễn ra như sau:

- Để sử dụng được các lời gọi đến POA bạn cần liên kết với gói PortableServer bằng lệnh

```
import org.omg.PortableServer.*;
```

- Đầu tiên cần lấy về tham chiếu đến POA gốc (Root POA). Điều này tương tự như bạn cần phải lấy về context gốc của dịch vụ đăng ký tên tnameserv trong giao tiếp CORBA cũ.

```
//Lấy về tham chiếu gốc của đối tượng Context POA
POA  rootPOA = POAHelper.narrow(
    orb.resolve_initial_references("RootPOA"));
```

- Bạn có thể nhờ POA gốc quản lý đối tượng AccountManager của mình. Tuy nhiên để minh hoạ chúng ta sẽ tạo một POA con khác của riêng mình. POA con mang tên bank_agent_poa được tạo ra như sau

```
// Tạo đối tượng myPOA mới  thêm vào đối tượng POA gốc
POA myPOA = rootPOA.create_POA( "bank_agent_poa",
                    rootPOA.the_POAManager(),
                                    policies );
```

Mỗi POA cần một chính sách (policy) để hoạt động (như thời hạn cho phép tồn tại của đối tượng, cách kích hoạt đối tượng ...). Chính sách này được tạo ra như sau:

```
        // Tạo chính sách qui định cách đối tượng hoạt động
trong POA
        org.omg.CORBA.Policy[] policies = {
            rootPOA.create_lifespan_policy(
                        LifespanPolicyValue.PERSISTENT)
        };
```

Bạn có thể tham khảo rất nhiều chính sách cho POA trong tài liệu của VisiBroker.

- Đối tượng AccountManager của ta được tạo ra và đưa vào trạng thái kích hoạt trong POA như sau:

```
        //Tạo đối tượng phục vụ Servant
        AccountManagerImpl managerServant =
                        new AccountManagerImpl();
        // Chỉ định tên (id) cho đối tượng
        byte[] managerId = "BankManager".getBytes();
        // Kích hoạt đối tượng chứa trong POA
        myPOA.activate_object_with_id(managerId,
                                managerServant);
```

Khi bạn kích hoạt một đối tượng trong POA đối tượng chưa thật sự hoạt động. Muốn đối tượng có thể triệu gọi được bởi trình khách bạn cần kích hoạt POA chứa đối tượng như sau:

```
        // Kích hoạt bộ quản lý POA
        rootPOA.the_POAManager().activate();
```

- Bước cuối cùng là yêu cầu ORB lắng nghe lời triệu gọi từ phía máy khách:

```
        orb.run();
```

Lệnh `orb.run()` tương tự lệnh `Naming.bind()` hay `obj.wait()` mà ta đã sử dụng trong các ví dụ về CORBA và RMI trước đây.

10.5.2. Gọi đối tượng của máy chủ phía máy khách

Quá trình triệu gọi đối tượng chủ (đã đăng ký với VisiBroker Smart Agent ở mục trên) từ máy khách không khó. Bạn chỉ cần khởi động trình ORB

```
org.omg.CORBA.ORB orb =
    org.omg.CORBA.ORB.init(args,null);
```

tiếp đến là lấy các lớp trợ giúp (do idl2java sinh ra) nhận về tham chiếu của đối tượng chủ như sau

```
// định tên của đối tượng CORBA cần gọi
byte[] managerId = "BankManager".getBytes();
// Truy tìm tham chiếu của đối tượng
Bank.AccountManager manager =
    Bank.AccountManagerHelper.bind(orb,
                "/bank_agent_poa", managerId);
```

lệnh `bind()` ở đây tương đương với `Naming.lookup()` của RMI hay `context.resolve()` của dịch vụ COS Naming đã được ta nghiên cứu ở phần trên.

Tìm hiểu về VisiBroker một cách chi tiết không thuộc phạm vi của giáo trình này. Với những khái niệm CORBA và RMI đã học bạn có thể tìm thấy những ví dụ rất dễ hiểu của VisiBroker do Inprise đưa ra. Chúng nằm trong thư mục cài đặt \VisiBroker\examples.

11. KẾT CHƯƠNG

Chương này đã giới thiệu với bạn một đề tài khá hóc búa đó là kỹ thuật lập trình với ngôn ngữ đặc tả CORBA trong Java. CORBA là một trong những cố gắng của tổ chức quốc tế nhằm tìm kiếm một mô hình phân tán đối tượng ổn định và dễ dùng. Bạn đừng vội nản lòng nếu thấy CORBA quá khô khan, không như thế giới lập trình giao diện đầy màu sắc. Ẩn chứa bên trong CORBA và RMI là một công nghệ làm nền tảng rất vững chắc cho các ứng dụng mạng có ích sau này.

Bắt đầu từ chương sau chúng ta sẽ tìm hiểu các đề tài khác thú vị hơn, đó là lập trình các ứng dụng Web với Java. Chúng ta sẽ trở về với CORBA và RMI ở chương cuối khi nghiên cứu về EJB (Enterprise Java Bean).

Chương 4

JAVA SERVLET VÀ LẬP TRÌNH WEB

Tóm tắt chương 4

Servlet là kỹ thuật lập trình phía máy chủ Java hiện sử dụng khá phổ biến trên thế giới. Chương này sẽ giúp bạn nắm vững một số cơ chế lập trình Web và cách thiết kế servlet đáp ứng yêu cầu phía máy khách. Servlet là một cố gắng của Sun đưa Java vào lĩnh vực Internet.

Các vấn đề chính sẽ được đề cập đến:

- ✓ Giới thiệu HTTP, CGI, Perl, ISAPI, ASP, Servlet, JSP
- ✓ Cài đặt và cấu hình Java Web Server (JWS 2.0)
- ✓ Thiết kế servlet, xây dựng servlet liên hoàn.
- ✓ Tìm hiểu trang SHTML, Session, Cookies.
- ✓ Truy vấn dữ liệu từ servlet bằng JDBC

1. GIỚI THIỆU CÁC KHÁI NIỆM LẬP TRÌNH WEB

Lập trình mạng trên Internet ngày nay không sớm thì muộn bạn cũng sẽ phải tìm hiểu về trình chủ Web Server. Ở chương 1 chúng ta đã thiết kế một chương trình Java Web Server đơn giản có khả năng trả về trang Web tĩnh (.html) cho trình khách (browser) theo giao thức http. Nhiệm vụ của trình chủ Web Server ban đầu chỉ thực hiện chức năng tìm kiếm và trả về tập tin .html do trình khách yêu cầu. Tuy nhiên, theo đà phát triển và mở rộng của mạng Internet, trình chủ đòi hỏi phải thực hiện nhiều chức năng hơn nữa. Web Server phải có khả năng tương tác với trình khách, xử lý dữ liệu do trình khách gửi đến, trả về kết quả động tùy thuộc vào nội dung dữ liệu mà trình khách yêu cầu ...

1.1. Giao thức HTTP

Giao thức HTTP cung cấp cho bạn cách thức trò chuyện giữa trình khách (thường là trình duyệt browser) và trình chủ Web Server khá hiệu quả. Với giao thức HTTP trình khách có thể gửi và yêu cầu trình chủ Web Server xử lý những lệnh sau

GET	*Yêu cầu trình chủ trả về nội dung một tài nguyên (thường là tập tin) theo địa chỉ định vị URL*
POST	*Chuyển dữ liệu từ trình khách lên trình chủ. Trình chủ tiếp nhận dữ liệu, tính toán và trả kết quả về cho trình khách*
HEAD	*Yêu cầu trình chủ trả về thông tin mô tả của một tài nguyên (tập tin).*
PUT	*Đưa một tài nguyên (tập tin) lên trình chủ*
DELETE	*Yêu cầu trình chủ xóa một tài nguyên (tập tin) nào đó*

Thường trong giao tiếp giữa trình khách và trình chủ Web Server chúng ta chỉ dùng GET để lấy kết quả trả về từ trình chủ và POST để đưa dữ liệu từ trình khách lên trình chủ xử lý. Ví dụ, khi bạn gõ địa chỉ http://www.microsoft.com/index.html có nghĩa là bạn yêu cầu trình duyệt (browser) ở máy khách dùng giao thức HTTP gửi lệnh GET lên máy chủ của Microsoft lấy về tập tin index.html. Lệnh POST của giao thức HTTP thường được dùng trong các trang web có chứa thẻ <FORM>. Ví dụ, trang Web sau sẽ cho phép người dùng nhập liệu vào ô văn bản và nhấn nút Submit. Dữ liệu trong ô văn bản do người dùng nhập vào sau đó sẽ được tự động chuyển lên trình chủ Web Server ở địa chỉ www.myserver.com yêu cầu trang web mang tên calculate.jsp xử lý

```
<HTML>
    <FORM
        ACTION=http://www.myserver.com/calculate.jsp
```

```
        METHOD=POST>
      <INPUT type=text>
      <INPUT type=submit>
    </FORM>
</HTML>
```

Chức năng GET và POST sẽ được chúng ta nghiên cứu chi tiết hơn trong các ví dụ ở phần sau.

Giao thức HTTP không lưu vết trạng thái trong giao dịch giữa trình khách và trình chủ. Mỗi lần trình khách gửi yêu cầu và được trình chủ trả lời thì kết nối chấm dứt. Vì lý do này giao thức HTTP còn được gọi là giao thức phi trạng thái (stateless protocol).

1.2. CGI (Common Gateway Interface)

Lệnh GET của giao thức HTTP có thể yêu cầu trình chủ Web Server trả về một tập tin html (hay trang web) hiện hữu trên máy chủ theo địa chỉ URL. Tuy nhiên nếu trình chủ chỉ thực hiện chức năng này thì sẽ rất hạn chế trong việc phục vụ trình khách. Khi nhận được lệnh GET hay POST từ trình khách, trình chủ có thể cho thực thi một chương trình và chương trình này sẽ tự động sản sinh ra nội dung của tập tin trả về cho trình khách. Như vậy lúc này tập tin trả về cho trình khách sẽ không tồn tại trên đĩa cứng của trình chủ, nó được tạo ra từ một chương trình. Chương trình này thường được gọi là CGI. Một chương trình CGI của trình chủ có thể sản sinh ra hàng ngàn trang Web theo yêu cầu của trình khách. Thật ra chương trình CGI đơn thuần chỉ là một chương trình .exe (trên Windows) hoặc chương trình thực thi (trên Unix) bình thường. Bạn có thể xây dựng chương trình CGI từ rất nhiều ngôn ngữ như C, Pascal, Visual Basic ...

Ví dụ khi bạn gửi yêu cầu http://www.myserver.com/cgi-pro.exe, trình chủ Web Server không truy tìm tập tin cgi-app.exe để trả về cho trình khách. Thay vào đó trình chủ sẽ gọi chương trình cgi-pro.exe thực thi. Nhiệm vụ của cgi-pro.exe là sinh ra trang web trả về cho trình khách. Với Pascal bạn có thể viết chương trình cgi-pro.exe như sau

```
begin
  writeln("<HTML>);
  writeln("<B> Today is<B>"+DateToStr(GetDateTime()));
  writeln("</HTML>");
end.
```

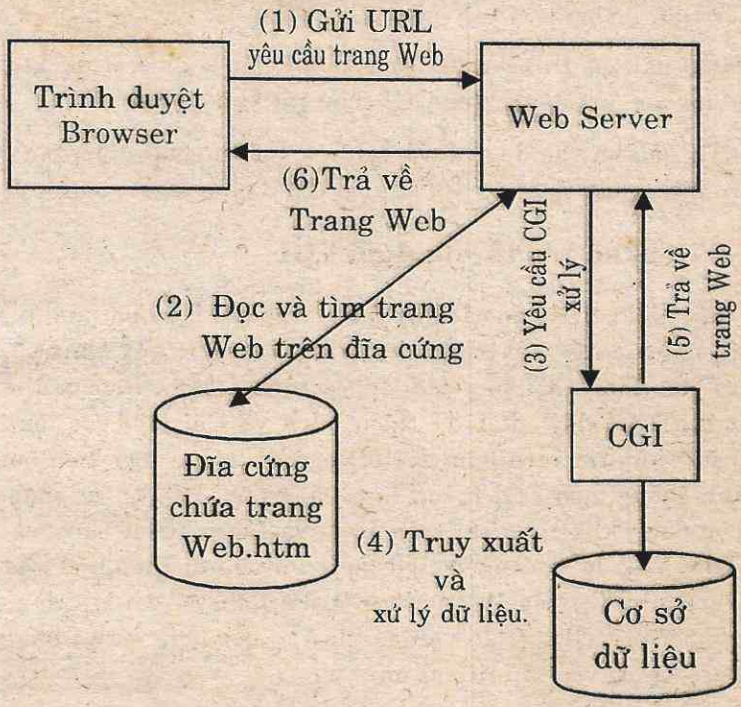
Mỗi lần cgi-pro.exe được gọi chương trình sẽ trả về cho bạn trang Web với nội dung hiển thị ngày giờ khác nhau. Cụ thể trên máy khách bạn sẽ nhận về tập tin có nội dung sau

```
<HTML>
  <B>Today is <B> Wed, 13 2000 15:PM:00
<HTML>
```

Ích lợi của chương trình CGI là tạo được những trang Web tùy biến. Như bạn thấy, bên trong chương trình CGI chúng ta có thể tính toán, kết nối truy vấn cơ sở dữ liệu, lưu các thông tin do trình khách gửi đến ... nội dung dữ liệu do trình CGI trả về cho máy khách do đó sẽ phong phú và đa dạng hơn. Hình 4-1 là cơ chế hoạt động của chương trình CGI trên Web Server so với cách thức truy xuất trang Web tĩnh thông thường

Thông thường Browser sẽ gửi địa chỉ URL của trang Web lên server (1). Web server sẽ tìm nội dung trang Web trên máy chủ (2) sau đó gửi trả trang Web tìm được về cho Browser (6). Với CGI, thay vì thực hiện bước (2), Web server sẽ gọi chương trình .EXE hay CGI (3). Chương trình CGI sẽ tính toán xử lý, truy xuất số liệu từ cơ sở dữ liệu (4), sinh ra trang Web gửi lại Web server (5). Cuối cùng Web Server đưa trang Web về cho Browser trên máy client (6).

Chương trình JavaWebServer ở chương 1 của ta chỉ thực hiện thao tác (2) và (6). JavaWebServer không có khả năng điều khiển CGI. Chúng ta sẽ sử dụng các trình Web Server chuyên nghiệp hơn như IIS của Microsoft hay Java Web Server của Sun cho việc thử nghiệm các ứng dụng làm việc theo cơ chế CGI sau này.



Hình 4-1 Cơ chế làm việc của ứng dụng CGI

Cơ chế làm việc của các chương trình CGI tỏ ra rất hiệu quả trong các ứng dụng truy xuất Web trên Internet. Ngày nay CGI hiện vẫn còn được dùng rất rộng rãi trên các trình Web Server chạy trên máy chủ Unix và Linux. Nhược điểm duy nhất của CGI đó là tiêu tốn tài nguyên và một phần giới hạn về tốc độ thực thi. Mỗi khi có yêu cầu gửi đến từ trình khách, trình chủ Web Server lại phải triệu gọi ứng dụng CGI (chương trình .EXE) theo các bước:

- Nạp chương trình vào bộ nhớ.
- Thực thi chương trình.
- Trả kết quả về cho trình khách.

- Giải phóng trình CGI khỏi bộ nhớ.

Việc lưu vết trạng thái trong mỗi kết nối giữa trình khách và trình chủ của ứng dụng Web CGI cũng rất khó quản lý.

CGI sau này sẽ được Java cải tiến thành công nghệ Servlet tối ưu hơn mà chúng ta sẽ có dịp nghiên cứu ở phần sau.

1.3. Perl hỗ trợ thông dịch CGI

Viết trình CGI cho các ứng dụng Web đòi hỏi bạn phải biết một trong các ngôn ngữ lập trình như C hay Pascal (trong Unix và Linux bạn có thể dùng CGI hay các lệnh script có sẵn). Bạn có thể dùng một trình CGI duy nhất để thông dịch và thực thi các lệnh chứa trong một tập tin kịch bản (script) nào đó. Ngôn ngữ kịch bản Perl là trình thông dịch CGI mang tính năng này. Perl có cú pháp tựa C và được dùng khá rộng rãi trên Unix và Linux (trên Unix và Linux Perl được xem là một ngôn ngữ lập trình chính thống có khả năng biên dịch như C, trên Windows hiện Perl chỉ có vai trò thông dịch cho các script CGI). Ví dụ bạn có thể viết các lệnh script bằng ngôn ngữ Perl và lưu vào tập tin .pl như sau

```
Ví dụ 4-1 Hello.pl
printf("<HTML>");
printf("Hello Perl");
printf("</HTML>");
```

Khi bạn gửi yêu cầu http://www.myserver.com/hello.pl lên máy chủ Web Server. Trình chủ Web Server được cấu hình để gọi chương trình CGI Perl.exe. Chương trình Perl.exe sẽ đọc tập tin kịch bản hello.pl. Thực thi các lệnh chứa trong kịch bản và trả kết quả về cho trình khách.

Như bạn thấy Perl đã mở rộng tính năng của các trình CGI phục vụ cho Web Server. Mở đầu cho thời kỳ viết các ứng dụng biên dịch kịch bản (scripting) phục vụ cho phía máy chủ (server side).

Với Java, thực thi và biên dịch các lệnh kịch bản phía máy chủ Web Server được biết đến bằng một công nghệ khá mới mẽ và thú vị đó là JSP. Chúng ta cũng sẽ lần lượt tìm hiểu chi tiết về JSP trong phần sau.

1.4. ISAPI, ASP, Servlet, JSP

Nhược điểm của trình CGI trên máy chủ như bạn đã biết đó là vấn đề tốc độ. Mỗi lần trình CGI thực thi là mỗi lần Web Server phải nạp và giải phóng trình CGI khỏi bộ nhớ. Công nghệ Web đưa ra rất nhiều giải pháp khắc phục yếu điểm của các trình CGI.

Windows cung cấp cho bạn cách thức xây dựng các trình CGI ở dạng thư viện liên kết động DLL. Ứng dụng CGI kiểu mới này còn được gọi là các ứng dụng ISAPI. Chương trình ISAPI chỉ phải nạp một lần vào bộ nhớ khi lần đầu tiên Web Server triệu gọi ứng dụng. Do không phải nạp lại nhiều lần nên vấn đề tốc độ mà các chương trình CGI gặp phải đã được giải quyết triệt để. Hơn nữa ứng dụng ISAPI phục vụ Web Server hiệu quả và ít tiêu tốn tài nguyên của hệ thống hơn các ứng dụng CGI thông thường.

Để viết ISAPI bạn có thể sử dụng bất kỳ công cụ phát triển trên Windows nào hổ trợ việc biên dịch thư viện liên kết động DLL (như Visual C++, Inprise Delphi ...).

Về phía Java, câu trả lời tương tự cho ISAPI của Windows đó là Servlet. Servlet là các chương trình Java chỉ cần nạp một lần vào máy ảo. Servlet sẽ phục vụ mọi yêu cầu từ máy khách gửi đến tương tự chức năng của một chương trình CGI hay ISAPI.

*Nếu thích tìm hiểu về ISAPI hay cách thức xây dựng một chương trình CGI trong Windows bạn có thể tham khảo tài liệu "**Lập trình Delphi**" mà chúng tôi đã xuất bản.*

Đối với ngôn ngữ kịch bản, Microsoft đề xuất phương án thiết kế các trang Web ASP (Active Server Page). ASP là các trang Web chứa

mã lệnh viết bằng ngôn ngữ Visual Basic kết hợp với các thẻ định dạng của ngôn ngữ HTML. Khi máy khách triệu gọi một tập tin .asp, trình chủ Web Server sẽ yêu cầu trình dịch ASP chuyển tải mã lệnh ASP thành kết xuất HTML trả về cho trình khách. Ví dụ, trang ASP

Ví dụ 4-2 HELLO.ASP

```
<HTML>
<% Response.write("Hello User") %>
<% Response.write("Today is ")+ Date() %>
</HTML>
```

Khi máy khách yêu cầu trang ASP bằng địa chỉ URL http://www.myserver.com/hello.asp trình chủ Web Server sẽ diễn dịch và trả về kết quả sau cho trình duyệt trên máy khách

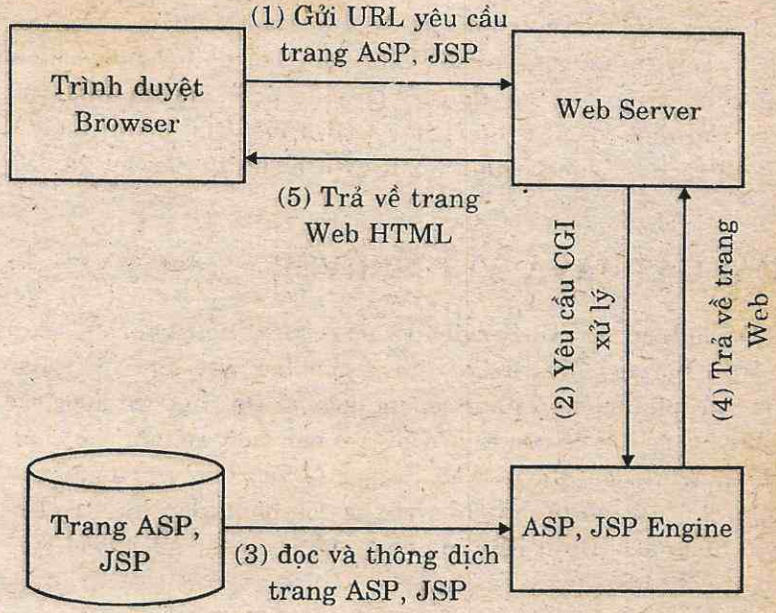
```
<HTML>
  Hello User
  Today is Fri 13, 2000
</HTML>
```

Không chịu thua kém Microsoft, Sun đưa ra khái niệm tương tự là Java Server Page (JSP). Thay vì trang Web viết bằng ngôn ngữ Visual Basic như ASP, các trang JSP viết bằng ngôn ngữ Java. Ví dụ

Ví dụ 4-3 HELLO.JSP

```
<HTML>
<% out.println("Hello User"); %>
<% Date date = new Date() %>
<% out.println("Today is ")+ date %>
</HTML>
```

Cơ chế làm việc của các trang JSP, ASP có thể được minh hoạ ở hình 4-2



Hình 4-2 Cơ chế làm việc của ứng dụng ASP, JSP

Bộ diễn dịch ASP Engine hiện nay được cài đặt kèm theo trình chủ IIS (Internet Information Service) là Web Server của hệ điều hành Windows phiên bản NT và 2000. Bộ diễn dịch JSP Engine thường được cài đặt trong các trình chủ Web Server "hiểu Java" như JRun, WebLogic, hay Java Web Server.

Với các trình chủ không hiểu Java như IIS của Windows bạn có thể cài các plug-in thông dịch Java như ServletExec hay JRun Connector. Các công cụ này chúng tôi đều thu thập trên đĩa CD với đầy đủ tài liệu để bạn tham khảo.

Tóm lại, cơ bản một trình chủ Web Server sẽ trả về một tài nguyên theo yêu cầu của máy khách. Tài nguyên do Web Server trả về có thể là tĩnh (ở dạng tập tin tồn tại trên đĩa), có thể là động (sinh ra do chương trình). Việc sinh ra các tài nguyên động bằng chương trình có thể dựa vào CGI, ASP, Perl, JSP, Servlet ... Trong

các chương ở phần sau chúng ta sẽ tập trung nghiên cứu về Servlet và JSP. Đây là hai công nghệ Web then chốt do hãng Sun đề xướng nhằm phát triển các ứng dụng Web phía máy chủ thuần Java. Bạn có thể tìm hiểu về ASP, Perl hay CGI ở một giáo trình khác. Khái niệm và cách sử dụng chúng hoàn toàn tương tự Servlet và JSP của Java.

2. CÀI ĐẶT JAVA WEB SERVER

Để nghiên cứu về Servlet và JSP trước tiên bạn phải có trình chủ Web Server "hiểu Java" (Java enabled Web Server). Nghĩa là trình chủ phải có khả năng thông dịch và gọi thực thi được các đơn thể Java theo đặc tả của Sun. Để giúp bạn tiện nghiên cứu chúng tôi sử dụng trình chủ Java Web Server (JWS) bản thử nghiệm (trial version 2.0) lấy về từ địa chỉ website của hãng Sun. Bạn có thể liên hệ với nhà sách Minh Khai để chép đĩa CD hoặc download trực tiếp về từ địa chỉ http://www.sun.com

Trình JWS 2.0 mang đầy đủ chức năng của một Web Server thông thường đồng thời cung cấp cho bạn khả năng triệu gọi mã Java thông qua JSP và Servlet.

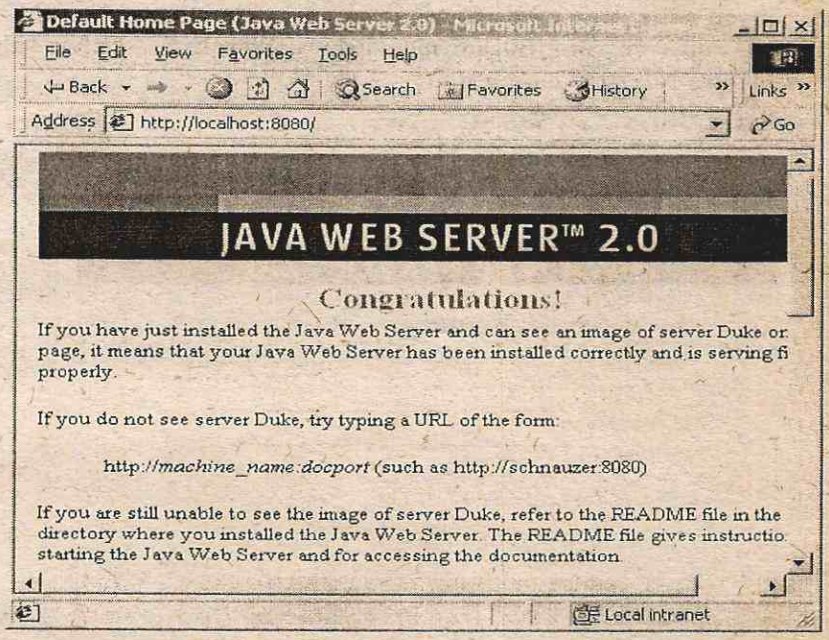
Bạn cài đặt JWS 2.0 vào thư mục C:\JavaWebServer2.0. Chuyển vào thư mục \bin bạn khởi động các dịch vụ Web bằng lệnh sau

```
C:\JavaWebServer2.0\bin>httpd.exe
javawebserver: This version of the Java Web Server (2.0)
javawebserver: for more information about this product
contact http://www.sun.com/software/jwebserver/
```

JWS 2.0 cung cấp cho bạn 2 dịch vụ Web hoạt động trên hai cổng: cổng 8080 là dịch vụ Web thông thường và cổng 9090 là dịch vụ Web phục vụ cho cấu hình trình chủ. Các dịch vụ này hoạt động ở hậu trường. Chính vì vậy sau khi khởi động httpd bạn cứ để cửa sổ dòng lệnh DOS-Prompt hoạt động trong tình trạng chờ. Các dịch vụ Web luôn trong trạng thái sẵn sàng phục vụ bạn.

2.1. Dịch vụ Web thông thường

Nếu cài đặt và khởi động trình httpd.exe thành công bạn có thể gõ địa chỉ http://localhost:8080 để kiểm tra (thay địa chỉ localhost bằng địa chỉ IP của máy bạn nếu có). Kết quả trả về trên trình duyệt Browser được hiển thị như hình 4-3



Hình 4-3 Trang chủ của trình JWS 2.0

Trang Web mặc định của dịch vụ Web hoạt động trên cổng 8080 mang tên index.htm đặt trong thư mục C:\JavaWebServer2.0\public_html. Thư mục này cũng được xem là thư mục gốc của dịch vụ Web. Ví dụ bạn có thể tạo trang MyPage.html của riêng mình, đặt vào thư mục C:\JavaWebServer2.0\public_html và yêu cầu trình duyệt truy xuất trang Web bằng địa chỉ URL như sau

http://localhost:8080/MyPage.html

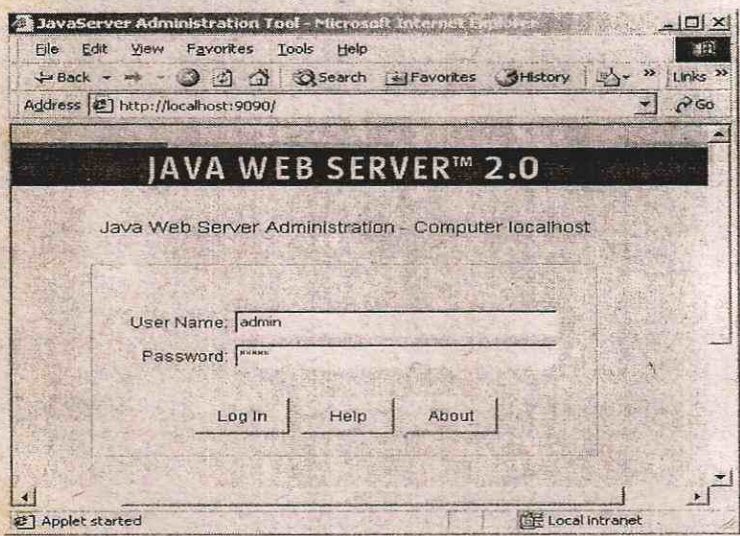
Thông thường các trình chủ Web Server hoạt động trên cổng 80. Chúng ta sẽ tìm hiểu cách đổi cổng hoạt động cho dịch vụ Web ở mục sau. Lợi ích của việc cấu hình dịch vụ Web Server phục vụ cổng

80 đó là bạn không phải gõ số hiệu cổng trên địa chỉ URL khi yêu cầu trình duyệt truy tìm trang Web. Ví dụ hai địa chỉ URL sau đều được trình duyệt trả về cùng một trang Web index.html

http://www.java.sun/index.html

http://www.java.sun:80/index.html

Tuy nhiên do 80 là cổng khá thông dụng của dịch vụ Web nên Sun chuyển sang cổng 8080 để tránh cho bạn khỏi đụng phải những socket đã mở trên cổng 80 bởi một dịch vụ Web thật sự khác (như của Windows NT/2000 chẳng hạn)



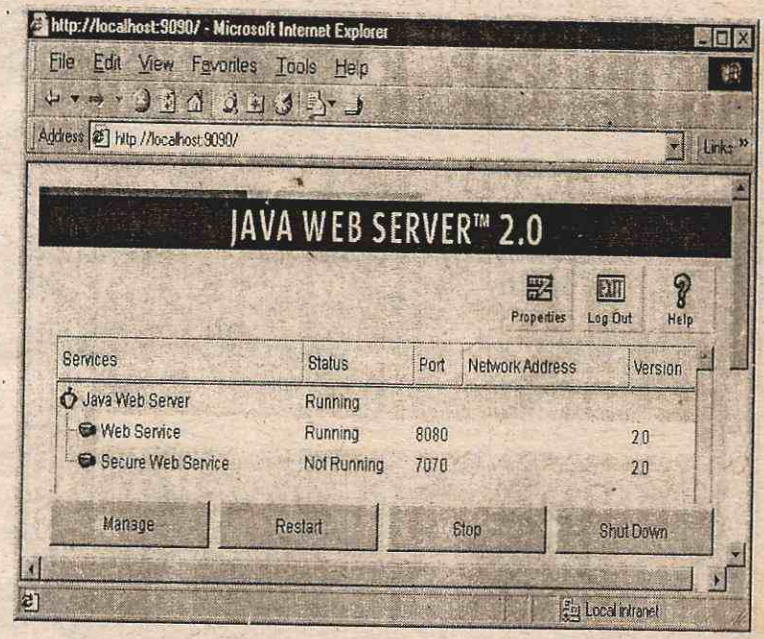
Hình 4-4 Màn hình đăng nhập cấu hình dịch vụ Web của trình chủ

2.2. Dịch vụ Web cấu hình trình chủ

Dịch vụ cấu hình trình chủ JWS2.0 hoạt động trên cổng 9090. Bạn đánh địa chỉ URL http://localhost:9090/ để truy xuất dịch vụ cấu hình trình chủ (thay địa chỉ localhost bằng địa chỉ IP của máy bạn nếu có). Hình 4-4 là màn hình đăng nhập (login) hiển thị bởi trình duyệt Browser Internet Explorer sau khi bạn gõ vào địa chỉ URL trên

Với vai trò là người quản trị Web Server bạn có thể gõ vào tên đăng nhập (username) là admin và mật khẩu (password) admin. Bạn sẽ được hướng dẫn đi tiếp vào các trang Web cấu hình mà chúng ta sẽ xem qua sau đây.

3. CẤU HÌNH JAVA WEB SERVER

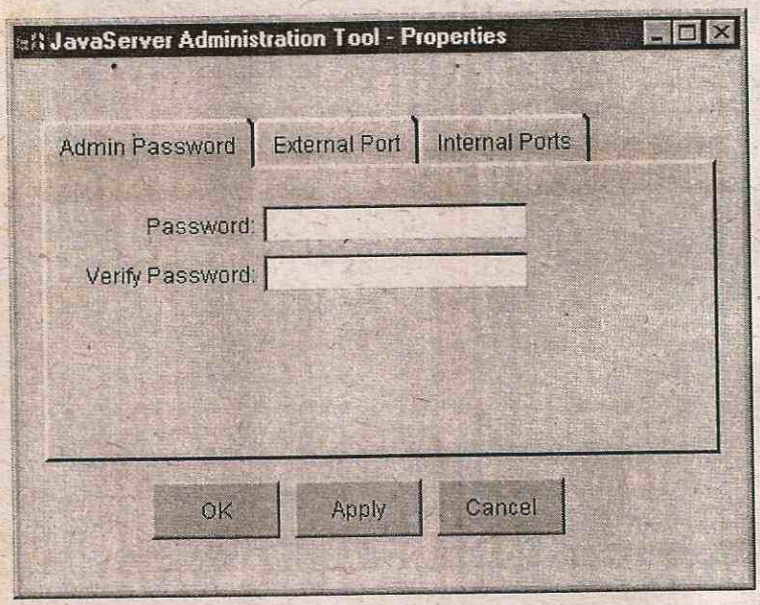


Hình 4-5 Giao diện cấu hình dịch vụ Web

Bạn có thể bỏ qua mục cấu hình Java Web Server này và chuyển sang nghiên cứu ngay về Servlet ở mục sau. Tuy nhiên, cấu hình Web Server cũng là bước cần thiết để bạn tìm hiểu về môi trường mà Servlet hay các trang JSP hoạt động. Dưới đây là tóm tắt các chức năng của trình chủ Java Web Server mà ta sẽ gặp phải trong quá trình xây dựng ứng dụng Java Servlet hay JSP trên Web

Sau khi đăng nhập vào dịch vụ quản trị Web Server của trình chủ theo địa chỉ http://localhost:9090/. Trang Web cấu hình các thông số cho dịch vụ Web được trình bày ở hình 4-5.

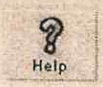
Kích chuột vào nút Properties nếu bạn muốn thay đổi một số tùy chọn của người quản trị. Cửa sổ hình 4-6 cho phép bạn thay đổi password và số cổng (mặc định 9090) của dịch vụ cấu hình Web dành cho người quản trị.



Hình 4-6 Cửa sổ thay đổi tùy chọn cho người quản trị

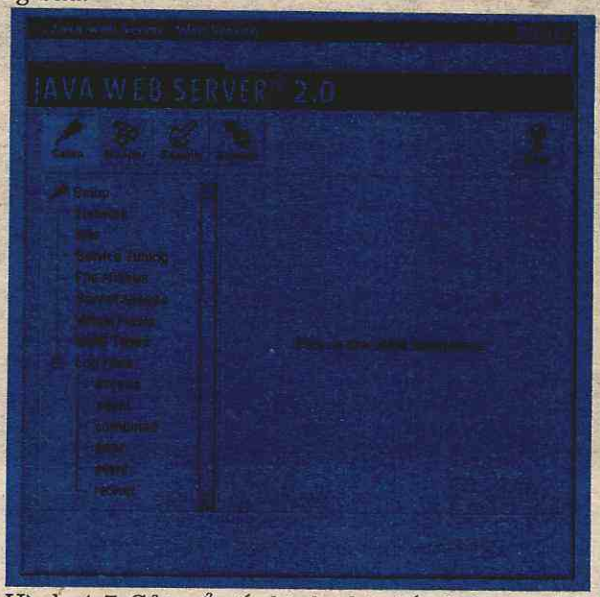
Chọn *Admin Password* nếu bạn muốn thay đổi mật khẩu. Chọn *External Port* nếu bạn muốn thay đổi cổng mặc định 9090. Chọn Internal Port nếu bạn muốn thay đổi cổng dùng cho mục đích riêng của người quản trị

Kích chuột vào nút Logout nếu bạn muốn thoát khỏi trang Web cấu hình, trở về màn hình đăng nhập.



Kích chuột vào nút Help bạn sẽ đọc được tài liệu trợ giúp của Java về cách sử dụng và cài đặt chương trình JWS2.0

Như bạn thấy ở hình 4.5, hiện chúng ta đang có hai dịch vụ Web: một chạy trên cổng 8080 (mang tên *Web Service*) và một chạy trên cổng 7070 (mang tên *Secure Web Service*). Dịch vụ bảo mật Secure Web Service chủ yếu phục vụ cho giao thức https hay cơ chế kết nối bảo mật SSL (Security Socket Layer) sẽ được chúng ta đề cập trong chương sau.

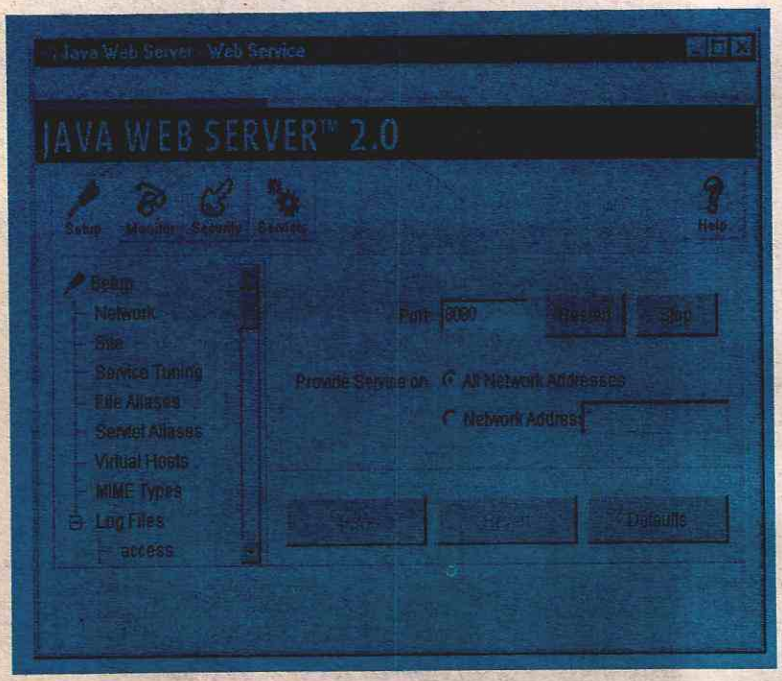


Hình 4-7 Cửa sổ cấu hình chi tiết dịch vụ Web

Để cấu hình cho dịch vụ Web Service trên cổng 8080 bạn kích đôi chuột vào mục Web Service. Trình duyệt sẽ mở cửa sổ cấu hình thứ hai chi tiết hơn liên quan đến dịch vụ Web (hình 4-7)

Từ cửa sổ này bạn có thể cấu hình các tùy chọn sau

- *Network*: Chọn mục này bạn có thể đặt lại số hiệu cổng cho dịch vụ Web. Kích hoạt hoặc ngưng quá trình phục vụ của dịch vụ Web trên cổng chỉ định (hình 4-8).

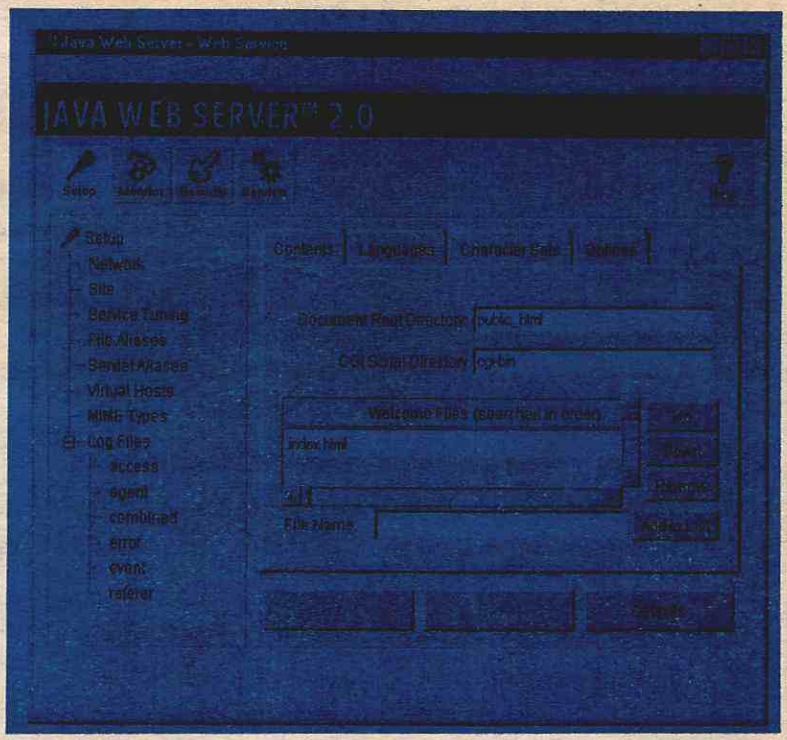


Hình 4-8 Cửa sổ cấu hình của mục chọn Network

- *Site*: Chọn mục này bạn có thể chỉ định thư mục gốc (webroot). Mặc định khi người dùng truy cập vào website của bạn thư mục C:\JavaWebServer2.0\pubic_html được xem là thư mục gốc. Bạn có thể đổi đường dẫn này theo nhu cầu của mình. Tùy chọn *CGI Scripts Directory* chỉ định thư mục chứa các đoạn mã lệnh kịch bản có thể triệu gọi mà không gây nguy hiểm đến máy chủ. Ngoài ra bạn có thể chỉ định ở mục *Welcome files* danh sách các file sẽ được dùng làm trang chủ khi người dùng yêu cầu địa chỉ URL trống, ví dụ như với địa chỉ trống http://localhost:8080/ dịch vụ Web sẽ trả về tập tin index.html ở đầu danh sách.

Mục *Site* cũng cho phép bạn chỉ định ngôn ngữ diễn dịch trang Web (*Languages*), tập ký tự sử dụng (*Character Sets*) và một số tùy chọn kiểm tra bảo mật khác ở phần *Options* (Hình 4-9)

- *Service Tunning*: Mục chọn này dùng để hiệu chỉnh một số tính năng nhằm giúp trình chủ tăng tốc trong quá trình phục vụ trình khách (như giới hạn số tuyến thread khi xử lý đồng thời các kết nối song song, dự trữ vùng nhớ cache, cho phép số kết nối tối đa được phép kết nối với trình chủ, giới hạn thời gian chờ timeout cho các thao tác xử lý tiêu tốn thời gian của hệ thống ...). Hình (4-10)



Hình 4-9 Cửa sổ cấu hình cho mục chọn Site

- *File Alias*: Mục này cho phép bạn ánh xạ tên thư mục hay còn gọi là tạo thư mục ảo (hình 4-11)

  Tất cả các thư mục con bên dưới thư mục gốc (webroot) đều có thể dễ dàng truy xuất bằng đường dẫn tương đối URL. Ví dụ, nếu C:\JavaWebServer2.0\public_html được xem là thư mục gốc webroot, địa chỉ URL trỏ đến thư mục này được truy xuất như sau

  http://localhost:8080/

  với thư mục con C:\JavaWebServer2.0\public_html\mypages bạn có thể truy xuất theo địa chỉ URL

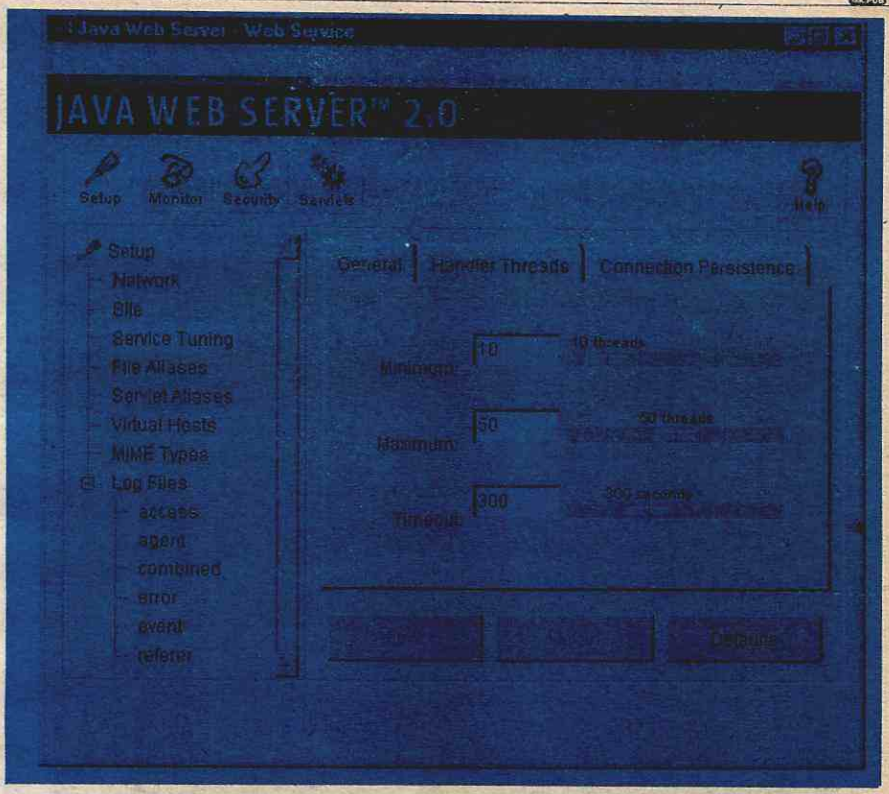
  http://localhost:8080/mypages/

Tuy nhiên với kỹ thuật ánh xạ tên (alias) bạn có thể đặt thư mục mypages ở một vị trí khác trên đĩa cứng (thậm chí với một tên khác) mà địa chỉ URL truy xuất đến vẫn không thay đổi. Ví dụ, bạn hãy tạo một dòng mới (nhấn nút Add) đặt *Alias Name* trị /mypages, đặt *Path Name* trị $server_home/helpme (tương đương với đường dẫn thư mục C:\JavaWebServer2.0\helpme. Biến môi trường $server_name tương đương với vị trí thư mục C:\JavaWebServer2.0). Khi đó thư mục helpme tuy không nằm trong thư mục public_html nhưng vẫn có thể được truy xuất đến bằng cùng một địa chỉ URL

http://localhost:8080/mypages/

- *Servlet Alias*: Tương tự việc ánh xạ tên tập tin, mỗi chương trình Servlet trên máy chủ đều có thể mang một bí danh (alias). Ví dụ với bí danh

  /*.jhtml đặt cho Servlet pageCompile



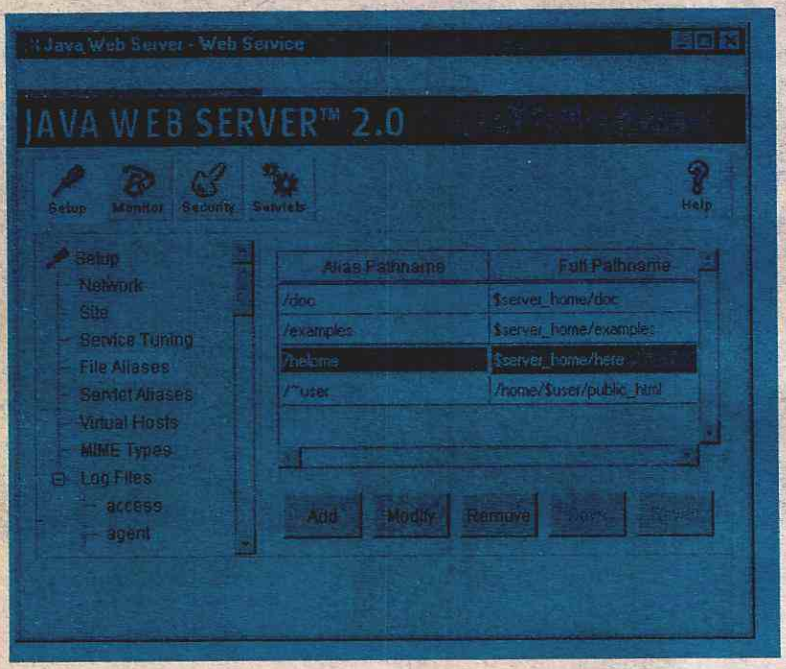
Hình 4-10 Cửa sổ cấu hình Service Tuning

khi bạn gõ địa chỉ URL http://localhost:8080/mypage.jhtml thì trình chủ sẽ triệu gọi Servlet pageCompile yêu cầu Servlet này đọc và diễn dịch tập tin mypage.jhtml sau đó trả kết quả về cho máy khách (ở đây ký tự * tương tự dấu đại diện trong cách đặt tên file của DOS). Hay với bí danh

/getDateTime đặt cho Servlet DateTime

khi bạn gõ địa chỉ URL http://localhost:8080/getDateTime trình chủ sẽ triệu gọi Servlet DateTime trả về cho bạn trang Web với thông tin về ngày giờ hiện hành.

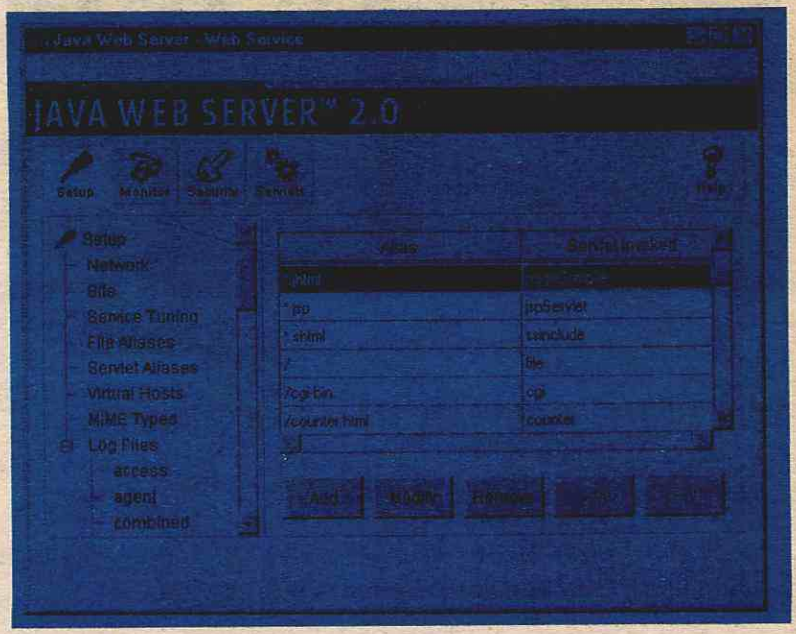
Cửa sổ cấu hình cho *Servlet Alias* được trình bày ở hình 4-12. Bạn chọn *Alias* để đặt tên bí danh cho Servlet (có thể dùng ký tự đại diện *). Chọn Servlet Invoke để đưa vào tên lớp thật sự của Servlet cần gọi tương ứng. Khi viết Servlet ở phần sau bạn sẽ thật sự hiểu rõ hơn về cách triệu gọi servlet theo bí danh này.



Hình 4-11 Cửa sổ cấu hình mục chọn File Alias

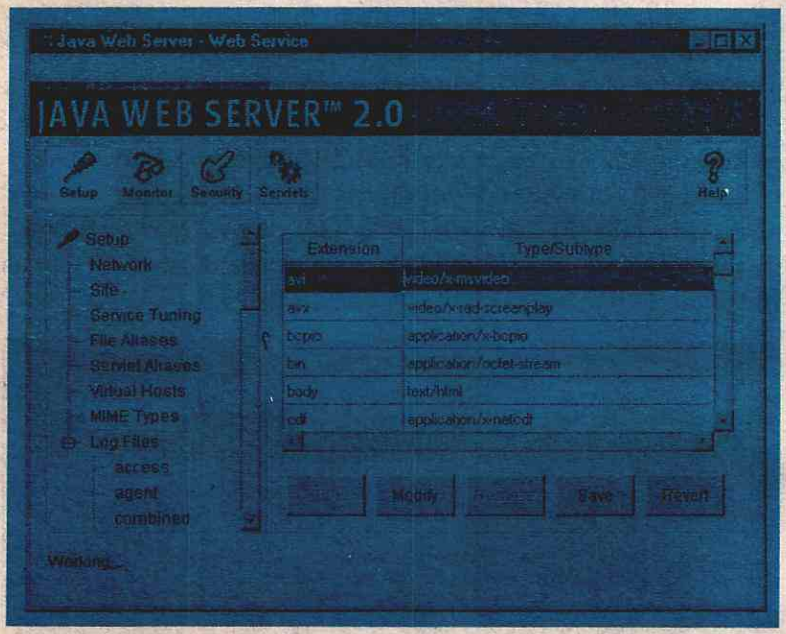
- *Virtual Host*: Mục này cho phép bạn định nhiều tên host của máy chủ trỏ đến các thư mục gốc webroot khác nhau. Thường tùy chọn này được áp dụng cho máy chủ có một địa chỉ IP nhưng ánh xạ lên nhiều tên host khác nhau (hoặc máy chủ có nhiều địa chỉ IP do sử dụng nhiều card giao tiếp mạng).

- *MIME type*: Cho phép bạn chỉ định một danh sách các kiểu tài nguyên mà dịch vụ Web có thể xử lý và hiểu được. Ví dụ file có đuôi .wav thuộc về các tập tin kiểu "audio". MIME thường được định nghĩa theo mẫu type/sub-type. Trong đó type là kiểu phân loại tổng quát của tài nguyên còn sub-type là kiểu phân loại phụ. Ví dụ tập tin văn bản .txt được định nghĩa kiểu MIME là text/plain. Tập tin .htm thật ra cũng chỉ là một tập tin văn bản nhưng được định nghĩa sub-type là text/html. Việc chỉ định kiểu MIME đối với một tài nguyên là rất quan trọng. Ví dụ, trình chủ có thể gửi tài nguyên mang kiểu MIME video/x-msvideo về cho trình khách (browser), trình khách có thể nhận dạng ra ngay tài nguyên là một tập tin video dạng .avi của Windows và triệu gọi ứng dụng tương ứng để hiển thị tài nguyên. Hình 4-13 là cửa sổ tùy chọn để bạn thêm vào các kiểu MIME.



Hình 4-12 Cửa sổ cấu hình mục chọn Servlet Alias

- *Log Files:* Mục này cho phép bạn chọn và định cấu hình cho các tập tin lần vết (log file). Những tập tin log này thường cung cấp cho bạn chi tiết thông tin về lỗi, các hoạt động diễn ra phía hậu trường của dịch vụ Web. Theo dõi ngày giờ, số lần truy cập của trình khách vào máy chủ ...



Hình 4-13 Cửa sổ tùy chọn cấu hình MIME

4. XÂY DỰNG SERVLET ĐẦU TAY

4.1. Cơ chế hoạt động của Servlet

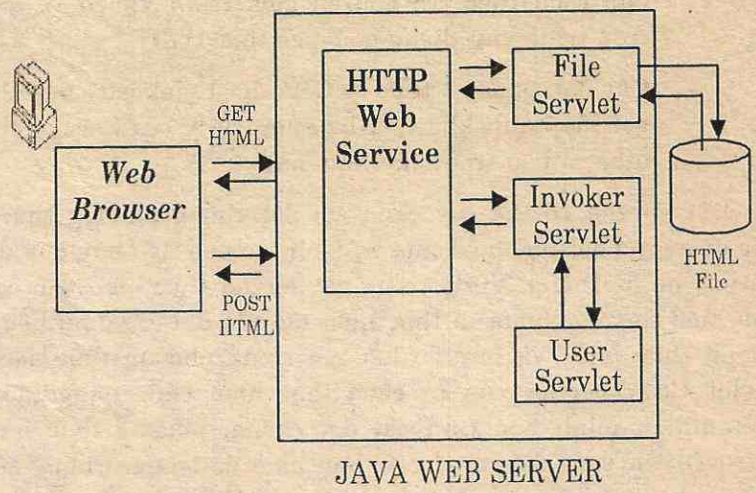
Cách thức trình khách (browser) triệu gọi servlet của dịch vụ Web server trên máy chủ được minh họa ở hình 4-14.

Trình khách gửi lệnh HTTP GET (hay POST) lên máy chủ yêu cầu dịch vụ Web (HTTP Web Service) cung cấp một tài nguyên. Tài

nguyên mà trình khách yêu cầu có thể là một tập tin (.html hay hình ảnh). Tài nguyên cũng có thể ở dạng thông tin động do servlet sinh ra

> *Servlet thật sự là một mẫu chương trình viết bằng ngôn ngữ Java được trình chủ Web triệu gọi sản sinh ra kết quả kết xuất phù hợp trả về cho trình khách.*

Trình chủ Web Server hiểu Java JWS 2.0 phục vụ hoàn toàn dựa vào servlet. Nếu bạn yêu cầu tài nguyên là một tập tin trình chủ sẽ dùng một chương trình servlet nhỏ gọi là File Servlet đọc nội dung tập tin trên đĩa cứng sau đó trả về cho trình khách. Nếu bạn yêu cầu một trình servlet khác (user servlet) thực thi để sinh ra thông tin động thì JWS 2.0 sẽ sử dụng Invoker Servlet triệu gọi servlet mà bạn mong muốn. Kết quả sinh ra sau đó cũng sẽ được gửi về trình khách tương tự như cơ chế đọc tài nguyên trên đĩa cứng mà File Servlet thực hiện.



Hình 4-14 Cơ chế hoạt động của Servlet trên máy chủ

4.2. Thiết kế Servlet CSProperties

Phần dưới đây chúng ta sẽ xây dựng một Servlet. Khi được triệu gọi servlet này sẽ trả về trang web HTML cho trình khách. Trang Web này ghi ra lời chào "Hello Servlet" đồng thời hiển thị các thông tin đọc được từ máy khách do trình duyệt chuyển lên. Các thông tin này bao gồm: tham số từ trình khách gửi đến trình chủ, các thông tin cấu hình ngay tại trình chủ Web Server. Do Servlet của ta có khả năng liệt kê những thông tin về tài nguyên của hệ thống ở cả hai phía khách chủ nên với Servlet đầu tay này ta đặt tên là CSProperties (Client Server Properties).

Khi tạo Servlet bạn chỉ cần tuân thủ theo hai bước sau

- Tạo một lớp Servlet mới kế thừa từ lớp javax.servlet.http.HttpServlet. (Thật sự thì lớp HttpServlet kế thừa từ lớp tổng quát hơn đó là javax.servlet.GenericServlet) HttpServlet cung cấp cho bạn cách thức trò chuyện giữa trình khách với Servlet trên trình chủ thông qua giao thức HTTP.
- Cài đặt phương thức doGet() hoặc doPost() đáp ứng lời gọi của trình khách. Khả năng xử lý của Servlet chủ yếu diễn ra bên trong hai hàm này.

