Phát triển Hệ thống phân tán

Biên tập bởi:

Khoa CNTT ĐHSP KT Hưng Yên

Phát triển Hệ thống phân tán

Biên tập bởi:

Khoa CNTT ĐHSP KT Hưng Yên

Các tác giả:

Khoa CNTT ĐHSP KT Hưng Yên

Phiên bản trực tuyến:
http://voer.edu.vn/c/8fddcc3d

MỤC LỤC

1. Chương 1: Các mô hình ứng dụng
   1.1. Các mô hình ứng dụng
2. Chương 2: Lập trình phân tán với COM+/MTS
   2.1. Lập trình phân tán với COM+/MTS
3. Chương 3: Lập trình phân tán đối tượng với RMI
   3.1. RMI và lập trình phân tán đối tượng
   3.2. Cài đặt đối tượng phân tán
   3.3. Chuyển tham số trong các lời gọi từ xa
   3.4. Tìm hiểu RMIREGISTRY và các cách đăng kí đối tượng
   3.5. Dùng một đối tượng sản sinh nhiều đối tượng (FACTORY OBJECT)
   3.6. Kỹ thuật gọi đối tượng ở xa bằng phương thức động(DYNAMIC METHOD INVOKE)
   3.7. Tự động kích hoạt các đối tượng từ xa (ACTIVATION)
   3.8. Trình mồi nạp ứng dụng từ xa
   3.9. Ứng dụng RMI
4. Chương 4: Lập trình phân tán đối tượng với Corba
   4.1. Giới thiệu về Corba
   4.2. Ánh xạ giữa ngôn ngữ IDL và Java
   4.3. Corba và mô hình phân tán.
   4.4. Xây dựng Ứng dụng Corba khách/chủ trong môi trường java
   4.5. Tìm hiểu JNDI và dịch vụ quản lí trên CosNaming của Corba
   4.6. Lưu địa chỉ tham chiếu của đối tượng Corba ở dạng chuỗi
   4.7. Tham chiếu ngược (Callback) của đối tượng Corba
   4.8. Gọi các phương thức động (Dynamic Method Call) của đối tượng Corba
   4.9. RMI trên IIOP (RMI-OVER-IIOP)
   4.10. Corba và trình môi giới VISIBROKER
5. Chương 5: Lập trình phân tán REMOTING trong .NET
   5.1. Lập trình phân tán REMOTING trong .NET
6. Chương 6: Lập trình phân tán với WEB SERVICE
   6.1. Lập trình phân tán với WEB SERVICE
Tham gia đóng góp

Chương 1: Các mô hình ứng dụng

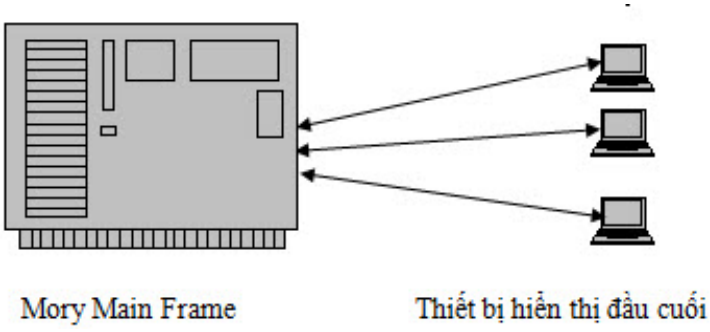
Các mô hình ứng dụng

Mô hình Client/Server, mô hình n lớp (n-tier),.. là những thuật ngữ mà chúng ta thường được nghe nói đến khi xây dựng một ứng dụng mạng. Mỗi mô hình có những đặc điểm khác nhau với những thuận lợi và khó khăn riêng. Là người phát triển ứng dụng thì bạn cần phải dựa vào đặc điểm của từng mô hình để đưa ra quyết định đỳng đắn trong việc chọn mô hình phù hợp cho ứng dụng. Ứng dụng sẽ hoạt động hiệu quả, dễ nâng cấp, bảo trì nếu được lựa chọn mô hình phù hợp, ngược lại, ứng dụng sẽ hoạt động kém hiệu quả và khó nâng cấp, bảo trì nếu mô hình ứng dụng không phù hợp. Do vậy việc lựa chọn mô hình phù hợp cho ứng dụng rất quan trọng.

Trong chương này chúng ta sẽ được giới thiệu khái quá về các mô hình ứng dụng. Các đặc điểm chính của từng mô hình, ưu nhược điểm của từng mô hình và cuối cùng là một vài lời khuyên khi lựa chọn mô hình phù hợp cho ứng dụng.

Mô hình tính toán tập trung

Thời kỳ đầu khi máy tính cá nhân còn rất đắt. khiến người ta nghĩ đến mô hình tính toán tập trung. Trong mô hình này có một máy tính lớn (main frame) và nhiều thiết bị đầu cuối (terminal). Tất cả các xử lý đều được thực hiện trên máy main frame. Các thiết bị đầu cuối chỉ có nhiệm vụ hiển thị giao diện người dựng và tương tác với người sử dụng, chúng không có bất cứ xử lý nào. Chính vì vậy năng lực tính toán của máy main frame nhanh chóng giảm xuống khi có nhiều người sử dụng đồng thời.



Mô hình Desktop

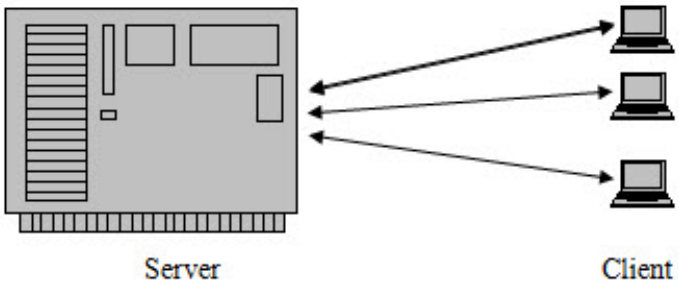
Khi giá máy tính cá nhân giảm xuống nhanh chúng thì mô hình tập trung không còn phù hợp nữa. Mỗi người sử dụng sở hữu một máy tính cỏ nhân. Lúc này mô hình của ứng dụng cũng thay đổi theo để phù hợp. Các ứng dụng được viết ra để chạy trên máy tính

cỏ nhân. Quá trình xử lý trong các ứng dụng kiểu này được thực hiện toàn bộ trên máy tính đó.

Mô hình 2 lớp (2 - tier)

Khi mạng máy tính phát triển, xuất hiện nhu cầu chia sẻ tài nguyên (như CPU, bộ nhớ, CSDL, tệp tin, máy in...). Lúc này xuất hiện mô hình Client/Server. Cả Client và Server đều tham gia vào quá trình xử lý. Mô hình client/server có ảnh hưởng lớn nhất đến ngành cụng nghệ phần mềm. Mô hình client/server giúp cho những máy tính riêng lẻ, khả năng xử lý thấp có thể chạy được những ứng dụng rất lớn, phức tạp. Mô hình client/server giúp phát triển một phương thức phát triển ứng dụng mới là việc phõn chia ứng dụng thành nhiều lớp (tier), mỗi lớp thực hiện những chức năng chuyên biệt.

Mô hình Client/Server được xem là tiêu biểu của mô hình 2lớp (two-tier). Trong mô hình này có nhiều Client và một Server. Giữa các Client và Server liên lạc với nhau qua hệ thống mạng. Ở mô hình này khi client yêu cầu một thông tin gì đó thì client sẽ gửi yêu cầu đến cho server, server xử lý các yêu cầu từ client rồi phản hồi những thông tin mà client cần.



Trong mô hình 2 lớp điển hình thì lớp Server thường là các Server CSDL. Server này có nhiệm vụ quản lý phần dữ liệu của chương trình như: Quản lý dữ liệu, bảo mật, quản lý người dựng, thực thi các truy vấn, ràng buộc và những thủ tục lưu trữ. Còn Client thì thường gồm các tác vụ sau: Hiển thị giao diện người dựng, tương tác với CSDL, kiểm soát lỗi...

M ô hình 2 lớp được chia làm 2 loại:

- *Tính toán tập trung ở máy khách (fat client – thin server):*

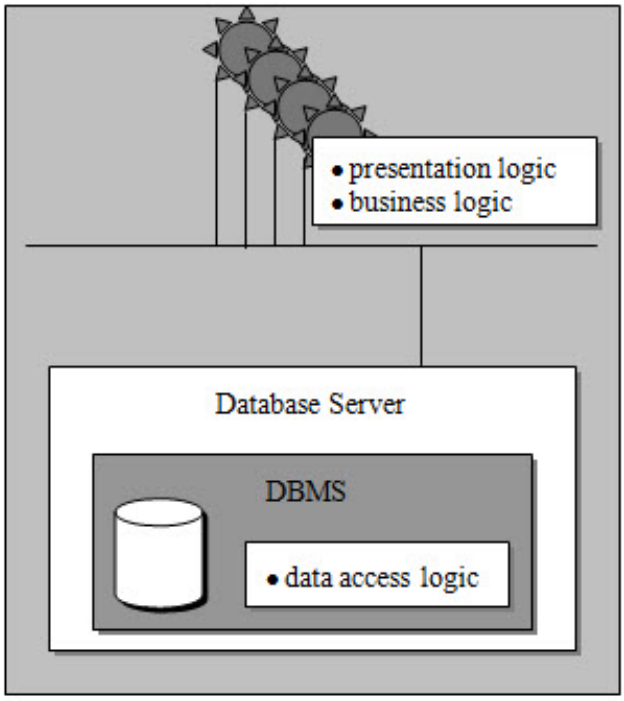
Ở mô hình này, phía client thường phải đảm nhận cả 2 lớp là giao diện và chức năng, còn server thường chỉ đảm nhận nhiệm vụ của lớp CSDL.

Ưu điểm của mô hình này là đơn giản, thích hợp cho những ứng dụng nhỏ. Tuy nhiên nó cũng có một số nhược điểm sau:

+ Do tính toán nghiêng về phía client nhiều nên đòi hỏi client phải có cấu hình đủ mạnh.

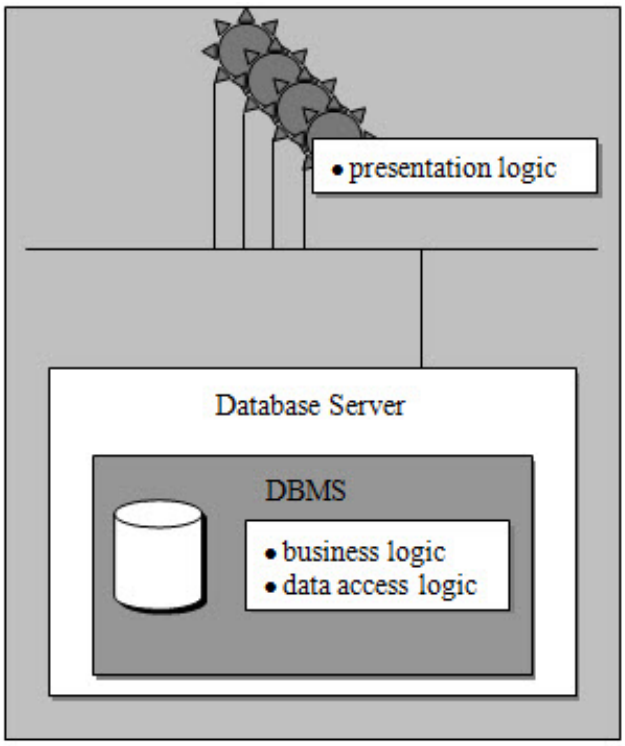
+ Do lớp chức năng của chương trình nằm ở phía client nên khi chương trình cần nâng cấp thì sẽ rất khó khăn vì phải cập nhật lại chương trình nằm ở toàn bộ các client.

+ Do mọi thao tác nằm trên client nên thao tác yêu cầu dữ liệu giữa client và server nhiều dẫn đến tốc độ chương trình chậm.



- Tính toán tập trung ở máy chủ (fat server – thin client):

Ở mô hình này, máy khách chỉ đảm nhiệm phần giao diện còn máy chủ thực hiện chức năng của tầng chức năng và tầng CSDL.



Ngược lại với loại fat client – thin server thì loại này có ưu điểm là giảm sự lưu thông trên mạng và tốc độ nhanh hơn do lớp chức năng của chương trình nằm ở máy chủ cùng với CSDL, do đó mọi thao tác với CSDL của chương trình đều được thực hiện ngay trên máy chủ và thông tin lưu thông trên mạng chỉ là những kết quả được trả về cho người dùng sau khi được tính toán. Mặt khác, do phần lõi của chương trình được đặt tập trung tại máy chủ nên việc cập nhật chương trình sẽ dễ dàng hơn. Tuy nhiên, do công việc được tập trung quá nhiều tại máy chủ nên đòi hỏi cấu hình máy chủ phải đủ mạnh, đặc biệt là khi có nhiều máy khách cùng truy xuất tới máy chủ cùng lúc.

Mô hình Client/Server là bước tiến lớn so với mô hình tính toán tập trung và mô hình Desktop. Các máy Client đó biết chia sẻ gánh nặng xử lý với Server. Điều này có nghĩa là khi có nhiều Client kết nối vào đồng thời cũng không làm năng lực của Server giảm xuống nhanh chúng. Với sự cải tiến của hệ thống mạng, các ứng dụng Client/Server ngày nay chạy tương đối nhanh. Người sử dụng ngồi từ nhiều nơi khác nhau có thể truy cập vào CSDL đặt trên máy Server với thời gian ít hơn. Tuy nhiên, vẫn còn tồn tại nhiều bất lợi trong mô hình này như:

- Thiếu tính mở rộng khi quy mô của ứng dụng lớn lên, chẳng hạn như số máy khách tăng lên. Lúc này nếu có nhiều máy khách kết nối vào đồng thời sẽ làm năng lực của Server giảm xuống nhanh chóng.
- Các kết nối đến máy chủ phải được duy trì. Mà mỗi Server thường chỉ giới hạn số kết nối tối đa. Cho nên không thể có quá nhiều kết nối được mở đồng thời.
- Khó khăn trong việc bảo mật ứng dụng.
- Mã lệnh khó sử dụng lại vì nó thường được tập trung tại phía Client.

- Khó bảo trì hệ thống khi cần thay đổi các quy tắc nghiệp vụ. Mỗi khi cần nâng cấp ứng dụng cần cập nhật lại trên tất cả các máy khách.

Do mô hình Client/Server có các nhược điểm nêu trên nên nó chỉ thích hợp với các ứng dụng có quy mô vừa và nhỏ.

Mô hình ba lớp (3 – tier)

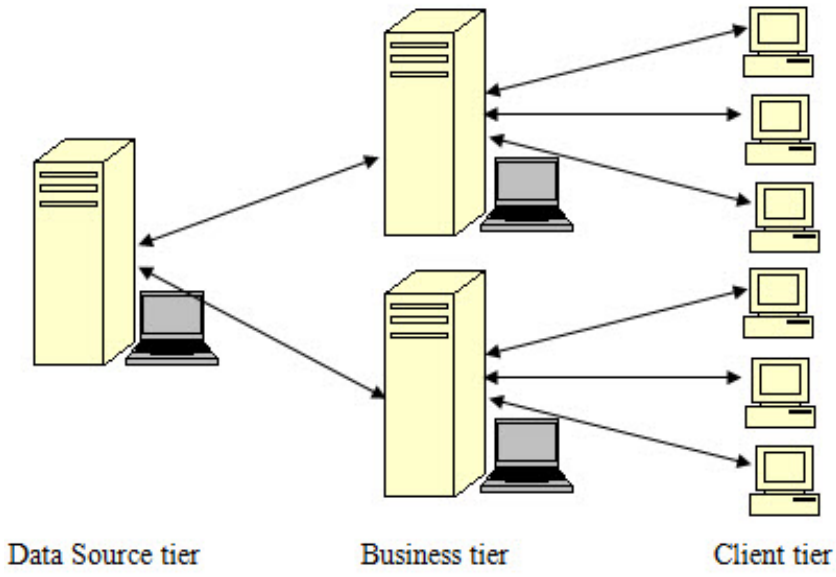
Mô hình 3 lớp ra đời nhằm giải quyết các yếu điểm của mô hình 2 lớp. Một ứng dụng thực tế thường được chia làm 3 lớp:

o Lớp giao diện (Presentation logic): lớp này là cầu nối giữa người dùng với ứng dụng, cung cấp những chức năng ứng dụng cho người dùng và nhận lệnh từ người dựng cho ứng dụng. Lớp này được thiết kế sao cho càng thân thiện với người dựng càng tốt.

o Lớp nghiệp vụ (Business logic): đây là phần lõi của một chương trình, cung cấp tất cả những chức năng nghiệp vụ của chương trình cho lớp giao diện bên trên. Trong mô hình 2 lớp thì các chức năng này thường được nằm ở phía Client nay được chuyển vào lớp nghiệp vụ.

o Lớp CSDL (Data Access logic): lớp này cung cấp khả năng truy xuất đến CSDL cho lớp nghiệp vụ nếu cần. Lớp dịch vụ dữ liệu đại diện cho một hay nhiều kho lưu trữ dữ liệu của chương trình.

Trong mô hình này các xử lý được sẩy ra tại cả ba tầng của ứng dụng. Mỗi lớp sẽ gồm một vài chức năng riêng biệt.



Trong mô hình 3 lớp, các chức năng của chương trình được tách ra thành 3 lớp riêng biệt. Việc tách lớp này làm cho các phần của chương trình độc lập hơn, đáng tin cậy hơn, chương trình trở nên linh động hơn trong việc thay thế, nâng cấp và do đó mô hình này rất thích hợp với những ứng dụng có yêu cầu thay đổi thường xuyên.

Mô hình đa lớp (n – tier)

Việc phân chia ứng dụng thành nhiều lớp còn giúp cho ứng dụng trở nên dễ dàng thay đổi, cập nhật và đáng tin cậy hơn. Khi quy mô của ứng dụng được mở rộng, mô hình n – tier không còn đáp ứng được nữa thì người ta nghĩ đến mô hình đa lớp (n – tier). Tuy nhiên ta không nên chia ứng dụng thành quá nhiều lớp dẫn đến khó xây dựng, bảo trì và nâng cấp ứng dụng.

Tại sao lại cần viết các ứng dụng nhiều lớp

Lợi ích của phương pháp lập trình hướng thành phần

Các ứng dụng thông thường thường được viết dưới dạng đơn thể (monolithic) có nghĩa là ứng dụng được biên dịch thành một file thực thi duy nhất. Các ứng dụng này thường không thể tùy biến hoặc không thể nâng cấp các chức năng bên trong nó một cách "động" được. Nó cần phải được sửa đổi chương trình nguồn và biên dịch lại.

Do đó, để chương trình đơn giản hơn trong việc sửa đổi cũng như nâng cấp, người ta thường chia chương trình chính ra thành nhiều thành phần nhỏ chạy tương đối độc lập nhau.

Lợi ích của việc phân chia ứng dụng thành các thành phần:

- Làm cho ứng dụng dễ tùy biến (customize) hơn.
- Có thể xây dựng ứng dụng nhanh hơn bằng cách sử dụng các thành phần có sẵn. Các thành phần có sẵn thường đã được biên dịch và được chứa trong các thư viện các thành phần như DLL, EXE hoặc các ActiveX….
- Dễ dàng chuyển đổi một ứng dụng bình thường thành một ứng dụng phân bố có thể sử dụng các thành phần được phân bố trên mạng.

Lúc này đòi hỏi các thành phần phải thoả mãn các ràng buộc:

- Các thành phần phải độc lập với ngôn ngữ mà nó được tạo ra. Một thành phần nếu được tạo bởi bởi ngôn ngữ VB thì có thể sử dụng được trong các ứng dụng viết bằng VC và ngược lại...
- Vì các thành phần phải dấu đi ngôn ngữ mà nó được tạo ra nên nó phải được sử dụng ở dạng nhị phân.

- Các thành phần có thể được nâng cấp mà không làm hỏng ứng dụng khác. Một thành phần mới có thể chạy tốt với cả ứng dụng cũ và mới.
- Các thành phần phải trong suốt về vị trí trên mạng. Một ứng dụng đối xử với các thành phần nằm trong cùng quá trình, khác quá trình hoặc nằm ở một máy khác đều như nhau.

Lợi ích của các ứng dụng phân tán

Một ứng dụng phân tán có tất cả các lợi ích của phương pháp lập trình hướng thành phần. Ngoài ra nó còn có các lợi ích sau:

- Ứng dụng có khả năng mở rộng cao, nếu tăng thêm người sử dụng chỉ cần tăng thêm số máy chủ ở tầng trung gian.
- Các component có tính sử dụng lại cao.
- Dễ nâng cấp và bảo trì ứng dụng.
- Dễ dàng quản lý bảo mật.
- ..

Lựa chọn mô hình cho ứng dụng

Có rất nhiều lựa chọn khác nhau cho một ứng dụng. Mỗi mô hình có một vài ưu nhược điểm khác nhau. Vì vậy việc lựa chọn một mô hình đóng đắn là một việc không dễ chút nào. Để lựa chọn đóng mô hình cho ứng dụng thì trước tiên ta cần trả lời các câu hỏi sau:

Cơ sở dữ liệu:

Hệ quản trị CSDL là gì? Việc lựa chọn một hệ CSDL phù hợp cũng góp phần không nhỏ vào sự thành công của ứng dụng. Việc lựa chọn hệ CSDL nào cho phù hợp với ứng dụng của bạn phụ thuộc vào rất nhiều yếu tố như:

- Bạn có đủ tiền để mua một hệ quản trị CSDL lớn như Oracle, SQL Server .. hay không?
- Dữ liệu của bạn có nhiều hay không?
- Dữ liệu có nhậy cảm hay không?
- Các giao dịch có thường xuyên sẩy ra hay không?....

Mô hình

Bạn cần lựa chọn mô hình phù hợp cho ứng dụng, việc lựa chọn một mô hình đóng phụ thuộc vào rất nhiều yếu tố khác nhau như:

- Số người dùng có tăng lên đột ngột trong tương lai hay không?

- Bạn có muốn dễ dàng trong việc bảo trì ứng dụng hay không?
- Cơ sở hạ tầng (hệ thống mạng, khả năng của các máy tính) có tốt hay không?
- Kiến thức nền tảng của đội ngũ lập trình viên?...

Xây dựng các quy tắc nghiệp vụ

Nếu ứng dụng của bạn gồm nhiều tầng thì bạn cần phải quyết định đưa các chức năng nào vào tầng nào. Việc quyết định nên đưa chức năng nào vào tầng nào là một việc tương đối khó và nó ảnh hưởng trực tiếp đến hiệu quả của ứng dụng.

Chương 2: Lập trình phân tán với COM+/MTS

Lập trình phân tán với COM+/MTS

Thường thì tầng trung gian này là các thành phần COM được viết bằng một trong các ngôn ngữ lập trình hỗ trợ giao thức COM (như Visual Basic, Visual C++,...) và được quản lý bởi Micrsoft Transaction Server (MTS). Khi tầng trung gian là các thành phần COM và được quản lý bởi MTS thì chúng được gọi là COM+/MTS.

COM+/MTS có nhiệm vụ quản lý tất cả các component, điều khiển, giám sát mọi tác vụ như: tạo mới một thành phần, quản lý các hàng đợi, giao dịch... COM+/MTS luôn lắng nghe các kết nối từ máy khách, khi các yêu cầu chấm dứt (không còn sử dụng đến nữa) thì nó giải phóng tài nguyên và kết nối đó.

Giới thiệu sơ lược về COM:

COM (Component Object Model) là một cách thức để viết các thành phần phần mềm (software componet) cung cấp các chức năng phục vụ cho các ứng dụng, các thành phần khác. Viết một thành phần COM cũng như viết một hàm API động, hướng đối tượng. Các thành phần COM có thể được nối kết với các ứng dụng cũng như các thành phần COM khác.

COM là gì ?

COM chỉ là một sự đặc tả. Nó chỉ ra làm thế nào để xây dựng các thành phần có thể thay thế một cách "động". COM cung cấp một "chuẩn" để các ứng dụng và các thành phần phải tuân theo để chúng có thể hoạt động được với nhau và sử dụng lại thư viện của nhau.

Thành phần COM là gì ?

Thành phần COM là những đoạn mã hiện thực các hàm, các chức năng được chứa trong file .DLL hoặc .EXE. Thành phần COM cần phải thỏa mãn được những "chuẩn" của COM. Các thành phần COM có thể được liên kết động. Các thành phần COM có thể được "bao gói" lại vì nó thỏa mãn các ràng buộc:

o Thành phần COM hoàn toàn độc lập với ngôn ngữ hiện thực nó. Thành phần COM có thể được hiện thực bằng nhiều ngôn ngữ như Visual C++, Visual Basic, Java …..

o Thành phần COM được sử dụng dưới dạng mã nhị phân.

o Thành phần COM có thể được nâng cấp mà không làm hỏng chương trình đang chạy.

o Thành phần COM trong suốt về vị trí.

Giao diện (Interface)

Giao diện là tất cả đối với COM. Các ứng dụng chỉ có thể nhìn thấy và truy xuất đến các thành phần COM thông qua giao diện của nó.

Giao diện làm tăng mức độ độc lập giữa ứng dụng và các thành phần COM. nhờ có giao diện mà một thành phần COM có thể được thay thế động mà không ảnh hưởng đến ứng dụng đang chạy.

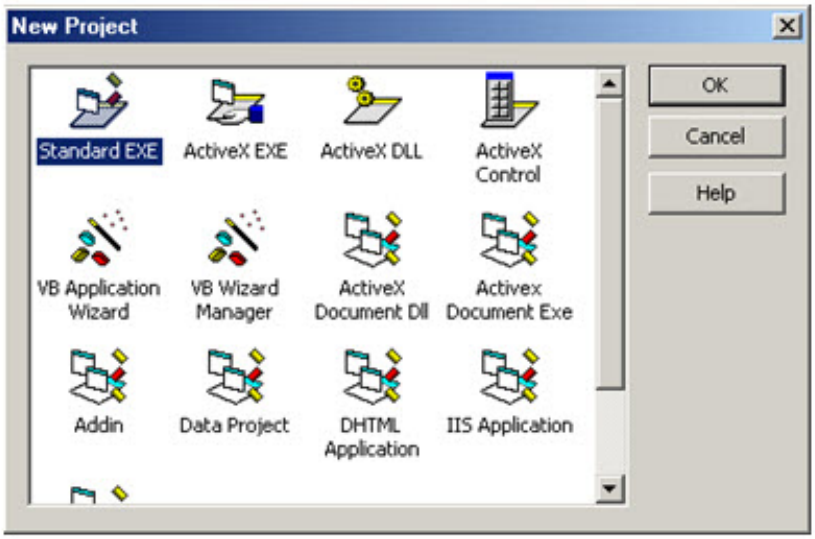
Giao diện không bao giờ thay đổi. Nếu muốn nâng cấp một giao diện thì giao diện mới phải tồn tại song song với giao diện cũ.

COM trong Windows:

Trong Windows, các thành phần COM cũng cũng như giao diện của COM đều được gán một ID. ID của thành phần COM được gọi là CLSID (class identifier); của giao diện COM là IID (interface identifier). Các ID này được gọi chung là GUID (globally unique identifier) là một cấu trúc 16 byte và được xác định duy nhất trên toàn thế giới. Tất cả các ID này đều được chứa trong registry của windows cùng với thông tin về các thành phần hoặc giao diện mà nó đại diện.

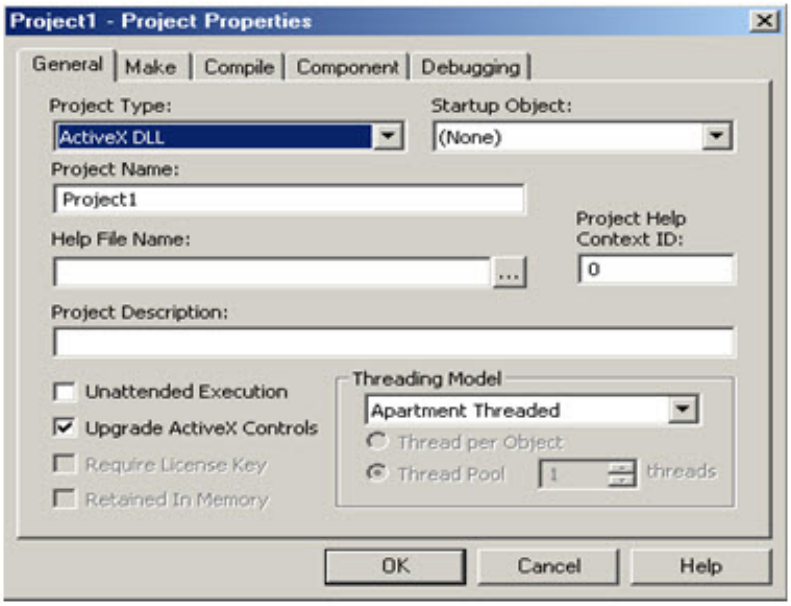
Khi ứng dụng muốn sử dụng COM, nó sẽ gọi một hàm API của windows là CoCreateInstance với thông số truyền vào là CLSID và IID tương ứng với thành phần và giao diện mà thành phần đó hỗ trợ để có được giao diện mong muốn.