Một servlet có thể tùy chọn cài đặt chồng phương thức init() hoặc destroy() để thực hiện một số khởi tạo ban đầu hay dọn dẹp tài nguyên cho hệ thống. Ví dụ trong các servlet thực hiện truy xuất cơ sở dữ liệu thường chúng ta thực hiện tạo kết nối cơ sở dữ liệu trong phương thức init() và hủy bỏ kết nối trong phương thức destroy(). Servlet CSProperties của ta chưa thực hiện chức năng này. Tuy nhiên nhằm minh hoạ tính cài đặt chồng, phương thức init() và destroy() vẫn được đưa vào với thân hàm để trống. Chúng sẽ được sử dụng trong phần sau. Mã nguồn servlet CSProperties được trình bày ở ví dụ 4-4

Ví dụ 4-4 CSProperties.java

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

        /**
         * Servlet này sẽ phản hồi các thông tin về tài nguyên
         * của trình khách và trình chủ đồng thời  ghi ra lời
         * chào "Hello Servlet"
         */
public class CSProperties extends HttpServlet
{

          /**
           * Thực hiện lệnh HTTP POST
           *
           * @Tham số req: Những yêu cầu hay dữ liệu do trình
 khách gửi lên
           * @Tham số resp: Phản hồi từ phía trình chủ
           */
 public void doPost(HttpServletRequest req,
                    HttpServletResponse resp)
        throws ServletException, java.io.IOException
 {
 // Chỉ định nội dung kiểu tài nguyên trả về cho trình khách
 //(Còn gọi là MIME)
   resp.setContentType("text/html");
            // Tạo luồng xuất PrintWriter để chuyển kết quả từ
            // máy chủ về cho trình khách
   java.io.PrintWriter out =
     new java.io.PrintWriter(resp.getOutputStream());
            // Định dạng trang Web HTML trả về
```

```
out.println("<html>");
out.println("<head>");
out.println("<title>Java Servlets -
                        Properties</title>");
out.println("</head>");
out.println("<h2><center>");
out.println("Hello Servlet </center></h2><br>");
out.println("Information About You");
out.println("<br>");

        // Tạo bảng chứa thông tin về trình khách
out.println("<center><table border>");
out.println("<tr>");
out.println("<td>Method</td>");
out.println("<td>" + req.getMethod() + "</td>");
out.println("</tr>");
out.println("<tr>");
out.println("<td>User</td>");
out.println("<td>" + req.getRemoteUser() + "</td>");
out.println("</tr>");
out.println("<tr>");
out.println("<td>Client</td>");
out.println("<td>" + req.getRemoteHost() + "</td>");
out.println("</tr>");
out.println("<tr>");
out.println("<td>Protocol</td>");
out.println("<td>" + req.getProtocol() + "</td>");
out.println("</tr>");

        //Lấy các thông số do trình khách gửi lên
java.util.Enumeration enum =
                        req.getParameterNames();
while (enum.hasMoreElements()) {
 String name = (String) enum.nextElement();
 out.println("<tr>");
 out.println("<td>Parameter '" + name + "'</td>");
 out.println("<td>" + req.getParameter(name)
                                    + "</td>");
 out.println("</tr>");
}
 out.println("</table></center><br><hr><br>");
```

```
            // Tạo bảng chứa thông tin về trình chủ
    out.println("<h2><center>");
    out.println("Server Properties</center></h2>");
    out.println("<br>");
    out.println("<center><table border width=80%>");
    java.util.Properties props = System.getProperties();
    enum = props.propertyNames();
    while (enum.hasMoreElements()) {
     String name = (String) enum.nextElement();
     out.println("<tr>");
     out.println("<td>" + name + "</td>");
     out.println("<td>" + props.getProperty(name) +
                                                "</td>");
     out.println("</tr>");
    }
    out.println("</table></center>");

            // Trang Web HTML đã định dạng xong
    out.println("</html>");
    out.flush();
   }

        /**

         * <p>Thực hiện lệnh HTTP GET

         *

         * @Tham số req: Những yêu cầu hay dữ liệu do trình
khách gửi lên

         * @Tham số resp: Phản hồi từ phía trình chủ

         */
public void doGet(HttpServletRequest req,
                  HttpServletResponse resp)
     throws ServletException, java.io.IOException
{
   doPost(req, resp);
}

        /**
```

```
        * Phương thức khởi tạo của servlet.
        * @tham số: cfg các thông tin của môi trường ngoài
truyền vào servlet
        */
public void init(ServletConfig cfg)
                                    throws ServletException
  {
   super.init(cfg);
  }
        /**
        * Phương thức dọn dẹp của servlet. Phương thức này
được gọi khi Web server
        * giải phòng servlet khỏi bộ nhớ
        */
public void destroy()
  {
   super.destroy();
  }
}
```

CSProperties servlet của ta phục vụ lệnh doPost() và doGet() do trình khách gửi lên. Phương thức doGet() đơn giản chỉ gọi lại doPost() để thực hiện cùng một chức năng Đối tượng HttpServletRequest được dùng để lấy về thông tin của trình khách. Luồng xuất PrintWriter tạo điều kiện để servlet gửi dữ liệu về trình duyệt browser trên máy khách.

Bạn biên dịch CSProperties.java từ dòng lệnh tương tự như biên dịch chương trình Java thông thường

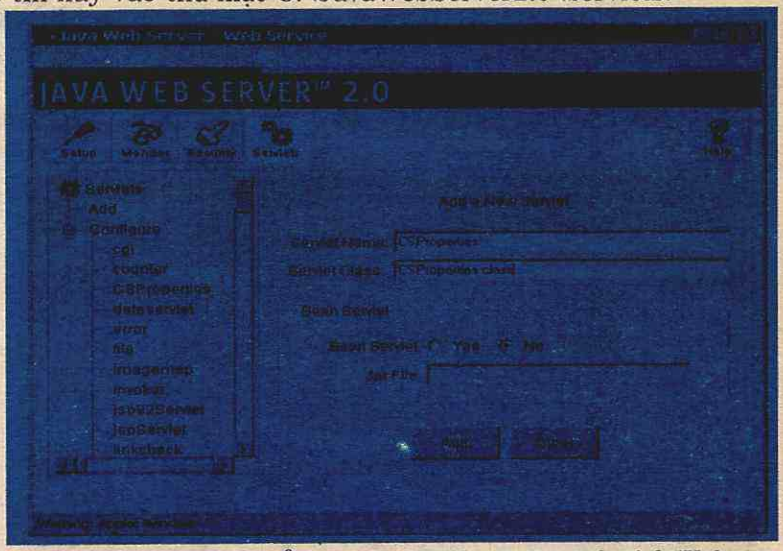
```
javac CSProperties.java
```

Kết quả ta thu được tập tin CSProperties.class thật sự là một đối tượng Java Servlet.

4.2.1 Cấu hình Servlet

Trước khi gọi được servlet CSProperties chúng ta cần cấu hình trên dịch vụ Java Web server. Quá trình cấu hình sẽ chỉ định tên và cách thức mà dịch vụ Web dùng nạp servlet. Bạn mở trang Web quản trị dịch vụ Web http://localhost:9090/ mà ta đã tìm hiểu ở phần đầu. Sau khi đăng nhập bạn chọn nút *Servlet*. Từ cửa sổ cấu hình cho *Servlet* bạn chọn mục *Add* (hình 4-15). Gõ vào tên cho servlet ở ô *Servlet Name* (tên của servlet không nhất thiết phải trùng với tên lớp .class) ở đây để dễ nhớ ta đặt trùng tên là CSProperties. Gõ vào tên lớp CSProperties.class cho servlet ở ô *Servlet class*. Nếu servlet của bạn được đóng gói thành tập tin .jar bạn có thể chọn mục *Yes* ở phần *Bean Servlet*. Nhấn nút *Add*, dịch vụ Web sẽ đưa servlet của bạn vào danh sách các servlet mà trình chủ có thể phục vụ.

Để dịch vụ Web tìm được tập tin CSProperties.class bạn chép tập tin này vào thư mục C:\JavaWebServer2.0\servlets.



Hình 4-15 Cửa sổ đăng ký servlet với trình chủ Web

4.2.2 Viết trang Web HTML triệu gọi Servlet

Sau khi cấu hình, để gọi được servlet CSProperties bạn cần phải thiết kế một trang HTML thực hiện lời triệu gọi dịch vụ Web trên máy chủ theo giao thức HTTP POST. Nội dung trang Web Properties.html được trình bày ở ví dụ 4-5 dưới đây

Ví dụ 4-5 Properties.html

```
<HTML>
  First Servlet Testing <br>
  <FORM ACTION=/servlet/CSProperties METHOD=POST>
     <INPUT TYPE=TEXT NAME=yourName>
     <INPUT TYPE=Submit VALUE="Call servlet">
  </FORM>
</HTML>
```

Bạn chép tập tin properties.html vào thư mục Web gốc C:\JavaWebServer2.0\public_html.

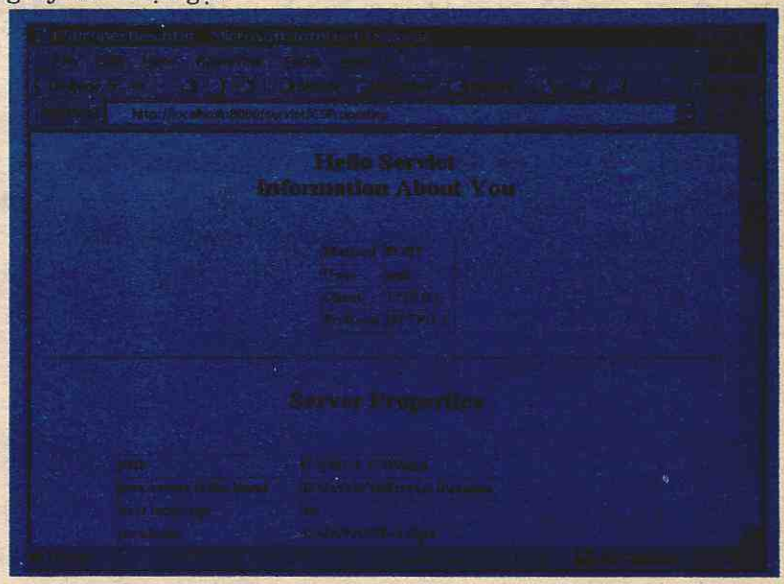
Trước hết bạn gõ địa chỉ http://localhost:8080/properties.html để hiển thị trang Web thực hiện lời triệu gọi servlet. Trang Web được hiển thị như hình 4-16. Bạn gõ tên của mình vào ô textbox và kích chuột vào nút nhấn *Call Servlet* trên trang Web. Trình duyệt browser sẽ thực hiện gọi CSProperties bằng phương thức POST. Kết quả trả về của CSProperties được hiển thị ngay trên trình duyệt browser (hình 4-17).

Hình 4-16 Trang Web trước khi triệu gọi Servlet

Như bạn thấy, để trang Web có thể thực hiện phương thức POST ta sử dụng thẻ <FORM>

```
<FORM ACTION=/servlet/CSProperties METHOD=POST>
   <INPUT TYPE=TEXT NAME=yourName>
   <INPUT TYPE=Submit VALUE="Call servlet">
</FORM>
```

Thuộc tính ACTION với METHOD=POST của thẻ <FORM> sẽ chỉ định lời gọi đến servlet CSProperties của ta trên máy chủ bằng lệnh HTTP POST. Mặc định nếu bạn để đường dẫn /servlet/ thì dịch vụ Web sẽ truy tìm các servlet mà nó hiện có trong danh sách đã đăng ký để triệu gọi.



Hình 4-17 Kết quả trả về sau của servlet CSProperties

Trong servlet CSProperties, ta cài đặt thêm phương thức doGet() thực hiện cùng chức năng với doPost() (như bạn thấy, doGet() gọi lại doPost()). Phương thức doGet() phục vụ cho lệnh HTTP GET. Trừ khi bạn dùng thẻ <FORM> với thuộc tính METHOD=POST, nếu bạn gõ một địa chỉ URL, ngầm định trình duyệt sẽ sử dụng lệnh HTTP GET để yêu cầu trình chủ Web phục vụ.

Lúc này phương thức doGet() của servlet sẽ được gọi thay cho doPost(). Ví dụ nếu cài đặt doGet() bạn có thể gọi servlet CSProperties trực tiếp từ địa chỉ URL như sau

http://localhost:8080/servlet/CSProperties?yourName=JavaWorld

Kết quả nhận được hoàn toàn tương tự như khi bạn nhấn nút *Call Servlet*. Sự khác nhau giữ POST và GET chủ yếu là về cơ chế chuyển dữ liệu. Lệnh HTTP GET chuyển đối số ngay trên địa chỉ URL trong khi lệnh HTTP POST chuyển dữ liệu của đối số theo luồng. Ở ví dụ trên, địa chỉ URL thực hiện lệnh GET chứa đối số *yourName* (phân cách bằng dấu chấm hỏi). Nội dung của đối số là JavaWorld (*?yourName=JavaWorld*). Trong khi với lệnh POST đặt trong FORM, đối số của bạn được thể hiện ở dạng ô văn bản textbox mang tên yourName, và được trình duyệt chuyển đi ẩn lên trình chủ khi người dùng nhấn nút yêu cầu tác vụ POST của <FORM>

Bạn cũng có thể đặt bí danh (alias) cho Servlet với một tên khác. Bạn chọn *Servlet Alias* (xem mục 3 *Cấu hình Web Server*). Chọn *Add*, gõ vào ô *Alias* bí danh */myproperties*, gõ vào ô *Servlet Invoke* tên đăng ký CSProperties. Lúc này bạn có thể chỉ định đường dẫn để gọi servlet trực tiếp như sau

```
<FORM ACTION=/myproperties METHOD=POST>
    <INPUT TYPE=Submit VALUE="Call servlet">
</FORM>
```

Nếu bạn cài đặt doGet() thì servlet có thể triệu gọi ngay trên địa chỉ URL

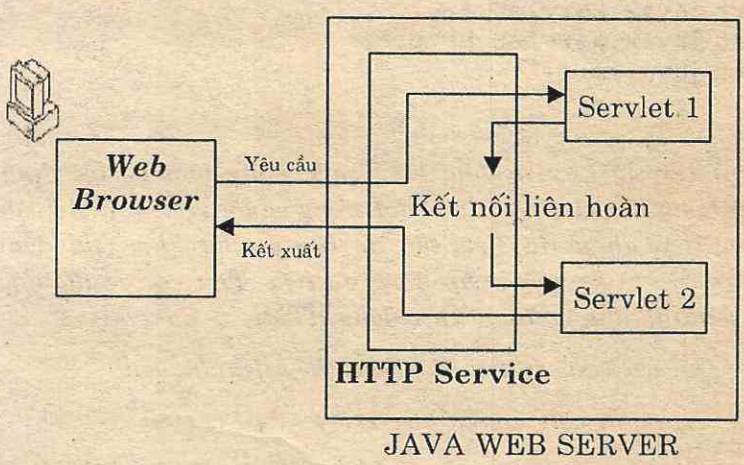
http://localhost:8080/myproperties

Thông thường nếu servlet của bạn chỉ định dạng trang Web hoặc đơn giản là trả về tài nguyên cho trình khách thì nên cài đặt doGet(). Nếu servlet cần tiếp nhận khối lượng lớn dữ liệu chứa trong thẻ <FORM> do người dùng nhập vào thì bạn nên cài đặt doPost(). Tuy nhiên như đã nêu bạn cũng có thể cài đặt cả hai phương thức doGet() và doPost() với cùng một chức năng (như ta đã làm ở ví dụ CSProperties servlet).

5. SERVLET LIÊN HOÀN (CHAINING SERVLET)

5.1. Servlet liên hoàn là gì

Tương tự như khái niệm đường ống (piping) của DOS và Unix, bạn có thể kết hợp các servlet liên hoàn lại với nhau để tạo ra một kết quả kết xuất tổ hợp duy nhất. Tính liên hoàn thể hiện ở chỗ kết xuất đầu ra của một servlet này lại là đầu vào để servlet khác xử lý. Đầu ra của servlet cuối cùng chính là kết quả trả về cho trình duyệt gọi chuỗi servlet. Cơ chế làm việc của các servlet liên hoàn được minh họa ở hình 4-18.



Hình 4-18 Cơ chế triệu gọi Servlet liên hoàn

5.2. Xây dựng ví dụ về servlet liên hoàn : bộ định dạng bảng (TableFilter)

Chúng ta hãy bắt tay vào tìm hiểu cách xây dựng các servlet có tính liên hoàn. Servlet mà ta xây dựng mang tên TableFilter. TableFilter là một bộ định dạng bảng dữ liệu thô (do một servlet khác đưa đến) thành dạng bảng HTML. Servlet của ta thực hiện chức năng phân tích từng dòng dữ liệu. Tìm các dòng cho biết số cột của bảng cần định dạng, xác định xem bảng dữ liệu có cần dòng tiêu

đề hay không ... tất cả dữ liệu của các dòng tiếp theo sẽ được đưa vào khung lưới bằng các thẻ định dạng HTML. Với servlet này sẽ giúp cho các servlet khác chuyển dữ liệu ở dạng thô (như chỉ cần đặt các hàng cột bằng dấu phân cách tab hoặc dấu phẩy) mà không cần quan tâm đến vấn đề định dạng bảng chi tiết. Công việc này được giao lại cho servlet sau cùng là TableFilter trong chuỗi liên hoàn thực hiện.

Servlet của ta được xây dựng như sau

Ví dụ 4-6 TableFilter.java

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

    /**

    Đây là servlet thực hiện chức năng làm bộ lọc định
dạng bảng. Servlet này được đặt trong chuỗi servlet liên hoàn
nhằm tiếp nhận dữ liệu thô do các servlet khác gửi đến.
servlet của ta sẽ quét từng dòng dữ liệu. Tìm ra các thẻ đặc
biệt để định dạng dữ liệu theo bảng HTML.

    Qui ước của dữ liệu thô được thể hiện như sau

      <!--table columns=n header=[yes|no]-->

    Trong đó n là số cột của bảng. Nếu header=yes thì dòng
dữ liệu thô đầu tiên sẽ được dùng làm tiêu đề cho việc định
dạng bảng. Nếu header=no thì bảng được định dạng không có
phần tiêu đề. Các cột dữ liệu theo sau phải cách nhau bằng
dấu phẩy (,) để phân định số cột.

    Bảng được nhận dạng là kết thúc khi gặp chuỗi qui ước

      <!--end table-->.

    Ví dụ bạn có thể sử dụng dữ liệu thô của bảng như sau

      <!--table columns=2 header=yes-->
```

```
                Company, Telphone
                SunMicrosystem,743555
                Microsoft,123456
                IBM,478583
              <!--end table-->
        */

public class TableFilter extends HttpServlet
{
        /**
          Thực hiện dịch vụ service của HTTP
          @Tham số req: Yêu cầu của trình khách
          @tham số resp; Kết quả trả về cho trình khách
        */
 public void service(HttpServletRequest req,
                     HttpServletResponse resp)
          throws ServletException, java.io.IOException
  {
            // Lấy về tất cả các thông tin tiêu đề cần thiết của
 servlet trước đó
   java.util.Enumeration e = req.getHeaderNames();
          //Kết xuất phần tiêu đề
  while (e.hasMoreElements()) {
    String header = (String)e.nextElement();
    String value = req.getHeader(header);
    resp.setHeader(header, value);
   }
            // Lấy về luồng xuất nhập dữ liệu
   ServletInputStream in = req.getInputStream();
```

```
ServletOutputStream out = resp.getOutputStream();

        // Chỉ xử lý kiểu tài nguyên có kiểu MIME biết
trước là dạng text
  String type = req.getContentType();

  if (type.equals("text/html") ||
     type.equals("text/table") || {
     resp.setContentType("text/html");

       // Tạo luồng đệm tiếp nhận đọc từng dòng dữ liệu thô
     BufferedReader br =
         new BufferedReader(new InputStreamReader(in));
   boolean inTable = false;
   int tableCols = 0;
   boolean headerRow = false;

          // Đọc cho đến khi hết dữ liệu thô
       while (true) {
    String s = br.readLine();

           // null cho biết đã đến cuối vùng dữ liệu
         if (s == null) {
     break;
    }

          // Nếu chúng ta đang ở vị trí giữa các dòng dữ liệu
    if (inTable) {

         // Nếu kết thúc bảng dữ liệu
     if (s.startsWith("<!--end table")) {
      out.println("</table></center>");
      inTable = false;
     }
     else {

     // Chúng ta đang đọc dữ liệi thô, gọi hàm định dạng dòng

     // dữ liệu theo thẻ HTML
      s = formatRow(s, tableCols, headerRow);
      headerRow = false;
     }
```

```
    }
    else {

            // Tìm vị trí bắt đầu của bảng
     if (s.startsWith("<!--table")) {

            //Tìm số cột qui định của bảng
      int pos = s.indexOf("columns=");
      tableCols = 0;
        if (pos >= 0) {

            // Lấy về số cột
       String cols = s.substring(pos + 8);
       int endPos = cols.indexOf(" ");
       if (endPos > 0) {
        cols = cols.substring(0, endPos);
       }
       tableCols = Integer.parseInt(cols);
      }

     //Nếu có quy định tiêu đề cho bảng = "yes" chúng ta sẽ lấy

     //dòng dữ liệu đầu tiên làm dòng tiêu đề
      pos = s.indexOf("header=");
      if(pos >= 0) {
       String flag = s.substring(pos + 7);
       headerRow = flag.startsWith("yes");
      }

    // Nếu dữ liệu lấy về hợp lệ, ta thực hiện định dạng đưa

    // dòng dữ liệu vào từng cột của khung lưới
      if (tableCols > 0) {
       out.println(s);
       s = "<center><table border>";
       inTable = true;
      }
     }
    }
    out.println(s);
   }
  } else {
```

```
        // Tài nguyên không chỉ định kiểu MIME, đơn giản chúng
        // ta phản hồi lại toàn bộ nội dung mà không cần định dạng
    while (true) {
     int b = in.read();
     if (b == -1) {
      break;
     }
     out.write(b);
    }
   }
   out.close();
  }

        /**
         * Định dạng dòng dữ liệu, đưadữ liệu  vào khung lưới
bằng thẻ HTML
         */
private String formatRow(String line, int cols,
                                        boolean header)
  {
   String s = "<tr>";
   int pos = line.indexOf(",");
   int lastPos = 0;

            // Duyệt qua tất cả các cột
      for (int i = 0; i < cols; i++) {
    if (pos < 0) {
     pos = line.length();
    }

      // Chèn thẻ HTML thích hợp để định dạng cột hoặc dòng
    if (header) {
     s += "<th>";
    }
    else {
     s += "<td>";
    }

            //Tìm dữ liệu của cột kế tiếp
```

```
    if (pos > 0) {
    s += line.substring(lastPos, pos);
    lastPos = pos;
    if (pos < line.length()) {
     lastPos = pos + 1;
     pos = line.indexOf(",", lastPos);
    }
    else {
     pos = 0;
    }
   }

      // Chèn thẻ HTML thích hợp để định dạng cột hoặc dòng
   if (header) {
    s += "</th>";
   }
   else {
    s += "</td>";
   }
  }

          // Trả về dòng dữ liệu đã được định dạng
  return s;
 }
 /**
 * Phương thức khởi tạo được gọi khi Servlet nạp vào bộ nhớ
 * Bạn có thể dùng phương thức này để khởi tạo các biến cần thiết
        */
public void init(ServletConfig cfg)
                        throws ServletException
 {
  super.init(cfg);
 }

        /**
        * Phương thức hủy Servlet. Phương thức này được gọi
khi servlet kết thúc quá trình hoạt động và giải phóng khỏi bộ
```

nhớ. Bạn có thể lợi dụng phương thức hủy để giải phóng các tài nguyên đã cấp phát trong phương thức init() nếu có.

```
        */
public void destroy()
 {
  super.destroy();
 }

        /**

        * Trả về mục đích sử dụng của mà servlet. Phương
thức này cung cấp thông

        * tin cho người sử dụng servlet.

        */
public String getServletInfo() {
   return "Table Filter for Chaining";
 }
}
```

Ở ví dụ trên TableFilter không cài đặt phương thức doPost() hoặc doGet(). TableFilter cài đặt phương thức service(). Phương thức service() giúp bạn tiếp nhận dữ liệu gửi đến là kết xuất của một servlet khác. Trong phương thức service() trước hết ta phản hồi lại tất cả phần thông tin header do servlet trước đó sản sinh ra (nếu bạn muốn thay đổi nội dung của phần kết xuất header thì có thể thực hiện ở bước này)

```
        //Phản hồi phần tiêu đề
 while (e.hasMoreElements()) {
    String header = (String)e.nextElement();
    String value = req.getHeader(header);
    resp.setHeader(header, value);
  }
```

Tiếp đến, sau khi lấy về luồng xuất nhập bằng lệnh

```
        // Lấy về luồng xuất nhập dữ liệu
  ServletInputStream in = req.getInputStream();
```

```
ServletOutputStream out = resp.getOutputStream();
```

servlet TableFilter của ta xác định kiểu tài nguyên MIME cần xử lý có phải là văn bản hay không. TableFilter chỉ chấp nhận kiểu "text/html" và "text/table". Bởi vì, TableFilter thay đổi kết xuất của tài nguyên cho nên nó qui định lại kiểu MIME bằng cách gọi resp.setContentType();

```
// Chỉ xử lý kiểu tài nguyên có kiểu MIME biết
trước là dạng text
String type = req.getContentType();
if (type.equals("text/html") ||
  type.equals("text/table") || {
//Định lại kiểu MIME kết xuất
  resp.setContentType("text/html");
```

Bước còn lại chỉ là phân tích dữ liệu để định dạng bảng. Đầu tiên TableFilter tìm dấu hiệu cho biết nơi bắt đầu dữ liệu

```
if (s.startsWith("<!--table")) {
```

Vòng lặp while được dùng để duyệt dữ liệu cho đến khi nhận dạng được dấu hiệu kết thúc thì thoát

```
if (s.startsWith("<!--end table")) { break;
```

Quá trình phân tích sẽ dựa vào dấu phân cách hoặc dấu phẩy để phân định cột dữ liệu. Bảng được định dạng bằng cách dùng các thẻ con <td> và <tr> của thẻ <TABLE> trong ngôn ngữ HTML để qui định dòng và cột.

Như vậy là chúng ta đã hoàn tất servlet TableFilter thực hiện chức năng lọc dữ liệu thô do các servlet khác gửi đến. Để sử dụng được TableFilter ta phải xây dựng servlet phát sinh ra dữ liệu thô theo yêu cầu. Servlet RowData ở ví dụ 4-7 dưới đây sẽ thực hiện điều đó

Ví dụ 4-7 RowData.java

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

```
public class RowData extends HttpServlet
{
 public void doGet(HttpServletRequest req,
                   HttpServletResponse resp)
   throws ServletException, java.io.IOException
  {
          //Trả về kiểu dữ liệu chứa nội dung tài nguyên
 (content-type)
   resp.setContentType("text/html");
          //Tạo luồng xuất để ghi kết quả trả về
   java.io.PrintWriter out =
     new java.io.PrintWriter(resp.getOutputStream());
          //In nội dung
   out.println("<html>");
   out.println("<h2><center>");
   out.println("Java Servlets Sample Data Ouput
                                        </center></h2>");
   out.println("<br>");
          //Bảng dữ liệu thô
   out.println("<!--table columns=2 header=yes-->");
   out.println("Comapany, Telephone");
   out.println("Sun Microsystem,743555");
          //Thêm vào đây các dòng dữ liệu ...
   out.println("Microsoft,123456");
   out.println("IBM,478583");
   out.println("Fujitsu,0618563");
   out.println("</html>");
          //Kết thúc kết xuất
   out.flush();
   out.close();
 }
        /* Phương thức khởi dựng */
 public void init(ServletConfig cfg)
                  throws ServletException
  {
```

```
    super.init(cfg);
  }

        /* Phương thức hủy */
  public void destroy()
  {
    super.destroy();
  }
}
```

Bạn lưu mã nguồn TableFilter.java và RowData.java vào thư mục C:\JavaWebServer2.0\servlets. Chuyển vào thư mục này đánh lệnh biên dịch

```
javac TableFilter.java
javac RowData.java
```

Kết quả thu được là hai tập tin servlet .class. Thực hiện đăng ký hai servlet này với dịch vụ Web tương tự CSProperties. Chọn *Servlet*, chọn *Add* từ trang Web cấu hình trình chủ (địa chỉ http://localhost:9090/). Đặt tên cho hai servlet cùng với tên lớp (TableFilter tương ứng với TableFilter.class và RowData tương ứng với RowData.class). Java Web Server cung cấp cho bạn hai cách kết nối sự liên hoàn của hai servlet này với nhau.

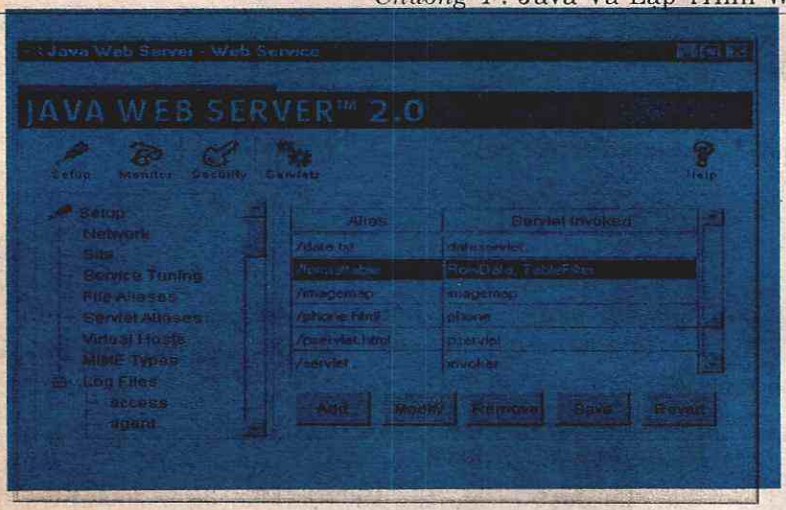
5.3. Cấu hình kết nối và gọi servlet liên hoàn

5.3.1. Cấu hình servlet liên hoàn bằng bí danh (alias)

Bạn chọn *Servlet Aliases* từ trang Web cấu hình (xem mục 4-9). Nhấn phím *Add*, gõ vào ô cột *Alias* chuỗi */formattable*. Gõ vào cột *Servlet Invoke* chuỗi *RowData, TableFilter* (hình 4-19). Bạn lưu ý chúng ta đặt RowData và TableFilter cách nhau bằng dấu phẩy. Với cách đặt bí danh này khi bạn gõ địa chỉ triệu gọi

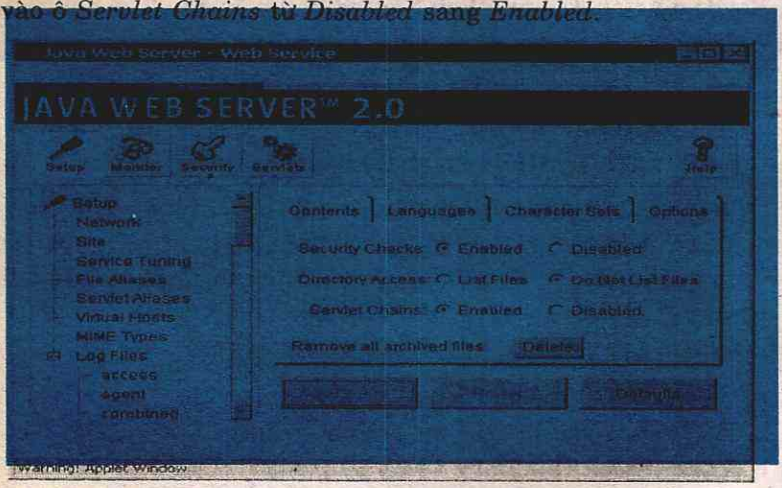
http://localhost:8080/formattable

trình chủ Web server sẽ gọi thực thi sevlet RowData trước. Tiếp đến kết quả trả về của RowData sẽ được servlet TableFilter đón nhận và xử lý. Cuối cùng TableFilter sẽ trả kết quả đã định dạng về cho trình khách.



Hình 4-19 Cửa sổ cấu hình đặt bí danh cho servlet

Mặc định Java Web Server không cho phép các lời gọi servlet liên hoàn với nhau. Để bật chức năng này bạn chọn mục *Site* trên trang Web cấu hình (hình 4-20). Tiếp đến chọn tab-*Options*. Đánh dấu vào ô *Servlet Chains* từ *Disabled* sang *Enabled*.



Hình 4-20 Cửa sổ tùy chọn cho phép thực hiện servlet liên hoàn

Quá trình cấu hình đã hoàn tất. Để kiểm tra kết quả bạn hãy thử gõ vào trình duyệt địa chỉ

http://localhost:8080/servlet/RowData

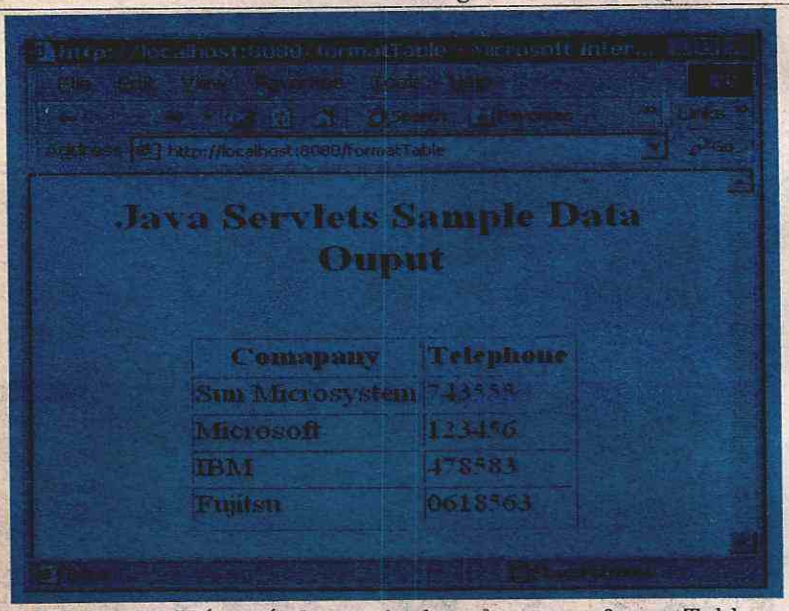
Địa chỉ trên sẽ trả về kết quả là dữ liệu thô thể hiện ở hình 4-21. Bây giờ nếu bạn gõ địa chỉ

http://localhost:8080/servlet/formattable

Kết quả trả về là sự kết hợp liên hoàn giữ servlet RowData và TableFilter (hình 4-22) . Bảng dữ liệu do RowData xuất ra giờ đây được định dạng trong khung lưới hiển thị rõ ràng và đẹp hơn.

Hình 4-21 Kết xuất thô từ servlet RowData

Với các trình chủ khác Java Web Server như JRun hay WebLogic bạn có thể tham khảo thêm tài liệu mà các nhà cung cấp kèm theo.



Hình 4-22 Kết xuất đã qua bộ lọc của servet formatTable

5.3.2. Cấu hình servlet liên hoàn bằng cách chỉ định kiểu MIME

Cách thứ hai để kết nối các lời gọi servlet liên hoàn là sử dụng tính năng định kiểu tài nguyên MIME khi kết xuất. Bạn để ý, khi bắt đầu ghi kết xuất trả về cho trình khách bao giờ servlet của ta cũng chỉ định kiểu MIME cho tài nguyên mà nó thể hiện bằng cách gọi phương thức resp.setContentType(). Ví dụ

```
resp.setContentType("text/html");
```

sẽ cho biết dữ liệu trả về là loại tập tin văn bảng HTML. Servlet TableFilter của ta căn cứ vào hai kiểu MIME là text/html và text/table để quyết định xem có nên phân tích và xử lý dữ liệu hay không. Với mỗi kiểu MIME Java Web Server cho phép bạn gọi một servlet tương ứng xử lý (tương tự như mỗi loại tập tin Windows đều cho phép bạn kết hợp với một chương trình xử lý vậy). Nếu servlet RowData của ta xuất ra dữ liệu có kiểu MIME là text/table và bạn

qui định text/table cần phải được xử lý bởi TableFilter thì trình chủ Web Server sẽ triệu gọi servlet này. Để chỉ định loại tài nguyên MIME cần được xử lý bởi một servlet nào đó hiện bạn phải cấu hình bằng tay (không có giao diện đồ họa). Bạn vào thư mục

C:\JavaWebServer2.0\properties\server\javawebserver\webpageservice

mở tập tin mimeservlets.properties. Tập tin này có nội dung như sau

Ví dụ 4-8 mimeservlets.properties

```
# This file maps mime-types to the servlets which process them
# This is used by the filter manager to set up chains of servlets
# where the output of one servlet gets piped to the input of
# another servlet based on the mime-type that the servlet specifies
# with setContentType("mime-type")
#
# The default servlet for all mime-types is file. Do not set this
# explicitly.
#
# Entries in this file should be of the form
# mime-type/servletname
# ie.
# foo/bar=fooServlet
# where fooServlet is defined in servlets.properties
java-internal/parsed-html=pageCompile
text/table=TableFilter
```

Ta thêm vào kiểu MIME text/table cần phải được xử lý bởi TableFilter (bạn có thể tự qui định một kiểu MIME khác của riêng mình). Tiếp đến trong mã nguồn của RowData.java bạn thay lệnh

```
resp.setContentType("text/html");
```

bằng lệnh

```
resp.setContentType("text/table");
```

Đăng ký và nạp lại hai servlet RowData và TableFilter. Bật tùy chọn cho phép gọi servlet liên hoàn (hình 4-20). Bạn có thể phải khởi động lại trình chủ Web server bằng lệnh httpd để tập tin cấu hình mimeservlets.properties được cập nhật. Lúc này nếu bạn gọi servlet bằng địa chỉ

http://localhost:8080/servlet/RowData

kết quả trả về không phải là dữ liệu thô nữa mà sẽ là một bảng dữ liệu đã định dạng thông qua TableFilter.

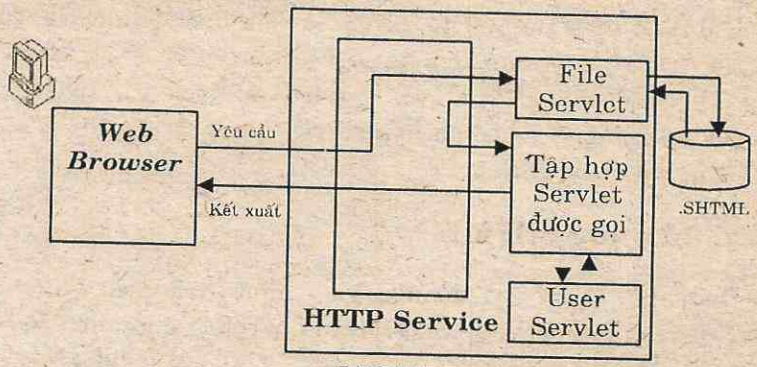
6. INCLUDE SERVLET

Ngoài việc gọi liên hoàn các servlet, kiến trúc của trình chủ Web Java còn cho phép bạn kết hợp kết quả kết xuất của nhiều servlet lại với nhau thành một kết quả duy nhất trả về cho trình khách. Cơ chế này còn được gọi là *Server-side include*.

Bạn đặt các lời gọi servlet trong thẻ qui ước <SERVLET>. Thẻ này không phải là thẻ chuẩn HTML để trình khách xử lý. <SERVLET> là thẻ được dùng phía trình chủ. Các tập tin sử dụng thẻ <SERVLET> thường đặt trong file .shtml thay cho .html (s_ với ý nghĩa *server*).

Hình 4-23 là cơ chế diễn dịch thẻ include <SERVLET> của trình chủ Web Server

- Khi bạn triệu gọi một tập tin .shtml bằng địa chỉ URL trình chủ Web sẽ đọc tập tin .shtml trên đĩa.
- Trình chủ Web gửi tập tin .shtml cho servlet mang tên $SInclude. Servlet đặc biệt này đóng vai trò như bộ diễn dịch. Nó sẽ truy tìm các thẻ <SERVLET> có trong tập tin .shtml để triệu gọi.
- Tất cả các servlet tương ứng với thẻ <SERVLET> qui định có trong tập tin .shtml đều được gọi thực thi.
- Trình chủ sẽ trả kết quả tổ hợp về cho trình khách sau khi $SInclude đã diễn dịch và thực thi xong các servlet.



Hình 4-23 Cơ chế triệu gọi Servlet tổ hợp

6.1 Tìm hiểu thẻ <SERVLET>

Thẻ <SERVLET> đơn giản có cú pháp như sau

```
<SERVLET name=SERVLET_NAME
   code=SERVLET.CLASS
   codebase=SERVLET_CODE_BASE
   INIT_PARAM1=VALUE1
   INIT_PARAM2=VALUE2
   INIT_PARAMn=VALUE
    <param name=PARAM1 value =PARAM_VALUE1
     param name=PARAM2 value =PARAM_VALUE2
     param name=PARAMn value =PARAM_VALUEn>
</SERVLET>
```

Nếu bạn để ý sẽ thấy thẻ `<SERVLET>` phía máy chủ có cú pháp gần giống thẻ `<APPLET>` phía máy khách trong ngôn ngữ HTML. Trong thẻ `<SERVLET>` thuộc tính `name` là tên của servlet cần gọi, `code` là tên tập tin .class của servlet. `codebase` là đường dẫn đến nơi chứa tập tin .class (có thể là đường dẫn thư mục hay đường dẫn URL), `INIT_PARAMn` là các đối số khởi tạo cho servlet, `param name` là các tham số tùy chọn do người viết servlet định nghĩa (bạn có thể lấy về giá trị PRAM_VALUE bằng đối tượng HttpServletRequest).

Khi diễn dịch thể <SERVLET> bộ diễn dịch $SInclude (cũng là một servlet) sẽ tìm xem tên name của servlet có trong danh sách đăng ký hay không. Nếu có, servlet sẽ được gọi. Nếu không, $SInclude sẽ nạp và triệu gọi lớp servlet trong thuộc tính code dựa vào đường dẫn codebase.

6.2. Sử dụng SHTML

Dưới đây là một ví dụ đơn giản minh họa cách sử dụng thẻ <SERVLET>. Trang Web .SHTML của ta có nội dung như sau

Ví dụ 4-9 guideline.shtml

```
<!-- Header !-->
<SERVLET code=StandardHeader>
    <param name="title"  value =" Java Network
                                  Programming">
</SERVLET>
 <b>
 RMI Programming <br>
 CORBA Technique  <br>
 Servlet Server-side Programming <br>
 Java Server Page - JSP <br>
 Enterprise Java Bean <br>
<!-- Footer !-->
<SERVLET code=StandardFooter>
</SERVLET>
```

Nội dung của trang guidline.shtml chứa hai thẻ <SERVLET>. Thẻ <SERVLET> thứ nhất triệu gọi servlet mang tên StandardHeader sinh ra định dạng phần đầu tiêu đề cho tài liệu (header). Thẻ <SERVLET> thứ hai triệu gọi servlet mang tên StandardFooter sinh ra định dạng phần cuối trang cho tài liệu (footer). Ngoài hai thẻ <SERVLET> ra bạn có thể dùng bất kỳ thẻ HTML nào phục vụ cho mục đích định dạng riêng của mình. Trình dịch $SInclude chỉ quan tâm đến thẻ <SERVLET>, các thẻ HTML khác vẫn được giữ nguyên trong quá trình biên dịch. Bạn lưu tập tin guideline.shtml này vào thư mục $web_server\public_html của dịch

vụ Web (thay $web_server bằng thư mục C:\JavaWebserver2.0 hoặc thư mục mà bạn cài đặt JSW2.0)

Hai servlet StandardHeader và StandarFooter có mã nguồn cài đặt như sau

Ví dụ 4-10 StandardHeader.java

```
import javax.servlet.*;
  /**
  * Servlet dùng định dạng phần tiêu đề cho trang tài liệu
  */
public class StandardHeader extends GenericServlet
{
 public void service(ServletRequest req,
                     ServletResponse resp)
      throws ServletException, java.io.IOException
  {
          // Tạo luồng xuất
   java.io.PrintWriter out =
    new java.io.PrintWriter(resp.getOutputStream());
          // Lấy về nội dung tham số mang tên title
   String titles[] = req.getParameterValues("title");
   String title = "";
   if (titles != null) {
    if (titles.length > 0) {
     title = titles[0];
    }
   }
          // Xuất ra định dạng phần tiêu đề đầu trang
   out.println("<html>");
   out.println("<head>");
   out.println("<title>" + title + "</title>");
   out.println("</head>");
   out.println("<body>");
```

```
  out.println("<font size=\"4\"
                face=\"Arial\" color=\"red\">");
  out.println("<br><br>");
  out.println("<strong>" + title +
                          "</strong></font>");
  out.println("<br><hr><br>");
     // Kết thúc phần định dạng
  out.flush();
  out.close();
 }
public String getServletInfo() {
 return "Standard Header servlet";
}
/*
 *   Phương thức khởi tạo
 */
public void init(ServletConfig cfg)
                  throws ServletException
 {
  super.init(cfg);
 }
/*
 *   Phương thức hủy
 */
public void destroy()
 {
  super.destroy();
 }
}
```

Ví dụ 4-11 StandardFooter.java

```
import javax.servlet.*;
/**
 * Servlet dùng định dạng phần cuối trang của tài liệu
```

```
 */
public class StandardFooter extends GenericServlet
{
 public void service(ServletRequest req,
                     ServletResponse resp)
        throws ServletException, java.io.IOException
  {
    // Tạo luồng xuất
   java.io.PrintWriter out =
    new java.io.PrintWriter(resp.getOutputStream());
    // Xuất ra định dạng cuối trang cho tài liệu
   out.println("<br><hr>");
   out.println("
        <a href=\"mailto:minhkhai@bookstore.com\">");
   out.println("Book store");
   out.println("</a>");
   out.println("<font size=\"-1\">");
   out.println("<i>Questions or
                          Comments?</i></font>");
   out.println("</body>");
   out.println("</html>");
    // Kết thúc kết xuất
   out.flush();
   out.close();
  }
 public String getServletInfo() {
     return "Standard footer servlet ";
 }
/*
 *   Phương thức khởi tạo
 */
 public void init(ServletConfig cfg)
              throws ServletException
  {
   super.init(cfg);
  }
/*
```

```
 *   Phương thức hủy
 */
 public void destroy()
  {
   super.destroy();
  }
}
```

Ở ví dụ trên, servlet của ta không áp dụng cho lệnh HTTP GET hay POST cụ thể cho nên bạn không cần phải kế thừa lớp HttpServlet và cài đặt phương thức doPost() hay doGet(). Thay vào đó lớp servlet của ta kế thừa lớp servlet tổng quát hơn đó là GenericServlet. Khi cài đặt GenericServlet bạn chỉ cần xây dựng phương thức service() ghi kết xuất trả về cho trình chủ Web Server là đủ.

Lưu mã nguồn của hai tập tin StandardHeader.java và StandardFotter.java vào thư mục C:\JavaWebServer2.0\servlets. Chuyển vào thư mục này bạn đánh lệnh biên dịch

```
javac StandardHeader.java
javac StandardFooter.java
```

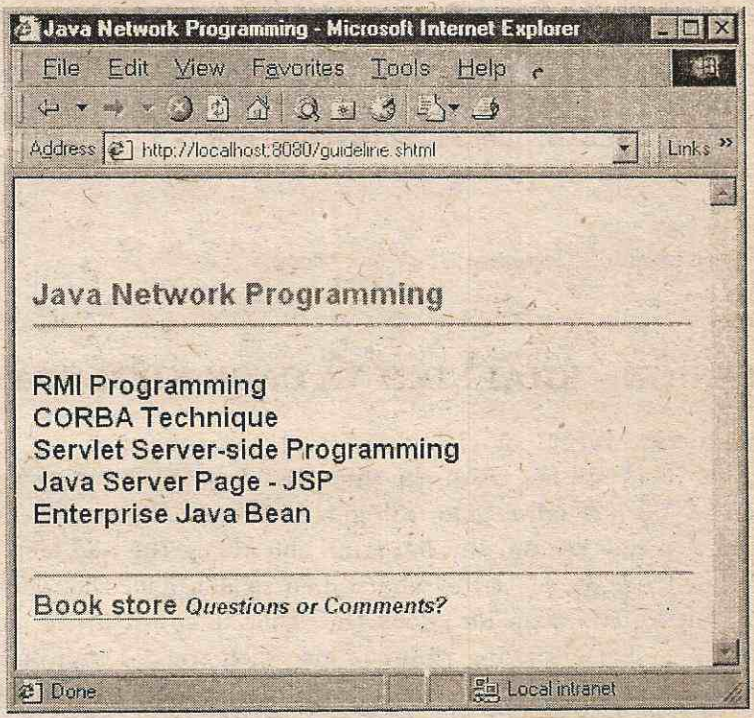
Đăng ký hai servlet này với dịch vụ Web của trình chủ. Bạn đặt tên cho servlet trùng tên với tập tin lớp: StandardHeader và StandardFooter.

Gõ địa chỉ sau để kiểm tra kết quả

http://localhost:8080/guideline.shtml

Kết quả chương trình hiển thị trên trình duyệt sau khi trình chủ Web diễn dịch trang Web guidline.shtml được thể hiện ở hình 4-24.

Bạn có thể chọn *View Source* từ trình duyệt IE để xem kết quả tổng hợp nhận về phía trình khách (ví dụ 4-12).



Hình 4-24 Kết xuất từ trang SHTML

Ví dụ 4-12 guidline.shtml kết xuất phía trình duyệt

```
<html>
<head>
<title> Java Network Programming</title>
</head>
<body>
<font size="4" face="Arial" color="red">
<br><br>
<strong> Java Network Programming</strong></font>
<br><hr><br>
   <b>
   RMI Programming <br>
   CORBA Technique  <br>
   Servlet Server-side Programming  <br>
```

```
    Java Server Page - JSP <br>
    Enterprise Java Bean <br>
   <!-- Footer !-->
   <br><hr>
<a href="mailto:minhkhai@bookstore.com">
    Book store
</a>
<font size="-1">
<i>Questions or Comments?</i></font>
</body>
</html>
```

4. SESSION – KHÁI NIỆM VÀ CÁCH KIỂM SOÁT

Giao thức HTTP là giao thức phi trạng thái. Mỗi lần trình khách gửi một lệnh lên trình chủ (như GET hay POST) và nhận kết quả trả về thì quá trình giao tiếp giữa khách chủ chấm dứt. Trong lần giao tiếp hay kết nối thứ hai trình chủ không thể nào biết được trạng thái trước đó của trình khách diễn ra đến đâu. Điều này đôi lúc gây bất lợi cho quan hệ khách chủ. Thực tế có rất nhiều trường hợp cần phải lưu giữ trạng thái của giao dịch trước đó. Ví dụ như: sau khi người dùng đăng nhập (login) với đúng username và password thì các lần truy xuất tiếp theo trình chủ phải nhớ được trạng thái login thành công trước đó để cho phép trình khách truy xuất các trang web kế tiếp. Hay như trong các ứng dụng mua bán hàng trên mạng, mỗi lần mở một trang Web giới thiệu mặt hàng nào đó khách hàng được phép chọn lấy mặt hàng mình thích lưu lại và đi tiếp sang các trang Web khác. Khi khách hàng đã hoàn tất quá trình lựa chọn, các mặt hàng mới được tổng kết để thanh toán (ứng dụng này thường gọi là shopping card).

Với servlet, bạn có thể quản lý trạng thái giao dịch của trình khách và trình chủ thông qua khái niệm session. Khi bạn mở trình duyệt browser và thực hiện kết nối với trình chủ trong lần đầu tiên, bạn đã tạo ra một session mới (còn gọi là phiên làm việc). Khi bạn đóng trình duyệt thì session cũng kết thúc. Mỗi session được trình chủ tạo ra và theo dõi bằng một số định danh (id) duy nhất. Nếu bạn

mở nhiều cửa sổ trình duyệt khác nhau, bạn sẽ có nhiều session tương ứng với số session id khác nhau. Trên trình chủ bạn có thể yêu cầu dịch vụ Web server tạo ra và lưu giữ các biến kiểm soát quá trình làm việc của session. Các biến này chỉ có giá trị riêng biệt trong từng session và sẽ bị hủy đi khi session chấm dứt.

7.1 Sử dụng biến session

Dưới đây là một ví dụ minh họa cho thấy cách làm việc và lưu giữ biến session. Chương trình servlet của ta là một bộ đếm (counter). Bộ đếm này sẽ đặt trị 0 khởi đầu. Nếu trong cùng một session bạn gọi trình chủ nhiều lần thì biến đếm counter sẽ tự động tăng lên. Lưu ý, bạn sẽ thấy mỗi session có một số id do trình chủ sinh ra khác nhau và do đó sự thay đổi của biến counter trong mỗi session cũng sẽ khác nhau.

Ví dụ 4-13 JCounter.java

```
import javax.servlet.*;
import javax.servlet.http.*;
    /**
     * Servlet JCounter này được dùng để theo dõi quá trình
 làm việc của
     *  một session thông qua biến đếm count
     */
public class JCounter extends HttpServlet
{
      // Định nghĩa tên biến mà session cần lưu giữ
 static final String COUNTER_KEY = "JCounter.count";
      /**
      * Thực hiện lệnh HTTP GET
      */
 public void doGet(HttpServletRequest req,
                   HttpServletResponse resp)
      throws ServletException, java.io.IOException
 {
```

```
        // Lấy về đối tượng session của trình chủ
        // Nếu đối số truyền cho getSession() bằng true,
session mới sẽ được
        // tạo ra nếu nó chưa có trước đó.
HttpSession session = req.getSession(true);
    // Định kiểu content-type cho dữ liệu trả về trình khách
resp.setContentType("text/html");
        // Lấy về luồng xuất
java.io.PrintWriter out = resp.getWriter();
    // Kiểm tra xem biến count đã tồn tại trước đó hay chưa
int count = 1;
Integer i = (Integer) session.getValue(COUNTER_KEY);
        //Nếu giá trị count đã được session lưu giữ trước đó
if (i != null) {
 count = i.intValue() + 1;   //Tăng trị count lên 1
}
        //Yêu cầu session lưu lại trị count mới
session.putValue(COUNTER_KEY, new Integer(count));
            // Kết xuất kết quả trả về cho trình khách
out.println("<html>");
out.println("<head>");
out.println("<title>Session Counter</title>");
out.println("</head>");
out.println("<body>");
        //In ra số định danh id tương ứng với session
out.println("Your session ID is <b>" +
                            session.getId());
        //In ra số đếm
out.println("</b> and you have hit this page <b>" +
                    count +
        "</b> time(s) during this browser session");
out.println("<form method=GET action=\"" +
                    req.getRequestURI() + "\">");
out.println("<input type=submit " +
                    "value=\"Hit page again\">");
out.println("</form>");
```

```
            // Kết thúc kết xuất
    out.println("</body>");
    out.println("</html>");
    out.flush();
  }
        /*
         * Phương thức khởi tạo
         */
 public void init(ServletConfig cfg)
                        throws ServletException
  {
   super.init(cfg);
  }
        /*
         * Phương thức khởi hủy
         */
 public void destroy()
  {
   super.destroy();
  }
}
```

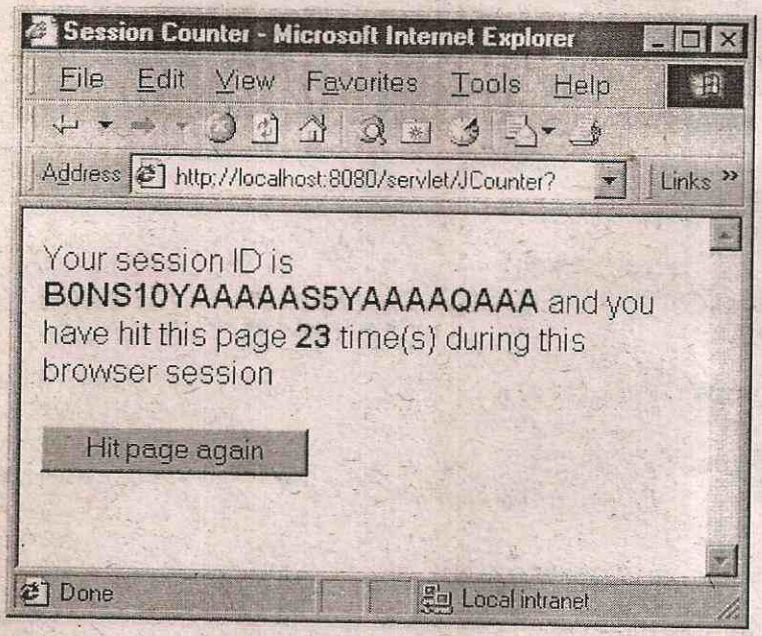
Lưu mã nguồn JCounter.java vào thư mục $web_server\servlets. Biên dịch JCounter

```
javac  JCounter.java
```

Đăng ký servlet JCounter.class với tên JCounter. Để kiểm tra và chạy chương trình bạn gõ địa chỉ sau vào trình duyệt

http://localhost:8080/servlet/JCounter

Kết quả kết xuất được hiển thị như hình 4-25.



Hình 4-25 Kết xuất của servlet JCounter

Như bạn thấy, số id của session là một chuỗi do trình chủ Web Server tự sinh ra (*B0NS10YAAAAAS5YAAAAQAAA*). Nếu bạn mở nhiều trình duyệt và gõ cùng một địa chỉ thì sẽ thấy số id của mỗi session hiện ra trên trang Web là khác nhau. Tương tự khi bạn nhấn nút *Hit page again* trên trang web thì biến count được tăng trị. Giá trị tăng của biến count ở mỗi session không phụ thuộc vào các session khác.

7.2. Kiểm soát và quản lý dữ liệu của session

Có 3 vấn đề bạn cần phải nhớ khi sử dụng biến trong mỗi session đó là: chuyển đổi session (swapping), phân bổ lại session (relocated) và lưu giá trị của session xuống các thiết bị lưu trữ để sử

dụng về sau (persisted). Bộ quản lý servlet cùng với dịch vụ Web của trình chủ sẽ làm công việc này giúp bạn. Bất kể, quản lý theo cách nào nếu biến đối tượng của bạn sử dụng trong mỗi session có cài đặt tính năng hay giao tiếp java.io.Serializable thì chúng sẽ được bảo toàn trong quá trình trình chủ thực hiện thao tác chuyển đổi, phân bổ hoặc lưu giá trị của session xuống đĩa cứng.

- *Chuyển đổi session*: Tất cả các bộ quản lý hay diễn dịch servlet (còn gọi là servlet engine) đều bị giới hạn về tài nguyên của hệ thống. Chính vì vậy tại một thời điểm chỉ có một số tài nguyên session được giữ thường trực trong bộ nhớ. Các kết nối hay session nào của trình khách ít sử dụng nhất sẽ được tạm thời lưu xuống đĩa. Trong quá trình lưu session xuống đĩa nếu biến đối tượng của bạn cài đặt tính năng Serializable thì toàn bộ đối tượng sẽ được lưu trữ dễ dàng. Nếu trình khách yêu cầu sử dụng một session mà sesion đó hiện không có trong bộ nhớ thì trình chủ sẽ khôi phục lại session từ vùng lưu trữ trên đĩa cứng. Trong trường hợp bạn sử dụng và khai báo biến bên trong session không có tính năng Serializable thì bộ quản lý servlet không có cách nào lưu được toàn bộ nội dung của session xuống vùng nhớ tạm trên đĩa. Session khi đó chỉ có cách giữ lại trong bộ nhớ. Bạn có thể xem lại kỹ thuật tuần tự hoá đối tượng (object serializable) trong chương 2.

- *Phân bổ lại session*: Trong đặc tả của Sun, không có yêu cầu nào đòi hỏi servlet phải được chạy trên một máy ảo Java duy nhất. Trên máy chủ server có nhiều kỹ thuật để tận dụng tối ưu tài nguyên của hệ thống trong đó có kỹ thuật chuyển đổi máy ảo để cân bằng tài nguyên. Ví dụ trường hợp có quá nhiều servlet chạy trên một máy ảo Java, trình chủ có quyền chuyển đổi servlet đang chạy trên máy ảo này sang máy ảo Java khác. Java tận dụng tối đa kỹ thuật RMI mà ta đã học trong chương 2. Các đối tượng của bạn muốn dễ dàng chuyển đổi giữa các máy ảo thì phải có tính năng Serializable. Khi cần phân bổ lại tài nguyên của

một session sang máy ảo Java khác, trình chủ luôn bảo đảm vấn đề đồng bộ hoá các biến session giúp bạn.

- *Lưu trữ session*: Máy chủ đôi khi cần khởi động lại để giải quyết một số sự cố (như thêm bộ nhớ, dung lượng lưu trữ, thay đổi cấu hình phần cứng ...). Trong trường hợp này bạn không thể hủy ngang mọi kết nối và trạng thái mà các session đang tồn tại. Một lần nữa nếu các biến mà bạn sử dụng trong session có tính năng Serializable trình chủ sẽ cho phép bạn lưu toàn bộ các session xuống đĩa cứng. Tất máy khởi động lại sau đó phục hồi các session về trạng thái trước đó.

Không có gì tồn tại mãi mãi, session cũng vậy bạn có thể qui định một khoảng thời gian nào đó session hết hiệu lực. Khi session hết hiệu lực trình chủ sẽ giải phóng toàn bộ tài nguyên mà session chiếm giữ. Nếu trình khách vẫn tiếp tục gửi yêu cầu đến một session đã hết hiệu lực, trình chủ sẽ tạo ra một session khác (với định danh id mới) để trình khác tái kết nối. Điều này rất dễ thấy nếu bạn sử dụng các chương trình gửi nhận thư (mail) trên web. Ví dụ Yahoo mail sau khi bạn login vào hộp thư sẽ cho phép session có hiệu lực trong vòng 8 tiếng. Sau 8 tiếng nếu bạnvẫn còn tiếp tục muốn đọc hay gửi mail bạn phải thực hiện đang nhập (login) lại.

Đối tượng HttpSession cung cấp cho bạn một số phương thức quản lý và lấy về thông tin của session như sau

`String getId()`	Trả về số chuỗi id là số duy nhất phân biệt session
`int getMaxInactiveInterval()`	Trả về thời gian cho phép session hiệu lực (tính bằng giây)
`int getCreationTime()`	Trả về thời điểm session được tạo ra (tính bằng khắc)

`int getLastAccessedTime()`	Trả về thời gian truy cập hiệu lực sau cùng nhất của session
`void setMaxinactiveInterval (int interval)`	Đặt lại thời gian hiệu lực của session
`void invalidate()`	Vô hiệu lực giá trị của sesion. yêu cầu trình chủ tạo lại session mới

Dưới đây là chương trình servlet ví dụ cho thấy một số thông tin về session mà bạn có thể sử dụng.

7.3. Khám phá session

SessionInfo là một servlet cung cấp thông tin về thời gian hiệu lực của một session. Chúng ta sử dụng các phương thức tiện ích mà đối tượng HttpSession cung cấp như getCreationTime(), getLastAccessTime() ... để lấy về thời gian liên quan đến session. Mã nguồn của servlet được trình bày ở ví dụ 4-14 sau đây

Ví dụ 4-14 SessionInfo.java

```
import javax.servlet.*;
import javax.servlet.http.*;
public class SessionInfo extends HttpServlet
{
        /*
         * Thực hiện lệnh HTTP GET
        */
 public void doGet(HttpServletRequest req,
                   HttpServletResponse resp)
         throws ServletException, java.io.IOException
  {
          // Định kiểu tài nguyên MIME
   resp.setContentType("text/html");
          // Lấy về luồng xuất
   java.io.PrintWriter out = resp.getWriter();
```

```
        //Kết xuất nội dung trang web trả về cho trình khách
out.println("<html>");
out.println("<head>");
out.println("<title>Servlet Session
                                    Explore</title>");
out.println("</head>");
out.println("<body>");
out.println("<h2>Servlet Session
                                  Explore</h2><br>");
out.println("<form method=POST action=\"" +
                    req.getRequestURI() + "\">");
        //Lấy về đối tượntg session
HttpSession session = req.getSession(true);
        //In ra định danh id của session
out.println("Session ID "+  session.getId()+"<br>");
        //In ra thời gian hiệu lực tối đa của session
out.println("Max time "
    +session.getMaxInactiveInterval()+"<br>");
    //In ra lần truy xuất hiệu lực của session sau cùng nhất
out.println("Last access "
    +session.getLastAccessedTime()+"<br>");
        //In ra thời điểm tạo lập session
out.println("Creation time "
                +session.getCreationTime() +"<br>");
        //In ra thời gian mà session đã hoạt động
long accesstime=(session.getLastAccessedTime()-
                    session.getCreationTime())/1000;
out.println("Access time "+ accesstime +"<br>");
//Tạo nút nhấn thực hiện lệnh POST vô hiệu lực session
out.println("<input type=submit " +
            "value=\"Invalid current session\">");
out.println("</form>");
        // Kết thúc kết xuất
```

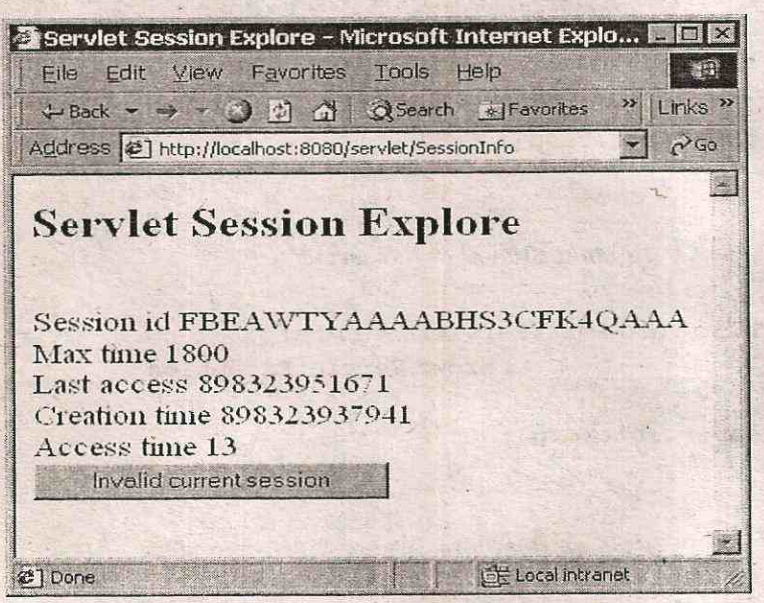
```
    out.println("</body>");
    out.println("</html>");
    out.flush();
  }
      /*
       * Thực hiện lệnh HTTP POST khi người dùng nhấn
nút submit trên
       * trang web
       */
public void doPost(HttpServletRequest req,
                   HttpServletResponse resp)
        throws ServletException, java.io.IOException
  {
          //Lấy về đối tượng session
    HttpSession session = req.getSession(true);
  //Vô hiệu lực đối tượng session hiện tại
    session.invalidate();
  //Gọi lại phương thức doGet() trả về kết xuất cho trinh khách
    doGet(req,resp);
  }
      /*
       * Phương thức khởi tạo servlet
       */
public void init(ServletConfig cfg)
                   throws ServletException
  {
    super.init(cfg);
  }
      /*
       * Phương thức hủy servlet
       */
public void destroy()
```

```
  {
   super.destroy();
  }
}
```

Biên dịch SessionInfo, đăng ký servlet SessionInfo với trình chủ. Bạn gõ địa chỉ sau để triệu gọi servlet

http://localhost:8080/servlet/SessionInfo

Kết quả trả về là thông tin liên quan đến thời gian hiệu lực của session (hình 4-26). Như bạn thấy, mặc định session có giá trị hiệu lực trong vòng 1800 giây (~ 30 phút). Bạn hãy nhấn nút *Refresh* trên trình duyệt IE. Khi trang web cập nhật lại thời gian *Last access* và *Access time* sẽ thay đổi. Khi thời gian *Access time* bằng với 1800, session của bạn sẽ bị mất tác dụng. Bạn có thể gọi session.setMaxinactiveInterval() để đặt lại thời gian mặc định cho session.



Hình 4-26 Thông tin về thời gian của session

Bạn cũng có thể nhấn nút *Invalid current session* trên trang Web khi đó lệnh POST sẽ gửi lên trình chủ. SessionInfo sẽ gọi `doPost()`. Phương thức `doPost()` gọi `session.invalidate()` để vô hiệu hóa session ngay tức thời mà không cần phải chờ thời gian qui định kết thúc. Trong ví dụ trên, khi ghép thuộc tính ACTION với thẻ <FORM> ta gọi hàm `req.getRequestURI()`. Hàm này đơn giản chỉ dùng trả về địa chỉ URL tương đối nơi phát sinh yêu cầu từ trình khách.

Qua ví dụ trên, bạn đã thấy session dùng để quản lý và phân định các phiên giao dịch của trình khách và trình chủ. Bạn có thể sử dụng session để lưu các biến theo dõi trạng thái của giao dịch (khắc phục nhược điểm phi trạng thái của giao thức HTTP). Để tạo ra một biến session đơn giản bạn chỉ cần cung cấp tên và giá trị của biến. Ví dụ

```
session.putValue("Counter", new Integer(13));
```

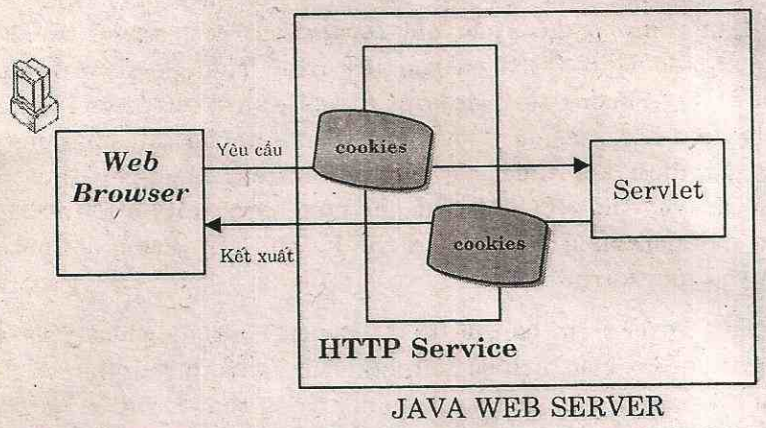
sẽ lưu biến tên Counter với giá trị là đối tượng số nguyên 13. Để lấy về giá trị của biến session bạn gọi

```
Interger i=session.getValue("Counter");
```

Session được xem là cách tốt nhất để lưu giữ trạng thái của giao dịch phía máy chủ.

8. TÌM HIỂU COOKIES

Nếu bạn muốn lưu trạng thái giao dịch phía máy khách thì sao? Trên nguyên tắc, trình khách không được phép ghi dữ liệu hoặc gọi thực thi các chương trình trên máy khách (điều này là tất nhiên và là một yêu cầu nghiêm ngặt của ứng dụng Internet. Bởi vì nếu cho phép các trang Web được thoải mái thao tác trên trình khách thì không loại trừ khả năng máy khách của bạn sẽ bị nhiễm virus hoặc bị tin tặc - hacker xâm nhập ăn cắp dữ liệu). Nhưng nếu bạn muốn lưu vết lại thông tin trên máy khách liệu có được không?



Hình 4-27 Truyền cookies giữa trình khách và trình chủ

Trình duyệt cho phép bạn thực hiện điều này thông qua kỹ thuật sử dụng cookies (bánh quy).

8.1. Tạo cookie lưu vết thông tin phía trình khách

Cookie là một mẫu thông tin. Đơn giản mẫu thông tin cookies gồm tên và giá trị thể hiện (tương tự biến). Trình duyệt cho phép bạn tạo ra các mẫu tin cookie và ghi nhớ xuống máy khách. Bạn có thể dùng ngôn ngữ kịch bản Java Script để tạo mẫu tin cookie. Cookie cũng có thể do trình chủ (cụ thể là servlet) tạo ra và yêu cầu trình duyệt lưu xuống máy khách.

Trình khách và trình chủ có thể gửi và nhận các mẫu tin cookie của nhau trong quá trình thực hiện các lệnh HTTP POST hay GET. Cơ chế truyền cookie giữa trình khách và trình chủ được minh họa ở hình 4-27.

Dưới đây là ví dụ cho thấy cách sử dụng cookie giữa máy khách (trình duyệt) và máy chủ (tương tác với servlet).

Ví dụ 4-15 cookieshow.html

```
<HTML>
Here is cookie at client<br>
```

```
<SCRIPT LANGUAGE=JAVASCRIPT>
 function getCookieVal(offset){
   var endstr=document.cookie.indexOf(";",offset);
   if (endstr==-1) endstr=document.cookie.length;
       return unescape(document.cookie.substring(
                                        offset,endstr));
 }
 function setCookie(name, value){
  document.cookie=name+"="+escape(value);
 }
```

//Ghi biến mang tên pagecolor mang trị yellow xuống máy khách

```
 setCookie("pagecolor","yellow");
```

//In toàn bộ cookies của máy khách ra trang Web

```
 document.write(document.cookie);
 </SCRIPT>
<FORM ACTION=/servlet/CookieListing METHOD=GET>
    <INPUT TYPE=Submit VALUE="Send cookies to server">
 </FORM>
</HTML>
```

Hình 4-28 Hiển thị cookies phía trình duyệt

Trang web của ta mang tên cookieshow.html. Trang web này sử dụng Java Script để tạo ra mẫu tin cookie mang tên "pagecolor" với giá trị "yellow". Như bạn thấy hàm setCookie() của ta rất đơn giản, setCookie() chỉ gọi đối tượng document của trang web để lưu trị và tên cookie

```
setCookie("pagecolor","yellow")
```

trang web Cookieshow.html cũng cung cấp cho bạn hàm getCookieVal() để đọc lại giá trị của cookie. Nếu đọc kỹ hàm getCookieVal() bạn sẽ thấy thật sự các mẫu tin cookie chỉ là các khối dữ liệu văn bản (text). Mỗi mẫu tin cookie phân cách nhau bằng dấu chấm phẩy (;). Hàm nội tại escape() của Java Script giúp bạn loại bỏ các ký tự đặc biệt trong cookie hoặc chuyển các ký tự đặc biệt thành dạng mã hoá khác.

Bạn chép tập tin cookieshow.html vào thư mục $web_server\public_html. Kiểm tra kết quả kết xuất bằng địa chỉ sau

http://localhost:8080/cookieshow.html

Hình 4-28 là nội dung mẫu tin cookie được hiển thị trên trình duyệt IE. Đừng ngạc nhiên, ngoài mẫu tin cookie "pagecoloe=yellow" bạn có thể nhận được một số cookie khác do trình duyệt tạo ra (ví dụ HTMLA=FONTSIZE=LARGE là một mẫu tin cookie do IE tự tạo trong quá trình hiển thị trang Web).

8.2. Gửi cookie lên trình chủ

Trên đây là cách mà các ứng dụng Web phía máy khách có thể tạo và yêu cầu trình duyệt lưu cookie xuống đĩa cứng. Mục đích của mẫu tin cookie là lưu vết thông tin giữa trình khách và trình chủ. Trong trang Web cookieshow.html ta đã thiết kế thẻ <FORM> gọi đến servlet mang tên CookieListing. Khi bạn nhấn nút "*Send cookies to server*" trình duyệt sẽ gửi yêu cầu triệu gọi servlet CookieListing trên máy chủ. Đồng thời toàn bộ cookie trên máy khách cũng sẽ được trình duyệt đóng gói chuyển đi. Servlet CookieListing của ta trên máy chủ được thiết kế để tiếp nhận cookie của trình khách như sau

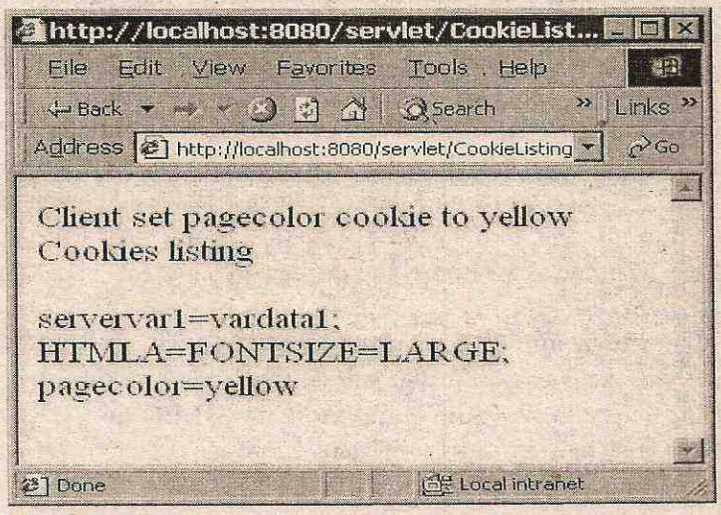
Ví dụ 4-16 CookieListing.java

```
import javax.servlet.*;
import javax.servlet.http.*;
public class CookieListing extends HttpServlet
{
        /*
         * Phương thức này duyệt qua danh sách các mẫu tin
 cookie và trả về
         * giá trị của một cookie (hay nội dung cookie) theo tên.
         */
 public String getCookieValue(Cookie cookies[],
                                        String name){
   for (int i =0; i< cookies.length; i++)
   {
       Cookie aCookie = cookies[i];
               //Tìm thấy cookie theo tên chỉ định
       if (aCookie.getName().equals(name))
                        return aCookie.getValue();
   }
   return null;
 }
       /*
        * Thực hiện lệnh HTTP GET
       */
 public void doGet(HttpServletRequest req,
                   HttpServletResponse resp)
           throws ServletException, java.io.IOException
  {
           // định kiểu tài nguyên MIME
   resp.setContentType("text/html");
       // Lấy về luồng xuất
   java.io.PrintWriter out = resp.getWriter();
```

```
        //Kết xuất nội dung trang Web
out.println("<html>");
        //Tiếp nhận danh sách cookie do trình khách gửi lên
Cookie[] cookies=req.getCookies();
        //Tìm giá trị của mẫu tin cookie mang tên pagecolor
String backcolor=getCookieValue(cookies,
                                "pagecolor");
if (backcolor==null) {
   out.println("Client do not have pagecolor
                                cookie <br>");
   backcolor="white";
} else{
   out.println("Client set pagecolor cookie
                        to "+backcolor+"<br>");
            // Tùy theo nhu cầu trình chủ có thể thêm vào
            // cookie mới trả về cho trình khách.
   resp.addCookie(new
                Cookie("servervar1","vardata1"));
}
        //Lấy giá trị cookie pagecolor của trình khách thiết
        // lập để làm màu nền cho trang kết xuất
out.println("<body bgcolor="+backcolor+">");
        //Yêu cầu trình khách sử dụng Java Script liệt kê
        // tất cả các cookies nhận về được từ trình chủ
out.println("Cookies listing <br><br>");
out.println("<SCRIPT LANGUAGE=JAVASCRIPT>");
out.println("document.write(document.cookie);");
out.println("</SCRIPT>");
            // Kết thúc kết xuất
out.println("</body>");
out.println("</html>");
out.flush();
}
```

```
        /*
        * Phương thức khởi tạo servlet
        */
public void init(ServletConfig cfg)
                    throws ServletException
  {
   super.init(cfg);
  }
        /*
        * Phương thức hủy servlet
        */
public void destroy()
  {
   super.destroy();
  }
}
```

Bạn biên dịch CookieListing.java. Đăng ký servlet này với trình chủ. Trở lại trang Web cookieshow.html bạn nhấn vào nút "*Send cookies to server*". Kết quả trả về được hiển thị ở hình 4-29.



Hình 4-29 Cookies trả về từ trình chủ

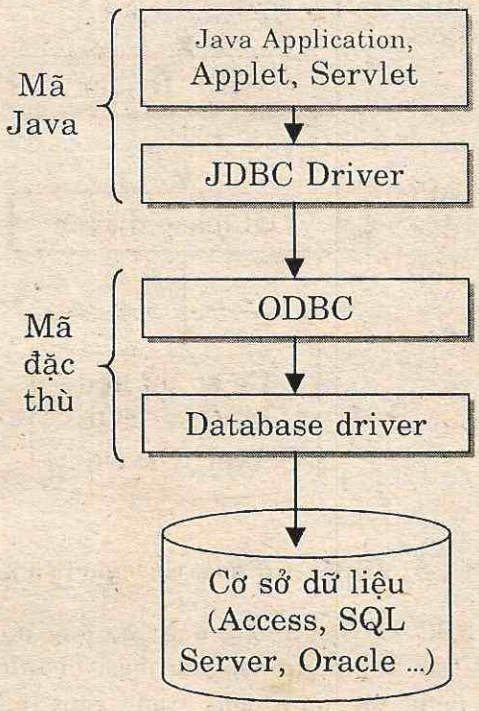
CookieListing của ta tiếp nhận được mẫu tin cookie mang tên "pagecolor=yellow" từ trình khách bằng cách gọi hàm getCookieValue(). Hàm này sử dụng danh sách các cookie do đối tượng req trả về để tìm ra cookie mang tên "pagecolor". Màu nền cho trang web phản hồi từ CookieListing được định màu như giá trị cookie pagecolor đã yêu cầu (màu vàng). Đồng thời, CookieListing của ta cũng tạo thêm một cookie mới mang tên servervar1 với giá trị vardata1 trả về cho trình khách. Phía trình khách, trang web của ta tạo mã Java Script gọi document.write(document.cookie); để in ra toàn bộ các cookie mang về từ trình chủ .

9. SỬ DỤNG JDBC TRUY XUẤT CƠ SỞ DỮ LIỆU VỚI SERVLET

Hầu như công việc lập trình mà servlet thường thực hiện nhất đó là lưu trữ thông tin và truy xuất cơ sở dữ liệu (database) cung cấp thông tin cho trình khách. Việc truy xuất cơ sở dữ liệu trong servlet dựa vào trình truy xuất JDBC mà Java hỗ trợ. JDBC là các trình đạo diễn cho phép bạn truy cập vào cơ sở dữ liệu của các hệ như Access, MS SQL Server, Oracle ... bằng ngôn ngữ truy vấn SQL. Theo đặc tả của Sun có 4 kiểu JDBC sau

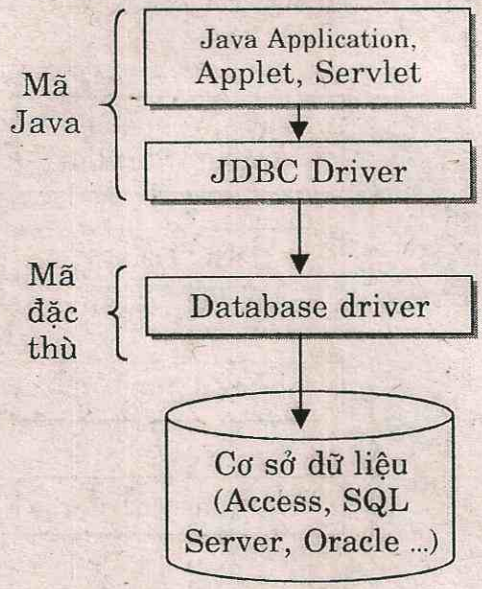
***Kiểu 1**: JDBC sử dụng cầu nối ODBC*. Nếu bạn đã từng làm việc với các hệ cơ sở dữ liệu trên Windows bạn sẽ biết đến cách kết nối ODBC (Open Database Connectivity). ODBC là cách kết nối tổng quát nhất đến mọi hệ cơ sở dữ liệu thông qua trình quản lý ODBC của Windows. Mỗi hệ cơ sở dữ liệu sẽ cung cấp một trình điều khiển (driver) có khả năng đọc hiểu được cơ sở dữ liệu của chúng. Trình điều khiển này sẽ được đăng ký với bộ quản lý ODBC. Tất cả các lệnh truy xuất cơ sở dữ liệu sau đó được thực thi bằng lệnh SQL. Java cung cấp cho bạn khả năng sử dụng JDBC để gọi ODBC và bằng cách này bạn có thể kiểm soát rất nhiều hệ quản trị cơ sở dữ liệu khác nhau. Quan hệ cầu nối JDBC-ODBC được thể hiện ở hình

4-30. Sun cung cấp sẵn thư viện kết nối JDBC-ODBC kèm theo bộ JDK và bạn có thể sử dụng miễn phí.



Hình 4-30 Truy xuất cơ sở dữ liệu JDBC-ODBC

***Kiểu 2**: JDBC kết nối trực tiếp với các trình điều khiển cơ sở dữ liệu*. Khi Java phát triển, các nhà cung cấp hỗ trợ các trình điều khiển JDBC kết nối trực tiếp với trình điều khiển đặc thù của mỗi hệ cơ sở dữ liệu (native database driver) mà không cần phải qua trung gian ODBC của Windows nữa. Các kết nối này nhanh và hiệu quả hơn kiểu 1 JDBC-ODBC. Tuy nhiên bạn phải mua các trình điều khiển cở sở dữ liệu đặc thù do những nhà phát triển cơ sở dữ liệu hay bên thứ 3 (third-party) cung cấp (hình 4-31).

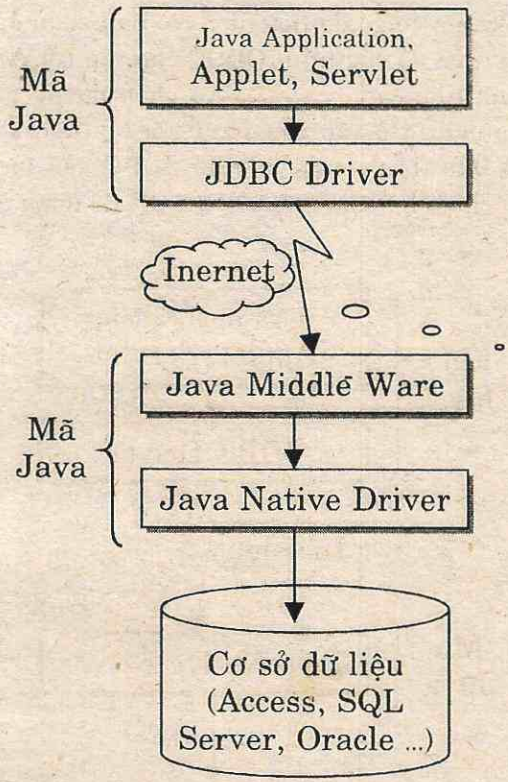


Hình 4-31 JDBC truy xuất cơ sở dữ liệu kiểu 2

***Kiểu 3**: JDBC kết nối thông qua các ứng dụng mạng trung gian*. Ngày nay, các ứng dụng đều gắn liền với Internet. Theo mô hình lập trình phân tán, ứng dụng trung gian (middle-ware) trên một máy chủ sẽ chịu trách nhiệm kết nối với cơ sở dữ liệu. Nhiệm vụ của trình khách là sử dụng những trình điều khiển JDBC gọn nhẹ có khả năng giao tiếp với ứng dụng trung gian qua Internet để truy vấn dữ liệu (hình 4-32).

***Kiểu 4**: JDBC kết nối thông qua các trình điều khiển đặc thù ở xa*. Mô hình kết nối theo kiểu 3 do phải qua ứng dụng trung gian (middle-ware) nên có thể ảnh hưởng đến tốc độ truy xuất cơ sở dữ liệu phía máy khách. Java đưa ra kết nối JDBC kiểu 4 cho phép trình khách sử dụng các trình điều khiển gọn nhẹ kết nối trực tiếp

vào trình điều khiển cơ sở dữ liệu đặc thù (thuần Java) trên máy chủ ở xa thông qua mạng Internet (hình 4-33).

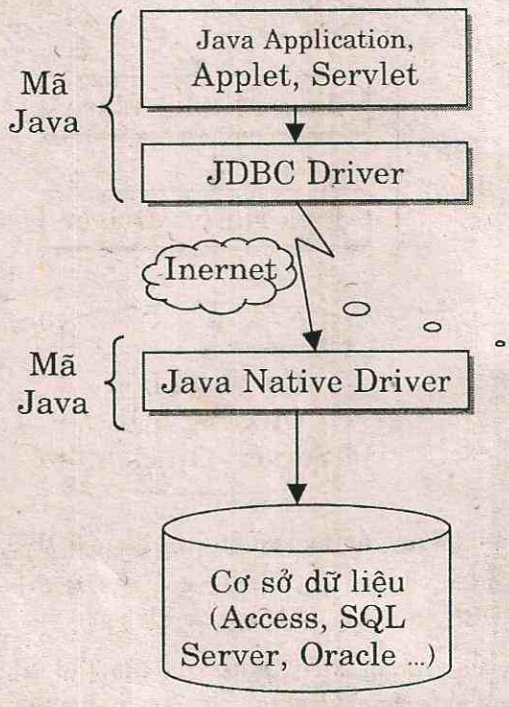


Hình 4-32 Truy xuất cơ sở dữ liệu bằng JDBC kiểu 3

Dưới đây là ví dụ về cách truy xuất dữ liệu bằng trình điều khiển JDBC-ODBC theo kiểu 1 trong servlet.

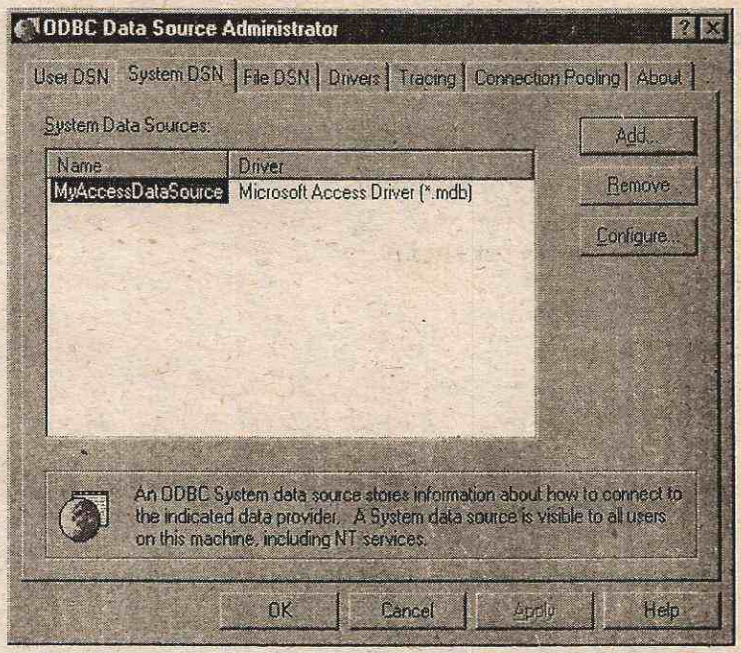
Trước hết bạn dùng MS Access để tạo tập tin cơ sở dữ liệu MyData.mdb với một bảng (table) dữ liệu mang tên Employee như hình 4-34.

Tiếp đến, để tạo kết nối ODBC bạn vào *Control Panel* của Windows chọn *ODBC Data Sources (32 bits)* (hình 4-35) . Chọn tab-System DSN và nhấn vào nút *Add* bạn sẽ thấy hiển thị danh sách các trình điều khiển cơ sở dữ liệu hiện có. Bạn chọn *Microsoft Access Driver (*.mdb)*. Cửa sổ cấu hình ODBC cho tập tin Access xuất hiện (hình 4-35). Bạn nhấn nút *Select* ... và chọn đường dẫn đến tập tin Access MyData.mdb. Gõ vào chuỗi kết nối MyAccessDataSource ở ô *Data Source Name* (hình 4-35). Trình JDBC sau này sẽ truy xuất ODBC dựa vào chuỗi kết nối trên. Nhấn OK để đóng cửa sổ cấu hình ODBC.

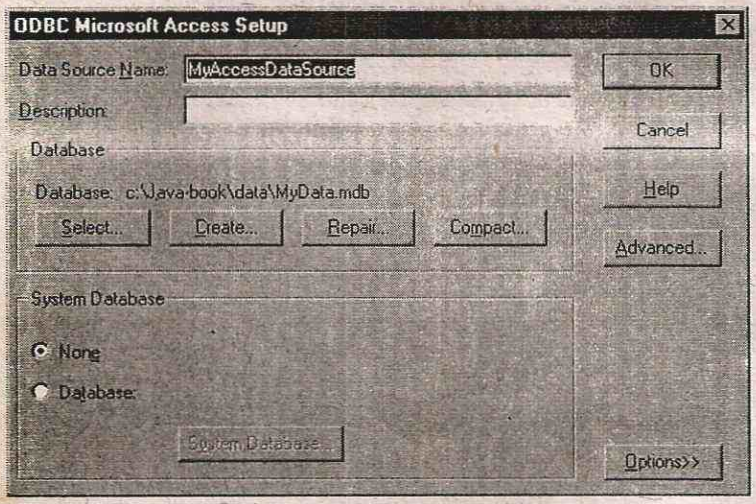


Hình 4-33 Truy xuất cơ sở dữ liệu bằng JDBC kiểu

Hình 4-34 Bảng dữ liệu Access



Hình 4-35 Cửa sổ cấu hình ODBC Data Sources



Hình 4-36 Cấu hình kết nối ODBC với tập tin .mdb của Access

Cách truy xuất bằng JDBC trong Java thông qua ODBC được thể hiện bằng mã nguồn của servlet EmmployList sau đây

Ví dụ 4-17 EmployeeList.java

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

    /**
     * Servlet này sẽ sử dụng JDBC-ODBC để trích dữ liệu
  từ bảng Employee trong cơ sở dữ liệu của Access. Dữ liệu lấy
  về được định dạng thành bảng HTML hiển thị phía trình
  duyệt của máy khách
    */
public class EmployeeList extends HttpServlet
{
    /**
```

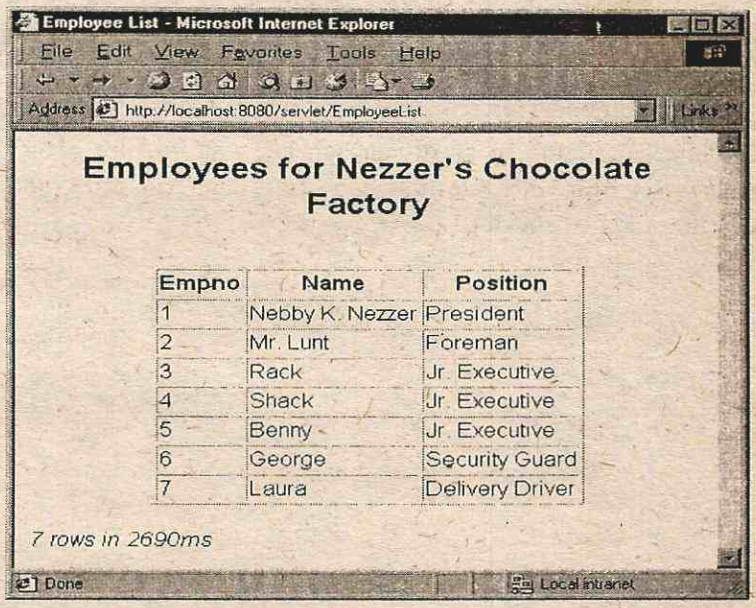
```
        * Thực hiện lệnh HTTP GET
        */
public void doGet(HttpServletRequest req,
                  HttpServletResponse resp)
        throws ServletException, java.io.IOException
 {
        // Định kiểu MIME cho tài nguyên trả về
  resp.setContentType("text/html");
        // Lấy về luồng xuất
  java.io.PrintWriter out =
   new java.io.PrintWriter(resp.getOutputStream());
        //Kết xuất định dạng HTML
  out.println("<html>");
  out.println("<head>");
  out.println("<title>Employee List</title>");
  out.println("</head>");
  out.println("<h2><center>");
  out.println("Employees for
                 Nezzer's Chocolate Factory");
  out.println("</center></h2>");
  out.println("<br>");
  java.util.Properties props =
          new java.util.Properties();
  props.put("user", "karlmoss");
  props.put("password", "larryboy");
  query("sun.jdbc.odbc.JdbcOdbcDriver",
     "jdbc:odbc:MyAccessDataSource",
     props,
     "SELECT Empno, Name, Position FROM Employee",
     out);
  out.println("</html>");
  out.flush();
  out.close();
 }
public void init(ServletConfig cfg)
             throws ServletException
 {
```

```
  super.init(cfg);
 }

public void destroy()
 {
  super.destroy();
 }

      /**
       * Truy vấn dữ liệu
       */
private boolean query(String driverName,
            String connectionURL,
            java.util.Properties props,
            String query,
            java.io.PrintWriter out)
  boolean rc = true;

        // Đối tượng kết nối JDBC
  Connection con = null;
     //Đối tượng thực hiện câu lệnh truy vấn
  Statement stmt = null;

        // Đối tượng nhận dữ liệu trả về
  ResultSet rs = null;

        // Đo thời gian truy vấn
  long startMS = System.currentTimeMillis();
  int rowCount = 0;
  try {

        // Tạo trình điều khiển
   Class.forName(driverName).newInstance();

        // Tạo kết nối
   con = DriverManager.getConnection(
                        connectionURL, props);

        // Tạo câu lệnh truy vấn
   stmt = con.createStatement();
```

```
        // Thực thi câu lệnh truy vấn và nhận kết quả trả về
    rs = stmt.executeQuery(query);

        // Định dạng bảng dữ liệu theo HTML
    rowCount = formatTable(rs, out);
  }
  catch (Exception ex) {

        // Bẫy lỗi
    out.println("Exception!");
    ex.printStackTrace(out);
    rc = false;
  }
  finally {
    try {

        //Cố gắng đóng kết nối ngay cả khi có lỗi xảy ra
      if (rs != null) {
        rs.close();
      }
      if (stmt != null) {
        stmt.close();
      }
      if (con != null) {
        con.close();
      }
    }
    catch (Exception ex) {

        // Các lỗi khác
    }
  }

        // Kết xuất thời gian hoàn thành câu truy vấn và số
dòng dữ liệu đọc được
  if (rc) {
    long elapsed = System.currentTimeMillis() - startMS;
    out.println("<br><i>" + rowCount + " rows in " +
          elapsed + "ms</i>");
  }
  return rc;
```

```
}
    /**
        * Định dạng bảng HTML dựa vào tập dữ liệu lưu
trong ResultSet
    */
private int formatTable(java.sql.ResultSet rs,
            java.io.PrintWriter out)
                        throws Exception
 {
  int rowCount = 0;
        // Tạo bảng hiển thị
  out.println("<center><table border>");
     //Trích dữ liệu
  ResultSetMetaData rsmd = rs.getMetaData();
  int columnCount = rsmd.getColumnCount();
        // Định dạng dòng tiêu đề
  out.println("<tr>");
  //
  for (int i = 0; i < columnCount; i++) {
    out.println("<th>" +
          rsmd.getColumnLabel(i + 1) +
         "</th>");
  }
  out.println("</tr>");
        //Định dạng các dòng dữ liệu
  while (rs.next()) {
   rowCount++;
   out.println("<tr>");
        //Định dạng từng cột
   for (int i = 0; i < columnCount; i++) {
    out.println("<td>" +
          rs.getString(i + 1) +
          "</td>");
   }
```

```
            // Hết dòng
    out.println("</tr>");
    }

            // Kết bảng
    out.println("</table></center>");
    return rowCount;
  }
}
```



Hình 4-37. Kết quả kết xuất từ truy vấn cơ sở dữ liệu

Biên dịch và đăng ký servlet EmployeeList với trình chủ. Bạn triệu gọi servlet này bằng địa chỉ URL sau

http://localhost:8080/servlet/EmployeeList

Kết quả hiển thị trên trình duyệt được trình bày ở hình 4-37.

10. KẾT CHƯƠNG

Chương này đã giới thiệu với bạn ý nghĩa và cách sử dụng servlet trong môi trường lập trình Java trên Web. Bạn cũng đã học

cách cấu hình trình chủ Java Web Server (JWS 2.0). Cách viết một servlet thông thường đáp ứng lệnh HTTP GET, POST từ trình khách. Chúng ta cũng đã xây dựng servlet liên hoàn là các servlet có khả năng xử lý tiếp kết xuất từ servlet khác. Trang SHTML là các trang HTML sử dụng thêm thẻ <SERVLET> để đưa servlet vào xử lý trên máy chủ ở dạng nhúng. Session là cách kiểm soát và quản lý kết nối phía trình chủ. Cookies là mẫu thông tin lưu vết phía trình khách. Sử dụng session và cookies là giải pháp thay thế tính phi trạng thái của giao thức HTTP trong giao tiếp giữa trình khách và trình chủ. Bạn cũng đã học cách sử dụng JDBC-ODBC để truy xuất cơ sở dữ liệu trong servlet.

Chương sau chúng ta sẽ đi sâu hơn vào một số tính năng nâng cao của servlet như tính bảo mật, giao tiếp giữa Applet-Servlet, gọi phương thức RMI từ servlet ...

Chương 5

SERVLET VÀ CÁC TÍNH NĂNG NÂNG CAO

Tóm tắt chương 5

Bạn đã biết cách viết servlet. Chương này sẽ giúp bạn tìm hiểu sâu hơn về servlet ở cấp độ truyền dữ liệu và bảo mật tài nguyên trên trình chủ. Chúng ta sẽ tìm hiểu về kỹ thuật xuyên hầm (tunneling) triệu gọi đối tượng ở xa theo giao thức HTTP và một số trường hợp áp dụng trong thực tế.

Các vấn đề chính sẽ được đề cập đến:

- ✓ *Triệu gọi servlet từ ứng dụng khách Java*
- ✓ *Servlet gọi các đối tượng RMI*
- ✓ *Giao tiếp với Applet ở phía khách và Servlet ở trình chủ*
- ✓ *Cách bảo vệ tài nguyên trên trình chủ, giới thiệu HTTPS và SSL.*

1. KỸ THUẬT TRIỆU GỌI SERVLET KHÔNG CẦN TRÌNH DUYỆT

Trong các ví dụ triệu gọi servlet ở chương trước ta vẫn phải dựa vào trình duyệt (browser) phía máy khách. Nếu bạn muốn viết một ứng dụng độc lập không chạy trên trình duyệt thì có cách nào triệu gọi servlet của một trình chủ Web server ở xa hay không? Chúng ta hoàn toàn có thể thực hiện được điều này. Servlet trên máy chủ chịu trách nhiệm tiếp nhận các yêu cầu phía máy khách gửi đến theo giao thức HTTP, kết quả kết xuất của servlet trả về cũng dùng giao thức HTTP. Nếu sử dụng trình duyệt trên máy khách, trình duyệt đã tự động định dạng lệnh HTTP như GET, POST hộ bạn. Nhiệm vụ của bạn chỉ là gõ địa chỉ URL tương ứng với servlet để triệu gọi servlet

trên máy chủ. Một khi không lợi dụng vào trình duyệt (browser) cũng đồng nghĩa với chúng ta phải tự làm lấy những công việc này.

TestDataStream dưới đây là một chương trình ví dụ triệu gọi servlet không cần dùng trình duyệt. Chương trình của ta tự mở ổ cắm socket kết nối với servlet theo địa chỉ URL chỉ định. Ghi các lệnh HTTP gửi vào luồng xuất của ổ cắm kết nối. Lấy dữ liệu phản hồi từ servlet mang tên DataStreamEcho (sẽ được chúng ta thiết kế sau) trên máy chủ và in ra màn hình của trình khách.

Ví dụ 5-1 TestDataStream.java

```
import java.io.*;
/**
 * Chương trình này minh hoạ cách dùng giao thức HTTP chuyển
dữ liệu
 * và lấy kết quả trả về từ trình chủ (cụ thể là servlet)
 */
public class TestDataStream
{
 public static void main(String args[])
  {
   // Đối số thứ nhất được dùng làm địa chỉ URL triệu gọi servlet
   if (args.length == 0) {
    System.out.println("\nServlet URL must
                                    be specified");
    return;
   }
   try {
    //Kết nối với servlet
    System.out.println("Attempting to connect to "
                                    + args[0]);
```

```
// Tạo đối tượng URL
java.net.URL url = new java.net.URL(args[0]);
// Mở socket kết nối với trình chủ
java.net.URLConnection con = url.openConnection();
// Khởi tạo kết nối
con.setUseCaches(false); //Không dùng cache đệm
con.setDoOutput(true);   //Khởi tạo luồng xuất
con.setDoInput(true);    //Khởi tạo luồng nhập
// Chúng ta thực hiện chuyển dữ liệu sang luồng ByteArray để
// dễ tính được chiều dài của dữ liệu cần chuyển đi
ByteArrayOutputStream byteOut = new
                    ByteArrayOutputStream();
// Tạo luồng xuất để gửi dữ liệu đi. Luồng xuất này lấy dữ
// liệu từ luồng lọc ByteArrayOutputStream
DataOutputStream out = new
            DataOutputStream(byteOut);
System.out.println("Writing test data");
// Chuyển dữ liệu ra luồng chuẩn bị gửi đi
out.writeBoolean(true);
out.writeByte(1);
out.writeChar(2);
out.writeShort(3);
out.writeInt(4);
out.writeFloat(5);
out.writeDouble(6);
out.writeUTF("Hello, Java");
//Chuyển dữ liệu hết vào bộ đệm
out.flush();
// Chuyển bộ đệm thành một mảng các byte nhị phân
byte buf[] = byteOut.toByteArray();
```

```
// Định dạng lệnh HTTP và kiểu giữ liệu chuyển đi là dữ liệu
nhị phân
con.setRequestProperty("Content-type",
            "application/octet-stream");
// Chỉ định chiều dài của vùng dữ liệu gửi đi
con.setRequestProperty("Content-length",
            "" + buf.length);
// Lấy về luồng xuất dữ liệu từ kết nối với servlet
DataOutputStream dataOut =
   new DataOutputStream(con.getOutputStream());
//Chuyển dữ liệu lên trình chủ cho servlet xử lý
dataOut.write(buf);
//Kết thúc kết xuất
dataOut.flush();
dataOut.close();
/* Chuẩn bị đọc kết quả trả về của servlet */
System.out.println("Reading response");
// Lấy về luồng nhập từ kết nối
DataInputStream in =
   new DataInputStream(con.getInputStream());
// Đọc kết quả trả về từ servlet
boolean booleanValue = in.readBoolean();
byte byteValue = in.readByte();
char charValue = in.readChar();
short shortValue = in.readShort();
int intValue = in.readInt();
float floatValue = in.readFloat();
double doubleValue = in.readDouble();
String stringValue = in.readUTF();
// Đóng luồng nhập
in.close();
```

```
        //In kết quả lấy về được
        System.out.println("Data read: " +
                booleanValue + " " +
                byteValue + " " +
                ((int) charValue) + " " +
                shortValue + " " +
                intValue + " " +
                floatValue + " " +
                doubleValue + " " +
                stringValue);
    }
    catch (Exception ex) {
     ex.printStackTrace();
    }
  }
}
```

Chương trình của ta mở socket bằng cách nhận đối số thứ nhất là địa chỉ URL trỏ đến servlet trên máy chủ

```
java.net.URL url = new java.net.URL(args[0]);
 // Mở socket kết nối với trình chủ
java.net.URLConnection con = url.openConnection();
```

Sau khi mở kết nối socket dữ liệu được chuyển lên servlet trên máy chủ bằng cách dùng luồng DataOutputStream. Ta không xuất luồng này trực tiếp ra socket mà tạm thời đưa vào bộ đệm ByteArrayOutputStream. Luồng DataOutputStream cho phép xuất ra các kiểu dữ liệu cơ bản như int, char, String ...

```
out.writeInt(4);
out.writeFloat(5);
out.writeDouble(6);
...
```

Sau khi luồng DataOutputStream đã hoàn tất quá trình xuất dữ liệu ra bộ đệm ta yêu cầu bộ đệm chuyển dữ liệu vào socket. Servlet trên máy chủ sẽ tiếp nhận dữ liệu và phản hồi về cho trình khách. Giai đoạn tiếp theo, trình khách sử dụng lớp DataInputStream đọc lại dữ liệu do trình chủ trả về. Dữ liệu sau đó được hiển thị ra màn hình.

```
int intValue = in.readInt();
float floatValue = in.readFloat();
double doubleValue = in.readDouble();
...
System.out.println("Data read: " + ...
```

Bạn lưu ý một điều, trước khi chuyển dữ liệu đi ta muốn trình chủ servlet hiểu được lệnh gửi đi là HTTP. Chúng ta định dạng phần header cho lệnh HTTP và chiều dài phần dữ liệu gửi đi như sau

```
// Định dạng lệnh HTTP và kiểu giữ liệu chuyển đi là dữ liệu
nhị phân
con.setRequestProperty("Content-type",
                        "application/octet-stream");
// Chỉ định chiều dài của vùng dữ liệu gửi đi
con.setRequestProperty("Content-length",
                                  "" + buf.length);
```

DataStreamEcho.java dưới đây là mã nguồn cài đặt cho servlet trên máy chủ

Ví dụ 5-2 DataStreamEcho.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
/**
 * servlet tiếp nhận dữ liệu gửi đến từ ứng dụng độc lập của trình
khách
```

```
 * bằng luồng xuất nhập DataOutputStream / DataInputStream
 */
public class DataStreamEcho extends HttpServlet
{
 public void service(HttpServletRequest req,
                     HttpServletResponse resp)
      throws ServletException, java.io.IOException
  {
   // Lấy về luồng nhập để đọc dữ liệu do trình khách gửi đến
   DataInputStream in =
      new DataInputStream(req.getInputStream());
   // Định dữ liệu trả về cho trình khách là dữ liệu nhị phân
   resp.setContentType("application/octet-stream");
   // Chúng ta sử dụng bộ đệm kết xuất dữ liệu trước khi gửi đi
   //Sử dụng luồng đệm sẽ cho biết chiều dài thực tế của dữ liệu
    //cần chuyển
   ByteArrayOutputStream byteOut = new
                        ByteArrayOutputStream();
   // Tạo luồng xuất đưa dữ liệu vào bộ đệm
   DataOutputStream out = new
                     DataOutputStream(byteOut);
   //Đọcdữ liệu do trình khách gửi đến
   boolean booleanValue = in.readBoolean();
   byte byteValue = in.readByte();
   char charValue = in.readChar();
   short shortValue = in.readShort();
   int intValue = in.readInt();
   float floatValue = in.readFloat();
   double doubleValue = in.readDouble();
   String stringValue = in.readUTF();
```

```
// Phản hồi - ghi dữ liệu nhận được từ trình khách ra luồng xuất
out.writeBoolean(booleanValue);
out.writeByte(byteValue);
out.writeChar(charValue);
out.writeShort(shortValue);
out.writeInt(intValue);
out.writeFloat(floatValue);
out.writeDouble(doubleValue);
out.writeUTF(stringValue);
// Yêu cầu luồng xuất chuyển dữ liệu ra bộ đệm
out.flush();
// Lấy về mảng nhị phân chứa dữ lliệu
byte[] buf = byteOut.toByteArray();
// định chiều dài dữ liệu cần gửi
resp.setContentLength(buf.length);
// Yêu cầu servlet lấy về luồng xuất chuẩn bị gửi dữ liệu
ServletOutputStream servletOut =
                    resp.getOutputStream();
// Gửi dữ liệu về cho trình khách
servletOut.write(buf);
servletOut.close();
}
}
```

Chạy chương trình

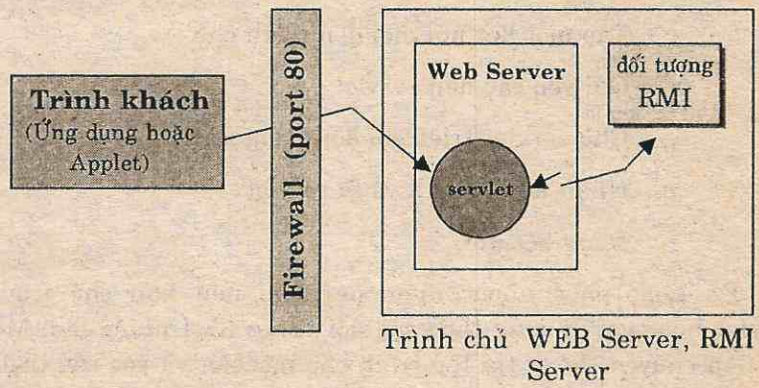
```
>java TestDataStream
    http://localhost:8080/servlet/DataStreamEcho
Attempting to connect to
    http://localhost:8080/servlet/DataStreamEcho
Writing test data
Reading response
Data read: true 1 2 3 4 5.0 6.0 Hello, Java
```

Thật sự ngoài lớp luồng DataOutputStream/DataInputStream ra bạn còn có thể sử dụng lớp ObjectOutputStream hoặc ObjectInputStream để thực hiện trao đổi nguyên một đối tượng giữa trình khách và servlet trên trình chủ. Đối tượng muốn chuyển tải đi trọn vẹn bắt buộc bạn phải cài đặt giao tiếp Serializable cho đối tượng. Cơ chế chuyển đối tượng giữa trình khách và trình chủ đã được chúng ta đề cập trong kỹ thuật RMI ở chương trước. Phần tiếp theo sau đây sẽ giới thiệu với bạn một kỹ thuật mới khá hấp dẫn. Kỹ thuật "xuyên hầm" (tunneling).

2. KỸ THUẬT XUYÊN HẦM (TUNNELING) VỚI GIAO THỨC HTTP

2.1. Khái niệm về kỹ thuật xuyên hầm (tunneling)

Trong chương 2 khi học về RMI chúng ta đã đề cập đến vấn đề tường lửa (firewall) và cổng kết nối (port) khiến cho các ứng dụng RMI hay ứng dụng phân tán đối tượng khó lòng triệu gọi được phương thức của nhau trực tiếp. Tuy nhiên servlet sẽ giúp bạn vượt qua trở ngại này.



Hình 5-1. Cơ chế triệu gọi đối tượng RMI thông qua servlet

Hình 5-1 trên đây là cơ chế triệu gọi đối tượng RMI từ trình khách thông qua đối tượng servlet trên máy chủ. Như bạn thấy mặc dù tường lửa cấm hầu hết các port kết nối từ trong ra ngoài nhưng port 80 (cổng của trình chủ Web) hầu như luôn luôn được mở. Cổng 80 thường dùng cho giao thức HTTP nhận và gửi yêu cầu về các trang Web. Do không gọi trực tiếp được các đối tượng RMI từ máy khách chúng ta phải nhờ đến servlet chạy trên máy chủ làm trung gian. Ta gửi yêu cầu theo giao thức HTTP đến trình chủ triệu gọi servlet. Servlet của ta tiếp đến sẽ thay mặt trình khách triệu gọi đối tượng RMI và nhận kết quả trả về cho trình khách nếu có.

Do sử dụng giao thức HTTP để đi xuyên qua tường lửa tương tác được với đối tượng ở xa cho nên kỹ thuật này còn được gọi là xuyên hầm (tunneling). Kỹ thuật xuyên hầm là một giải pháp khá hữu hiệu để áp dụng công nghệ lập trình phân tán đối tượng mà không bị cản trở bởi các nhà quản trị Internet.

Tuy nhiên, giao thức HTTP có nhược điểm đó là giao thức phi trạng thái. Chính vì vậy mỗi lần trình khách muốn triệu gọi một phương thức của đối tượng RMI thông qua servlet, trình khách phải thực hiện các bước sau

- Tạo một kết nối mới đến trình chủ
- Gửi yêu cầu đến servlet
- Chờ servlet triệu gọi đối tượng RMI
- Nhận kết quả trả về từ servlet
- Đóng kết nối.

Để khắc phục nhược điểm này bạn nên hạn chế truy xuất servlet hoặc gọi phương thức của đối tượng RMI nhiều lần. Mỗi lần gọi nên chuyển thông tin lên trình chủ một lần và yêu cầu trình chủ thực hiện càng nhiều tác vụ càng tốt trước khi trả kết quả về cho trình khách.

2.2. Xây dựng các lớp đối tượng thực hiện lời gọi xuyên hầm

Chúng ta sẽ xây dựng mô hình triệu gọi phương thức của tất cả các lớp đối tượng ở xa thông qua servlet trung gian một cách tổng quát nhất.

Thường khi triệu gọi một phương thức nào đó chúng ta phải cung cấp những thông tin sau: lớp đối tượng chứa đựng phương thức, tên phương thức gọi, đối số truyền cho phương thức, trị trả về của phương thức. Mô hình của ta sẽ hỗ trợ cả cho việc gọi đối tượng RMI và các đối tượng Java thông thường. Để đáp ứng yêu cầu này, trước tiên chúng ta xây dựng lớp MethodDesriptor với những thông tin mô tả cần dùng cho cả hai phía khách chủ như sau

Ví dụ 5-3 MethodDescriptor.java

```
import java.io.Serializable;
public class MethodDescriptor
                                    implements Serializable
{
  public String servletURL;        //Địa chỉ servlet cần gọi
  public String className;         //Tên lớp cần gọi
  public String methodName;        //Tên phương thức cần gọi
  public Object arglist[];         //Đối số truyền cho phương thức
  public Class parameterTypes[];   //Kiểu đối số
  public Object returnvalue;       //Trị trả về của phương thức
  public boolean isRMIcall=false;  //Phương thức gọi là RMI?
}
```

Mục đích của ta là sử dụng đối tượng MethodDescriptor chứa đựng thông tin chuyển đến trình chủ và nhận kết quả trả về sau khi trình chủ thực hiện xong phương thức cần gọi. Trình chủ là một servlet sẽ trích các thông tin trong MethodDescriptor, thực hiện triệu gọi phương thức của một lớp đối tượng nào đó (do trường

className qui định). Kết quả trả về sẽ được servlet đặt vào trường returnvalue và gửi MethodDescriptor trở về trình khách. Trường hợp bạn gọi phương thức của lớp đối tượng thông thường: className sẽ chứa tên lớp. Trường hợp bạn gọi phương thức của đối tượng RMI: className sẽ chứa địa chỉ tham chiếu đến tên đối tượng. Cờ isMethodcall cho servlet trung gian trên trình chủ biết phương thức mà trình khách cần gọi là của đối tượng RMI. Vì phương thức của ta được gọi thông qua servlet trung gian, trường servletURL trong MethodDescriptor sẽ cho biết địa chỉ tham chiếu đến servlet trung gian này.

MethodDescriptor phải di chuyển trên mạng cho nên bạn cần cho phép lớp đối tượng này có khả năng Serializable. Nội dung của đối tượng Serializable được Java cho phép mang đi và khôi phục lại ở bất kỳ máy ảo Java nào.

Để trình khách gọi được phương thức của đối tượng trên máy chủ thông qua giao thức HTTP ta xây dựng lớp TunnelCall hỗ trợ cho việc chuyển thông tin về phương thức gọi lên máy chủ và nhận lết quả trả về từ servlet như sau

Ví dụ 5-4 TunnelCall.java

```
import java.io.*;
/*
   Lớp hỗ trợ cho phép triệu gọi phương thức từ xa thông qua giao
thức HTTP
*/
public class TunnelCall
{
 MethodDescriptor md;   //Lưu thông tin về phương thức gọi
 public TunnelCall(MethodDescriptor md)
 {
   this.md=md;
 }
```

```
/*
    Trả về kết quả của phương thức cần gọi
*/
public Object getReturnValue(){
   return md.returnvalue;
}
/*
      Thực thi phương thức cần gọi
*/
public void execMethod() throws Exception
{
    System.out.println("Attempting to connect to "
                                        + md.servletURL);
    //Thực hiện kết nối với servlet trung gian trên máy chủ
    java.net.URL url = new java.net.URL(md.servletURL);
    // Mở socket kết nối
    java.net.URLConnection con = url.openConnection();
    // Khởi tạo luồng xuất nhập
    con.setUseCaches(false);
    con.setDoOutput(true);
    con.setDoInput(true);
    // Tạo luồng đệm
    ByteArrayOutputStream byteOut = new
                              ByteArrayOutputStream();
    // Tạo luồng xuất để gửi đối tượng lên trình chủ
    ObjectOutputStream out = new
                     ObjectOutputStream(byteOut);
    //Chỉ đối tượng MethodDescriptor ra luồng đệm
    System.out.println("Writing test data");
    out.writeObject(md);
```

```
    // Chuyển dữ liệu ra bộ đệm
    out.flush();
    // Lấy kích thước bộ đệm cần gửi đi
    byte buf[] = byteOut.toByteArray();
// Định dạng giap thức HTTP yêu cầu dữ liệu gửi đi dạng nhị phân
    con.setRequestProperty("Content-type",
                "application/octet-stream");
    // Chiều dài dữ liệu cần gửi
    con.setRequestProperty("Content-length",
                "" + buf.length);
    //Lấy về luồng xuất của socket
    DataOutputStream dataOut =
        new DataOutputStream(con.getOutputStream());
    //Gửi dữ liệu trong bộ đệm lên trình chủ
    dataOut.write(buf);
    dataOut.flush();
    dataOut.close();
    /* Nhận dữ liệu hay kết quả trả về từ servlet */
    System.out.println("Reading response");
    // Lấy về luồng nhập của socket
    ObjectInputStream in =
        new ObjectInputStream(con.getInputStream());
    //Đọc dữ liệu trả về
    md = (MethodDescriptor)in.readObject();
    System.out.println((String)md.returnvalue);
    // Đóng luồng
    in.close();
    System.out.println("call success");
  }
}
```

TunnelCall cung cấp cho bạn execMethod() để gọi phương thức qui định trong MethodDescriptor, getReturnValue() để nhận kết quả trả về từ phương thức gọi.

Để sử dụng TunnelCall đơn giản bạn chỉ cần tạo lớp MethodDescriptor, điền vào các thông tin cần thiết của phương thức muốn gọi sau đó triệu gọi TunnelCall.execMethod(). Phần sau ta sẽ viết chương trình kiểm tra các hoạt động của TunnelCall còn hiện tại ta phải thiết kết servlet trung gian trên trình chủ. Bạn hãy cài đặt ServletBroker theo mã nguồn của ví dụ sau

Ví dụ 5-4 ServletBroker.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.lang.reflect.*;
import java.rmi.*;
public class ServletBroker extends HttpServlet
{
 public void service(HttpServletRequest req,
                     HttpServletResponse resp)
          throws ServletException, java.io.IOException
  {
   try{
    // Lấy về luồng nhập để đọc dữ liệu từ trình khách gửi lên
   ObjectInputStream in =
      new ObjectInputStream(req.getInputStream());
  // Định dạng dữ liệu cần gửi về trình khách theo giao thức HTTP
   resp.setContentType("application/octet-stream");
  //Tạo luồng đệm để ghi dữ liệu
  ByteArrayOutputStream byteOut = new
                          ByteArrayOutputStream();
  // Tạo luồng xuất để ghi dữ liệu ra luồng đệm
```

```
ObjectOutputStream out = new
            ObjectOutputStream(byteOut);
 // Đọc dữ liệu là đối tượng gửi đến từ trình khách
MethodDescriptor md = (MethodDescriptor)
                                in.readObject();
 /*
Sau khi đã tiếp nhận được đối tượng MethodDescriptor mô tả
phương thức gọi do trình khách yêu cầu  công việc còn lại của
servlet là thay mặt trình khách triệu gọi phương thức của đối tượng.
Servlet sẽ sử dụng kỹ thuật "phản chiếu" (reflect) dựa vào tên và các
thông tin lớp để yêu cầu đối tượng bất kỳ thực thi phương thức gọi
*/
Object o;
Class c;
//Nếu trình khách triệu gọi đối tượng Java  thông thường
if (!md.isRMIcall){
    //Tạo thể hiện của đối tượng
   c=Class.forName(md.className);
   o= c.newInstance();
}
//Nếu trình khách cần gọi đối tượng RMI
else {
    //Lấy về tham chiếu của đối tượng RMI ở xa
   o=Naming.lookup(md.className);
   c = o.getClass();
}
 //Triệu gọi phương thức và nhận kết quả trả về
Method theMethod;
theMethod = c.getMethod(md.methodName,
                        md.parameterTypes);
```

```
    md.returnvalue=theMethod.invoke(o, md.arglist);
    System.out.println((String)md.returnvalue);
    // Ghi kết quả trả kết quả về cho trình khách ra bộ đệm
    out.writeObject(md);
    out.flush();
    // Tạo khối dữ liệu cần gửi về cho trình khách
    byte[] buf = byteOut.toByteArray();
    // Định chiều dài dữ liệu cần gửi về
    resp.setContentLength(buf.length);
    // Gửi dữ liệu về trình khách
    ServletOutputStream servletOut =
                        resp.getOutputStream();
    servletOut.write(buf);
    servletOut.close();
    }catch (Exception e){
       System.out.println(e);
    }
  }
}
```

ServletBroker sử dụng kỹ thuật reflect để gọi đối tượng bất kỳ trên trình chủ. Bạn có thể hình dùng ServletBroker thay mặt trình khách thực hiện phương thức gọi. Quá trình cài đặt các lớp cần thiết phía khách chủ đã hoàn tất. Bạn hãy biên dịch ServletBroker.java. Đăng ký ServletBroker với trình chủ. Dưới đây là ví dụ kết nối sử dụng lớp TunnelCall triệu gọi phương thức từ trình khách.

2.3. Triệu gọi phương thức của một lớp bất kỳ trên trình chủ thông qua ServletBroker

Trong ví dụ dưới đây, HelloProgrammer là một đối tượng trên máy chủ. Trình khách TunnelTesting không gọi phương thức sayWelcome() của HelloProgrammer một cách trực tiếp. Thay vào đó

TunnelTesting điền thông tin cần gọi vào đối tượng MethodDescriptor và nhờ ServletBroker triệu gọi.

Lớp HelloProgrammer trên trình chủ có nội dung như sau

Ví dụ 5-5 HelloProgrammer.java

```
class HelloProgrammer{
  public String sayWelcome(String name){
      return "Hi "+name;
  }
}
```

Chương trình TunnelTesting của trình khách được cài đặt như ví dụ 5-6 dưới đây

Ví dụ 5-6 TunnelTesting.java

```
import java.io.*;
public class TunnelTesting
{
 public static void main(String args[])
  {
    //Lấy đối số dòng lệnh thứ nhất làm địa chỉ trỏ đến
ServletBroker
   if (args.length == 0) {
    System.out.println("\nServlet URL must be
                                        specified");
    return;
   }
   try {
    System.out.println("Attempting to connect to "
                                        + args[0]);
    System.out.println("Call tunnel method");
     //Tạo đối tượng để lưu thông tin về phương thức cần gọi
    MethodDescriptor md=new MethodDescriptor();
```

```
      //Điền thông tin về phương thức cần gọi
      md.servletURL=args[0];
      md.className="HelloProgrammer";
      md.methodName="sayWelcome";
      md.parameterTypes=new Class[] {String.class};
      md.arglist=new Object[] {"Java Programmer"};
      md.isRMIcall=false;
      //Tạo đối tượng TunnelCall
      TunnelCall tn=new TunnelCall(md);
      //Yêu cầu triệu gọi phương thức
      tn.execMethod();
      //Nhận kết quả trả về
      String resultvalue=(String)tn.getReturnValue();
      System.out.println(resultvalue);
    }
    catch (Exception ex) {
       ex.printStackTrace();
    }
  }
}
```

Biên dịch chương trình

```
javac TunnelTesting.java
```

Chạy chương trình

```
java TunnelTesting
      http://localhost:8080/servlet/ServletBroker
```

Kết quả kết xuất

```
Call tunnel method
Attempting to connect to
      http://localhost:8080/servlet/ServletBroker
Writing test data
```

```
Reading response
Hi Java Programmer
call success
```

Lớp TunnelCall đã đóng gói hầu hết các thao tác xuất nhập và chuyển gửi dữ liệu lên servlet trung gian (ServletBroker) theo giao thức HTTP. Trình khách không cần phải thực hiện thao tác nặng nhọc này cũng như không cần biết phương thức được gọi ra sao. Trình khách chỉ cần kết quả trả về sau lời triệu gọi là đủ. Đây cũng chính là ý tưởng lập trình phân tán được mở rộng ra trong hầu hết các trình chủ Web Server ngày nay.

Dưới đây là cách triệu gọi đối tượng RMI thông qua servlet trung gian ServletBroker

2.4. Triệu gọi phương thức của đối tượng RMI thông qua ServletBroker

Trước hết chúng ta hãy xây dựng lớp đối tượng RMI `TheForeigner` đơn giản như sau

Ví dụ 5-7 TheForeigner.java

```
import java.rmi.*;
interface TheForeigner extends Remote{
    //Trả về thông tin của đối tượng TheForeigner
    public String aboutMe(String name)
                                    throws RemoteException;
}
```

Phần cài đặt cho đối tượng RMI TheForeigner được thực hiện như sau

Ví dụ 5-8 TheForeignerImpl.java

```
import java.rmi.*;
public class TheForeignerImpl  implements TheForeigner
{
```

```
  String nation;
  public TheForeignerImpl(String country){
     nation=country;
  }
  public String aboutMe(String name)
                                  throws RemoteException
  {
      return ("My name is "+name+ ", I am "+nation);
  }
}
```

Đơn giản TheForeignerImpl chỉ cài đặt phương thức aboutMe() bằng cách trả về lời chào tự giới thiệu của đối tượng. Tuy đơn giản nhưng bạn cũng sẽ thấy được cách truyền tham số và cách gọi hàm mà trình khách sẽ thực hiện sau này. Trình Setup dưới đây sẽ thực hiện cài đặt đối tượng RMI trước khi trình khách có thể triệu gọi

Ví dụ 5-9 Setup.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
public class Setup {
   public static void main(String args[])
                                  throws Exception
   {
      // Tạo đối tượng RMI
     TheForeigner obj=new
                    TheForeignerImpl("Vietnamese");
     //Tạo bộ đăng ký rmiregistry
    System.out.println("Create Registry")
    LocateRegistry.createRegistry(1099);
    //Đăng ký đối tượng RMI với máy ảo Java
   UnicastRemoteObject.exportObject(obj);
```

```
    System.out.println("Bind RMI object");
    System.out.println("Waiting for client
                                        request ...");
    //Ràng buộc đối tượng RMI với registry
    Naming.bind("rmi://localhost/vietnamese", obj);
  }
}
```

Bạn biên dịch các tập tin chương trình như sau

```
>javac TheFroreigner.java
>javac TheForeignerImpl.java
>rmic TheForeignerImpl
>javac Setup.java
```

Bạn chạy chương trình đăng ký đối tượng RMI trên một cửa sổ DOS-prompt riêng biệt

```
>java Setup.java
Create Registry
Bind RMI object
Waiting for client request ...
```

Trình khách TunnelRMITesting được viết để triệu gọi phương thức aboutMe() của TheForeigner thông qua ServletBroker trung gian được cài đặt như sau

Ví dụ 5-10 TunnelRMITesting.java

```
import java.io.*;
public class TunnelRMITesting
{
 public static void main(String args[])
  {
    // Sử dụng đối số thứ nhất của chương trình làm địa chỉ tham
chiếu
    //đến servlet trung gian ServletBroker
```

```
  if (args.length == 0) {
   System.out.println("\nServlet URL must be
   *                                        specified");
   return;
  }
   //Kết nối với ServletBroker
  try {
   System.out.println("Attempting to connect to "
                                          + args[0]);
   System.out.println("Call method tunnel");
    //Tạo đối tượng chứa thông tin về phương thức gọi
   MethodDescriptor md=new MethodDescriptor();
   md.servletURL=args[0];
    //Tham chiếu đến đối tượng RMI
   md.className="rmi://localhost/vietnamese";
   md.methodName="aboutMe";
   md.parameterTypes=new Class[] {String.class};
   md.arglist=new Object[] {"Minh Khai"};
    //Cho biết phương thức cần gọi thuộc về đối tượng RMI
   md.isRMIcall=true;
    // Tạo đối tượng TunnelCall
   TunnelCall tn=new TunnelCall(md);
    //Triệu gọi phương thức
   tn.execMethod();
   String returnvalue= (String)tn.getReturnValue();
   System.out.println(returnvalue);
  }
  catch (Exception ex) {
   ex.printStackTrace();
  }
 }
```

```
}
```

Bạn biên dịch chương trình

```
>java TunnelRMITesting.java
```

Chạy chương trình từ dòng lệnh như sau

```
>java TunnelRMITesting
  http://localhost:8080/servlet/ServletBroker
```

Kết quả kết xuất phía trình khách

```
Call method tunnel
At tempting to connect to
  http://localhost:8080/servlet/ServletBroker
Writing test data
Reading response
My name is Minh Khai, I am Vietnamese
call success
```

Như bạn thấy, để gọi được phương thức của đối tượng RMI ta chỉ cần thay tên lớp className bên trong MethodDescriptor bằng địa chỉ tham chiếu đến đối tượng

```
md.className="rmi://localhost/vietnamese";
```

đồng thời đưa thêm thông tin cho ServletBroker biết lời gọi phương thức thuộc về đối tượng RMI

```
md.isRMIcall=true;
```

Lớp TunnelCall thực sự mở ra cho bạn cách giao tiếp với tất cả các đối tượng trên trình chủ thông qua giao thức HTTP phổ biến. Không riêng gì ứng dụng trên trình khách, bạn hoàn toàn có thể sử dụng TunnelCall bên trong một Applet để thực hiện kết nối và triệu gọi servlet hay đối tượng RMI trên máy chủ. Dưới đây là cách thức trò chuyện giữa Applet và ServletBroker mà bạn có thể áp dụng.

3. GIAO TIẾP GIỮA APPLET VÀ SERVLET

Để giao tiếp và gọi đối tượng servlet trên máy chủ, Applet cũng sẽ thực hiện việc mở kết nối HTTP, chuyển dữ liệu lên servlet và nhận kết quả trả về. Lớp TunnalCall sẽ giúp bạn thực hiện công việc này bên trong Applet rất dễ dàng.

Ví dụ 5-11 TunnelApplet.java

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class TunnelApplet extends Applet
                                implements ActionListener
{
 String result="Please press button";
 public void init()
 {
   Button button=new Button("Call RMI Object");
   button.addActionListener(this);
   add(button);
 }
 public void paint(Graphics g){
   g.drawString(result,10,50);
 }
 /* xử lý tình huống khi người dùng nhấn nút button */
 public void actionPerformed(ActionEvent e){
    String connection="http://localhost:8080/
                              servlet/ServletBroker";
    try {
    System.out.println("Attempting to connect to "
                                        + connection);
    System.out.println("Applet call tunnel");
```

```
    MethodDescriptor md=new MethodDescriptor();
    md.servletURL=connection;
    md.className="rmi://localhost/vietnamese";
    md.methodName="aboutMe";
    md.parameterTypes=new Class[] {String.class};
    md.arglist=new Object[] {"Minh Khai"};
    md.isRMIcall=true;
    //Sử dụng TunnelCall để triệu gọi phương thức của đối tượng
RMI
    TunnelCall tn=new TunnelCall(md);
    tn.execMethod();
    result=(String)tn.getReturnValue();
    System.out.println( result);
    //Kết xuất kết quả ra cửa sổ Applet
    repaint();
   }
   catch (Exception ex) {
        ex.printStackTrace();
   }
 }
}
```

TunnelApplet của ta thực hiện cách thức gọi đối tượng RMI hoàn toàn tương tự chương trình TunnelRMITesting. Có điều thay vì thực hiện lời gọi trong main() Applet của ta tạo ra một nút nhấn. Khi người dùng kích chuột vào nút nhấn *"Call RMI Object"* trên Applet hàm xử lý actionPerformed() sẽ được kích hoạt và thực thi phương thức gọi.

Để kiểm tra TunnelApplet bạn biên dịch tập tin nguồn .java sang tập tin lớp. class

```
>javac TunnelApplet.java
```

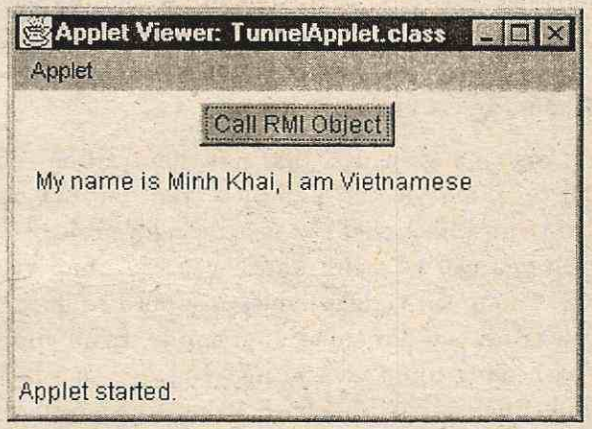
tiếp đến tạo trang Web để hiển thị Applet như sau

Ví dụ 5-12 AppletTest.html

```
<HTML>
 Call Applet Tunnel Demo <br>
 <APPLET code=TunnelApplet.class width=100 height=100>
 </APPLET>
</HTML>
```

Bạn dùng trình AppletViewer để hiển thị nội dung trang Web chứa applet (kết quả hiển thị được minh hoạ ở hình 5-2).