Cách tạo COM(.dll) và COM(.exe) trong Visual Basic



Khi tạo một project mới trong visual basic, ta có thể chọn các dạng mẫu có sẵn để tạo một project mới. Để tạo thành phần COM ta có hai lựa chọn dạng COM(.dll) hoặc dạng COM(.exe) trong hộp thoại New Project tùy theo ứng dụng của chúng ta.

Khi tạo một dự án mới dạng ActiveX (.dll) hay ActiveX (.exe) Visual Basic sẽ tự động thiết lập các thuộc tính mặc định cho ự án đó. Ta có thể thay đổi những thuộc tính đó để chương trình thực hiện tốt hơn. Sau đây là các thuộc tính cơ bản của một ự án thành phần COM.



+ *StartupObject*: Startup Object cho phép ta chỉ định liệu code có được thực thi khi thành phần được load lên hay không. Đa ố thì ta thiết lập thuộc tính này là None, code được thực thi khi visual basic tạo một thể hiện của class.

+ *Project Name*: là tên của dự án mà ta đang tạo. Tên này cũng là ID của thành phần COM trong windows registry. Tên của dự án kết hợp với tên của class tạo nên ProgID. Client sẽ dùng tên này để tạo một đối tượng COM.

+ *Threading Model (ActiveX .dll)*: Trong dự án ActiveX (.dll), Thread Model cho phép ta lựa chọn Single Threaded hoặc Apartment Threaded. Giá trị mặt định là Apartment Threaded.

- Apartment Threaded về hình thức thực hiện chức năng multithreading giống như Windows 95, Windows 98, and

Windows NT. Đa phần các thành phần COM thực hiện với Visual Basic đều thực hiện với Apartment Threaded.

- Single Threaded: khi lựa chọn dạng này thì mọi client sẽ chạy trên một thread duy nhất, khi một client gởi một yêu cầu

tạo thread đến, mà thread đang phục vụ cho một client khác thì client vừa gởi yêu cầu phải đợi cho đến khi thread phục vụ

xong client kia mới đáp ứng lời gọi của mình. Với dạng này thì lợi điểm là dễ kiểm soát việc tạo thread, nhưng ngược lại khả năng phục vụ kém.

- Những lợi điểm của sự lựa chọn Apartment Threaded.

. Xử lý được vấn đề có nhiều yêu cầu tạo thread từ phía nhiều client. Tự động quản lý việc cung cấp thread phục vụ cho nhiều client, nhưng tại một thời điểm chỉ có một client được phục vụ.

+ *Threading Model (ActiveX .exe):* Trong dự án ActiveX Exe ta có những lựa chọn cho Threading Model như sau:

- Chỉ có một thread phục vụ: Chỉ cho tất cả client dùng chung một thread

- Chỉ định một con số cụ thể thread dùng chung, và số thread này được quản lý để phục vụ cho nhiều client theo phương pháp round-robin

- Mọi yêu cầu tạo phục vụ từ nhiều client sẽ được đáp ứng bằng cách tạo cho mỗi client một thread mới. Điều này gây khó khăn cho việc quản lý các đối tượng do client tạo ra.

Cách đăng ký, gỡ bỏ đăng ký COM(.dll) và COM(.exe) trong Windows

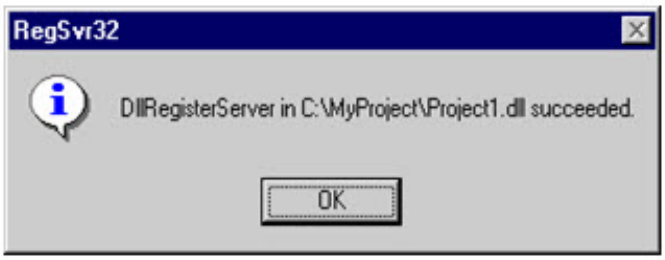
Sau khi biên dịch một dự án COM thành một file dạng dll hay dạng exe ta phải đăng ký các thành phần COM này để có thể sử dụng được có nhiều cách để đăng ký sau đây là cách tiêu biểu để đăng ký:

+ *Cách đăng ký COM DLL* : Để đăng ký được thành phần COM (.dll) ta thực hiện tuần tự các bước sau:

o Nhấn vào nút *start* của Windows

o Trên *Start* menu, click *Run*.

o Trong *Run*dialog box, ta gõ vào như sau:



Regsvr32.exe <DLLPath>\<DLL name>.dll

o Click vào nút OK. Nếu đăng ký thành công sẽ hiển thị dialog thông báo sau

Cách gỡ bỏ đăng ký của thành phần COM (.dll) cũng thực hiện các bước như trên, nhưng trong hộp thoại *Run*ta gõ vào như sau:

Regsvr32.exe /u <DLLPath>\<DLL name>.dll

+ *CáchđăngkýCOM(.exe)*:

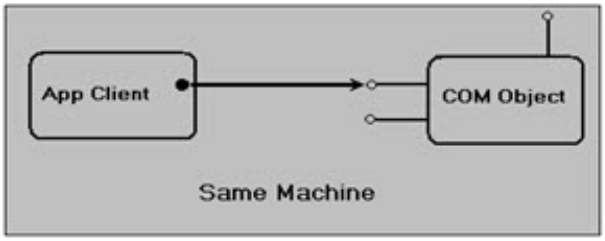
Cách đăng ký hay gỡ đăng ký của COM Exe cũng thực hiện các bước tương tự như việc đăng ký COM Dll. Nhưng trong hộp thoại *Run*ta gõ vào như sau

. <Đường dẫn của file exe đã biên dịch>\<tên file Exe> /regserver trong trường hợp muốn đăng ký.

. <Đường dẫn của file exe đã biên dịch>\<tên file Exe> /unregserver trong trường hợp muốn gỡ bỏ đăng ký.

Các kiến trúc của COM/DCOM:

Ứng dụng Client và ứng dụng COM chạy trên cùng một máy:

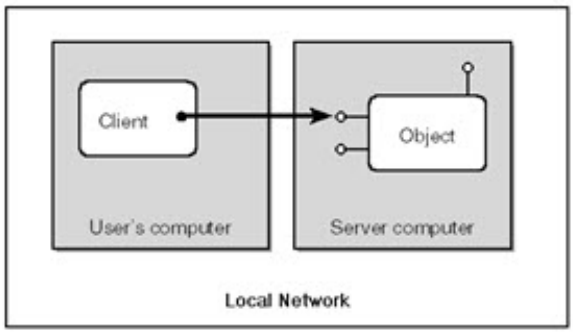


Mục đích xây dựng thành phần đối với mô hình này là :

. Ứng dụng COM có thể được tái sử dụng cho nhiều ứng dụng Client khác nhau.

. Làm cho chương trình dễ viết, dễ kiểm tra lỗi, nâng cấp mà không ảnh hưởng đến những ứng dụng Client đang chạy nó.

Ứng dụng Client chạy một máy, Ứng dụng COM chạy một máy. Nhưng hai máy này cùng một mạng local.



Lúc này thành phần COM được gọi là DCOM. Có 3 cách để giao tiếp giữa ứng dụng Client và Ứng dụng COM:

o ActiveX (.exe)

o Dùng MTS (Microsoft Transaction Server) để triển khai ActiveX DLL từ xa

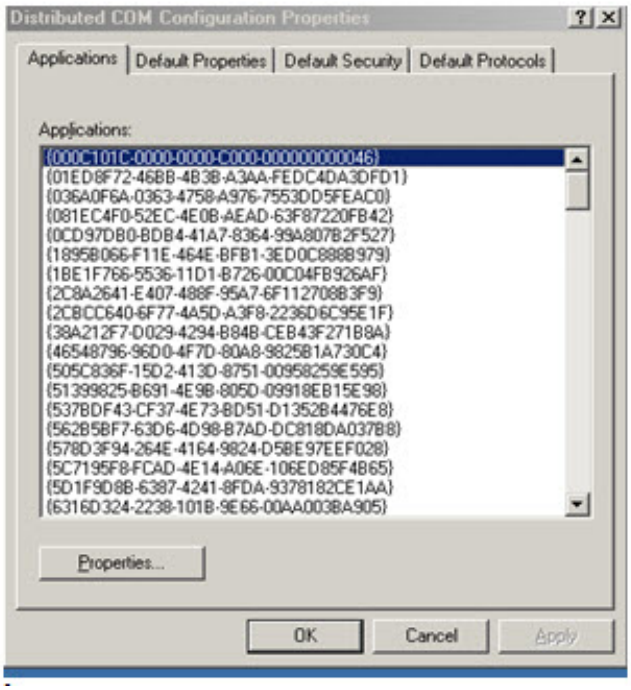
o RDS (Remote Data Access)

Giới thiệu về ActiveX (.exe)

Cũng là một ứng dụng COM nhưng được xây dựng cho phép chia sẻ ứng dụng cho nhiều ứng dụng Client ở các máy khác nhau(trong cùng một mạng) sử dụng. ActiveX (.Exe) chính là hình thức dạng DCOM.

Sau khi xây dựng và biên dịch một ứng dụng COM dạng ActiveX (.exe) để những ứng dụng Client có thể liên kết và sử dụng được ta phải chạy tập tin Dcomcnfg.exe của Windows để cấu hình cho ứng dụng COM này.

? Giới thiệu Dcomcnfg.exe để cấu hình ứng dụng COM (ActiveX Exe)



Trong giao diện chính của DCOM Config có 3 tabs chính sau đây:

? Applications.

? Default Properties.

? Default Security.

+ Applications Tab:

Applications Tab hiển thị những đối tượng đã đăng ký vào trong khóa registry: HKEY_CLASSES_ROOT\AppId\

Mỗi đối tượng được hiển thị trong Applications Tab, đặt tính của mỗi ứng dụng có thể được hiển thị bằng cách chọn đối tượng

và nhấn vào nút “Properties…” hay nhấn đúp vào đối tượng cần xem.

+ Default Properties Tab:

Những giá trị được hiển thị trong Default Properties Tab có thể được tìm thấy trong registry của Windows ở vị trí sau : HKEY_LOCAL_MACHINE\Software\Microsoft\ OLE

Tiêu đề đầu tiên trong Default Properties Tab là một check box : "Enable Distributed COM on this computer" khi check box này được đánh dấu sẽ cho phép tạo đối tượng DCOM. Ngược lại không tạo được đối tượng DCOM.

Lưu ý: tất cả những thiết lập có hiệu lực khi ta khởi động lại hệ thống.

Thành phần thứ 2 của Default Properties Tab là Default Distributed COM Communication Properties thành phần này được chia làm 2 mục sau:

- Default Authentication Level (Packet Level) Mục này có nhiều giá trị chọn như sau:

Name Description

None No authentication.

Connect Authentication occurs when a connection is made to the server. Connectionless protocols do not use this.

Call The authentication occurs when a RPC call is accepted by the server. Connectionless protocols do not use this.

Packet Authenticates the data on a per-packet basis. All data is authenticated.

Packet Integrity This authenticates that the data has come from the client, and checks that the data has not been modified.

Packet Privacy In addition to the checks made by the other authentication techniques, this encrypts the packet.

Default May vary depending upon operating system.

- Default Impersonation Level

Name Description

Anonymous The client is anonymous. This setting is not currently supported by DCOM.

Identify The server can impersonate the client to check permissions in the ACL (Access Control List) but cannot access system objects.

Impersonate The server can impersonate the client and access system objects on the client's behalf.

Delegate In addition to the Impersonate level, this level can impersonate the client on calls to other servers. This is not supported in the current release of DCOM.

+ Default Security Tab

Có 3 sự lựa chọn ở Default Security tab. Những giá trị của tab này được lưu trữ trong Windows Registry tại vị trí: HKEY_LOCAL_MACHINE\Software\Microsoft\OLE

Cách cấu hình Visual Basic DCOM dùng Dcomcnfg.exe

Việc cấu hình được thực hiện theo những bước sau:

o Run Dcomcnfg.exe

o Tại Applications Tab chọn ứng dụng DCOM cần cấu hình.

o Nhấn đúp vào ứng dụng DCOM vừa chọn.

o Chọn Security Tab

o Thực hiện những thay đổi sau.

- CUSTOM ACCESS PERMISSIONS: Everyone -allow access System -allow access Interactive -allow access

CUSTOM LAUNCH PERMISSIONS: Everyone -allow launch System -allow launch Interactive -allow launch

- Chọn Identity Tab, chọn "The Interactive User."

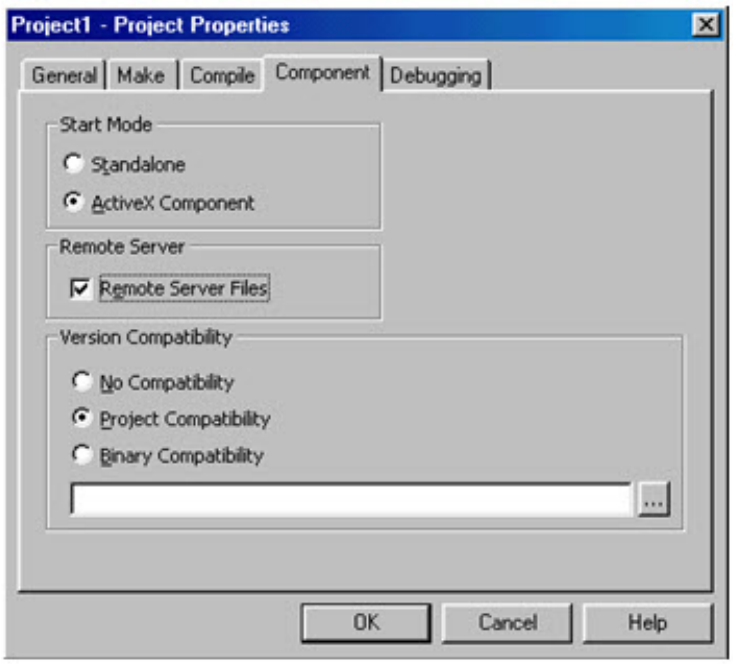
- Nhấn OK, nhấn OK một lần nữa.

? *Cáchhiệnthực một ứngdụngDCOM dạng ActiveX(.exe) dùng Visual Basic*.

? *TạomộtdựánDCOMdạngActiveX(.exe)*

1. Mở Visual Basic tạo một dự án dạng ActiveX EXE

2. Sau khi tạo xong dự án, vào menu Project\Project Properties thiết lập như sau:



- Việc ở Start Mode chọn ActiveX Component sẽ cho phép ta chạy ứng dụng server dưới môi trường phát triển Visual Basic. Nếu không chọn thì ứng dụng server sẽ thoát ra ngay khi ta chạy nó trong Visual Basic.

- Remote Sever đánh dấu vào Remote Server Files sẽ làm cho Visual Basic tạo ra VBR(Visual Basic Registry) và TLB (Type Library) ngay khi ta xây dựng xong ứng dụng server.

- Cuối cùng Version Compatibility chọn Project Compatibility, sau này ta sẽ đổi sang lựa chọn Project Compatibility.

3. Biên dịch ứng dụng server

? Trước khi phân bố ứng dụng, ta mở lại ứng dụng server, vào project properties của ứng dụng server thiết lập lại Version Compatibility chọn Binary Compatibility, tại text box dùng browser dẫn tới tập tin .exe của ứng dụng server. Save ứng dụng lại, biên dịch lại ứng dụng server.

? Triển khai ứng dụng server để ứng dụng Client có thể giao tiếp được.

Để triển khai ứng dụng phân bố vừa tạo ra ta phải sử dụng công cụ *"PackageandDeploymentWizard"* của Visual

Basic Tools bằng cách chọn Start\Program\Microsoft Visual Basic 6.0\Microsoft Visual Basic Tools. Tiếp theo thực hiện tuần tự theo các bước sau:

1. Dùng Browser chọn dự án server mới vừa tạo, nhấn vào nút Package

2. Chọn Dependency File trong Package Type.

3. Nhấn Next để chuyển qua bước tiếp theo.

4. Nhấn Next tiếp.

5. Chọn No khi được hỏi về hỗ trợ Remote Automation

6. Nhấn Next trên màn hình Included Files.

7. Nhấn Next trên màn hình Cab Information.

8. Nhấn Next trên Install Locations.

9. Nhấn Finish.

? *TạochươngtrìnhSetupchoClient.*

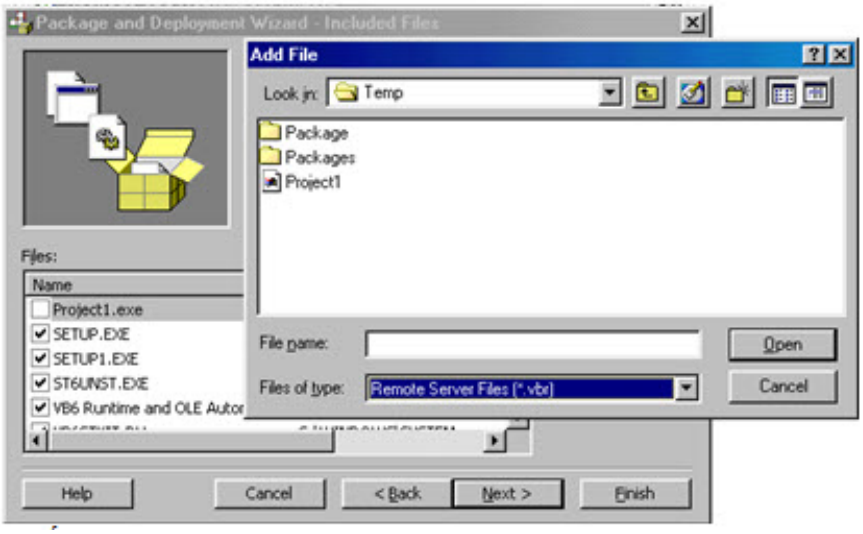
1. Khởi động chương trình Package and Deployment một lần nữa.

2. Nhấn vào nút Package

3. Chọn Standard Setup Package trong danh sách Package Type.

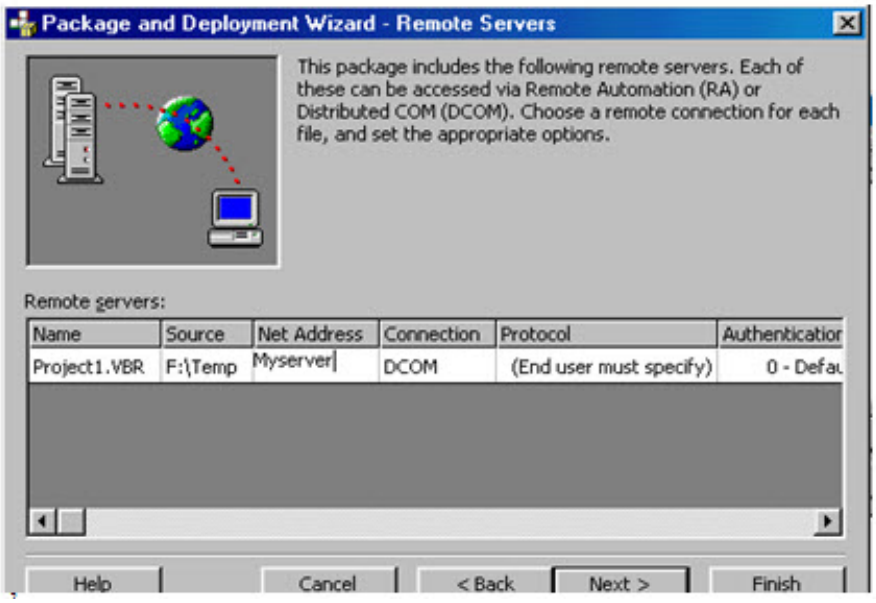
4. Chọn một folder mà nó sẽ được truy cập qua mạng.

5. Trong màn hình Included Files, Bỏ đánh dấu không chọn dự án server(.exe).



6. Nhấn vào nút Add, Chọn VBR ở combo box file type, chọn file .vbr

7. Nhấn Next.



8. Ở màn hình Remote Servers, gõ vào Net Address của Server.

9. Nhấn chọn Next đến hết. Nhấn Finish.

? Cài đặt trên máy chạy ứng dụng DCOM và trên máy chạy ứng dụng Client.

+ Chạy chương trình Setup.exe được tạo bởi Package and Deployment Wizard cho ứng dụng server trên máy chạy ứng dụng COM

+ Chạy chương trình Setup.exe exe được tạo bởi Package and Deployment Wizard cho ứng dụng Client trên máy chạy ứng dụng Client.

Dùng MTS để triển khai ActiveX DLL từ xa

Đối với Windows NT thì MTS được tách riêng ra thành một ứng dụng riêng biệt. Muốn sử dụng MTS thì phải cài đặt Option Pack 4 cho Windows NT. Còn đối với Windows 2000 thì MTS được tích hợp vào IIS 5.0, đưa COM và MTS tích hợp lại trong COM+ Applications.

? *Giới thiệu MTS (Microsoft Transaction Server)*

MTS là một dịch vụ trên Windows giống như các dịch vụ khác như IIS, File hay Print …, có nghĩa là ta có thể chạy (start) hoặc ngưng chạy (stop) chúng khi cần và MTS chạy background để phục vụ cho các ứng dụng cần đến nó

Trong Windows NT 4.0, MTS được coi như là một phần phụ (add-on), được cài thông qua NT4 Option Pack. Còn trong Windows 2000 nó trở thành một phần mặc định của chính hệ điều hành.

Ngoài chức năng chính là quản lý giao dịch, MTS còn quản lý nhiều thứ khác "MTS quản lý cách mà ứng dụng sử dụng các thành phần" hay nói cách khác là MTS quản lý các thành phần giúp cho các ứng dụng sử dụng các thành phần đó được hiệu quả hơn.

Các bước thông thường để sử dụng một thành phần:

1. Tạo một instance của thành phần.

2. Khởi động giá trị của các thuộc tính của thành phần.

3. Sử dụng thành phần.

4. Giải phóng thành phần ra khỏi bộ nhớ.

Thông thường, việc khởi tạo và giải phóng thành phần nằm ở đầu và ở cuối một thủ tục hay chương trình. Nếu sử dụng tốt hơn thì khởi tạo thành phần ngay khi cần đến và giải phóng sau khi chắc chắn không còn cần dùng đến thành phần đó nữa. Tuy nhiên trong khoảng thời gian sử dụng thành phần đó thường có rất nhiều thao tác không sử dụng thành phần hoặc không có thao tác nào sử dụng đến thành phần, nhưng vẫn phải giữ thành phần đó trong bộ nhớ chờ được sử dụng sau ở các thao tác khác mà không thể giải phóng thành phần vì khi giải phóng thì các trạng thái bên trong của thành phần cũng bị xóa đi. Điều đó tạo nên một sự lãng phí rất lớn tài nguyên của hệ thống.

MTS giải quyết vấn đề trên bằng cách: khi một ứng dụng cần một thành phần thì MTS sẽ cung cấp 1 instance của thành phần đó cho ứng dụng. Nếu một ứng dụng thứ 2 muốn sử dụng thành phần đó, trong khi ứng dụng 1 không thực sự sử dụng nó (tuy nhiên ứng dụng thứ 1 vẫn chưa giải phóng thành phần) thì MTS sẽ đưa instance mà ứng dụng thứ 1 đang giữ cho ứng dụng thứ 2 sử dụng. Nếu ứng dụng thứ 1 đột nhiên muốn sử dụng tiếp thành phần thì MTS sẽ tìm trong các instance của thành phần đó có instance nào không thực sự được sử dụng bởi ứng dụng khác không. Nếu có thì MTS sẽ đưa instance đó cho ứng dụng 1 sử dụng. Nếu tất cả các instance của thành phần đó đều đang được sử dụng thì MTS sẽ tạo một instance mới và giao nó cho ứng dụng thứ 1 sử dụng.

Quản lý giao dịch trong MTS: MTS cung cấp môi trường giao dịch cho các thành phần. Các thành phần trong cùng một môi trường sẽ thực hiện một giao dịch, có nghĩa là nếu toàn bộ các thành phần trong môi trường cùng thực thi thành công nhiệm vụ

của mình thì giao dịch coi như là thành công. Nếu một trong các thành phần vì lý do nào đó không thể thực thi nhiệm vụ của mình thì toàn bộ các công việc của các thành phần khác sẽ bị hủy bỏ và hệ thống sẽ trở lại trạng thái ban đầu, trạng thái trước

khi giao dịch được thực hiện.

? *Các bước để triển khai một thành phần từ xa trong Windows 2000 dùng MTS:*

? *Các bước thực hiện trên máy chạy thành phần COM DLL muốn triển khai từ xa*.

1. Tạo một thành phần COM dạng ActiveX DLL

2. Nhấn Start \ Settings \ Control Panel

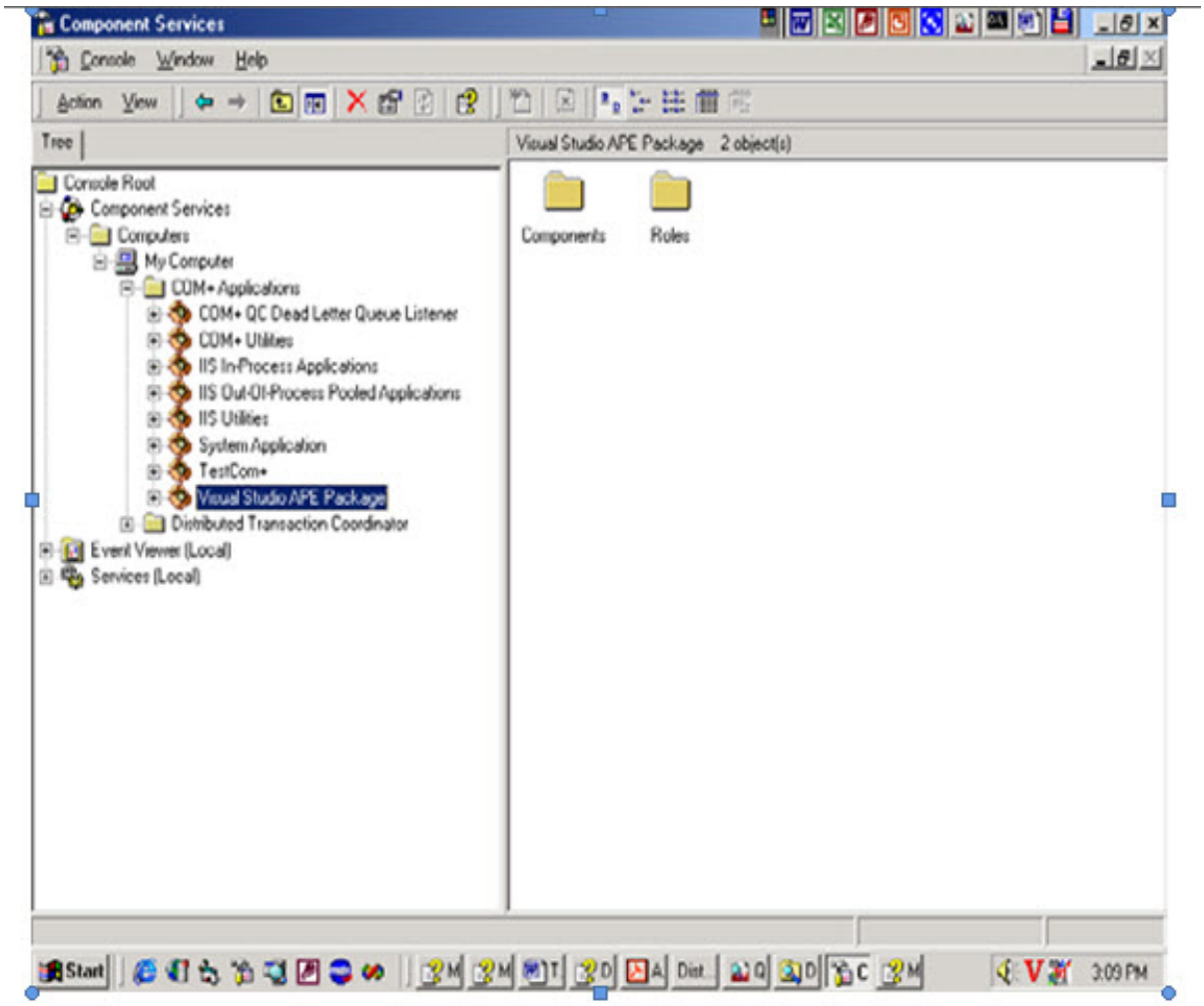
3. Trong cửa sổ Control Panel mở folder Administrative Tools

4. Mở Component Services

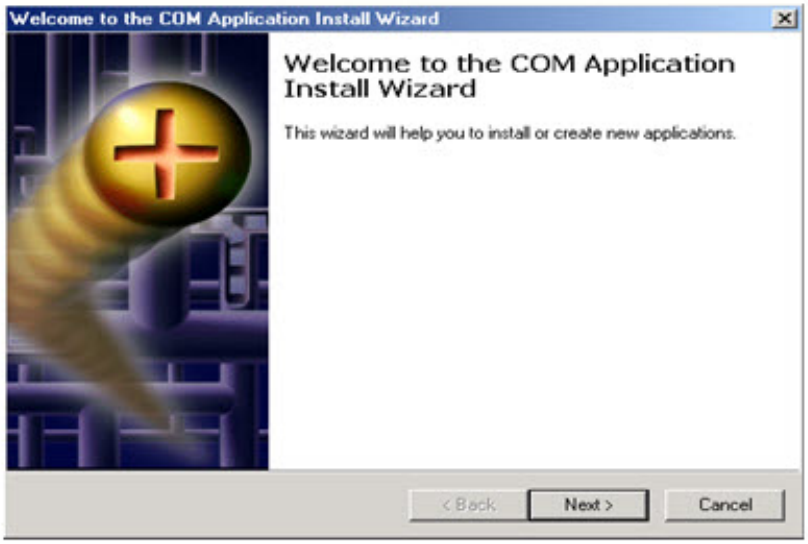
5. Trong console tree của Component Services, Nhấn đúp vào Computers

6. Nhấn đúp vào My Computers, nhấn đúp vào COM+ Applications

7. Nhấn phải chuột vào COM+ Applications, trỏ tới New rồi nhấn vào Application.



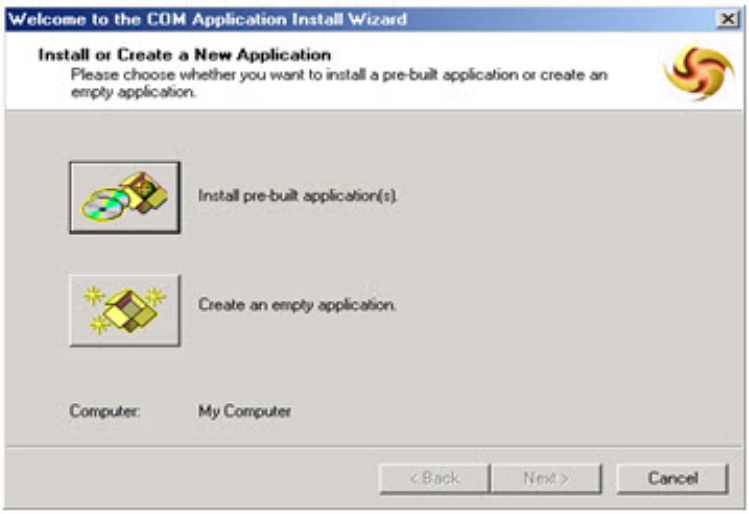
8. Xuất hiện hộp thoại COM Application Install Wizard, nhấn Next.