```
>appletviewer AppletTest.html
```



Hình 5-2 Applet triệu gọi đối tượng RMI thông qua giao thức HTTP

4. TƯƠNG TÁC KHÁCH/CHỦ VÀ VẤN ĐỀ BẢO MẬT (WEB SECURITY)

4.1. Chứng thực quyền truy cập trình chủ thông qua giao thức HTTP

Một trong những cách thông thường để bảo mật sự xâm nhập bất hợp lệ từ máy khách vào máy chủ là yêu cầu trình khách chứng thực quyền truy cập cho phép.

Thông thường khi trình khách yêu cầu một tài nguyên nào đó được bảo vệ bởi trình chủ, trình chủ sẽ gửi trả lại yêu cầu bắt trình khách hiển thị hộp thoại đòi người dùng nhập vào mật khẩu (password) và tên định danh dùng đăng nhập (username). Nếu người dùng phía trình khách nhập đúng thông tin xác thực quyền truy cập, trình chủ sẽ cho phép tài nguyên được sử dụng. Ngược lại quá trình đăng nhập và truy xuất trên trình chủ từ phía máy khách sẽ bị từ chối. Cơ chế này đòi hỏi sự hợp tác ở cả hai phía khách/chủ và thường được gọi là thử thách-trả lời (challenge-response).

Với giao thức HTTP việc truyền/gửi dữ liệu để xác thực quyền truy cập bằng mật mã (điều khiển bởi trình chủ) được thực hiện theo hai cách: cách gửi mật mã giản đơn (basic authentication) và cách gửi mật mã băm (digest authentication).

- Cách gửi mật mã giản đơn (basic authentication) được sử dụng một cách tự nhiên và khá phổ biến trên hầu hết các trình chủ Web. Khi trình khách yêu cầu một tài nguyên được bảo vệ, trình chủ sẽ gửi yêu cầu về trình khách yêu cầu nhập vào username, password. Thông tin về username, password sẽ được "mã hoá" và chuyển về trình chủ đối chiếu. Nếu hợp lệ, trình chủ sẽ cho phép trình khách đăng nhập vào tài nguyên. Việc "mã hóa" các thông tin về mật khẩu tại trình khách (trước khi gửi đi và thường do trình duyệt đảm nhận) sử dụng thuật toán mã hoá Base64. Thuật toán này không khó khăn gì cho các lập trình viên hay các tay tin tặc chịu khó giải mã. Chính vì vậy, cách truyền mật mã lên trình chủ theo kiểu này thường được xem như gửi mật mã "trơn" (clear password) mà hầu hết trên đường đi đến trình chủ, nếu thông tin bị chặn lại để giải mã thì hoàn toàn có khả năng mật khẩu sẽ bị lộ.
- Cách gửi mật mã băm (digest authentication) được phát triển để khắc phục yếu điểm của cách gửi mật mã giản đơn. Với cách này, mật mã do người dùng nhập vào không cần phải chuyển đi trên mạng (và do đó hạn chế khả năng mật khẩu bị đánh cắp). Khi

trình khách yêu cầu tài nguyên được bảo vệ bởi trình chủ, trình chủ sẽ tính toán một mã số đặc biệt và gửi trả về trình khách yêu cầu người dùng nhập vào username, password. Mật khẩu nhập vào sẽ được trình khách (thường là trình duyệt) tính toán kết hợp với mã số của trình chủ theo một thuật toán đặc biệt để sinh ra một mã số khác. Mã số do trình khách sinh ra được gửi ngược về trình chủ. Trình chủ tính toán mã số do trình khách gửi trả theo thuật toán băm đến đúng vị trí chứa password (mật khẩu) do trình chủ lưu trữ. Quyền truy cập vì thế sẽ được chứng thực. Cơ chế này không đảm bảo 100% an toàn trong quá trình trao đổi về thông tin xác thực quyền truy cập giữa khách/chủ. Tuy nhiên nó tốt hơn cách gửi mật mã giản đơn. Một điều đáng tiếc là hiện nay hầu như các trình duyệt phía máy khách chưa hỗ trợ mạnh tính năng này.

Dưới đây là một ví dụ về truy cập servlet có sự bảo vệ bởi trình chủ. HttpLogin servlet được dùng để in ra tên người dùng từ trình khách đã đăng nhập thành công vào trình chủ Java.

Ví dụ 5-13 HttpLogin.java

```
import javax.servlet.*;
import javax.servlet.http.*;
public class HttpLogin extends HttpServlet
{
 public void doGet(HttpServletRequest req,
                   HttpServletResponse resp)
        throws ServletException, java.io.IOException
  {
   // Định kiểu tài nguyên MIME trả về cho trình khách
   resp.setContentType("text/html");
   // Lấy về luồng xuất để ghi dữ liệu
   java.io.PrintWriter out = resp.getWriter();
```

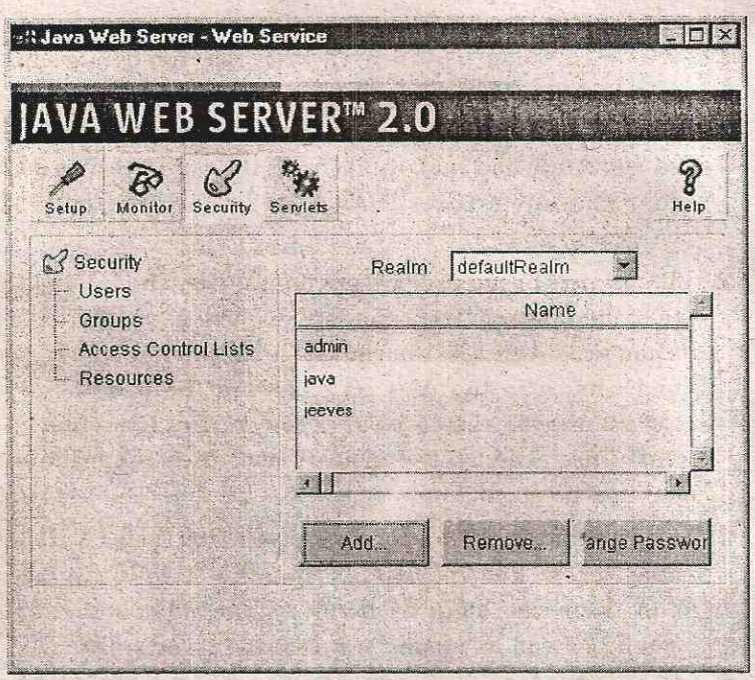
```
    // Yêu cầu trình khách không sử dụng cache để lưu giữ trang
kết xuất
    resp.setHeader("Expires",
                "Tues, 01 Jan 1980 00:00:00 GMT");
    // Lấy về tên người dùng đã đăng nhập
    String user = req.getRemoteUser();
    // Kết xuất nội dung trang web
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Welcome</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<center><h2>Welcome " + user +
          "</h2></center>");
    out.println("</body>");
    out.println("</html>");
    out.flush();
  }
 public void init(ServletConfig cfg)
                        throws ServletException
  {
   super.init(cfg);
  }
 public void destroy()
  {
   super.destroy();
  }
}
```

Bạn biên dịch và đăng ký HttpLogin như các servlet thông thường khác.

Để bảo vệ servlet HttpLogin chỉ cho một số người có mật khẩu đăng nhập bạn cần phải qua các bước cấu hình sau:

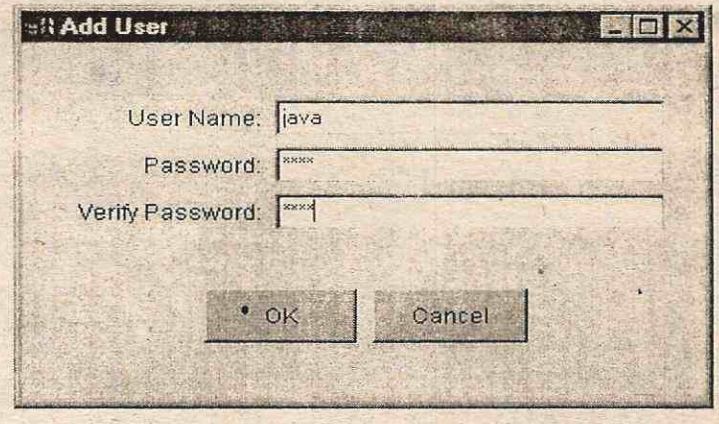
1. Bạn kích chuột vào dịch vụ Web Service (trang chủ quản trị trình chủ http://localhost:9090). Cửa sổ cấu hình xuất hiện bạn kích chuột vào nút nhấn *Security* để hiển thị cửa sổ cấu hình bảo mật (hình 5-3). Để cấu hình bảo mật bạn có 4 khái niệm *Users*, *Group*, *ACLs* (Access Control Lists) và *Resource*.
2. Trước tiên là tạo user (người dùng). Bạn chọn defaultRealm từ ô danh sách trên cùng. Tiếp đến chọn mục *Users* và nhấn nút *Add*. Trình chủ sẽ hiển thị hộp thoại yêu cầu bạn nhập vào tên user và password (hình 5-4). Chúng ta tạo ra user có tên là java, password tùy bạn chọn. Nếu có nhiều user bạn có thể quản lý user theo từng nhóm (chọn Group).
3. Kế tiếp là tạo danh sách cấp quyền truy xuất (ACLs). Chọn mục *Access Control Lists*. Trình chủ sẽ hiển thị cửa sổ để bạn thêm vào danh sách ACLs. Bạn nhấn nút *Add ACLs* sau đó gõ vào tên tượng trưng *servletsecure* (hình 5-5).
4. Với mỗi danh sách ACLs tương ứng bạn nhấn nút *Add Permision* .... Trình chủ sẽ hiển thị cửa sổ con để bạn cấp quyền. Bạn chọn tên java từ danh sách *Principal Name*. Chọn đánh dấu Get, Post từ ô *Pemissions*. Chúng ta cho phép Servlet thực hiện lệnh HTTP GET và POST (hình 5-6). Bạn có thể chọn cấu hình thêm cho servlet bằng cách chọn mục *Servlet* trên góc phải (hình 5-7). Đánh dấu vào ô *Permissions Load Servlet*.
5. Chỉ định tài nguyên mà danh sách điều khiển ACLs sẽ kiểm soát. Bạn chọn mục *Resources*. Trình chủ sẽ hiển thị cửa sổ cấu hình như hình 5-8. Bạn nhấn nút *Add* ... , trên cửa sổ chỉ định tài nguyên (hình 5-9) bạn chọn servletsecure từ ô danh sách *ACL*, chọn HttpLogin từ ô danh sách *Servlet*.

Quá trình cấu hình bảo vệ servlet đã hoàn tất.

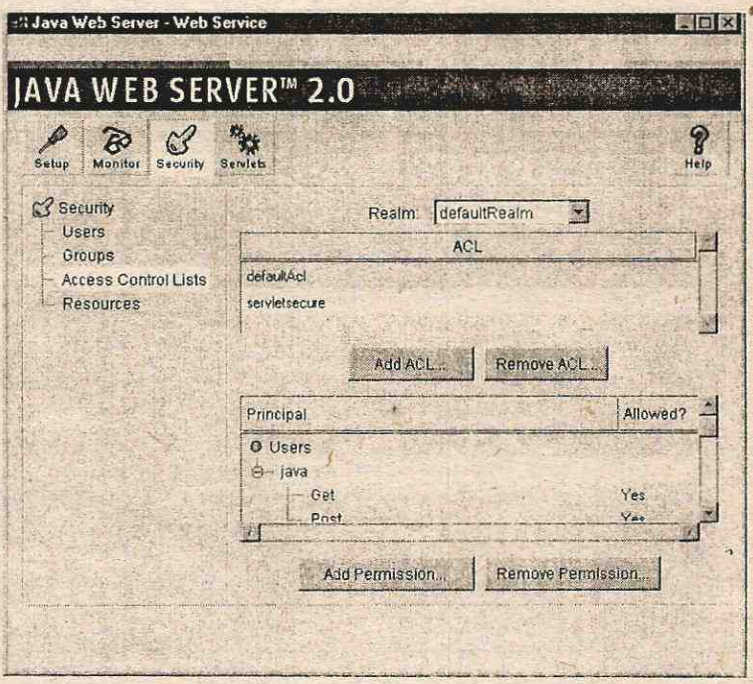


Hình 5-3 Tạo người dùng cho phép truy cập trình chủ

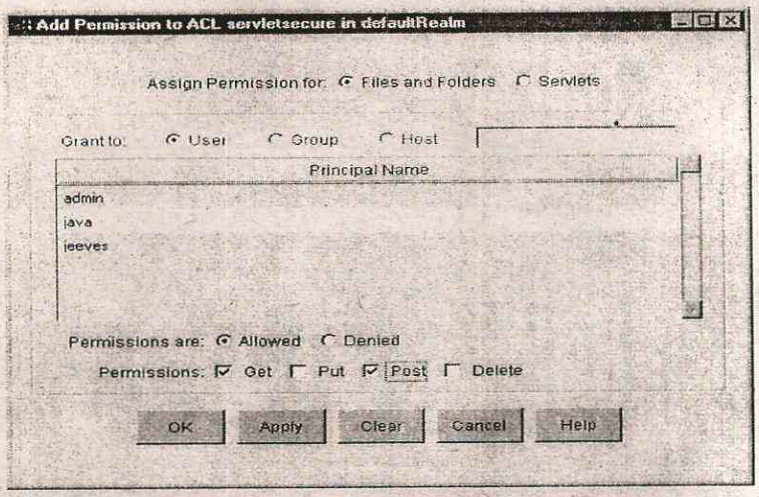
Bây giờ nếu bạn gõ địa chỉ triệu gọi http://localhost:8080/servlet/HttpServlet bạn sẽ bị trình duyệt (browser) chặn lại yêu cầu nhập username và password (hình 5-10). Nếu gõ đúng mật khẩu trình chủ sẽ tự động triệu gọi HttpLogin trả kết xuất của HttpLogin về cho trình khách. Nếu bạn gõ sai mật khẩu đăng nhập, trình chủ sẽ trả về trang Web thông báo lỗi tài nguyên bị cấm truy xuất.



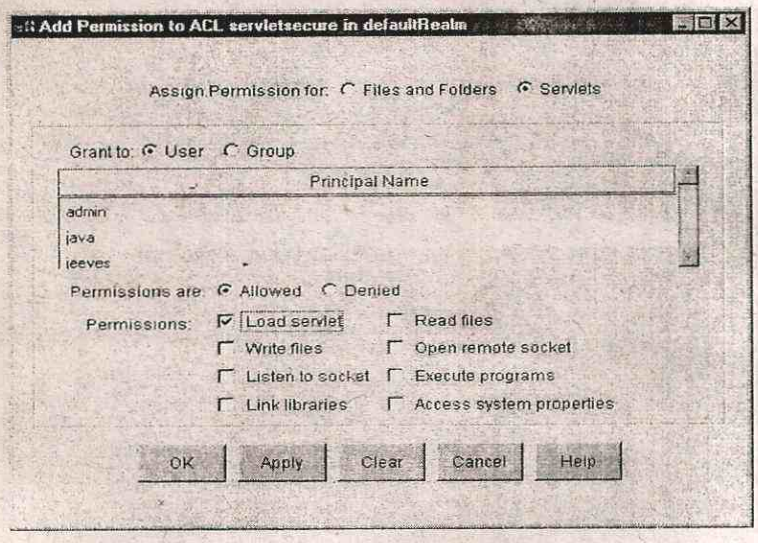
Hình 5-4 Tạo người dùng



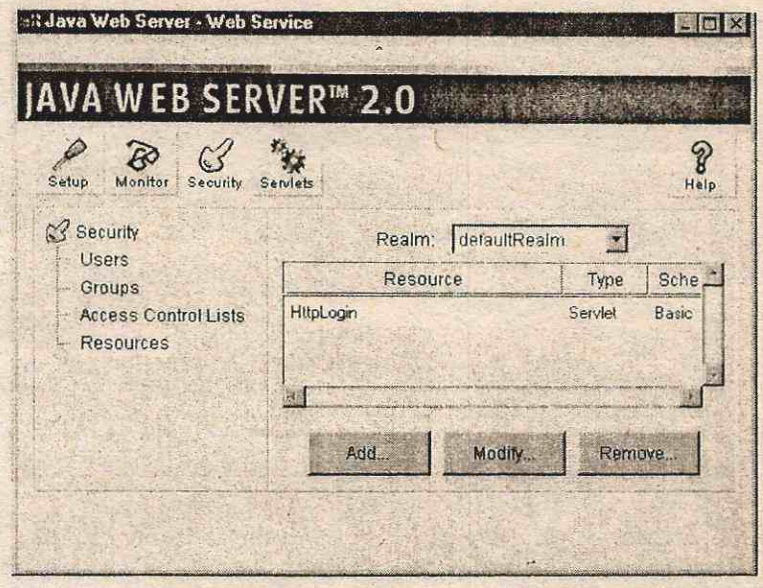
Hình 5-5 Cấp quyền truy cập ACLs



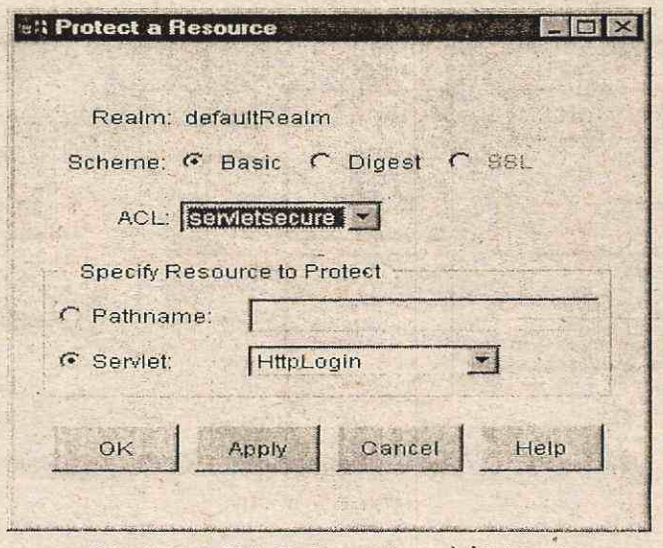
Hình 5-6 Cấp quyền triệu gọi HTTP



Hình 5-7 Cấp quyền cho servlet

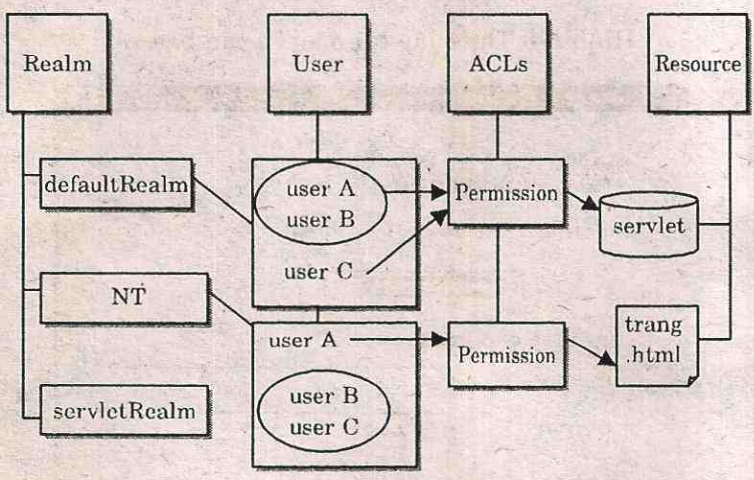


Hình 5-8 Thiết lập tài nguyên cần bảo vệ



Hình 5-9 Chỉ định servlet cần bảo vệ

Hình 5-10 Màn hình đăng nhập do trình duyệt (browser) hiển thị



Hình 5-11 Cơ chế phân quyền bảo mật tài nguyên trên trình chủ

Cơ chế kiểm soát trên trình chủ được minh hoạ cụ thể trên hình 5-11. Bạn quản lý cơ chế bảo mật thông qua khái niệm phạm vi (realm). Mỗi phạm vi bao gồm cơ chế kiểm soát và quản lý user độc lập. Trong quá trình cấu hình nếu bạn chọn phạm vi là NT, trình chủ sẽ cho phép bạn sử dụng ngay quyền đăng nhập của các user có sẵn trên hệ điều hành Windows NT/2000. Ở ví dụ trên ta chọn defaultReaml để tạo ra các user riêng của trình chủ Java Web Server. Với mỗi phạm vi realm bạn có một danh sách các người dùng (user). Các user có thể được tổ chức theo nhóm (group). Để truy xuất vào một tài nguyên (resource) ta tạo ra các quyền truy xuất. Những thông tin về quyền truy xuất được lưu riêng biệt thành từng mục trong một danh sách gọi là ACLs (danh sách kiểm soát quyền truy cập - Access Control Lists). Để chỉ định tài nguyên được bảo vệ (servlet hoặc trang web .html) bạn gán từng mục ACLs thích hợp cho tài nguyên. Bên cạnh đó các mục ACLs cũng sẽ được gán cho một user hoặc một nhóm (group) nào đó.

Khi người dùng đăng nhập vào hệ thống theo đúng mật khẩu, trình chủ sẽ dựa vào ACLs được gán cho user để xác định mức độ được phép sử dụng tài nguyên của trình khách.

Cách thức nhờ trình chủ bảo vệ tài nguyên tuy cấu hình hơi phức tạp nhưng lại là cách đơn giản và khá hiệu quả đối với người quản trị. Nhược điểm hơi khó chịu của phương pháp bảo vệ này là bạn không kiểm soát được quá trình tương tác phía máy khách. Trình duyệt phía máy khách luôn tự động hiện cửa sổ login như hình 5-10.

Các ứng dụng thương mại Web thường chọn cách đăng nhập và chứng thực quyền truy cập của người dùng thiết thực hơn. Đó là cách điều khiển thông qua lệnh POST và GET của giao thức HTTP.

4.2. Chứng thực quyền truy cập bằng trang HTML

Trình chủ sẽ thiết kế một trang Web đăng nhập yêu cầu người dùng nhập vào mật khẩu. Mật khẩu và định danh (username) của

người dùng sau đó được chuyển dưới dạng dữ liệu bình thường lên trình chủ bằng lệnh HTTP POST. Cách này cũng không an toàn tuyệt đối bởi mật khẩu của bạn chỉ được truyền đi ở dạng thô và không khó khăn gì cho người giải mã khi đón được dữ liệu trên đường đi. Tuy nhiên ưu điểm của phương pháp này đó là trang Web đăng nhập phía trình khách hoàn toàn có thể thiết kế theo giao diện Web bắt mắt, đồng thời phía trình chủ việc điều khiển và lưu thông tin đăng nhập của trình khách sẽ dễ dàng và tùy biến hơn (chẳng hạn trình chủ có thể chủ động kiểm tra mật khẩu trong cơ sở dữ liệu của riêng mình thay vì để dịch vụ Web thực hiện công việc này).

Ví dụ 5-14 ServletLogin.java

```
import javax.servlet.*;
import javax.servlet.http.*;
/**
 * servlet này sẽ tạo trang web HTML cho phép người dùng nhập vào mật
khẩu.
 * Nếu thông tin đăng nhập hợp lệ, servlet sẽ hiển thị thông báo login thành
 * công và cho phép trình khách truy cập tiếp vào các tài nguyên khác
 */
public class ServletLogin extends HttpServlet
{
 public static String USER_KEY = "ServletLogin.user";
 public static String FIELD_USER = "username";
 public static String FIELD_PASSWORD = "password";
public void doGet(HttpServletRequest req,
                  HttpServletResponse resp)
       throws ServletException, java.io.IOException
  {
    // Định kiểu tài nguyên MIME trả về cho trình khách
   resp.setContentType("text/html");
```

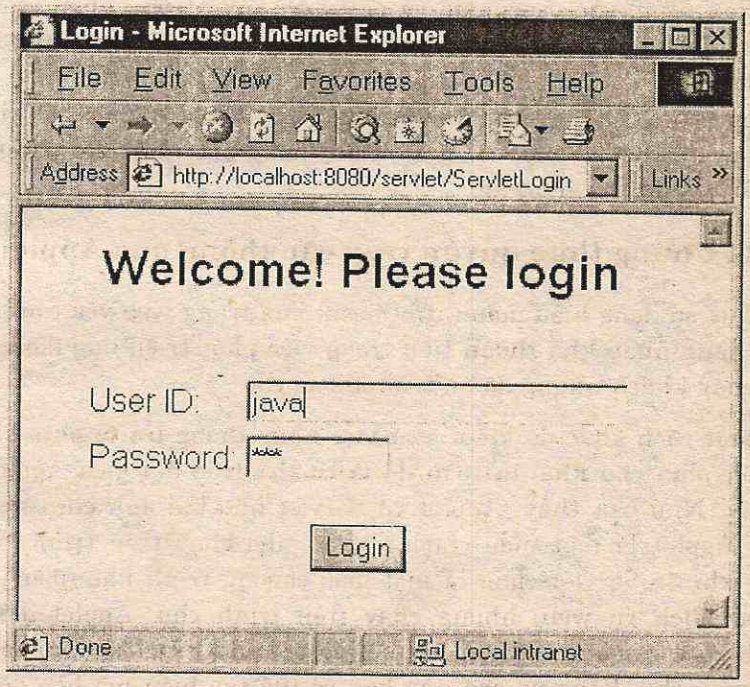
```
// Lấy về luồng xuất để ghi dữ liệu
java.io.PrintWriter out = resp.getWriter();
// Yêu cầu trình khách không sử dụng cache để lưu giữ trang
kết xuất
resp.setHeader("Expires",
                "Tues, 01 Jan 1980 00:00:00 GMT");
//Lấy về địa chỉ phát sinh yêu cầu từ trình khách
String uri = req.getRequestURI();
//Lấy về session hiện hành. Chúng ta lưu mật khẩu và tên người
//dùng trong biến session để xác thực rằng người dùng đã truy
cập hợp lệ
HttpSession session = req.getSession(true);
String user = (String) session.getValue(USER_KEY);
if (user == null) {
// Người dùng chưa đăng nhập. Hiển thị trang web (form) để người
//dùng login
 login(out, uri);
 return;
}
//Kết xuất trang Web khi người dùng đăng nhập thành công
trước đó
out.println("<html>");
out.println("<head>");
out.println("<title>Wecome</title>");
out.println("</head>");
out.println("<body>");
out.println("<center><h2>Welcome to our
                                site!</h2>");
out.println("<br>");
out.println("You have successful login...");
out.println("</center>");
```

```
    out.println("</body>");
    out.println("</html>");
    out.flush();
  }
  /**
   * Thực hiện lệnh  HTTP POST
  */
  public void doPost(HttpServletRequest req,
                     HttpServletResponse resp)
      throws ServletException, java.io.IOException
  {
    // Định kiểu tài nguyên MIME trả về cho trình khách
    resp.setContentType("text/html");
    // Lấy về luồng xuất để ghi dữ liệu
    java.io.PrintWriter out = resp.getWriter();
    // Nếu người dùng chưa đăng nhập – thực hiện việc kiểm tra
    HttpSession session = req.getSession(true);
    String user = (String) session.getValue(USER_KEY);
    if (user == null) {
      String username = req.getParameter(FIELD_USER);
      String password = req.getParameter(FIELD_PASSWORD);
      if (!validUser(username, password)) {
        //Hiển thị trang web thông báo người dùng đăng nhập không
hợp lệ
        out.println("<html>");
        out.println("<title>Invalid User</title>");
        out.println("<body><center><h2>
                        Invalid User!</h2><br>");
        out.println("Press the 'Back'
                            button to try again");
        out.println("</center></body></html>");
```

```
    out.flush();
    return;
   }
   // Người dùng đăng nhập thành công. Lưu trạng thái login vào
   // biến session
   session.putValue(USER_KEY, username);
  }
  // Định hướng người dùng vào trang web chứa tài nguyên được
  // truy xuất Ở đây ta cho phép người dùng gọi đến servlet
  // counter có nhiệm vụ hiển thị số lần trang Web được gọi
  resp.sendRedirect("http://localhost:8080/
                                    servlet/counter");
 }
 /**
  * Định dạng trang web yêu cầu người dùng login
  * @param out Luồng xuất
  * @param uri Địa chỉ trỏ đến servlet kiểm tra việc login
  *            (cũng chính là servlet này)
  */
 protected void login(java.io.PrintWriter out,
                                    String uri)
      throws java.io.IOException
 {
  out.println("<html>");
  out.println("<head>");
  out.println("<title>Login</title>");
  out.println("<center><h2>Welcome!
                        Please login</h2>");
  out.println("<br><form method=POST action=\"" +
              uri + "\">");
```

```
    out.println("<table>");
    out.println("<tr><td>User ID:</td>");
    out.println("<td><input type=text name=" +
                    FIELD_USER + " size=30></td></tr>");
    out.println("<tr><td>Password:</td>");
    out.println("<td><input type=password name=" +
                FIELD_PASSWORD + " size=10></td></tr>");
    out.println("</table><br>");
    out.println("<input type=submit value=\"Login\">");
    out.println("</form></center></body></html>");
  }
  /**
   * Kiểm tra username và password của người dùng
   * Trả về trị true nếu việc kiểm tra hợp lệ
   */
  protected boolean validUser(String username,
                              String password)
  {
    boolean valid = false;
      //Ở bước này bạn có thể sử dụng cơ chế truy tìm password và
      //username trong cơ sở dữ liệu. Tuy nhiên vì đơn giản ta sẽ
      // kiểm tra bằng cách qui định đơn giản là password và
username trùng nhau.
    if ((username != null) && (username.length() > 0)) {
     valid = username.equals(password);
    }
    return valid;
  }
  /* Phương thức khởi tạo servlet */
  public void init(ServletConfig cfg)
                    throws ServletException
```

```
    {
     super.init(cfg);
    }
  /* Phương thức hủy servlet */
  public void destroy()
    {
     super.destroy();
    }
}
```



Hình 5-12 Trang Web HTML dùng đăng nhập

Như bạn thấy ServletLogin cài đặt cả hai phương thức HTTP là doGet() và doPost(). Phương thức doGet() để hiển thị màn hình đăng

nhập cho phép người dùng điền vào thông tin. Phương thức doPost() dùng để kiểm tra và xác thực mật khẩu do người dùng gửi lên.

Bạn biên dịch và đăng ký ServletLogin với dịch vụ Web. Tiếp đến gõ địa chỉ triệu gọi servlet này như sau

http://localhost:8080/servlet/ServletLogin

Nếu lần đầu tiên bạn gõ địa chỉ này, phương thức doGet() của ServletLogin sẽ được gọi. Phương thức này sinh ra trang web đăng nhập gửi về trình khách và được trình duyệt hiển thị như hình 5-12.

Bạn nhập vào username và password (nhớ trùng nhau vì đó là điều ta qui ước với ServletLogin). Nhấn nút Login trên trang web. Phương thức doPost() được gọi để chứng thực thông tin nhập vào. Nếu quá trình chứng thực thành công bạn sẽ được hướng dẫn đi đến tài nguyên khác. Ngược lại màn hình thông báo nhập sai password sẽ hiển thị để bạn nhập lại.

4.3. Chứng thực quyền truy cập thông qua Applet

Tuy sử dụng biểu mẫu (html form) làm trang web yêu cầu người dùng đăng nhập khá thuận tiện trong việc phát triển ứng dụng web phía máy khách nhưng độ an toàn không cao.

Một cách khác để kiểm soát việc nhập thông tin từ phía khách gửi lên phía chủ khá an toàn đó là cơ chế trao đổi giữa Applet và Servlet. Như bạn thấy, Applet và Servlet như hai anh em cùng gia đình. Applet là một mẫu chương trình nhỏ chạy trên trình khách trong khi đó Servlet cũng là một mẫu chương trình nhỏ (được triệu gọi bởi dịch vụ web) nhưng chạy trên trình chủ. Applet ở trình khách có thể mở socket để kết nối và trao đổi trực tiếp với servlet trên máy chủ. Bằng cơ chế này bạn có thể tùy biến mã hóa dữ liệu theo cách riêng của mình ngay tại trình khách trước khi gửi lên servlet xử lý. Vì là cách mã hóa và gửi thông tin đi là do bạn chủ động điều khiển ở cả hai phía khác/chủ nên vấn đề bảo mật sẽ cao hơn.

Dưới đây là một ví dụ minh họa. LoginApplet là một Applet chạy trên trình khách giao tiếp với AppletLogin là một servlet chạy trên máy chủ. LoginApplet và AppletLogin sẽ kết hợp trao đổi mật khẩu, xác thực quyền đăng nhập của người dùng từ trình khách.

Ví dụ 5-15 LoginApplet.java

```
import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.*;
/*
Applet này được dùng để thu thập thông tin về định danh đăng
nhập (username) và mật khẩu (password) do người dùng nhập vào.
Những thông tin này sau đó được chuyển cho servlet trên máy chủ
xử lý. Servlet sẽ xác thực tính hiệu lực.. Nếu hợp lệ servlet sử gửi
thông tin về cho Applet hướng dẫn cách truy cập vào tài nguyên được
cho phép của trình chủ.
*/
public class LoginApplet extends Applet
{
 // Định nghĩa các thành phần giao diện
 TextField username;
 TextField password;
 Label message;
 Button login;
 String codeBase;
 String servlet;
 String nextDoc;
 String sessionId;
 /**
  * Khởi tạo applet
  */
```

```
public void init()
  {
    //Lấy  địa chỉ tham chiếu đến servlet trên máy chủ từ tham số
codebase
    codeBase = "" + getCodeBase();
    servlet = getParameter("servlet");
    // phân tích chuỗi codebase chứa địa chỉ servlet.
    // Loại bỏ ký tự '/' của chuỗi nếu có
    if (servlet != null) {
     if (servlet.startsWith("/") &&
                 codeBase.endsWith("/")) {
      codeBase = codeBase.substring(0,
                        codeBase.length() - 1);
     }
    }
    // Lấy thông tin về session id do phương thức servlet sinh ra
trước đó
    sessionId = getParameter("id");
    // Định màu nền cho Applet
    setBackground(Color.white);
    // Định cách bố trí các thành phần giao diện
    setLayout(new BorderLayout(0, 5));
    // Đưa các thành phần giao diện vào applet
    message = new Label();
    add("North", message);
    Panel p = new Panel();
    p.setLayout(new GridLayout(2, 2, 30, 20));
    p.add(new Label("Enter user name:"));
    username = new TextField(10);
    p.add(username);
    p.add(new Label("Enter password:"));
```

```
    password = new TextField(10);
    password.setEchoCharacter('*');
    p.add(password);
    add("Center", p);
    // Tạo nút nhấn login (tương tự nút submit của trang web .html)
    login = new Button("Login");
    add("South", login);
  }
  /**
   * Xử lý tình huống khi người dùng kích chuột vào nút nhấn login
   */
  public boolean handleEvent(Event event)
  {
    if ((event != null) && (event.id ==
                        event.ACTION_EVENT)) {
      if (event.target == login) {
        message.setText("");
        // Lấy về nội dung username và password
        String user = username.getText();
        String pw = password.getText();
    // Lưu ý: Bạn có thể mã hóa password trước khi chuyển đi ở đây
    // Xác định xem username và password có hợp lệ hay không
        boolean valid = false;
        try {
         valid = validate(user, pw);
        }
        catch (Exception ex) {
         ex.printStackTrace();
        }
        // Hiển thị thông báo login không hợp lệ
        if (!valid) {
```

```
        message.setText("Invalid user -
                        please try again");
      }
      else {
        // Đăng nhập thành công
        try {
          getAppletContext().showDocument(new
                                URL(nextDoc));
        }
        catch (Exception ex) {
          message.setText("Invalid document: " + nextDoc);
          ex.printStackTrace();
        }
      }
    }
  }
  return false;
}
/**
 * Hàm này được dùng để xác thực tính hợp lệ của thông tin đăng
 * nhập applet sẽ kết nối và chuyển dữ liệu trực tiếp cho servlet
 */
protected boolean validate(String user, String pw)
                                throws Exception
{
  boolean valid = false;
  // Lấy về địa chỉ URL tham chiếu đến servlet
  java.net.URL url = new java.net.URL(codeBase +
                                servlet);
  // Kết nối với servlet
  java.net.URLConnection con = url.openConnection();
```

```
// Khởi tạo kết nối
con.setUseCaches(false);
con.setDoOutput(true);
con.setDoInput(true);
// Tạo luồng đệm
ByteArrayOutputStream byteOut = new
                    ByteArrayOutputStream();
// Tạo luồng xuất để gửi dữ liệu
DataOutputStream out = new
                    DataOutputStream(byteOut);
// Gửi trả session id về máy khách
out.writeUTF(sessionId);
// Gửi định danh và mật khẩu lên trình chủ
out.writeUTF(user);
out.writeUTF(pw);
// Chuyển dữ liệu xuất ra bộ đệm
out.flush();
//Lấy về khối dữ liệu chứa trong bộ đệm
byte buf[] = byteOut.toByteArray();
// Định dạng dữ liệu gửi theo giao thức HTTP
con.setRequestProperty("Content-type",
            "application/octet-stream");
// Định kích thước dữ liệu cần gửi đi
con.setRequestProperty("Content-length",
            "" + buf.length);
// Lấy về luồng xuất từ socket và gửi dữ liệu lên trình chủ
DataOutputStream dataOut =
    new DataOutputStream(con.getOutputStream());
dataOut.write(buf);
// Đóng kết nối từ luồng xuất
```

```
    dataOut.flush();
    dataOut.close();
    // Lấy về luồng nhập từ socket để đọc kết quả trả về từ servlet
    DataInputStream in =
      new DataInputStream(con.getInputStream());
    // Đọc kết quả cho biết quá trình kiểm tra đăng nhập hiệu lực
hay không
    valid = in.readBoolean();
  // Nếu người dùng đăng nhập thành công. Đọc thông tin cho phép
  //người dùng truy xuất vào tài nguyên mà trình chủ cho phép
    if (valid) {
     nextDoc = in.readUTF();
    }
    //Đóng kết nối từ luồng nhập
    in.close();
    return valid;
  }
}
```

LoginApplet sẽ được hiển thị bên trong một trang Web. Trang Web này do servlet AppletLogin sinh ra. Mã nguồn của servlet AppletLogin được cài đặt như sau

Ví dụ 5-16 AppletLogin.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
/**
servlet này sẽ tạo trang HTML phía trình khách để hiển thị
LoginApplet. Applet phía trình khách có nhiệm vụ chuyển thông tin
đăng nhập của người dùng lên cho servlet xử lý. Servlet của ta tiếp
```

nhận thông tin đăng nhập trong phương thức doPost(), kiểm tra tính xác thực và trả về kết quả cho trình khách.

```
*/
public class AppletLogin extends HttpServlet
{
 public static String USER_KEY = "ServletLogin.user";
 static java.util.Hashtable crossRef;
 public void doGet(HttpServletRequest req,
                   HttpServletResponse resp)
        throws ServletException, java.io.IOException
  {
    // Định kiểu tài nguyên MIME trả về cho trình khách
   resp.setContentType("text/html");
    // Lấy về luồng xuất để ghi dữ liệu
   java.io.PrintWriter out = resp.getWriter();
    // Yêu cầu trình khách không sử dụng cache để lưu giữ trang
kết xuất
   resp.setHeader("Expires",
              "Tues, 01 Jan 1980 00:00:00 GMT");
   //Lấy về session hiện hành. Chúng ta lưu mật khẩu và tên người
   //dùng trong biến session để xác thực rằng người dùng đã truy
cập hợp lệ
   HttpSession session = req.getSession(true);
   String user = (String) session.getValue(USER_KEY);
   if (user == null) {
     // Kiểm tra trị session id chứa trong bảng băm crossRef
    user = (String) crossRef.get(session.getId());
    if (user != null) {
      // Tìm thấy user trong bảng băm.
      // Lưu lại user vào biến session. Loại bỏ user khỏi bảng băm
```

```
      session.putValue(USER_KEY, user);
      crossRef.remove(session.getId());
    }
  }
  if (user == null) {
    // Chưa có user nào đăng nhập. Hiển thị form yêu cầu đăng
nhập
    login(out, req);
    return;
  }
  // Kết xuất cho biết qua trình đăng nhập thành công
  out.println("<html>");
  out.println("<head>");
  out.println("<title>Wecome</title>");
  out.println("</head>");
  out.println("<body>");
  out.println("<center><h2>
                Welcome to our site!</h2>");
  out.println("<br>");
  out.println(" You have successful login ...");
  out.println("</center>");
  out.println("</body>");
  out.println("</html>");
  out.flush();
}
/**
 * Thực hiện lệnh HTTP POST
*/
public void doPost(HttpServletRequest req,
        HttpServletResponse resp)
 throws ServletException, java.io.IOException
```

```
{
    // Lấy về luồng nhập để đọc dữ liệu do Applet trình khách gửi
lên
    DataInputStream in =
        new DataInputStream(req.getInputStream());
    // Định kiểu tài nguyên MIME trả về cho trình khách dạng nhị
    // phân. Applet phía trình khách sẽ tiếp nhận dữ liệu này
    resp.setContentType("application/octet-stream");
   // Tạo luồng đệm
    ByteArrayOutputStream byteOut = new
                        ByteArrayOutputStream();
    // Tạo luồng xuất để gửi dữ liệu
    DataOutputStream out = new
                DataOutputStream(byteOut);
    // đọc giá trị session id do trình khách chuyển lên
    String sessionId = in.readUTF();
    // đọc dữ liệu chứa username và password
    String user = in.readUTF();
    String password = in.readUTF();
    // Bạn có thể thực hiện quá trình giải mã thông tin về
passwaord ở đây.
    // Giai đoạn này được thực hiện trong trường hợp  trình khách
    // có thực hiện việc mã hoá
    // ...
    // Kiểm tra tính hợp lệ của username và password
    if (!validUser(user, password)) {
   // Trả về thông tin cho Applet thông báo đăng nhập không hợp lệ
     out.writeBoolean(false);
    }
    else {
```

```
    //Người dùng đăng nhập hợp lệ. Đặt thông tin về user vào bảng
    //băm crossRef
    crossRef.put(sessionId, user);
    //Thông báo cho trình khách quá trình đăng nhập thành công
    out.writeBoolean(true);
    //Định hướng cho phép trình khách thâm nhập tài nguyên
    out.writeUTF("http://localhost:8080/
                                servlet/counter");
    }
    // Chuyển kết xuất ra bộ đệm
    out.flush();
    // Lấy kích thước khối dữ liệu từ bộ đệm
    byte[] buf = byteOut.toByteArray();
    // Định chiều dài dữ liệu cần gửi đi
    resp.setContentLength(buf.length);
    // Gửi dữ liệu về cho Applet của trình khách
    ServletOutputStream servletOut =
                        resp.getOutputStream();
    servletOut.write(buf);
    //đóng kết nối
    servletOut.close();
  }
  /**
   * Định dạng trang đăng nhập login
   */
  protected void login(java.io.PrintWriter out,
                       HttpServletRequest req)
                         throws java.io.IOException
  {
    // Lấy về đối tượng session
```

```
    HttpSession session = req.getSession(true);
    //Kết xuất trang web
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Login</title>");
    out.println("<center><h2>
                Welcome! Please login</h2>");
    //Mã HTML dùng hiển thị Applet phía trình khách
    out.println("<applet width=200 height=120");
    out.println("  name=\"LoginApplet\"");
    out.println("  codebase=http://localhost:8080/");
    out.println("  code=\"LoginApplet\">");
    out.println("<param name=\"servlet\" value=\"" +
                        req.getRequestURI() + "\">");
    out.println("<param name=\"id\" value=\"" +
                        session.getId() + "\">");
    out.println("</applet>");
    out.println("</center></body></html>");
  }
  /**
   * Kiểm tra username và password
   * trả về true nếu việc kiểm tra thành công
   */
  protected boolean validUser(String username,
                              String password)
  {
    boolean valid = false;
  // Kiểm tra theo qui ước password=username. Bạn có thể thực hiện
  //quá trình đọc cơ sở dữ liệu một cách chi tiết hơn.
    if ((username != null) && (username.length() > 0)) {
      valid = username.equals(password);
```

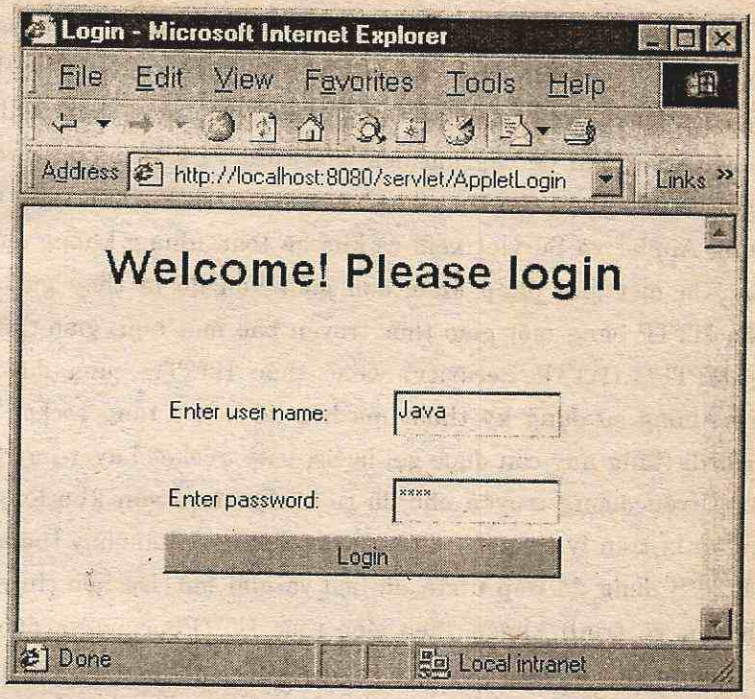
```
    }
    return valid;
  }
/* Phương thức khởi tạo servlet */
 public void init(ServletConfig cfg)
              throws ServletException
  {
    // Tạo bảng băm crossRef để lưu thông tin hỗ trợ servlet
    if (crossRef == null) {
     crossRef = new java.util.Hashtable();
    }
    super.init(cfg);
  }
/* Phương thức hủy servlet */
 public void destroy()
  {
    super.destroy();
  }
}
```

Bạn biên dịch AppletLogin.java và LoginApplet.java thành tập tin .class. Đăng ký servlet AppletLogin với dịch vụ web. Chép Applet LoginApplet.class vào thư mục $web_server\public_html.

Để kiểm tra, bạn gõ địa chỉ để triệu gọi servlet AppletLogin như sau

http://localhost:8080/servlet/AppletLogin

Lần đầu tiên khi bạn chưa đăng nhập, trình chủ sẽ trả về màn hình login như hình 5-13. Bạn nhập vào username và password tiếp đến nhấn nút login của Applet trên trang web. Applet sẽ gửi thông tin về cho servlet xác thực xem thông tin của bạn nhập vào là hợp lệ hay không.



Hình 5-13 Đăng nhập bằng Applet

Màn hình đăng nhập vào trang Web quản trị trình chủ http://localhost:9090 chính là một Java Applet thực hiện kỹ thuật chứng thực này.

4.4. Bảo vệ trình chủ bằng giao thức HTTPS an toàn (SSL)

Trên đây là một số cách bảo vệ trình chủ và các tài nguyên truy xuất thông qua cơ chế đăng nhập yêu cầu trình khách cung cấp username và password. Điều đáng quan tâm đó là dữ liệu truyền

theo giao thức HTTP không được bảo vệ trong quá trình chuyển từ trình khách lên trình chủ. Kỹ thuật chứng thực đơn giản (basic authentication) hay chứng thực bằng cơ chế băm (digest authentication) chỉ đề cập mã hóa password và giải mã password ở đầu gửi và đầu nhận còn thì giao thức truyền vẫn là HTTP. Kỹ thuật sử dụng Applet và Servlet xem ra khá an toàn nhưng không được tự nhiên. Có một giải pháp khác đơn giản hơn đó là thay giao thức truyền HTTP bằng một giao thức truyền bảo mật hơn, giao thức mở rộng HTTPS (HTTP Security). Giao thức HTTPS cũng dựa trên HTTP nhưng sử dụng kỹ thuật mã hoá dữ liệu ở tầng socket trước khi gửi đi (tầng này còn được gọi là Security Socket Layer hay SSL). Dữ liệu trên đường truyền cho dù có bị đọc được vẫn khó lòng giải mã (ít ra là trên lý thuyết). Trên trình chủ bạn phải chạy thêm dịch vụ HTTPS dùng để tiếp nhận dữ liệu và giải mã. Dữ liệu chuyển từ trình chủ về trình khách theo giao thức HTTPS cũng sẽ được mã hóa. Trình khách (hay trình duyệt browser) nhận được dữ liệu sẽ thực hiện quá trình giải mã ngược lại.

Điều đáng mừng là hầu hết các trình duyệt mới hiện nay như IE hay Netscape đều chấp nhận hỗ trợ HTTPS. Bạn chỉ cần chọn trình chủ nào chịu phục vụ HTTPS là đủ. Để truy xuất một tài nguyên theo giao thức HTTPS bạn thay tiếp đầu ngữ http:// trong địa chỉ URL bằng https://.

Trình chủ Java Web Server JWS 2.0 hỗ trợ bạn dịch vụ HTTPS chạy trên cổng 7070. Để khởi động dịch vụ HTTPS trước hết bạn cần phải tạo khóa và thuật toán mã hóa mà HTTPS sẽ áp dụng. Bạn chạy chương trình authstore.exe trong thư mục C:\JavaWebServer2.0\bin. Chương trình này hiển thị cửa sổ để bạn nhập vào thông tin tạo

khóa (như tên công ty, địa chỉ, mật mã ...). Khóa được tạo ra sẽ lưu vào tập tin mang tên C:\JavaWebServver2.0\keys. Khởi động lại dịch vụ Web httpd. Vào lại trang chủ cấu hình http://localhost:9090 bạn sẽ thấy dịch vụ HTTPS mang tên *Secure Web Service* được khởi động ở trạng thái *running*. Thư mục gốc mặc định của dịch vụ này đặt ở C:\JavaWebServer2.0\secure_html. Bạn nên đặt tài nguyên của mình vào thư mục này cho tất cả các truy xuất theo giao thức https:// phía trình khách.

Để kiểm tra và truy xuất tài nguyên theo giao thức HTTPS bạn gõ địa chỉ URL sau

https://localhost:7070/

Với giao thức HTTPS tất cả dữ liệu gửi đi đều được tầng SSL mã hóa và giải mã bạn không cần phải quan tâm đến vấn đề bảo mật dữ liệu trên đường truyền từ trình khách đến trình chủ hay ngược lại.

5. KẾT CHƯƠNG

Qua chương này bạn đã nắm được một số tính năng khác hữu ích của servlet như: có thể triệu gọi từ ứng dụng khách thông thường mà không cần đến trình duyệt. Applet và servlet có thể giao tiếp được với nhau bằng cách chuyển dữ liệu theo giao thức HTTP. Kỹ thuật xuyên hầm (tunneling) là giải pháp mà trình khách có thể triệu gọi phương thức của các đối tượng RMI ở xa theo giao thức HTTP xuyên qua tường lửa. Bạn cũng đã học cách bảo vệ tài nguyên

trên trình chủ bằng cách cấu hình Java Web Server, sử dụng biểu mẫu đăng nhập (HTML Form) hoặc Applet, sử dụng giao thức HTTPS thay cho giao thức HTTP không an toàn.

Servlet cung cấp cho bạn cách thức xây dựng những ứng dụng Web mạnh mẽ đáp ứng yêu cầu ở cả hai phía khách chủ. Tuy nhiên Sun cung cấp cho bạn cách viết ứng dụng Web đơn giản hơn nữa. Đó là cơ chế viết các trang JSP (Java Server Page) thay cho servlet. Chúng ta sẽ nghiên cứu về JSP trong chương tiếp theo.

Chương 6

JSP (JAVA SERVER PAGE)

Tóm tắt chương 6

Chương này sẽ trình bày với bạn kỹ thuật xây dựng trang jsp (Java Server Page). Jsp là cách kết hợp ngôn ngữ HTML truyền thống và ngôn ngữ Java phía máy chủ (server side) để sinh ra các trang web động phục vụ các ứng dụng web uyển chuyển hơn.

Các vấn đề chính sẽ được đề cập đến:

- ✓ *Jsp và các khái niệm mở đầu. Xây dựng trang jsp.*
- ✓ *Tìm hiểu các thẻ jsp.*
- ✓ *Truy xuất cơ sở dữ liệu với trang jsp*
- ✓ *Trang jsp và truy xuất đối tượng phân tán RMI*

1. JSP THAY CHO SERVLET

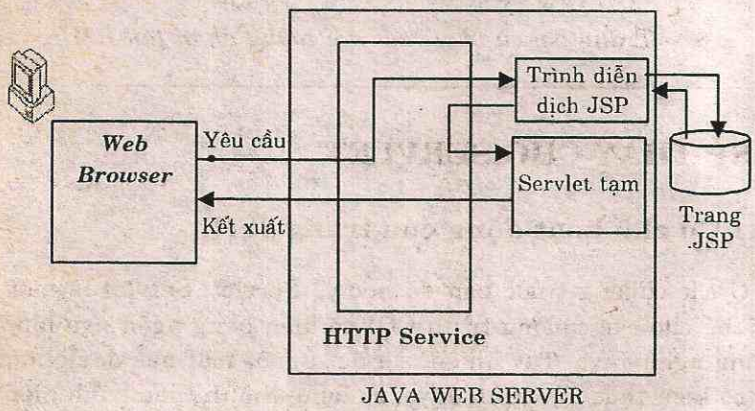
1.1 Cơ chế hoạt động của trang JSP

Ở các chương trước bạn đã học về Servlet. Servlet là một cách thay thế cho các chương trình CGI cổ điển bằng ngôn ngữ lập trình và công nghệ Java. Tuy nhiên, servlet ở một mặt nào đó đòi hỏi bạn phải có kiến thức lập trình Java, sử dụng thành thạo trình biên dịch javac để biên dịch servlet ra tập tin .class. Đăng ký servlet với trình chủ Web trước khi sử dụng. (Những công việc này bạn đã làm thành thạo trong chương 4 và 5).

JSP (Java Server Page) là một cách đơn giản hơn nữa để người dùng (nhất là lập trình viên Java không chuyên) tiếp cận được hướng lập trình Web phía máy chủ hiệu quả và nhanh hơn. Nếu bạn

chưa biết tường tận Java hay Servlet bạn vẫn có thể thiết kết được ứng dụng Web đáp ứng nhu cầu người dùng phía trình khách thông qua các trang JSP.

Servlet đưa mã HTML vào lệnh Java trong khi ngược lại JSP đưa lệnh Java vào các mã (hay thẻ) HTML. Tương tự các trang SHTML chứa thẻ <SERVLET> được trình chủ thông dịch trước khi trả kết xuất về cho máy khách. Các trang JSP chứa các thẻ đặc biệt qui định gần giống thẻ của ngôn ngữ HTML (thật ra các thẻ trong trang JSP theo chuẩn XML mà chúng ta sẽ tìm hiểu trong chương sau). Khi bạn yêu cầu một trang JSP, trình chủ sẽ đọc trang JSP từ đĩa cứng, bộ diễn dịch JSP (JSP Page Compiler) sẽ diễn dịch mã lệnh Java chứa trong trang JSP thành một servlet. Sau đó trình chủ Java Web Server sẽ triệu gọi servlet trả kết xuất thuần HTML về cho trình khách. Cơ chế hoạt động của trang JSP được minh họa như hình 6-1



Hình 6-1 Cơ chế triệu gọi trang JSP

1.2 Xây dựng trang JSP

Dưới đây là minh họa cách viết mã lệnh Java trong trang .jsp.

Ví dụ 6-1 FirstPage.jsp

```
<HTML>
<H1>
Hello JSP !
</H1> <P>
<%
    //In ra ngày giờ hiện hành trên máy chủ
   out.println("The time now is: " + new
                    java.util.Date()+"<br>");
    //In một dãy các chuỗi Hello từ font chữ cỡ 6 đến font chữ cỡ 1
   for (int i=6; i>0; i--) {
      out.println("<FONT Size="+i+"> Hello <FONT>");
   }
%>
</HTML>
```

Bạn lưu ví dụ 6-1 thành tập tin mang tên FirstPage.jsp vào thư mục $web_server\public_html ($web_server là thư mục bạn cài đặt trình chủ hiểu Java JWS 2.0, thường là C:\JavaWebServer2.0 - Bạn xem lại chi tiết cách cài đặt và cấu hình trình chủ JWS 2.0 ở chương 4).

Để kiểm tra kết quả kết xuất của trang .jsp bạn gõ địa chỉ triệu gọi URL như sau

http://localhost:8080/FirstPage.jsp

Kết quả hiển thị trên trình duyệt phía máy khách được thể hiện ở hình 6-2

Hình 6-2 Kết xuất trang JSP

Nếu chọn *View | Source* từ trình đơn của IE bạn có thể thấy trang FirstPage.jsp được diễn dịch thành mã HTML phía máy khách như sau

```
<HTML>
<BODY>
<H1>
Hello JSP !
</H1> <P>
The time now is: Mon Apr 27 23:47:12 PDT 1998<br>
<FONT Size=6> Hello <FONT>
<FONT Size=5> Hello <FONT>
<FONT Size=4> Hello <FONT>
<FONT Size=3> Hello <FONT>
```

```
<FONT Size=2> Hello <FONT>
<FONT Size=1> Hello <FONT>
</HTML>
```

Như bạn thấy, bên trong trang FirstPage.jsp ở ví dụ 6-1, cặp thẻ

```
<%
    //Mã java
%>
```

chính là qui định của trình diễn dịch JSP cho phép bạn đặt mã Java trong đó. Đặc tả JSP cung cấp rất nhiều thẻ cho phép bạn tùy biến khi nhúng lệnh Java trong trang JSP. Các thẻ này sẽ kết hợp với các lệnh hay thẻ chuẩn HTML để tạo ra kết xuất cuối cùng trả về trình khách.

1.3. JSP hay Servlet?

Chức năng của trang JSP hoàn toàn giống với servlet tuy nhiên bạn không cần phải biên dịch trang JSP bằng tay. Trình chủ Web server sẽ thực hiện công việc này hộ bạn. Kết xuất của trang FirstPage.jsp ở ví dụ 6-1 hoàn toàn tương đương với kết xuất của servlet HelloPage dưới đây

Ví dụ 6-2 HelloPage.java

```
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloPage extends HttpServlet
{
  /* Thực hiện lệnh HTTP GET */
  public void doGet(HttpServletRequest req,
                    HttpServletResponse resp)
```

```
        throws ServletException, java.io.IOException
  {
   // Định kiểu tài nguyên MIME trả về trình khách
  resp.setContentType("text/html");
   // Lấy về luồng xuất
  java.io.PrintWriter out = resp.getWriter();
   // Kết xuất trang Web cho trình khách
  out.println("<html>");
  out.println("<H1>");
  out.println("Hello JSP !");
  out.println("</H1> <P>");
  out.println("The time now is: "
                  + new java.util.Date()+"<br>");
  for (int i=6; i>0; i--) {
    out.println("<FONT Size="+i+"> Hello <FONT>");
  }
  out.println("</html>");
  out.flush();
 }
/* Khởi tạo servlet */
public void init(ServletConfig cfg)
                  throws ServletException
 {
  super.init(cfg);
 }
/* Phương thức hủy servlet */
```

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

Bạn biên dịch HelloPage.java thành tập tin .class, đăng ký HelloPage.class với trình chủ Web. Sau đó triệu gọi servlet bằng địa chỉ URL

http://localhost:8080/servlet/HelloPage

Bạn sẽ nhận được kết xuất hiển thị phía trình duyệt như hình 6-2 tương tự kết xuất của trang FirstPage.jsp. Rõ ràng, viết trang JSP đơn giản hơn viết servlet. Nếu có sự so sánh bạn sẽ thấy, khi viết servlet bạn tự tạo đối tượng luồng xuất

```
java.io.PrintWriter out = resp.getWriter();
```

để kết xuất kết quả trả về cho trình khách. Trong khi đó nếu viết trang JSP bạn không cần phải thực hiện điều này. Bộ diễn dịch JSP đã tạo sẵn đối tượng out cho bạn sử dụng.

Thật sự thì trang JSP được biên dịch thành servlet phía hậu trường, kết xuất của trang JSP thật ra là kết xuất của servlet. Để thấy được cơ chế này sau khi trang FirstPage.jsp hiển thị bạn có thể chuyển vào thư mục $web_server\tmpdir\default\pagecompile\jsp, trình chủ JWS2.0 tự động biên dịch FirstPage.jsp thành 3 tập tin

_FirstPage.java : Tập tin nguồn dịch jsp thành mã servlet tương ứng (bạn có thể đọc nội dung tập tin này nếu muốn tìm hiểu sâu vào cách thông dịch từ jsp thành servlet của trình diễn dịch jsp)

_FirstPage.class : Servlet đã được biên dịch

_FirstPage.dat : Dữ liệu hỗ trợ trong quá trình biên dịch servlet. (Thật sự tập tin .dat này là một đối tượng lưu các dòng văn bản thuần HTML của trang JSP)

Tùy theo mỗi trình chủ khác nhau (như JRun, WebLogic ...) mà bạn sẽ thấy các tập tin diễn dịch .jsp ra servlet khác nhau. Theo cơ chế của JSP bạn không cần phải quan tâm đến những tập tin phụ này. Nếu bạn xóa các tập tin này đi, trình chủ sẽ tự động tạo mới lại.

Nếu bạn tự hỏi theo cơ chế này trang JSP có thể phải thực hiện lâu (hay chậm) hơn servlet thuần túy. Điều này đúng, nhưng chỉ là lần đầu khi trang JSP được biên dịch. Ở lần triệu gọi kế tiếp từ máy khách, trình chủ không cần dịch lại trang JSP nữa (trừ khi bạn có thay đổi nội dung trang JSP). Mã JSP lúc này là mã Java nhị phân (byte-code) được gọi thực thi trực tiếp. Chính vì lý do này, trang .jsp xét về mặt tốc độ sẽ nhanh hơn hẳn các trang web thực hiện cơ chế diễn dịch hiện nay như .asp (Active Server Page) của Microsoft hay trang .pl của trình CGI Perl.

Thật sự JSP là cạnh tranh của Sun đối với công nghệ ASP của Microsoft. ASP cũng là các trang Web thực hiện cơ chế diễn dịch nhưng thay cho các lệnh Java là các lệnh Visual Basic kết hợp với kiến trúc triệu gọi đối tượng thành phần COM/DCOM. ASP chạy trên trình chủ IIS của Microsoft. ASP không có cơ chế dịch ra mã nhị phân như JSP. Nếu muốn có sự so sánh chi tiết về JSP và ASP bạn có thể truy xuất địa chỉ http://java.sun.com/ trên trang chủ của Sun chuyên về Java.

Viết trang JSP đơn giản hơn viết Servlet và bạn hoàn toàn có thể tận dụng mọi chức năng của Servlet bên trong trang JSP. Tuy

nhiên servlet thường được xem như những đối tượng thành phần (component) nhúng vào trình chủ (tương tự đối tượng ActiveX hay COM của Microsoft). Chính vì vậy servlet thường được dùng cho các chức năng xử lý phức tạp (như giao tiếp với Applet phía trình khách, thực hiện bảo mật tài nguyên, chứng thực mật khẩu ...). Trong khi đó trang JSP được dùng cho các thao tác truy xuất hay xử lý đơn giản (như trình bày giao diện, định dạng trang HTML, triệu gọi các thành phần JavaBean hay servlet khác).

Kết hợp JSP và servlet là cách tốt nhất khi bạn chọn phát triển ứng dụng Web bằng Java phía máy chủ.

1.4. Trang JSP tiếp nhận dữ liệu phía máy khách

Với servlet bạn thường cài đặt hai phương thức doGet() và doPost(). doGet() (tương ứng với lệnh HTTP GET) thường trả về tài nguyên cho trình khách trong khi đó doPost() (tương ứng với lệnh HTTP POST) thường dùng tiếp nhận dữ liệu do trình khách gửi lên.

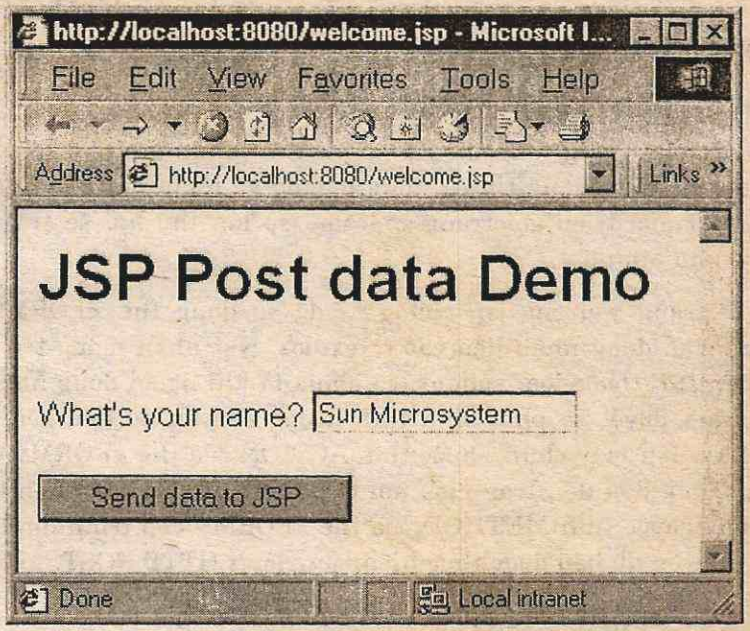
Với trang JSP bạn hoàn toàn có thể lấy về dữ liệu do trình duyệt phía máy khách chuyển lên để xử lý. Dưới đây là một ví dụ minh họa. Trang welcome.jsp của ta yêu cầu người dùng nhập vào tên họ. Nếu tên khác rỗng, trang JSP sẽ trả về lời chào

Ví dụ 6-3 welcome.jsp

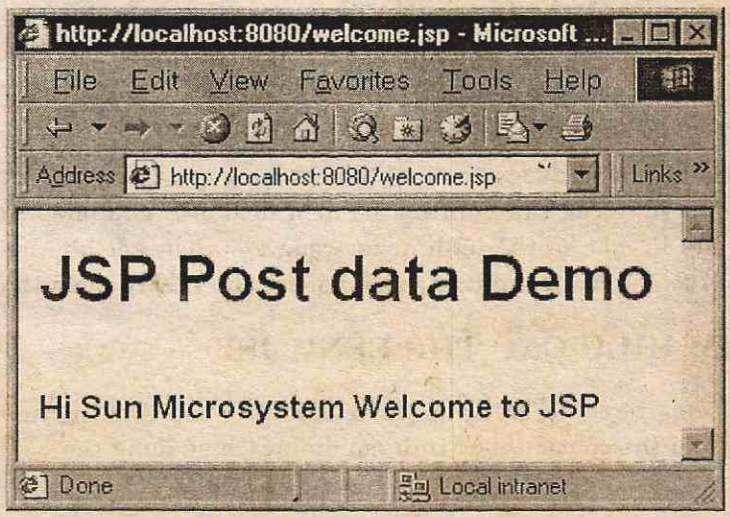
```
<HTML>
<H1> JSP Post data Demo </H1> <br>
<%
  //Lấy về dữ liệu do trình khách gửi lên nếu có
  String username=request.getParameter("yourname");
  if (username !=null){
    //In ra tên họ do người dùng nhập vào
```

```
    out.println("<STRONG> Hi "+
            username+" Welcome to JSP </STRONG>");
  } else {
  //Nếu lần đầu tiên người dùng mở trang jsp thì hiển thị mẫu form
  //cho phép người dùng nhập vào tên họ
  >%
    <FORM ACTION= welcome.jsp  METHOD=POST>
      What's your name?
      <INPUT TYPE=text  NAME=yourname> <p>
      <INPUT TYPE=submit
                          VALUE='Send data to JSP'>
    </FORM>
  <%
   }
  %>
</HTML>
```

Trong ví dụ trên, đối tượng request là đối tượng mặc định trình JSP cho phép bạn dùng để triệu gọi lấy về các tham số hay dữ liệu do trình khách chuyển lên. Thực sự bạn có thể xem request tương đương với đối tượng HttpServletRequest mà ta đã từng dùng khi viết servlet. Ở đây, bộ diễn dịch JSP đã tự tạo ra đối tượng request bạn chỉ việc sử dụng.



Hình 6-3 Trang JSP nhập dữ liệu phía trình khách



Hình 6-4 Kết quả trả về của trang JSP sau khi xử lý dữ liệu

Chép welcome.jsp vào thư mục $web_server\public_html. Bạn gõ địa chỉ URL sau để kiểm tra

http://localhost:8080/welcome.jsp

Kết quả trang welcome.jsp sẽ hiển thị như hình 6-3. Bạn nhập vào tên họ hay thông tin cá nhân nào đó và nhấn nút *Send data to JSP*. Kết quả triệu gọi trang welcome.jsp lần thứ hai sẽ trả thông tin về như hình 6-4.

Ở trang welcome.jsp chúng ta đã sử dụng thẻ <FORM> cho phép người dùng nhập liệu vào ô textbox. Nút nhấn mang tên *Send data to JSP* thuộc loại submit (có nghĩa là khi người dùng nhấn nút này trình duyệt sẽ phát sinh tình huống chuyển dữ liệu bên trong <FORM> lên máy chủ). Thuộc tính ACTION của thẻ <FORM> sẽ chỉ định đường dẫn đến trang jsp nơi tiếp nhận dữ liệu của trình khách gửi lên. Thuộc tính METHOD của thẻ <FORM> cho trình duyệt biết cách chuyển dữ liệu được thực hiện bằng lệnh HTTP POST.

Tuy trang welcome.jsp hết sức đơn giản nhưng nó bao hàm toàn bộ cơ chế nhập liệu và chuyển dữ liệu từ trình khách lên trình chủ. Nếu lướt trên mạng, đi vào những trang web yêu cầu đăng ký thông tin cá nhân – như khi sử dụng dịch vụ web của Yahoo mail, Hot mail chẳng hạn - bạn đều gặp phải những biểu mẫu FORM thực hiện kỹ thuật nhập liệu tương tự chúng ta đã làm với welcome.jsp.

Để tận dụng được sức mạnh khi sử dụng các trang JSP chúng ta cần nắm vững các thẻ lệnh mà bộ diễn dịch JSP cho phép sử dụng. Dưới đây là một số thẻ lệnh quan trọng giúp bạn kết hợp ngôn ngữ và đối tượng Java trong trang JSP với các thẻ HTML quen thuộc.

2. TÌM HIỂU CÁC THẺ LỆNH JSP

Tương tự thẻ HTML, thẻ lệnh JSP cũng bao gồm thẻ mở và thẻ đóng. Mỗi thẻ có các thuộc tính qui định cách sử dụng thẻ.

Thực sự các thẻ JSP được xây dựng theo đặc tả và chuẩn XML (Extension Makup Language) nên có hơi khác với thẻ HTML. Chuẩn XML không xem chữ hoa và chữ thường giống nhau. Chính vì vậy khi sử dụng thẻ lệnh JSP bạn nên chú ý kiểu chữ và cố gắng viết đúng cú pháp.

2.1. Thẻ <jsp:scriptlet> hay <% %>

Với thẻ này bạn có thể đặt các đoạn mã lệnh Java ở giữa cặp thẻ tương tự một chương trình java thông thường. Ví dụ

```
<H1> HTML Text here
<jsp:scriptlet>
  //Ở đây là mã lệnh Java
  out.println("The time now is: " + new
                                  java.util.Date()+"<br>");
  for (int i=6; i>0; i--) {
     out.println("<FONT Size="+i+"> Hello <FONT>");
  }
</jsp:scriptlet>
```

Khi gặp thẻ này, bộ diễn dịch JSP sẽ biết được lệnh cần thực hiện tiếp theo là mã lệnh Java. Và bạn hoàn toàn có thể sử dụng mọi cú pháp Java ở đây, kể cả lệnh chú thích thông thường.

Nếu bạn muốn viết tắt thì có thể sử dụng cặp thẻ <% %> thay thế. Ví dụ sau là hoàn toàn tương đương với khi dùng thẻ <jsp:scriptlet>

```
<%
  //Ở đây là mã lệnh Java
  out.println("The time now is: " + new
                                  java.util.Date()+"<br>");
```

```
 for (int i=6; i>0; i--) {
    out.println("<FONT Size="+i+"> Hello <FONT>");
 }
%>
```

Cú pháp <% vẫn thường được lập trình viên JSP sử dụng hơn.

2.2. Thẻ khai báo và thực hiện biểu thức <%! , <%=

Bạn có thể dùng thẻ <%! để khai báo một biến dùng cho toàn trang jsp. Biến khai báo phải theo đúng cú pháp của ngôn ngữ Java. Thẻ <%= được dùng để hiển thị một biểu thức (expression). Ví dụ

```
<HTML>
<%! int count=12; // Khai báo biến count %>
<%! String description="Increase count variable";
                    // Khai báo biến description %>
<% = description %>
<% = ++count %>
</HTML>
```

Riêng biểu thức nằm trong thẻ <%= không được tận cùng bằng dấu chấm phẩy (;) như các lệnh hay khai bao biến của ngôn ngữ Java. Thực tế, ở ví dụ trên lệnh Java sẽ được JSP diễn dịch như sau

```
int count=12;
String description="Increase count variable";
out.println(description);
out.println(++count );
```

Chính vì vậy bạn đừng nên đặt dấu (;) vào cuối biểu thức trong thẻ <%=.

Bạn cũng có thể dùng thẻ <%! để khai báo hàm hoặc một phương thức bên trong trang JSP.

```
<%!
    public int maxNumber(int x, int y)
    {
        if (x>y) return x;
        else return y;
    }
%>
```

2.3. Thẻ nhúng mã nguồn <%@ include file %>

Bạn có thể nhúng một trang .jsp hay trang .html vào trang jsp hiện hành bằng thẻ <%@ include file %>. Thẻ này tương tự chỉ dẫn #include trong ngôn ngữ C. Cú pháp đầy đủ của thẻ này là

```
<%@ include file ="URL or FilePath"%>
```

Ví dụ, tập tin **testfile.jsp** có nội dung

```
<HTML>
   Include page demo
   <% out.println("<STRONG> Today is " %>
   <%@ include file ="datepage.jsp"%>
</HTML>
```

và **datepage.jsp** có nội dung như sau

```
<% = new java.util.Date() %>
<br> Result output from datapage.jsp
```

Khi diễn dịch **testfile.jsp**, bộ dịch JSP sẽ kết hợp thẻ include thành trang jsp

```
<HTML>
```

```
  Include page demo
  <% out.println("<STRONG> Today is " %>
  <% = new java.util.Date() %>
  <br> Result output from datapage.jsp
</HTML>
```

Kết xuất phía trình duyệt sẽ là

```
Include page demo. Today is Nov-30,2000 2:38:40
Result output from datapage.jsp
```

2.4. Thẻ chỉ dẫn biên dịch trang <%@ page %>

Thẻ <%@page %> chỉ dẫn một số tính chất biên dịch áp dụng cho toàn trang jsp. Bạn có thể sử dụng thẻ này để khai báo các thư viện import của java, chỉ định tùy chọn trang jsp có cần giữ trên cache bộ nhớ của trình chủ để tăng tốc hay không ... Ví dụ để khai báo sử dụng các thư viện java

```
<HTML>
<%@ page import="java.sql.*" %>
<%
  Connection con;   //Các đối tượng của gói thư viện
java.sql
  Statement stmt;
  ResultSet rs;
  try {
        ...
%>
</HTML>
```

Hay ví dụ dưới đây sẽ chỉ thị chuyển đến trang error.html nếu trang jsp hiện tại gặp lỗi trong quá trình thực thi

```
<%@ page errorPage="error.html" %>
```

Nếu muốn trang jsp tắt điều khiển session bạn có thể thực hiện chỉ dẫn

```
<%@ page session="false" %>
```

2.5. Thẻ chuyển hướng <jsp:forward>

Thẻ này giúp bạn chuyển hướng trang web sang địa chỉ khác. Ví dụ, khi xử lý trang nhận dữ liệu đăng nhập (login page) bạn kiểm tra mật khẩu, nếu hợp lệ bạn chuyển người dùng đến trang tài nguyên cho phép truy cập. Nếu không hợp lệ, bạn chuyển người dùng đến trang thông báo lỗi.

```
<%
   if(request.getParameter("password")=="admin") {
%>
      <jsp:forward page="errorLogin.jsp">
      </jsp:forward>
<%
  } else{
%>
      <jsp:forward page="successLogin.jsp">
      </jsp:forward>
<%
   }
%>
```

Bạn cũng có thể sử dụng thẻ `<jsp:forward>` này để chuyển một trang jsp đến servlet khác. Ví dụ

```
<jsp:forward
      page="http://localhost:8080/servlet/snoop">
```

2.6. Thẻ sử dụng thành phần Bean <jsp:useBean>

Bạn có thể tự tạo các lớp đối tượng Java và triệu gọi chúng từ bên trong trang jsp. Hướng theo công nghệ thành phần (component) Java gọi những đối tượng có thể gắn vào những ứng dụng là thành phần Bean. JavaBean là thành phần đối tượng Java được thiết kế theo một số qui tắc nội suy nhất định (instrospector). Ví dụ nếu lớp đối tượng X của bạn cài đặt phương thức

```
int getSomething();
int setSomething(int data)
```

thì mặc nhiên đối tượng của bạn được nhận định là có một thuộc tính mang tên *something* kiểu `int`. (Bạn xem *Giáo trình lý thuyết và bài tập Java* - chúng tôi đề cập đến việc thiết kế JavaBean rất chi tiết).

Dưới đây là một thành phần đối tượng Bean cung cấp thuộc tính message cho phép bạn lấy về và thay đổi thông điệp thể hiện của Bean.

Ví dụ 6-4 SimpleBean.java

```
package MyBeans;
public class SimpleBean {
 private String message = "No message specified";
 public String getMessage()
 {
    return(message);
 }
  public void setMessage(String message)
  {
    this.message = message;
  }
 }
```

Bạn biên dịch SimpleBean.java thành tập tin .class. Tiếp đến để sử dụng được thành phần Bean này bạn vào thư mục $web_server\classes tạo thư mục con MyBeans. Chép SimpleBean vào thư mục vừa tạo (lý do SimpleBean của ta được đặt trong gói MyBeans và JWS2.0 thường truy tìm các thành phần Beans trong thư mục $web_server\classes).

Thẻ <jsp:useBean> dùng để khai báo và khởi tạo Bean. Bạn gọi SimpleBean bằng thẻ jsp như sau

```
<jsp:useBean id="test" class="MyBeans.SimpleBean" >
</jsp:useBean>
```

Đặc tả XML cho phép bạn viết gộp thẻ đóng và thẻ mở trong trường hợp giữa hai thẻ này không chứa nội dung. Vì vậy bạn có thể viết khai báo Bean ngắn gọn như sau

```
<jsp:useBean id="test" class="MyBeans.SimpleBean" />
```

Nếu không muốn sử dụng thẻ jsp bạn có thể dùng lệnh Java khai báo đơn giản như sau

```
<%
  MyBeans.SimpleBean test=new MyBeans.SimpleBean()
%>
```

2.7. Thẻ đặt thuộc tính cho Bean <jsp:setProperty>

Bạn sử dụng thẻ này để triệu gọi một phương thức nào đó của Bean. Ví dụ, để gọi phương thức setMessage() của SimpleBean bạn sử dụng lệnh sau

```
<jsp:setProperty
      name     ="test"
      property="message"
      value    ="Hello WWW" />
```

Dựa vào giá trị property=message của thẻ, trình dịch JSP sẽ gọi phương thức setMessage() của biến test (là một thành phần Bean được khai báo bởi thẻ <jsp:useBean> trước đó).

Bạn lưu ý, giá trị property là message chứ không cần phải viết đầy đủ và đúng với tên phương thức setMessage của Bean.

Bạn cũng có thể viết ngắn gọn bằng lệnh Java sau

```
<%
    test.setMessage("Hello WWW");
%>
```

2.8. Thẻ lấy thuộc tính của Bean <jsp:getProperty>

Ngược với thẻ <jsp:setProperty>, thẻ <jsp:getProperty> dùng để lấy về nội dung của một thuộc tính. Ví dụ

```
<jsp:getProperty name="test" property="message" />
```

sẽ triệu gọi thành phần Bean mang tên test và gọi phương thức getMessage() của thành phần này.

Dưới đây là ví dụ sử dụng kết hợp các thẻ cho phép sử dụng Bean trong trang JSP

Ví dụ 6-5 TestBean.jsp

```
<HTML>
Call Bean in JSP webpage <br>
<jsp:useBean id="test" class="MyBeans.SimpleBean" />
<jsp:setProperty name="test"
        property="message"
        value="Hello WWW" />
<H1>Message:
<I>
  <jsp:getProperty name="test" property="message" />
</I>
</H1>
```

```
</HTML>
```

Kết quả kết xuất phía trình khách

```
Call Bean in JSP webpage
Message: Hello WWW
```

Bạn cũng có thể thu được kết quả tương tự nếu gọi Bean bằng các lệnh Java trong trang JSP như sau

Ví dụ 6-6 TestBean2.jsp

```
<HTML>
Call Bean in JSP webpage <br>
<%
    MyBeans.SimpleBean test= new MyBeans.SimpleBean();
   test.setMessage("Hello WWW");
%>
<H1>Message:
<I>
 <%
     out.println(test.getMessage());
 %>
</I>
</H1>
</HTML>
```

Thật sự thì các thẻ <jsp:useBean>, <jsp:getProperty>, <jsp:setProperty> có vẻ hơi phức tạp và hình thức. Sử dụng các lệnh thuần Java bạn vẫn thu được kết quả tương tự như ví dụ 6-5 trên đây.

3. CÁC ĐỐI TƯỢNG MẶC ĐỊNH CỦA TRANG JSP

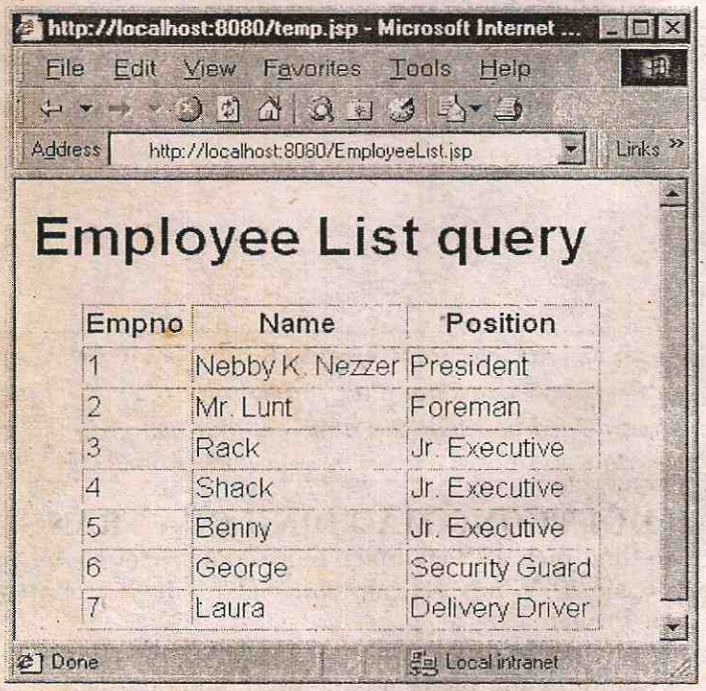
Trình diễn dịch JSP cho phép bạn sử dụng một số đối tượng đã khai báo trước. Điều này sẽ giúp bạn viết mã lệnh trong trang jsp nhanh hơn servlet.

- Đối tượng **out** : xuất phát từ lớp PrintWriter. Bạn có thể sử dụng đối tượng này để định dạng kết xuất gửi về máy khách. Ví dụ

```
<% out.println("Result "+7*3); %>
```

- Đối tượng **request**: xuất phát từ lớp HttpServletRequest. đối tượng này giúp bạn lấy về các tham số hay dữ liệu do trình khách chuyển lên.
- Đối tượng **response**: tương tự đối tượng **out**, đối tượng **response** dùng để đưa kết xuất trả về trình khách. Tuy nhiên đối tượng **out** được dùng thường xuyên hơn. **out** hỗ trợ thêm luồng đệm để tăng tốc kết xuất.
- Đối tượng **session**: xuất phát từ lớp HttpSession. Bạn sử dụng đối tượng session để theo dõi kết nối và lưu vết một phiên làm việc giữa trình khách và trình chủ (bạn xem lại cách sử dụng session ở chương 4)

4. TRUY XUẤT CƠ SỞ DỮ LIỆU TRONG TRANG JSP



Hình 6-5 Kết quả truy vấn dữ liệu của trang JSP

Thực sự trang jsp không khác gì servlet. Bạn có thể dễ dàng dùng trình JDBC để truy xuất cơ sở dữ liệu của các hệ như Access, SQL Server ... Dưới đây là ví dụ truy vấn bảng dữ liệu Employee trong cơ sở dữ liệu Access MyData.mdb (thông qua ODBC) đã được chúng ta thiết lập trong chương 4 mục 9.

Ví dụ 6-7 EmployeeList.jsp

```
<HTML>
<H1> Employee List query <H1> <br>
<%@ page import="java.sql.*" %>
<%!
 private ResultSet query(String driverName,
                         String connectionURL,
                         String query)
 {
    // Khai báo đối tượng kết nối JDBC
   Connection con = null;
    // Khai báo câu lệnh truy vấn
   Statement stmt = null;
    //Khai báo tập dữ liệu
   ResultSet rs = null;
   try {
     //Nạp trình điều khiển JDBC
    Class.forName(driverName).newInstance();
     // Tạo kết nối với cơ sở dữ liệu
    con = DriverManager.getConnection(connectionURL);
     //Tạo câu lệnh truy vấn
    stmt = con.createStatement();
```

```
      // Thực thi lệnh truy vấn dữ liệu
      rs = stmt.executeQuery(query);
      return rs;
    }
    catch (Exception ex) {
          //Xử lý lỗi ở đây
    }
    return rs;
  }
%>
<%
 //Truy vấn dữ liệu
 ResultSet rs = query("sun.jdbc.odbc.JdbcOdbcDriver",
          "jdbc:odbc:MyAccessDataSource",
          "SELECT Empno, Name, Position FROM Employee");
 if (rs==null) {
   out.println("Can not query data");
 } else {
     //Duyệt tập dữ liệu kết quả và kết xuất thành trang web
   ResultSetMetaData rsmd = rs.getMetaData();
   int columnCount = rsmd.getColumnCount();
   // Tạo khung lưới (bảng HTML)
   out.println("<center><table border>");
   //Tạo dòng tiêu đề
   out.println("<tr>");
   for (int i = 0; i < columnCount; i++) {
     // Bạn lưu ý chỉ số bắt đầu của cột dữ liệu là 1
```

```
    out.println("<th>" +
            rsmd.getColumnLabel(i + 1) +
            "</th>");
  }
  //Hết dòng tiêu đề
  out.println("</tr>");
  // Duyệt các dòng tiếp theo của tập dữ liệu
  while (rs.next()) {
    // Bắt đầu dòng
    out.println("<tr>");
    // Kết xuất các cột
    for (int i = 0; i < columnCount; i++) {
      // Kết xuất ra khung lưới
      out.println("<td>" +
              rs.getString(i + 1) +
              "</td>");
    }
    //Hết dòng
    out.println("</tr>");
  }
  // Kết thúc bảng
  out.println("</table></center>");
}
%>
</HTML>
```

Bạn gõ vào địa chỉ kiểm tra URL sau

http://localhost:8080/EmployeeList.jsp

kết quả truy vấn dữ liệu sẽ được hiển thị trên trình duyệt phía máy khách như hình 6-5

5. TRANG JSP TRIỆU GỌI ĐỐI TƯỢNG RMI, CORBA

Bởi vì trang JSP chứa hầu như toàn bộ mã Java cho nên bạn có thể triệu gọi đối tượng phân tán như RMI hay CORBA theo cách rất tự nhiên. Ví dụ dưới đây là một đối tượng RMI SmallCalculator có khả năng tính toán đơn giản. SmallCalculator chạy trên một máy chủ độc lập và được triệu gọi từ bên trong trang jsp.

Ví dụ 6-8 SmallCalculator.java

```
package jsprmi;
import java.rmi.*;
/* Đặc tả interface cho đối tượng SmallCalculator */
public interface SmallCalculator extends Remote
{
   public int minus(int number)
                        throws RemoteException;
   public int add(int x, int y)
                        throws RemoteException;
   public int subtract(int x, int y)
                        throws RemoteException;
   public int multiple(int x, int y)
                        throws RemoteException;
   public int divide(int x, int y)
                        throws RemoteException;
```

```
}
```

Cài đặt đối tượng SmallCalculator

Ví dụ 6-9 SmallCalculatorImpl.java

```
package jsprmi;
import java.rmi.*;
public class SmallCalculatorImpl
                    implements jsprmi.SmallCalculator
{
  //Thực hiện đổi dấu một số
  public int minus(int number)
                    throws RemoteException{
    return (-number);
  }
  //Thực hiện phép cộng
  public int add(int x, int y)
                    throws RemoteException{
    return (x+y);
  }
  //Thực hiện phép trừ
  public int subtract(int x, int y)
                    throws RemoteException{
    return (x-y);
  }
  //Thực hiện phép nhân
  public int multiple(int x, int y)
                    throws RemoteException{
```

```
    return (x*y);
  }
  //Thực hiện phép chia
  public int divide(int x, int y)
                        throws RemoteException {
    return (x/y);
  }
}
```

Chương trình cài đặt đối tượng RMI SmallCalculator có mã nguồn như sau

Ví dụ 6-10 SetupCalc.java

```
package jsprmi;
import java.rmi.*;
import java.rmi.server.*;
public class SetupCalc{
  public static void main(String args[])
                        throws Exception{
    //Tự tạo bộ đăng ký registry
    LocateRegistry.createRegistry(1099);
    //Tạo đối tượng RMI
    jsprmi.SmallCalculator calculator=new
                    jsprmi.SmallCalculatorImpl();
    UnicastRemoteObject.exportObject(calculator);
    System.out.println("Binding object ...");
    Naming.bind("rmi://localhost/calculator",
                                    calculator);
```

```
    System.out.println("Waiting for
                                client request...");
  }
}
```

Để cài đặt đối tượng SmallCalculator bạn cần làm theo những bước sau

Chuyển vào thư mục $web_server\classes (thay $web_server bằng C:\JavaWebServer2.0 hay thư mục mà bạn cài đặt JWS2.0), tạo thư mục con jsprmi. Chép toàn bộ mã nguồn SmallCalculator.java, SmallCalculatorImpl.java, SetupCalc.java vào thư mục này.

Biên dịch mã nguồn

```
$web_server\classes>javac jsprmi\*.java
```

Tạo lớp trung gian cho đối tượng RMI

```
$web_server\classes>rmic jsprmi.SmallCalculatorImpl
```

Chạy chương trình đăng ký đối tượng RMI

```
$web_server\classes\>java jsprmi.SetupCalc
Binding object ...
Waiting for client request
```

Mặc định JWS2.0 truy tìm các lớp do người dùng tạo ra trong thư mục $web_server\classes. Nếu bạn không đặt lớp đối tượng trong gói package nào đó thì JWS sẽ không tìm thấy. Chính vì vậy, nếu muốn viết các thành phần Bean hay sử dụng những lớp tự tạo bạn nên đưa chúng vào gói và đặt trong thư mục \classes. Điều này tùy thuộc vào mỗi trình chủ Web khác nhau. Bạn nên tham khảo thêm tài liệu của mỗi trình chủ Web Java đặc trưng.

Quá trình thiết lập đối tượng RMI đã xong. Dưới đây là trang web simplecalculator.jsp có khả năng triệu gọi đối tượng SmallCalculator từ phía trình duyệt ở máy khách.

Ví dụ 6-11 simplecalculator.jsp

```
<HTML>
<BODY>
 <%@ page import="java.rmi.*,jsprmi.*" %>
 <%
    //Tiếp nhận các đối số do trình khách chuyển lên
    //Phép tính hay toán tử cần thực hiện
   String op=request.getParameter("operator");
    //Toán hạng thứ nhất
   String xs=request.getParameter("numtext1");
     //Toán hạng thứ hai
   String ys=request.getParameter("numtext2");
 //Nếu chưa có dữ liệu cần tính toán thì hiển thị biểu mẫu form cho
 //phép người dùng nhập vào số tính toán từ trang web
    if ( ( xs==null ) || (ys==null)){
 %>
 <FORM ACTION=simplecalculator.jsp METHOD=POST>
 <!-- Toán hạng thứ nhất -->
 <INPUT id=num1 name=numtext1 >
  <!-- Danh sách toán tử -->
 <SELECT id=selectoperator name=operator>
     <OPTION selected value=add>+</OPTION>
```

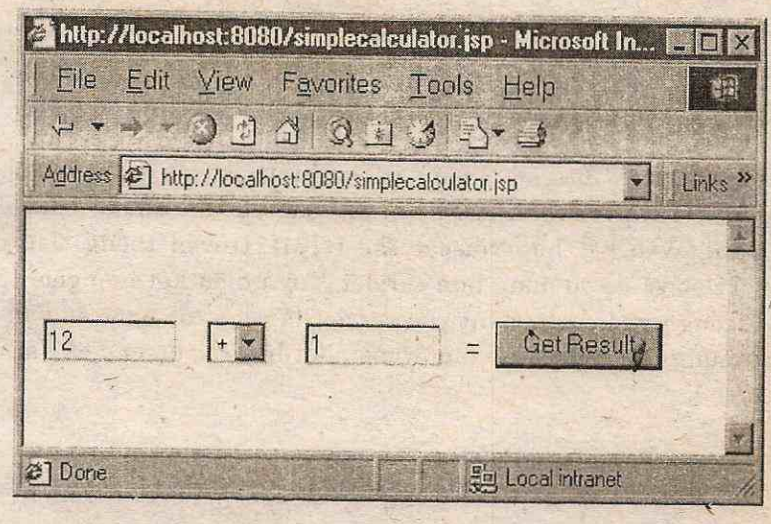
```
      <OPTION value=subtract>     -</OPTION>
      <OPTION value=multiple>     *</OPTION>
      <OPTION value=divide>       /</OPTION>
  </SELECT>
  <!-- Toán hạng thứ hai -->
  <INPUT id=num2 name=numtext2 >
  <!-- Nút nhấn kích hoạt gửi dữ liệu về trình chủ-->
  <INPUT id=button1 type=submit value="Get Result"></P>
</FORM>
<%
  } //Triệu gọi đối tượng RMI SmallCalculator
  else {
    int result=0;
    int x=Integer.parseInt(xs);  //Chuyển chuỗi thành số
    int y=Integer.parseInt(ys);
    //Tham chiếu đến đối tượng RMI
    SmallCalculator calculator=(SmallCalculator)
          Naming.lookup("rmi://localhost/calculator");
    //Gọi các phương thức của đối tượng tính toán kết quả
    if (op.equals("add")){
       result=calculator.add(x,y);
    }
    if (op.equals("subtract")){
       result=calculator.subtract(x,y);
    }
    if (op.equals("multiple")){
```

```
        result=calculator.multiple(x,y);
    }
    if (op.equals("divide")){
        result=calculator.divide(x,y);
    }
    //Kết xuất kết quả trả về trình khách
    out.println("<H1> The result is "+result+"<H1>");
  }
%>
</BODY>
</HTML>
```

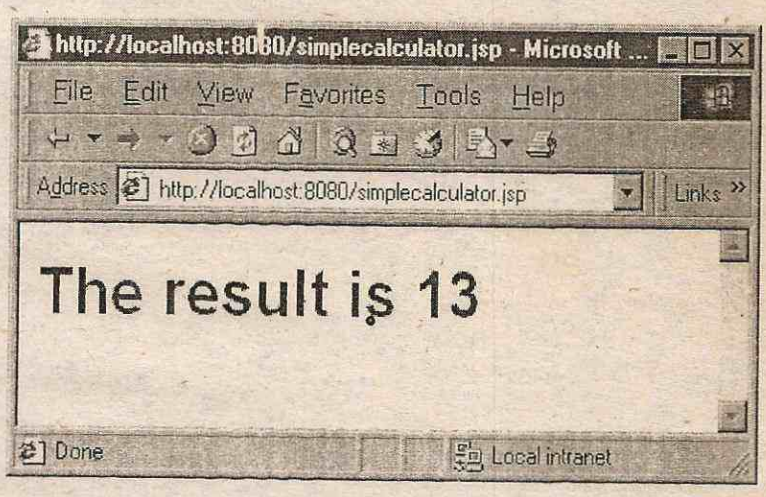
Bạn chép simplecalculator.jsp vào thư mục $web_server\public_html. Để kiểm tra bạn gõ địa chỉ URL như sau

http://localhost:8080/simplecalculator.jsp

kết quả trang web được hiển thị như hình 6-6. Ở bước này đối tượng RMI chưa được triệu gọi. Bạn cần nhập vào số cần tính toán ở hai ô textbox. Tiếp đến chọn toán tử từ danh sách, và nhấn nút '*Get Result*'. Khi bạn kích hoạt nút nhấn, trình duyệt sẽ gửi dữ lliệu về máy chủ. Do thuộc tính ACTION của thẻ <FORM> trong trang web trỏ đến simplecalculator.jsp nên trang simplecalculator.jsp được gọi lại. Trong lần thứ hai này trang simplecalculator.jsp thật sự triệu gọi đối tượng RMI sử dụng dữ liệu do trình khách gửi đến yêu cầu SmallCalculator tính toán kết quả và trả về cho trình khách (hình 6-7).



Hình 6-6 Kết xuất trang web simplecalculator.jsp lần đầu



Hình 6-7 Kết xuất trang simplecalculator sau khi triệu gọi đối tượng RMI

6. KẾT CHƯƠNG

Chương này đã giới thiệu với bạn kỹ thuật lập trình web bằng trang jsp (Java Server Page). Với jsp bạn có thể nhúng lệnh Java vào trang Web kết hợp với các thẻ HTML truyền thống. Jsp thay cho Servlet và dễ sử dụng hơn servlet. Tuy nhiên kết hợp giữa trang jsp và công nghệ component của servlet là cách tốt nhất để bạn xây dựng những ứng dụng web mạnh mẽ phát huy tác dụng ở cả hai phía khách/chủ.

Chương 7

JAVA MAIL

Tóm tắt chương 7

Chương này chúng ta sẽ dành nghiên cứu về đề tài gửi và nhận thư điện tử (e-mail) bằng Java. Ngày nay thư điện tử (gọi tắt là mail) gắn liền với sự phát triển của Internet. Với Java bạn hoàn toàn có đầy đủ công cụ và thư viện để tạo nên những ứng dụng gửi/nhận mail chuyên nghiệp.

Các vấn đề chính sẽ được đề cập đến:

- ✓ *Gửi mail bằng lớp đối tượng SmtpClient*
- ✓ *Tìm hiểu Java Mail API và các khái niệm*
- ✓ *Xây dựng chương trình Web Mail bằng servlet*
- ✓ *Web Mail kết nối với Yahoo Mail*

1. GỬI MAIL VỚI SMTPCLIENT

Java phiên bản 1.2 trở đi hỗ trợ cho bạn gói sun.net.smtp dùng để gửi mail rất thuận tiện. Gói sun.net.smtp không được công bố chính thức trong các tài liệu của Sun nhưng trong thư viện kèm theo JDK bạn sẽ tìm thấy gói này (bạn dùng Winzip mở tập tin jdk_home\jre\lib\rt.jar) Gói sun.net.smtp bao gồm 2 lớp chính là SmtpClient.class và SmtpPrintStream.class dùng để chuyển dữ liệu và gửi mail từ máy khách lên trình chủ mail theo giao thức SMTP (Simple Mail Transfer Protocol). Để gửi mail đến một máy chủ nào đó bạn phải cho biết: địa chỉ mail người gửi, địa chỉ mail người nhận, địa chỉ máy chủ (nơi chương trình quản lý mail hay còn gọi là mail server hoạt động). Dưới đây là một ví dụ sử dụng lớp SmtpClient gửi mail đến bất kỳ địa chỉ nào bạn muốn.

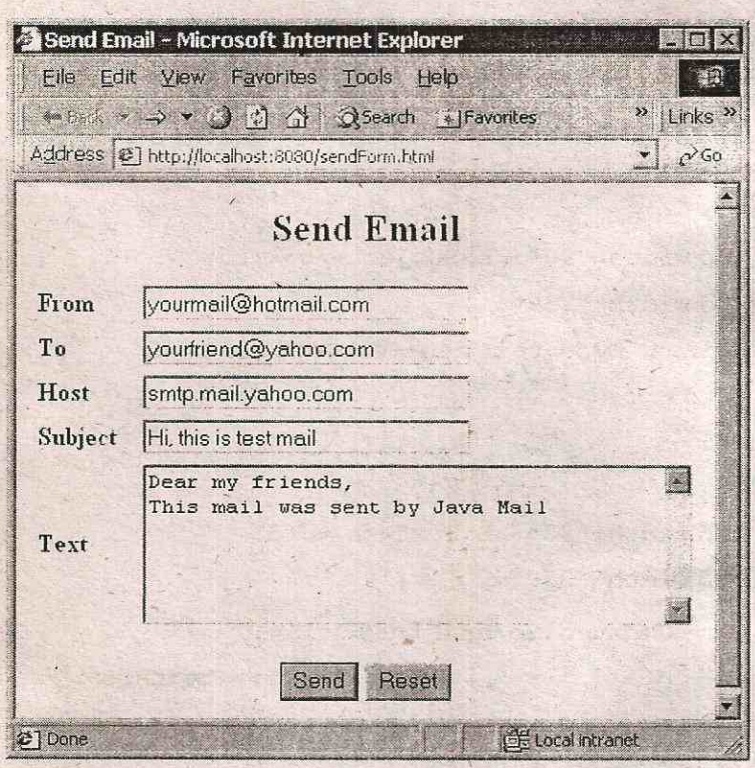
1.1. Gửi mail bằng servlet

Chúng ta thiết kế trang web tĩnh sendForm.html dùng để tiếp nhận dữ liệu gửi mail (như địa chỉ, nội dung mail ...) và servlet mang tên SendMail. Servlet SendMail sẽ tạo đối tượng SmtpClient yêu cầu SmtpClient chuyển nội dung mail lên máy chủ.

Ví dụ 7-1 sendForm.html

```
<html>
<head>
<title>Send Email</title>
<center><h2>Send Email</h2>
<form method=POST action="/servlet/SendMailServlet">
  <table>
   <tr>
    <td width="98"><b>From</b></td>
    <td width="259">
     <input type=text name=mail_from size=30>
    </td>
   </tr>
   <tr>
    <td height="20" width="98"><b>To</b></td>
    <td height="20" width="259">
     <input type=text name=mail_to size=30>
    </td>
   </tr>
   <tr>
    <td width="98"><b>Host</b></td>
```

```
    <td width="259">
     <input type=text name=mail_host size=30>
    </td>
   </tr>
   <tr>
    <td width="98"><b>Subject</b></td>
    <td width="259">
     <input type=text name=subject size=30>
    </td>
   </tr>
   <tr>
    <td width="98"><b>Text</b></td>
    <td width="259">
     <textarea name=mail_body cols=40
                                rows=6></textarea>
    </td>
   </tr>
  </table>
  <br>
<input type=submit value="Send">
<input type=reset value="Reset">
</form></center>
</body>
</html>
```



Hình 7-1 Trang web gửi mail hiển thị phí trình duyệt

Ví dụ 7-2 SendMailServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import sun.net.smtp.*;
public class SendMailServlet extends HttpServlet
{
 public static String MAIL_FROM = "mail_from";
 public static String MAIL_SUBJECT = "subject";
```

```
public static String MAIL_BODY = "mail_body";
public static String MAIL_TO = "mail_to";
public static String MAIL_HOST = "mail_host";
//Xử lý lệnh HTTP POST nhận  dữ liệu mail từ trình khách và gửi
// mail lên trình chủ mail server
public void doPost(HttpServletRequest req,
                   HttpServletResponse resp)
    throws ServletException, java.io.IOException
  {
    // Định nội dung MIME
    resp.setContentType("text/html");
    //Lấy về luồng xuất
    java.io.PrintWriter out =
      new java.io.PrintWriter(resp.getOutputStream());
    // Lấy về các địa chỉ mail
    String from = req.getParameter(MAIL_FROM);
    String to = req.getParameter(MAIL_TO);
    String subject = req.getParameter(MAIL_SUBJECT);
    String body = req.getParameter(MAIL_BODY);
    String host = req.getParameter(MAIL_HOST);
    try {
      // Tạo đối tượng SmtpClient để gửi mail
      SmtpClient mailer = new SmtpClient(host);
      // Đặt địa chỉ người gửi và địa chỉ người nhận
      mailer.from(from);
      mailer.to(to);
      // Tạo luồng để gửi dữ liệu mail
```

```
    java.io.PrintStream ps = mailer.startMessage();
    // Gửi phần tiêu đề mail
    ps.println("From: " + from);
    ps.println("To: " + to);
    ps.println("Subject: " + subject);
    // Gửi phần thân mail
    ps.println(body);
    // Gửi mail lên máy chủ
    mailer.closeServer();
    // Kết xuất thông báo mail đã gửi thành công
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Send Email</title>");
    out.println("<body><center>");
    out.println("<h2>Your email has been sent!</h2>");
    out.println("</center></body></html>");
  }
  catch (Exception ex) {
    // Kết xuất thông báo mail gửi bị lỗi
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Send Email Error</title>");
    out.println("<body><center>");
    out.println("<h2>Error sending your email</h2>");
    out.println("<br>Message=" + ex.getMessage());
    out.println("</center>");
```

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

Bạn chép tập tin sendForm.html vào thư mục $web_server\public_html. Biên dịch servlet gửi mail SendMailServlet.java. Tiếp đến đăng ký servlet này với trình chủ web JWS2.0. Bạn gõ địa chỉ URL kiểm tra như sau

http://localhost:8080/sendForm.html

kết quả trang web hiển thị như hình 7-1. Bạn nhập vào ô textbox **From** địa chỉ mail của bạn, ô **To** địa chỉ mail của người gửi, ô **Host** địa chỉ máy chủ của mail server (bạn có thể liên hệ với người quản trị mạng để biết được địa chỉ mail server. Ở đây nếu bạn có đăng ký và dùng dịch vụ mail của Yahoo bạn có thể gõ địa chỉ smtp.mail.yahoo.com, đây là địa chỉ thực tế của Yahoo mail). Ô **subject** là tiêu đề tóm tắt của mail. Ô **Text** bạn gõ vào nội dung mail.

Sau khi nhập liệu xong bạn nhấn nút *Send*. Thuộc tính ACTION=/servlet/SendMailServlet của thẻ <FORM> sẽ triệu gọi servlet SendMailServlet thực hiện việc gửi mail theo yêu cầu và dữ liệu do trình khách gửi lên.

Bạn lưu ý mã nguồn của trang web tĩnh sendForm.html. Các ô textbox trong trang web này được đặt tên như mail_from, mail_to, mail_body, mail_subject ... Khi servlet SendMailServlet được triệu gọi nó sẽ dựa và những tên này để lấy về dữ liệu của trình khách

```
public static String MAIL_FROM = "mail_from";
public static String MAIL_SUBJECT = "subject";
public static String MAIL_BODY = "mail_body";
```

```
...
String from = req.getParameter(MAIL_FROM);
String to = req.getParameter(MAIL_TO);
String subject = req.getParameter(MAIL_SUBJECT);
...
```

1.2. Gửi mail bằng trang jsp

Nếu thích viết JSP hơn servlet bạn có thể thay SendMailServlet bằng trang jsp sau

Ví dụ 7-3 sendMail.jsp

```
<HTML>
<%@ page import="sun.net.smtp.*" %>
<%
 String MAIL_FROM = "mail_from";
 String MAIL_SUBJECT = "subject";
 String MAIL_BODY = "mail_body";
 String MAIL_TO = "mail_to";
 String MAIL_HOST = "mail_host";
 //Lấy dữ liệu từ trình khách
 String from = request.getParameter(MAIL_FROM);
 String to = request.getParameter(MAIL_TO);
 String subject = request.getParameter(MAIL_SUBJECT);
 String body = request.getParameter(MAIL_BODY);
 String host = request.getParameter(MAIL_HOST);
 try {
     // Tạo đối tượng SmtpClient dùng gửi mail
    SmtpClient mailer = new SmtpClient(host);
```

```
    // Định địa chỉ người gửi và địa chỉ người nhận
    mailer.from(from);
    mailer.to(to);
    // Tạo luồng xuất để gửi thông điệp mail
    java.io.PrintStream ps = mailer.startMessage();
    // Xuất phần tiêu đề mail
    ps.println("From: " + from);
    ps.println("To: " + to);
    ps.println("Subject: " + subject);
    ps.println(); //Cách dòng trước khi gửi phần thân
    // Xuất phầnt thân mail
    ps.println(body);
    // Gửi mail đi
    mailer.closeServer();
    //Thông báo mail được gửi thành công
    out.println("<H1> Your mail has been
                                    send success <H1>");
  }catch (Exception ex) {
    //Thông báo mail gửi bị lỗi
    out.println("<H1> Mail Send Error <H1>");
  }
%>
</HTML>
```

Để gọi được trang sendMail.jsp bạn chép sendMail.jsp vào thư mục $web_server\public_html. Tiếp đến thay thuộc tính ACTION của thẻ <FORM> trong ví dụ 7-1 với nội dung sau

```
<form method=POST action="sendMail.jsp">
```

gõ lại địa chỉ URL: http://localhost:8080/sendForm.html để kiểm tra. Bạn cũng sẽ gửi được mail tương tự như khi dùng SendMailServlet.

Lưu ý, Bạn có thể sẽ phải thực hiện thêm thao tác phụ sau: mở tập tin rt.jar bằng trình WinZip (tập tin này nằm trong thư mục jdk_home\jre\lib\ hoặc $web_server\jre\lib). Bung các lớp SmtpClient.class, SmtpPrintStream.class, SmtpProtocolException.class vào thư mục $web_server\classes. Trình diễn dịch JSP của JWS2.0 khi biên dịch thường tìm các lớp thư viện trong thư mục $web_server\classes. Mặc định gói sun.net.smtp không nằm trong đường dẫn này. Bạn phải giúp trình chủ JWS2.0 thực hiện thủ công. Với servlet bạn không cần phải làm điều này.

2. GỬI/NHẬN MAIL BẰNG JAVA MAIL API

2.1. Các khái niệm Mail

SmtpClient là cách đơn giản nhất để bạn gửi mail đi. Trong trường hợp bạn muốn nhận mail về hay sử dụng các tính năng nâng cao của dịch vụ mail thì sao? Java cung cấp cho bạn một tập hợp các dịch vụ hay giao tiếp lập trình ứng dụng hạt nhân (API) phục vụ cho việc thiết kế các ứng dụng mail.

Trước khi bắt tay vào thiết kế chương trình web mail bằng Java Mail API ở phần sau, bạn nên biết qua một số khái niệm cơ bản trong lĩnh vực gửi nhận mail sau đây

- *Mail Server:* Theo mô hình hoạt động khách/chủ, trình chủ của bạn là một dịch vụ đóng vai trò người chủ phục vụ trình khách. Mail Server thật ra là một chương trình mở socket lắng nghe các yêu cầu (hay lệnh gửi mail) từ trình khách đưa đến. Web Server sẽ tiếp nhận nội dung mail, phân bố mail đến các máy chủ khác, cho phép trình khách truy cập vào máy chủ để nhận mail về, bảo vệ mail ... Chính vì vậy, trước khi nhận hay gửi mail bạn phải

biết rõ địa chỉ IP của máy chủ mail server. Địa chỉ này thường gọi là mail host.

- *Giao thức gửi mail (Mail transport protocol):* Để gửi mail đến máy chủ trình khách phải sử dụng một giao thức trò chuyện với mail server. Tương tự trình duyệt dùng giao thức HTTP để trò chuyện với trình chủ Web Server. Các trình khách muốn bắt tay với trình chủ mail server và gửi mail lên máy chủ sẽ sử dụng giao thức SMTP (Simple Mail Transport Protocol). SMTP khá nổi tiếng và được hầu hết các trình chủ mail trên thế giới sử dụng (như bạn thấy ở ví dụ 7-1 Yahoo mail cũng sử dụng giao thức gửi mail SMTP bằng địa chỉ smtp.mail.yahoo.com). Địa chỉ IP của máy chủ nhận mail gửi đi thường được gọi là *outgoing mail address*.

- *Giao thức lưu trữ mail (Mail store protocol):* Khi trình chủ mail server tiếp nhận được mail, trình chủ phải tiến hành lưu trữ mail theo cách nào đó để trình khách có thể dễ dàng truy cập vào và nhận mail về. Hiện nay POP3 (Post Office Protocol) và IMAP (Interner Message Access Protocol) là hai giao thức lưu trữ và lấy mail từ hộp thư (inbox) trên máy chủ khá phổ biến. Địa chỉ của máy chủ lưu trữ mail cho phép máy khách truy cập vào để nhận mail về được gọi là *incoming mail address*.

2.2. Thiết kế chương trình Web mail

Web mail là các chương trình gửi nhận mai trực tuyến (online) bằng internet. Mở trang web bằng trình duyệt, kết nối vào internet bạn có thể gửi mail đi và nhận mail về ở bất kỳ nơi nào. Yahoo Mail (của công ty Yahoo) và Hotmail (của Microsoft) là hai ứng dụng web mail rất nổi tiếng hiện nay.

Để xây dựng ứng dụng web mail bằng servlet hay jsp ngoài việc có trình chủ hiểu Java (như JWS2.0, JRun, WebLogic ...) bạn cần thêm hai thư viện hỗ trợ khác là mail.jar và activation.jar. Chúng tôi đã chép hai thư viện này trên đĩa CD bạn có thể tham khảo. Nếu

muốn cập nhật phiên bản mới nhất bạn có thể download trực tuyến tại địa chỉ http://java.sun.com/products/javamail của Sun.

2.3. Các bước cơ bản khi xây dựng ứng dụng web mail

Để đọc mail từ máy chủ, cơ bản bạn cần thực hiện các bước sau

- Tạo đối tượng mail Session (bạn lưu ý đối tượng Session ở đây dành cho ứng dụng mail, khác với HttpSession mà chúng ta đã tìm hiểu ở chương trước). Yêu cầu người dùng chọn các giao thức truy cập vào máy chủ lưu trữ mail (POP3 hay IMAP ...).
- Tạo đối tượng `Store` chuẩn bị lấy mail về. Để tạo được đối tượng `Store` bạn cần yêu cầu người dùng nhập vào tên và mật khẩu chứng thực quyền truy cập hộp thư hợp pháp.
- Sử dụng đối tượng `Store` để lấy về đối tượng `Folder` (Thực sự thì mail của bạn được lưu theo cấu trúc tương tự cấu trúc cây thư mục). Mỗi `Folder` sẽ chứa một loại thư mà bạn muốn lấy. Như `Folder` Inbox chứa các thư gửi đến, Folder Trash chứa các thư vừa bị xóa, `Folder` Sent chứa các thư gửi đi ...
- Duyệt đối tượng `Folder` để lấy về danh sách mail.

Để gửi mail đi, cơ bản bạn cần thực hiện các bước sau

- Yêu cầu người dùng chọn các giao thức gửi (SMTP) và lưu trữ mail (POP3 hay IMAP ...). Lưu các thông tin này vào đối tượng Session.
- Tạo đối tượng Message. Đối tượng này sẽ chứa các thông tin về mail cần gửi đi như địa chỉ người gửi, địa chỉ người nhận, tiêu đề mail, nội dung mail ...
- Gọi Transport.send() để gửi mail đi.

Trước khi cài đặt mã nguồn chi tiết cho chương trình web mail bạn hãy tắt dịch vụ httpd của JWS2.0. Chép mail.jar và

activation.jar vào thư mục C:\JavaWebServer\lib. Trên cửa sổ DOS-prompt nơi httpd chạy. Bạn đặt lại đường dẫn CLASSPATH như sau

```
set classpath=%classpath%;
        C:\JavaWebServer2.0\lib\mail.jar;
        C:\JavaWebServer2.0\lib\activation.jar
```

2.3.1. Xây dựng lớp SendMailUser

Chúng ta sẽ bắt đầu cài đặt ứng dụng web mail bằng việc thiết kế lớp SendMailUser. Lớp này cung cấp đầy đủ các thông tin kết nối và thực hiện quá trình đăng nhập vào máy chủ mail trước khi bạn có thể tiến hành các thao tác nhận hay gửi mail khác.

Ví dụ 7-4 SimpleMailUser.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.mail.*;
/**
 * Lớp này sẽ đóng gói các thông tin của người dùng khi đăng nhập (login)
 *  vào hộp thư. Bạn dùng SimpleMailUser để lưu trữ các thông tin như giao
 * thức gửi mail, giao thức nhận hoặc lưu trữ mail. Địa chỉ người gửi người
 * nhận ...
 */
public class SimpleMailUser
{
 Session m_session;
 boolean m_loggedIn;
 Store m_store;
 Folder m_inbox;
 String m_storeProtocol;
```

```
String m_storeHost;
String m_transportProtocol;
String m_transportHost;
String m_user;
String m_fromName;
/**
 * Phương thức khởi dựng SimpleMailUser
 */
public SimpleMailUser()
 {
   // Tạo đối tượng mail Session
   java.util.Properties props = System.getProperties();
   m_session = Session.getDefaultInstance(props, null);
 }
/**
 * Trả về đối tượng session cho biết thông tin về trạng thái kết
nối hiện hành
 */
public Session getSession()
 {
   return m_session;
 }
/**
 * Lấy về nội dung (hay folder) của hộp thư gửi đến (còn gọi là
Inbox)
 */
public Folder getInbox()
```

```
  {
    return m_inbox;
  }
  /**
   * Trả về true nếu người dùng đăng nhập thành công vào máy chủ
mail
   */
  public boolean isLoggedIn()
  {
    return m_loggedIn;
  }
  /**
   * Trả về giao thức nhận hay lưu trữ mail. Ví dụ giao thức POP3
   */
  public String getStoreProtocol()
  {
    return m_storeProtocol;
  }
  /**
   * Trả về địa chỉ mail của máy chủ
   */
  public String getStoreHost()
  {
    return m_storeHost;
  }
  /**
   * Trả về giao thức gửi mail (ví dụ SMTP)
```

```
 */
public String getTransportProtocol()
 {
  return m_transportProtocol;
 }
/**
 * Trả về địa chỉ của máy chủ nhận mail gửi đến
 */
public String getTransportHost()
 {
  return m_transportHost;
 }
/**
 * Trả về tên của người dùng hiện đang truy xuất hộp thư
 */
public String getUser()
 {
  return m_user;
 }
/**
 * Đặt địa chỉ gửi đến
 */
public void setFromName(String name)
 {
  m_fromName = name;
 }
/**
```

```
   * Trả về địa chỉ người gửi
   */
  public String getFromName()
   {
    return m_fromName;
   }
  /**
   * Trả về danh sách các giao thức gửi mail đi mà thư viện mail
cung cấp
   */
  public String[] getTransportNames()
   {
    return getProtocolNames(Provider.Type.TRANSPORT);
   }
  /**
   * Trả về danh sách các giao thức nhận mail về do thư viện mail
cung cấp
   */
  public String[] getStoreNames()
   {
    return getProtocolNames(Provider.Type.STORE);
   }
  /**
   * Đăng nhập vào máy chủ chứa mail
   * @store          Giao thức nhận hay lưu trữ mail của máy
chủ (POP3, IMAP)
   * @storeHost      Địa chỉ máy chủ lưu trữ mail
   * @transport      Giao thức chuyển mail
```

```
 * @transportHost      Địa chỉ máy chủ nhận mail
 * @user               Tên người dùng
 * @password           Mật khẩu đăng nhập
 */
public void login(String store, String storeHost,
        String transport, String transportHost,
        String user, String password)
                        throws MessagingException
 {
  try {
    // Đóng các thông tin kết nối trước đó. Tạo kết nối mới
   close();
    //Tạo thông tin về mail Session mới
   java.util.Properties props =
                        System.getProperties();
   props.put("mail.host", storeHost);
   m_session = Session.getDefaultInstance(props,
                                          null);
    // Tạo đối tượng URL lưu địa chỉ kết nối vào hộp thư
   URLName url = new URLName(store, storeHost, -1,
                "INBOX", user, password);
   m_store = m_session.getStore(url);
   m_store.connect();
//Yêu cầu lấy về đối tượng Folder (hộp thư In box) chứa danh sách
//các mail gửi đến
   m_inbox = m_store.getFolder("INBOX");
   m_inbox.open(Folder.READ_WRITE);
```

```
      // Đặt thuộc tính và các thông tin về giao thức và địa chỉ kết
nối
      m_storeProtocol = store;
      m_storeHost = storeHost;
      m_transportProtocol = transport;
      m_transportHost = transportHost;
      m_user = user;
      m_loggedIn = true;
    }
    finally {
      //Giải phóng tài nguyên nếu đăng nhập không thành công
      if (!m_loggedIn) {
       if (m_store != null) {
        m_store.close();
        m_store = null;
       }
      }
    }
  }
  /**
   * Trả về chuỗi mô tả tên của một giao thức dựa và số chỉ định
kiểu giao thức
   */
  protected String[] getProtocolNames(
                                    Provider.Type type)
  {
    // Tạo đối tượng mảng Vector
```

```
    java.util.Vector v = new java.util.Vector();
    // Lấy về danh sách các nhà cung cấp giao thức mail
    Provider p[] = m_session.getProviders();
    if (p != null) {
     for (int i = 0; i < p.length; i++) {
       // Lấy về kiểu giao thức chỉ định
      if (p[i].getType() == type) {
        // Lưu vào mảng
       v.addElement(p[i]);
      }
     }
    }
    // Chuyển số giao thức thành danh sách chuỗi chứa tên các giao
thức
    String s[] = new String[v.size()];
    for (int i = 0; i < v.size(); i++) {
      Provider prov = (Provider) v.elementAt(i);
      s[i] = prov.getProtocol();
    }
    return s;
   }
  /**
   * Đóng và giải phóng tất cả kết nối mail
   */
  public void close()
   {
    m_loggedIn = false;
```

```
    try {
      // Đóng hộp thư
      if (m_inbox != null) {
        m_inbox.close(false);
      }
      //Đóng kết nối
      if (m_store != null) {
        m_store.close();
      }
    }
    catch (MessagingException ex) {
      // Xử lý lỗi ở đây theo ý bạn
    }
    m_inbox = null;
    m_store = null;
  }
}
```

Hầu như việc cài đặt SimpleMailUser chủ yếu tập trung vào phương thức `login()`. Phương thức này sẽ cố gắng kết nối với máy chủ mail theo các giao thức chỉ định (cùng với tên username/password hợp lệ) và lưu lại trạng thái kết nối bằng đối tượng Session. Bên cạnh đó, SimpleMailUser cũng cung cấp cho bạn các phương thức để lấy về những thông tin hỗ trợ như giao thức gửi, giao thức nhận, thiết lập địa chỉ mail gửi đi ...

2.3.2. Xây dựng lớp SendMailReader

Lớp SendMailReader dưới đây là phần cài đặt chính của ứng dụng web mail. Đây là một servlet thực hiện hầu hết các chức năng

như: tạo màn hình đăng nhập, cho phép kết nối với trình chủ mail. Đọc mail từ hộp thư, cho phép gửi mail đến một địa chỉ chỉ định. Nếu bạn chưa quen với cơ chế gửi mail trên web bạn có thể vào site Yahoo hoặc Hotmail của Microsoft để tham khảo. SendMailReader có thể tóm tắt các phương thức cài đặt như sau

```
public class SimpleMailReader extends HttpServlet{
 //Đăng nhập và kết nối với máy chủ mail
 public void login()
 //Hiển thị danh sách các mail có trong hộp thư của bạn
 public void showInbox()
 //Hiển thị chi tiết nội dung mail
 public void showMessage()
 //Cho phép đọc các file đính kèm theo mail (attachment)
 public void showPart()
 //Tạo trang web thiết lập màn hình đăng nhập (login)
 public void createLoginForm()
 //Tạo trang web thiết lập màn hình gửi mail
 public void createWriteForm()
 //Thực hiện chức năng gửi mail đi
 public void sendMessage()
 //Tạo trang web hiển thị thông báo lỗi
 public void createErrorPage()
}
```

Khi bạn gửi mail đi ngoài việc nhập vào nội dung mail là các dòng văn bản (text) bạn còn có thể đính kèm các file như file hình ảnh, chương trình nhị phân ...Việc đọc mail sẽ bao gồm luôn cả quá trình phân tích nội dung các tập tin đính kèm. Khi hiển thị mail đọc được chúng ta chỉ hiển thị những nội dung ở dạng văn bản Phương thức showMessage() sẽ thực hiện điều này. Đối với các tập tin đính

kèm, phương thức showPart() sẽ chịu trách nhiệm đọc và trả dữ liệu là tập tin về cho máy khách (thường được xem như lấy file đính kèm về máy khách – download file). showInbox() được dùng để liệt kê danh sách các mail nhận được chứa trên máy chủ mail server. Ngoài ra, trước khi gửi hay nhận mail bạn phải thực hiện thao tác đăng nhập (còn gọi là login) vào máy chủ. Phương thức login() sẽ chịu trách nhiệm kết nối và chứng thực quyền truy cập của người dùng vào mail server (thông qua username/password). Phương thức createLoginForm() dùng tạo màn hình login (bao gồm các ô textbox để người dùng nhập vào username và password đồng thời chọn các giao thức gửi/nhận mail). Phương thức createWriteForm() sẽ tạo màn hình gửi mail. Bạn gọi phương thức sendMessage() để gửi mail đi. Phương thức createErrorPage() tạo ra trang web thông báo lỗi gặp phải trong quá trình gửi và nhận mail.