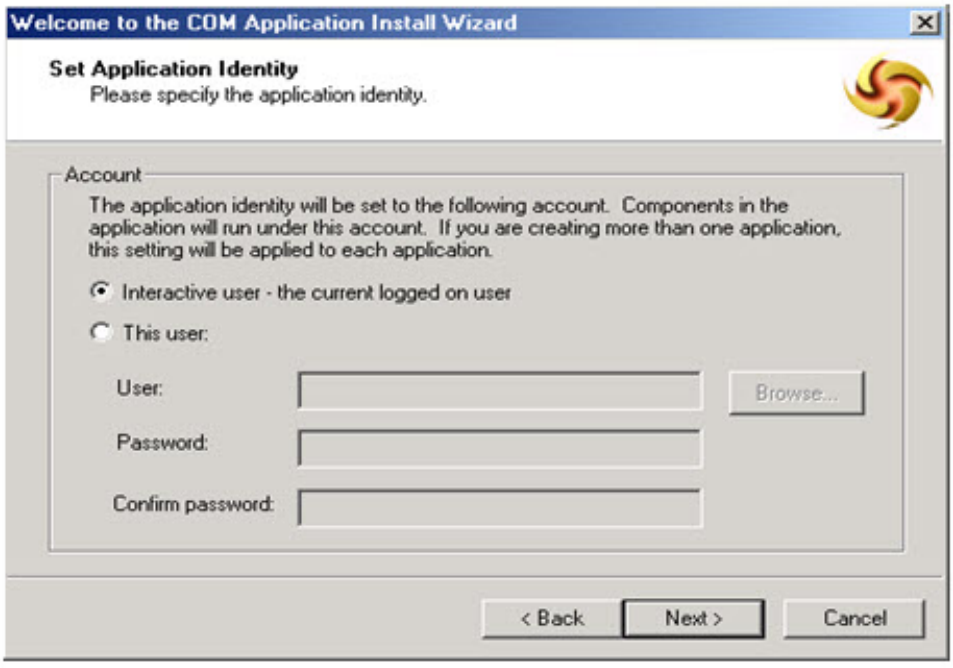


9. Tại đây sẽ có hai lựa chọn. Chọn *CreateanemtyApplication*

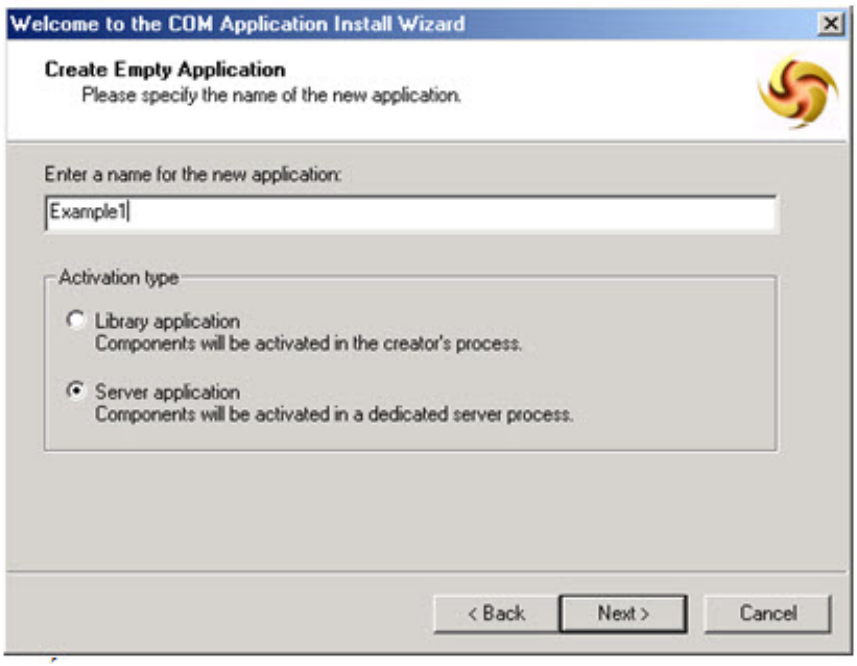
- *Install pre-built application(s)* Nếu muốn xây dựng lại ứng dụng mà ta đã tạo trước đó.

- *CreateanemtyApplication* : Nếu tạo một ứng dụng trống rồi import thành phần COM (.dll) vào.





10. Gõ vào vào một cái tên nào đó cho ứng dụng vừa tạo, nhấn vào Next

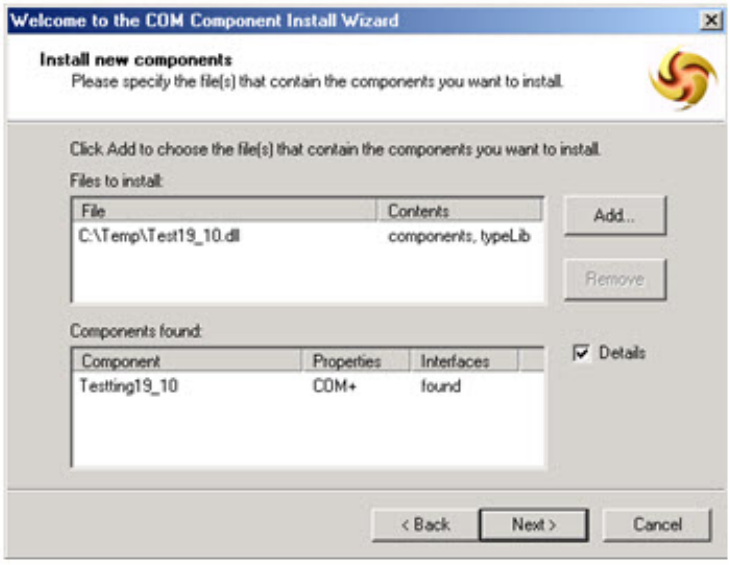


11. Tiếp theo xuất hiện hộp thoại có hai lựa chọn, ta lựa chọn giá trị default của nó. Nhấn tiếp vào Next.

12. Nhấn Finish.

13. Nhấn đúp vào ứng dụng Example1 vừa tạo. nhấn đúp vào Components

14. Nhấn phải chuột vào Components, trỏ tới New, nhấn vào Component.



15. Nhấn vào Next, Hộp thoại Import or Install a Component xuất hiện, chọn Install new Component(s).

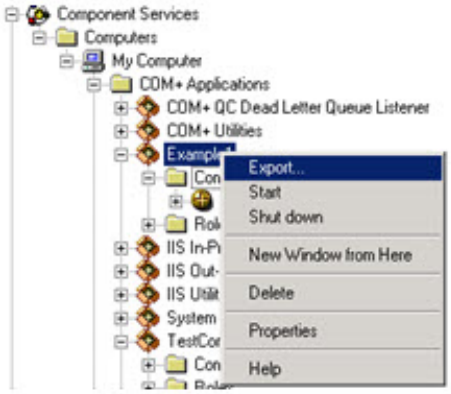
16. Dẫn đến chỗ lưu trữ file .dll, nhấn nút Open

17. Nhấn vào nút Next. Rồi nhấn vào Finish để kết thúc.

Để cho phép ứng dụng Client truy cập vào thành phần chạy dưới MTS, ta phải đăng ký vào Registry của máy Client để báo với nó rằng thành phần từ xa.

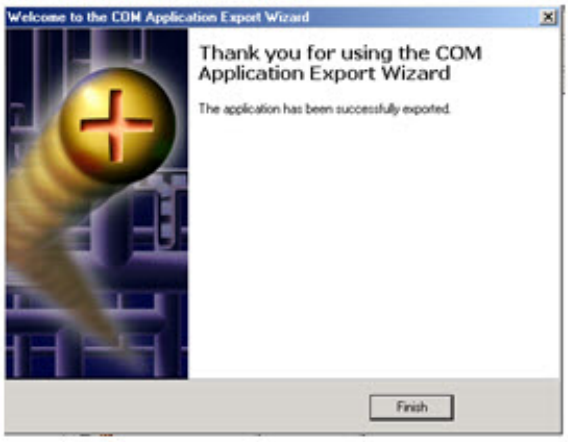
Để Client nhận ra thành phần MTS phân phát từ xa, ta làm như sau:

1. Nhấn phải chuột vào Example1 vừa tạo. Nhấn vào Export…



2. Dùng nút Browse để chọn một thư mục rỗng để export gói. Cho vào tên tập tin, nhấn Save.

3. Nhấn Next. Sau một lúc, hiển thị ra một hợp thoại thông báo sau:



4. Nhấn vào Finish để hoàn tất.

? *Các bước thực hiệntrênmáychạyứngdụng*Client.

1. Chép thư mục mà ta vừa Export được ở trên vào máy Client

2. Chạy tập tin .MSI trong thư mục này để đăng ký vào máy Client.

Muốn gỡ bỏ đăng ký này ra khỏi máy Client ta có thể vào Control Panel\Add/Remove Programs để gỡ bỏ ra.

c. Dùng RDS (Remote Data Access): Kỹ thuật này không nên sử dụng đối với mô hình này vì tốc độ chậm, kỹ thuật này được dùng cho trường hợp máy chạy ứng dụng và máy chạy ứng dụng COM không cùng mạng. Kỹ thuật này sẽ được giới thiệu sau.

Kếtluận:Tóm lại đối với mô hình kiến trúc máy chạy ứng dụng Client và máy chạy ứng dụng COM cùng một mạng local thì

ta chọn giải pháp "**DùngMTS(MicrosoftTransactionServer) đểtriểnkhaiActiveXDLLtừxa".**

Ưu điểm của giải pháp này sau:

o So với dạng ActiveX (.exe) thì dạng này quản lý các thể hiện thành phần và quản lý giao dịch tốt hơn nhiều nhờ

sự hỗ trợ của MTS.

o Về vấn đề quản lý security thì dạng này cũng hỗ trợ tốt hơn so với ActiveX (.exe).

o Còn so với RDS thì kỹ thuật này sẽ làm cho ứng dụng chạy nhanh hơn. Vì kỹ thuật RDS được dùng hỗ trợ cho việc truy cập từ xa đối với những máy không cùng mạng. Ngoài ra kỹ thuật này còn hỗ trợ cho những ứng dụng WEB.

Khuyết điểm:

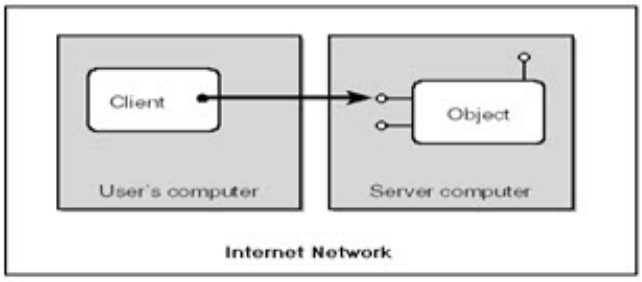
oMáy chạy ứng dụng COM của giải pháp này chỉ có thể chạy trên hệ điều hành Windows NT, hoặc Windows 2000.

Còn giải pháp dùng ActiveX(.exe) và giải pháp dùng RDS, máy chạy ứng dụng COM có thể chạy trên nền

Windows 98, Windows NT, Windows 2000.

oSo với ứng dụng dùng giải pháp dùng ActiveX(.exe) thì ứng dụng dùng giải pháp này sẽ chạy chậm hơn vì có kết hợp với MTS.

Máy chạy ứng dụng Client và máy chạy ứng dụng COM khác máy, 2 máy này không cùng mạng.



Để hiện thực việc giao tiếp giữa ứng dụng Client và ứng dụng COM đối với mô hình này thì phải dùng RDS(Remote

Data Access).

? *Giới thiệu về RDS*:

RDS là một mô hình lập trình cho phép ứng dụng Client có thể tạo một thể hiện của một ứng dụng COM đặt từ xa thông qua mạng Internet bằng các giao thức http, https. RDS hỗ trợ mạnh cho những ứng dụng Web, cho phép những ứng dụng Web viết những đoạn Script thao tác dữ liệu trên Server và lấy dữ liệu về thông qua thành phần RDSServer.DataFactory, RDS.DataControl.

lưu ý: để dùng RDS để truy cập DCOM dưới đây thì ta phải dùng MTS triển khai thành phần COM đó thì ta mới có thể dùng Set object = DataSpace . CreateObject(" ProgId ", " Myserver ") Còn nếu dùng trong dạng COM thì phải khai báo với hình thức In-process.

Protocol	Syntax
HTTP	**Set** *object* = *DataSpace*.**CreateObject("***ProgId***", "http://***awebsrvr***")**
HTTPS	**Set** *object* = *DataSpace*.**CreateObject("***ProgId***", "https://***awebsrvr***")**
DCOM	**Set** *object* = *DataSpace*.**CreateObject("***ProgId***", "***computername***")**
In-process	**Set** *object* = *DataSpace*.**CreateObject("***ProgId***", "")**

Vì RDS cho phép ứng dụng Client có thể thao tác dữ liệu thông qua mạng Internet nên vấn đề bảo mật đường truyền là vấn đề quan trọng để tránh việc thất lạc dữ liệu hay việc đánh cắp dữ liệu thì người lấy được những dữ liệu này cũng không thể xem

những dữ liệu này được. Vì vậy RDS có hỗ trợ việc tạo đối tượng qua mạng Internet dùng giao thức https. Giao thức này được sử dụng với công nghệ bảo mật đường truyền SSL(Secure Sockets Layer).

? *Giới thiệu SSL(Secure Socket Layer)*

Trên Web, SSL là giao thức thông dụng cho việc giao tiếp có mã hóa giữa browser và server. Sự kết hợp giữa một giải thuật mã/giải mã, một hàm giản lược thông điệp và một cách định danh được gọi là một bộ mật mã(cipher suite). SSL có thể hỗ trợ nhiều bộ mật mã.

Để sử dụng SSL, Web Server cần phải có bằng chứng nhận(Certificate) do một nhà cung cấp chứng nhận (CA – Certificate Authority) như Verisign cấp. Ngoài ra còn có thể sử dụng bằng chứng nhận do Certificate Server của Windows NT Server hay của Windows 2000 server cấp (cũng giống như của CA nhưng không có giá trị quốc tế).

+ *MộtquátrìnhSSL* :

Mục đích của một quá trình SSL là định danh server(hoặc có thể định danh client) và cuối cùng có được một khóa session cho việc giao tiếp giữa client và server trong một phiên làm việc (session).

+ Các bước của một quá trình SSL:

1. Client mở một cầu nối đến server và gởi thông điệp “ClientHello”. Thông điệp còn kèm theo một số thông tin như version của SSL, các bộ mật mã và giải thuật nén dữ liệu mà client hỗ trợ.

2. Server trả lời bằng thông điệp “Server Hello”. Thông điệp kèm theo thông tin về bộ mật mã và giải thuật nén mà server chọn cùng với một SessionID tương ứng của Client. Nếu các bộ mã được hỗ trợ bởi client và server không hợp nhau thì server sẽ gởi một thông báo “handshake failure” và chấm dứt session.

3. Server gởi “bằng chứng nhận” của nó để định danh nó đối với Client.

4. Server yêu cầu “bằng chứng nhận” của client. Điều này không bắt buộc và ít khi xảy ra.

5. Nếu client có “bằng chứng nhận” thì nó sẽ gởi cho server. Nếu không, nó sẽ gởi một thông báo không có bằng.

6. Client gởi thông điệp “trao đổi khóa” cùng với một bao điện tử chứa khóa session do client tạo ngẫu nhiên và được mã hóa bằng khóa public của server được lấy từ bằng chứng nhận của server.

7. Cả Client và Server cùng gởi thông báo “ChangeCipherSpec” để xác nhận cả hai đã sẵn sàng giao tiếp.

8. Cả hai cùng gởi thông báo “chấm dứt” phần thủ tục và bắt đầu giao tiếp có bảo mật bằng khóa session ở trên. Ngoài ra thông điệp cuối cùng này còn có chứa hash của toàn bộ quá trình làm thủ tục trên để cả hai cùng kiểm tra xem quá trình có bị trục trặc gì không.

? *CấuhìnhRDStrongWindows2000(MáychạyứngdụngCOM)*

Có hai sự cấu hình liên quan đến Remote Data Access: RDS security và IIS security.

+ *RDS security*:

Để ứng dụng Client có thể dùng RDS để giao tiếp với ứng dụng COM thì trên thư mục ảo(wwwroot đối với windows 2000) của máy chạy ứng dụng COM phải có thư mục MSADC, trong đó có tập tin *msadcs.dll*.

Ngoài ra cần phải khai báo bằng tay trong registry của máy chạy ứng dụng COM (mở ra bằng cách chạy tập tin regedit.exe) ProgId của ứng dụng COM (tên dự án.tên những class) tại vị trí sau:

\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\
Parameters\ADCLaunch\

+ *IIS security*: Để thiết lập IIS security thực hiện lần lược các bước sau:

1. Mở Control Panel, mở *Administrative Tools*. mở *InternetServicesManager*.

2. Nhấn đúp lên *Default Web Site*.

3. Nhấn phải chuột trên thư mục ảo MSADC, nhấn Properties.

4. Trên *Directory Security*tab, chỗ *IPAddressandDomainNameRestrictions*, nhấn Edit.

- Nếu muốn tất cả các Client có thể giao tiếp với ứng dụng chọn *GrantedAccess*.

- Nếu muốn chỉ cho phép một số Client nào đó có thể truy cập đến ứng dụng, chọn Client rồi Add vào danh sách.

? Ngoài ra ta có thể dùng RDS để truy cập đến những ứng dụng COM được khai triển từ xa dùng MTS thông qua Internet. Dùng RDS để truy cập những ứng dụng DCOM dạng ActiveX EXE.

Chương 3: Lập trình phân tán đối tượng với RMI

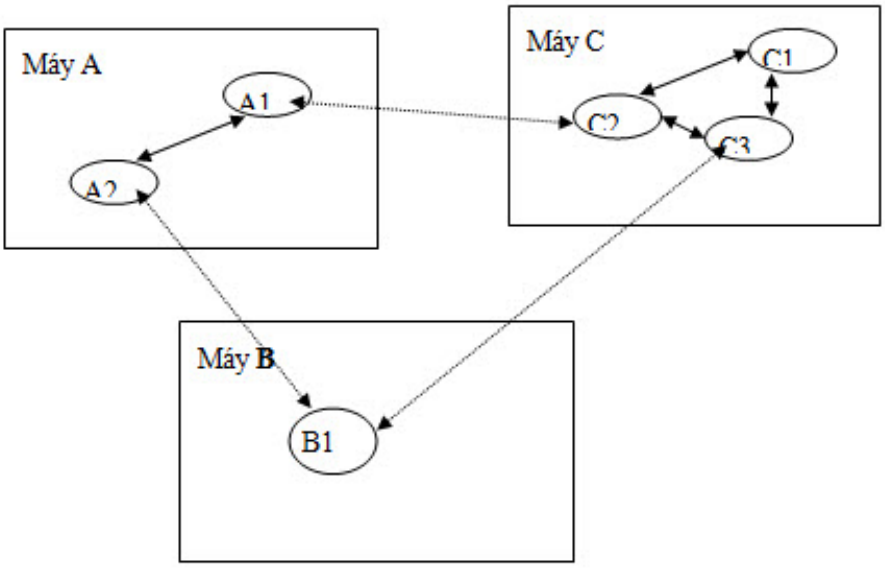
RMI và lập trình phân tán đối tượng

Thông thường mã lệnh của một chương trình khi thực thi được tập trung trên cùng một máy, đây là cách lập trình truyền thống. Sự phát triển như vũ bão của mạng máy tính đặc biệt là mạng Internet toàn cầu, đã khiến các chương trình truyền thống này không còn đáp ứng được yêu cầu nữa. Các chương trình bây giờ yêu cầu phải có sự hợp tác xử lý, tức là mã lệnh của nó đã không tập trung trên một máy mà được phân tán trên nhiều máy. Khi một ứng dụng có mã lệnh thực thi được phân tán trên nhiều máy thì chương trình đó được gọi là chương trình phân tán và việc lập trình để tạo ra các chương trình này được gọi là lập trình phân tán. Có rất nhiều công nghệ lập trình phân tán như: DCOM, CORBA, RMI, EJB.. trong đó RMI là công nghệ thuần Java và dễ lập trình nhất.

Thông thường nếu các đối tượng đượ tập trung trên cùng một máy thì bạn có thể triệu gọi các phương thức của nó bằng cách gửu cho nó một thông báo, câu hỏi đặt ra là làm thế nào để có thể triệu gọi các phương thức của một đối tượng nằm trên một máy khác. đây chính là nội dung của lập trình phân tán mã lệnh RMI (Remote Method Invoke – tạm dịch là triệu gọi phương thức từ xa). RMI là cách thức giao tiếp giữa các đối tượng Java có mã lệnh cài đặt (bao gồm cả phương thức và thuộc tính) nằm trên các máy khác nhau có thể triệu gọi lẫn nhau.

Hình sau mô hình triệu gọi đối tượng phân tán. Trên máy A các đối tượng A1, A2 gọi các phương thức của nhau được gọi là triệu gọi phương thức cục bộ (local method invoke) đây là cách lập trình hướng đối tượng truyền thống vẫn sử dụng, tương tự các đối tượng B1, B2, B3 là các đối tượng cục bộ.

Tuy nhiên các đối tượng Java có thể triệu gọi phương thức của một đối tượng nằm trên một máy khác dựa vào giao thức triệu gọi từ xa RMI. Trong mô hình dưới đây thì lời triệu gọi phương thức của đối tượng B2 (nằm trên máy B) từ đối tượng A1 (nằm trên máy A) là lời gọi phương thức từ xa.



Gọi phương thức từ xa và các vấn đề phát sinh

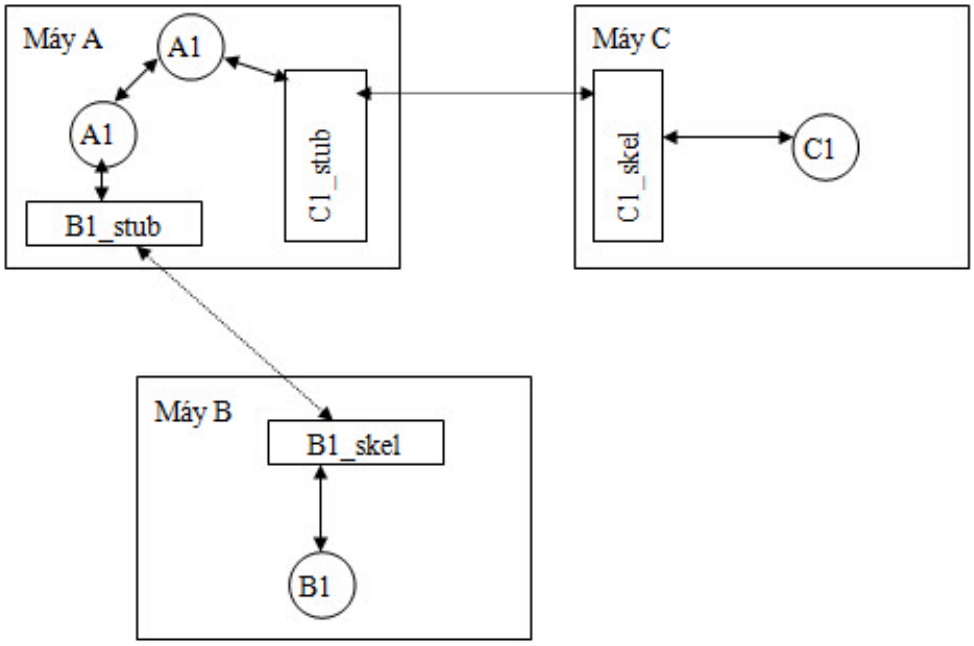
Việc triệu gọi một phương thức từ xa thoạt nhìn có vẻ đơn giản nhưng thực tế lại phức tạp hơn triệu gọi phương thức cục bộ. Các đối tượng trên hai máy khác nhau hoạt động trên hai tiến trình khác nhau (có hai không gian địa chỉ khác nhau) nên việc tham chiếu biến địa chỉ đối tượng là khác nhau. Ví dụ khi bạn truyền một đối tượng cho một phương thức triệu gọi từ xa thì thực sự bạn truyền một tham chiếu đối tượng đến phương thức từ xa, tuy nhiên vùng nhớ thực sự của đối tượng lại nằm trên một máy khác. Lời gọi phương thức cục bộ luôn trả về kết quả thông qua ngăn xếp trong khi lời gọi phương thức từ xa kết quả trả về phải thông qua kết nối mạng chính vì vậy các sự cố về truyền thông luôn có thể sẩy ra, như vậy việc bắt và kiểm soát lỗi trong các ứng dụng phân tán là rất đắt.

Vai trò của các lớp trung gian

Đối với lời gọi phương thức cục bộ, các tham số truyền vào phương thức cũng như kết quả trả về từ phương thức được thực hiện thông qua ngăn xếp (stack) trong khi lời gọi phương thức từ xa phải được đóng gói và chuyển qua mạng.

Để đơn giản và trong suốt đối với người lập trình đối tượng Java trên hai máy khác nhau không triệu gọi phương thức của nhau một cách trực tiếp mà thông qua lớp trung gian. Lớp trung gian tồn tại ở cả hai phía: phía máy khách (nơi gọi phương thức của đối tượng ở xa) và máy chủ (nơi đối tượng thực sự được cài đặt). Phía máy khác lớp trung gian này được gọi là stub (lớp móc), phía máy chủ lớp trung gian được gọi là skeletion (lớp nối).

Ta có thể hình dung lớp trung gian stub và skel là hai người trung gian giúp các đối tượng ở xa có thể giao dịch được với nhau.



Trong hình trên bạn có đối tượng C1 được cài đặt trên máy C. trình biên dịch Java giúp ta tạo ra hai lớp trung gian C1_stub và C1_skel. Lớp C1_stub được mang về máy A. Khi A1 trên máy A triệu gọi phương thức của đối tượng C1 nó sẽ chuyển lời gọi phương thức cho lớp trung gian C1_stub. Lớp trung gian C1_stub có trách nhiệm đóng gói các tham số chuyển tham số qua mạng đến phương thức được triệu gọi của đối tượng C1. Trên máy C lớp trung gian C1_Skel có nhiệm vụ nhận các tham số và chuyển vào vùng địa chỉ thích hợp sau đó gọi phương thức tương ứng. Kết quả trả về (nếu có) của phương thức do C1 trả về sẽ được lớp C1_Skel đóng gói và chuyển ngược về cho trình khách. Trên máy khách lớp trung gian C1_Stub chuyển giao kết quả cuối cùng lại cho A1. Bằng cơ chế này A1 luôn nghĩ rằng đối tượng C1 đang tồn tại ngay trên cùng máy với nó như các đối tượng cục bộ khác. Hơn nữa nhờ có lớp trung gian C1_Stub mà khi kết nối mạng gặp sự cố thì 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 đối tượng C1 được cài đặt trên máy A? C1_Stub trên máy A chỉ thực hiện việc chuyển tham số, nhận kết quả trả về nếu có và thực hiện các giao thức mạng, nó không phải là hình ảnh của C1. Để làm được điều này đối tượng C1 cần cung cấp một giao diện tương ứng với các phương thức mà các đối tượng trên máy khác có thể triệu gọi. Chúng ta tiếp tục làm sáng tỏ điều này khi nói về giao diện từ xa.

Cài đặt đối tượng phân tán

Giao diện từ xa

Khi bạn muốn tạo ra một sự vật từ xa thì bạn che mạt nạ cài đặt nền bằng cách truyền qua một giao diện. Vậy khi khách thu được một tham chiếu đến đối tượng từ xa thì thực chất đó là một giao diện.

Khi tạo ra một giao diện từ xa, thì bạn phải tuân theo các hướng dẫn sau:

1. Giao diện từ xa phải là một giao diện public, tức là khi tạo ra một giao diện từ xa ta phải thêm từ khoá public vào trước định nghĩa giao diện. Bằng không, khi bạn tham chiếu đến đối tượng từ xa bạn sẽ thu được một ngoại lệ
2. Giao diện từ xa phải là giao diện được kế thừa từ giao diện Remote
3. Mỗi phương pháp trong giao diện từ xa phải khai báo RemoteException trong mệnh đề throws bên bất cứ ngoại lệ nào khác, tức là tất cả các phương thức trong giao diện từ xa đều phải ném ra ngoại lệ RemoteException
4. Nếu tham số truyền cho phương thức hoặc giá trị nhận về từ thức triệu gọi từ xa là một đối tượng thì đối tượng đó phải triển khai giao diện **Remote** hoặc giao diện **Serializable**

Thông thường bạn thường thấy một giao diện từ xa có cấu trúc như sau:

```
import java.rmi.*;

public interface RemoteInterface extends Remote{

[public] ReturnDataType method1([DataType arg1,][ DataType arg2,…] ) throws RemoteException;

[public] ReturnDataType method2() throws RemoteException;

}
```

Ví dụ 1-1: Sau đây là một giao diện từ xa đơn giản của ứng dụng HelloRMI

HelloRMI.java

```
import java.rmi.*;

public interface HelloRMI extends Remote{
```

```
public String sayHello() throws RemoteException;

}
```

Nhìn vào giao diện này ta thấy nó giống bất kỳ giao diện nào khác ngoại trừ nó được mở rộng từ giao diện Remote và tất cả các phương thức trong giao diện này đều phải nem ra ngoại lệ *RemoteException*, bạn hãy nhớ rằng tất cả các phương thức được khai báo trong giao diện sẽ tự động là public, thế nên trong giao diện trên bạn có thể bỏ từ khoá public khi khai báo phương thức sayHello.

Bạn tiến hành biên dịch javac HelloRMI.java bạn sẽ thu được tệp tin HelloRMI.class

Triển khai giao diện từ xa

Sau khi bạn tạo ra giao diện từ xa, công việc tiếp theo mà bạn cần phải làm là triển khai tất cả các phương thức trong giao diện từ xa.

Ví dụ 1-2: Sau đây là cài đặt của giao diện từ xa HelloRMI

HelloRMIImpl.java

```
import java.rmi.*;

public class HelloRMIImpl implements HelloRMI {

public String sayHello() throws RemoteException

{

return "Hello RMI";

}

}
```

Vấn đề đặt ra là làm thế nào để cài đặt đối tượng HelloRMI lên một máy (máy 2) và gọi phương thức sayHello() của HelloRMI từ máy khác (máy 1)? Như đã nêu ở trên ta không gọi được phương thức sayHello của HelloRMI một cách trực tiếp mà cần có thêm hai lớp trung gian là HelloRMIImpl_Stub và HelloRMIImpl_Skel. Dựa vào lớp HelloRMIImpl.class, trình biên dịch *rmic.exe* của Java sẽ giúp ta tạo ra hai lớp trung gian Stub và Skel. Bạn mở cửa sổ DOS – Prompt và gõ vào dòng lệnh

Rmic HelloRMIImpl.class

Kết quả bạn sẽ thu được hai tệp *HelloRMIImpl_Stub.class và HelloRMIImpl_Skel.class*

Cài đặt, đăng kí đối tượng từ xa

Bước tiếp theo sau khi bạn cài đặt giao diện từ xa là công việc đăng ký nó với trình rmiregistry theo mẫu sau:

Ví dụ 1-4: Ví dụ sau là chương trình đăng ký và cài đặt đới tượng HelloRMIImpl với trình chủ rmiregistry.

Setup.java

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

import java.rmi.*;

import java.net.*;

public class Setup {

public static void main(String[] args) {

// tạo ra một thể hiện của đối tượng từ xa

HelloRMI h=new HelloRMIImpl();

try {

// Khai báo đối tượng có khả năng triệu gọi từ xa

UnicastRemoteObject.exportObject(h);

// đăng ký nó với trình rmiregistry

Naming.bind("//localhost/hello”, h);

// có thể thay phương thức bind bởi phương thức rebind như sau:

// Naming.rebind("//localhost/hello”, h);

}

catch (MalformedURLException ex) {
```

```
}

catch (AlreadyBoundException ex) {

}

catch (RemoteException ex) {

}

}

}
```

Công việc đầu tiên bạn phải làm khi cài đặt đối tượng từ xa là tạo ra một thể hiện của đối tượng từ xa.

HelloRMI h=new HelloRMIImpl(); bạn có thể thay bằng dòng lệnh

```
HelloRMIImpl h=new HelloRMIImpl();
```

Tiếp theo bạn gọi phương thức tĩnh exportObject của lớp UnicastRemoteObject để máy ảo java biết đối tượng h (HelloRMI) là đối tượng có khả năng truy xuất từ xa.

```
UnicastRemoteObject.exportObject(h);
```

Lưu ý để sử dụng được lớp *UnicastRemoteObject* bạn phải khai báo import java.rmi.server.*; ở đầu chương trình

Bước cuối cùng là bạn đặt cho đối tượng h một cái tên gợi nhớ và đẵng ký tên này với bộ đăng ký rmiregistry. Phương thức tĩnh của lớp Naming sẽ thực hiện điều này

```
Naming.bind("[rmi:]//hostname[:port]/name", h);
```

Phương thức bind có hai tham số: tham số thứ nhất là một 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]/name
```

Trong đó:

- mri là tên giao thức,đây là phần tuỳ chọn, mặc định giao thức là rmi nên bạn có thể bỏ qua.
- Hostname là tên của máy chủ hoặc địa chỉ IP của máy chủ nơi đối tượng phân tán đang tồn tại.
- Port là số hiệu cổng của chương trình rmiregistry, đây là tham số tuỳ chọn, nếu bỏ qua tham số này thì cổng mặc định là 1009
- Name là tên gợi nhớ của đối tượng phân tán

Phương thức bind sẽ đi vào vòng lặp vô hạn chờ kết nối từ máy kách.

Các chương trình phía máy khách sẽ dựa vào chuỗi định vị URL mà ta đăng ký với trình rmiregistry để truy tìm tham chiếu đến đối cần dùng.

Bạn có thể khởi động bộ đăng ký rmiregistry này ở đâu? Bạn mở cửa sổ DOS-Prompt và cho chạy chương trình rmiregistry này từ dòng lệnh như sau:

Chuyển tham số trong các lời gọi từ xa

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 kkhi biến đối tượng được truyền vào phương thức nếu bên trong phương thức 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 Nuamber 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 được tăng trị lên 13.

Tuy nhiên các kiểu dữ liệu nguyên tố như int ,float .double,char ,byte,long…lại được truyền theo 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?Thực 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 lại hai nguyên tắc sau:

*Tất cả các kiểu dữ liệu đơn giản như int ,char …dều được truyền theo tham trị .

*Tất cả các dữ liệu kiểu đối tượng muốn truyền qua mạng đều buộc phảI cài đặt một 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.

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 caì đặt giao tiếp Serializable cho lên cúng được truyền cho các lời gọi hàm hay phương thức ở xa theo trị.

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 .Tại máy chủ đối tượng sẽ được bung ra lại trạng thái ban đầu và đưa vào sử dụng.Quá trình đóng gói 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 sữ liệu qua lại giữa hai lớp trung gian _Stub và _Skel trong thuật ngữ lạp trình phân tán gọi là mashaling data.

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

Dưới đây là một chương trình ví dụ cho thấy cách chuyển đổi 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 bóng lên máy chủ , trình chủ sẽ tiếp nhận quả bóng và trả về cho trình khách.

Bước1: Thiết kế lớp Ball.class được dùng làm tham số chuyển qua mạng giữa trình khách và trình chủ , được cài đặt như sau:

+ Ví dụ: Ball.class

```
Import java.io.*;

Public class Ball implements Serializable{

Int weight =o;

Public Ball(int w){

Weght=w;

}

Public int getweight(){

Return weight;

}
```

```
Public void setweight(int w){

Weight=w;

}

}
```

Lớp Ball chỉ dùng để biểu diễn một quả bóng có trọng lượng weight. Bnạ có thể gọi hàm **getweight(), setweight()** để lấy về hoặc thiết lập trọng lượng cho qaủ bóng . 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 hóa

```
Public class Ball implements Serializable{

…

}
```

Nừu bạn cảm thấy từ ngữ "tuần tự hóa " hay "Serializable" nghe có thể khó hiểu cũng không nên bận tâm .

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

+ Ví dụ: pingServer.java

```
Import java.rmi.*;

Public interface PingServer extends Remote{

Public Ball ping(Ball b) thows RemoteException;

}
```

PingServer được đặt lên 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 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 Ballcho 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: PingServerImpl.java

Import java.rmi.*;

Public class PingServerImpl implements PingServer{

Public Ball ping(Ball b) thows RemoteException{

System.out.println("client send a ball objectweight"+b.getWeight());

b.setWeight(b.getWeight()+15);

return b;

}

}

Trình chủ của ta tiếp nhận quả bóng từ trình khách . Tăng trọng lượng của bóng lên và ném trả về cho trình khách

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

+Ví dụ: Setup.java

Import java.rmi.*;

Import java.rmi.server.*;

Public class Setup{

Public static void main(String[] args) thows Exception{

PingServer server=new PingServerImpl();

UnicastRemoteObject.exportObject(server);

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: Client.java

```
Import java.rmi.*;

public class client {

public static void main(String[]arge) thows Exception {

Ball ball=new Ball(12);

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:\RMI\ByValue. Chuyển vào thư mục này thực hiện c:\RMI\ByValue\>javac *.java tiếp đến tạo các tập tin trung gian _Stub và _Skel bằng lệnh:

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

Kết quả thu được các tệp tin sau:

Ball.class

PingServer.class

PingServerImpl_Stub.class

PingServerImpl_Skel.class

Setup.class

Client.class

Bước 7: Cài đặt đối tượng và thực thi chương trình

- khởi động bộ đăng ký Rmiregistry:

C:\RMI\ByVlaue\>Start rmiregistry

- gọi trình Setup để đăng ký đối tượng PingServer

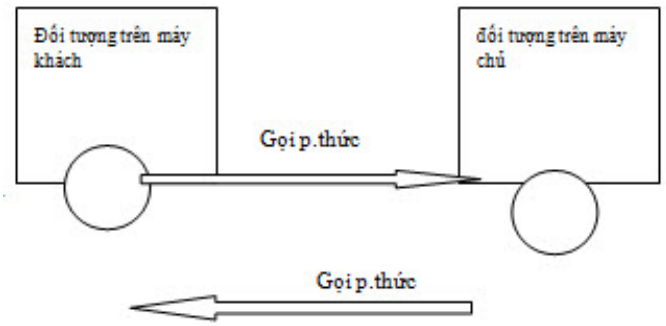
C:\RMI\ByValue\>Start java Setup

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 tới



Tốc độ thực thi của chương trình . Có cách nào mà trình chủ cói thể tham chiếu và xử lý trực tiếp được đối tượng dang 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 bằng trình chủ

Xây dựng chương trình chuyển đối tượng qua mạng theo 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 như thường lệ ,trình khách sẽ liên lạc với bộ đăng ký rmiregitry để tìm tham chiếu đên đối tượngAtServer 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

Bước 1: Đặc tả giao tiếp interface cho đối tượng Atclient.

Ví dụ: Atclient.java

Import java.rmi.*;

Public interface Atclient extends Remote{

Public void callClientMethod(String message) thows 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:AtServer.java

Import java.rmi.*;

Public interface AtServer extends Remote{

Public void RegisterClient(AtClient c) thows RemoteException;

Public void CallServerMethod(String Message) thows 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 máy khách. Phương thức thứ hai **callServerMethod()** được dùng để cung cấp các dịch vụ cho trinhf khách

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

Ví dụ: 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 :AtServerImpl.java

Import java.rmi.*;

Public class AtServerImpl implements AtServer{

Atclient client;

Public void registerClient (AtClient c) throws RemoteException()_Public void RegisterClient(AtClient c) thows RemoteException{client=c;}

}

Public void callServerMehtod(String message) throws RemoteException{

System.out.println(message);

For(int i=1; i<10;i++){

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

Client.callClientMethod(msg);

}

}

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

Ví dụ: Setup.java

Import java.rmi.*;

Import java.rmi.server.*;

Public class setup{

Public static void main(String[] args) throws Exception{

```
AtServer server=new AtServerImpl();

UnicastRemoteObject.exportObject(server);

Naming.bind("rmi://localhost/serverobject",server);

System.out.println("waiting for client request…");

}

}
```

Bước 6: thiết kế chương trình clientgọi phương thức của Atserver.

Ví dụ: Client.java

```
Import java.rmi.*;

Import java.rmi.server.*;

Public class Client{

Public static void main(string [] args) throws Exception{

AtClient client=new AtclientImpl();

UnicastRemoteObject.exportObject(client);

AtServer server=(AtServer) Namin.lookup("rmi://localhost/serverobject");

Server.registerClient(client);

Server.callServerMethod("client contact Server");

}

}
```

Bởi vì AtClient là một đối tượng có khả năeng tham chiếu xa cho nên tương tự đối tượng trên máy chủ chúng ta phảI gọi **unicastRemoteObject()** 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. Phương thức này mục đích này 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 đôI tượng AtClient của khách trên máy rmiregistry.

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

C:\RMI\Byef\>javac *.java

Tìm hiểu RMIREGISTRY và các cách đăng kí đối tượng

Trước giờ chúng ta vãn 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 chương trình dịch vụ chạy ở hậu trường ,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 cổng mặc định

Ví dụ c:\j2sdk1.4\bin>Rmiregistry.exe2003

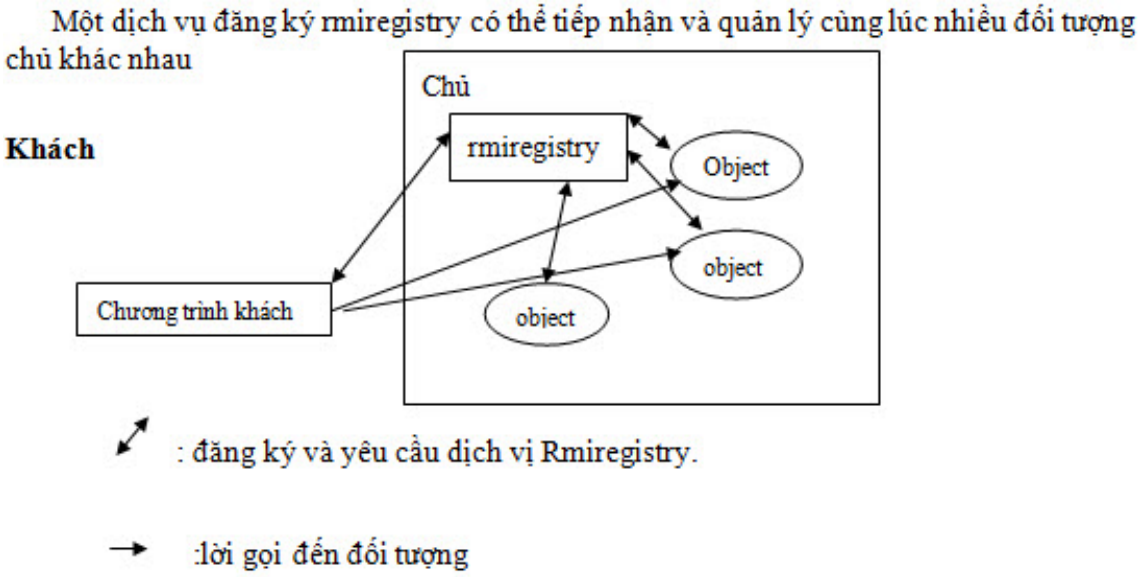
sẽ chạy dịch vụ đăng ký rmiregistry trên cổng 2003.Đối tượng trên máy chủ có thể đăng ký đến dịch vụ này bằng cách gọi Naming.bind() theo cách :

MmyObject o=new MyObjectImpl();

Naming.bind("rmi://localhost:2003/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=(M yObject) Naming.lookup("rmi://localhost:2003/myobjectname");



Cơ chế đăng ký nhiều đối tượng với rmiregistry

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 gi bạn

chỉ ciệc yêu cầu cửa hàng cung cấp danh sách dịch vụ hiện có. Chọn ra một dịch vụ sau đó yêu cầu dịch vụ đó thực hiện chức năng của mình.

CHƯƠNG TRÌNH DUYỆT CÁC ĐỐI TƯỢNG CÓ TRONG RMIREGISTRY

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

Bước 1:Xây dựng chương trình duyệt rmiregistry.

ví dụ: Traverse.java

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

Pubic calss Traverse{

Public static void main (String[] args) throws Exception{

String hostAddr=”127.0.0.1”;

System.out.println(“connecting registry…”);

Registry registru\y=LocateRegistry.getregistry(hostAddr);

String objectAvailable[]=registry.list();

System.out.println(“Registry object:”);

For(int i=0;i<objectAvailable.lengh;i++){

System.out.println(“objectAvailable[i]”);

}

}

}
```

Biên dịch chương trình :

Javac Traverse.java

Bước 2:khởi đọng và đăng ký một số đối tượng rmiregistry.

Khởi động bộ đăng ký :

Start rmiregistry

Đăng ký đối tượng Calculator với bọ đăng ký:

C:\rmi\Calculator>java Setup

Đăng ký đối tượng Hello với bộ đăng ký :

C:\rmi\ByValue>java Setup

Bước 3: Chạy chương trình Traverse:

c:\rmi\Traverse>java Traverse

myCalculator

myhello

pingobject

Bạn sẽ liệt kê được tổng cộng 3 đối tượng đã đăng ký với rmiregistry.Chương trình của tấử dụng lớp **LocateRegistry**. Chứa trong gói java.rmi.registry. Bạn cung cấp địa chỉ Ipcủa máy ảo nơi rmiregistry đang chạy, phương thức tĩnh GetRegistry() cua rlớp này sẽ trả về đối tượng registry.

Registry registry=LocateRegistry.getRegistry(hostAddr);

TẠO BỘ ĐĂNG KÝ CỤC BỘ

Java cho phép bạn tự tạo bộ đăng 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 trên có thể tự đăng ký đố tượng hello theo cách sau:

Ví dụ: Setup.java

Import java.rmi.*;

Import java.rmi.server.*;

Import java.rmi.registry.*;

```
Public class setup{

Public static void main(String[]args) throws Exception{

LocateRegistry.createRegistry(1099);

helloImpl hello=new HelloImpl();

UnicastRemoteObject.exportObject(hello);

System.out.println("Registering object…");

Naming.bind("rmi:/localhost/myhello",hello);

System.out.println("Waiting for client request…");

}

}
```

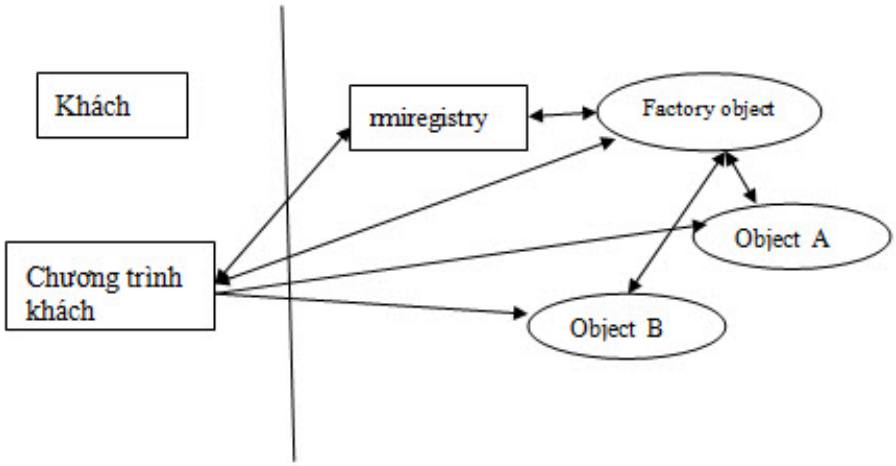
Bạn chỉ cần biên dịch và gọi chương trình java setup của ví dụn trên từ máy chủ. Setup sẽ tự động tạo bộ đăng ký rmiregistry ở cổng mặc định 1099 sau đó tự đăng ký đối tượng với rmiregistry do mình tạo ra. Tuy nhiên cách này chỉ có thể áp dụng một đối tượng. Nếu một đối tượng nào đó đã tạo ra bộ đăng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

Dùng một đối tượng sản sinh nhiều đối tượng (FACTORY OBJECT)

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 đối tượng này. Bạn có thể xây dựng một đối tượng duy nhất và đăng ký đối tượng với rmiregistry. Nhiêm vụ của đối tượng trên chỉ dùng để tạo ra các đối tượng con khác.Một đối tượng như vậy như vậy trong mô hình hướng đối tượng gọi là factory object .

Khi muốn yêu cầu tham chiếu hay triệu gọi một dối 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 dối tượng con cần thiết



Mô hình hoạt động của đối tượng Factory

CÀI ĐẶT ỨNG DỤNG FACTORY

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

Bước 1: Đặc tả giao tiếp và đặt đối tượng timer.

Ví dụ: timer.java

```
import java.rmi.*;

import java.util.*;

Public interface Timer extends Remote{

Public date getCurrentTime() throws RemoteException;

}
```

Ví dụ: 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: News.java

```
import java.rmi.*;

public interface News extends Remote{

public String gettodayNews() throws RemoteException;
```

}

Ví dụ: NewsImpl.java.

import java.rmi.*;

public class NewsImpl implements News{

String newslist[]={“Sun released new java version 1.4 and Mobie”,

“Microsoft annuonce Windows.Net 2003”,

“Fujitsu Win Telcomunication Prọect”

};

Static int index=0;

Public String getTodayNews() throws RemoteException{

System.out.println(“client request news”);

Index++;

If(index>2) index=0;

Return newslist[index];

}

}

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

Bước 3:Đặc tả giao tiếp và cài đặt dối tượng FacetoryService

ví dụ: FacetoryService.java

import java.rmi.*;

```
public interface FacetoryService extends Remote{
public News createNews() throws RemoteException;
public Times createTimes() throws RemoteException;
}
```

Ví dụ: FacetoryServiceImpl.java

```
import java.rmi.*;
import java.rmi.server.*;
public class FacetoryServiceImpl implements FacetoryService{
News newobject=new NewsImpl();
Timer timerobject=new TimerImpl();
Public FacetoryServiceImpl(){
Try{
UnicastRemoteObject.exportObject(Newsobject);
UnicastRemoteObject.exportObject(Timesobject);
}catch(Exception e){
System.out.println(e);
}
}
Public News createNews() throws RemoteException{
Return newsobject;
}
Public Timer createnews () 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 để thể hiện của hai đối tượng này đồng thời gọi **UnicastRemoteObject** () 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 vói rmiregistry.

Ví dụ: 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");

FactoryService obj=new FactoryServiceImpl ();

UnicastRemoteObject.exportObject(obj);

Naming.bind("rmi://localhost/myfactory",obj);

}

}
```

Trình setup.Java vẫn như thường lệchịu trách nhiệm 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 **miregistry** 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 như sau:

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

Ví dụ: Client.Java

Import java.rmi.*;

Import java.rmi.server.*;

Public static voidmain (String ags{}) throws Exception{

FactoryService service=(Factoryservce) Naming.lookup(“mi://locahost/ myfactory”);

News news=service.createNews();

Timer timer=service.createTimer();

System.out.println(“current”+timer.getcurrentTimer());

System.out.println(“We have new”+news.getTodaynews());

Thead.sleep(1000);

System.out.println(“current”+timer.getcurrentTimer());

System.out.println(“We have new”+news.getTodaynews());

}

}

Nếu như bạn thấy trình khách của ta chỉ việc tham chiếu đến FactoryService bằng hàm:

FactoryServiceservice=(Factoryservice)Name.lookup(“rmi://localhost/ myfactory”);

Có đối tượng service trình khách yêu cấud service cung cấp cho mình tham chiếu đến 2 đối tượngcon News và Timer như sau:

News news=service.createNews();

Timer timer**=service.createTime();**

Việc truy xuất News và Timer diễn ra không có gì khó khăn lắm. Chương tình client của ta gọi **getcurrentTime(), geTodaysNew()** để lấy về giờ hiện hành và không có tin trên máy chủ.

Bước 6: Biên dịch và chạy chương trình.

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

C:\RMI\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 giá biên dịch tiếp theo sau:

C:\RMI\Factory>rmic Timerimpl

C:\RMI\Factory>rmic NewsImpl

C:\RMI\Factory>rmic FactoryServiceImpl

Khởi động bộ đăng ký rmiregisty

C:\RMI\Factory>start rmiregisty

Khỏi động trình đăng ký đối tượng FactoryService.

C:\RMI\Factory>Start java Setup

Chương trình khách:

C:\RMI\Factory>java clien

Kết quả kết xuất trên máy khách.

Current Mon Jun 27 21:53:56 PDT 2003

We have news: Microft announce Windows 2003

Current Mon Jun 27 21:53:57 PDT 2003

We have news: Fujitsu setup new factory in Vietnam

Kết quả kết xuất trên máy chủ.

Lient request time

Lient request news

Lient request time

Lient 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 {EntePrise Java Bean} rất nổi tiếng trong các phiên bản Java 1.2 trở đi. Chúng ta sẽ nghiên cứu về EJB trong chương trình sau. Lúc đó bạn sẽ xó dịp tham gia lại kỹ thuật này.

Kỹ thuật gọi đối tượng ở xa bằng phương thức động(DYNAMIC METHOD INVOKE)

Qua các chương trình và các ví dụ ở trên ta thấy muốn khách quan 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 ở dạng tường minh sau khi nhận được tham chiếu đến đối tượng bằng hàm Naming.lookup().