Chi tiết cài đặt các phương thức được trình bày trong ví dụ 7-5 dưới đây (bạn lưu ý chúng ta sử dụng lớp hỗ trợ SimpleMailUser đã được xây dựng ở ví dụ 7-4)

Ví dụ 7-5 SendMailReader.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.mail.*;
import javax.mail.internet.*;
/**
 * Servlet này đóng vai trò chính trong ứng dụng web mail.
   Servlet sẽ sinh ra các biểu mẫu (form) đăng nhập cho phép người
dùng kết nối với máy chủ mail server
*/
public class SimpleMailReader extends HttpServlet
{
 public static String MAIL_USER =
```

```
                                "SimpleMailReader.user";
public static String FORM_TITLE = "Simple Mail Reader";
public static String FORM_TRANSPORT = "transport";
public static String FORM_TRANSPORTHOST =
                                        "transportHost";
public static String FORM_STORE = "store";
public static String FORM_STOREHOST = "storeHost";
public static String FORM_USER = "user";
public static String FORM_PASSWORD = "password";
public static String FORM_ACTION = "action";
public static String FORM_MSG = "msg";
public static String FORM_PART = "part";
public static String FORM_FROM = "from";
public static String FORM_TO = "to";
public static String FORM_CC = "cc";
public static String FORM_SUBJECT = "subject";
public static String ACTION_LOGIN = "Login";
public static String ACTION_LOGOFF = "Logoff";
public static String ACTION_VIEW = "View";
public static String ACTION_WRITE = "New Message";
public static String ACTION_SEND = "Send";
/**
 * Thực hiện lệnh HTTP GET
 *
*/
public void doGet(HttpServletRequest req,
```

```
                    HttpServletResponse resp)
        throws ServletException, java.io.IOException
  {
    // Lấy về đối tượng Session quản lý kết nối của servlet
   HttpSession session = req.getSession(true);
    // Lấy về tên người dùng được phép đăng nhập (login) vào hộp
thư
   SimpleMailUser mailUser =
    (SimpleMailUser) session.getValue(MAIL_USER);
    // Tạo  mới đối tượng SimpleMailUser lưu các thông tin đăng
nhập mail
   if (mailUser == null) {
    mailUser = new SimpleMailUser();
    //Yêu cầu servlet Session lưu lại đối tượng mailUser
    session.putValue(MAIL_USER, mailUser);
   }
    //Kiểm tra xem người dùng đã đăng nhập vào hộp thư hay
chưa
   if (!mailUser.isLoggedIn()) {
        //Tạo màn hình login nếu chưa login trước đó
    createLoginForm(req, resp, mailUser);
   }
   else {
    // Người dùng đã đăng nhập hợp lệ vào hộp thư.
    // Kiểm tra thao tác cần phải đáp ứng trình khách
   String action = req.getParameter(FORM_ACTION);
    // Nếu trình khách muốn xem nội dung chi tiết của mail
```

```
if (ACTION_VIEW.equals(action)) {
  // Lấy về số thứ tự của thông điệp mail
 String s = req.getParameter(FORM_MSG);
 int msg = 0;
 try {
  msg = Integer.parseInt(s);
 }
 catch (Exception ex) {
    // Lỗi nếu s không phải là số
 }
  // Lấy về số thứ tự của phần nội dung đính kèm theo mail
 s = req.getParameter(FORM_PART);
 int part = -1;
 try {
  part = Integer.parseInt(s);
 }
 catch (Exception ex) {
    // Nếu không có tập tin đính kèm theo mail thì hiển thị
    // mail ở dạng văn bản text bình thường
 if (part == -1) {
  showMessage(req, resp, mailUser, msg);
 }
 else {
    //Hiển thị danh sách các tập tin đính kèm (attach file)
  showPart(req, resp, mailUser, msg, part);
 }
}
```

```
    else {
        //Hiển thị danh sách các mail trong hộp thư inbox
      showInbox(req, resp, mailUser);
    }
  }

 }
/**
 * Thực hiện lệnh HTTP POST
 */
public void doPost(HttpServletRequest req,
                   HttpServletResponse resp)
  throws ServletException, java.io.IOException
 {
  // Lấy về đối tượng Session quản lý kết nối của servlet
  HttpSession session = req.getSession(true);
// Lấy về tên người dùng được phép đăng nhập (login) vào hộp thư
  SimpleMailUser mailUser =
    (SimpleMailUser) session.getValue(MAIL_USER);
  // Khi phương thức doPost() được gọi chắc chắn người dùng đã đăng
  //nhập trước đó. Nếu không tìm thấy đối tượng mailUser lưu trong
  //session trước đó có nghĩa là session đã hết thời hạn sử dụng hoặc
  //người dùng cố tình đăng nhập không thông qua trang web login
  //do phương thức doGet() tạo ra
  if (mailUser == null) {
   createErrorPage(req, resp,
                   "Invalid session", null);
```

```
return;
}
// Kiểm tra thao tác cần phải đáp ứng trình khách
String action = req.getParameter(FORM_ACTION);
if (ACTION_LOGIN.equals(action)) {
  //Đăng nhập hay kết nối với máy chủ mail
 login(req, resp, mailUser);
}
else
  if (ACTION_LOGOFF.equals(action)) {
    // Người dùng thoát khỏi hộp thư
  mailUser.close();
    //Trở về màn hình login
  createLoginForm(req, resp, mailUser);
 }
 else
   if (ACTION_WRITE.equals(action)) {
   // Tạo màn hình hay trang web cho phép gửi mail đi
  createWriteForm(req, resp, mailUser);
 }
 else
    if (ACTION_SEND.equals(action)) {
   //Yêu cầu thao tác gửi mail thực hiện
  sendMessage(req, resp, mailUser);
 }
}
/**
```

```
 * Phương thức này sẽ thử đăng nhập (login) vào máy chủ chứa mail
 *
 * @req  Đối tượng req lấy từ servlet dùng để nhận dữ liệu do
        trình khách chuyển lên
 * @resp Đối tượng req lấy từ servlet dùng để trả kết xuất về cho
        trình khách
 * @mailUser Đối tượng lưu thông tin về người sử dụng mail
 */
public void login(HttpServletRequest req,
                  HttpServletResponse resp,
                  SimpleMailUser mailUser)
   throws ServletException, java.io.IOException
 {
  String transport = req.getParameter(FORM_TRANSPORT);
  String transportHost =
                req.getParameter(FORM_TRANSPORTHOST);
  String store = req.getParameter(FORM_STORE);
  String storeHost = req.getParameter(FORM_STOREHOST);
  String user = req.getParameter(FORM_USER);
  String password = req.getParameter(FORM_PASSWORD);
  // Sử dụng các giao thức mail do trình khách gửi lên cùng với thông
  //tin đăng nhập như username/password để kết nối với máy chủ mail
  try {
   mailUser.login(store, storeHost,
          transport, transportHost,
          user, password);
  }
```

```
  catch (MessagingException ex) {
    createErrorPage(req, resp, "Unable to login", ex);
    return;
  }
    //Hiển thị danh sách các mail có trong hộp thư inbox
  showInbox(req, resp, mailUser);
}
/**
 * Phương thức này được dùng để hiển thị danh sách các mail có trong hộp
 * thư inbox
 * @req  Đối tượng req lấy từ servlet dùng để nhận dữ liệu do trình
        khách chuyển lên
 * @resp Đối tượng req lấy từ servlet dùng để trả kết xuất về cho trình
khách
 * @mailUser Đối tượng lưu thông tin về người sử dụng mail
 */
public void showInbox(HttpServletRequest req,
            HttpServletResponse resp,
            SimpleMailUser mailUser)
  throws ServletException, java.io.IOException
  {
    // Định kiểu MIME trả về trình khách
  resp.setContentType("text/html");
      // Lấy về luồng xuất
  java.io.PrintWriter out = resp.getWriter();
  // Yêu cầu trình khách nạp lại trang web từ trình chủ. Không lấy lại
   //nội dung trang web từ cache của trình duyệt
  resp.setHeader("Expires",
```

```
                    "Tues, 01 Jan 1980 00:00:00 GMT");
 // Lấy về đường dẫn URL nơi xuất phát yêu cầu HTTP POST
 String uri = req.getRequestURI();
 // Kết xuất trang web
 out.println("<html>");
 out.println("<head>");
 out.println("<title>" + FORM_TITLE + "</title>");
 out.println("<body><center><h2>");
 //Thông tin về người đăng nhập
 out.println("INBOX for " + mailUser.getUser() +
        " using " + mailUser.getStoreProtocol() +
        " on host " + mailUser.getStoreHost());
 out.println("</h2>");
 try {
   // Lấy về đối tượng Folder (tượng trưng cho hộp thư)
  Folder inbox = mailUser.getInbox();
   // Lấy về số thư (mail) hiện có trong hộp thư (inbox)
  int n = inbox.getMessageCount();
   // Lấy về danh sách các thông điệp có trong hộp thư
  Message[] msgs = inbox.getMessages();
  out.println("Total number of messages: " +
                                  n + "<br>");
  out.println("<form action=\"" + uri + "\"
                          METHOD=\"POST\">");
  //Tạo nút nhấn cho phép người dùng thoát khỏi hộp thư
  out.println("<input type=submit name=" +
                  FORM_ACTION + " value=\"" +
```

```
                    ACTION_LOGOFF + "\">");
 //Tạo nút nhấn cho phép người dùng gọi form gửi mail
out.println("<input type=submit name=" +
                    FORM_ACTION + " value=\"" +
                    ACTION_WRITE + "\">");
out.println("</form>");
 //Hiển thị danh sách các mail nhận được trong hộp thư
out.println("<table border>");
out.println("<tr><th></th>");
out.println("<th>From</th>");
out.println("<th>Sent</th>");
out.println("<th>Subject</th></tr>");
 // Duyệt qua tất cả các thông điệp để in ra
for (int i = 0; i < n; i++) {
 Message m = msgs[i];
  // Bỏ qua các thông điệp mail đã đánh dấu xóa
 if (m.isSet(Flags.Flag.DELETED)) {
   continue;
 }
 out.println("<tr>");
  // Tạo đường nối kết (link) cho phép người dùng kích chuột
  // vào xem chi tiết mail
 out.println("<td><a href=\"" +
        uri + "?" + FORM_ACTION + "=" +
        ACTION_VIEW + "&" + FORM_MSG + "=" + i +
        "\">" + ACTION_VIEW + "</a>");
  //Lấy và hiển thị địa chỉ người gửi
```

```
    Address from[] = m.getFrom();
    Address addr = null;
    if ((from != null) && (from.length > 0)) {
     addr = from[0];
    }
    out.println("<td>" + getAddress(addr) + "</td>");
     // Hiển thị ngày gửi
    java.util.Date date = m.getSentDate();
    String s = "";
    if (date != null) {
     s = "" + date;
    }
    out.println("<td>" + s + "</td>");
     // Hiển thị tiêu đề mail (mail subject)
    s = m.getSubject();
    if (s == null) {
     s = "";
    }
    out.println("<td>" + s + "</td>");
    out.println("</tr>");
   }
   out.println("</table>");
  }
  catch (MessagingException ex) {
   out.println("<br>");
   out.println("ERROR: " + ex.getMessage());
```

```
    }
    // Hoàn tất kết xuất
    out.println("</center>");
    out.println("</body>");
    out.println("</html>");
    out.flush();
  }
/**
 * Phương thức này được dùng để hiển thị nội dung chi tiết của thông
 * điệp mail
 * @req  Đối tượng req lấy từ servlet dùng để nhận dữ liệu do trình
        khách chuyển lên
 * @resp Đối tượng req lấy từ servlet dùng để trả kết xuất về cho trình
khách
 * @mailUser Đối tượng lưu thông tin về người sử dụng mail
 * @msg  Số thứ tự của thông điệp mail cần hiển thị
 */
public void showMessage(HttpServletRequest req,
                        HttpServletResponse resp,
                        SimpleMailUser mailUser,
                        int msg)
      throws ServletException, java.io.IOException
  {
  //Định kiểu tài nguyên MIME trả về cho trình khách
    resp.setContentType("text/html");
    // Lấy về luồng xuất
    java.io.PrintWriter out = resp.getWriter();
    // Yêu cầu trình khách nạp lại trang web từ trình chủ. Không lấy lại
```

```
//nội dung trang web từ cache của trình duyệt
resp.setHeader("Expires",
                    "Tues, 01 Jan 1980 00:00:00 GMT");
// Lấy về đường dẫn URL nơi xuất phát yêu cầu HTTP POST
String uri = req.getRequestURI();
// Kết xuất trang web
out.println("<html>");
out.println("<head>");
out.println("<title>" + FORM_TITLE + "</title>");
out.println("<body>");
try {
  // Lấy về đối tượng Folder (tượng trưng cho hộp thư)
 Folder inbox = mailUser.getInbox();
  // Lấy về danh sách các thông điệp có trong hộp thư
 Message[] msgs = inbox.getMessages();
  //Tìm đến thông điệp cần xem chi tiết nội dung
 Message m = msgs[msg];
  // Hiển thị ngày
 out.println("Date: " + m.getSentDate() + "<br>");
  // Hiển thị địa chỉ người gửi
 Address a[] = m.getFrom();
 out.println("From: " + formatAddresses(a) + "<br>");
  // Hiển thị địa chỉ người nhận
 a = m.getRecipients(Message.RecipientType.TO);
 out.println("To: " + formatAddresses(a) + "<br>");
  // Hiển thị địa chỉ người được thông báo phụ
  // (còn gọi là cc hay carbon-copy)
```

```
a = m.getRecipients(Message.RecipientType.CC);
out.println("Cc: " + formatAddresses(a) + "<br>");
 //Hiển thị phần tiêu đề mail
String s = m.getSubject();
if (s == null) {
 s = "";
}
out.println("Subject: <b>" + s + "</b><br><hr>");
 // Hiển thị nội dung thông điệp
Object o = m.getContent();

 // Kiểm tra kiểu tài nguyên cần hiển thị phải là dạng văn bản
text
if (m.isMimeType("text/plain")) {
 out.println("<pre>" + o + "</pre>");
}
else
  //Nội dung thông điệp bao gồm nhiều  tập tin đính kèm
 if (m.isMimeType("multipart/*")) {
  // Lấy về các phần đính kèm
 Multipart mp = (Multipart) o;
  // Duyệt qua danh sách các file đính kèm
 for (int j = 0; j < mp.getCount(); j++) {
  Part part = mp.getBodyPart(j);
   // Lấy về kiểu nội dung file đính kèm
  String contentType = part.getContentType();
  if (contentType == null) {
```

```
        out.println("Bad content type for part " + j);
        continue;
      }
      ContentType ct = new ContentType(contentType);
      if (j != 0) {
        out.println("<hr>");
      }
        // Hiển thị các tập tin dạng văn bản
      if (ct.match("text/plain")) {
        out.println("<pre>" +
                part.getContent() + "</pre>");
      }
      else {
        //Đối với các tập tin nhị phân hay hình ảnh.
        //Tạo đường link cho phép người dùng kích chuột
download về
        String desc = "Attachment ";
        s = part.getFileName();
        if (s != null) {
          desc += s;
        }
        // Tạo đường link liên kết đến file đính kèm cần
download
        out.println("<td><a href=\"" +
              uri + "?" + FORM_ACTION + "=" +
              ACTION_VIEW + "&" + FORM_MSG + "=" + msg +
              "&" + FORM_PART + "=" + j +
```

```
            "\">" + desc + "</a>");
        }
      }
    }
    else {
      //Kiểu tài nguyên không biết
      out.println(m.getContentType());
    }
  }
  catch (MessagingException ex) {
    out.println("<br>");
    out.println("ERROR: " + ex.getMessage());
  }
  // Kết thúc kết xuất
  out.println("</body>");
  out.println("</html>");
  out.flush();
}
/**
 * Cho phép lấy về  các file đính kèm theo mail
 * @req  Đối tượng req lấy từ servlet dùng để nhận dữ liệu do
        trình khách chuyển lên
 * @resp Đối tượng req lấy từ servlet dùng để trả kết xuất về cho
        trình khách
 * @mailUser Đối tượng lưu thông tin về người sử dụng mail
 * @msg  Số thứ tự của thông điệp mail cần hiển thị
 * @part  Số thứ tự của tập tin đính kèm cần lấy
```

```
 */
public void showPart(HttpServletRequest req,
            HttpServletResponse resp,
            SimpleMailUser mailUser,
            int msg, int part)
 throws ServletException, java.io.IOException
 {
  // Lấy về luồng xuất
  java.io.PrintWriter out = resp.getWriter();
  try {
   // Lấy về đối tượng Folder (tượng trưng cho hộp thư)
   Folder inbox = mailUser.getInbox();
   // Lấy về danh sách các thông điệp có trong hộp thư
   Message[] msgs = inbox.getMessages();
   //Tìm đến thông điệp cần xem
   Message m = msgs[msg];
   // Tìm đến tập tin đính kèm cần lấy
   Multipart mp = (Multipart) m.getContent();
   Part p = mp.getBodyPart(part);
   // Lấy kiểu nội dung thể hiện của tài nguyên
   String contentType = p.getContentType();
   if (contentType == null) {
    out.println("Invalid message part at " + part);
   }
   else {
    ContentType type = new ContentType(contentType);
     // Định kiểu nội dung tài nguyên sẽ trả về cho trình khách
```

```
    resp.setContentType(type.getBaseType());
     // Đọc nội dung tập tin đính kèm vào luồng
    java.io.InputStream in = p.getInputStream();
    int b;
    while (true) {
     b = in.read();
     if (b == -1) {
      break;
     }
      //Ghi nội dung tậo tin đính kèm trả về trình khách
     out.write(b);
    }
   }
  }
  catch (MessagingException ex) {
   out.println("<br>");
   out.println("ERROR: " + ex.getMessage());
  }
  out.flush();
 }
 /**
  * Phân tích đối tượng Address trả về chuỗi mô tả tên người gửi
  */
 protected String getAddress(Address address)
  {
   String name = "";
```

```
    if (address != null) {
     name = "" + address;
     if (address instanceof InternetAddress) {
      String personal =
        ((InternetAddress) address).getPersonal();
      if (personal != null) {
          name = personal;
      }
     }
    }
    return name;
   }
  /**
   * Định dạng mảng chứa danh sách tên các địa chỉ
   */
  protected String formatAddresses(Address addresses[])
   {
    String s = "";
    if (addresses != null) {
     for (int i = 0; i < addresses.length; i++) {
      if (i > 0) {
       s += ", ";
      }
      s += getAddress(addresses[i]);
     }
    }
```

```
    return s;
  }
/**
 * Phương thức này sẽ kết xuất trang web tạo màn hình đăng nhập
 *
 * @req  Đối tượng req lấy từ servlet dùng để nhận dữ liệu do
        trình khách chuyển lên
 * @resp Đối tượng req lấy từ servlet dùng để trả kết xuất về cho
        trình khách
 * @mailUser Đối tượng lưu thông tin về người sử dụng mail
 */
public void createLoginForm(HttpServletRequest req,
                    HttpServletResponse resp,
                    SimpleMailUser mailUser)
  throws ServletException, java.io.IOException
  {
    //Định kiểu tài nguyên MIME trả về cho trình khách
    resp.setContentType("text/html");
    // Lấy về luồng xuất
    java.io.PrintWriter out = resp.getWriter();
    // Yêu cầu trình khách nạp lại trang web từ trình chủ. Không lấy lại
    //nội dung trang web từ cache của trình duyệt
    resp.setHeader("Expires",
                    "Tues, 01 Jan 1980 00:00:00 GMT");
    // Lấy về đường dẫn URL nơi xuất phát yêu cầu HTTP POST
    String uri = req.getRequestURI();
    // Kết xuất trang web
    out.println("<html>");
```

```
out.println("<head>");
out.println("<title>" + FORM_TITLE + "</title>");
out.println("</head>");
out.println("<body>");
out.println("<center><h2>Welcome to the
                                Simple Mail Reader");
out.println("<br><br>Please Log In</h2>");
out.println("<form ACTION=\"" + uri +
      "\" METHOD=\"POST\" ENCTYPE=" +
      "\"application/x-www-form-urlencoded\">");
out.println("<table>");
// Tạo danh sách cho phép chọn giao thức gửi mail
out.println("<tr><td width=\"30%\">
                Transport Protocol:</td>");
out.println("<td width=\"70%\"><select name=" +
              FORM_TRANSPORT + " size=1>");
String s[] = mailUser.getTransportNames();
for (int i = 0; i < s.length; i++) {
 out.println("<option value=\"" + s[i] + "\">" +
       s[i].toUpperCase());
}
out.println("</select></td></tr>");
//Tạo danh sách cho phép chọn các giao thức lưu trữ hay nhận mail
out.println("<tr><td width=\"30%\">
                        Store Protocol:</td>");
out.println("<td width=\"70%\"><select name=" +
```

```
                    FORM_STORE + " size=1>");
s = mailUser.getStoreNames();
for (int i = 0; i < s.length; i++) {
 out.println("<option value=\"" + s[i] + "\">" +
        s[i].toUpperCase());
}
out.println("</select></td></tr>");
// Tạo ô textbox cho phép nhập vào địa chỉ mail của máy chủ
out.println("<tr><td>Mail Host:</td>");
out.println("<td><input type=\"text\" name=" +
      FORM_STOREHOST + " size=40></td></tr>");
// Ô textbox yêu cầu nhập tên người dùng và mật khẩu để mở
hộp thư
out.println("<tr><td>User Name:</td>");
out.println("<td><input type=\"text\" name=" +
       FORM_USER + " size=40></td></tr>");
out.println("<tr><td>Password:</td>");
out.println("<td><input type=\"password\" name=" +
       FORM_PASSWORD + " size=40></td></tr>");
out.println("</table><br>");
// Nút nhấn kích hoạt việc gửi dữ liệu lên trình chủ.
out.println("<input type=\"submit\" value=\"" +
       ACTION_LOGIN + "\" name=" + FORM_ACTION + ">");
// Nút nhấn cho phép xóa dữ liệu nhập lại
out.println("<input type=\"reset\"
                        value=\"Reset\">");
out.println("</form></center>");
```

```
    out.println("</body>");
    out.println("</html>");
    out.flush();
  }
/**
 * Tạo màn hình cho phép gửi mail
 *
 * @req  Đối tượng req lấy từ servlet dùng để nhận dữ liệu do
        trình khách chuyển lên
 * @resp Đối tượng req lấy từ servlet dùng để trả kết xuất về cho
        trình khách
 * @mailUser Đối tượng lưu thông tin về người sử dụng mail
 */
public void createWriteForm(HttpServletRequest req,
                            HttpServletResponse resp,
                            SimpleMailUser mailUser)
  throws ServletException, java.io.IOException
  {
    //Định kiểu tài nguyên MIME trả về cho trình khách
    resp.setContentType("text/html");
    // Lấy về luồng xuất
    java.io.PrintWriter out = resp.getWriter();
    //Kết xuất trang web
    out.println("<html>");
    out.println("<head>");
    out.println("<title>" + FORM_TITLE + "</title>");
    out.println("<center><h2>New Message</h2>");
```

```
out.println("<br><form method=POST action=\"" +
      req.getRequestURI() + "\">");
 // Lấy lại tên người nhận sau cùng mà người dùng gửi mail đi
String fromName = mailUser.getFromName();
if (fromName == null) {
 fromName = "";
}
out.println("<table>");
 // Ô textbox cho phép nhập vào địa chỉ người gửi
out.println("<tr><td>From:</td>");
out.println("<td><input type=text name=" +
      FORM_FROM + " size=40 value=\"" + fromName +
      "\"></td></tr>");
 // Địa chỉ người nhận
out.println("<tr><td>To:</td>");
out.println("<td><input type=text name=" +
      FORM_TO + " size=40></td></tr>");
 // Địa chỉ người được gửi thêm
out.println("<tr><td>Cc:</td>");
out.println("<td><input type=text name=" +
      FORM_CC + " size=40></td></tr>");
 // Ô textbox cho phép nhập vào tiêu đề mail
out.println("<tr><td>Subject:</td>");
out.println("<td><input type=text name=" +
      FORM_SUBJECT + " size=40></td></tr>");
 //Ô văn bản cho phép nhập vào nội dung mail
out.println("<tr><td>Message:</td>");
```

```
    out.println("<td><textarea name=" + FORM_MSG +
            " cols=40 rows=6></textarea></td></tr>");
    out.println("</table><br>");
    //Nút nhấn kích hoạt thao tác gửi mail
    out.println("<input type=submit name=" +
            FORM_ACTION + " value=\"" +
            ACTION_SEND + "\">");
    //Nút nhấn cho phép xóa dữ liệu nhập lại
    out.println("<input type=reset value=\"Reset\">");
    out.println("</form></center></body></html>");
    // Kết thúc kết xuất
    out.println("</body>");
    out.println("</html>");
    out.flush();
  }
  /**
   * Thực hiện thao tác gửi mail
   *
   * @req  Đối tượng req lấy từ servlet dùng để nhận dữ liệu do
          trình khách chuyển lên
   * @resp Đối tượng req lấy từ servlet dùng để trả kết xuất về cho
          trình khách
   * @mailUser Đối tượng lưu thông tin về người sử dụng mail
   */
  public void sendMessage(HttpServletRequest req,
                HttpServletResponse resp,
                SimpleMailUser mailUser)
```

```
throws ServletException, java.io.IOException
{
  //Định kiểu tài nguyên MIME trả về cho trình khách
  resp.setContentType("text/html");
  // Lấy về luồng xuất
  java.io.PrintWriter out = resp.getWriter();
  // Kết xuất trang web
  out.println("<html>");
  out.println("<head>");
  out.println("<title>" + FORM_TITLE + "</title>");
  out.println("<body>");
  // Lấy về các thông tin địa chỉ
  String from = req.getParameter(FORM_FROM);
  String to = req.getParameter(FORM_TO);
  String cc = req.getParameter(FORM_CC);
  String subject = req.getParameter(FORM_SUBJECT);
  String msg = req.getParameter(FORM_MSG);
  // Kiểm tra địa chỉ khác rỗng
  if ((from == null) || (from.trim().length() == 0)) {
   out.println("'From' name must be given");
  }
  else if ((to == null) || (to.trim().length() == 0)) {
   out.println("'To' address(es) must be given");
  }
  else {
   try {
     // Tạo đối tượng thông điệp
```

```
Message m = new
        MimeMessage(mailUser.getSession());
InternetAddress[] iAddr = null;
 // Định địa chỉ mail của người gửi
m.setFrom(new InternetAddress(from));
 //Lưu lại địa chỉ mail của người gửi
mailUser.setFromName(from);
 // Định địa chỉ mail của người nhận
iAddr = InternetAddress.parse(to, false);
m.setRecipients(Message.RecipientType.TO, iAddr);
 // Định địa chỉ mail gửi thêm
if ((cc != null) && (cc.trim().length() > 0)) {
 iAddr = InternetAddress.parse(cc, false);
 m.setRecipients(
        Message.RecipientType.CC, iAddr);
}
 // Định tiêu đề mail
m.setSubject(subject);
 // Định phần thân hay nội dung mail
m.setText(msg);
 // Chuyển hay gửi mail đi
Transport.send(m);
out.println("Message sent!");
}
catch (Exception ex) {
out.println("Unable to send message: " +
                        ex.getMessage());
```

```
    }
  }
      // Kết thúc
  out.println("</body>");
  out.println("</html>");
  out.flush();
}
/**
 * Tạo trang thông báo lỗi
 */
public void createErrorPage(HttpServletRequest req,
                HttpServletResponse resp,
                String msg,
                Exception ex)
 throws ServletException, java.io.IOException
 {
  //Định kiểu tài nguyên MIME trả về cho trình khách
  resp.setContentType("text/html");
  resp.setHeader("Expires",
                "Tues, 01 Jan 1980 00:00:00 GMT");
   // Lấy về luồng xuất
  java.io.PrintWriter out = resp.getWriter();
   // Kết xuất trang web
  out.println("<html>");
  out.println("<head>");
  out.println("<title>" + FORM_TITLE + "</title>");
  out.println("</head>");
```

```
    out.println("<body>");
    out.println("<center><h2>" + msg + "</h2>");
    out.println("</center>");
    if (ex != null) {
     out.println("<br>Stack Trace:<br>");
     ex.printStackTrace(out);
    }
     // Kết thúc kết xuất
    out.println("</body>");
    out.println("</html>");
    out.flush();
   }
  /* Phương thức khởi tạo servlet */
  public void init(ServletConfig cfg)
                    throws ServletException
   {
    super.init(cfg);
   }
  /* Phương thức hủy servlet */
  public void destroy()
   {
    super.destroy();
   }
}
```

Quá trình cài đặt SendMailReader đã hoàn tất. Bạn hãy chép SendMailReader.java và SimpleMailUser.java vào thư mục C:\JavaWebServer2.0\servlets\. Thực hiện biên dịch

```
C:\JavaWebServer2.0\servlets>javac SimpleMailUser.java
C:\JavaWebServer2.0\servlets>javac SendMailReader.java
```

Tiếp đến, đăng ký servlet SendMailReader với trình chủ JWS2.0 (bạn xem lại cách đăng ký servlet ở chương 4). Cuối cùng ta đã có một chương trình web mail khá hoàn chỉnh.

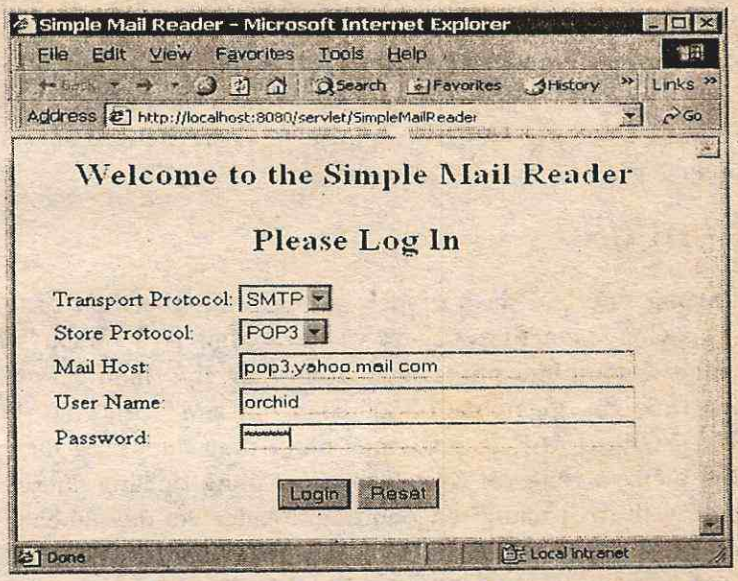
2.3.3. Sử dụng web mail

Để sử dụng chương trình web mail bạn gõ địa chỉ URL triệu gọi servlet SendMailReader như sau

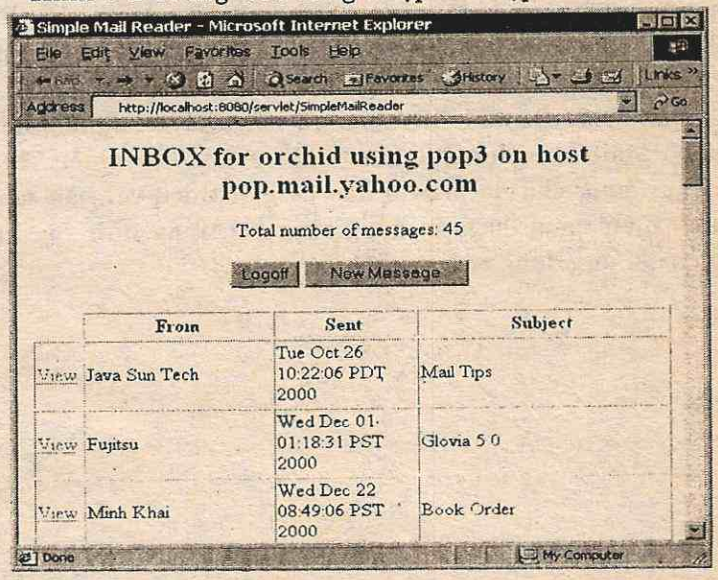
http://localhost:8080/servlet/SendMailReader

Trang web đăng nhập sẽ hiển thị trên trình duyệt như hình 7-2. Từ trang web này bạn có thể gửi và nhận mail hầu như với tất cả các máy chủ mail trên thế giới (nếu bạn kết nối với Internet), hoặc các máy chủ mail trong nội bộ công ty bạn (Intranet).

Vấn đề quan trọng chỉ là biết địa chỉ mail server và các giao thức chuyển/nhận mail mà máy chủ phục vụ. Ví dụ để nhận mail từ hộp thư Yahoo Mail bạn có thể chọn SMTP từ danh sách *Transport Protocol*. Chọn POP3 từ danh sách *Store Protocol*. Đánh vào địa chỉ máy chủ pop.yahoo.mail.com ở ô *Mail Host* (đây là địa chỉ máy chủ của Yahoo mail mà chúng tôi sưu tầm được). Gõ tên đăng nhập (username) và mật khẩu (password) của Yahoo (nếu chưa có bạn có thể vào thẳng site Yahoo đăng ký miễn phí). Nhấn nút *Login*. Nếu bạn kết nối Internet và thông tin đăng nhập hợp lệ bạn sẽ mở được hộp thư Inbox của Yahoo. Kết quả danh sách các mail trong hộp thư sẽ được hiển thị như hình 7-3. Nếu muốn xem chi tiết hay nội dung mail bạn có thể kích chuột vào liên kết View bên cột trái.



Hình 7-2 Trang web đăng nhập vào hộp thư (inbox)



Hình 7-3 Kết quả lấy mail từ Yahoo theo giao thức POP3

Nếu muốn gửi mail bạn hãy kích chuột vào nút nhấn *New Message* trên trang web. Trình chủ sẽ trả về trang web khác cho phép bạn nhập vào thông tin gửi mail. Nhập vào địa chỉ người gửi, người nhận, nội dung mail và nhấn nút *Send*. Thông điệp mail của bạn sẽ được gửi đến mọi nơi trên thế giới.

KẾT CHƯƠNG

Chương này bạn đã học cách gửi và nhận thư điện tử (e-mail) bằng thư viện Java SmtpClient và Java Mail API. Muốn gửi hay nhận mail thường bạn phải biết địa chỉ máy chủ (mail server) cộng với giao thức gửi, giao thức nhận. Hiện nay SMTP là giao thức gửi mail còn POP3, IMAP là giao thức nhận (hay đọc) mail được dùng khá phổ biến và rộng rãi trên Internet. Chúng ta cũng đã xây dựng ứng dụng web mail mang tên SendMailReader với đầy đủ chức năng đọc, gửi mail ngay trên trang web thông qua servlet. Bạn có thể sử dụng các hàm Java Mail API mà SendMailReader đã dùng để viết hẳn một ứng dụng mail riêng biệt không cần đến trình chủ Web hay servlet. Tuy nhiên ưu điểm của web mail là có thể truy cập mọi lúc mọi nơi đơn giản chỉ bằng trình duyệt. Bạn có thể tìm hiểu sâu hơn nữa và phát triển SendMailReader trở nên chuyên nghiệp hơn (hãy lấy Yahoo Mail hay Hotmail làm ví dụ mẫu). Tuy nhiên chúng tôi muốn tạm dừng chủ đề mail ở đây để giới thiệu với bạn một công nghệ mới hiện đang được áp dụng rộng rãi trong lĩnh vực Internet và phát triển ứng dụng web – XML.

Chương 8

XML

Tóm tắt chương 8

Chương này sẽ giới thiệu và giúp bạn tìm hiểu về ngôn ngữ định nghĩa cấu trúc và lưu trữ dữ liệu XML. XML cũng không kém phần hấp dẫn so với HTML. Nếu như HTML mạnh về định dạng dữ liệu thì XML lại thiên về mô tả dữ liệu.

Các vấn đề chính sẽ được đề cập đến:

- ✓ *Khái niệm cơ bản về XML*
- ✓ *Cú pháp XML*
- ✓ *XML và cơ sở dữ liệu, lưu trữ thông tin.*
- ✓ *Xử lý dữ liệu XML.*
- ✓ *HTML, XML và XSL*

1. KHÁI NIỆM CƠ BẢN VỀ XML

Trong hầu hết các ngôn ngữ lập trình việc mô tả cấu trúc dữ liệu thường đóng vài trò chủ yếu để xây dựng nên ứng dụng. Ví dụ, đối tượng Memo chứa thông tin về địa chỉ e-mail của một cá nhân được mô tả bằng ngôn ngữ Java như sau

```
public class Memo{
 public String to;
 public String from;
 public String cc;
 public String subject;
 public String body;
}
```

Hay trong C bạn có thể diễn đạt Memo bằng cấu trúc

```
struct Memo{
  char* to;
  char* from;
  char* cc;
  char* subject;
  char* body;
}
```

Với Pascal/Delphi cấu trúc Memo được biểu diễn theo cách

```
Memo = class
  to   :string;
  from:string;
  cc:string;
  subject:string;
  body:string;
end;
```

Nếu diễn đạt bằng ngôn ngữ SQL của cơ sở dữ liệu bạn có thể tạo ra bảng dữ liệu Memo tương ứng sau:

```
CREATE TABLE MEMO( to      VARCHAR(20),
                   from    VARCHAR(20),
                   cc      VARCHAR(20),
                   subject VARCHAR(50),
                   body    VARCHAR(200));
```

Còn rất nhiều cách biểu diễn cấu trúc dữ liệu khác được áp dụng trong các hệ quản trị dữ liệu, trong các trình ứng dụng. Tuy nhiên chúng không được thống nhất và thường chỉ có trình ứng dụng nào tạo ra định dạng của cấu trúc dữ liệu thì mới có khả năng xử lý cấu trúc dữ liệu đó. Nếu bạn muốn chuyển đổi dữ liệu qua lại giữa các trình ứng dụng thường chúng ta phải qua công đoạn xuất (export) từ trình ứng dụng này và nhập (import) dữ liệu lại vào trình ứng dụng khác. Với điều kiện cả hai trình ứng dụng đều cho phép chuyển đổi và nhập vào khuôn dạng dữ liệu thích hợp.

XML (Extension Makup Language) ra đời nhằm giúp việc mô tả dữ liệu trở nên đơn giản và dễ xử lý hơn. XML mô tả dữ liệu dựa trên việc dùng các thẻ định nghĩa (tương tự thẻ của ngôn ngữ định dạng HTML). Ví dụ thông tin về đối tượng Memo có thể được diễn đạt bằng ngôn ngữ XML như sau

```
<MEMO>
 <TO>        </TO>
 <FROM>      </FROM>
 <CC>        </CC>
 <SUBJECT>   </SUBJECT>
 <BODY>      </BODY>
</MEMO>
```

Bạn có thể tự hỏi, vậy thì dữ liệu được truyền vào cấu trúc mô tả XML được thực hiện như thế nào? Dữ liệu XML được đặt giữa hai thẻ đóng và thẻ mở. Ví dụ nếu ngôn ngữ Java gán dữ liệu cho đối tượng Memo theo cách:

```
public class printObject{
   public static void main(String args[]){
 Memo  m=new Memo();
      m.to      ="Jody";
      m.from    ="Bill";
```

```
        m.cc      ="Philip";
        m.subject ="Subject document";
        m.body    ="Hello, World!";
      }
    }
```

Dữ liệu của XML được đặt trong các thẻ tương ứng như sau

```
<MEMO>
 <TO>      Jodie</TO>
 <FROM>    Bill</FROM>
 <CC>      Philip</CC>
 <SUBJECT> Subject document</SUBJECT>
 <BODY>    Hello, World!</BODY>
</MEMO>
```

Bạn lưu ý tên các thẻ XML là do bạn tự ý đặt, điều này cũng tương tự như việc bạn tự đặt tên cho các trường (field) của bảng dữ liệu.

2. CÚ PHÁP XML

Tên các thẻ XML là do bạn tự đặt theo nhu cầu của mình. Tuy nhiên các thẻ phải tuân theo một số qui định đơn giản của chuẩn XML.

2.1. Thẻ đóng và thẻ mở

Mỗi thẻ XML bao gồm thẻ đóng và thẻ mở (tương tự thẻ HTML). Các trình dịch và phân tích nguồn XML thường phân biệt chữ hoa và chữ thường trong việc phân biệt tên thẻ. Ví dụ

```
<NAME> Your name </NAME>
```

trong đó <NAME> chính là thẻ mở còn </NAME> là thẻ đóng. Chuỗi Your Name đặt giữa thẻ đóng và thẻ mở chính là dữ liệu thể hiện của thẻ. Tuy nhiên nếu bạn viết

```
<NAME> Your name </name>
```

thì trình dịch XML sẽ bắt lỗi lý do <NAME> là thẻ mở nhưng </name> không phải là thẻ đóng của <NAME> (name chữ thường).

2.2. Thẻ rỗng

Nếu bạn chỉ muốn mô tả cấu trúc dữ liệu mà chưa có dữ liệu đặt giữa hai thẻ XML thì bạn có thể sử dụng thẻ rỗng rút gọn. Ví dụ, khai báo

```
<NAME></NAME>
```

có thể thay thế bằng <NAME/>. Chẳng hạn đối tượng MEMO khai báo bằng XML có thể mô tả ngắn gọn như sau

```
<MEMO>
 <TO/>
 <FROM/>
 <CC/>
 <SUBJECT/>
 <BODY/>
</MEMO>
```

2.3. Thuộc tính của thẻ

Một thẻ còn có thể chứa thêm khai báo thuộc tính đặc thù cho thẻ. Ví dụ trong ngôn ngữ HTML bạn thường thấy thẻ <APPLET> chuẩn được khai báo như sau

```
<APPLET class="HelloWorld.class" width=100 height=300>
```

```
</APPLET>
```

trong đó class, width, height chính là thuộc tính của thẻ.

XML cũng cho bạn tự qui định thuộc tính thể hiện của thẻ. Ví dụ đối tượng quả táo (Apple) được định nghĩa bằng XML có thuộc tính weight, price như sau

```
<APPLE weight="13" price="1.5 USD">
    This is a big Apple
</APPLE>
```

3. XEM DỮ LIỆU XML BẰNG TRÌNH DUYỆT IE

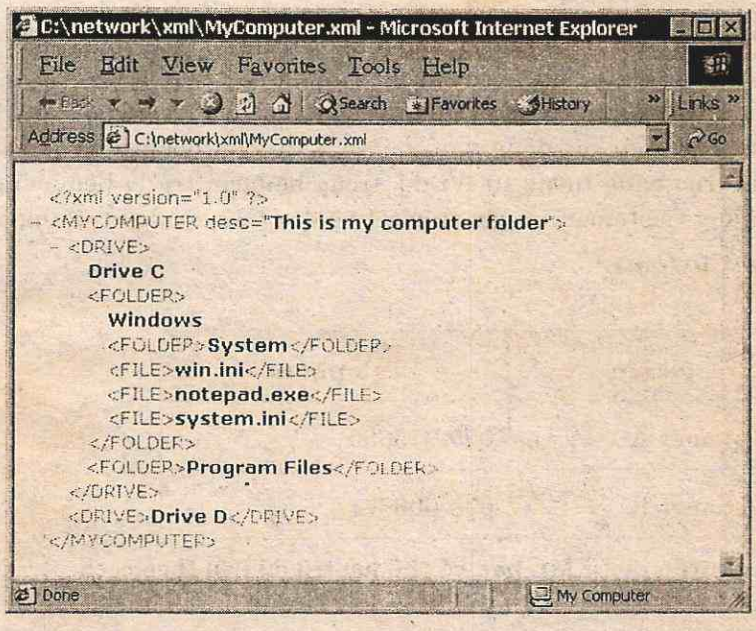
Thật sự thì các thẻ XML được xây dựng theo cấu trúc hình cây, thẻ con nằm trong lòng thẻ cha và phân cấp sâu tùy ý. Trình duyệt Internet Explorer cho phép bạn xem dữ liệu XML ngay trong trang Web.

Ví dụ cấu trúc cây thư mục của máy tính bạn có thể được biểu diễn bằng dữ liệu XML sau đây

Ví dụ 8-1 MyComputer.xml

```
<?xml version="1.0"?>
<MYCOMPUTER desc="This is my computer folder">
 <DRIVE> Drive C
  <FOLDER>
    Windows
    <FOLDER>System </FOLDER>
    <FILE> win.ini   </FILE>
    <FILE> notepad.exe </FILE>
    <FILE> system.ini </FILE>
```

```
    </FOLDER>
    <FOLDER>
        Program Files
    </FOLDER>
  </DRIVE>
  <DRIVE> Drive D
  </DRIVE>
</MYCOMPUTER>
```



Hình 8-1 Trình duyệt IE hiển thị dữ liệu XML

Bạn lưu tập tin MyComputer.xml xuống đĩa cứng, dùng trình duyệt IE mở tập tin này. Căn cứ vào đuôi tập tin .xml, trình duyệt

IE sẽ hiển thị cấu trúc dữ liệu XML như hình 8-1. Bạn có thể kích chuột vào các nhánh phân cấp để thu gọn hay bung các thẻ con bên trong thẻ cha.

Trình duyệt IE bắt lỗi rất chặt chẽ nếu bạn viết cú pháp XML sai. Đối với trang HTML, nếu gặp các thẻ bất thường hoặc không hiểu trình dịch HTML (thường là trình duyệt) sẽ bỏ qua. Đối với XML quá trình diễn dịch hoặc phân tích dòng dữ liệu sẽ dừng lại khi một thẻ viết không đúng cú pháp.

Tuy không yêu cầu nhưng khai báo thẻ <?xml version="1.0"?> sẽ giúp cho trình duyệt cũng như các công cụ phân tích dữ liệu XML nhận diện được phiên bản đặc tả XML mà dữ liệu thể hiện.

4. XML VÀ CƠ SỞ DỮ LIỆU

Trong các hệ cơ sở dữ liệu thường bạn lưu dữ liệu ở dạng bảng (table) bao gồm các dòng và cột. Với XML bạn hoàn toàn có thể mô tả cấu trúc bảng tương tự. Ví dụ, trong hệ quản trị dữ liệu mang tên DB, bảng Customer, có cấu trúc cột và dòng dữ liệu lưu trữ như sau

Bảng Customer

Name	Telephone
Customer A	090-715650
Customer B	090-606671

Chuyển qua XML bạn có thể lưu trữ dữ liệu theo cách

Ví dụ 8-2 Customer.xml

```
<xmlDB>
```

```
 <customer>
   <name>Customer A</name>
   <telephone> 090-715650 </ telephone >
   <name>Customer B</name>
   <telephone >091-606671 </telephone >
 </customer>
</xmlDB>
```

Không chỉ một, bạn còn có thể định nghĩa nhiều bảng dữ liệu cùng lúc bên trong tập tin XML.

5. XML THAY THẾ CHO CÁC TẬP TIN CẤU HÌNH

Hầu như XML là cách tốt nhất để bạn diễn đạt thông tin lưu trữ nhất quán và dễ hiểu. Ngày nay các trình ứng dụng đều có xu hướng sử dụng cấu trúc XML để lưu trữ thông tin cấu hình thay cho những tập tin .ini cũ. Ví dụ tập tin win.ini của Windows trước đây được lưu trữ ở dạng văn bản (text file) như sau

Ví dụ 8-3 win.ini

```
[windows]
load=
run=
[Desktop]
Wallpaper=(None)
TileWallpaper=1
WallpaperStyle=0
Pattern=(None)
[intl]
```

```
sCurrency=$
sDate=/
sDecimal=.
sLongDate=dddd, d MMMM yyyy
sShortDate=MM/dd/yyyy
[Fonts]
[FontSubstitutes]
Helv=MS Sans Serif
Tms Rmn=MS Serif
Times=Times New Roman
[TrueType]
FontSmoothing=1
```

Bạn có thể thay thế định dạng của tập tin win.ini trên đây bằng mô tả dữ liệu theo cấu trúc XML sau

Ví dụ 8-4 win.xml

```
<winconfig>
<windows>
 <load/>
 <run/>
</windows>
<Desktop>
<Wallpaper> (None) </Wallpaper>
<TileWallpaper> 1 </TileWallpaper>
<WallpaperStyle>0 </WallpaperStyle>
```

```
<Pattern> (None) </Pattern>
</Desktop>
<intl>
<sCurrency> $ </sCurrency>
<sDate> / </sDate>
<sDecimal> . </sDecimal>
<sLongDate> dddd, d MMMM yyyy </sLongDate>
<sShortDate> MM/dd/yyyy </sShortDate>
</intl>
<Fonts>
<FontSubstitutes>
<Font Helv="MS Sans Serif" />
<Font TmsRmn="MS Serif" />
<Font Times="Times New Roman" />
</FontSubstitutes>
<TrueType>
<FontSmoothing> 1 </FontSmoothing>
</TrueType>
</Fonts>
</winconfig>
```

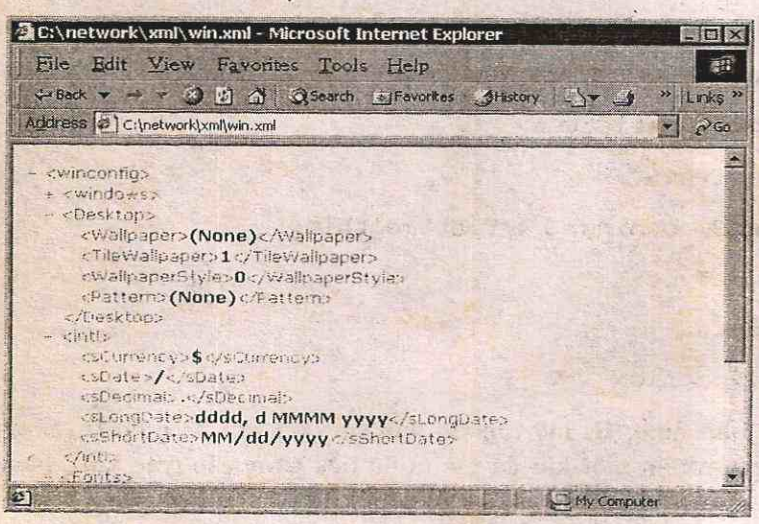
Bạn dùng IE mở tập tin win.xml kết quả sẽ hiển thị như hình 8-2. Dĩ nhiên một khi đã mô tả dữ liệu bằng cấu trúc XML bạn phải có cách nào đó truy xuất và đọc được dữ liệu chứa trong cấu trúc này. Microsoft trong kiến trúc Windows từ phiên bản 2000 trở đi hỗ trợ rất mạnh khả năng đọc và phân tích dữ liệu XML. Các nhà phát

triển ứng dụng cũng ngày càng hướng đến việc cung cấp công cụ xử lý XML trở nên đa dạng và phong phú hơn.

Nếu bạn đã từng thấy sự thay thế ngoạn mục của ngôn ngữ định dạng trang HTML thay cho các tài liệu như Word, RTF (rich text format). Có lẽ trong tương lai bạn sẽ thấy XML thay thế hay bổ sung cho các kiến trúc định dạng dữ liệu đơn giản như Access hay FoxPro. XML không định thay thế toàn bộ nhưng ít nhất XML cũng đem lại các mô tả và định nghĩa dữ liệu tốt hơn những bậc tiền bối của nó.

6. ĐỌC VÀ XỬ LÝ DỮ LIỆU XML

Có lẽ minh họa thú vị nhất là kết hợp được dữ liệu XML vào trang Web HTML. Ví dụ dưới đây sẽ cho phép bạn gọi đối tượng ActiveX (do Windows hỗ trợ), sử dụng mã Java Script đọc và phân tích dữ liệu của XML. Dữ liệu sau khi đọc được sẽ đưa vào các trường thể hiện để HTML hiển thị.



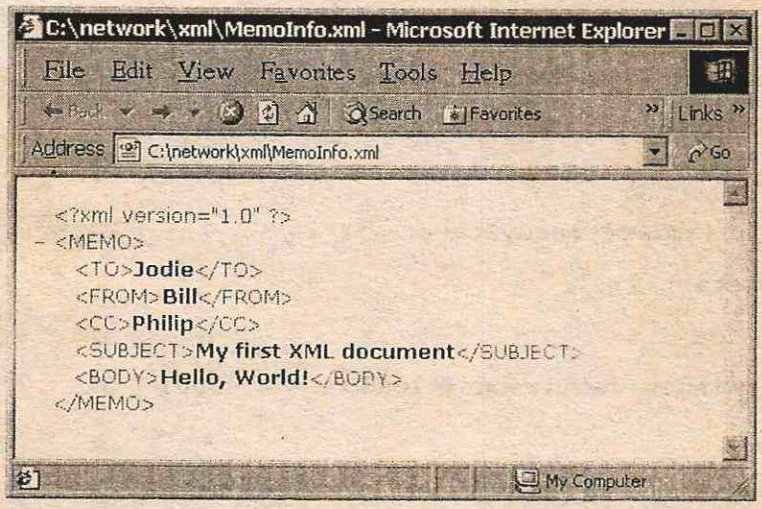
Hình 8-2 Nội dung tập tin cấu hình win.xml

Trước hết ta xây dựng dữ liệu XML là đối tượng Memo như ví dụ 8-5 sau

Ví dụ 8-5 MemoInfo.xml

```
<?xml version="1.0"?>
<MEMO>
 <TO >      Jodie</TO>
 <FROM>     Bill</FROM>
 <CC>       Philip</CC>
 <SUBJECT>My first XML document</SUBJECT>
 <BODY>     Hello, World!</BODY>
</MEMO>
```

Bạn lưu tập tin MemoInfo.xml vào thư mục C:\Network\xml. Cấu trúc của MemoInfo.xml được trình duyệt thể hiện ở hình 8-3.



Hình 8-3 Dữ liệu thể hiện trong MemoInfo.xml

Tiếp theo bạn tạo tập tin ReadMemo.htm đọc và hiển thị dữ liệu của MemoInfo.xml như sau

Ví dụ 8-6 Readmemo.htm

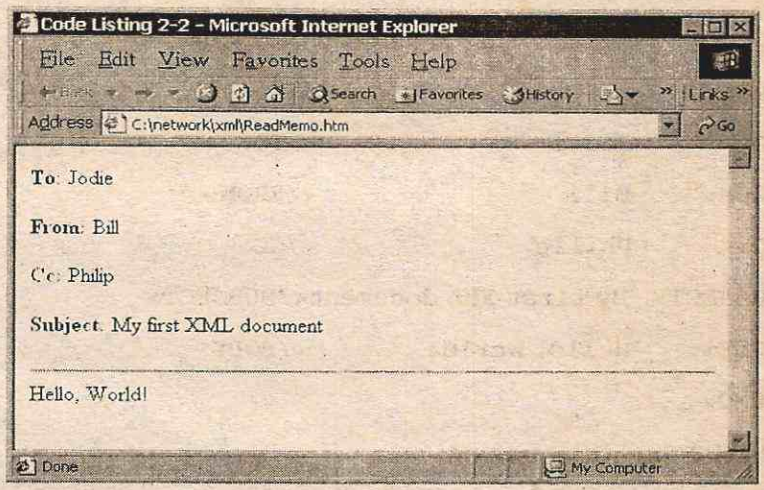
```
<HTML>
 <HEAD>
  <SCRIPT LANGUAGE="JavaScript"
                    FOR=window EVENT=onload>
   loadDoc();
  </SCRIPT>
  <SCRIPT LANGUAGE="JavaScript">
   var rootElem;
   //Tạo đối tượng ActiveX có khả năng phân tích dữ liệu XML
   var xmlDoc = new ActiveXObject("microsoft.xmldom");
   //Nạp dữ liệu XML
   xmlDoc.load("MemoInfo.xml");
   function loadDoc()
    {
    if (xmlDoc.readyState == "4")
     start();
    else
     window.setTimeout("loadDoc()", 4000)
    }
   function start()
    {
```

```
    //Phân tích dữ liệu XML. Bắt đầu từ thẻ trên cùng (root
element)
    rootElem = xmlDoc.documentElement;
    todata.innerText =
        rootElem.childNodes.item(0).text;
    fromdata.innerText=
        rootElem.childNodes.item(1).text;
    ccdata.innerText=
        rootElem.childNodes.item(2).text;
    subjectdata.innerText=
        rootElem.childNodes.item(3).text;
    bodydata.innerText=
        rootElem.childNodes.item(4).text;
    }
  </SCRIPT>
  <TITLE>Code Listing 2-2</TITLE>
</HEAD>
<BODY>
  <DIV ID="to" STYLE="font-weight:bold;font-size:16">
   To:
   <SPAN ID="todata"
          STYLE="font-weight:normal"></SPAN>
  </DIV>
  <BR>
```

```
<DIV ID="from" STYLE="font-weight:bold;
                                font-size:16">
 From:
 <SPAN ID="fromdata"
          STYLE="font-weight:normal"></SPAN>
</DIV>
<BR>
<DIV ID="cc" STYLE="font-weight:bold;
                              font-size:16">
 Cc:
 <SPAN ID="ccdata"
         STYLE="font-weight:normal"></SPAN>
</DIV>
<BR>
<DIV ID="subject"
             STYLE="font-weight:bold;font-size:16">
 Subject:
 <SPAN ID="subjectdata"
             STYLE="font-weight:normal"></SPAN>
</DIV>
<BR>
<HR>
<SPAN ID="bodydata"
            STYLE="font-weight:normal"></SPAN>
```

```
</BODY>
</HTML>
```

Bạn lưu ReadMemo.htm vào cùng thư mục C:\Network\xml. Dùng trình duyệt IE mở trang Web này bạn sẽ thấy dữ liệu của tập tin MemoInfo.xml hiển thị kết hợp với các thẻ HTML như hình 8-4



Hình 8-4 Kết hợp dữ liệu XML và định dạng HTML

7. HTML, XML VÀ XSL

HTML là ngôn ngữ định dạng văn bản dựa trên các thẻ được dùng khá phổ biến và rộng rãi. Tuy nhiên dữ liệu mà trang HTML thể hiện cùng với định dạng văn bản thường trộn lẫn vào nhau. XML mở ra hướng tách biệt dữ liệu khỏi trang định dạng HTML. Bạn định nghĩa tập dữ liệu trong file XML. Đồng thời bạn xây dựng bộ khung (layout template) trình bày cho dữ liệu chứa trong tập tin XSL (Extention Style Language). XSL chứa các qui ước và dạng hiển thị dữ liệu của XML. Kết hợp giữa XML và XSL cuối cùng sẽ tạo nên trang web hiển thị theo định dạng HTML. Nếu bạn đã sử dụng MS Word

thì khái niệm này tương tự việc trộn dữ liệu vào các mẫu (template) của tài liệu. Dưới đây là ví dụ về sự kết hợp giữa XML và XSL.

Ví dụ 8-7 MemoInfo.xml

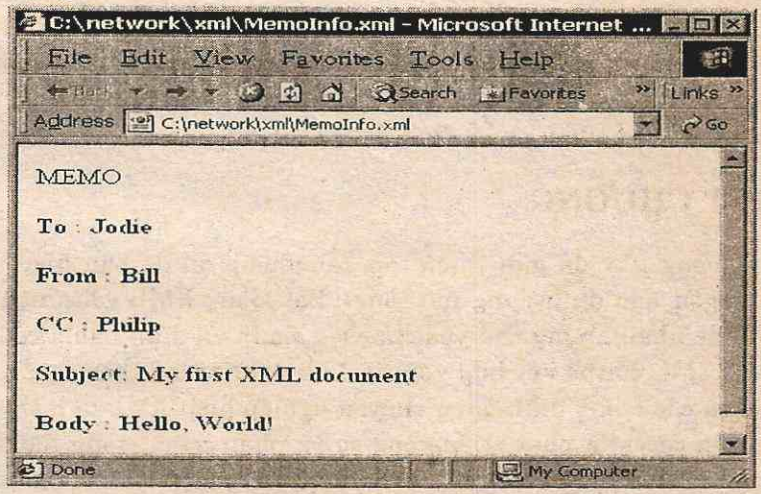
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
                    href="layoutmemo.xsl"?>
<MEMO>
 <TO >       Jodie                  </TO>
 <FROM>      Bill                   </FROM>
 <CC>        Philip                 </CC>
 <SUBJECT>  My first XML document</SUBJECT>
 <BODY>      Hello, World!          </BODY>
</MEMO>
```

Ví dụ 8-8 layoutmemo.xsl

```
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
 <BODY>
  MEMO <br></br>
  <xsl:for-each select="MEMO">
    <STRONG>
      To      :  <xsl:value-of select="TO"/>  <br></br>
      From    :  <xsl:value-of select="FROM"/> <br></br>
      CC      :  <xsl:value-of select="CC"/>
                                              <br></br>
      Subject:  <xsl:value-of select="SUBJECT"/>
```

```
                                                    <br></br>
      Body  :  <xsl:value-of select="BODY"/> <br></br>
    </STRONG>
   </xsl:for-each>
  </BODY>
</HTML>
```

Bạn lưu tập tin MemoInfo.xml và layoutmemo.xsl vào cùng thư mục. Dùng trình duyệt IE mở trang MemoInfo.xml, kết quả sẽ được định dạng và hiển thị như hình 8-5.



Hình 8-5 Kết xuất của XML và XSL

Như bạn thấy tuy đọc tập tin dữ liệu xml nhưng IE không hiển thị trực tiếp dữ liệu của file MemoInfo.xml theo cấu trúc cây. Do thẻ lệnh qui định

```
<?xml-stylesheet type="text/xsl"
                    href="layoutmemo.xsl"?>
```

IE sẽ kết hợp dữ liệu của MemoInfo.xml với định dạng qui ước của tập tin layoutmemo.xsl để tạo nên kết xuất tương tự trang HTML. Các thẻ trong tập tin layoutmemo.xsl như

```
<xsl:for-each select="MEMO">
    <xsl:value-of select="TO"/>
    ...
</xsl:for-each>
```

Chính là qui định của ngôn ngữ định kiểu XSL duyệt và đọc dữ liệu từ XML.

XSL làm cho quá trình định dạng trang tách biệt khỏi dữ liệu. Bạn có thể dùng nhiều trang XSL khác nhau kết hợp với một tập tin dữ liệu XML để sinh ra hàng loạt các trang web có cùng nội dung dữ liệu nhưng khác nhau phần định dạng.

8. KẾT CHƯƠNG

Chương này đã giới thiệu tóm tắt nhưng đủ để bạn hiểu rõ về XML (ngôn ngữ định dạng mở rộng). Bạn dùng XML với nhiều mục đích khác nhau nhưng chủ yếu cho việc mô tả và định nghĩa cấu trúc dữ liệu. XML có thể kết hợp với các định dạng trang theo chuẩn XSL để tạo ra các trang định dạng chuyên nghiệp hơn. XML là hướng tập trung của các nhà phát triển công cụ và phần mềm trong tương lai. Chúng tôi tạm dừng về đề tài XML ở đây để giới thiệu với bạn trong chương sau một kiến trúc đối tượng Java mới ra đời và hiện rất được rất nhiều lập trình viên Java chú ý đến - Kiến trúc phân tầng Enterprise Java Bean (EJB).

Chương 9

ENTERPRISE JAVA BEAN (EJB)

Tóm tắt chương 9

Chương này sẽ trình bày với bạn mô hình lập trình mạng khá mới mẻ, lập trình ứng dụng đa tầng Enterprise Java Bean (EJB). Bạn sẽ được tìm hiểu một số khái niệm cũng như kỹ thuật lập trình mới về kiến trúc phân tầng đang được đưa vào sử dụng rất rộng rãi trong nền công nghiệp phần mềm hiện nay. Có từ phiên bản Java 1.2, EJB đã được Sun hoàn thiện và chính thức xem như một phần của nền Java 2 trong bộ JDK 1.3 và các phiên bản sau này.

Các vấn đề chính sẽ được đề cập đến:

- ✓ *Mô hình khách/chủ và mô hình đa tầng trong Enterprise Java Bean*
- ✓ *Tìm hiểu mô hình hoạt động của Bean bên trong trình chứa (container).*
- ✓ *Xây dựng và triệu gọi các thành phần Bean EJB*
- ✓ *Sử dụng trình đóng gói deploytool*
- ✓ *EJB hiện tại và tương lai*

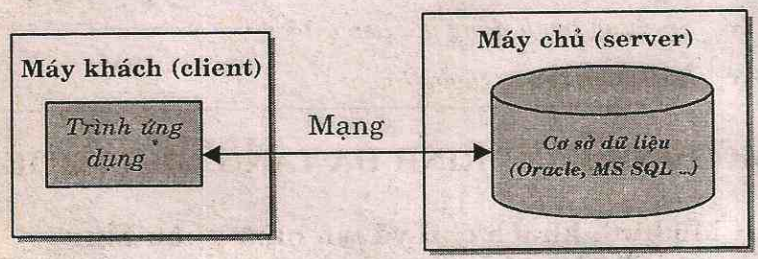
1. MÔ HÌNH LẬP TRÌNH ĐA TẦNG (MULTI-TIER)

1.1 Mô hình khách/chủ và mô hình phân tầng

Sự phát triển như vũ bão của hệ thống mạng nhất là mạng Internet đã khiến cho máy tính trở nên gần gũi và phục vụ đắc lực cho con người hơn. Tuy nhiên bên cạnh đó một thách thức lớn đặt ra cho các nhà phát triển ứng dụng là lập trình mạng ngày càng trở

nên phức tạp. Mô hình lập trình đơn lẻ truyền thống đã bị thay đổi rất nhiều. Ngày nay, bạn không còn đơn thuần ngồi viết những ứng dụng để chạy trên một máy duy nhất. Chương trình ứng dụng đòi hỏi sự tương tác từ nhiều phía người dùng, chia sẻ tài nguyên, triệu gọi từ xa, liên kết giao tác, phân tán dữ liệu ... Với những yêu cầu trên mô hình khách/chủ (hay client/server) đã ra đời và tồn tại rất lâu (thậm chí cho đến ngày nay). Theo mô hình khách/chủ tất cả các thao tác xử lý phức tạp đều được chuyển giao cho máy chủ xử lý. Máy khách chỉ đóng vai trò gửi yêu cầu và hiển thị dữ liệu. Điển hình của mô hình này là các ứng dụng cơ sở dữ liệu. Máy khách của bạn là trình ứng dụng có khả năng kết nối và truy vấn dữ liệu từ một máy chủ ở xa (như các máy chủ quản lý cơ sở dữ liệu Oracle, MS SQL Server, DB2 ...) – Hình 9-1.

Mô hình khách chủ đã tỏ ra hiệu quả trong suốt thời gian dài. Mặc dù vậy cả máy khách và máy chủ ngày càng bị quá tải bởi độ phức tạp và nhu cầu của người dùng. Các trình ứng dụng của máy khách trở nên cồng kềnh với vô số trình điều khiển phải cài đặt để kết nối với cơ sở dữ liệu của máy chủ (như ODBC, Oracle Client Driver, MS SQL Driver, MS Access Driver ...). Hơn nữa bất kỳ dịch vụ nào của trình chủ đều yêu cầu trình khách phải cài đặt một đơn thể kết nối tương ứng.



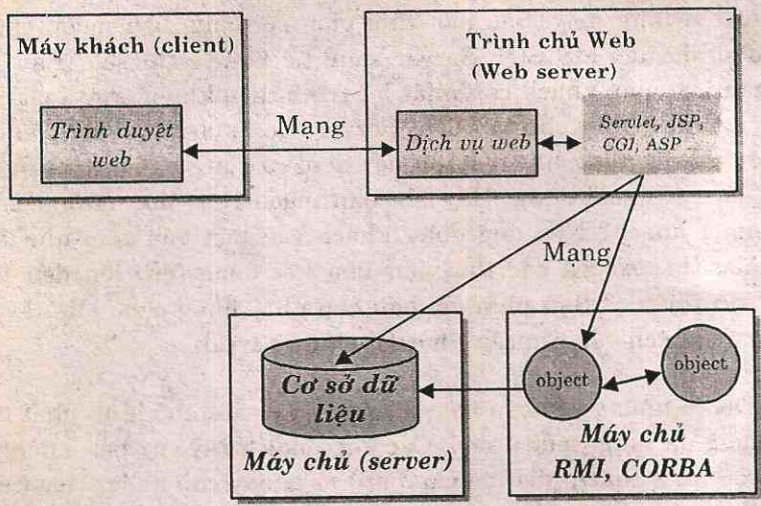
Hình 9-1. Mô hình khách/chủ (client/server)

Hãy hình dung bạn quản lý và phân phối ứng dụng đến 100 máy khách nằm ở nhiều quốc gia khác nhau (một trở ngại về mặt địa lý). Bạn cài đặt trình điều khiển kết nối cơ sở dữ liệu MS SQL

Server lên từng máy. Sau một thời gian sử dụng, bạn muốn chuyển hệ cơ sở dữ liệu MS SQL Server sang hệ Oracle thì sao? Bạn phải yêu cầu 100 máy khách cập nhật lại trình điều khiển truy xuất cơ sở dữ liệu Oracle thay cho MS SQL Server. Đó là chưa kể việc cấu hình và kết nối với từng hệ cơ sở dữ liệu xem ra cũng không phải là đơn giản đối với người dùng. Hay khi bạn muốn thay đổi mã nguồn của ứng dụng khách? Nếu ứng dụng khách của bạn bao gồm nhiều tập tin thực thi .exe và các thư viện liên kết động (dll) lên đến hàng chục Mb thì sao? Bạn phải gửi bản cập nhật hàng chục Mb này đến 100 máy khách yêu cầu cập nhật lại chương trình.

Tất cả những yêu cầu trên đã làm phát sinh mô hình phát triển ứng dụng đa tầng (multi-tier). Các ứng dụng xử lý của bạn không cài đặt trên máy khách nữa mà cài đặt ở một máy chủ khác. Mục tiêu là làm cho máy khách trở nên gọn nhẹ, dễ cấu hình, dễ thay đổi phía máy chủ. Nếu bạn cần thay đổi mã nguồn của trình ứng dụng bạn chỉ cần thay đổi trên một máy chủ. Tất cả các trình khách khi kết nối vào máy chủ chạy ứng dụng sẽ luôn được máy chủ phục vụ phiên bản chương trình mới nhất.

Web là một ví dụ điển hình nhất của mô hình ứng dụng đa tầng. Trình chủ Web nằm trên một máy chủ, trình khách chỉ cần dùng trình duyệt (browser) kết nối vào máy chủ và có thể truy cập được mọi thông tin cũng như dịch vụ. Trình chủ sử dụng java servlet, trang jsp hay các ứng dụng cgi để kết nối với cơ sở dữ liệu, đối tượng phân tán RMI, CORBA (nằm trên một máy chủ khác) xử lý tính toán và trả kết quả về cho trình khách thông qua các trang web tĩnh (hình 9-2). Mô hình đa tầng đã phân rã chức năng một cách cụ thể, cho phép ứng dụng chạy trên nhiều máy chủ khác nhau. Dễ dàng cho người phát triển, nhà cung cấp cũng như người sử dụng trên mạng diện rộng nhất là mạng Internet. Đó là lý do tại sao các nhà phát triển đều mong muốn hướng đến việc thiết kế ứng dụng web.



Hình 9-2. Mô hình ứng dụng Web

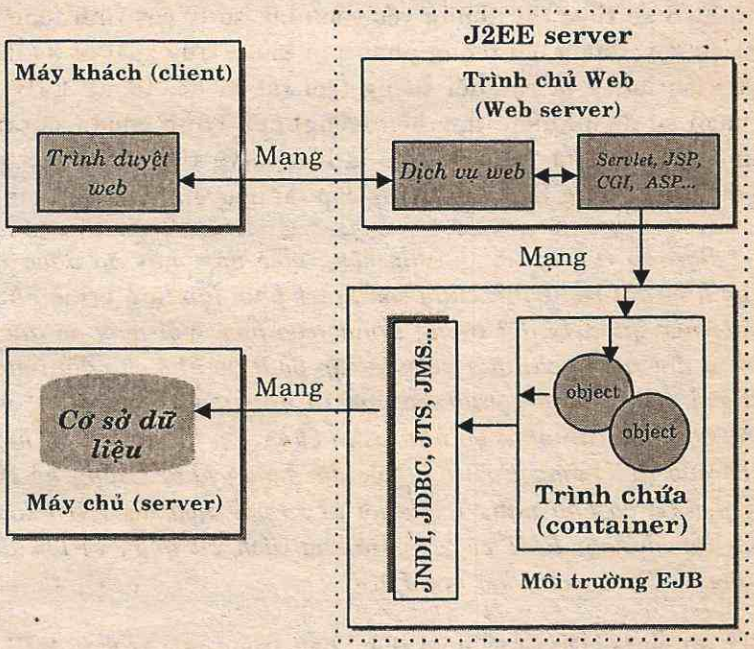
1.2. Mô hình Enterprise Java Bean

Để xây dựng các ứng dụng đa tầng chúng ta cần phải có cơ sở hạ tầng như: sự hỗ trợ của hệ thống mạng, hệ điều hành, ngôn ngữ lập trình, các công cụ phát triển, môi trường mà ứng dụng hoạt động ... Sun muốn tích hợp những công nghệ mạng của ứng dụng đa tầng vào nền Java một cách hoàn chỉnh hơn nữa. Như bạn thấy ở hình 9-2, trang jsp hay servlet có thể triệu gọi đối tượng phân tán RMI hay CORBA, truy xuất cơ sở dữ liệu từ nhiều máy chủ khác nhau. Bản thân đối tượng RMI hay CORBA cũng có khả năng tương tác với các hệ cơ sở dữ liệu độc lập.

Ở chương 2 và 3 chúng ta đã có dịp nghiên cứu chi tiết về lập trình với đối tượng phân tán RMI và CORBA. Các đối tượng phân tán mà bạn thiết kế trong chương đầu hoạt động rất độc lập, không có một sự kết hợp hoặc ràng buộc nào với các đối tượng khác. RMI chỉ là cơ sở hạ tầng để phát triển các đối tượng triệu gọi nhau từ xa. Các đối tượng muốn hoạt động tốt và hiệu quả hơn cần tồn tại trong

một môi trường có khả năng kiểm soát chúng. Hãy hình dung, nếu đối tượng của bạn là một chương trình thì nó cần một môi trường là hệ điều hành. Hệ điều hành sẽ cung cấp tài nguyên, kiểm soát quá trình lưu trữ, thực thi, chấm dứt của chương trình (thường quá trình này gọi là chu trình hoạt động của đối tượng - life circle).

Tương tự, Sun muốn xây dựng mô hình trong đó các đối tượng được sự hỗ trợ tối đa của môi trường mà nó tồn tại. Đó là mô hình Enterprise Java Bean (EJB). EJB là mô hình giúp các đối tượng Bean hoạt động kết hợp với nhau một cách có định hướng và hiệu quả hơn.



Hình 9-3. Mô hình ứng dụng phân tầng EJB

Sun đưa ra đặc tả về môi trường mà các đối tượng Bean sẽ hoạt động gọi là *trình chứa* (container). Các trình chứa sẽ kiểm soát việc khởi tạo, cung cấp tài nguyên cho đối tượng, lưu trữ, phục hồi đối

tượng ... Bên cạnh đó, các trình chứa container hoạt động bên trong một môi trường dịch vụ khác cao hơn đó là EJB. Java gọi môi trường bao trùm sự tương tác của các thành phần này là nền chủ J2EE (Java 2 Platform Enterprise Edition).