VD:

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

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

Hello và PingServer là các lớp giao tiếp của đối tượng mà nhà cung cáp phải gửi đến. Cách gọi hàm này được xem là cách gọi hàm tĩnh. Bởi trình khách sau khi chuyển kiểu đôi 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 có lớp giao tiếp Interface của đối tượng để chuyển kiểu. Trường hợp này bạn có thể sử sụng 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 đang nấm giữ. Hàng 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 tương Friend

Ví dụ : Friend.java

Import java.rmi.* ;

Public interface Friend extends Remote {

Public String greeting (Strinh name) throws RemoteException;

Bước 2 : Cài đặt chi tiết cho đối tượng

Ví dụ : FriendImpl.Java

Import java.rmi.* ;

Public class FriendImpl implements Friend {

Public String greeting (Strinh name) throws RemoteException {

Return (“Hi” + Name + “! Nice to meetyou”);

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

Ví dụ : Setup.Java

Import java.rmi.*;

Inport java.rmi.server.* ;

Public class setup {

Public static void main (String arg[]) throws Exception{

Friend obj = new FriendImpl();

UnicastRemoteObject.exportObject(obj);

System.out.println(“Friend waiting for client request...”);

Naming.bind(“rmi:// localhost / myfriend”,obj)

}

}

Bươc 4: Viêt chương trình khách

Import java.rmi.*;

Inport java.lang.reflect.* ;

Public class Client {

Public static void main (String arg[]) throws Exception{

// Tìm tham chiếu của đói tượng

Object o = Naming.lookup (

// Truy tìm xuất xứ của lớp đối tượng

Class c = o.getCl ass () ;

// Định kiểu tham số của phương thức cần gọi

Class [] parameterTypes = new Class [] {String.class};

// Truy tìm phương thức cần gọi

Method thêMethod;

TheMethod = c.getMethod ("Greeting",parameterTypes);

// Định số truyền vào phương thức cần gọi

Object[] arguments = new Object [] {"Johny"}

// Gọi phương thức và nhận kết quả trả về

String result = (String) thêMethod.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 :\\ RMI \ Dynamic. Chuyển vào thư mục này để biên dịch mã nguồn.

C :\\ RMI \ Dynamic> Javac*.java

- Biên dịch lớp trung gian :

C :\\ RMI \ Dynamic> rmic FriendImpl

- Khởi động bộ đăng ký rmiregistry :

C :\\ RMI \ Dynamic> Start rmiregistry

- Cài đặt đối tượng:

C :\\ RMI \ Dynamic> start Java Setup

Friend waiting for client resquest…

- Gọi đối tượng từ trình khách :

C :\\ RMI \ Dynamic> star 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 thưởng 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 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 kiểm tham số của phương thức cần gọi :*

Class [] parameterTypes = new Class [] {String.class};

// *Yêu cầu Java truy tìm phương thức cần gọi :*

Method thêMethod;

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 tham trỏ đến. Trước khi gọi phương thức chúng ta cần khởi tạo danh sách đối số để truyền vào phương thức như sau:

// *Đinh đối số cần vào phương thức càn 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) thêMethod.invoke (o, arguments);

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

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

ở ví dụ trên thay cho lời gọi động bạn có thể thu được cùng 1 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.

Tự động kích hoạt các đối tượng từ xa (ACTIVATION)

Quy trình mà ta vẫn thực hiện từ trước đến giờ khi cài đăt 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 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 10000 đối tượng đều có khả năng phục vụ cho máy khách . Vậy thì để sử dụng được 10000 đối tượng này trước tiên bạn phải xin cấp phát bộ nhớ cho tất cả các 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 1 thời điểm.Java cung cấp cơ chế tự động kích hoạt (activation) đối tượng trên mấy 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 dặt 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 đói tượng khi máy khách có nhu 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 : 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 1 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 : 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.new.Instance 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 trọ cho kỹ thuật tự kích hoạt nằm trong gói thư viện Java.rmi.Activation.*) .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, đói số thứ nhất cho biết định danh kích hoạt (ActivationID) do dịch vụ kích hoạt rmid truyền vào .Đối số thứ 2 (Data) có kiểu đối tượng MarshalledObject là đối tượng tổng quát có thể truyền qua lại trên mạng(MarshalledObject là đối tượng được cài đặt giao tiếp SerialiZable) dùng cho mục đích chung khi bạn muốn truyền thônng tin từ bên ngoài vào trong quá trình đối tượng khởi động. Bạn phải chuyển những thông tiu này lên cho lớp cha bằng cách gọi Super(id,0).

Lóp cha là lớp Activatable sẽ thông báo và đăng ký sự hiện diện của đối tượng với dịch vu rmid và máy ảo java. Bạn sẽ nhận rõ cơ chế hoạt động này trong các bước cài đặt chương trình setup.java tiếp theo sau đây :

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

Ví dụ : Setup

Import java.rmi.* ;

Import java.rmi.activation.* ;

Import java.util . properties ;

Public class Setup {

// Chương trình đăng ký đối tượng MyAutoObject với rmid va 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 java1.2 bạn bắt buộc phải chỉ định tập tin chính sách policy quy định chuyển cho máy ảo*

// Chạy đối tượng kích hoạt

// *Properties là bảng thuộc tính 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 exampleGroup = new ActivationGroupDesc (props , ace) ;

// Dựa vào thông tin của nhóm, chúng ta yêu cầu hệ thống cung cấp 1 định danh tính kích hoạt cho nhóm

//

ActivationGroupID agi = ActivationGroup.getSystem ().registryGroup(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. Dịnh 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 kí 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 càn truyền thông tin gì cả nên data được đặt giá trị bằng null

MarshalledObject data = null;

// *Tạo mô tả về đối tượng MyAutoObject cần kích hoạt*

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

// Đăng kí đối tượng với dịch vụ kích hoạt rmid

MyAutoObject mri = (MyAutoObject) Activatable.registry(Desc);

// Đăng kí tên của đối tượng với rmiregistry đẻ máy khách truy tìm

//

System.out.println(“registry Object with rmiregistry”);

Naming.rebind (“rmi : // localhost / MyAutoObjectImpl” , mri);

System.out.println(“avainable for client request”);

System.exit(0);

}

}
```

Bước đăng kí đối tượng với dịch vụ kích hoạt rmid trông có vẻ phức tạp nhưng đó là nhưng bước java yêu cầu. Các đối tượng cần kích hoạt được quản lí theo từng nhóm. Trước tiên bạn cần chỉ định môI trường mà nhóm sẽ hoạt động :

```
ActivationGroupDesc.commandEnviroment ace = null ;
```

ở đây ta sử dụng các giá trị mặc định của môi trường java nên biến ace được đặt giá trị null

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

```
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.getsysstem().registerGroup(ExampleGroup);
```

Tiếp theo là tạo nhóm

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

Sau đó là mô tả về đối tượng cần kích hoạt

ActivationGroupDesc desc = new ActivationGroupDesc(“MyAutoObjectImpl”,location , data);

Bước cuối cùng là đăng kí 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 trên 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 kí 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 kí với dịch vụ rmid

MyAutoObjectImpl = (MyAutoObjectImpl) 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 : Client.java

Import java.rmi.*;

Public Class Client{

Public static void main (String args[]).{

// Thay localhost bằng địa chỉ IP thích hợp nếu bạn 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 chế độ tự động kích hoạt cảu trình chủ diễn ra như thế nào. Theo cách thông thường trình khách gọi Naming.lookup () để về lấy tham chiều và gọi trực tiếp đến phương thức callMeRemotely()của đối tượng chủ .Trước khi thực hiện biên dịch bạn cần tạo tập tin chứa các quy định bảo mật khi chạy chương trình như sau :

Ví dụ: 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 windowns để chỉnh sửa và lưu tập tin dươi tên policy

Bước 5: Biên dịch chương trình và chạy chương trình.Bạn lưu tất cả mã nguồn vào thư mục C:\RMI \ Activate .Chuyển vào thư mục nảy để biên dich và chạy chương trình theo thứ tự sau :

Biên dịch các tập tin .class:

C:\RMI \Activate> javac*.java

Tạo lớp trung gian

C;\RMI\Activate>rmic MyAutoObjectImpl

Khởi động bộ đăng ký Registry:

C:\RMI\Activate>Start rmiregistry

Khởi động bộ quản lý kích hoạt đối tượng:

C:\RMI\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 cho Java tiếp nhận . ở đây ta yêu cầu rmid khởi động nhờ vào tính chất bảo mật quy định trong tập tin policy)

Khởi động trình cài đặt setup đăng ký đói tương:

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 quy 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

Tùy 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 đọng hiệu quả hơn.

Trình mồi nạp ứng dụng từ xa

Điều thú vị của hệ thống mạng là cho phép bạn kêt nối giao tiếp và điều khiển từ xa. Một câu hỏi đặt ra là có cách nào viết 1 ứng dụng hoàn chỉnh đăt 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 lạp về 1 lớp chương trình trên máy chủ theo giao thức http . RMIClassloader cũng chịu trách nhiệm đem về tất cả các lóp con hay lớp thư viện mà lóp chương trình cần đến .Điều này tương tự cơ chế làm việc của Applet.Trong trường hợp Applet , trình duyêt browser chính thức là bộ nạp lơp dễ thấy nhất.Tất cả những lớp thư viện hay lớp con mà Applet dùng đến đều dược đặt trên máy chủ và được trình duyệt tự động download về đầy đủ trước khi gọi Applet thực thi.

Dưới đây là 1 chương trình ví dụ triệu goi j ứng dụng từ xa.ứng dụng của ta được thiết kế theo mô hình phan tán , ứng dụng sẽ goi j phương thức của 1 đối tượng ở xa . Đối tượng này là một từ điển có trách nhiệm dỏ tìm và trả về ý nghĩa của 1 từ do ứng dụng yêu cầu. Toàn bộ ứng dụng được đặt trên web server của 1 máy chủ .Chương trình khách chỉ cần dùng lớp RMIClassloader để triệu gọi ứng dụng từ web server.

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 1 máy đơn) hoặc địa chỉ IP do người quản tri mạng cấp cho bạn .

Thư mục gốc của webserver mà ta đã xây dựng trước đây là : C ; \ webserver\webroot\ Bạn vào thư mục này tạo thêm thư mục con DictApp để chứa chương trình ứng dụng . Đường dẫn đầy đủ là C ; \ webserver\webroot\ DictApp.

Đường dẫn URL tương ứng là http://172.16.11.12 / DictApp/ (Hoặc http://172.0.0.1 /DictApp/ nếu bạn thử nghiem trên máy cục bộ)

Dưới đay mà mã nguồn của các lớp cài đặt trên máy chủ :

Ví dụ : Dictionary Java

```
Import Java.rmi.*;

/*Lớp giao tiếp đặc tả các phương thức của đối tương Dictionary*/
```

Public interface Dictionnary extends Remote {

Public string FindWord (String KeyWord) throws RemoteException ; }

Ví dụ :DictionaryImplJava

Inport Java.rmi,*;

/*Lóp cài dặt đối tượng Dictionary*/

Public Class DictionaryImpl Impl implements Dictionary {

String woodlist [ ] ; //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(woodlist[i]) return meaning[i];

}

Return”word not found”;

}

}

Ví dụ : DictionaryServerjava

Inport java.rmi.server.*;

Import java.rmi.* ;

Import java.rmi.registry.* ;

/**Lớp kiến tạo và đăng ký đối tượng Dictionary với rmiregistry* */

Public class DictionaryServer Ơ

Public static void main (String args [ ]) {

Try {

Dictionary d= new DictionaryImpl ();

UnicastRemoteObject.exportObject(d);

//*Thay địa chỉ IP của máy chủ bằng địa chỉ IP của máy bạn nêu 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 Naming.blind(“//127.0.0.1/ mydictionary”,d);

} catch (Exception e) {

System . out . println(e) ;

}

}

}

Ví dụ : 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 / my dictionary") ;

//*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 tâp tin này bạn sẽ thu được 4 lóp 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 tương 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:\ webserver \ webroot \ DictApp

Với C:\webserver \ webroot đượ coi là đường dẫn bắt đầu (webroot) của webserver.Khởi động webserver mà ta đã tạo ở chương 1 từ dong lệnh :

Java Javawebserver

Bước 2: Khởi động rmiregistry cách từ dòng lệnh :

rmiregistry.exe

Bước3 : Khởi đông chương trình đăng kí đối tượng Dictionary:

DictApp> Java

- Djava.rmi.server.codebase = “http://127.0.0.1 /DictApp / Dictionary

Server ”

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

\ DictApp>java DictionaryClient

Để kiểm tra kết quả xuất . Tuy nhiên DictionaryClient không nhằm chạy trên máy chủ. Nó có thể được triệu gọi từ máy khách ở xa . Có 1 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 RMIClassloader được đề cập ngay sau đây.

THIẾT KẾ CHƯƠNG TRÌNH GỌI ỨNG DỤNG PHÍA MÁY KHÁCH

Trên máy khách bạn không cần bất kỳ lóp 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 mà nguồn của ví dụ 2-46 dươi đây vào thư mục bất kỳ.

Ví dụ : 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 tù xa .Java gọi cơ chế này là cơ chế mồi (BootStrap)

Chương trình mồi có thể được viết tổng quát hơn như sau :

Ví dụ :BootStrapApp.java

```
Import java.rmi.*;

Import java.rmi.server.* ;

Public Class BootStrapApp {

Puclic Static void main (String Args [ ]) throws Exxeption {

IF ( Args. Length!=2 ) {

System.out.println(“Syntax”);

System.out.println(“Java BootStrapApp < URL> < AppClassName>”);

Return ;

}

// Thiết lập lớp phòng vệ cho phía máy khách

System.SetsecurityManager( new RIMSecurityManager());

// Nạp lớp ứng dụng từ máy chủ Dictionary

Class ClientClass = RMIClassloader.loadClass ( Args [ 0 ] ,args [ 1]);

// Khởi tạo thể hiện ứng dụng thông qua giao tiếp Runnable

Runnable c = (Runnable) ClientClass.newInstance();

//Chạy ứng dụng ngay trên máy khách

c.run();

}
```

Bạn lên sử dụng BootStrapApp để gọi DictionaryClient từ xa như sau:

C:\ testing> java BootStrapApp http:// 127.0.0.1 /DictApp / DictionaryClient

BootStrapApp cần 2 tham số dòng lệnh . Tham số thứ nhất là địa chỉ URL viêt hoa nơi lớp ứng dụng có thể được nạp về bằng giao thức http .Tham số thứ 2 là tên lớp ứng dụng chỉ có 1 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 dãn URL > < Tên lớp ứng dụng >

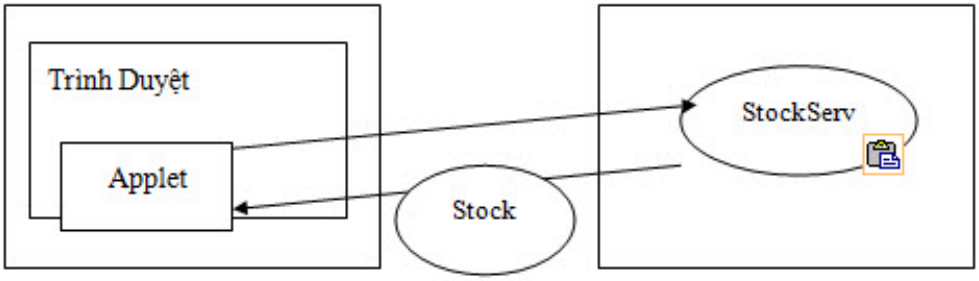
Đọc đến đây có thể bạn có thể hỏi mọi thứ với Java sao “Nhiêu khê ” lắm thế ?

Thật ra , cho đến hôm nay rất ít khi bạn phai xây dựng 1 ứng dụng lớn sử dụng ký thuật RMI như chúng tôi vừa trình bày . Java đã hoàn thiện và đưa ra mô hình J2EE hiệu quả và dễ dàng hơn bao trùm tất cả những khái niemj mà RMI cung cấp. Tuy nhiên , J2EE chính xác cũng dựa trên RMI cho nên thiết nghĩ một khi năm vững về ký thuật xử lý và lập trình đối tượng với RMI bạn sẽ tự tin hơn khi tiếp xúc với môi trường J2EE sẽ được chúng ta nghiên cưu trong phần cuối chương.

Ứng dụng RMI

Để kết thúc chương trình này chúng ta hãy xây dựng 1 ứng dụng RMI có ích thực sư. 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 : Microsof , IBM , Sun .... Cơ chế hoạt của chương trình dựa vào mô hình phân tán đối tượng và ký thuật gọi hàm từ xa RMI

Bạn hãy hình dung chúng ta có 1 đối tượng ở xa chịu trách nhiệm cung cấp giá cổ phiếu hiện hành của tất cả công ti trên thế giới . Máy khách là 1 chương tring Applet chạy trên trình duyệt Browse . Applet sẽ gửi yêu cầu đến đối tượng chủ yêu cầu mối giây phảI 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 gàn 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 1 mô hình mẫu đơn giản . Thiết kế 1 ứng dụng thực thụ đòi hỏi chúng ta phải xây dựng 1 giao diện Applet ở phía máy khách đồng thời tính toán chính xác thời gian để gửi dữ liêuk từ máy chủ về cho máy khách ở xa .Hình 2-12 là cơ chế giao tiếp giữa trình khách và đối tượng trên máy chủ .



Cơ chế truyền thông tịn 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 tạo 1 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 : Stock Java.until.* ;

// **

* 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 implement 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.nextGaussion() * 1.0));

Change = Math.abs(change); // Thay đổi này tùy bạn quyết định

Float newCurrent = Current + Change;

// *Không cho phép giá cổ phiếu vượt qua mức trần*

If (newCurrent < 0 | | newCurrent > MAX_VALUE)

Change = 0 ;

Current + = change;

Return change;

}

}

Chúng ta sử dụng đối tượng Random của gói Java.until để tạo số ngẫu nhiên mô phỏng sự biến động giá cổ phiếu. Phương thức update () trong Stock dùng để lấy về thay đổi giá cổ phiếu của 1 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 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 lai ký thuât CallBack ở mục 7-4)

Ví dụ : StockNotify,java

Import java.rmi.*;

Import java.Util,* ;

/ **

- StockNotify dùng để nhận các thông tin cập nhật về giá trị cố 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 phiều được cập nhật
- Stocks Danh sách cổ phiếu của các công ty đã dăng ký với trình chủ yêu cầu cập nhật

* /

Void Update(Date date , Stock [ ] Stocks) throws RemoteExce ption ;

}

Máy khách sẽ thực hiện 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 sớm nhất .(Chúng ta sẽ thực hien 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 : StockWatch. 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 cố phiếu mới nhất

*/

Public StockWatch (St ring stock. StockNotify Obj) throws RemoteException;

/**

- Hủy bỏ yêu cầu cập nhật giá cổ phiếu của 1 công ty nào đó

- Stock Tên cổ phiếu cần hủy bỏ
- Obj Đối tượng trình khách yêu cầu hủy bỏ cổ phiếu

*/

Public Void Canel (String stock , StockNotify Obj) throw s 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 canel All (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 StockServer

Ví dụ : StockServer,java

Import java.rmi.*;

Import java.rmi.server.*;

Import java .rmi.Registry.LocatRegistry;

Import java .util.*;

Public Class StockServer implement 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 notifyer = null

/* *Danh sách các công ty bán cổ phiếu* */

Private static String [] = {"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 trị 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 stock (Vector) notifyTable.get(obj);
// Nếu chưa có thì tạo danh sách cổ phiếu mới
If (stocks == null) {
Stocks = newVector();
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 )

{

Vector stocks = (Vector) notifyTable.get(obj);

Stock[ ] stockList = null;

If (stocks ! = null){

Enumeration enum = stocks.elements();

stockList = new stocks [ 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 (s tockNotify obj)

{

notifyTable.remote(obj);

}

/**

* *Sinh giá ngẫu nhiên cho từng cổ phiếu*

*/

Private void generateUpdate()

{

Enumeration enum = stockTable.elements();

While (enum.hasMoreElements()) {

Stock stock = (stock)enum.nextElement();

}

}

/**

* *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 2s 1 lần
Thread.currentThread().Sleep(2000);
} catch (interruptException e) {
}
// Ghi nhận ngày giờ giá cổ phiếu thay đổi
Date date = new Date();
// Tạo giá mới cho cổ phiếu
GenerateUpdate();
// Yêu cầu khách nhận dữ liệu đã cập nhập
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 kí registry. Đăng kí và
răng buộc đối tượng chủ với registry
*/
public static void main (string args [ ])
{
// Cài đặt lớp bảo vệ cho máy chủ
System.setSecurityMana ger(new RMISecurityManager());
Try {
// Tự tạo bộ đăng kí registry hoạt động trên cổng 1234
System.out.println("StockServer.main : creating server");
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);

// *Rằng 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.printStrackTrace();

}

}

}

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 2s 1 lần tính toán giá cổ phiếu 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 1 Applet ở máy khách (hay trình khách) có khả năng triệu gọi phương thức của StcokSever trên máy chủ

Bước 5 : Xây dựng Applet và cài đặt đối tượng StockNotify

Ví dụ: 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 StockSever nên ta cần gọi lệnh UnicastRemoteObject.exportObject() để thông báo với mấy 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 liêm 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.remoteElementAt(0);
}
Time.addElement(date);
// Lưu lại dữ liệu giá cổ phiếu . Bạn cần 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 .exporObject( 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 kí 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 occurrent during initialization.Check the log"));
Add(new Lablel (e.getClass().getName() + ":" +e.getMessage()));
System.out.println ("got exception:" +e.getMessage());
```

```
e.printStackTrace();
return;
}
}
// 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();
Dimersion 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.i+10,150,scale(updates [ 0].current));
}
Int x = 150,inc = 10;
For (int i = 0; i< size; i++) {
If (updates [ i] 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;
}
}
}
}
/**
* Đối tọa 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.nextElements ();
Data,cb.mouseHere = display;
```

```
}

}

/**

* Lớp StockData chứa cập nhật của giá cổ phiếu vè 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 .remoteElementAt(0);
}
If (updates.size() <numUpdates) {
For (int i= updates.size();i<numUpdates-1;i++) {
Updates.addElement(stock);
}
}
Updates.addElement(stock);
}
}
}
/**
* Lớp SensitiveCheckbox dùng để 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=date;
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(Graphic 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(Color.color);

g.fillRect(5,5,8,8);

g.setColor (Color .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 1 trang web . Ban tao 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 quy định cơ chế bảo mật của Applet và trình chủ như sau:

Mở trình soạn thảo notepad trong windows nhập vào chết 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 dich và chay chương trình ở cả 2 phía khách chủ. Bạn hãy lưu tất cả mã nguồn và tập tin Security.Policy, StockApp.html vào thư mục

C:\ RMI \ Stock \. Chuyển vào thư mục này chuyển vào thư mục này thực hiện các bước biên dich như sau :

Biên dich mã nguồn :

C: \ RMI \ Stock> Javac * Java

Biên dich tập tin trung gian cho đối tương StockServer:

C: \ RMI \ Stock > rmic StockServer

Biên dịch tập tin trung gian cho đối tương StockNotify:

C: \ RMI \ Stock > rmic StockApplet

(Bạn lưu ý , StockApplet của ta do cài đặt giao tiếp StockNotify nên bản thân nó là 1 đối tượng có thể phục vụ cho giao tiết StockNotify thật sự)

Tạo và đăng ký đối tượng tren máy chủ

C: \ RMI \ Stock > java Djava.Security.Policy= Security.Policy Stock Server

(Chúng ta kô cần khởi động chương trình rmiregistry , Stockserver tự tạo bộ đăng ký Registry ở cổng 1234 với registry chính bản thân nó.)

Khởi động Applet trên máy khách :

C: \ RMI \ Stock > AppletViewr StockApp.html

Kết quả chương trình sẽ được hiển thị như hình 2-11.Applet liệt kê 1 bảng giá cổ phiếu cùng với đồ thị biểu diễn biến động của giá cổ phiều đang tăng hay giảm trông rất sinh đọng > Bạn có thể di chuyển chuột dến từng loại cổ phiếu của các công ty khác nhau. Kích chuột vào ô Checkbox , khi bạn không đánh đau 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 StockSever , StockNotify là do trình chủ đảm nhiệm.

Chương 4: Lập trình phân tán đối tượng với Corba

Giới thiệu về Corba

Như chúng ta đã biết, Java là ngôn ngữ thuần đối tượng nhất trong các ngôn ngữ lập trình, mọi khái niệm, mọi thứ, mọi nơi đều quy về đối tượng. Các đối tượng trong ngôn ngữ lập trình hướng đối tượng được thiết kế bằng ngôn ngữ nào thì chỉ có mã lệnh tương ứng của ngôn ngữ đó mới truy xuất được chúng. Vấn đề, làm thế nào để các đối tượng được thiết kế bằng các ngôn ngữ lập trình khác nhau có thể triệu gọi và sử dụng lẫn nhau?

Sự ra đời của Java đã giúp cho mọi ứng dụng và đối tượng của nó có thể chạy mọi lúc mọi nơi trên nhiều hệ điều hành. Tuy nhiên, thế giới lập trình là vô cùng đa dạng và phong phú, có rất nhiều ngôn ngữ lập trình "tiền bối" chứa các thư viện đã được thiết kế xây dựng công phu đang hoạt động rất hiệu quả. Vì thế mà ta không thể bắt tất cả các lập trình viên chuyển hết sang dùng java. Yêu cầu đặt ra là cần có một "tiếng nói" chung cho tất cả các ngôn ngữ lập trình. Và Corba ra đời nhằm đáp ứng yêu cầu đó.

Corba (Common Object Request Broker Architecture) – tạm dịch là kiến trúc môi giới các đối tượng. Corba được xây dựng bởi một tổ chức nghiên cứu OMG (Object Manager Group) với sự hợp tác của hơn 800 công ty.

Các ngôn ngữ lập trình đều có các điểm chung là các lời gọi hàm, thủ tục, tham số truyền, trị trả về… Ngôn ngữ đặc tả ánh xạ các điểm chung đó thành những ngôn ngữ lập trình khác nhau. Corba là ngôn ngữ đặc tả giao tiếp, nó định nghĩa nhiều dịch vụ Corba hoạt động với nhiều ngôn ngữ lập trình khác nhau và không chỉ với nền java.

Corba xác định hai vấn đề cơ bản:

+ Thực thể mà cho phép liên lạc giữa 2 tiến trình được gọi là một trình môi giới yêu cầu đối tượng (Object Request Broker - ORB).

+ Một giao thức được ORB dùng để liên lạc giữa nhiều tiến trình, được gọi là IIOP (Internet Interoperability Protocol). Corba quy định một tập các mô tả hàm kiểu dữ liệu cách khai báo để bạn đặc tả đối tượng.

Corba còn được gọi là ngôn ngữ đặc tả giao tiếp (IDL – Interface Description Language). Nó mô tả chức năng của đối tượng thông qua hàm, phương thức, thuộc tính…nó không chứa bất kỳ cài đặt mã lệnh nào. Và đặc tả đối tượng dựa trên khái niệm

interface trong java. Trong 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 ở mức độ trừu tượng nhất có thể. Giả sử ở bước này bạn không muốn áp đặt với một ngôn ngữ 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 thesting)

Tiếp theo bạn muốn cài đặt ngôn ngữ khác như Pascal. Điều này không hề khó nếu bạn biết được cách khai báo tương ứng giữa đặc tả và các từ khoá cũng như kiểu dữ liệu trong Pascal cài đặt .

Ví dụ:

Ngôn ngữ đặc tả Pascal

Thefunction function

Theinteger integer

Get_price get_price

As :

Thestring string

Quá trình trên được gọi là ánh xạ từ ngôn ngữ đặc tả sang ngôn ngữ cài đặt. Đó là ý tưởng của các nhà thiết kế và xây dựng nên kiến trúc Corba. Corba qui định về một tập mô tả hàm, kiểu dữ liệu, cách khái báo để bạn đặc tả đối tượng. Đặc tả đối tuợng cũng giống như khái niệm giao diện trong java. Chỉ là mô tả các chức năng của đối tượng thông qua hàm, phương thức, thuộc tình... Không có mã lệnh nào trong ngôn ngữ đặc tả. Chính vì vậy Corba còn gọi là ngôn ngữ đặc tả giao tiếp (*Inteface Decsription Language hay IDL*).

Ví dụ: Đặc tả đối tượng Calculator bằng ngôn ngữ IDL của Corba

- Tạo file Calculator.idl

```
interface Calculator {

long addNumber ( in long x, in long y );

};
```

- Để chuyển file đặc tả này sang các ngôn ngữ lập trình khác chúng ta có thể dùng như sau:
- idl2cpp Calculator.idl // chuyển sang C++
- *idlj Calculator.idl // chuyển sang Java*
- Kết quả là chúng ta có được tập tin CalculatorOperations.Java như sau:

```
public interface CalculatorOperations

{

int addNumber(int x, int y);

} // interface CalculatorOperations
```

Bước cài đặt hàm addNumber() là do lập trình viên thực hiện. Corba chỉ giúp triệu gọi hàm addNumber() từ đối tượng CalculatorOperations.

Ánh xạ giữa ngôn ngữ IDL và Java

Ta thấy, ngôn ngữ IDL không có các điều khiển như do, for, while, if, else. Nó chỉ là ngôn ngữ đặc tả. Java cung cấp cho bạn công cụ dùng để chuyển đổi giữa khai báo IDL và khai báo java.

- Để chuyển từ một lớp giao diện của IDL của Java sang đặc tả IDL dùng lệnh:

Rmic –idl YourObject.class

- Để chuyển từ đặc tả IDL sang một lớp giao diện của IDL của Java dùng lệnh:

idlj YourObject.idl

Ví dụ : Ánh xạ giữa corba và java.

- Đặc tả trong Corbar (Corba IDL):

module {

interface MathLibrary {

long add(in long x, in long y);

string About(in string version);

}};

- Tương ứng với Java :

package Math;

public interface MathLibrary {

int add (int x, int y);

String About(String version);

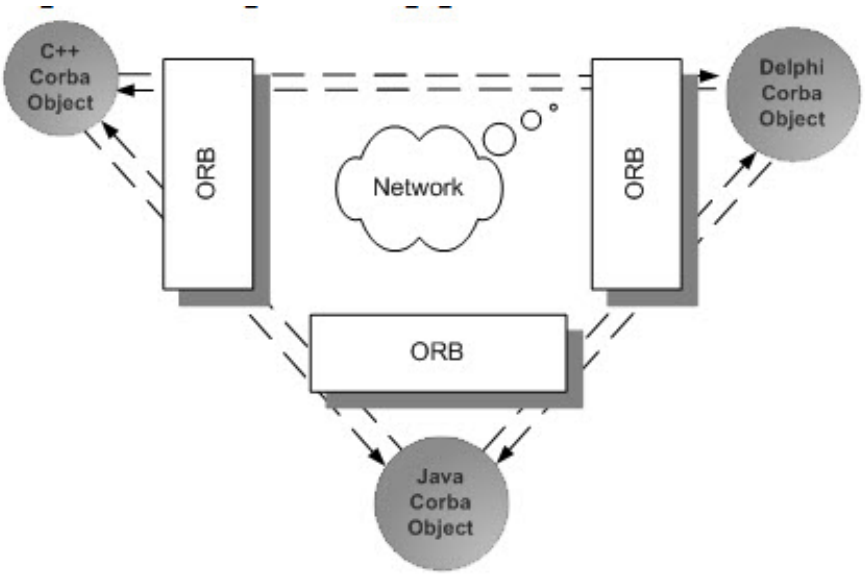
}

Ngôn ngữ đặc tả trong mô hình Corba gần giống với ngôn ngữ C. Corba đưa ra từ khóa **in** cho các biến truyền vào theo trị và từ khóa **out** để lấy trị trả về.

IDL Corba	Java
moduleinterfacecontantbooleanchar,wcharoctectstring,wstringshort,unsigned shortlong,unsigned longlong long,unsigned long longfloatdoubleenum,struct,union sequence,arrayexceptionoperation	packageinterface,help finalbooleancharbytej

Corba và mô hình phân tán.

Chương trình môi giới trung gian ORB.

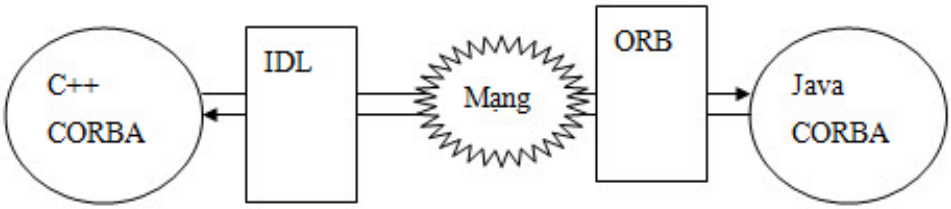


Triệu gọi đối tượng Corba thông qua môi trường môi giới ORB

Các đối tượng sau khi được xây dựng bằng các ngôn ngữ lập trình khác nhau muốn triệu gọi lẫn nhau phải thông qua chương trình môi giới trung gian của Corba gọi là ORB.

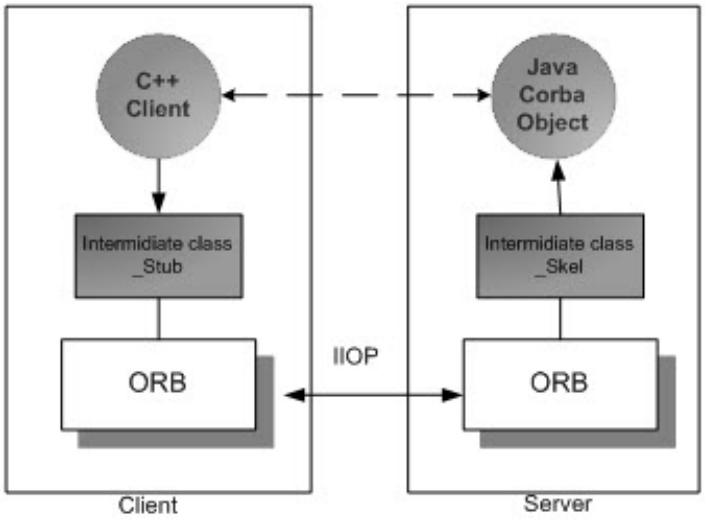
ORB hoàn toàn độc lập với máy ảo java. Ta có thể chạy độc lập Java với ORB tuy nhiên từ Java 1.2 đã nhúng luôn ORB nên ta có thể thử nghiệm các ứng dụng Corba ngay trên nền Java mà không cần trình ORB của các nhà cung cấp khác.

Các đối tượng dù được cài đặt bằng bất kì ngôn ngữ nào nếu như tuân theo ngôn ngữ IDL đều là đối tượng của Corba. Dựa vào các đặc tả IDL, ORB sẽ biết tên phương thức cần gọi, kiểu đối số truyền cho phương thức, trị trả về.



ORB giao tiếp thông qua các thông tin đặc tả IDL

Cơ chế làm việc của ORB.



Giao tiếp giữa trình khách và đối tượng Corba trên máy chủ

- Phần cài đặt đối tượng Corba trên máy chủ diễn ra như sau:
- Đặc tả đối tượng bằng ngôn ngữ IDL của Corba.
- Chuyển đặc tả đó thành mô tả của ngôn ngữ cụ thể. Dựa vào đặc tả IDL yêu cầu trình biên dịch tạo ra lớp trung gian _SKel.
- Cài đặt đối tượng trên bằng ngôn ngữ java.
- Phần kết nối với đối tượng máy khách.
- Dùng đặc tả IDL của đối tượng yêu cầu trình biên dịch tạo lớp trung gian _Stub.
- Khi trình khách triệu gọi đến đối tượng chủ, lời gọi sẽ được chuyển qua _Stub trên máy khách, gửi đến trình môi giới ORB. Trình ORB máy khách gửi đến trình ORB máy chủ theo giao thức IIOP. Trình ORB máy chủ chuyển lời gọi cho _Skel máy chủ. Cuối cùng đối tượng Corba trên trình chủ sẽ thực thi phương thức của nó.

Giao thức IIOP làm nền tảng cho tất cả các đối tượng có thể giao tiếp với nhau trên mạng Internet một cách dễ dàng.

Xây dựng Ứng dụng Corba khách/chủ trong môi trường java

Xây dựng đối tượng Corba.

Chúng ta sẽ xem xét một ví dụ sử dụng Corba

Đầu tiên chúng ta sẽ định nghĩa IDL cho đối tượng từ xa.

```
interface Icalculator {

double TinhToan(in double so1, in double so2, in char pt);

};
```

Bây giờ chúng ta có thể biên dịch IDL dùng lệnh idlj. Tham số -fall phát sinh mã cả 2 phía máy khách và máy chủ:

```
idlj -fall Icalculator.idl
```

Sau khi biên dịch IDL, sẽ phát sinh một số tập tin sau:

IcalculatorStub. Java: là mã stub.

Icalculator.Java: là giao diện từ xa..

IcalculatorHolder.Java là mã hỗ trợ.

IcalculatorHelper.Java là mã hỗ trợ.

IcalculatorOperations.Java là mã hỗ trợ.

IcalculatorPOA.Java là mã skeleton và là lớp cha chúng ta thừa kế khi cài đặt mã bên phía máy chủ.

Viết đối tượng gọi từ xa.

Tiếp đến chúng ta sẽ cài đặt lớp bên phía máy chủ được gọi là clsRemote và thừa kế từ lớp gốc phát sinh khi dịch IDL:

```
class clsRemote extends _lctIRemoteObImplBase{
```

```
public clsRemoteO() {

}

public double TinhToan (double so1, double so2, char pt){

switch(pt){

case '+':

return so1 + so2;

case '-':

return so1 - so2;

case '*':

return so1 * so2;

case '/':

return so1 / so2;

case '%':

return so1 % so2;

}

return 0;

}

}
```

Chương trình bên phía máy chủ

Tiếp đến chúng ta cài đặt mã bên phía máy chủ, để thực hiện việc khởi tạo đối tượng và đăng ký máy chủ tên:

```
import org.omg.Corba.*;

import org.omg.CosNaming.*;
```

```
import org.omg.CosNaming.NamingContextExtPackage.*;

public class clsRegReO {

public static void main(String[] args) throws Exception{

System.out.println("Setup lctRemoteO");

ORB orb = ORB.init(args,null);

clsRemoteO rm = new clsRemoteO();

orb.connect(rm);//Ket noi O Corba voi moi truong ORB

//Lay dang ky den dich vu tnameserv

org.omg.Corba.Object thamChieSV = orb.resolve_initial_references("NameService");

//Chuyen kieu tham chieu den tnameserv|ve NamingContext

NamingContext nct = NamingContextExtHelper.narrow(thamChieSV);

//Tao ten cua doi tuong

NameComponent nc = new NameComponent("lctRO","");

//Tao duong dan de luu ten doi tuong

NameComponent path[] = {nc};

//Rang buoc doi tuong do vao tnameservices

nct.rebind(path,rm);

System.out.println("Server waiting....");

System.in.read();

}

}
```

Qua đây ta thấy Corba hơn hẳn RMI ở chỗ đối tượng RMI chỉ gọi được bởi các trình khách viết bằng Java còn các đối tượng trong Corba có thể gọi được bởi trình khách viết bằng các ngôn ngữ khác như C++, delphi…

Chương trình bên phía máy khách

Cuối cùng chúng ta tạo ra một chương trình bên phía máy khách để thực hiện lời gọi phương thức từ xa:

```
import org.omg.Corba.*;

import org.omg.CosNaming.*;

class clsRunClient {

public clsRunClient() {

}

public static void main(String[] args) throws Exception {

ORB orb = ORB.init(args, null);

//Tham chieu den dich vu tnameservices

org.omg.Corba.Object tnsv = orb.resolve_initial_references("NameService");

NamingContext nct = NamingContextExtHelper.narrow(tnsv);

NameComponent nc = new NameComponent("lctRO", "");

NameComponent path[] = {

nc};

lctIRemoteOb conS = lctIRemoteObHelper.narrow(nct.resolve(path));

double a = 13, b = 3;

System.out.println("Tổng hai số là: " + conS.TinhToan(a, b, '+'));

System.out.println("Hieu hai số là: " + conS.TinhToan(a, b, '-'));

System.out.println("Tich hai số là: " + conS.TinhToan(a, b, '*'));
```

System.out.println("Thuong hai số là: " + conS.TinhToan(a, b, '/'));

System.out.println("Mod hai số là: " + conS.TinhToan(a, b, '%'));

}

}

Chạy ứng dụng Corba

Để thực thi đoạn mã ví dụ, chúng ta phải khởi động RMI Registry, chạy đoạn mã bên phía máy chủ một lần, sau đó chúng ta có thể chạy chương trình khách:

idlj -fall -oldImplBase lctPhepToan.idl

Javac *.Java

start tnameserv

Java clsRegReO

pause

Kết quả in ra màn hình:

Tổng hai số là: 16

Hiệu hai số là: 10

Tích hai số là: 39

Thương hai số là: 4,333333

Mov hai số là: 1

Các cách khởi động trình môi giới ORB.

Trong ví dụ trên ta khởi động trình môi giới ở hai phía khách chủ bằng lệnh:

ORB orb = ORB.init(args, null);

Phương thức tĩnh init của ORB cần hai đối số:

- Đối số thứ nhất dùng để nhận các tuỳ chọn (Nếu có).

- Đối số thứ hai do người lập trình chỉ định.

Nếu cả hai đối số này cùng mang giá trị Null ORB sẽ dùng các thông số của hệ thống mặc định.

Trong trường hợp bạn không dùng các thông số mặc định của ORB bạn cần khởi tạo các đối số cho phương thức.ORB.init();

Tìm hiểu đối tượng Corba được cài đặt phía máy chủ

Dưới đây là nội dung tập tin khi bạn dùng lệnh :

idlj - fall - oldImplBase Example .idl

Ví dụ 4.7.1: _IcalculatorImplBase.Java

```
public abstract class _IcalculatorImplBase extends org.omg.Corba.portable.ObjectImpl

implements Icalculator, org.omg.Corba.portable.InvokeHandler

{

// Constructors

public _IcalculatorImplBase ()

{

}

private static java.util.Hashtable _methods = new java.util.Hashtable ();

static

{

_methods.put ("TinhToan", 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: // Icalculator/TinhToan

{

double so1 = in.read_double ();

double so2 = in.read_double ();

char pt = in.read_char ();

double $result = (double)0;

$result = this.TinhToan (so1, so2, pt);

out = $rh.createReply();

out.write_double ($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:Icalculator:1.0"};

public String[] _ids ()

{

return (String[])__ids.clone ();

}

} // class _IcalculatorImplBase
```

Ví dụ 4.7.2: IcalculatorStub.Java

```
public class _IcalculatorStub extends org.omg.Corba.portable.ObjectImpl implements Icalculator

{

public double TinhToan (double so1, double so2, char pt)

{

org.omg.Corba.portable.InputStream $in = null;

try {

org.omg.Corba.portable.OutputStream $out = _request ("TinhToan", true);

$out.write_double (so1);

$out.write_double (so2);

$out.write_char (pt);
```

```
$in = _invoke ($out);
double $result = $in.read_double ();
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 $rm) {
return TinhToan (so1, so2, pt );
} finally {
_releaseReply ($in);
}
} // TinhToan
// Type-specific Corba::Object operations
private static String[] __ids = {
"IDL:Icalculator:1.0"};
public String[] _ids ()
{
return (String[])__ids.clone ();
}
private void readObject (java.io.ObjectInputStream s) throws java.io.IOException
{
```

```
String str = s.readUTF ();

String[] args = null;

java.util.Properties props = null;

org.omg.Corba.Object obj = org.omg.Corba.ORB.init (args, props).string_to_object (str);

org.omg.Corba.portable.Delegate delegate = ((org.omg.Corba.portable.ObjectImpl) obj)._get_delegate ();

_set_delegate (delegate);

}

private void writeObject (java.io.ObjectOutputStream s) throws java.io.IOException

{

String[] args = null;

java.util.Properties props = null;

String str = org.omg.Corba.ORB.init (args, props).object_to_string (this);

s.writeUTF (str);

}

} // class _IcalculatorStub
```

Ví dụ 4.7.3: Icalculator

```
public interface Icalculator extends IcalculatorOperations, org.omg.Corba.Object, org.omg.Corba.portable.IDLEntity

{

} // interface Icalculator
```

Ví dụ 4.7.4:IcalculatorHelper

```
abstract public class IcalculatorHelper
```

```
{
private static String _id = "IDL:Icalculator:1.0";
public static void insert (org.omg.Corba.Any a, Icalculator 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 Icalculator 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 (IcalculatorHelper.id (),
"Icalculator");
}
return __typeCode;
}
```

```
public static String id ()

{

return _id;

}

public static Icalculator read (org.omg.Corba.portable.InputStream istream)

{

return narrow (istream.read_Object (_IcalculatorStub.class));

}

public static void write (org.omg.Corba.portable.OutputStream ostream, Icalculator
value)

{

ostream.write_Object ((org.omg.Corba.Object) value);

}

public static Icalculator narrow (org.omg.Corba.Object obj)

{

if (obj == null)

return null;

else if (obj instanceof Icalculator)

return (Icalculator)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 ();

_IcalculatorStub stub = new _IcalculatorStub ();

stub._set_delegate(delegate);

return stub;

}

}

}
```

Ví dụ 4.7.5: IcalculatorHolder.Java

```
public final class IcalculatorHolder implements org.omg.Corba.portable.Streamable

{

public Icalculator value = null;

public IcalculatorHolder ()

{

}

public IcalculatorHolder (Icalculator initialValue)

{

value = initialValue;

}

public void _read (org.omg.Corba.portable.InputStream i)

{

value = IcalculatorHelper.read (i);

}
```

```
public void _write (org.omg.Corba.portable.OutputStream o)
{
IcalculatorHelper.write (o, value);
}
public org.omg.Corba.TypeCode _type ()
{
return IcalculatorHelper.type ();
}
}
```

Ví dụ 4.7.6: IcalculatorOperations.Java

```
public interface IcalculatorOperations{
double TinhToan (double so1, double so2, char pt);
} // interface IcalculatorOperations
```

Các tập tin trung gian cài đặt phía máy khách

Dưới đây là nội dung tập tin khi bạn dùng lệnh :

idlj - fall - oldImplBase Example.idl

Ví dụ 4.8.1: IcalculatorOperations .Java

```
public interface IcalculatorOperations{
double TinhToan (double so1, double so2, char pt);
} // interface IcalculatorOperations
```

Ví dụ 4.8.2: IcalculatorHolder.Java

```
public final class IcalculatorHolder implements org.omg.Corba.portable.Streamable
```

```
{
public Icalculator value = null;
public IcalculatorHolder ()
{
}
public IcalculatorHolder (Icalculator initialValue)
{
value = initialValue;
}
public void _read (org.omg.Corba.portable.InputStream i)
{
value = IcalculatorHelper.read (i);
}
public void _write (org.omg.Corba.portable.OutputStream o)
{
IcalculatorHelper.write (o, value);
}
public org.omg.Corba.TypeCode _type ()
{
return IcalculatorHelper.type ();
}
}
```

Ví dụ 4.8.3: IcalculatorHelper.Java

```
abstract public class IcalculatorHelper

{

private static String _id = "IDL:Icalculator:1.0";

public static void insert (org.omg.Corba.Any a, Icalculator 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 Icalculator 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 (IcalculatorHelper.id (),
"Icalculator");

}
```

```
return __typeCode;

}

public static String id ()

{

return _id;

}

public static Icalculator read (org.omg.Corba.portable.InputStream istream)

{

return narrow (istream.read_Object (_IcalculatorStub.class));

}

public static void write (org.omg.Corba.portable.OutputStream ostream, Icalculator value)

{

ostream.write_Object ((org.omg.Corba.Object) value);

}

public static Icalculator narrow (org.omg.Corba.Object obj)

{

if (obj == null)

return null;

else if (obj instanceof Icalculator)

return (Icalculator)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 ();

_IcalculatorStub stub = new _IcalculatorStub ();

stub._set_delegate(delegate);

return stub;

}

}

}
```

Ví dụ 4.8.4: Icalculator.Java

```
public interface Icalculator extends IcalculatorOperations, org.omg.Corba.Object,
org.omg.Corba.portable.IDLEntity

{

} // interface Icalculator
```

Ví dụ 4.8.5:_IcalculatorStub .Java

```
public class _IcalculatorStub extends org.omg.Corba.portable.ObjectImpl implements
Icalculator

{

public double TinhToan (double so1, double so2, char pt)

{

org.omg.Corba.portable.InputStream $in = null;

try {

org.omg.Corba.portable.OutputStream $out = _request ("TinhToan", true);
```

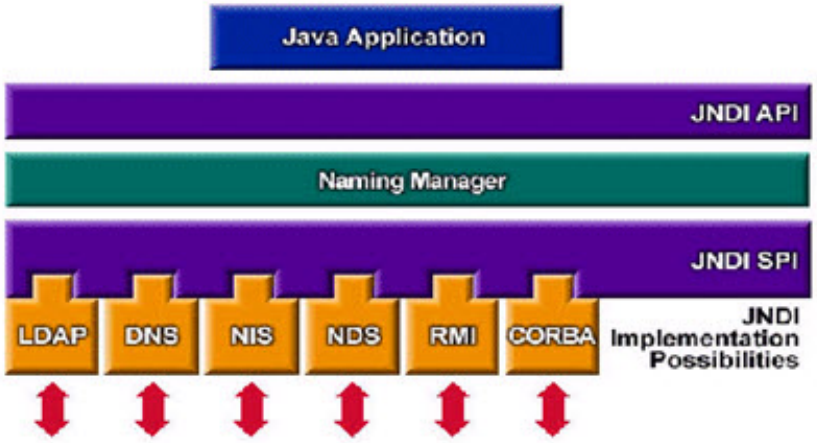
```
$out.write_double (so1);
$out.write_double (so2);
$out.write_char (pt);
$in = _invoke ($out);
double $result = $in.read_double ();
return $result;
} catch (org.omg.Corba.portable.ApplicationException $ex) {
$in = $ex.getInputStream ();
String _id = $ex.getId ();
throw new org.omg.Corba.MARSHAL (_id);
} catch (org.omg.Corba.portable.RemarshalException $rm) {
return TinhToan (so1, so2, pt );
} finally {
_releaseReply ($in);
}
} // TinhToan
// Type-specific Corba::Object operations
private static String[] __ids = {
"IDL:Icalculator:1.0"};
public String[] _ids ()
{
return (String[])__ids.clone ();
```

```
}

private void readObject (java.io.ObjectInputStream s) throws java.io.IOException

{

String str = s.readUTF ();

String[] args = null;

java.util.Properties props = null;

org.omg.Corba.Object obj = org.omg.Corba.ORB.init (args, props).string_to_object (str);

org.omg.Corba.portable.Delegate delegate = ((org.omg.Corba.portable.ObjectImpl) obj)._get_delegate ();

_set_delegate (delegate);

}

private void writeObject (java.io.ObjectOutputStream s) throws java.io.IOException

{

String[] args = null;

java.util.Properties props = null;

String str = org.omg.Corba.ORB.init (args, props).object_to_string (this);

s.writeUTF (str);

}

} // class _IcalculatorStub
```

Tìm hiểu JNDI và dịch vụ quản lí trên CosNaming của Corba

JNDN và dịch vụ Naming của Corba

Kiến trúc JNDI cung cấp một hệ thống đặt tên chuẩn độc lập API được xây dựng bởi một điều khiển hệ thống đặt tên đặc biệt. Lớp này giúp ta cách ly ứng dụng từ nguồn dữ liệu thực tế, và bởi vậy nó không có ý nghĩa với các ứng dụng truy nhập một LDAP, RMI, DNS, hay dịch vụ khác. Nói cách khác, JNDI là độc lập (của) bất kỳ sự thi hành dịch vụ thư mục đặc biệt nào, và bất kỳ thư mục nào có thể được sử dụng miễn là bạn cung cấp dịch vụ (hay các điều khiển) cho nó như hình sau:



Cấu trúc của JNDI

Điều này là ghi nhớ quan trọng về JNDI, nó cung cấp cả một giao diện lập trình ứng dụng (API) lẫn một giao diện nhà cung cấp dịch vụ (SPI). Điều này thực sự có ý nghĩa – cho ứng dụng của bạn tương tác với naming hay một dịch vụ thư mục ở đó phải là một nhà cung cấp dịch vụ JNDI cho naming và dịch vụ thư mục đặc biệt – giống như trình điều khiển JDBC kế thừa giá trị của giao diện JDBC cho hệ thống dữ liệu đặc biệt. Như các ứng dụng đang phát triển, bạn không cần phải lo lắng về JNDI SPI. Bạn chỉ cần chắc chắn rằng bạn có một nhà cung cấp dịch vụ cho mỗi naming hay thư mục bạn muốn sử dụng.

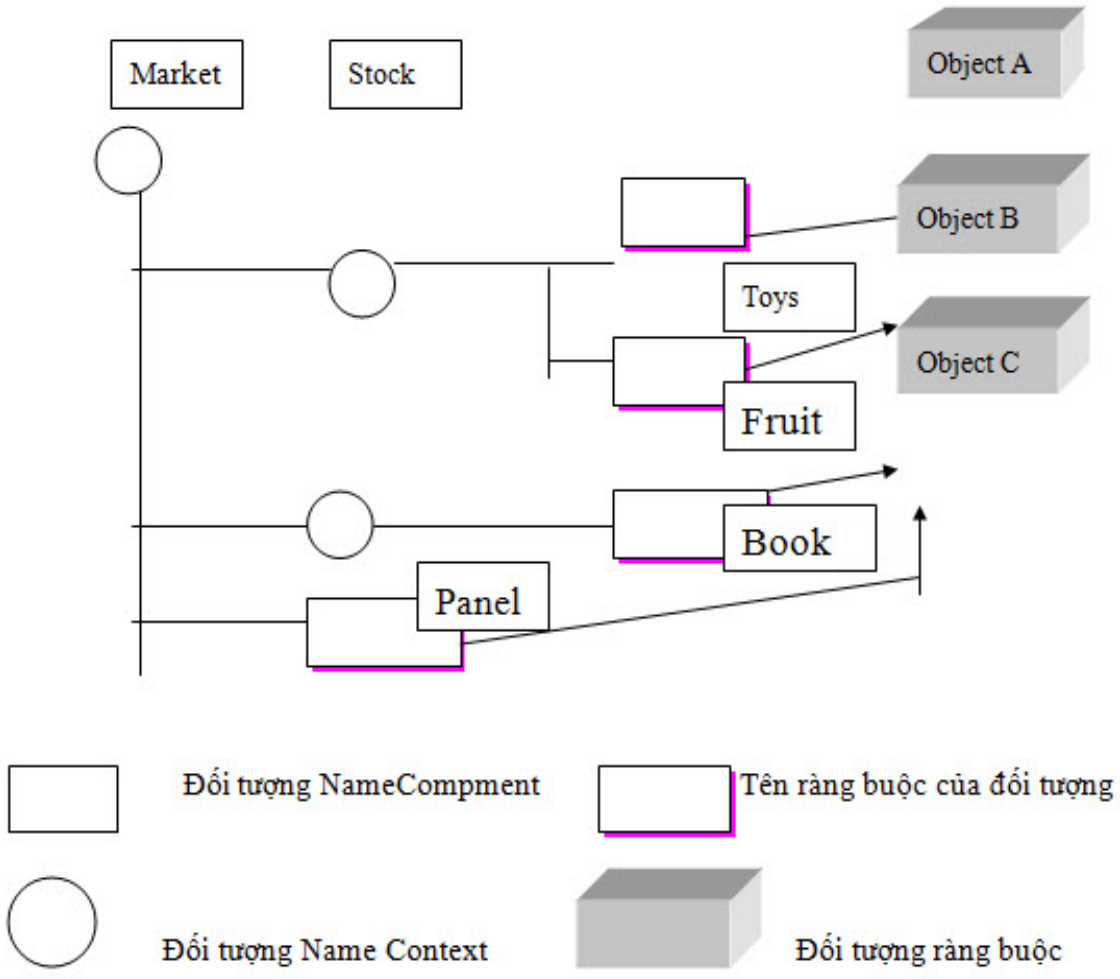
Cũng dựa vào đặc tả JNDI của Java nhưng Corba xây dựng một cơ chế lưu trữ tên và tham chiếu của đối tượng khác với dịch vụ Naming của RMI.

Corba phân ra khái niệm NamingContext và NamingCompoment 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 NamingCompoment hay NamingContext khác. NamingContext được đặt tên dựa vào NamingCompoment.

NamingCompoment 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 phải là đối tượng Corba)

NamingCompoment và NamingContext quan hệ với nhau theo cấu trúc hình cây tương tự cấu trúc cây thư mục của hệ thống tệp 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"



Mô hình ràng buộc và lưu trữ đối tượng của dịch vụ COS Naming

Như bạn thấy ở hình trên Market là một đối tượng NamingContext. Nó cho phép tìm ra 3 thành phần tên: stock, shelf và penel. Bản thân stock, shelf là hai NamingContext. Stock chứa hai 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.

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\sotck\Toys. Muốn tham chiếu đến Object C bạn chỉ đường dẫn Market\panel hoặc Market\shelf\book. Tuy nhiên bạn đang đứng ở vị trí Stock bạn chỉ cần tham chiếu đến Object A theo cách: stock\toys. Đó chính là ngữ cảnh (context) mà bạn sẽ gặp sau này trong dịch vụ của Corba. Với một đối tượng tuỳ theo mà ta có cách truy xuất theo tên của đối tượng khác nhau.

Các hàm cơ bản

- Tạo NamingContext:
    - new_context()
    - New_bind_context(NameComponent[] path)
- Hủy NamingContext:
    - Destroy()
- Lấy về đối tượng NamingContext:
    - resolve_initial_references(ObjectId id)
    - Resolve(NameComponent[] path)
- Tạo một bind
    - *Bind(NameComponent[] path, Object obj)*
    - *Bind_context(NameComponent[] path, NameContext nc)*
    - Rebind(NameComponent[] path, Object obj)
- Hủy một bind
    - *Unbind(NameComponent[] path)*
- Liệt kê các binding trong một context

List(NameContext nc, BindingList bl, BindingIterator bt)

Lưu địa chỉ tham chiếu của đối tượng Corba ở dạng chuỗi

Nếu bạn cảm thấy dich vụ COS Naming của Corba quá phức tạp thì vẫn còn cách truy xuất khác dễ dàng hơn mà không cần dung 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 trước. Chương trình Setup.java sau đây không dùng đến dịch vụ COS Naming của Corba mà vẫn có thể hoạt được.

Ví dụ 6.1: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 đến Corba với trình môi giới ORB

orb.connect(servant);

//Nhờ trình ORB chuyển tham chiếu của đói 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);

outputstream. 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 tệp tin t.tmp để lưu nội dung chuỗi xuống file.Bởi vì kiểu String mang đặc tính Serialiable(tuần tự hoá)nên nếu được khôi phạu 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 6.2 Client.Java

```
import org.omg.Corba.*;

import org.omg.CosNaming.*;

import java.io.*;
```

```
public class Client {

public static void main(String args []) thr ows 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”);

ObjectInp utStream instream=ne w ObjectInputSteam(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.C ORBA.Object o= orb.string_to_object(genericRef);

//Chuyển đổi kiểu đối tượng Corba trở về tham chiếu cụ thể

Generic generic= GenericHelper.narrow(0);

//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 COSNaming.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 phía 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 buộc 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 nêm mất hiệu lực.