Hình 9-3 là mô hình phân tầng J2EE của ứng dụng mạng. Trình chủ kết nối với dịch vụ Web (dịch vụ cung cấp giao tiếp theo giao thức HTTP). Dịch vụ web sẽ thay mặt trình khách triệu gọi các ứng dụng web như CGI, Servlet hoặc trang web điều khiển jsp. Các thành phần điều khiển web này (web component) sẽ triệu gọi đối tượng object trong môi trường EJB. Các đối tượng object trong môi trường EJB sẽ thực thi những yêu cầu như: xử lý các tính toán phức tạp, triệu gọi những đối tượng phân tán khác, truy xuất cơ sở dữ liệu ... Một điều lưu ý là các đối tượng (gọi tắt là đối tượng EJB) hoạt động dưới sự điều khiển (hay bên trong) một trình chứa (container). Nhiệm vụ trình chứa như đã nêu là bảo vệ và thực hiện các tác vụ kiểm soát đối tượng EJB (khởi tạo, cấp tài nguyên, hủy đối tượng ...)

Bạn có thể tự hỏi những công việc này máy ảo Java đã thực hiện tại sao trình chứa container phải lặp lại? Trình chứa container quản lý đối tượng ở mức cao hơn. Với máy ảo Java, tạo ra đối tượng chỉ đơn giản là cấp phát bộ nhớ cho đối tượng thì với trình chứa container, tạo ra đối tượng ngoài việc gọi máy ảo Java cấp phát bộ nhớ trình chứa còn thông báo sự hiện diện của đối tượng EJB đến các đối tượng khác. Đăng ký cho phép EJB có khả năng triệu gọi từ xa với dịch vụ JNDI (bạn xem lại chương RMI về ý nghĩa của dịch vụ quản lý tên đối tượng JNDI), hỗ trợ kết nối JDBC ...

Trình chứa container hoạt động bên trong môi trường EJB. Môi trường EJB là nơi cung cấp các dịch vụ của Java như: dịch vụ JNDI, dịch vụ quản lý giao dịch JTS (Java Transaction Service), dịch vụ chuyển giao thông điệp (Java Message Service), dịch vụ kết nối cơ sở dữ liệu JDBC ...

Trình chứa container sẽ thay mặt đối tượng triệu gọi các chức năng giao dịch của môi trường EJB. Trong một môi trường EJB có thể có nhiều trình chứa cùng hoạt động.

Tóm lại bạn cần nhớ một số khái niệm cơ bản sau trước khi đi vào tìm hiểu và xây dựng ứng dụng phân tầng Enterprise Java Bean.

- Enterprise Java Bean (EJB): Kiến trúc lập trình phân tầng trong nền Java 2.
- Thành phần web (web component): các đơn thể hay chương trình con được gọi bởi trình chủ web (trang jsp, servlet, trang html)
- Trình chứa (container): môi trường điều khiển các đối tượng Enterprise Java Bean. Trình chứa sẽ kiểm soát quá trình kiến tạo, hoạt động và hủy đối tượng. Bạn triệu gọi đối tượng Enterprise Java Bean thông qua trình chứa.
- Đối tượng EJB: các đối tượng có thể hoạt động bên trong trình chứa container và tuân theo một cài đặt nhất định (hay đặc tả EJB) sẽ được chúng ta nghiên cứu sau.
- Môi trường EJB: nơi cung cấp các dịch vụ Java hỗ trợ trình chứa.

Bạn lưu ý: đối tượng Enterprise Java Bean (EJB) hoàn toàn khác với đặc tả Java Bean. Java Bean là những đối tượng hay mã Java cho phép phân biệt thuộc tính cũng như phương thức của đối tượng khi triệu gọi. Java Bean thường được xem như những thành phần giao diện gắn hay kết nối vào ứng dụng kiểu mô hình lắp ghép (bạn xem *Giáo trình và lý thuyết bài tập Java* để biết thêm chi tiết các xây dựng Java Bean). Ngược lại EJB là những đối tượng tuân theo đặc tả của môi trường hoạt động tích hợp ở cấp xí nghiệp. Các đối tượng EJB không dùng thể hiện giao diện. Các đối tượng EJB thực hiện những chức năng tính toán, kết nối lưu trữ dữ liệu, đưa ra kết quả và giải pháp thiết thực cho doanh nghiệp. (có lẽ nếu Sun

thay bằng thuật ngữ khác thì đỡ nhầm lẫn hơn. Java Bean đã ra đời và tồn tại rất lâu trước khi có Enterprise Java Bean)

Khi phát triển ứng dụng mạng diện rộng, kiến trúc J2EE là mô hình phát triển lý tưởng và dễ bảo trì nhất mà bạn có thể chọn. Mô hình này được Java đưa ra từ phiên bản Java 1.2. Nó không nằm chung với công cụ JDK. Bạn phải cài đặt J2EE như một nhánh riêng. Tuy nhiên J2EE và JDK chính là nội dung chủ yếu của nền Java Platform 2.

Trong giáo trình này chúng tôi sử dụng JDK 1.3 cộng với nền J2EE mới nhất hiện nay 1.2.1. Bạn có thể chép về trình cài đặt j2ee mới nhất tại địa chỉ http://java.sun.com. Nếu không bạn có thể liên hệ nhà sách Minh Khai để chép đĩa CD với đầy đủ công cụ Java trong đó có trình cài đặt j2ee.

Mặc định trong các ví dụ minh họa sau này được xem như bạn cài đặt J2EE vào thư mục C:\j2sdkee1.2.1.

Sun không yêu cầu bạn phải sử dụng J2EE để phát triển ứng dụng đa tầng. Sun chỉ đưa ra đặc tả của mô hình EJB. J2EE là một phần cài đặt trong đặc tả của Sun đưa ra. Hiện bạn có thể tìm thấy trên thị trường rất nhiều trình chủ xây dựng và hỗ trợ kiến trúc Enterprise Java Bean như WebLogic, WebSphere (của hãng IBM), Oracle Application Server (của Oracle). Tuy nhiên, J2EE là phần mềm miễn phí và tuân thủ nghiêm ngặt các đặc tả của Sun nhất (bù lại bạn sẽ không có được những dịch vụ hỗ trợ cao cấp như trình điều khiển JDBC đa dụng, chia sẻ kết nối, quản lý tài nguyên hiệu quả ...).

Thuật ngữ Bean là một tên gọi tượng hình cho đối tượng (bean mang ý nghĩa là hạt café Java). Ở một mặt nào đó bạn có thể xem Bean đồng nghĩa với từ object (đối tượng).

2. TÌM HIỂU ĐỐI TƯỢNG EJB

Trước khi đi vào chi tiết xây dựng đối tượng EJB, chúng ta cần tóm tắt các loại đối tượng EJB có thể hoạt động bên trong trình chứa container. Sun định nghĩa các dạng đối tượng Bean như sau

2.1. Bean thao tác (Session Bean)

Nếu đối tượng Bean của bạn chỉ thực hiện các hành vi xử lý, tính toán đơn thuần thì chúng được phân loại thành Bean thao tác (session bean). Bean thao tác chỉ có nhiệm vụ phục vụ trình khách (nơi triệu gọi đối tượng) trong một phiên kết nối (session). Những thao tác nhất thời không đòi hỏi đến việc thể hiện dữ liệu thường bao gồm: tính toán, phân tích, thống kê, đưa ra giải pháp từ một tập dữ liệu mẫu tạm nào đó. Thao tác phục vụ trình khách của đối tượng Bean thường được thể hiện thông qua phương thức cho phép triệu gọi. Đối với Bean thao tác, đối tượng lại được chia ra làm hai loại khác đó là

2.1.1. Bean thao tác phi trạng thái (Stateless bean)

Bean thao tác phi trạng thái là các thành phần Bean không lưu lại trạng thái của giao dịch trước đó để sử dụng cho lần giao dịch sau. Ví dụ như khi trình khách triệu gọi đối tượng Bean yêu cầu cho biết thông tin về thời tiết. Đối tượng Bean trả về thời tiết hiện hành của một vùng và chấm dứt quan hệ với trình khách. Bean thao tác phi trạng thái là đối tượng Bean đơn giản, dễ dùng, dễ thiết kế và quản lý nhất trong tất cả các loại đối tượng EJB.

2.1.1. Bean thao tác lưu vết trạng thái (Stateful bean)

Bean thao tác lưu vết trạng thái là các thành phần Bean cần lưu lại kết quả hay vị trí giao dịch trước đó để phục vụ cho các lần giao dịch tiếp theo. Các Bean này thường phục vụ cho những thao tác đòi hỏi phải qua nhiều bước triệu gọi trước khi trả về kết quả cuối cùng. Ví dụ nếu đối tượng Bean của bạn phục vụ cho việc trắc nghiệm bài thi. Bean phải lưu lại kết quả tạm thời của những câu trả lời trước

khi bạn có thể triệu gọi phương thức của Bean cho biết đánh giá sau cùng về bài thi.

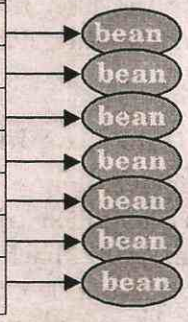
Các ví dụ về Bean thao tác sẽ được xây dựng cụ thể hơn ở phần sau.

2.2. Bean thực thể (Entity Bean)

Đối tượng thường mô phỏng một thực thể (entity) nào đó trong cuộc sống thực. Trong các hệ quản trị cơ sở dữ liệu ngày nay bạn có thể lưu trữ thông tin ở dạng bảng (table). Ví dụ bảng personel mang các thông tin về quản lý nhân sự được mô tả như sau

Personal

ID	Name	Age
1	Lan	26
2	Dung	22
3	Phuong	26
4	Thuy	24
5	Duong	25
6	My	27
7	Thang	28



Như bạn thấy thực tế mỗi hàng dữ liệu trong bảng personel là một thông tin về đối tượng. EJB cho phép ánh xạ dữ liệu trong mô hình quan hệ thành các đối tượng Java. Ví dụ, bạn có thể biểu diễu bảng personel trên đây thành 7 đối tượng Java với tên gọi (name) và số tuổi (age) khác nhau. Thay vì phải truy xuất vào cơ sở dữ liệu để thay đổi giá trị của từng dòng bạn có thể trực tiếp thay đổi ngay trong giá trị thể hiện của Bean. Khi bạn đã ánh xạ từng dòng dữ liệu thành đối tượng Bean chúng được gọi là các Bean thực thể (Entity Bean).

Các Bean thao tác (session bean) sẽ tương tác với Bean thực thể, nhằm thay đổi và truy vấn dữ liệu của Bean thực thể. Khi dữ liệu của Bean thực thể thay đổi gián tiếp dòng dữ liệu mà Bean thực thể ánh xạ trong bảng dữ liệu cũng sẽ thay đổi theo. Bằng cách này, mô hình EJB cho phép đối tượng hóa hoàn toàn từ thao tác xử lý cho đến dữ liệu thể hiện. Bạn không còn phải phụ thuộc vào mô hình phẳng của các bảng dữ liệu nữa.

Bởi vì đối tượng Bean thực thể là một ánh xạ từ dữ liệu của các hệ cơ sở dữ liệu nên bạn phải có cách nào đó đưa được dữ liệu từ các hệ cơ sở dữ liệu này vào bên trong đối tượng Bean. Việc quản lý dữ liệu của Bean thực thể vì thế được phân ra làm hai loại

2.2.1. Bean thực thể tự quản lý (Bean Managed)

Bean thực thể tự quản lý là các thành phần Bean có khả năng tự truy vấn các hệ cơ sở dữ liệu để lấy về dữ liệu mà nó thể hiện. Ví dụ trong phương thức khởi dựng Bean thực thể có thể dùng các câu lệnh SQL như SELECT, INSERT (thông qua JDBC) để tự tìm hoặc thêm mới dữ liệu mà nó thể hiện vào các bảng trong cơ sở dữ liệu.

2.2.2. Bean thực thể quản lý bởi trình chứa (Container Managed)

Thành phần Bean có thể không cần phải sử dụng lệnh SQL để tự tìm kiếm hay tạo mới dữ liệu mà nó thể hiện. Thay vào đó, đối tượng Bean chỉ cần khai báo các tên trường (field) hay cột dữ liệu tương ứng với các bảng (table) trong hệ cơ sở dữ liệu. Trình chứa container sẽ tự động thực hiện công việc truy vấn dữ liệu giúp thành phần Bean. Đối tượng Bean thể hiện dữ liệu theo kiểu này được gọi là *Bean thực thể quản lý bởi trình chứa*.

Trên đây là tóm tắt các đối tượng Bean trong mô hình phân tầng J2EE và môi trường EJB. Các đối tượng Bean này sẽ được chúng ta tìm hiểu chi tiết hơn dưới đây.

3. XÂY DỰNG ĐỐI TƯỢNG BEAN EJB

3.1. Thiết kế thành phần EJB

Chúng ta sẽ bắt đầu bằng việc thiết kế thành phần EJB đơn giản nhất. Thành phần Bean phi trạng thái (stateless Bean) mang tên Converter. Bean Converter thực hiện chức năng chuyển đổi tỉ giá giữa đồng Yên Nhật với đồng USD, Euro

Đối tượng Bean Converter thuộc loại Bean thao tác (session Bean) bản thân Bean không thể hiện dữ liệu, nó chỉ cung cấp phương thức xử lý các yêu cầu tính toán, chuyển đổi ngoại tệ. Tất cả các Bean thao tác đều yêu cầu bạn cài đặt giao tiếp javax.ejb.SessionBean.

Thành phần Bean Converter có mã nguồn được cài đặt như sau

Ví dụ 9-1 ConverterEJB.java

```
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
public class ConverterEJB implements SessionBean {
  //Phương thức khởi dựng đối tượng Bean
  public ConverterEJB() {}
  //Chuyển đổi USD sang Yên
  public double dollarToYen(double dollars) {
    return dollars * 121.6000;
  }
  //Chuyển đổi Yên sang đồng Euro
  public double yenToEuro(double yen) {
    return yen * 0.0077;
```

```
  }
  /* Các phương thức cần cài đặt cho interface SessionBean */
  public void ejbCreate() {
    System.out.println(" ejbCreate call !");
  }
  public void ejbRemove() {
    System.out.println(" ejbRemove call !");
  }
  public void ejbActivate() {
    System.out.println(" ejbActivate call !");
  }
  public void ejbPassivate() {
    System.out.println(" ejbPassivate call !");
  }
  public void setSessionContext(SessionContext sc) {
    System.out.println(" setSessionContext call !");
  }
}
```

Như bạn thấy đối tượng ConverterEJB của ta cung cấp hai phương thức chính là dollarToYen() và yenToEuro() để chuyển đổi ngoại tệ. Đây chính là hai phương thức thể hiện rõ mục đích phục vụ của thành phần Bean. Chúng còn được gọi là các phương thức thể hiện logic nghiệp vụ (business logic) mà Bean cần thực hiện.

Như đã nêu, tất cả các thành phần đối tượng EJB đều phải hoạt động bên trong và chịu sự kiểm soát của trình chứa container. Giao tiếp SessionBean mà đối tượng Bean cài đặt chính là cách để trình chứa container có thể trao đổi và điều khiển được thành phần Bean. Giao tiếp này đặc tả và qui định các phương thức sau:

Phương thức	Mục đích
`ejbCreate()`	Trình chứa sẽ triệu gọi phương thức này để tạo ra đối tượng Bean
`ejbRemove()`	Trình chứa triệu gọi phương thức này để hủy đối tượng Bean
`ejbPassivate()`	Trình chứa triệu gọi phương thức này để thông báo cho Bean biết bản thân đối tượng Bean sắp bị đưa ra vùng nhớ tạm (lưu Bean xuống đĩa cứng - tạm thời không sử dụng).
`EjbActivate()`	Trình chứa triệu gọi phương thức này thông báo cho thành phần Bean biết bản thân đối tượng được đưa trở về trạng thái kích hoạt (khôi phục Bean từ vùng nhớ tạm - đưa Bean trở về trạng thái hoạt động)
`setSessionContext()`	Trình chứa triệu gọi phương thức này để tạo điều kiện cho thành phần Bean nhận được các thông tin từ môi trường trình chứa container (thông qua đối tượng SessionContext).

Nếu không muốn xử lý những tình huống này bạn có thể cài đặt rỗng. ConverterEJB đơn giản chỉ in ra màn hình câu thông báo cho biết các phương thức đã được trình chứa gọi đến.

Trình khách không gọi đối tượng ConverterEJB trực tiếp (thực tế thì trình chứa container không cho phép bạn gọi đến phương thức của ConverterEJB). Khi bạn cài đặt ConverterEJB vào trình chứa, trình chứa yêu cầu bạn phải tạo ra giao tiếp (interface) để trình chứa có thể đứng trung gian sử dụng giao tiếp này liên lạc giữa trình khách và đối tượng Bean bên trong trình chứa. Lớp giao tiếp này được xây dựng như sau

Ví dụ 9-2 Converter.java

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Converter extends EJBObject {
  public double dollarToYen(double dollars)
                                    throws RemoteException;
  public double yenToEuro(double yen)
                                    throws RemoteException;
}
```

Theo đặc tả và qui định của kiến trúc Enterprise Java Bean lớp giao tiếp trung gian mà trình chứa sử dụng triệu gọi phương thức của thành phần Bean phải kế thừa từ lớp javax.ejb.EJBObject. Bạn có thể thắc mắc Converter chỉ là đặc tả interface và không liên quan gì đến lớp ConverterEJB thì làm sao có thể triệu gọi các phương thức dollarToYen() và yenToEuro(). Bạn không lo điều này, khi tìm hiểu quá trình cài đặt ở phần sau bạn sẽ thấy trình cài đặt (deploytool) sẽ tự động sinh ra các lớp kết nối cần thiết.

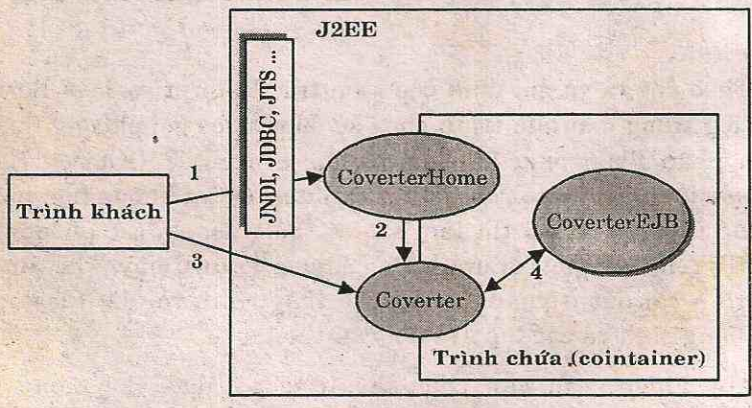
Tuy nhiên, bạn vẫn chưa thể triệu gọi được đối tượng Bean trong trình chứa. Muốn tham chiếu được đến thành phần Bean từ máy khách, EJB đòi hỏi bạn phải tạo lớp "mồi". Lớp này được gọi là

lớp chủ (Home). Từ trình khách, bạn gọi lớp chủ để tạo ra và lấy về tham chiếu của giao tiếp Converter. Bạn không cần cài đặt lớp chủ, bạn chỉ cần đặc tả lớp chủ home là một lớp giao tiếp interface kế thừa lớp EJBHome. Trong lớp chủ này bạn khai báo duy nhất phương thức create() có kiểu trả về là lớp giao tiếp Converter như sau

Ví dụ 9-3 ConverterHome.java

```
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;
public interface ConverterHome extends EJBHome
{
    Converter create()
            throws RemoteException, CreateException;
}
```

Bạn không phải cài đặt cho giao tiếp ConverterHome. Trình đóng gói deploytool sẽ thực hiện công việc này.



Hình 9-4 Giao tiếp giữa trình khách và đối tượng EJB trong trình chứa

Có thể có quá nhiều qui định bắt buộc phải làm và tạo ra khá nhiều các lớp trung gian trước khi sử dụng được đối tượng Bean EJB. Tuy nhiên nếu bạn đã nghiên cứu về RMI thì cơ chế EJB không khác gì mô hình "*xưởng chế tác*" (factory) mà ta đã xây dựng trong chương 2 (mục 10). Hình 9-4 là mô hình triệu gọi thật sự của trình khách đến thành phần Bean bên trong trình chứa trên máy chủ.

1. Trình khách sử dụng dịch vụ JNDI yêu cầu lấy về tham chiếu đến lớp chủ ConverterHome.
2. Trình khách gọi phương thức create() của lớp giao tiếp chủ ConvertorHome yêu cầu tạo ra tham chiếu đến lớp Converter.
3. Sau khi có tham chiếu đến lớp Converter, trình khách sẽ sử dụng lớp Converter triệu gọi phương thức của đối tượng Bean trong trình chứa container.
4. Trình chứa đón nhận lời triệu gọi từ Converter, thay mặt trình khách gọi đến thành phần Bean thực tế bên trong do trình chứa quản lý.

Bạn nên nhớ, ConverterHome và Converter là các interface. Ta chỉ cài đặt ConverterEJB. Khi bạn sử dụng trình đóng gói deploytool, trình đóng gói sẽ tự tạo ra các đối tượng tương thích cài đặt cho giao tiếp ConverterHome và Converter.

3.2. Biên dịch và cài đặt đối tượng EJB

Khi biên dịch các đối tượng EJB bạn nên trỏ đường dẫn classpath đến thư viện j2ee.jar. Để không làm ảnh hưởng đến biến môi trường CLASSPATH bạn nên tạo tập tin biên dịch .bat với các thiết lập như sau

ConverterCompile.bat

```
REM    Chỉ định đường dẫn đến thư mục cài đặt J2EE
set J2EE_HOME=C:\j2sdkee1.2.1
```

```
REM   Chỉ định đường dẫn đến thư mục cài đặt nền JDK
set JAVA_HOME=C:\jdk1.3
REM   Chỉ định thư viện j2ee cần dùng cho quá trình biên dịch
set CPATH=.;%J2EE_HOME%\lib\j2ee.jar
REM   Biên dịch chương trình
javac -classpath %CPATH% *.java
```

Lưu các mã nguồn của ví dụ 9-1, 9-2, 9-3 vào thư mục C:\Network\ejb\converter. Chuyển vào thư mục này bạn đánh lệnh biên dịch

```
C:\Network\ejb\converter >ConverterCompile.bat
```

Kết quả thu được tập tin ConverterEJB.class, Convert.class và ConvertHome.class.

Tiếp theo là quá trình đóng gói các lớp này đưa vào trình chứa.

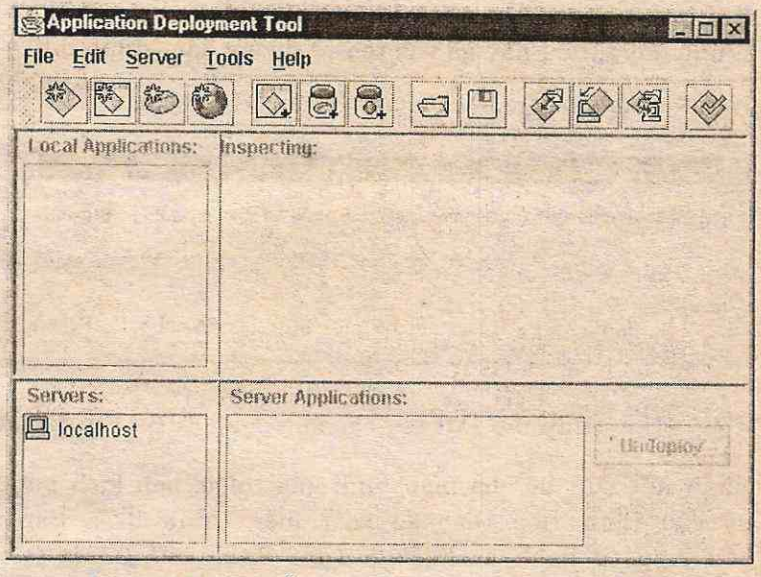
3.2.1. Khởi tạo môi trường J2EE

Bạn chuyển vào thư mục C:\j2sdkee1.2.1\bin đánh lệnh khởi động như sau

```
C:\j2sdkee1.2.1\bin>set j2EE_HOME=C:\j2sdkee1.2.1
C:\j2sdkee1.2.1\bin>set java_home=C:\jdk1.3
C:\j2sdkee1.2.1\bin>j2ee -verbose
J2EE server Listen Port: = 1049
Naming service started: :1050
Published the configuration object ...
Binding DataSource, name = jdbc/Cloudscape,
url =jdbc:cloudscape:rmi:CloudscapeDB;create=true
Web service started: 9191
Web service started: 8000
Web service started: 7000
```

```
J2EE server startup complete.
```

Môi trường J2EE cung cấp cho bạn 3 dịch vụ Web, dịch vụ web HTTP thông thường hoạt động trên cổng 8000, dịch vụ Web HTTPS hoạt động trên cổng 7000, dịch vị Web dùng cho việc đóng gói và cài đặt thành phần EJB hoạt động trên cổng 9191.



Hình 9-5 Cửa sổ trình đóng gói đối tượng EJB

Bạn hãy để trình chủ j2ee chạy ở hậu trường trên một cửa sổ DOS-Prompt độc lập. j2ee cung cấp cho bạn trình chứa container sẵn sàng tiếp nhận bất kỳ thành phần EJB nào cài đặt theo đúng đặc tả của Enterprise Java Bean. Phần cài đặt ConverterEJB vào bên trong trình chứa được thực hiện bởi trình đóng gói deploytool mà bạn phải thực hiện sau đây.

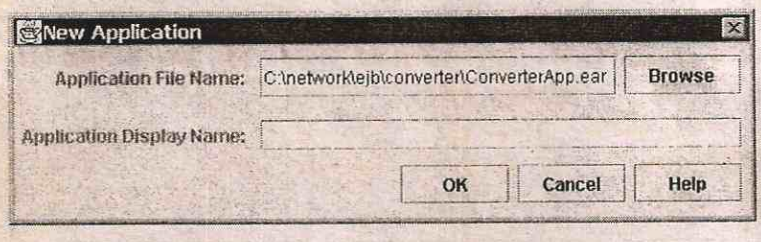
3.2.2. Đóng gói thành phần Bean với deploytool

Bạn mở cửa sổ DOS-Prompt khác, chuyển vào thư mục C:\j2sdkee1.2.1\bin đánh lệnh sau

```
C:\j2sdkee1.2.1\bin>set j2EE_HOME=C:\j2sdkee1.2.1
C:\j2sdkee1.2.1\bin>set java_home=C:\jdk1.3
C:\j2sdkee1.2.1\bin>deploytool
```

Cửa sổ trình đóng gói cho đối tượng Bean sẽ hiển thị như hình 9-5. Trước hết, bạn chọn *New Application* để tạo ra tập tin ứng dụng .ear. Khi bạn chọn mục này deploytool sẽ yêu cầu bạn nhập vào tên tập tin ứng dụng, bạn hãy gõ vào tên và đường dẫn (hình 9-6)

C:\network\ejb\converter\ConverterApp.ear

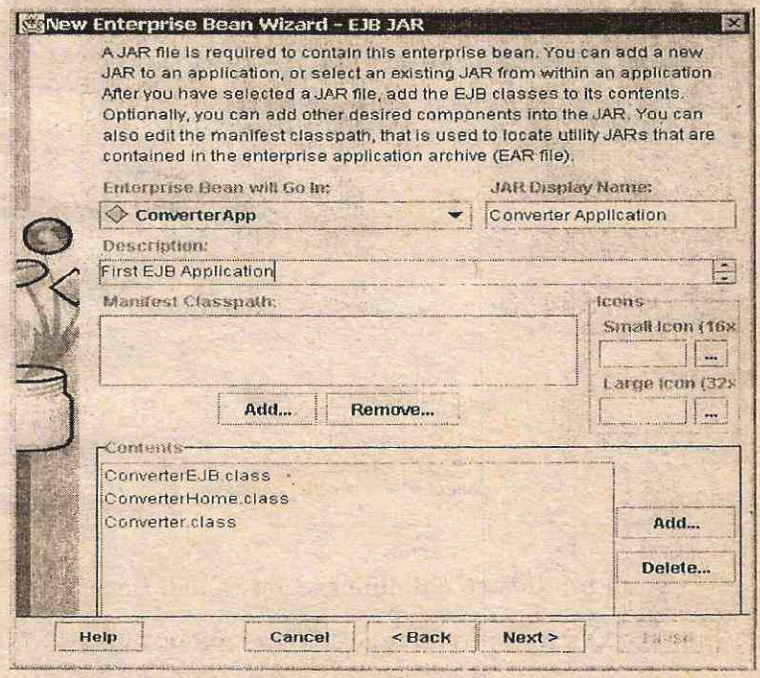


Hình 9-6 Đặt tên cho ứng dụng EJB

Nhấn nút *OK*, bỏ qua màn hình giới thiệu, bạn kích nút *Next*. Cửa sổ cấu hình tiếp theo sẽ xuất hiện (hình 9-7). Bạn chọn ConverterApp từ danh sách *Enterprise Bean will Go in*. Gõ vào Converter App ở ô *JAR Display Name*. Ở mục *Contents* bạn nhấn vào nút *Add* để đưa các lớp Bean vào. Bạn chọn thư mục chứa các lớp Converter.class, ConverterHome.class, ConverterEJB.class và nhấn nút OK để thêm vào. Nhấn nút *Next* để qua bước kết tiếp.

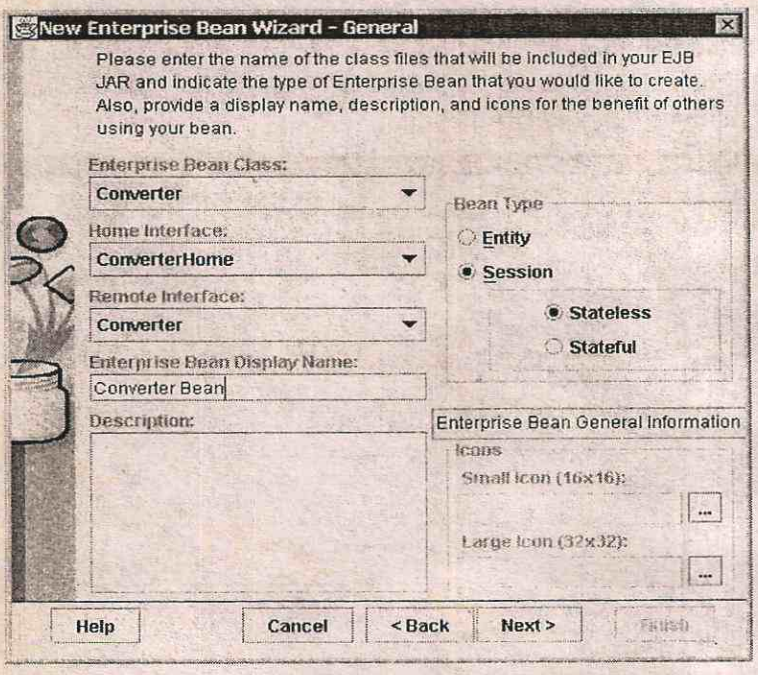
Cửa sổ chỉ định thuộc tính của ứng dụng sẽ xuất hiện như hình 9-8. Bạn chọn ConverterEJB.class từ ô danh sách *Enterprise Bean Class*, chọn ConverterHome.class từ ô danh sách *Home Interface*. chọn Converter từ mục *Remote Interface*. Do thành phần Bean của ta có tính chất là Bean thao tác (session Bean) cho nên bạn hãy chọn mục Session ở mục *Bean Type*, tiếp đến chọn Stateless để trình chứa container hiểu rằng Bean của bạn thuộc loại Bean thao tác phi

trạng thái. Tiếp đến nếu muốn mô tả cho mục đích sử dụng của Bean bạn có thể gõ vào ô *Enterprise display name* chuỗi Converter Bean. Nhấn nút *Finish* để kết thúc quá trình đóng gói (bạn có thể nhấn *Next* để đi đến cửa sổ cấu hình các thông tin bổ sung cho môi trường hoạt động của Bean, ở đây Converter Bean không cần cấu hình thêm cho các thông số môi trường). Kết quả khi trở về màn hình chính của cửa sổ deploytool được trình bày như hình 9-9. Cửa sổ deploytool được chia ra làm 4 phần. Phần thứ nhất Local Application là các đối tượng của ứng dụng EJB được lưu theo hình cây. Phần thứ hai *Inspecting* là nơi hiển thị các thông tin chi tiết về đối tượng. Phần thứ 3, *Servers* là địa chỉ máy chủ nơi trình chứa hoạt động (bạn có thể thêm vào địa chỉ máy chủ bằng cách chọn *Server|Add* từ trình đơn). Phần thứ 4, *Server Application* là cửa sổ hiển thị các ứng dụng EJB đã được đưa vào trình chứa.



Hình 9-7 Cửa sổ chọn các lớp cần thiết cho Bean

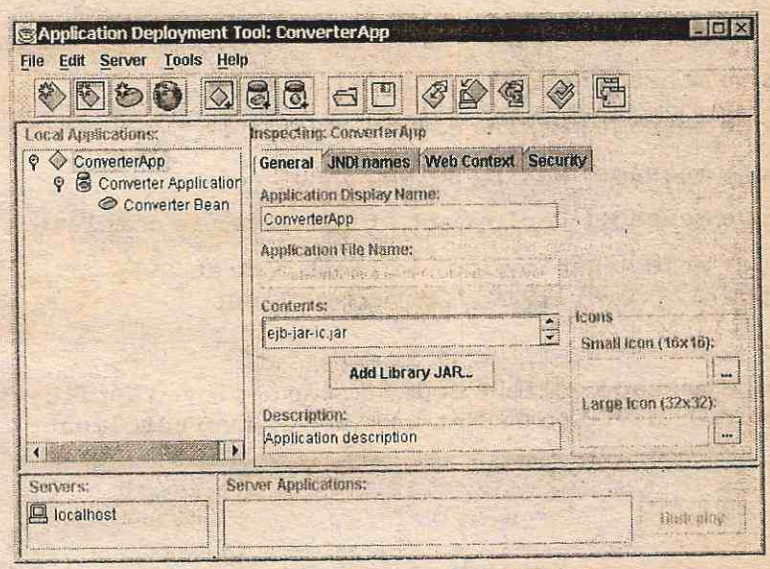
Đưa ứng dụng EJB vào trình chứa chính là mục đích của deploytool. Bạn chọn mục *Application | Deploy* từ trình đơn. Cửa sổ thông báo ứng dụng sẽ được đóng gói đưa vào trình chứa container như hình 9-10. Trong cửa sổ này bạn chọn mục *Target server* để chỉ định máy chủ nơi J2EE server hoạt động (J2EE đã được ta khởi động trong cửa sổ DOS-Prompt ở phần trên). Chọn mục *Return Client Jar*, trình đóng gói sẽ chọn ra những lớp interface cần thiết gộp vào tập tin .Jar để trình khách có thể sử dụng và triệu gọi thành phần Bean bên trong trình chứa container. Bạn gõ vào ô *Client JAR File name* đường dẫn



Hình 9-8 Cửa sổ chỉ định các lớp và loại Bean

```
C:\network\ejb\converter\ConverterAppClient.jar
```

Nhấn nút *Next* để chuyển qua màn hình kế tiếp. Trình đóng gói sẽ yêu cầu bạn nhập vào tên JNDI cho thành phần Bean (hình 9-11). Bạn gõ vào cột *JNDI Name* tên MyConverter. Trình khách sẽ dựa vào tên này để yêu cầu J2EE trả về tham chiếu đến lớp chủ ConverterHome. Nhấn Next quá trình đóng gói sẽ bắt đầu, trình đóng gửi lệnh đến máy chủ J2EE (bạn mở cửa sổ DOS-Prompt nơi J2EE đang chạy để thấy rõ quá trình đóng gói và cài đặt diễn ra) yêu cầu tiếp nhận ứng dụng EJB (hình 9-12).



Hình 9-9 Cửa sổ cài đặt ứng dụng EJB Converter

Trên cửa sổ DOS-Prompt nơi deploytool đang chạy bạn sẽ thấy những thông báo kết xuất sau

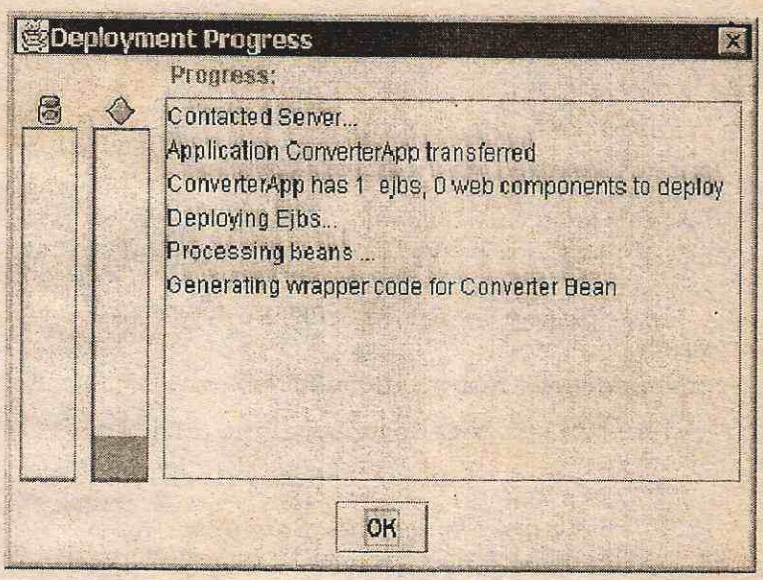
```
Deploy the application in
C:\j2sdkee1.2.1\bin\ConvertApp.ear on the server local

host saving the client jar as
C:\Network\EJB\ConverterClient

Sender object Deploy Tool : Deploy Converter on localhost

Remote message: Contacted Server....
```

```
Remote message: Application Converter transferred.

Remote message: Converter has 1 ejbs, 0 web components to
deploy.

Remote message: Deploying Ejbs....

Remote message: Processing beans ....

Remote message: Generating wrapper code for Converter
Yeen Dollar.

Remote message: Compiling wrapper code ....

Remote message: Compiling RMI-IIOP code ....

Remote message: Making client JARs ....

Remote message: Making server JARs ....

Remote message: Deployment of Converter is complete..

Sender object Deploy Tool : client code at
http://127.0.0.1:9191/ConverterClient

.jar

Remote message: Client code for the deployed application
Converter saved to C:\Network\EJB\ConverterClient.
```

Hình 9-10 Cửa sổ đóng gói chuyển Bean cho trình chứa

Hình 9-11 Cửa sổ đặt tên JNDI cho Bean

Nếu quá trình đóng gói và chuyển giao ứng dụng EJB cho trình chủ J2EE thành công thì xin chúc mừng bạn. Bạn đã hoàn tất quá trình cài đặt ứng dụng Enterprise Java Bean.



Hình 9-12 Quá trình biến dịch và chuyển Bean cho trình chứa

3.3. Viết trình khách triệu gọi thành phần Bean

Trình khách ConverterClient.java sau đây sẽ thực hiện triệu gọi thành phần Bean

Ví dụ 9-4 ConverterClient.java

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import Converter;
import ConverterHome;
public class ConverterClient {
  public static void main(String[] args) {
    try {
      //Khởi tạo môi trường EJB
      Context initial = new InitialContext();
      //Lấy về tham chiếu của lớp chủ EJB
      Object objref = initial.lookup("MyConverter");
      //Chuyển kiểu lớp đối tượng tổng quát về kiểu
ConverterHome
      ConverterHome home = (ConverterHome)
          PortableRemoteObject.narrow(objref,
                    ConverterHome.class);
      //Tạo ra đối tượng tham chiếu Bean Converter
      Converter currencyConverter = home.create();
      //Triệu gọi phương thức của Bean và in ra kết quả
      double amount =
          currencyConverter.dollarToYen(100.00);
```

```
        System.out.println("100 Dollar to Yen ="+
                                String.valueOf(amount));
        //Triệu gọi phương thức của Bean và in ra kết quả
        amount = currencyConverter.yenToEuro(100.00);
        System.out.println("100 Yen to Euro ="+
                                String.valueOf(amount));
    } catch (Exception ex) {
        System.err.println("Caught an
                        unexpected exception!");
        ex.printStackTrace();
    }
  }
}
```

Để biên dịch ConverterClient.java bạn cần phải có tập tin ConverterAppClient.jar đã được trình đóng gói deploytool tạo ra ở mục 3.2.2.

Bạn lưu ConverterClient.jar vào cùng thư mục

C:\network\ejb\converter

Thực hiện lệnh biên dịch như sau

- Chỉ định đường dẫn đến thư viện biên dịch

```
C:\network\ejb\converter> set
            CPATH=.;C:\j2sdkee1.2.1\lib\j2ee.jar
                    ;ConverterAppClient.jar
```

- Biên dịch chương trình

```
C:\network\ejb\converter>javac
          -classpath %CPATH% ConverterClient.java
```

Tiếp đến, bạn chạy trình khách từ dòng lệnh như sau

```
C:\network\ejb\converter>java
            -classpath %CPATH% ConverterClient
```

Kết quả kết xuất

```
100 Dollar to Yen = 12160.00
100 Yen to Euro = 0.77
```

3.4. Viết trang JSP triệu gọi thành phần EJB

Bạn có thể viết trang jsp đặt vào trình chủ Web để triệu gọi thành phần EJB như sau

Ví dụ 9-5 TestEJB.jsp

```
<HTML>
<BODY>
 <%@ page import="java.rmi.*,javax.naming.*" %>
 <%
   try {
      //Khởi tạo môi trường EJB
     Context initial = new InitialContext();
      //Lấy về tham chiếu của lớp chủ EJB
     Object objref = initial.lookup("MyConverter");
      //Chuyển kiểu lớp đối tượng tổng quát về kiểu
ConverterHome
     ConverterHome home = (ConverterHome)
          PortableRemoteObject.narrow(objref,
                    ConverterHome.class);
      //Tạo ra đối tượng tham chiếu Bean Converter
     Converter currencyConverter = home.create();
```

```
        //Triệu gọi phương thức của Bean và in ra kết quả
        double amount =
            currencyConverter.dollarToYen(100.00);
        out.println("100 Dollar to Yen ="+
                        String.valueOf(amount));
        //Triệu gọi phương thức của Bean  và in ra kết quả
        amount = currencyConverter.yenToEuro(100.00);
        out.println("100 Yen to Euro ="+
                        String.valueOf(amount));
    } catch (Exception ex) {
        out.println("Caught an
                    unexpected exception!");
        ex.printStackTrace();
    }
%>
</BODY>
</HTML>
```

Bạn có thể đặt trang TestEJB.jsp vào thư mục web gốc C:\JavaWebServer2.0\public_html và kiểm tra bằng địa chỉ URL sau

http://localhost:8080/TestEJB.jsp

4. TÌM HIỂU CÁCH XÂY DỰNG EJB CỦA TRÌNH ĐÓNG GÓI

Sau khi biên dịch và chuyển giao thành công đối tượng EJB vào trình chứa container bằng deploytool bạn có thể chuyển vào thư mục

C:\j2sdkee1.2.1\repository\<computer_name>\gnrtrTMP\Converter

trong thư mục này bạn sẽ thấy rất nhiều file phụ do trình đóng gói sinh ra để phục vụ cho Bean Converter. Trong số các tập tin phụ này bạn lưu ý đến ConverterHomeImpl.java và ConverterEJB_EJBObjectImpl.java. Đây là hai tập tin được tự động sinh ra để cài đặt cho lớp giao tiếp ConverterHome và Converter. Hai tập tin này có mã nguồn như sau (bạn lưu ý mã nguồn này là do deploytool sinh ra)

Ví dụ 9-6 ConverterHomeImpl.java

```
public final class ConverterHomeImpl
      extends com.sun.ejb.containers.EJBHomeImpl
      implements ConverterHome {
  public ConverterHomeImpl(com.sun.ejb.Container c)
                        throws java.rmi.RemoteException {
   super(c);
  }
  public Converter create() throws
        java.rmi.RemoteException,
        javax.ejb.CreateException {
     ConverterEJB_EJBObjectImpl ejbObject =
          (ConverterEJB_EJBObjectImpl)
                        this.createEJBObject();
     return (Converter)ejbObject.getStub();
  }
}
```

Ví dụ 9-7 ConverterEJB_EJBObjectImpl.java

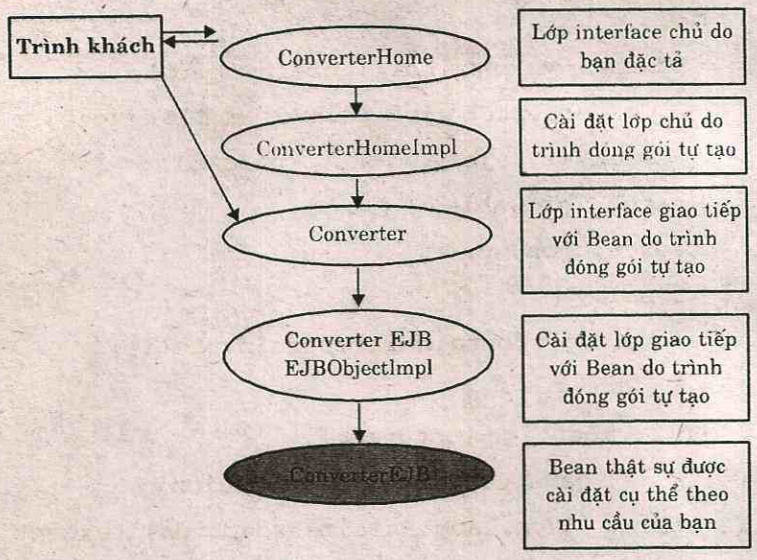
```
public final class ConverterEJB_EJBObjectImpl
  extends com.sun.ejb.containers.
```

```
    StatelessSessionEJBObjectImpl
  implements Converter {
  public ConverterEJB_EJBObjectImpl() throws
                        java.rmi.RemoteException {
  }
  public double dollarToYen(double param0)
                throws java.rmi.RemoteException{
    com.sun.ejb.Invocation i = new
                com.sun.ejb.Invocation();
    i.ejbObject = this;
    try {
      java.lang.Class[] cl = new java.lang.Class[1];
      cl[0] = double.class;
      i.method =
      Converter.class.getMethod("dollarToYen", cl);
    } catch(NoSuchMethodException e) {
    }
    double $retVal = 0;
    try {
      this.getContainer().preInvoke(i);
      ConverterEJB ejb = (ConverterEJB) i.ejb;
      $retVal = ejb.dollarToYen(param0);
    } catch(Throwable c) {
      i.exception = c;
    } finally {
      this.getContainer().postInvoke(i);
```

```
    }
      if(i.exception instanceof
              java.lang.RuntimeException) {
     throw (java.lang.RuntimeException)i.exception;
    } else if(i.exception instanceof
                  java.rmi.RemoteException) {
     throw (java.rmi.RemoteException)i.exception;
     } else if(i.exception != null) {
   throw new java.rmi.RemoteException("Unknown
                          exception", i.exception);
     }
     return $retVal;
}
public double yenToEuro(double param0) throws
                    java.rmi.RemoteException{
   com.sun.ejb.Invocation i = new
                  com.sun.ejb.Invocation();
     i.ejbObject = this;
     try {
      java.lang.Class[] cl = new java.lang.Class[1];
       cl[0] = double.class;
        i.method =
          Converter.class.getMethod("yenToEuro", cl);
   } catch(NoSuchMethodException e) {
    }
    double $retVal = 0;
```

```
        try {
            this.getContainer().preInvoke(i);
            ConverterEJB ejb = (ConverterEJB) i.ejb;
            $retVal = ejb.yenToEuro(param0);
        } catch(Throwable c) {
        i.exception = c;
        } finally {
            this.getContainer().postInvoke(i);
        }
        if(i.exception instanceof
                java.lang.RuntimeException) {
          throw (java.lang.RuntimeException)i.exception;
        } else if(i.exception instanceof
                java.rmi.RemoteException) {
            throw (java.rmi.RemoteException)i.exception;
        } else if(i.exception != null) {
            throw new java.rmi.RemoteException("Unknown
                                exception", i.exception);
    }
        return $retVal;
  }
}
```

Như bạn thấy, để gián tiếp triệu gọi thành phần Bean trong trình chứa container, các lớp ConverterEJB_EJBObjectImpl và ConverterHomeImpl đã dùng kỹ thuật reflect (phản chiếu) đã được chúng ta sử dụng trong chương 2. Tóm lại, diễn biến thật sự khi triệu gọi thành phần Bean sẽ diễn ra như sau



Hình 9-13 Hoạt động của các lớp triệu gọi Bean

Tuy quá trình xây dựng phải qua các bước phức tạp nhưng thực tế nếu là nhà phát triển Bean bạn chỉ cần đầu tư vào xây dựng lớp ConverterEJB là đủ. Kiến trúc Enterprise Java Bean tách biệt giữa người phát triển Bean (Bean developer) và người cài đặt Bean (Bean deployer). Người cài đặt Bean phải sử dụng thành thạo trình đóng gói, tạo ra các lớp trung gian để đưa Bean vào trình chứa, hướng dẫn lập trình viên phía trình khách kết nối với Bean, chuyển giao cho trình khách lớp hỗ trợ để triệu gọi Bean ...

Hình 9-13 cho bạn thấy rõ hơn hoạt động của lời triệu gọi Bean đến các lớp liên quan. Trình khách sử dụng dịch vụ JNDI tham chiếu đến lớp chủ (1). Lớp chủ sẽ triệu gọi phương thức tạo ra tham chiếu đến lớp giao tiếp (interface) của Bean trả về trình khách (2). Trình khách triệu gọi Bean thông qua lớp giao tiếp trung gian (3) do lớp chủ trả về. Lớp trung gian này là giao tiếp giữa trình chứa container và đối tượng Bean thật sự.

Thật sự thì toàn bộ đối tượng Bean của bạn được đóng gói trong tập tin .jar. Bạn có thể mở thư mục

C:\j2sdkee1.2.1\repository\<computer_name>\applications

trình đóng gói tạo ra gói Bean Converter như là một ứng dụng jar ConverterApp862815730623.jar. Khi bạn khởi động lại trình chủ j2ee tập tin .jar này sẽ được bung ra trở lại vào thư mục .\gnrtrTMP cùng cấp với thư mục .\applications.

5. THIẾT KẾT BEAN THAO TÁC LƯU VẾT TRẠNG THÁI (STATEFULL SESSION BEAN)

ConverterEJB mà bạn xây dựng trên đây là thành phần Bean EJB thao tác phi trạng thái (stateless session bean). Có những thành phần Bean đòi hỏi phải lưu giữ thông tin trong suốt quá trình giao dịch. CartEJB dưới đây là một ví dụ về Bean thao tác lưu vết trạng thái. CartEJB phục vụ thao tác mua bán sách, khi bạn chọn mua một quyển sách, bạn gọi CartEJB để thêm vào danh sách đặt mua. Sau khi hoàn tất quá trình chọn sách, bạn gọi CartEJB để lấy về danh mục các tựa sách đã chọn.

Thành phần CartEJB hoạt động bên trong trình chứa container có mã nguồn cài đặt như sau

Ví dụ 9-8 CartEJB.java

```
import java.util.*;
import javax.ejb.*;
public class CartEJB implements SessionBean {
  String customerName;
  String customerId;
  Vector contents;
```

```
//Tạo đối tượng EJB dựa vào tên khách hàng
public void ejbCreate(String person)
                    throws CreateException {
  if (person == null) {
    throw new CreateException("Null
                    person not allowed.");
  }
  else {
    customerName = person;
  }
  customerId = "0";
  contents = new Vector();
}
//Tạo đối tượng EJB dựa vào tên và mã số khách hàng
public void ejbCreate(String person, String id)
  throws CreateException {
  if (person == null) {
    throw new CreateException("Null person
                not allowed.");
  }
  else {
    customerName = person;
  }
    //Tạo đối tượng kiểm tra định danh của khách hàng sử dụng
Bean
  IdVerifier idChecker = new IdVerifier();
```

```
    if (idChecker.validate(id)) {
      customerId = id;
    }
    else {
      throw new CreateException("Invalid id: " + id);
    }
    contents = new Vector();
  }
  //Thêm tựa sách vào danh mục
  public void addBook(String title) {
    contents.addElement(title);
  }
  //Loại tựa sách ra khỏi danh mục
  public void removeBook(String title)
                              throws BookException {
    boolean result = contents.removeElement(title);
    if (result == false) {
      throw new BookException(title + " not in cart.");
    }
  }
  //Trả về danh mục sách đã chọn
  public Vector getContents() {
    return contents;
  }
  public CartEJB() {
  }
```

```
  //Các phương thức triệu gọi bởi trình chứa
  public void ejbRemove() {
    System.out.println("Bean remove");
  }
  public void ejbActivate() {
    System.out.println("Bean Activate");
  }
  public void ejbPassivate() {
    System.out.println("Bean Passivate");
  }
  public void setSessionContext(SessionContext sc) {
  }
}
```

Vì CartEJB là một Bean thao tác (session Bean) cho nên theo qui định của kiến trúc Enterprise Java Bean bạn phải cài đặt giao tiếp SessionBean (tương tự như Converter đã làm). CartEJB cung cấp hai phương thức khởi tạo Bean

```
public void ejbCreate(String person, String id)
```

và

```
public void ejbCreate(String person)
```

trình chứa container sẽ tự gọi phương thức này của Bean khi trình khách yêu cầu Bean khởi tạo. Phương thức thứ nhất yêu cầu bạn tạo Bean với tên khách hàng và số định danh id dành cho khách hàng khi tham gia mua sách. Phương thức khởi tạo thứ hai chỉ yêu cầu tên khách hàng, định danh khách hàng được mặc định là "0". CartEJB xây dựng lớp IdVerifier để kiểm tra định danh của khách hàng mua sách như sau

Ví dụ 9-9 IdVerifier.java

```
 * bằng luồng xuất nhập DataOutputStream / DataInputStream
 */
public class DataStreamEcho extends HttpServlet
{
 public void service(HttpServletRequest req,
                     HttpServletResponse resp)
      throws ServletException, java.io.IOException
  {
   // Lấy về luồng nhập để đọc dữ liệu do trình khách gửi đến
   DataInputStream in =
      new DataInputStream(req.getInputStream());
   // Định dữ liệu trả về cho trình khách là dữ liệu nhị phân
   resp.setContentType("application/octet-stream");
   // Chúng ta sử dụng bộ đệm kết xuất dữ liệu trước khi gửi đi
   //Sử dụng luồng đệm sẽ cho biết chiều dài thực tế của dữ liệu
    //cần chuyển
   ByteArrayOutputStream byteOut = new
                        ByteArrayOutputStream();
   // Tạo luồng xuất đưa dữ liệu vào bộ đệm
   DataOutputStream out = new
                      DataOutputStream(byteOut);
   //Đọcdữ liệu do trình khách gửi đến
   boolean booleanValue = in.readBoolean();
   byte byteValue = in.readByte();
   char charValue = in.readChar();
   short shortValue = in.readShort();
   int intValue = in.readInt();
   float floatValue = in.readFloat();
   double doubleValue = in.readDouble();
   String stringValue = in.readUTF();
```

```
    // Phản hồi - ghi dữ liệu nhận được từ trình khách ra luồng xuất
    out.writeBoolean(booleanValue);
    out.writeByte(byteValue);
    out.writeChar(charValue);
    out.writeShort(shortValue);
    out.writeInt(intValue);
    out.writeFloat(floatValue);
    out.writeDouble(doubleValue);
    out.writeUTF(stringValue);
    // Yêu cầu luồng xuất chuyển dữ liệu ra bộ đệm
    out.flush();
    // Lấy về mảng nhị phân chứa dữ lliệu
    byte[] buf = byteOut.toByteArray();
    // định chiều dài dữ liệu cần gửi
    resp.setContentLength(buf.length);
    // Yêu cầu servlet lấy về luồng xuất chuẩn bị gửi dữ liệu
    ServletOutputStream servletOut =
                    resp.getOutputStream();
    // Gửi dữ liệu về cho trình khách
    servletOut.write(buf);
    servletOut.close();
  }
}
```

Chạy chương trình

```
>java TestDataStream
    http://localhost:8080/servlet/DataStreamEcho
Attempting to connect to
    http://localhost:8080/servlet/DataStreamEcho
Writing test data
Reading response
Data read: true 1 2 3 4 5.0 6.0 Hello, Java
```

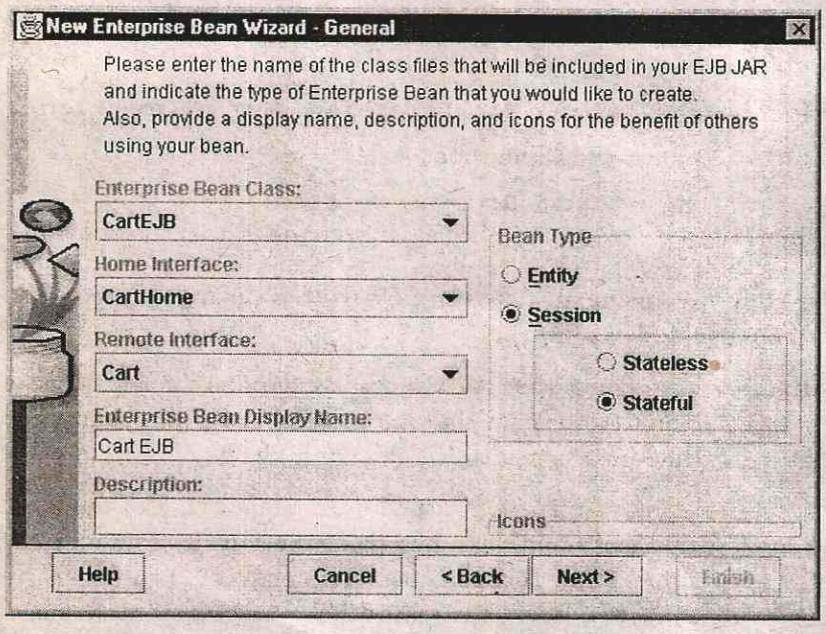
```
public interface Cart extends EJBObject {
  public void addBook(String title)
                throws RemoteException;
  public void removeBook(String title)
                throws BookException, RemoteException;
  public Vector getContents()
                throws RemoteException;
}
```

Như vậy, tương tự đối tượng Converter thường đối tượng EJB của bạn sẽ cần đến 3 lớp:

- Lớp Bean thực tế (CartEJB): lớp này hoạt động bên trong trình chứa container và cung cấp các phương thức thao tác thực sự có ích cho trình khách.
- Lớp giao tiếp xa (Cart): lớp này làm môi giới trung gian giữa đối tượng Bean bên trong trình chứa và trình khách. Trình chứa container sẽ kiểm soát và điều khiển hoạt động của Bean khi trình khách triệu gọi thông qua lớp giao tiếp này
- Lớp giao tiếp chủ (CartHome): Trình khách sẽ sử dụng lớp này để tạo ra lớp giao tiếp xa (Cart) trước khi triệu gọi được phương thức của Bean.

Để đưa CartEJB vào trình chứa container của J2EE, bạn biên dịch mã nguồn CartEJB.java, CartHome.java và Cart.java. Bạn dùng trình đóng gói deploytool để tạo ra gói .jar chứa thành phần Bean đồng thời chuyển giao Bean cho trình chứa J2EE. Bạn cũng nên yêu cầu trình đóng gói trả về tập tin CartAppClient.jar chứa các lớp cần dùng cho trình khách (xem lại cách đóng gói thành phần Bean ở mục 3.2.2). Bạn đặt tên JNDI cho thành phần Cart Bean là MyCart

(xem lại mục 3.2.2 hình 9-11). Nhớ chọn phần *Bean Type* trong quá trình đóng gói đặc tính Stateful (hình 9-14)



Hình 9-14 Cửa sổ chọn đặc tính Stateful cho Bean

Trình khách triệu gọi đối tượng CartEJB được viết như sau

Ví dụ 9-13 CartClient.java

```
import java.util.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
public class CartClient {
  public static void main(String[] args) {
    try {
```

```
  //Khởi tạo môi trường
Context initial = new InitialContext();
  //Lấy về tham chiếu của lớp home CartHome
Object objref = initial.lookup("MyCart");
  //Chuyển kiểu tham chiếu
CartHome home =
   (CartHome)PortableRemoteObject.narrow(objref,
                    CartHome.class);
Cart shoppingCart =
            home.create("Mr. Duke","123");
  //Triệu gọi phương thức Bean để thêm vào tựa sách
shoppingCart.addBook("Java Book");
shoppingCart.addBook("Programming Visual Basic");
shoppingCart.addBook("Network programming");
  //Lấy về danh mục các tựa sách đã chọn
Vector bookList = new Vector();
bookList = shoppingCart.getContents();
Enumeration enumer = bookList.elements();
  //In các tựa sách trong danh mục
while (enumer.hasMoreElements()) {
  String title = (String) enumer.nextElement();
  System.out.println(title);
}
  //Gọi các phương thức loại bỏ tựa sách khỏi danh mục
shoppingCart.removeBook("Programming
                              Visual Basic");
shoppingCart.remove();
```

```
    } catch (BookException ex) {
      System.err.println("Caught a BookException: "
                                    + ex.getMessage());
    } catch (Exception ex) {
      System.err.println("Caught an
                                unexpected exception!");
      ex.printStackTrace();
    }
  }
}
```

Bạn biên dịch và chạy chương trình CartClient như sau:

```
REM   Chỉ định thư viện cần dùng cho quá trình biên dịch
set CPATH=.;%J2EE_HOME%\lib\j2ee.jar;
                              CartAppClient.jar
REM   Biên dịch chương trình
     javac -classpath %CPATH% CartClient.java
REM   Chạy chương trình
     java -classpath %CPATH% CartClient.java
```

Kết quả kết xuất

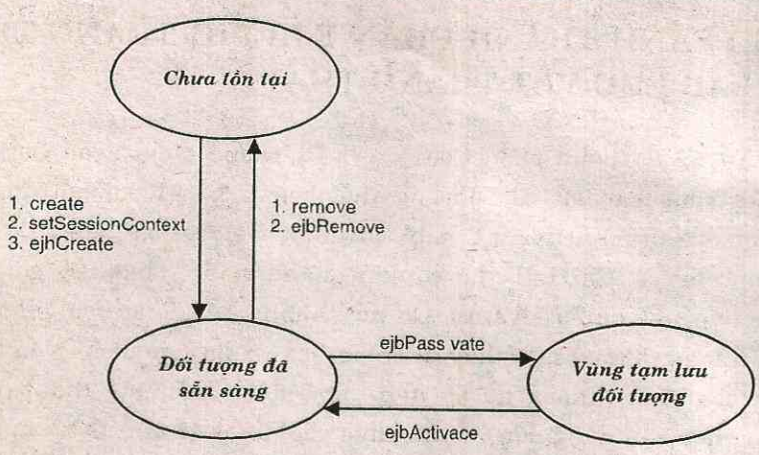
```
Java Book;
Programming Visual Basic
Network programming
```

Như bạn thấy từ kết quả kết xuất, biến thể hiện cho lớp CartEJB là shoppingCart có thể được triệu gọi nhiều lần. Mỗi lần gọi giao dịch trước đó (thêm sách vào danh mục) trạng thái được Bean lưu giữ lại. Bạn gọi getContent() để yêu cầu shoppingCart trả về danh mục sách cuối cùng.

6. SO SÁNH THÀNH PHẦN EJB PHI TRẠNG THÁI VÀ EJB LƯU VẾT TRẠNG THÁI

Tương tự thành phần ConverterEJB, thành phần Bean CartEJB của ta cũng cài đặt các phương thức mà SessionBean qui định là ejbRemove(), ejbActivate(), ejbPasivate(), setSessionContext(). Tuy nhiên chỉ có CartEJB là có khả năng trình chứa sẽ gọi đến ejbActive() và ejbPassivate(). Do các thành phần Bean lưu giữ trạng thái thường không được hủy đi trong quá trình giao dịch. Nếu trình chứa container không đủ bộ nhớ (hoặc đối với những thành phần Bean chưa cần dùng đến), trình chứa container sẽ tạm thời lưu Bean xuống bộ nhớ ngoài (đĩa cứng). Điều này không cần thiết đối với Bean phi trạng thái. Trình chứa container có thể hủy Bean phi trạng thái sau mỗi lần trình khách triệu gọi xong phương thức của Bean.

Phương thức ejbPassivate() được triệu gọi để thông báo cho Bean biết rằng nó sắp được lưu xuống vùng nhớ ngoài. Các Bean lưu vết trạng thái có thể dựa vào tình huống này để tạm cất những dữ liệu quan trọng vào nơi nào đó (như vào registry của Windows, lưu thông tin vào file tạm ...), sau đó thì Bean chuyển sang trạng thái "ngủ" (ngừng hoạt động). Khi có một lời triệu gọi Bean phát sinh mới từ trình khách, nếu trình chứa container không tìm thấy Bean đang hoạt động nó sẽ khôi phục lại Bean từ vùng nhớ tạm. Lúc này trình chứa sẽ gọi phương thức ejbActivate() để thông báo cho Bean biết trạng thái kích hoạt trở lại. Bean có thể dựa vào tình huống này để khôi phục lại trạng thái đã lưu trữ trước đó (trong phương thức ejbPassivate()). Hình 9-15 là cơ chế hoạt động cũng như điều khiển Bean lưu vết trạng thái của trình chủ Container.



Hình 9-15 Kích hoạt và tạm ngưng hoạt động của Bean trạng thái (Stateful Session Bean)

Với trình khách CartClient ở ví dụ trên, bạn hãy thử dùng lớp chủ triệu gọi phương thức tạo Bean nhiều lần. Ví dụ

```
Vector  cartList=new Vector();
for (int i=0; i<10000; i++){
  Cart cart=home.create();
  cartList.addElement(cart);
}
```

tiếp đến hãy thử triệu gọi từng đối tượng cart trong danh sách cartList, quan sát cửa sổ trình chứa (nơi j2ee đang chạy) bạn sẽ thấy thông báo của phương thức ejbActive() và ejbPassivate() được gọi.

7. XÂY DỰNG BEAN THỰC THỂ TỰ QUẢN LÝ (ENTITY BEAN-MANAGED)

Ở mục 2.2 bạn đã tìm hiểu Bean thực thể (Entity Bean) là đối tượng Bean ánh xạ dữ liệu từ môi trường ngoài như các dòng dữ liệu

trong bảng (table) dữ liệu chẳng hạn. Nếu Bean thao tác (session bean) chủ yếu thiên về xử lý tính toán và hoàn thành một tác vụ nào đó thì Bean thực thể thiên về việc thể hiện dữ liệu. Trong Bean thực thể bạn có thể có các phương thức tính toán kèm theo, chúng được gọi là các phương thức nghiệp vụ (business method) xử lý dữ liệu của Bean. Tuy nhiên bạn cũng có thể chỉ dùng Bean thực thể để trình bày dữ liệu, Bean thao tác sẽ được dùng để xử lý đối tượng Bean thực thể. Đây là cách kết hợp nên áp dụng giữa Bean thực thể và Bean thao tác.

Điểm khác biệt lớn nhất giữa Bean thực thể và Bean thao tác đó là Bean thực thể có khả năng lưu trữ và tái tạo lại bản thân Bean từ cơ sở dữ liệu. Đặc tính này thường gọi là khả năng tồn tại và lưu trữ được (persistence) Nếu như bạn đã từng làm việc với cơ sở dữ liệu thì nó cũng tương tự như việc lưu trữ dữ liệu . Dữ liệu trong cơ sở dữ liệu được lưu trữ lại và nó vẫn tồn tại thậm chí sau khi bạn đã thoát ra khỏi cơ sở dữ liệu server hoặc ra khỏi ứng dụng phục vụ .

Có hai loại kiểm soát Bean thực thể: Bean thực thể tự quản lý và Bean thực thể quản lý bởi trình chứa. Phần này chúng ta sẽ xây dựng một đối tượng Bean thực thể tượng trưng cho tài khoản ngân hàng (tài khoản ngân hàng thực sự là một đối tượng dữ liệu tồn tại cần được lưu trữ dài hạn).

Như đã nêu, Bean thực thể là thành phần ánh xạ từ cơ sở dữ liệu nên trước hết bạn phải tạo bảng dữ liệu bằng lệnh SQL có cấu trúc sau

```
CREATE TABLE account (
        id VARCHAR(3)
                CONSTRAINT pk_account PRIMARY KEY,
        firstname VARCHAR(24),
        lastname VARCHAR(24),
        balance DECIMAL(10,2));
```

Lưu ý, để tạo bảng dữ liệu bạn có thể dùng hệ Access, MS SQL Server hay Oracle. Tuy nhiên Sun cung cấp cho bạn hệ cơ sở dữ liệu Infomix Cloudscape miễn phí kèm theo bộ SDK. Phần sau sẽ hướng dẫn bạn cách tạo bảng dữ liệu chi tiết hơn bằng hệ Cloudscape.

Thành phần Bean thực thể ngoài việc thể hiện dữ liệu trong bản thân Bean khác với Bean thao tác (session Bean) còn thì mô hình hoạt động hoàn toàn tương tự thành phần Bean thao tác. Chúng ta xây dựng đối tượng thực thể Bean Account với những lớp cơ bản sau

Lớp	Mục đích
AccountEJB	Đối tượng Bean thực thể hoạt động bên trong trình chứa container.
Account	Lớp giao tiếp làm trung gian để trình khách triệu gọi Bean thông qua trình chứa.
AccountHome	Lớp chủ cho phép tạo ra đối tượng tham chiếu Account từ trình khách.

Mã nguồn của AccountEJB được trình bày ở ví dụ 9-14 sau đây

Ví dụ 9-14 AccountEJB.java

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.ejb.*;
import javax.naming.*;
public class AccountEJB implements EntityBean {
```

```
private String id;
private String firstName;
private String lastName;
private double balance;
private EntityContext context;
private Connection con;
//Tên tham chiếu đến đối tượng cho phép truy xuất cơ sở dữ liệu
private String
        dbName = "java:comp/env/jdbc/AccountDB";
//Phương thức cho phép ghi nợ vào tài khoàn
public void debit(double amount)
        throws InsufficientBalanceException {
  if (balance - amount < 0) {
    throw new InsufficientBalanceException();
  }
  balance -= amount;
}
//Phương thức cho phép ghi có vào tài khoàn
public void credit(double amount) {
  balance += amount;
}
//Trả về tên chủ tài khoản
public String getFirstName() {
  return firstName;
}
public String getLastName() {
```

```
    return lastName;
  }
  //Trả về số cân đối của tài khoản
  public double getBalance() {
    return balance;
  }
  /* Phương thức do trình chứa container điều khiển */
  //Được gọi để tạo đối tượng Bean
  public String ejbCreate(String id,
                          String firstName,
                          String lastName, double balance)
    throws CreateException {
    //Tài khoản âm
    if (balance < 0.00) {
      throw new CreateException
        ("A negative initial balance is not allowed.");
    }
    try {
    //Tạo mới dữ liệu của tài khoản tượng trưng cho thực thể Bean
      insertRow(id, firstName, lastName, balance);
    } catch (Exception ex) {
      throw new EJBException("ejbCreate: " +
        ex.getMessage());
    }
    //Lưu lại dữ liệu mà Bean thể hiện
    this.id = id;
    this.firstName = firstName;
```

```
    this.lastName = lastName;
    this.balance = balance;
    return id;
  }
```

//Phương thức này được triệu gọi để truy tìm dữ liệu của đối tượng Bean

//theo khóa chính (primary key) từ bảng dữ liệu

```
  public String ejbFindByPrimaryKey(String primaryKey)
    throws FinderException {
    boolean result;
    try {
      result = selectByPrimaryKey(primaryKey);
    } catch (Exception ex) {
      throw new EJBException("ejbFindByPrimaryKey: " +
        ex.getMessage());
    }
    if (result) {
      return primaryKey;
    }
    else {
      throw new ObjectNotFoundException
        ("Row for id " + primaryKey + " not found.");
    }
  }
```

//Phương thức này được triệu gọi để truy tìm dữ liệu của đối tượng Bean

```
// Bean theo tên và trả về danh sách các tài khoản đáp ứng yêu
cầu tìm kiếm
public Collection ejbFindByLastName(String lastName)
                    throws FinderException {
  Collection result;
  try {
    result = selectByLastName(lastName);
  } catch (Exception ex) {
    throw new EJBException("ejbFindByLastName " +
      ex.getMessage());
  }
  if (result.isEmpty()) {
    throw new ObjectNotFoundException("No
                                  rows found.");
  }
  else {
    return result;
  }
}
//Phương thức này được triệu gọi để truy tìm các đối tượng Bean
//co số dư nợ/có trong một khoảng nào đó
public Collection ejbFindInRange(double low,
                                  double high)
  throws FinderException {
  Collection result;
  try {
    result = selectInRange(low, high);
```

```
    } catch (Exception ex) {
      throw new EJBException("ejbFindInRange: " +
        ex.getMessage());
    }
    if (result.isEmpty()) {
      throw new ObjectNotFoundException("No rows
                                          found.");
    }
    else {
      return result;
    }
  }
//Phương thức này được triệu gọi để loại bỏ đối tượng Bean. điều này
//cũng đồng nghĩa với việc xóa bỏ dữ liệu khỏi bảng dữ liệu trong cơ sở
// dữ liệu quan hệ
public void ejbRemove() {
    try {
      deleteRow(id);
    } catch (Exception ex) {
      throw new EJBException("ejbRemove: " +
        ex.getMessage());
    }
  }
//Phương thức này được triệu gọi để truyền thông tin từ môi trường trình
//chứa vào trong Bean
public void setEntityContext(EntityContext context) {
    this.context = context;
```

```
    try {
      makeConnection();
    } catch (Exception ex) {
      throw new EJBException("Unable to connect
                    to database. " + ex.getMessage());
    }
  }
  public void unsetEntityContext() {
    try {
      con.close();
    } catch (SQLException ex) {
      throw new EJBException("unsetEntityContext: "
                                    + ex.getMessage());
    }
  }
  //Phương thức này được gọi khi trình chứa kích hoạt Bean
  public void ejbActivate() {
    id = (String)context.getPrimaryKey();
  }
  public void ejbPassivate() {
    id = null;
  }
  // Phương thức này được gọi khi trình chứa yêu cầu Bean nạp lại dữ liệu
  //từ bảng cơ sở dữ liệu
  public void ejbLoad() {
    try {
```

```
        loadRow();
      } catch (Exception ex) {
        throw new EJBException("ejbLoad: " +
          ex.getMessage());
      }
    }
    // Phương thức này được gọi khi trình chứa yêu cầu Bean lưu lại dữ liệu
    //vào bảng cơ sở dữ liệu
    public void ejbStore() {
      try {
        storeRow();
      } catch (Exception ex) {
        throw new EJBException("ejbLoad: " +
          ex.getMessage());
      }
    }
    // Trình chứa triệu gọi phương thức này để thông báo cho Bean biết dữ
    // liệu vừa được chèn vào bảng dữ liệu
    public void ejbPostCreate(String id, String firstName,
                              String lastName, double balance)
    {
    }
    /*      Các hàm phục vụ cho kết nối và truy xuất cơ sở dữ liệu
    */
    //Hàm tạo kết nối
    private void makeConnection() throws NamingException,
                                         SQLException {
```

```
    InitialContext ic = new InitialContext();
    //Lấy về tham chiếu của đối tượng cơ sở dữ liệu
    DataSource ds = (DataSource) ic.lookup(dbName);
    //Lấy về đối tượng kết nối
    con = ds.getConnection();
  }
  //Chèn dữ liệu vào bảng
  private void insertRow (String id, String firstName,
                          String lastName, double balance)
                                  throws SQLException {
      String insertStatement =
          "insert into account values ( ? , ? , ? , ? )";
      PreparedStatement prepStmt =
          con.prepareStatement(insertStatement);
      prepStmt.setString(1, id);
      prepStmt.setString(2, firstName);
      prepStmt.setString(3, lastName);
      prepStmt.setDouble(4, balance);
      prepStmt.executeUpdate();
      prepStmt.close();
  }
  //Xóa dữ liệu khỏi bảng
  private void deleteRow(String id) throws SQLException {
    String deleteStatement =
        "delete from account where id = ? ";
    PreparedStatement prepStmt =
```

```
        con.prepareStatement(deleteStatement);
    prepStmt.setString(1, id);
    prepStmt.executeUpdate();
    prepStmt.close();
  }
  //Truy tìm tài khoản theo khóa chính
  private boolean selectByPrimaryKey(String primaryKey)
                                throws SQLException {
    String selectStatement =
        "select id from account where id = ? ";
    PreparedStatement prepStmt =
        con.prepareStatement(selectStatement);
    prepStmt.setString(1, primaryKey);
    ResultSet rs = prepStmt.executeQuery();
    boolean result = rs.next();
    prepStmt.close();
    return result;
  }
  //Truy tìm danh sách tài khoản theo tên chủ tài khoản
  private Collection selectByLastName(String lastName)
    throws SQLException {
    String selectStatement =
        "select id " +
        "from account where lastname = ? ";
    PreparedStatement prepStmt =
        con.prepareStatement(selectStatement);
```

```
  prepStmt.setString(1, lastName);
  ResultSet rs = prepStmt.executeQuery();
  ArrayList a = new ArrayList();
  while (rs.next()) {
    String id = rs.getString(1);
    a.add(id);
  }
  prepStmt.close();
  return a;
}
//Truy tìm danh sách tài khoản theo số dư/nợ có
private Collection selectInRange(double low,
                    double high) throws SQLException {
  String selectStatement =
      "select id from account " +
      "where balance between ? and ?";
  PreparedStatement prepStmt =
      con.prepareStatement(selectStatement);
  prepStmt.setDouble(1, low);
  prepStmt.setDouble(2, high);
  ResultSet rs = prepStmt.executeQuery();
  ArrayList a = new ArrayList();
  while (rs.next()) {
    String id = rs.getString(1);
    a.add(id);
  }
```

```
    prepStmt.close();
    return a;
  }
  //Phương thức này dùng nạp dữ liệu từ bảng vào Bean
  private void loadRow() throws SQLException {
    String selectStatement =
        "select firstname, lastname, balance " +
        "from account where id = ? ";
    PreparedStatement prepStmt =
        con.prepareStatement(selectStatement);
    prepStmt.setString(1, this.id);
    ResultSet rs = prepStmt.executeQuery();
    if (rs.next()) {
      this.firstName = rs.getString(1);
      this.lastName = rs.getString(2);
      this.balance = rs.getDouble(3);
      prepStmt.close();
    }
    else {
      prepStmt.close();
      throw new NoSuchEntityException("Row for id " + id +
        " not found in database.");
    }
  }
  //Phương thức này dùng lưu dữ liệu từ Bean vào bảng
  private void storeRow() throws SQLException {
```

```
    String updateStatement =
        "update account set firstname = ? ," +
        "lastname = ? , balance = ? " +
        "where id = ?";
    PreparedStatement prepStmt =
        con.prepareStatement(updateStatement);
    prepStmt.setString(1, firstName);
    prepStmt.setString(2, lastName);
    prepStmt.setDouble(3, balance);
    prepStmt.setString(4, id);
    int rowCount = prepStmt.executeUpdate();
    prepStmt.close();
    if (rowCount == 0) {
      throw new EJBException("Storing row for id "
                                    + id + " failed.");
    }
  }
} // AccountEJB
```

Nếu đối tượng Bean thao tác cần cài đặt giao tiếp SessionBean thì đối tượng Bean thực thể yêu cầu bạn cài đặt giao tiếp EntityBean. Giao tiếp này hỗ trợ thêm các phương thức để trình chứa container điều khiển Bean thực thể như ejbLoad(), ejbStore().

Lớp AccountEJB sử dụng ngoại lệ InsufficientBalanceException để thông báo về tình trạng tài khoản hết số dư, lớp này do ta tự tạo và có mã nguồn như sau

Ví dụ 9-15 InsufficientBalanceException.java

```
public class InsufficientBalanceException
                                extends Exception {
  public InsufficientBalanceException() { }
  public InsufficientBalanceException(String msg) {
    super(msg);
  }
}
```

Như vậy bạn đã xây dựng xong thành phần Bean thực thể AccountEJB. Do Bean thực thể của ta tự quản lý cho nên việc kết nối, truy vấn, chèn dữ liệu vào bảng dữ liệu bạn phải tự thực hiện bằng các hàm như storeRow(), loadRow().

Lớp giao tiếp trung gian giữa trình chứa và thực thể Bean được xây dựng như sau

Ví dụ 9-16 Account.java

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Account extends EJBObject {
  public void debit(double amount)
    throws InsufficientBalanceException,
           RemoteException;
  public void credit(double amount)
    throws RemoteException;
  public String getFirstName()
    throws RemoteException;
  public String getLastName()
    throws RemoteException;
  public double getBalance()
```

```
    throws RemoteException;
}
```

Tiếp theo dưới đây là mã nguồn của lớp chủ AccountHome được trình khách tham chiếu để tạo ra lớp Account

Ví dụ 9-17 AccountHome.java

```
import java.util.Collection;
import java.rmi.RemoteException;
import javax.ejb.*;
public interface AccountHome extends EJBHome {
  public Account create(String id,
                        String firstName,
                        String lastName)
    throws RemoteException, CreateException;
  public Account findByPrimaryKey(String id)
          throws FinderException, RemoteException;
  public Collection findByLastName(String lastName)
    throws FinderException, RemoteException;
  public Collection findInRange(double low, double high)
    throws FinderException, RemoteException;
}
```

Khác với lớp chủ của Bean thao tác (session Bean), lớp chủ của Bean thực thể ngoài việc cung cấp phương thức create() để tạo ra giao tiếp Account cho trình khách, còn có thể cung cấp thêm các phương thức tìm kiếm như findPrimaryKey(), findByLastName(), findInRange() để tìm lại đối tượng Bean. Bạn để ý những phương thức này sẽ tương ứng với lời gọi ejbFindPrimaryKey(), ejbFindByLastName(), ejbFindInRange() mà ta cài đặt bên trong AccountEJB. Theo qui ước, khi bạn dùng lớp chủ (AccountHome)

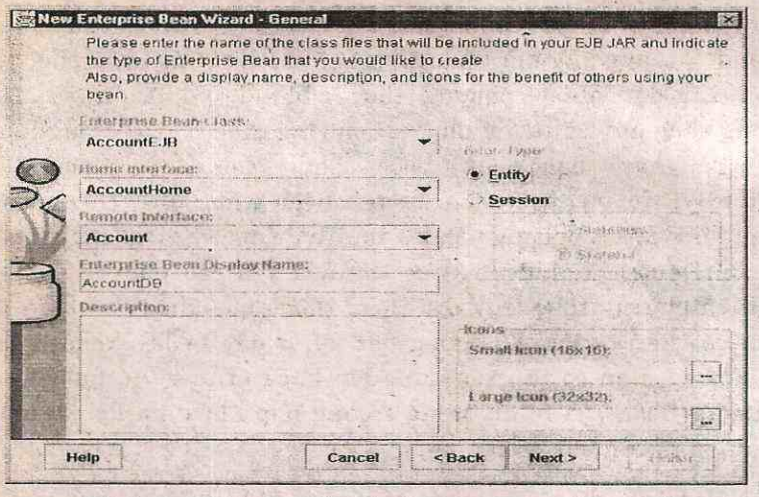
triệu gọi phương thức find_(), trình chứa sẽ triệu gọi phương thức ejbFind_() tương ứng mà Bean thực thể cài đặt. Các Bean thực thể có quyền cài đặt nhiều phương thức ejbFind_() khác nhau. Những phương thức này sẽ tương ứng với phương thức find_() trong lớp chủ. Mặc dù vậy bạn bắt buộc phải cài đặt phương thức ejbFindByPrimaryKey() bên trong Bean thực thể (AccountEJB), tương ứng với phương thức findByPrimaryKey() của lớp chủ (AccountHome). Mỗi Bean thực thể đều yêu cầu phải có một khóa chính và phương thức truy tìm Bean thông qua khóa chính đó. Khóa chính có thể là kiểu đối tượng bất kỳ, ở đây ta lấy khóa chính có kiểu String. Bạn có thể xây dựng lớp khóa primary tự tạo cho riêng đối tượng Bean của mình trong trường hợp khóa đòi hỏi sự kết hợp của nhiều cột dữ liệu trong bảng.

Nếu trình chứa container cần phải đồng bộ hóa biến thể hiện của một bean thực thể với các giá trị đặc trưng lưu trữ trong cơ sở dữ liệu, nó sẽ triệu gọi phương thức ejbLoad() và ejbStore() Phương thức ejbLoad() sẽ tái tạo biến thể hiện từ cơ sở dữ liệu. Phương thức ejbStore() sẽ ghi biến này vào cơ sở dữ liệu. Chương trình khách (client) không cần gọi hai phương thức ejbLoad() và ejbStore() trực tiếp.

Phương thức ejbLoad() yêu cầu phương thức loadRow() phát sinh ra lệnh SELECT của SQL và gán các dữ liệu trả về cho biến thể hiện .

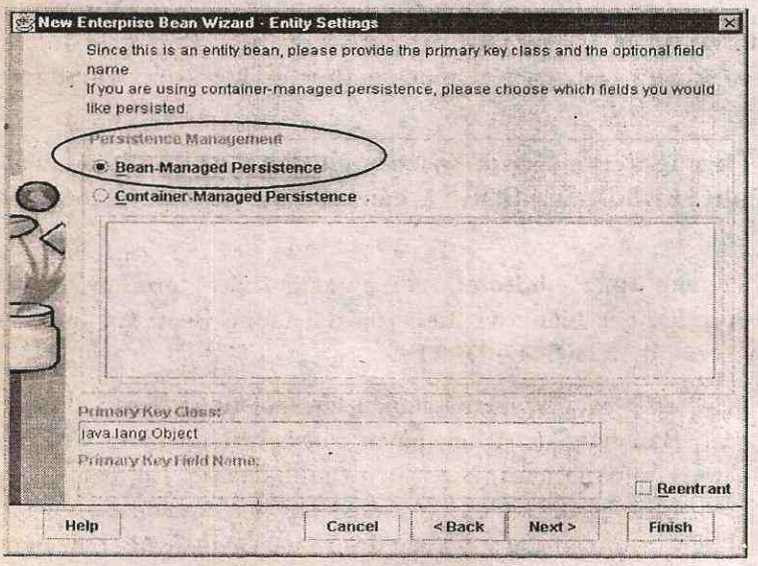
Phương thức ejbStore() gọi phương thức storeRow() , phương thức này lưu các biến thể hiện của đối tượng Bean trở lại cơ sở dữ liệu với câu lệnh SQL UPDATE.

Quá trình xây dựng mã nguồn cho đối tượng Bean thực thể đã hoàn tất. Bạn lưu tất cả mã nguồn của đối tượng Bean vào thư mục C:\network\ejb\account. Biên dịch thành các tập tin .class, tiếp đến dùng trình đóng gói deploytool thực hiện đưa AccountEJB vào trình chứa J2EE. Bạn cần lưu ý ở bước định kiểu *Bean Type* bạn nên chọn Entity Bean (hình 9-16).



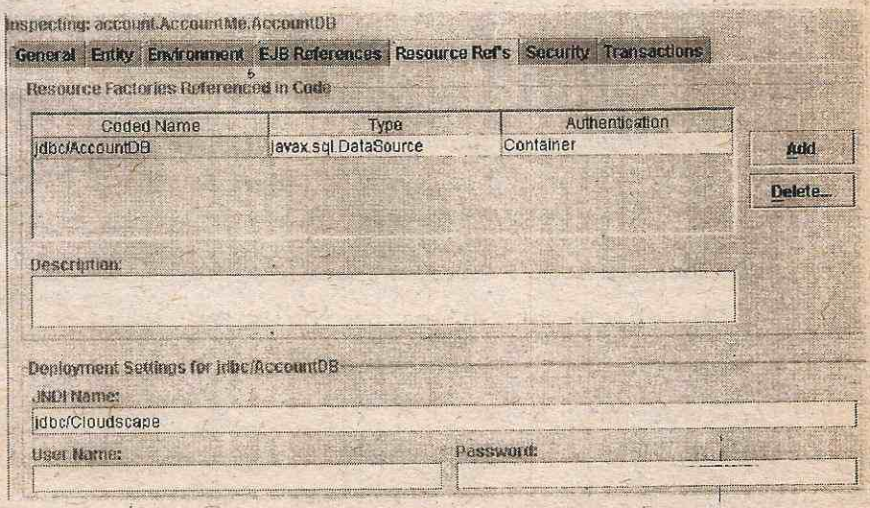
Hình 9-16 Định kiểu Entity cho Bean thực thể

Ở bước kế tiếp bạn chọn mục Bean-Managed để chỉ định thành phần Bean của ta tự quản lý việc truy xuất dữ liệu (Hình 9-17).



Hình 9-17 Cửa sổ chỉ định kiểu quản lý Bean

Bạn nhấn nút *Finish* để trở về cửa sổ deploytool chính. Chọn thành phần Bean từ cửa sổ bên trái đồng thời chọn tab-*Resource Ref's*. Bạn nhấn nút *Add* , thêm vào cột *Coded Name* chuỗi jdbc/AccountDB, chọn javax.sql.DataSource từ cột *Type*. Cuối cùng chọn Container từ cột *Authentication*. Ở ô *JNDI name* bên dưới bạn thêm vào chuỗi jdbc/Cloudscape. Đây là các thiết lập để bạn có thể truy xuất vào nguồn dữ liệu Cloudscape (hình 9-18). Thật ra, J2EE xây dựng cơ sở dữ liệu Cloudscape tương tự một đối tượng phân tán xa mà bất kỳ trình khách nào cũng đều có thể triệu gọi. Ở đây khi bạn khởi động trình chủ J2EE tên nguồn dữ liệu của Cloudscape đã được đặt là jdbc/Cloudscape. Bạn chỉ cần chỉ định tên này ở ô JNDI name là hoàn toàn dữ liệu có thể truy xuất được. Ngoài ra bạn cần đặt tên JNDI khác cho đối tượng Account của mình là MyAccount. Trình khách sẽ dựa vào tên MyAccount để truy tìm tham chiếu đến lớp chủ AccountHome.



Hình 9-18 Định tên và kiểu DataSource cho cơ sở dữ liệu cần gọi

Bạn chọn mục *Tool | Deploy Application* để đóng gói và đưa Bean vào trình chứa. Khi đóng gói Bean bạn nên yêu cầu trình

deploytool trả về tập tin AccountAppClient.jar chứa các lớp hỗ trợ cần dùng cho phía máy khách (chọn mục *Return Client Jar* trong cửa sổ hình 9-10).

Cuối cùng, dưới đây là mã nguồn của trình khách triệu gọi đối tượng Bean thực thể AccountEJB

Ví dụ 9-18 AccountClient.java

```
import java.util.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
public class AccountClient {
  public static void main(String[] args) {
    try {
      //Khởi tạo môi trường
      Context initial = new InitialContext();
      //Lấy về tham chiếu của lớp chủ AccountHome
      Object objref = initial.lookup("MyAccount");
      //Chuyển kiểu an toàn
      AccountHome home = (AccountHome)
          PortableRemoteObject.narrow(objref,
                                      AccountHome.class);
      //Tạo đối tượng Bean
      Account duke = home.create("123", "Duke",
                                 "Earl", 0.00);
      duke.credit(88.50);  //Ghi có tài khoản Duke
      duke.debit(20.25);   //Ghi nợ tài khoản Duke
      //In ra số cân đối của tài khoản
```

```
double balance = duke.getBalance();
System.out.println("balance = "
                + String.valueOf(balance));
duke.remove(); //Loại tài bỏ tài khoản
  //Tạo đối tượng tài khoản Bean khác
Account joe = home.create("836", "Joe",
                          "Jones", 0.00);
joe.credit(34.55); //Ghi có tài khoản
  //Truy tìm lại tài khoản
Account jones = home.findByPrimaryKey("836");
jones.debit(2.00);   //Ghi nợ tài khoản
  //Lấy về số cân đối của tài khoản
balance = jones.getBalance();
System.out.println("balance = "
                + String.valueOf(balance));
  //Tạo mới và ghi nợ/có các tài khoản khác
Account pat = home.create("456", "Pat",
                          "Smith", 0.00);
pat.credit(44.77);
Account john = home.create("730", "John",
                           "Smith", 0.00);
john.credit(19.54);
Account mary = home.create("268", "Mary",
                           "Smith", 0.00);
mary.credit(100.07);
  //Lấy về danh sách tất cả các tài khoản có tên là Smith
```

```
    Collection c = home.findByLastName("Smith");
    Iterator i=c.iterator();
    while (i.hasNext()) {
      Account account = (Account)i.next();
      String id = (String)account.getPrimaryKey();
      double amount = account.getBalance();
      System.out.println(id + ": "
                        + String.valueOf(amount));
    }
    //Lấy về các tài khoản có số dư nợ có trong khoản 20-99
    c = home.findInRange(20.00, 99.00);
    i=c.iterator();
    while (i.hasNext()) {
      Account account = (Account)i.next();
      String id = (String)account.getPrimaryKey();
      double amount = account.getBalance();
      System.out.println(id + ": "
                        + String.valueOf(amount));
    }
  } catch (InsufficientBalanceException ex) {
    System.err.println("Caught an
              InsufficientBalanceException: "
              + ex.getMessage());
  } catch (Exception ex) {
    System.err.println("Caught an exception." );
    ex.printStackTrace();
```

```
    }
  }
}
```

Bạn biên dịch AccountClient.java thành tập tin .class lưu vào thư mục C:\network\ejb\account.

Trước khi kiểm tra và chạy được chương trình bạn cần khởi động dịch vụ cơ sở dữ liệu của Cloudscape. Bạn hãy mở cửa sổ DOS-prompt riêng biệt, chuyển vào thư mục C:\j2sdkee1.2.1\bin đánh lệnh sau

Thiết lập biến môi trường trỏ đến thư mục cài đặt J2EE

```
set J2EE_HOME=C:\j2sdkee1.2.1
```

Khởi động cơ sở dữ liệu Cloudscape

```
C:\j2sdkee1.2.1\bin > cloudscape -start
```

Khởi động môi trường truy xuất cơ sở dữ liệu

```
C:\j2sdkee1.2.1\bin >java
-Dij.connection.CloudscapeDB=
      jdbc:rmi://localhost:1099/
      jdbc:cloudscape:CloudscapeDB\;create=true
-Dcloudscape.system.home=%J2EE_HOME%\cloudscape
-classpath %J2EE_HOME%\lib\cloudscape\client.jar;
           %J2EE_HOME%\lib\cloudscape\tools.jar;
           %J2EE_HOME%\lib\cloudscape\cloudscape.jar;
           %J2EE_HOME%\lib\cloudscape\RmiJdbc.jar;
           %J2EE_HOME%\lib\cloudscape\license.jar;
%CLASSPATH% -ms16m -mx32m COM.cloudscape.tools.ij
```

(Bạn chú ý viết các lệnh trên liền nhau, do không đủ chỗ nên chúng tôi phân dòng để bạn dễ hình dung)

Lệnh trên có thể tóm tắt như sau:

Dịch vụ cho phép kết nối được khởi động bằng lớp

```
java COM.cloudscape.tools.ij
```

Đối số dòng lệnh -Dij.connection.CloudscapeDB cho biết môi trường truy xuất cơ sở dữ liệu Cloudscape được xem như đối tượng phân tán RMI với chuỗi kết nối jdbc mang tên jdbc:cloudscape:CloudscapeDB.

Đối số dòng lệnh -Dcloudscape.system.home chỉ định thư mục chủ nơi Cloudscape hoạt động.

Đối số dòng lệnh -classpath chỉ định đường dẫn truy tìm các thư viện mà Cloudscape sẽ dùng đến.

Nếu lệnh thành công bạn sẽ thấy cơ sở dữ liệu Cloudscape khởi động và hiện ra dấu nhắc ij> sau

```
ij version 3.0 (c) 1997-2000 Informix Software, Inc.
CLOUDSCAPEDB* -
jdbc:cloudscape:CloudscapeDB\;create=true * = current
connection
ij>
```

Từ dấu nhắc này bạn có thể đánh mọi lệnh SQL như tạo bảng (CREATE TABLE), chèn (INSERT), xóa (DELETE), xem (SELECT) cập nhật dữ liệu (UPDATE). Bạn tạo bảng account phục vụ cho thực thể Bean AccountEJB bằng lệnh sau

```
ij> CREATE TABLE account (id VARCHAR(3)
                          CONSTRAINT pk_account PRIMARY KEY,
                          firstname VARCHAR(24),
                          lastname VARCHAR(24),
                          balance DECIMAL(10,2));
  0 rows inserted/updated/deleted
```

Mọi việc đã sẵn sàng, bạn chạy trình khách từ dòng lệnh như sau:

- Chỉ định đường dẫn classpath

```
C:\network\ejb\account>set
        classpath=.;C:\jsdkee1.2.1\lib\j2ee.jar;
                    AccountAppClient.jar
```

- Chạy trình khách

```
C:\network\ejb\account>java  AcountClient
```

- Kết quả kết xuất

```
balance = 68.25
balance = 32.55
456: 44.77
730: 19.54
268: 100.07
```

8. XÂY DỰNG BEAN THỰC THỂ DO TRÌNH CHỨA QUẢN LÝ (ENTITY BEAN CONTAINER-MAINAGED)

Trên đây là ví dụ về cách xây dựng đối tượng Bean thực thể tự quản lý ánh xạ dữ liệu từ các bảng dữ liệu bằng kết nối Jdbc. Như bạn thấy, khi tự quản lý việc truy xuất dữ liệu bạn phải tự thiết kế các hàm như

```
makeConnection();  //Tạo kết nối với cơ sở dữ liệu
storeRow();        //Lưu dữ liệu của Bean trở lại bảng
loadRow();         //Nạp dữ liệu từ bảng vào Bean
deleteRow();       //Xóa dữ liệu tương ứng của Bean trong bảng
```

```
insertRow();            //Chèn dữ liệu mới vào bảng
selectInRange();        //Chọn dữ liệu theo số dư
selectByLastName();     //Chọn dữ liệu theo tên
```

Việc lưu trữ và khôi phục dữ liệu của Bean đôi khi là một thao tác nhàm chán vì thật ra nó không thể hiện thực sự mục đích hoạt động của Bean. Kiến trúc Enterprise Java Bean cho phép bạn xây dựng các Bean thực thể tách hẳn cơ chế lưu trữ và cơ chế thao tác nghiệp vụ. Trình chứa container sẽ đảm trách việc kết nối, chèn, thêm, xóa dữ liệu. Cụ thể hơn trình chứa container sẽ giúp các nhà phát triển Bean tập trung vào xây dựng các phương thức nghiệp vụ. Loại bỏ thao tác tương tác với cơ sở dữ liệu không cần thiết.

Thành phần Bean thực thể chuyển giao thao tác truy xuất lại cho trình chứa container quản lý được gọi là Bean thực thể quản lý bởi trình chứa (container-managed Bean).

Xây dựng Bean thực thể quản lý bởi trình chứa không có gì khác với khi xây dựng Bean thực thể tự quản lý (như AccountEJB). Chỉ có bạn không cần phải viết các hàm kết nối hay sử dụng các lệnh SQL để tương tác với cơ sở dữ liệu. Công việc này được thực hiện trong quá trình đóng gói Bean bằng công cụ deploytool.

Dưới đây là ví dụ về thành phần Bean thực thể ProductEJB được quản lý bởi trình chứa. Đối tượng ProductEJB Bean mô tả thực thể một loại sản phẩm nào đó. Bạn có thể gọi getPrice(), setPrice() để thiết lập và lấy về giá bán của sản phẩm, getDescription() sẽ trả về tên mô tả của sản phẩm. Cài đặt cho ProductEJB được trình bày ở ví dụ 9-19

Ví dụ 9-19 ProductEJB.java

```
import java.util.*;
import javax.ejb.*;
public class ProductEJB implements EntityBean {
```

```
        //Các trường tương ứng với các cột dữ liệu trong bảng
dữ liệu
public String productId;
public String description;
public double price;
//Đối tượng lưu giữ lại môi trường trình chứa
private EntityContext context;
//Phương thức định giá sản phẩm
public void setPrice(double price) {
  this.price = price;
}
//Phương thức trả về giá của sản phẩm
public double getPrice() {
  return price;
}
//Trả về thông tin mô tả của sản phẩm
public String getDescription() {
  return description;
}
/* Phương thức điều khiển bởi trình chứa */
//Tạo đối tượng Bean thực thể
public String ejbCreate(String productId,
                        String description,
                        double price)
                    throws CreateException {
  if (productId == null) {
    throw new CreateException("The productId
```

```
                                        is required.");
    }
    //Lưu lại dữ liệu của Bean
    this.productId = productId;
    this.description = description;
    this.price = price;
    return null;
  }
  //Phương thức này cho phép nhận và lưu lại đối tượng quản lý
  // môi trường ngoài của trình chứa
  public void setEntityContext(EntityContext context) {
    this.context = context;
  }
  //Phương thức kích hoạt thực thể Bean
  public void ejbActivate() {
    productId = (String)context.getPrimaryKey();
  }
  //Phương thức tạm ngưng hoạt động của Bean thực thể
  public void ejbPassivate() {
    productId = null;
    description = null;
  }
  //Các phương thức loại bỏ, lưu trữ và nạp lại dữ liệu của Bean
  public void ejbRemove() { }
  public void ejbLoad() { }
  public void ejbStore() { }
  public void unsetEntityContext() { }
```

```
  public void ejbPostCreate(String productId,
                            String description,
                            double balance) { }
} // ProductEJB
```

Khác với AccountEJB, thực thể Bean ProductEJB như bạn thấy không chứa bất kỳ lệnh kết nối hay lệnh SQL truy cập cơ sở dữ liệu nào cả (mã lệnh do đó gọn và đơn giản hơn). Thực tế trình chứa container sẽ tự thêm dữ liệu mới vào bảng dữ liệu (trong cơ sở dữ liệu) và gọi ejbCreate() của ProductEJB theo đúng thứ tự cột dữ liệu trong bảng productEJBtable. Bảng productEJBtable này sẽ được trình đóng gói yêu cầu deploytool tạo ra bằng lệnh SQL sau

```
CREATE TABLE "ProductEJBTable" (
    "description" VARCHAR(255) ,
    "price" DOUBLE PRECISION NOT NULL ,
    "productId" VARCHAR(255),
        CONSTRAINT "pk_ProductEJBTable" PRIMARY KEY
                                    ("productId")
)
```

Với Bean thực thể tự quản lý bạn phải dùng lệnh SQL tạo bảng thủ công. Trong khi với Bean thực thể quản lý bởi trình chứa bạn không cần phải tạo tay bảng dữ liệu. Trình đóng gói sẽ dựa vào khai báo của Bean để tạo ra các trường tương ứng. Ví dụ ProductEJB đã khai báo

```
  public String productId;
  public String description;
  public double price;
```

Tương ứng deploytool sẽ tạo ra bảng ProductEJBTable với các mô tả của bảng dữ liệu như sau

```
"productId" VARCHAR(255),
"description" VARCHAR(255) ,
"price" DOUBLE PRECISION NOT NULL ,
```

Dưới đây là đặc tả interface cho lớp Product làm giao tiếp giữa ProductEJB và trình chứa container.

Ví dụ 9-20 Product.java

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Product extends EJBObject {
  public void setPrice(double price)
          throws RemoteException;
  public double getPrice() throws RemoteException;
  public String getDescription() throws RemoteException;
}
```

Như bạn thấy, thành phần ProductEJB chỉ tập trung vào sử dụng phương thức getPrice(), setPrice() và getDescription() của Bean. Hoàn toàn không quan tâm đến việc tiếp cận cơ sở dữ liệu. Đây chính là tính tự nhiên và ưu điểm của thực thể Bean quản lý bởi trình chứa. Lớp giao tiếp chủ ProductHome dùng để tham chiếu và tạo ra giao tiếp Product được xây dựng như sau.

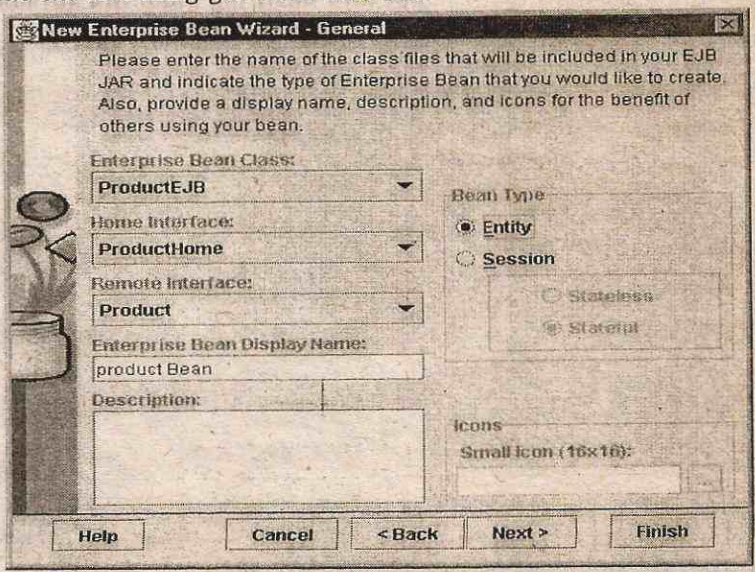
Ví dụ 9-21 ProductHome.java

```
import java.util.Collection;
import java.rmi.RemoteException;
import javax.ejb.*;
public interface ProductHome extends EJBHome {
  public Product create(String productId,
                        String description,
                        double balance)
        throws RemoteException, CreateException;
```

```
    public Product findByPrimaryKey(String productId)
            throws FinderException, RemoteException;
    public Collection findByDescription(
                                        String description)
            throws FinderException, RemoteException;
    public Collection findInRange(double low, double high)
            throws FinderException, RemoteException;
}
```

Lớp chủ ProductHome cung cấp các phương thức find_() nhưng bạn sẽ không tìm thấy cài đặt tương ứng ejbFind_() trong lớp Bean ProductEJB. Khi bạn đóng gói Bean trình deploytool sẽ tạo điều kiện để bạn cài đặt những phương thức này thông qua lệnh SQL SELECT. Bạn lưu mã nguồn của ProductEJB.java, Product.java, ProductHome.java vào thư mục C:\Network\ejb\product. Biên dịch mã nguồn thành tập tin .class.

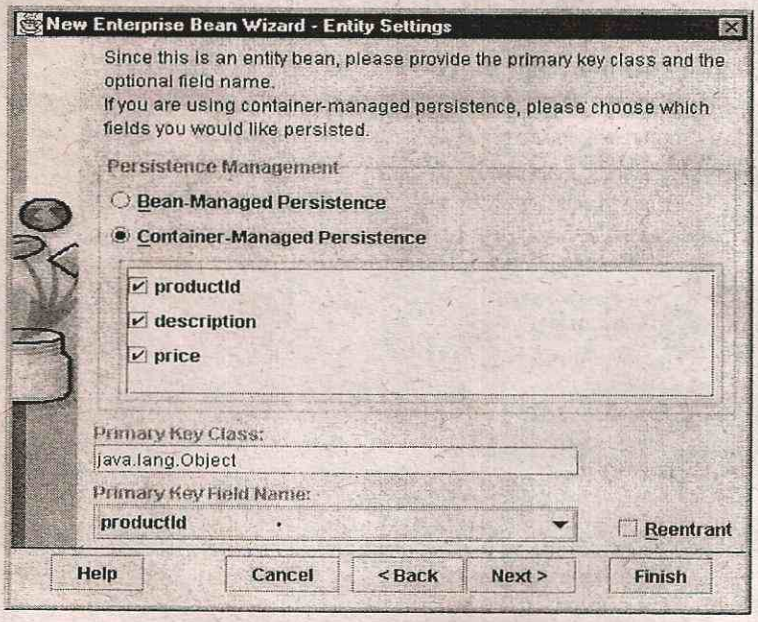
Đối với Bean thực thể quản lý bởi trình chứa bạn cần lưu ý một số thao tác khi đóng gói Bean như sau:



Hình 9-19 Cửa sổ chỉ định kiểu Bean thực thể

Ở bước chọn các lớp cần thiết để đóng gói bạn chọn productEJB từ danh sách *Enterprise Bean Class*. Chọn ProductHome từ danh sách *Home Interface*. Tiếp đến chọn Product từ danh sách *Remote Interface* (hình 9-19). (Bạn lưu ý để đơn giản chúng tôi không trình bày lại toàn bộ các bước đóng gói Bean, bạn tham khảo lại cách đóng gói chi tiết từng bước ở mục 3.2.2). Bạn chọn Entity Bean từ mục *Bean Type*.

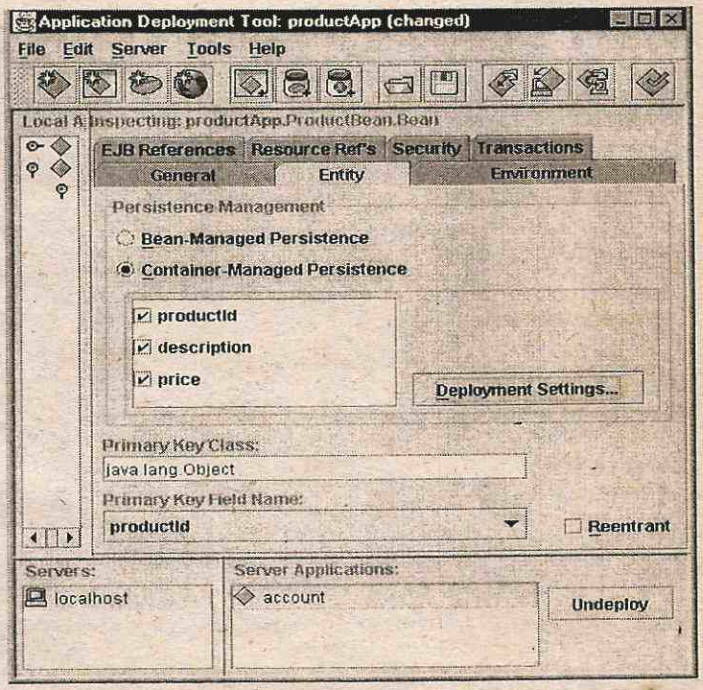
Nhấn nút *Next* để chuyển qua bước kế tiếp. ở bước này bạn chọn Container-Managed Persistence từ mục *Persistence Management* (hình 9-20). Khi bạn chọn mục này, trình đóng gói sẽ tự động tìm hiểu lớp ProductEJB mà bạn đã chỉ định trước đó. Trình đóng gói đủ thông minh để biết được rằng thành phần Bean của bạn yêu cầu tạo ra 3 trường productID, description và price. 3 trường này sẽ được liệt kê trong danh sách bên dưới. Bạn đánh dấu chọn, yêu cầu trình chứa tự tạo bảng dữ liệu và ràng buộc các trường dữ liệu vào thành phần Bean (hình 9-20).



Hình 9-20 Chọn các trường dữ liệu yêu cầu trình chứa quản lý

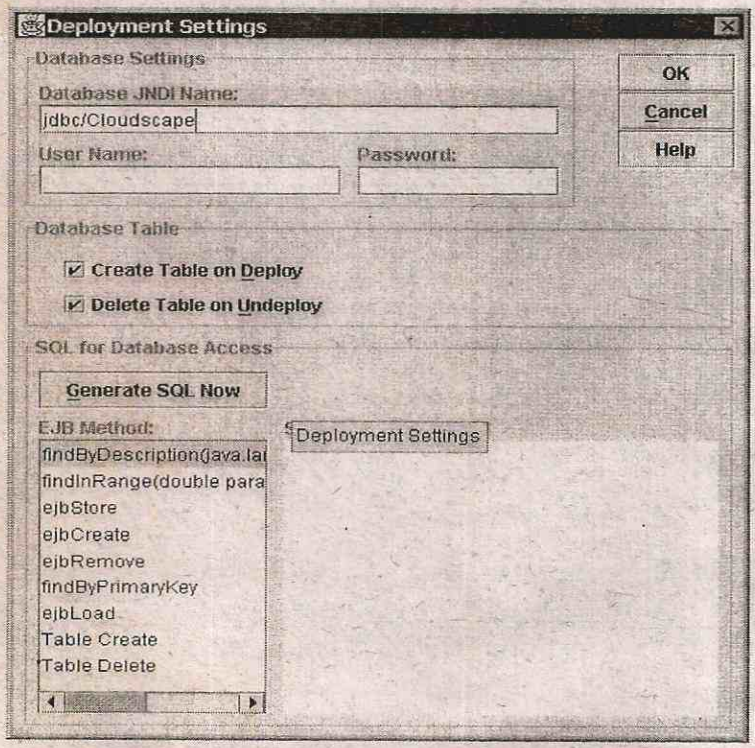
Ở danh sánh *Primary Key Field Name* bạn có thể chọn mục productID. Chúng ta muốn sử dụng productID có kiểu String làm khóa chính cho Bean (cũng là khóa chính trong bảng dữ liệu mà trình chứa sẽ tạo ra). Ở mục *Primary Key Class* bạn có thể thay thế java.lang.Object bằng lớp java.lang.String bởi, vì đó là kiểu đối tượng làm khóa chính mà ProductEJB sử dụng. Nếu bạn muốn khóa chính là một lớp phức hợp do bạn tự tạo thì bạn nên nhập vào tên lớp ở mục này.

Nhấn nút *Finish* để trở về cửa sổ chính của deploytool. Bạn chọn thành phần Bean bên cửa sổ nhánh trái, tương ứng chọn tab-Entity bên cửa sổ phải (hình 9-21). Kích chuột vào nút *Deployment Setting* Trình đóng gói sẽ hiển thị cửa sổ để bạn cấu hình hay thiết lập các lệnh SQL tương ứng cho các phương thức truy xuất cơ sở dữ liệu (hình 9-22)



Hình 9-21 Cửa sổ chính cấu hình Bean

Trước hết bạn gõ vào chuỗi kết nối với cơ sở dữ liệu jdbc/Cloudscape ở ô Database JNDI Name. Trong danh sách EJB Method bạn sẽ thấy xuất hiện các phương thức findByDesription(), findInRange(), ejbCreate(), ejbStore(), ejbLoad() ... những phương thức này trước đây đối với Bean tự quản lý (Bean-Managed) ta phải tự động thiết lập câu lệnh SQL (bạn xem lại ví dụ về đối tượng AccountEJB ở mục 7). Tuy nhiên với Bean quản lý bởi trình chứa, lệnh SQL sẽ được trình chứa tự động sinh ra và giúp bạn kiểm soát quá trình tương tác dữ liệu giữa thực thể Bean và cơ sở dữ liệu. Bạn hãy nhấn nút Generate SQL Now. Trình đóng gói sẽ sinh mã câu lệnh SQL tương ứng cho từng phương thức.



Hình 9-22 Cửa sổ chỉ định kết nối cơ sở dữ liệu và sinh mã SQL cho Bean

Tùy chọn *Create Table on Deploy* sẽ tự động tạo bảng dữ liệu để lưu thực thể Bean vào lúc bạn chuyển giao Bean cho trình chứa. Tùy chọn *Delete Table on Undeploy* sẽ xóa bảng dữ liệu nếu bạn rút Bean khỏi trình chứa. Cả hai mục này tương đương với mục Table Create và Table Delete trong danh sách *EJB Method*.

Sau khi yêu cầu *Generate SQL Now* phát sinh câu lệnh SQL cho từng phương thức bạn có thể tìm hiểu nội dung câu lệnh SQL bằng cách kích chuột vào tên phương thức trong danh sách *EJB Method*. Dưới đây là nội dung của từng câu lệnh kết hợp với phương thức

Tên phương thức	Lệnh SQL
`ejbCreate()`	*INSERT INTO "ProductEJBTable" ("description" , "price" , "productId") VALUES (? , ? , ?)*
`ejbStore()`	*UPDATE "ProductEJBTable" SET "description" = ? , "price" = ? WHERE "productId" = ?*
`ejbLoad()`	*SELECT "description" , "price" FROM "ProductEJBTable" WHERE "productId" = ?*
`ejbRemove()`	*DELETE FROM "ProductEJBTable" WHERE "productId" = ?*
`Table Create`	*CREATE TABLE "ProductEJBTable" ("description" VARCHAR(255) , "price" DOUBLE PRECISION NOT NULL, "productId" VARCHAR(255), CONSTRAINT "pk_ProductEJBTable" PRIMARY KEY ("productId"))*
`Table Delete`	*DROP TABLE "ProductEJBTable"*

Tên phương thức	Lệnh SQL
`findByPrimaryKey()`	*SELECT "productId" FROM "ProductEJBTable" WHERE "productId" = ?*
`findByDescription()`	*SELECT "productId" FROM "ProductEJBTable"*
`findInRange()`	*SELECT "productId" FROM "ProductEJBTable"*

Phương thức findByPrimaryKey() đã được deploytool cài đặt sẵn mệnh để WHERE để truy tìm thành phần Bean theo khóa chính. Tuy nhiên phương thức findByDescription() và findInRange() chỉ phát sinh lệnh SELECT. Bạn cần thêm vào mệnh đề WHERE cho hai phương thức này như sau

Đối với phương thức findByDescription() bạn thêm vào chuỗi

WHERE "description" = ?1

lệnh SQL đầy đủ giờ đây sẽ là

SELECT "productId" FROM "ProductEJBTable"
WHERE "description" = ?1

khi thực thi truy cập dữ liệu, chuỗi ?1 sẽ được thay thế bằng giá trị của đối số thứ nhất truyền cho hàm findByDescription().

Tương tự bạn thêm vào mệnh đề WHERE cho lệnh SQL của phương thức findInRange() như sau

WHERE "price" BETWEEN ?1 AND ?2

lệnh SQL đầy đủ của findInRange() sẽ là

SELECT "productId" FROM "ProductEJBTable"
WHERE "price" BETWEEN ?1 AND ?2

khi thực thi truy cập dữ liệu, chuỗi ?1 và ?2 sẽ được thay thế bằng giá trị của đối số thứ nhất và thứ hai truyền cho hàm findInRange().

Khi bạn chuyển ứng dụng EJB cho trình chứa lệnh CREATE TABLE sẽ được gọi để tạo ra bảng dữ liệu ProductEJBTable trước khi các lệnh SQL có thể thực thi.

Cuối cùng như bạn thấy trình đóng gói đã tạo sẵn những lệnh SQL thích hợp dành cho các phương thức ejbStore(), ejbLoad(), ejbRemove() ... (điều mà bạn phải tự làm khi muốn kiểm soát việc tương tác dữ liệu của Bean). Khi Bean hoạt động trình chứa container sẽ thay mặt Bean thực hiện các câu lệnh SQL trên.

Bạn chọn mục *Tool|Deploy Application* từ trình đơn để đóng gói và đưa ứng dụng Bean vào trình chứa. Đặt tên JNDI cho đối tượng Bean là "MyProduct". Khi đóng gói Bean bạn nên yêu cầu trình deploytool trả về tập tin ProductAppClient.jar chứa các lớp hỗ trợ cần dùng cho phía máy khách (chọn mục *Return Client Jar* trong cửa sổ hình 9-10).

Dưới đây là mã nguồn của trình khách triệu gọi đối tượng Bean ProductEJB

Ví dụ 9-22 ProductClient.java

```
import java.util.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
public class ProductClient {
  public static void main(String[] args) {
    try {
      //Khởi tạo môi trường
      Context initial = new InitialContext();
      //Lấy về tham chiếu của lớp chủ
      Object objref = initial.lookup("MyProduct");
```

```
//Chuyển kiểu an toàn
ProductHome home = (ProductHome)
        PortableRemoteObject.narrow(objref,
                ProductHome.class);
//Gọi Bean tạo ra sản phẩm mới
Product duke = home.create("123", "Ceramic Dog",
                                        10.00);
System.out.println(duke.getDescription()
                    + ": " + duke.getPrice());
duke.setPrice(14.00);
System.out.println(duke.getDescription()
                    + ": " + duke.getPrice());
duke = home.create("456", "Wooden Duck", 13.00);
duke = home.create("999", "Ivory Cat", 19.00);
duke = home.create("789", "Ivory Cat", 33.00);
duke = home.create("876", "Chrome Fish", 22.00);
//Truy tìm lại sản phẩm đã tạo
Product earl = home.findByPrimaryKey("876");
System.out.println(earl.getDescription()
                    + ": " + earl.getPrice());
//Truy tìm danh sách các sản phẩm có tên là Ivory Cat
Collection c = home.findByDescription("Ivory
                                        Cat");
Iterator i = c.iterator();
while (i.hasNext()) {
  Product product = (Product)i.next();
```

```
        String productId = (String)
                                product.getPrimaryKey();
        String description = product.getDescription();
        double price = product.getPrice();
        System.out.println(productId + ": "
                                + description + " " + price);
      }
          //Truy tìm các sản phẩm có giá từ 10.00 đến 20.00
      c = home.findInRange(10.00, 20.00);
      i = c.iterator();
      while (i.hasNext()) {
        Product product = (Product)i.next();
        String productId = (String)
                            product.getPrimaryKey();
        double price = product.getPrice();
        System.out.println(productId + ": " + price);
      }
    } catch (Exception ex) {
      System.err.println("Caught an exception." );
      ex.printStackTrace();
    }
  }
}
```

Bạn lưu tập tin nguồn trên đây vào cùng thư mục C:\Network\ejb\product. Thực hiện biên dịch trình khách thành tập tin .class.

Mọi việc đã sẵn sàng, bạn chạy trình khách từ dòng lệnh như sau:

- Chỉ định đường dẫn classpath

```
C:\network\ejb\product>set
        classpath=.;C:\jsdkee1.2.1\lib\j2ee.jar;
                    ProductAppClient.jar
```

- Chạy trình khách

```
C:\network\ejb\product>java  ProductClient
```

- Kết quả kết xuất

```
Ceramic Dog: 10.0
Ceramic Dog: 14.0
Chrome Fish: 22.0
999: Ivory Cat 19.0
789: Ivory Cat 33.0
123: 14.0
456: 13.0
999: 19.0
```

9. EJB VÀ CÁCH TRÌNH CHỨA QUẢN LÝ BEAN THÔNG QUA XML

Thật sự các thiết lập của bạn trong trình đóng gói deploytool được lưu vào tập tin .xml. Khi Bean được đóng thành tập tin .jar chuyển cho trình chứa, tập tin .xml lưu giữ những thông tin thiết lập cần thiết cho quá trình trình chứa container khởi động và điều khiển Bean. Ví dụ đối với Bean thực thể do trình chứa quản lý ProductEJB ở ví dụ trên chẳng hạn. Bạn chuyển vào thư mục C:\j2sdkee1.2.1\-repository\<computer_name>\applications (nơi J2EE dùng lưu các tập tin đóng gói Bean .jar của trình chứa), dùng

chương trình Winzip mở tập tin product903967584250Server.jar (bạn lưu ý con số kèm theo trong tên tập tin là do trình đóng gói tự sinh ra nên có thể thay đổi tùy theo thời điểm bạn đóng gói Bean). Bạn sẽ thấy tập tin cấu hình mang tên sun-j2ee-ri.xml. Nội dung tập tin này dùng chứa thông tin kiểm soát các phương thức của thực thể Bean ProductEJB như sau:

Ví dụ 9-23 sun-j2ee-ri.xml

```
<?xml version="1.0" encoding="Cp1252"?>
<j2ee-ri-specific-information>
 <enterprise-beans>
  <ejb>
   <ejb-name>Bean</ejb-name>
   <jndi-name>MyProduct</jndi-name>
  <ds-jndi-name> jdbc/Cloudscape </ds-jndi-name>
  <sql-statement>
    <method>
     <ejb-name>Bean</ejb-name>
     <method-intf>Home</method-intf>
     <method-name>
        findByDescription
     </method-name>
     <method-params>
      <method-param>java.lang.String</method-param>
     </method-params>
    </method>
    <sql>
       SELECT "productId" FROM "ProductEJBTable"
```

```
        WHERE "description"=?1</sql>
  </sql-statement>
 <sql-statement>
  <method>
   <ejb-name>Bean</ejb-name>
   <method-intf>Home</method-intf>
   <method-name>
      findInRange
   </method-name>
   <method-params>
     <method-param>double</method-param>
     <method-param>double</method-param>
   </method-params>
  </method>
  <sql>
     SELECT "productId" FROM "ProductEJBTable"
           WHERE "price" BETWEEN ?1 AND ?2
  </sql>
 </sql-statement>
 <sql-statement>
  <method>
   <ejb-name>Bean</ejb-name>
   <method-intf>Bean</method-intf>
   <method-name>
       ejbStore
   </method-name>
```

```
    <method-params />
   </method>
   <sql>
      UPDATE "ProductEJBTable" SET "description" = ? ,
             "price" = ? WHERE "productId" = ?
   </sql>
  </sql-statement>
  <sql-statement>
   <method>
    <ejb-name>Bean</ejb-name>
    <method-intf>Bean</method-intf>
    <method-name>
         ejbCreate
    </method-name>
    </method>
    <sql>
      INSERT INTO "ProductEJBTable" (
        "description" , "price" , "productId" )
        VALUES ( ? , ? , ? )
    </sql>
  </sql-statement>
  <sql-statement>
   <method>
    <ejb-name>Bean</ejb-name>
    <method-intf>Bean</method-intf>
    <method-name>
```

```
        ejbRemove
    </method-name>
    <method-params />
  </method>
  <sql>
      DELETE FROM "ProductEJBTable"
              WHERE "productId" = ?
  </sql>
</sql-statement>
<sql-statement>
  <method>
    <ejb-name>Bean</ejb-name>
    <method-intf>Home</method-intf>
    <method-name>
        findByPrimaryKey
    </method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </method>
  <sql>
      SELECT "productId" FROM "ProductEJBTable"
                WHERE "productId" = ?
  </sql>
</sql-statement>
<sql-statement>
```

```
<method>
 <ejb-name>Bean</ejb-name>
 <method-intf>Bean</method-intf>
 <method-name>
      ejbLoad
 </method-name>
 <method-params />
</method>
<sql>
    SELECT "description" , "price"
        FROM "ProductEJBTable" WHERE "productId" = ?
</sql>
</sql-statement>
<table-create-sql>
    CREATE TABLE "ProductEJBTable" (
            "description" VARCHAR(255) ,
            "price" DOUBLE PRECISION NOT NULL ,
            "productId" VARCHAR(255),
             CONSTRAINT "pk_ProductEJBTable"
              PRIMARY KEY ("productId") )
</table-create-sql>
<table-remove-sql>
  DROP TABLE "ProductEJBTable"
</table-remove-sql>
<create-table-deploy>true</create-table-deploy>
<delete-table-undeploy>true</delete-table-undeploy>
```

```
</cmp>
</ejb>
</enterprise-beans>
</j2ee-ri-specific-information>
```

Như bạn thấy, khi xây dựng Bean thực thể tự quản lý, bạn đặt lệnh truy vấn dữ liệu SQL ngay trong mã lệnh Java (cùng với mã lệnh tạo nên Bean). Trái lại, các Bean quản lý bởi trình chứa đặt lệnh truy xuất SQL bên ngoài Bean (trong tập tin cấu hình .xml kèm theo Bean). Tập tin .xml này còn rất nhiều mục bạn có thể thêm vào để điều khiển Bean bên ngoài mã Java. Theo đặc tả của Sun, mỗi thành phần Bean hoàn chỉnh (tập tin .jar) đều phải chứa tập tin cấu hình .xml lưu các thông tin giúp trình chứa container quản lý và kiểm soát được hoạt động của Bean khi đưa vào sử dụng.

Nếu bạn để trình chứa container quản lý các thao tác truy xuất dữ liệu thay cho Bean bạn sẽ có thêm một lợi điểm nữa là trình chứa container điều khiển luôn cả các giao tác (transaction) diễn ra khi truy cập hệ quản trị dữ liệu. Giao tác được xem là tác vụ rất quan trọng khi bạn xây dựng các ứng dụng liên quan đến cơ sở dữ liệu. Chúng bao gồm những thao tác như commit (xác nhận giao tác), rollback (hủy bỏ giao tác) cho những lệnh SQL liên quan đến thay đổi dữ liệu (như INSERT, hay UPDATE). Trình chứa container luôn bảo đảm rằng các lệnh SQL mà nó áp đặt lên Bean thực thể luôn thực hiện thành công giao tác truy xuất dữ liệu.

10. ENTERPRISE JAVA BEAN VÀ SỰ HỖ TRỢ CỦA CÁC TRÌNH CHỦ KHÁC

Các trình chủ hỗ trợ kiến trúc Enterprise Java Bean hiện có: WebLogic, Inprise Application Server (IAS), Oracle Application Server (OAS), JRun, IPlanet, WebSphere... J2EE chỉ là một phần cài đặt của đặc tả EJB. Mỗi trình chủ đều kèm theo trình đóng gói deploytool thích hợp để bạn biên dịch, đóng gói và đưa các lớp cần

thiết của Bean vào trình chứa. Các công cụ của những trình chủ khác nhau có thể cung cấp cách xây dựng và đóng gói Bean (tập tin .jar) vào những thư mục qui định khác nhau trong trình chủ (bạn nên tìm hiểu tài liệu của chúng). Một ứng dụng thương mại thật sự thường bạn không nên dùng j2ee (có sẵn trong bộ Java JDK Enterprise) mà nên tận dụng các tính năng cao cấp cũng như sự hỗ trợ từ phía các trình chủ như IAS hay OAS. Chúng là các trình chủ thương mại nên đáp ứng được cụ thể mục đích sử dụng và cách thiết kế chương trình cho khách hàng hơn. Ví dụ IAS (của Inprise) cung cấp cho bạn dịch vụ Web kết hợp với xây dựng các ứng dụng phân tầng Java, đóng gói các thành phần Bean, thiết kế Bean thực thể dựa vào cơ sở dữ liệu InterBase, cài đặt đối tượng CORBA dựa trên trình môi giới Visio-Broker. Hoặc OAS hoàn toàn dựa trên nền và phong cách xây dựng đối tượng của hệ cơ sở dữ liệu Oracle ...

Chúng tôi sẽ giới thiệu với bạn cụ thể hơn về cách xây dựng những ứng dụng EJB dựa trên các trình chủ đặc thù này trong giáo trình khác chuyên biệt hơn. Nếu bạn muốn tự nghiên cứu chúng tôi cũng đã chép sẵn trên đĩa CD trình chủ Weblogic, JRun và IAS cùng với tài liệu cần thiết để bạn tham khảo.

11. KẾT CHƯƠNG

Chương này đã giới thiệu với bạn khá đầy đủ về kiến trúc Enterprise Java Bean (EJB) mà Sun đưa ra từ phiên bản Java 1.2. Đây là mô hình phân tầng và lập trình đối tượng rất hiệu quả cho các ứng dụng mạng Java. Bạn xây dựng đối tượng (được gọi là các thành phần Bean) hoạt động bên trong trình chứa (container). Trình chứa đóng vai trò điều khiển và làm trung gian trong giao tiếp giữa Bean và trình khách. Tóm tắt có hai loại Bean: Bean thao tác (session Bean) và Bean thực thể (entity Bean). Bean thao tác có thể phi trạng thái (stateless) hoặc lưu vết trạng thái (stateful). Bean thực thể cũng được phân ra làm hai là Bean thực thể tự quản lý kết nối dữ liệu (Bean-managed) và Bean thực thể quản lý dữ liệu bởi trình chứa (contaner-managed Bean). Các bước xây dựng Bean cơ

bản gồm có: tạo ra hay cài đặt đối tượng Bean, xây dựng giao tiếp xa (remote interface) để trình chứa gọi đến Bean, xây dựng lớp giao tiếp chủ (Home interface) để trình khách có thể dùng để tham chiếu Bean từ xa đến lớp giao tiếp của trình chứa.

Bạn hãy tự xây dựng cho mình những thành phần Bean từ đơn giản đến phức tạp. Khi đó bạn sẽ hiểu rõ hơn về kiến trúc EJB của Java. Ngày nay, công nghệ lập trình đã chuyển hướng rất nhanh sang mô hình lập trình mạng đa tầng (multi-tier). Hiểu về Enterprise Java Bean sẽ giúp bạn dễ dàng tiến sâu hơn vào thế giới lập trình Java cũng như lập trình mạng sau này.

Cũng như lĩnh vực lập trình phân tán triệu gọi đối tượng từ xa, lập trình trong mô hình phân tầng EJB hiện rất mới mẻ và khá phức tạp đối với hầu hết lập trình viên (kể cả lập trình viên chuyên nghiệp). Bạn phải nắm vững và cố gắng tìm hiểu những khái niệm mới đang được các tổ chức quốc tế thế giới xây dựng, thay đổi hàng ngày. Một giáo trình chỉ đủ tóm tắt cũng như tạo bước khởi đầu cho bạn. Internet chính là nơi để bạn cập nhật kiến thức và những thông tin mới nhất về lập trình mà không giáo trình nào theo kịp.