Chú ý: *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).*

Tham chiếu ngược (Callback) của đối tượng Corba

RMI có khả 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 điều này.

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 gọi tham chiếu của AtClient đến trình chủ. AtServer sử dụng tham chiếu của AtClient để gọi ngược từ máy khách.

Bước 1:

Đặc tả đối tượng AtClient và AtServer bằng ngôn ngữ IDL được khai báo như sau:

Ví dụ 7.1:ClientServer.idl

```
Interface AtClient{

void callClientMethod(in string message);

};

Interface AtServer{

void registerClient(in AtClient c)

void callServerMethod(in string message);

};
```

Tập tin Client Server.idl đặc tả cả hai interface cho AtClient và AtServer. Bạn dùng trình biên dich idlj để dịch ClientServer.idl thành các tập tin Java phục vụ cho cả hai phía khách, chủ như sau:

```
idlj - fserver - oldImplBase ClientServer.idl

idlj - fclient - oldImplBase ClientServer.idl
```

Bước 2:

Cài đặt AtClient và AtServer. Giao diện (interface) AtClient được cài đặt thành đối tượng Corba AtClientServant còn giao diện AtServer được cài đặt thành đối tượng Corba AtServant như sau:

Ví dụ 7.2: AtClientServant.java

```
public class AtClientServant extends _AtClientImplBase{

public void callClientMethod(String message){

System.out.println(message);

}

}
```

Ví dụ 7.3: 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;

//Trieu goi phuong thuc cua doi tuong chay tren may khach

client.callClientMethod(msg);
```

```
}
}
}
```

Bước 3:

Xây dựng trình chủ dùng cài đặt và đăng ký đối tượng AtServer trên máy chủ

Ví dụ 7.4: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 Client");
//Tao doi tuong AtClient
ORB Orb=ORB.init(args,null);
AtServerServant servant= new AtServerServant();
//ket noi AtClient voi trình môi gioi
Orb.connect(servant);
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);
```

```
//Tao doi tuong Java

java.lang.Object obj=new java.lang.Object();

//VOng lap vo tận chờ nhân yêu cầu của máy khách

synchronized (obj){

obj.wait();

}

}

}
```

Bước 4:

Cài đặt trình khách,triệu gọi đối tượng AtServer trên máy chủ

```
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");

//Tao doi tuong AtClient

ORB Orb=ORB.init(args,null);

AtClientServant clientservant= new AtClientServant();

//ket noi AtClient voi trình môi gioi

Orb.connect(clientservant);

/*Yêu cau dich vu COS Naming truy tìm tham chieu cua AtServer thông qua tên g\ u1ECDi” 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));
Serverservant.registerClient(clientservant);
//Trieu goi doi tuong chu
Serverservant.callServerMethod("Client request");
}
}
```

Bước 5:

Biên dịch và chạy chương trình

Với chương trình bên server

javac *.java

start tnameserv

java Setup

Với chương trình bên Client

javac *.java

start tnameserv

java Client

Gọi các phương thức động (Dynamic Method Call) của đối tượng Corba

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 trong đốitượng Corba.Cơ chế này giúp bạn có thể gọi phương thức của một đối tượng Corba bấtkỳ mà không cần dùng đến lớp giao tiếp interface do trình chủ cung cấp. Điều này tương tự cơ chế gọi phương thức của những đối tượng RMI.Sau đây là 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().Chương trình gồmcác bước sau:

Bước 1:Đặc tả đối tượng Friend bằng ngôn ngữ IDL

Ví dụ: Friend.idl

Interface Friend{

String greeting (in string name);

};

Tiếp đến ta dùng trình dịch idlj để chuyển đổi đặc tả thành các tập tin cần dungcho cả hai phía khách chủ.

idl- fall Friend.idl

Lưu ý :Thay cho tham số dòng lệnh- fserver và –fclient hoặc ta dùng –fall để yêucầu trình dịch 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: FriendServant.Java

public class FriendServant extends_FriendImplBase{

public String greeting(String name){

return ”Hello”+ name+ ”from Corba!”;

```
}
}
```

Bước 3: Thiết kế chương trình dăng ký đối tượng Friend với dịch vụ COS Naming

Ví dụ: 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);
//Cho nhan yeu cau tu may khach
```

```
java.lang.Object obj=new java.lang.Object();

synchronized(obj){

obj.wait();

}

}

}
```

Như chúng ta đã thấy việ cài đặt và đăng ký đối tượng Corba không có gìđặc biệt.Nó hoàn toàn giống như những ví dụ trên Nhưng 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 : 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);

NamingComponent nc=new NameComponent("FriendObject","");

NameComponent path[]={nc};

//Chuyen tham chieu cua doi tuong Corba thanh doi tuong Java
```

```
java.lang.Object o=(java.lang.Object)nscontext.resolve(path);
//Truy tim xuat xu cua lop doi tuong
Class c=o.getClass();
//Dinh kieu phuong thuc cua tham so can goi
Class []parameterTypes=new Class[]{String.class};
//Truy tìm phương thức cần gọi
Method theMethod;
theMethod=c.getMethod("greeting",parametterType);
//Dinh doi so truyen vao phuong thuc can goi
java.lang.Object[] arguments=new jva.lang.Object[] {"Jerry"};
//Goi phuong thuc va nhan ket qua tra ve
String result=(String) theMethod.invoke(o,arguments);
System.out.println (result) ;
}
}
```

Bước 5: Biên dịch và chạy chương trình

Với chương trình bên server

javac *.java

start tnameserv

java Setup

Với chương trình bên Client

javac *.java

start tnameserv

java Client

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 phươngthức mà không cần đến lớp tham chiếu tường minh của đối tượng. Ví dụ trên được viết bằng kỹ thuật reflect trong Java sẽ được viết lại theo phong cách của Corba như sau:

Ví dụ: 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 NamingComponent("FriendObject","");

NameComponent path[]={nc};

//Lay ve tham chieu tong quat cua doi tuong Corba

Org.omg.Corba.Object o=nsContext.resolve(path);

//Yeu cau Corba truy tìm và tra ve tham chieu cua phuong thuc greeting
```

trong doi tuong Corba

Org.omg.Corba.Request request=o._request("greeting”);

//Tao doi tuong truyen vao ham greeting

Request.add_in_arg().insert_string("Jerry");

//Dinh kieu tra ve cua phương thức greeting

Request.set_return_type(orb.get_primitive_tc(org.omg.Corba.TCKind.

tk_string));

//Trieu goi phuong thuc greeting

Request.invoke();

//Lay ket qua tra ve sua khi goi ham

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.Request request=o._request(“greeting”);

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_retủn_type(ỏb.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 tham chiếu đến. Khi bạn triệu gọi phương thức, giá 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_long…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. Kết quả kết xuất hoàn toàn tương tự như ví dụ trên.

RMI trên IIOP (RMI-OVER-IIOP)

Một số khái niệm cơ bản

Common Object Request Broker Architecture (Corba) là đặc tả của Object Management Group - OMG giành cho việc đạt được tính tương tác giữa các nút tính toán phân tán. Mục tiêu của chúng đã định nghĩa một cấu trúc mà có thể cho phép những môi trường không đồng nhất có thể liên lạc tại mức đối tượng, không quan tâm đến người nào đã thiết kế ra hai điểm cuối của ứng dụng phân tán. Corba là một ngôn ngữ trung lập được thực thi rộng rãi hơn trên các nền platform khác nhau hơn là COM. Nhưng có vài tính không tương thích giữa những thực thi của các nhà cung cấp khác nhau, Corba sử dụng một giao thức gọi là IIOP (Internet Inter-ORB Protocol) để liên lạc giữa các hệ thống khác nhau.

Triệu gọi phương thức từ xa (RMI) và kiến trúc môi giới yêu cầu đối tượng chung (Corba) là tầng giữa hỗ trợ việc gọi phương thức phân tán. Tuy nhiên RMI trong Java có nhiều điểm khác và và hạn chế so với Corba.

Sự khác nhau của Corba và RMI trong Java thể hiện ở các điểm sau:

- RMI là một phần của bộ J2SDK và là hàm thư viện hỗ trợ các lời gọi phương thức từ xa và trả về giá trị cho các ứng dụng tính toán phân tán. Chúng ta giả sử rằng ngôn ngữ Java được sử dụng ở cả hai phía gọi và phía bên phương thức được gọi (hay gọi chung là ngôn ngữ thuần Java).

- Corba là một chuẩn công nghiệp cho phép gọi các phương thức từ xa và nhận kết quả trả về nhưng không giống như RMI mà nó có thể được sử dụng khi bên phía gọi và bên phía phương thức được gọi có thế sử dụng các ngôn ngữ lập trình khác nhau, bao gồm cả trường hợp là cả hai bên đều không sử dụng ngôn ngữ java.

- RMI là một tập hàm các thư viện đơn giản vì cả hai bên đều sử dụng cùng một ngôn ngữ lập trình và kiến trúc máy. Điều này sẽ làm cho vấn đề triệu gọi phương thức từ xa dễ giải quyết hơn.

- Như đã đề cập ở trước thì Corba định nghĩa nhiều dịch vụ, một trong những dịch vụ đó thực hiện chức năng tương tự như RMI, nhưng Corba thì hoạt động với nhiều ngôn ngữ lập trình khác nhau và không chỉ với nền java.

- Nền của Java chứa một Corba ORB, Corba dùng cùng kỹ thuật Stub/Skeleton như RMI nhưng không giống với RMI mà Corba phát sinh stub và skeleton từ một mô tả giao diện độc lập với ngôn ngữ được gọi là ngôn ngữ "Mô tả giao diện" (Interfaca Description

Language- IDL) thay vì mã nguồn của ngôn ngữ IDL xác định tên phương thức, cũng như tham số gọi và trả về theo một kiểu ngôn ngữ trung lập.

+ 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 lại sử dụng giao thức IIOP được chấp nhận và chuẩn hoá của hầu hết các đối tượng lớn.

Tại sao lại phải sử dụng RMI trên IIOP

Như trên chúng ta đã biết qua các khái niệm cơ bản về IIOP và RMI, sự khác nhau cơ bản giữa RMI với Corba….Nhưng nhìn chung RMI và Corba chưa có tiếng nói chung đồng nhất. Vì lý do này mà Java cung cấp cho ta cách cài đặt để các đối tượng RMI của Java có thể sử dụng giao thức IIOP để giao tiếp với các đối tượng của Corba. Khả năng này được gọi là RMI trên IIOP (RMI-Over IIOP). Một kỹ thuật RMI với cách cài đặt RMI-Over-IIOP sẽ vừa có thể giao tiếp được với các trình khách RMI vừa có thể giao tiếp dược với các đối tượng Corba.

Từ sơ đồ hình 6 ta thấy:

-Nếu trình khách viết bằng Java thì:

+ Có thể giao tiếp được với đối tượng chủ RMI theo giao thức thuần Java (JRMI) hoặc giao thức RMI-Over-IIOP

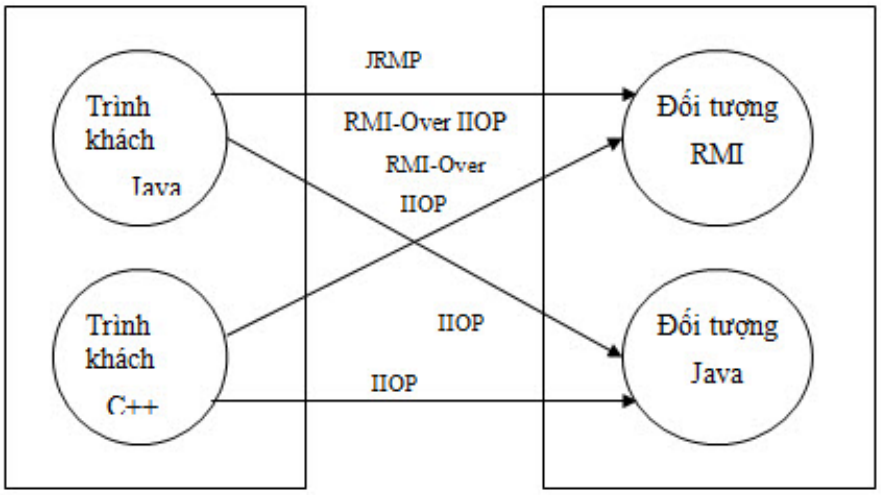
+Hay có thế giao tiếp được với đối tượng Corba theo giao thức IIOP

- Nếu trình khách viết bằng C++ thì:

+Có thể dùng giao thức IIOP để triệu gọi đối tượng Corba

+Hoặc chỉ có thể giao tiếp được với đối tượng RMI thông qua giao thức RMI-Over-IIOP mà thôi.

- Cả hai đối tượng RMI và Corba có thể giao tiếp được với nhau dựa trên giao thức RMI-Over-IIOP.



Mô tả các khả năng giao tiếp bằng giao thức RMI-Ovẻ IIOP so với các giao thức JRMP và IIOP giữa các đối tượng.

Xây dựng đối tượng RMI giao tiếp bằng IIOP

Với RMI-IIOP, stub và skeleton được phát sinh ngay từ định nghĩa đối tượng java. Thay vì sử dụng giao thức trong RMI để liên lạc giữa 2 tiến trình, RMI-IIOP sử dụng giao thức Corba IIOP để nó có thể gọi các đối tượng không được viết bằng ngôn ngữ java. Các bước phát triển bắt đầu với cài đặt một lớp phát sinh bằng IDL trong ngôn ngữ java. Lớp này được biên dịch với trình tiện ích rmic dùng cờ hiệu –iiop –d và một lớp bên phía máy chủ sử dụng COSNaming, truy cập JNDI (Java Naming and Directory Interface – JNDI).

Cài đặt và thực thi RMI-Over-IIOP

Các bước để tạo và biên dịch một chương trình RMI-Over-IIOP

Bước 1: Đặc tả giao tiếp Interface cho đối tượng RMIOjects

Ví dụ: RMIObject.Java

```
package rmi;

import java.rmi.*;

public interface RMIObject extends Remote {

public String tellAbout () throws RemoteException;

}
```

Bước 2: Cài đặt đối tượng RMIObject

Ví dụ: RMIObjectIpml.Java

```
package rmi;

import java.rmi.*;

public class RMIObjectImp1 implements RMIObject{

public String tellAboult() throws RemoteException{

System.out.print(“client asking”);

Return “ Toi la RMI day”;

}

}

}
```

Đối tượng của ta cung cấp một phương thức tellAboult () để “xưng danh” với trình khách. Như vậy bạn đã có một đối tượng RMI thuần nhất viết toàn bằng ngôn ngữ java. Bạn biên dịch đoạn chương trình này như sau:

```
javac *.java;
```

Với lớp đối tượng RMIObjectImpl.class thu được ta muốn đối tượng phải có khả năng giao tiếp bằng giao tiếp IIOP thay cho giao thức JRMI mà Java quy định đối với RMI. Điều này có thể thực hiện dễ dàng bởi trình dịch rmic.exe với thông số -iipp như sau

```
rmic.exe –iiop RMIObjectImpl
```

Kết quả bạn sẽ thu được hai tập tin trung gian là: RMIObjecStub.Java và RMIObjectTie.java. Đây là hai lớp chịu trách nhiệm chuyển đổi lời gọi giao thức JRMI thành giao thức IIOP và ngược lại.

Trình cài đặt cho đối tượng RMI “giả” đối tượng Corba được viết như sau:

Ví dụ :

```
package rmi;
```

```
import javax.rmi.*;

import javax.naming.*;

public class Setup{

public static void main(String [] args) throws Exception{

//Tạo thể hiện của đối tượng Java

RMIObject obj=new RMIObject();

// yeu cau may ao Java nhan dang obj la mot doi tuong cua Corba

PortableRemoteObject.exportObject(obj);

System.out.println("Binding object…..");

// lay ve dich vu Context cua dih vu dang ky ten

Context ctx=new InitialContext();

/* Rang buoc ten cua doi tuong voi dich vu quan ly tin.Sau loi goi bind()

Chuong rinh se roi vao trang thai cho cac y/cau tu may khach

*/

ctx.bind("My RMI as Corba",obj);

System.out.println("client yeu cau cho doi ");

}

}
```

Khi viết chương trình cài đặt,có một số điểm mà bạn cần chú ý như sau:

Để chuyển đối tượng RMI thành đối tượng có khả năng sử dụng giao thức IIOP chúng ta cần:

1. Thay thư viện *import java.naming*;*

import java.rmi.;*

bằng thư viện:

import javax.naming.;*

import javax.rmi.;*

2. Thay vì gọi *UniscastRemoteObject.exportObject()* để thông báo cho máy ảo Java biết sự hiện diện của đối tượng RMI thì chúng ta gọi:

PortableRemoteObject.exportObject(obj);

Lệnh trên sẽ thông báo tới máy ảo Java biết rằng đối tượng obj có thể giao tiếp từ xa bằng giao thức IIOP của Corba thay cho JRMI trong RMI.

Các đối tượng Corba cần đăng ký với dịch vụ quản lý tên tnameserv (COS naming) trước khi bị truy xuất bởi trình khách. Trong Corba để đăng ký dịch vụ COS Naming thì bạn 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 dich vụ COS Naming trên thư viện javax.naming cung cấp cho bạn đối tượng 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 COS Naming như sau:

ctx.bind ("My RMI as Corba", obj);

Để chỉ định đối tượng context gốc lấy về là đối tượng của dịch vụ COS Naming bạn chạy trình Setup từ dòng lệnh sau:

java -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory -Djava.naming.provider.url=iiop://localhost:900

- Chú ý:

1.Java sẽ sử dụng dịch vụ lớp khởi tạo com.sun.CNCtxFactory làm dịch vụ đăng ký tên

2.Tham số java.naming.provider.url chỉ định vị trí mà dịch vụ quản lý tên đang sử dụng (trước khi chạy trình Setup bạn nhớ khởi động tnameServerv ở một cửa sổ DOS-Prompt khác với cổng mặc định của tnameserverv là 900).

Như vậy là trình Setup đã biên dịch thành công đối tượng RMIObject có khả năng giao tiếp được với trình khách bằng giao thức IIOP của Corba.

Bước 3:Viết trình khách truy xuất đối tượng RMI theo giao thức IIOP:

Trình khách 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 có thể tạo ra các lớp trung gian. Và theo đó trình dịch rmic.exe sẽ giúp chuyển interface của Java sang đặc tả .idl của Corba. Chúng ta sẽ gọi chương trình dịch rmi.exe với tham số dòng lệnh .idl và yêu cầu rmic.exe diễn dịch lớp RMIObject từ Java sang IDL như sau: rmic.exe –idl RMIObject

Chú ý: Ở các phiên bản cũ của Java bạn có thể sử dụng trình diễn dịch Java2idl.exe thay cho lệnh .idl.

Sau khi trình biên dịch chạy xong ta thu được tập tin RMIObjec.idl như VD sau:

Ví dụ : RMIObject.idl

```
/* RMIObject.idl

Generated by rmic-idl.Do not edit

Friday,Ju 7, 2008 8:30:00 PM PDT

*/

#include “orb.idl”

#ifndef__RMIObject__

#define__RMIObject__

Interface RMIObject{

::Corba::WstringValue tellAbout();

};

#pragma ID RMIObject “RMI:RMIObject:0000000000000000”
```

#endif

Ta thấy sau khi trình biên dịch thực hiện xong ta có phương thức tellAbout() trả về kiểu chuỗi viết bằng ngôn ngữ của Corba. Chúng ta sẽ gọi trình jidl để sinh ra các tập tin trung gian với trình khách theo cách sau:

idlj –fclient RMIObject.idl

Và sau đây mà chương trình demo cài đặt cho trình khách:

Ví dụ : Client.Java

```
import org.omg.Corba.*;

import org.omg.CosNaming.*;

public class Client{

public static void main(String [] args) throws Exception{

//khoi dong trinh moi gioi trung gian

ORB orb=ORB.init (args,null);

//tim tham chieu cua nut context goc

org.omg.Corba.Object nameService=

orb.resolve_initial_references (“NameService”);

//lay ve tham chieu cua doi tuong

NamingContext nsContext=

NamingContextHelper.narrow (nameService);

NameComponent c=new NameComponent (“My RMI as Corba”,” ”);

Namecomponent path[]={nc};

RMIObjectservant= RMIObjectHelper.narrow

(nsContext.resolve(path));
```

```
//Trieu goi doi tuong

System.out.println ( servant.tellAbout() );

}

}
```

Biên dịch và chạy chương trình khách như sau:

Biên dịch:

javac *.java

Chạy chương trình:

java Client

Và kết quả thu được trên màn hình là

Toi la RMI day

Chương trình Client.Java trên đây hoàn toàn là một trình khách Corba bình thường.

Ngoài ra chúng ta có thể sử dụng đối tượng Context để viết trình khách theo phong cách RMI như ví dụ 6 dưới đây:

Ví dụ : ClientRMI.Java

```
import Javax.rmi.*;

import Javax.naming.*;

public class ClientRMI {

public static void main (String [] args) throws Exception {

Context cxt=new InitialContext ();

RMIObject servant = (RMIObject) PortableRemoteObject.narrow (

Cxt.lookup (“ My RMI as Corba ”), RMIObject.class );

System.out.println (servant.tellAbout() );
```

}

}

Và cũng tương tự như trình chủ để chạy được trình khách viết theo phong cách RMI truy tìm được dịch vụ COS Naming ta phải gọi máy ảo Java với tham số dòng lệnh tương tự như sau:

-Djava.naming.factory.initial=

com.sun.jndi.cosnaming.CNCtxFactory

-Djava.naming.provider.url=iiop://localhost:900

Và chúng ta cũng thu được kết quả tương tự như trên ví dụ 5 là

Toi la RMI day

Khởi tạo các thông số ngữ cảnh

Khi xây dựng chương trình khách và chỉ dùng đến đối tượng Context như ví dụ 3 và ví dụ 5, 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 đối tượng Context cho dòng lệnh chúng ta có thể thực hiện ngay các thiết lập này ngay trong mã lệnh. Điều này có thể làm dễ dàng như trong ví dụ dưới đây:

Ví dụ : ClientRMIConfig.Java

import java.until.*;

import Javax.naming.*;

import Javax.rmi.*;

public class ClientRMIconfig{

public static void main(String [] args)throws Exception {

//phan thiet lap khoi tao cho context ngay trong ma lenh

Hashtable hat=new Hashtable();

hat.put (“ java.naming.corba.applet”,this);

```
hat.put(“java.naming.factory.initial”,

”com.sun.jndi.cosnaming.CNCtxFactory”);

hat.put(“java.provider.url”,”iiop://localhost:900”);

Context cxt=new Initialcontext (hat);

RMIObject servant=(RMIObject) PortableRemoteObject.narrow

(cxt.lookup(“My RMI as Corba”),RMIObject.class);

System.out.println(servant.tellAbout() );

}

}
```

Như bạn đã thấy chúng ta đã sử dụng bảng hat kiểu Hashtable để lưu các giá trị thiết lập cho Context. Thay cho lời gọi khởi tạo InitialContext() với giá trị rỗng,đối tượng Context được gọi khởi động với thông số thiết lập từ bảng hat như sau:

```
Context cxt=new Initialcontext(hat);
```

Bằng cách này trình khách có thể chỉ cần gọi đơn giản từ dòng lệnh :

java ClientRMIConfig

Corba và trình môi giới VISIBROKER

Tổ chức OMG chỉ đưa ra các đặc tả tổng quát để xây dựng mô hình đối tượng phân tán để triệu gọi lẫn nhau. Nhưng trên thực tế thì để ứng dụng và thực thi thì các đối tượng đó lại không thể giao tiếp với nhau. Chính vì lẽ đó mà họ đã đưa ra trình môi giới trung gian để các đối tượng có thể giao tiếp với nhau một cách dễ dàng. Trình môi giới trung gian đó có tên gọi là ORB. Nhờ trình môi giới trung gian này mà các đối tượng Corba có thể trao đổi được với nhau theo giao thức IIOP.

Visibroker là một trình môi giới trung gian do hãng Borland cài đặt, nó hoàn toàn miễn phí và hỗ trợ java. Bạncó thể download miễn phí tạ địa chỉ: http://www.borland.com/visibroker.

Giới thiệu Visibroker của Borland

Borland là hãng phát triển công cụ lập trình rất nổi tiếng. Bạn đã từng gặp Borland qua các sản phẩm của ngôn ngữ lập trình như: C++, Pascal, Jbuider… Hơn ai hết chính Borland cũng đã tìm tòi và đưa ra cách tích hợp các ngôn ngữ với nhau. Và Corba ra đời cũng vì lẽ đó. Visibroker là một giải pháp dành cho những người lập trình theo hướng Corba. Visibroker là trình môi giới trung gian ORBcos khả năng giao tiếp đươc với các loại ngôn ngữ lập trình khác nhau.

Khi cài đặt đối tượng Corba trong Visibroker, về nguyên tắc không khác gì khi cài đặt đối tượng Corba. Bạn có thể tham khảo tại địa chỉ: http://www.borland.com/visibroker.

Xây dựng đối tượng chủ Corba

Cách xây dựng đối tượng chủ Corba sẽ dược làm sảng tỏ qua ví dụ mnh hoạ giao dịch trong ngân hàng thông qua đối tượng Corba. Đối tượng Account dùng để chứa thong tin của khách hàng. Còn đối tượng AccountManager dùng để quản lý và cấp phát đối tượng Account cho những khách hang nào muốn mở tài khoản.

Ví dụ : Đặc tả đối tượng Corba bằng ngôn ngữ idl

```
module Bank{

interface Account {

float balance();

};
```

```
interface AccountManage{

Account open (in String name);

};
```

Để thực thi được VD10.1 bạn cần tạo ra một thư mục C:\Corba\BankApp. Sau đó lưu tập tin Bank.idl này vào thư mục vừa tạo. Biên dịch Blank.idl sang Java như sau:

```
C:\Corba\BlankApp > Idl2Java Blank.idl
```

Chương trình idl2java.exe nằm trong thư mục cài đặt Visibroker\Bin. Kết quả bạn sẽ thấy tạo ra các tập tin sau trong thư mục con Bank:

Đối tượng Account:

Acount.Java

AccountHelper.Java

AccountHolder.Java

AccountPOA.Java

AccountOperation.Java

AccountPOATile.Java

Và đối tượng AccountManager

AccountManager.Java

AccountManagerHelper.Java

AccountManagerHolder.Java

AccountManagerOperations.Java

AccountManagerPOA.Java

AccountManagerPOATile.Java

Tổng quát tương tự bạn thu được các tập tin sau:

<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 thì ta thu được lớp <interfacename>_POA.java. Còn lớp <interfacename>_POATile.Java đóng vai trò là lớp trung gian giao tiếp giữa đối tượng và ORB.

Để cài đặt cho đối tượng Corba trong Visibroker bạn thừa kế lớp <interfacename>_POA như Ví dụ 8 sau:

Ví dụ: AccountImpl.Java:

```
// cai dat doi tuong Account

public class AccountImpl extends Bank.AccountPOA {

private float_balance;

public AccountImpl(fload balance){

_balance=balance;

public float balance(){

return _balance;

}

}

}
```

Đối tượng Account cung cấp cho ta phương thức balnace() trả về cân đối tài khoản khách hàng.Account đực tạo ra bởi phương thức open() của AccountManager như ví dụ 9 sau:

Ví dụ: AccountManagerImpl.Java

```
import org.omg.PortableServer.*;
import java.until.*;
public class AccountManagerImpl extends Bank.AccountManagerPOA
{
//bang chua cac tai khoan can quan ly
private Dictionary _accounts = new Hashtable();
//doi tuong tao so ngau nhien
private Random _random = new Random();
//phuong thuc nay duoc trieu goi de mo tai khoan cho khach hang mang ten name
public Bank.Account open(String name) {
//xem tai khoan da co hay chua
Bank.Account account = (Bank.Account) _accounts.get(name);
//Neu chua thi tao tai khoan moi
if(account == null) {
//Tao so tien ban dau cho tai khoan
float balance = Math.abs(_random.nextInt()) % 100000;
//tao doi tuong tai khoan moi
AccountImpl accountServant = new AccountImpl(balance);
try {
//chuyen doi tuong ve kieu Account
account = Bank.AccountHelper.narrow(
_default_POA().servant_to_reference(accountServant));
```

```
} catch (Exception e) { e.printStackTrace() ; }

//tai khaon duoc tao thanh cong

System.out.println("Created " + name + "'s account: " + account);

//luu tai hoan vao danh sach

_accounts.put(name, account);

}

//tra ve tai khoan cho khach hang

return account;

}

}
```

Để hiểu rõ hơn chúng ta cùng đi xây dựng chương trình Server và Client dùng để đăng ký đối tượng AccountManager với trình ORB của Visibroker được thực hiện như hai ví dụ dưới đây:

Ví dụ: Server.Java

```
import org.omg.PortableServer.*;

public class Server {

public static void main(String[] args) {

try {

//khoi tao trinh ORB

org.omg.Corba.ORB orb = org.omg.Corba.ORB.init(args,null);

//Lay ve tham chieu goc cua doi tuong ORB

POA rootPOA = POAHelper.narrow(

orb.resolve_initial_references("RootPOA"));
```

```
//Tao chinh sach quy dinh doi tuong hoat dong
org.omg.Corba.Policy[] policies = { rootPOA.create_lifespan_policy(
LifespanPolicyValue.PERSISTENT)
};
//Tao doi tuong MyPOA moi them vao doi tuong POA goc
POA myPOA = rootPOA.create_POA( "bank_agent_poa",
rootPOA.the_POAManager(), policies );
//Tao doi tuong phuc vu servant
AccountManagerImpl managerServant = new AccountManagerImpl();
//Chi dinh ten cho doi tuong
byte[] managerId = "BankManager".getBytes();
//Kich hoat doi tuong chua trong POA
myPOA.activate_object_with_id(managerId,managerServant);
//Kich hoat bo quan ly POA
rootPOA.the_POAManager().activate();
System.out.println( myPOA.servant_to_reference(managerServant) +
" is ready.");
// Cho yeu cau tu may khach
orb.run();
}
catch (Exception e) { e.printStackTrace(); }
}
```

}

Chương trình xây dựng cho phía Client

Ví dụ: Client.Java

```
public class Client {

public static void main(String[] args) {

//Khoi dong trinh trung gian ORB

org.omg.Corba.ORB orb = org.omg.Corba.ORB.init(args,null);

//Dinh ten cua doi tuong Corba can goi

byte[] managerId = "BankManager".getBytes();

//Truy tim tham chieu cua doi tuong

Bank.AccountManager manager =

Bank.AccountManagerHelper.bind(orb,

"/bank_agent_poa",managerId);

//Su dung doi so args[] lam ten tai khoan.Neu args[0] khong co ten tai khoan thi duoc dat la khongco

String name = args.length > 0 ? args[0] : "khongco";

//Yeu cau doi tuong AccountManager mo tai khoan

Bank.Account account = manager.open(name);

//Lay ve so can doi tai khoan

float balance = account.balance();

//Ket qua

System.out.println("The balance in " + name

+ "'s account is $" + balance);
```

}

}

Biên dịch và triệu gọi đối tượng Corba trong Visibroker

Để biên dịch chương trình chúng ta vừa xây dựng ở trên ta lưu tất cả vào một thư mục C:\corba\BlankApp. Visibroker sử dụng trình vbjc.exe để biên dịch và trình vbj.exe để thực thi chương trình.

Để biên dịch mã nguồn:

C:\corba\BlankApp > vbjc *.Java

Khởi động trình Visibroker ORB mang tên Visibroker Smart Agent. Vào thư mục C:\Visibroker\bin\ và 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 chủ và máy khách ta có:

Chạy chương trình đăng AccountManager như sau:

C:\corba\BlankApp >vbj Server

Chạy trình khách:

C:\corba\BlankApp > vbj Client

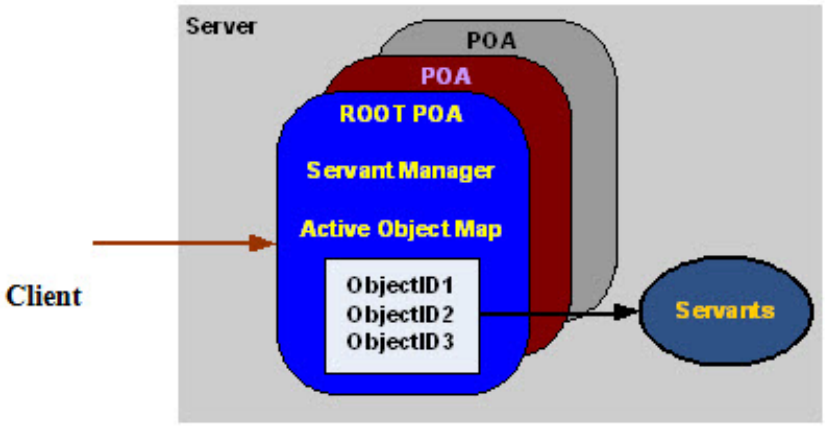
Và kết quả trả ra màn hình là :

The balance in khongco’s account is $779.56

Các đặc tả mới của Corba và Visibroker

Các đặc tả mới nhất của Corba là muốn xây dựng mô hình khả chuyển Portable cho các khái niệm bằng POA (Portable Object Adaptor)

POA là đối tượng chịu trách nhiệm quản lý và kích hoạt các đối tượng theo yêu cầu của trình khách. ORB thì cung cấp cho ta rất nhiều POA. Mỗi POA đều chứa các POA con được sinh 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. Đối tượng POA được quản lý theo định danh ID chứa trong danh sách kích hoạt. Xem hình dưới:



Cơ chế quản lý đối tượng Corba theo đặc tả mới

Khi trình khách yêu cầu gọi một đối tượng từ POA, lớp POA sẽ kiểm tra xem định danh ID của đối tượng cần gọi đã có trong bảng kích hoạt hay chưa. Nếu có rồi POA sẽ trả về tham chiếu của đối tượng trình khách triệu gọi. Còn trường hợp chưa có thì POA sẽ yêu cầu bộ quản ly đối tượng (Servant Manager) kiến tạo đối tượng để POA quản lý.

Các chính sách (policy) đăng ký phía máy chủ

Trình tự của quá trình đăng ký máy chủ AccountManager diễn ra như sau:

Để sử dụng lời gọi đến POA chúng ta cần liên kết với gói PortableServer bằng lệnh:

import org.omg.PortableServer.*;

Công việc đầu tiên trong trình tự của quá trình đăng ký máy chủ là ta tham chiếu đến thư mục gốc cũng như chúng ta đã làm với context của Corba.

//Lay ve tham chieu goc cua doi tuong ContextPOA

POA rootPOA = POAHelper.narrow (

Orb.resolve_initial_referentces (“RootPOA)) ;

//Tao doi tuong My POA mơi tham gia vao doi tuong POA goc

POA myPOA=rootPOA..create_POA (“ blank_agent_poa”,

rootPOA .the_POAManager() , policies) ;

Mỗi POA cần một chính sách để hoạt động (như thời hạn cho phép tồn tại của đối tượng…..). Các chính sách này được tạo ra nư sau:

```
//Tao chinh sách cac doi tuong hoat dong trong POA

org.omg.Corba.Policy [] policies = { rootPOA.create_lifespan_polyci

(LifespanPolicyValue.RERSISTENT) ; } ;

/* Doi tuong AccountManager duoc tao ra va dua vao trang thai cua

POA nhu sau */

//Tao doi tuong phuc vu servant

AccountManagerImpl managerSevant=

new AccoutManagerImpl ();

//Chi dinh ten ID cho doi tuong

byte [] managerID = “BlankManager” .getByte();

//Kich hoat doi tuong trong POA

myPOA.activate_object_with_id (managerId,managerServant);
```

Khi bạn kích hoạt một đối tượng trong POA thì đối tượng ấy chưa thực sự hoạt động.Muốn đối tượng ấy có thê triệu gọi được trình khách thì ta cần kích hoạt các POA như sau:

```
//Kich hoatbo quan ly POA

rootPOA.the_POAManager () .activate);

//Yeu cau Orb lang nghe loi trieu goi tu may khach

orb.run();
```

Gọi đối tượng máy chủ phía máy khách

Quá trình triệu gọi đối tượng chủ (sau khi đã đưng ký với trình Visibroker Smart Agent như trên) từ phía máy khách bạn chỉ cần khởi động trình ORB như sau:

orm.omg.Corba.ORB orb=org.omg.Corba.ORB.init(args,null);

Tiếp theo là lấy về các lớp trợ giúp do (idl2java sinh ra) và nhận về tham chiếu của đối tượng chủ như sau:

byte [] managerID = BlankManager.getByte();

//Truy tim tham chieu cua doi tuong

Bank.AccountManager manager= Bank.AccountManagerHelper.bind

(orb,"/bank_agent_poa",managerId);

Lệnh bind() ở đây tương tự như lệnh Naming.lookup() của RMI hay context.resolve() của dịch vụ Cosnaming.

Chương 5: Lập trình phân tán REMOTING trong .NET

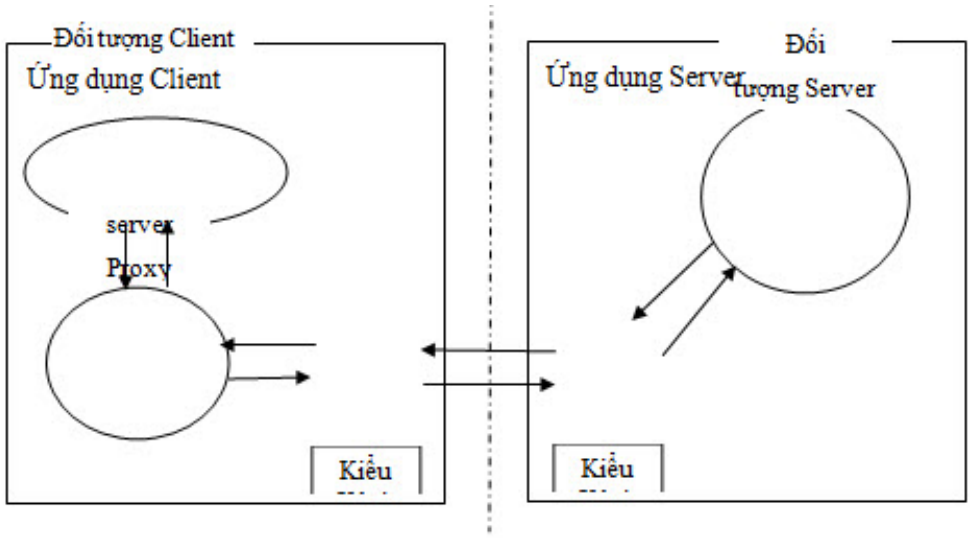
Lập trình phân tán REMOTING trong .NET

Tổng quan về Remoting

Remoting là gì?

Là một công nghệ lập trình phân tán của .Net, nó tương tự như công nghệ lập trình phân tán RMI của java. Nó cho phép máy khách triệu gọi phương thức của đối tượng từ xa.

serverProxyĐốitượng ServerĐối tượngClientCách thức hoạt động



-Client và Server cùng có giao diện của Remoting.

-Server triển khai đối tượng, đăng ký với Remoting channel services.

-Client kết nối tới remoting services để lấy tham chiếu đối tượng Remote

-Client trao đổi với server qua proxy class(interface) và Remoting services.

.**Net framework thì cung cấp rất nhiều những dịch vụ cái mà có sử dụng Remoting**.

- Những kênh truyền thông có nghĩa vụ truyền tải những thông báo qua lại giữa ứng dụng từ xa.

- Định dạng ghi thành mã và đọc ra thông báo trước khi chúng được chuyển tới bằng các kênh.
- Sự uỷ nhiệm gọi phương thức tới đối tượng thích hợp.
- Kích hoạt đối tượng từ xa và sự hỗ trợ trong suốt quá trình tồn tại.

Kênh và các định dạng kênh

- Kênh chuyền tải thông báo tới bằng đối tượng từ xa.
- Trước khi gọi một đối tượng từ xa, client đăng ký cái kênh đó trên mỗi ứng dụng cơ sở.
- Một máy tính không thể có nhiều hơn một kênh để lắng nghe từ một cổng.

.Net cung cấp kênh thực thi HTTP và TCP

- Kênh mặc định HTTP:Chuyển dữ liệu qua mạng theo định dạng XML bằng giao thức SOAP.
- Kênh mặc định TCP:Chuyển dữ liệu qua mạng theo dạng nhị phân bằng giao thức TCP.

VD:Chương trình đã đăng ký kênh TCP với cổng 8085.

using System;

using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Tcp;

TcpChannel chan=new TcpChannel(8985);

ChannelServices.RegisterChannel(chan);

- Kênh được sử dụng để vận chuyển những thông báo tới bằng đối tượng triệu gọi từ xa.Khi một Client gọi phương thức trên đối tượng từ xa, các thông số, giống như chi tiết khác cái có liên quan tới tất cả thì truyền qua kênh tới đối tượng từ xa. Một vài kết quả trả từ về theo con đường mà client đã gọi.
- Định dạng thì đã được ghi thanh mã và gửi dữ liệu vào những thông báo trước khi chúng được truyền lên trên Kênh.
    - Việc lựa chọn Kênh:

Bởi vì client có thể đăng ký một trong số nhiều Kênh trên server để giao tiếp với các đối tượng ở xa, bạn có thể lựa chọn Kênh phù hợp với cái bạn cần. Bạn cũng có thể sử dụng cái kênh đã có hoặc xây dụng một kênh sử dụng giao thức truyền thông khác.

- **Lựa chọn kênh theo tiêu chuẩn sau.**

- Kênh buộc phải đăng ký trước đối tượng đăng ký.Tối thiểu một kênh phải đăng ký với các bộ phận, thiết bị trước đối tượng từ xa có thể gọi đến.

- Kênh phải đăng ký trên nền tảng mỗi ứng dụng riêng. Một quá trình đơn giản có thể chứa nhiều ứng dụng.Khi một quá trình đã chắn gửi đi hoàn tất , toàn bộ kênh đã ghi lại và tự động huỷ đi.

- Nó thì không phù hợp khi đăng ký một kênh để lắng nghe trên cổng đã được đăng ký bởi một cái khác

Mặc dù kênh đã được đăng ký trên mỗi ứng dụng cơ sở, khác với ứng dụng khác trên một máy tính không thể đăng ký những kênh lắng nghe như những cái cổng.

Client có thể truyền đạt với các đối tượng từ xa bằng việc sử dụng kênh đã đăng ký. Cái khuôn mẫu từ xa đảm bảo cho những đối tượng từ xa kết nối phù hợp với kênh khi client thử kết nối với nó.Client có trách nhiệm chỉ rõ kênh trước khi kết nối và truyền đạt với đối tượng từ xa.

Để chỉ rõ kênh, bạn có thể sử dụng file cấu hình điều khiển từ xa của .Net hoặc bạn có thể gọi phương thức của kênh đã đăng ký trên lớp của kênh phục vụ.

Net framework cung cấp sự hỗ trợ cho kênh HTTP, TCP và SMTP.Bởi vì kênh điều khiển từ xa của .Net thì lặp đi lặp lại bạn có thể và lặp lại trong kênh thêm vào sự truyền tải duy nhất và ghi thành mã yêu cầu.

- Định dạng được sử dụng để ghi thành mã và đọc các thông báo.

Mỗi kênh cung cấp một định dạng mặc định riêng. Tuy nhiên bạn có thể chỉ rõ định dạng bạn muốn sử dụng.

Kênh HTTP:

Kênh HTTP sử dụng giao thức SOAP để truyền thông tin qua đối tượng từ xa.Toàn bộ những thông báo thì được truyền qua định dạng của SOAP. Ở nơi nào đó thông báo được thay đổi và chuyển thành XML và xuất đi theo định kỳ. sau đó yêu cầu gắn tiêu đề của SOAP phải có thêm một luồng.

Như một sự lựa chọn, bạn có thể chỉ rõ định dạng nhị phân, kết quả thì nằm trong luồng dữ liệu nhị phân.Trong một trường hợp khác, luồng dữ liệu được chuyển tới đích mà không thay đổi định dạng bằng cách sử dụng giao thức HTTP. Bạn có thể tạo ra một kỹ thuật chuẩn cho XML trong Web services bằng việc sử dụng kênh HTTP với định dạng SOAP mặc định.

Kênh TCP:

Từ các mặc định, kênh TCP sử dụng định dạng nhị phân và xuất tất cả những thông tin thành luồng nhị phân và chuyển luồng đó tới đích không đổi bằng cách sử dụng giao thức TCP.

Như một sự lựa chọn, bạn có thể chỉ rõ định dạng SOAP, với kết quả trong luồng dữ liệu XML. Bạn có thể thực hiện được tốt hơn bằng cách sử dụng kênh TCP với đing dạng nhị phân bạn có thể sử dụng XML web services.

VD:Một đoạn chương trình thể hiện việc đăng ký một kênh HTTP trên cổng 9999 bằng cách sử dụng Channelservices.RegisterChannel;

```
using System;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
//...
  TcpChannel chan = new TcpChannel(8085);
  ChannelServices.RegisterChannel(chan);
//...
```

- Chúng ta nên sử dụng kênh HTTP vì dữ liệu chuyển qua là XML không phụ thuộc vào hệ nền, không phụ thuộc vào ngôn ngữ lập trình, còn dữ liệu ở dạng nhị phân thì tuỳ từng dot net mà nó quy dịnh ta sẽ không rõ được.
- Dữ liệu định dạng bằng SOAP sẽ lớn hơn nên chuyển qua mạng sẽ chậm hơn

III.Kích hoạt

- Trước khi sử dụng đối tượng điều khiển từ xa thì trước tiên client buộc phải kích hoạt nó.
    - Bằng cách gọi mới:Activator.CreateInstance hoặc Activator.GetObject.
- Kích hoạt sử dụng trở lại lớp proxy bởi client để truy cập đối tượng điều khiển từ xa.
- Proxy miêu tả đối tượng từ xa trong ứng dụng cơ sở của client.
- Proxy sẽ xúc tiến việc gọi của client và trả lại kết quả và ngoại lệ.
- Kích hoạt ở phía server:Tự động thuyết minh bởi server

-Ở phía server hỗ trợ phương thức kích hoạt cả Single call và singleton .

- Single call: đối tượng chỉ sử dụng để chia sẻ dữ liệu cho một client và có nhiều đối tượng được tạo ra. Single call là đối tượng rất hữu ích trong:

+Đưa lên phía trên đầu tạo ra đối tượng thì không có ý nghĩa.

+Đối tượng thì cấu hình trong cách đưa dữ liệu lên.

+Trạng thái thông tin thì thường không cần giữa các phương thức gọi.

->Bởi vì đối tượng single call không lưu trữ trạng thái của thông tin giữa các phương thức gọi, chúng có một vài lần đề cập tới nhưng không được chấp nhân.

- Singleton :sử dụng khi muốn chia sẻ dữ liệu cho nhiều client khác nhau.và có một đối tượng được tạo ra.Bởi vậy đối tượng singleton có thể lưu trữ trạng thái của thông tin giữa yêu cầu của client Đối tượng singleton thì rất hữu ích khi bạn muốn chia sẻ dữ liệu giữa client và các phương thức yêu cầu, và trong khi phía đầu của việc tạo ra và giữ cho đối tượng đó luôn tồn tại.

->Bởi vì đối tượng singleton cso thể giữ lại được trạng thái trong giai thời kỳ dài.

- Kích hoạt ở phía client: Kích hoạt đối tượng ở client là kích hoạt trên yêu cầu từ client. Đây là phương thức kích hoạt đối tương thì giống như thành phần COM kích hoạt. Thao tác kích hoạt truy cập thì cho phép:

+Khi một client yêu cầu đối tượng server, kích về yêu cầu thông báo thì gủi tới ứng dụng từ xa.

+Server sau khi tạo ra một trường của lớp yêu cầu và trả ra đối tượng cho ứng dụng client đó chính là dẫn chứng.

+Một proxy thì tạo ra ở phía client bằng việc sử dụng đối tượng.

- Trạng thái liên tục giữa phương thức gọi trong ví dụ của client.

-Remoting framework hỗ trợ cả về phía server và client kích hoạt đối tượng từ xa.

-Điều kiện kích hoạt ở phía server là khi một client cố gắng truy cập tới đối tượng, một đối tượng tự động thuyết minh ở trên server

-Phía client:theo một hướng khác, điều kiện để đối tượng đó nói ra trong câu trả lời để kích hoạt yêu cầu từ client.

bạn có thể lựa chọn một kiểu kích hoạt để điều khiển việc thuyết minh trong suốt vòng đời của đối tượng.

-Một client kích hoạt đối tượng cso thể lưu giứ trạng thái giữa phương thức gọi của riêng client.Dù thế nào thì trạng thái thông tin thì không chia sẻ cho nhiều đối tương client. Mỗi yêu cầu về đối tượng từ xa, ví dụ trả ra một proxy trường hợp độc lập của kiểu server.

-Một chức năng hữu ích của đối tượng kích hoạt client thì xây dựng căn cứ có thể hợp quy cách bằng một ứng dụng để xây dựng đối tượng trong ứng dụng từ xa.

- Phương thức kích hoạt của đối tượng từ xa.

-Trước khi client sủ dụng đối tượng từ xa, đối tượng từ xa cần phải được kích hoạt rồi và client phải chứa proxy để truy cập tới đối tượng từ xa.

-Bạn có thể kích hoạt một đối tượng từ xa bằng cách gọi mới hoặc bằng cách gọi phương thức ở dưới của lớp kích hoạt:

*Activator.CreateInstance

Sử dụng để tạo đối tượng làm ví dụ.

*Activator.GetObject

Hầu hết mọi người sử dụng để kết nối tới một đối tượng đang chạy trên danh nghĩa URI.

- Vai trò của proxies trong việc kích hoạt đối tượng từ xa.

-Khi một client kích hoạt đối tượng từ xa thì client đó phải chứa proxy trong cái lớp làm ví dụ trên server.Một vài sự tương tác với sự xuất hiện đối tượng điều khiển từ xa bởi sự chuyển qua proxy.Một đối tượng proxy hành động như một đại diện của đối tượng từ xa và đảm bảo gọi tất cả những cái tạo ra trên proxy để tiến tới đúng đối tượng làm ví dụ.Tất cả phương thức cái mà gọi trên proxy tì tự động chuyên tiếp tới class từ xa và một vài kết quả trả về cho client.

Từ cái nhìn tổng quát thì truy cập giống như truy cập tạo ra vùng gọi.Một vài cái ngoại lệ được ném ra từ đối tương từ xa .Bởi những ngoại lệ trả ra cho client .Một client có thể sử dụng khối try/catch ở những đoạn cần thiết

Các bước xây dựng Remoting

Chúng ta sẽ theo dõi một ví dụ đơn giản:Xây dựng chương trình đảo xâu sử dụng remoting.

Xây dựng giao diện dùng chung cho cả Server và client

-Ở đây chỉ là khai báo các phương thức sẽ được gọi từ xa.

-Dùng chung cho client và server.

VD: *Public Interface Ixau*

Function Daoxau(ByVal st As String) As String

End Interface

- Chỉ có phần thân không có phần mô tả.

-Dùng để triệu gọi các phương thức khai báo trong server.

-Biên dịch thành .DLL để các server và client tham chiếu đến.

Xây dựng phía Server.(gồm 2 công việc)

*Những thông tin yêu cầu khi bạn đăng ký đối tượng với Remoting framework.

-Tên của đối tượng từ xa.

-Đối tượng URI cái mà client sẽ dùng đối tượng đó.

-Từ hoạt động kích hoạt , cách thức đối tượng thì là cái yêu cầu.

-Phương thức đối tượng có thể sừ dụng là singleton hoặc single call

- Cài đặt các phương thức triệu gọi từ xa nghĩa là triển khai giao diện và kế thừa .

+Tất cả các đối tượng phải được đăng ký với Remoting framework trước khi client có thể truy cập chúng.

+Ghi đè từ các phương thức triệu gọi từ xa.

Class IxauIpm inherits MarshalByrefObject Implements Ixau.IxauIpml

'xây dựng hàm đảo xâu

Public function Daoxau(byte st as String) Implements Ixau.IxauIpm.Daoxau

Dim m() as char=st.ToCharArray()

Dim i as Integer =m.length-1

Dim rs as String= ""

Do while i>=0

rs+=m(i)

i=i-1

Loop

End Function

-Đăng ký một đối tượng chứa các phương thức gọi từ xa với một kênh(channel)

+ Kế thừa *MarshalByrefObjectv* cần Inport

- Add References System.Runtime.Remoting, Interface R;
- System.Runtime.Remoting;
- System.Runtime.Remoting.Chanels;
- *System.runtime.remoting.Channles.Http;*

+Việc kế thừa từ *inherits MarshalByrefObject* là bắt buộc

+Khi inport ta tham chiếu tới thư viện của .Net :System.Runtime.Remoting

Cách khởi động Server:

-Tạo một kênh Httpchannel hoặc TcpChannel

HttpChanel chan = new HttpserverChanel(1234);

+Lớp httpserverchannel :Dùng cho server

+Số hiệu cổng để lắng nghe và kết nối từ xa truyền vào kênh thông tin để giao tiếp với client, số hiệu cổng phải được chỉ rõ và phải là cố định.

-Đăng ký nó với Remoting Chanel Server:

ChanelServies.RegisterChanel(Chan,False);

+Lấy thông tin về kiểu của đối tượng chứa các phương thức triệu gọi từ xa

Dim tp As Type = GetType(trienkhai)

-Đăng ký Remote Object với Remoting server:

RemotingConfiguration.RegisterWellKnownServiceType(tp, "cal",

WellKnownObjectMode.Singleton)

Trong đó: +Tham số thứ nhât tp:Kiểu của đối tượng cho phép Remoting

+ Tham số thứ hai cal:Tên của đối tượng cho phép Remoting hay còn gọi là bí danh.

+Tham số thứ 3 kiểu trả lời client của server:có thể là Singleton hoặc single call

Xây dựng phía Client

-Ở phía máy khách : Tham chiêu tới Tệp .dll ta sẽ tham chiếu tới kênh để giao tiếp vơi server.

-Tương tự như server, Ta cũng tạo một Project mới thêm vào những tham chiếu tới giao diện đối tượng từ xa.

-Kế thừa MarshalByrefObject cần Imports:

- System.Runtime.Remoting;
- System.Runtime.Remoting.Chanels;
- System.Runtime.Remoting.Chanels.Http

+Khi inport ta tham chiếu tới thư viện của .Net: System.Runtime.Remoting

-Tạo ra một kênh tương ứng với kiểu kênh của server

Dim chan As New HttpClientChannel

+Lớp httpclientchannel:Dùng cho Client

+Ngược với server, client không cần chỉ rõ số hiệu cổng nghĩa là khởi tạo không tham số để nó tự chọn số hiệu cổng chưa được sử dụng.

-Khai báo một tham chiếu tới giao diện remote

Dim tp As Type = GetType(giaodien.giaodienDX)

Dim rs1 As giaodien.giaodienDX = CType(RemotingServices.Connect(tp, "http://127.0.0.1:1234/cal"), giaodien.giaodienDX)

-Triệu gọi các phương thức:

Console.WriteLine(rs1.Daoxau("XAU CAN DAO"))

Console.ReadLine()

- Tất cả các viêc triệu gọi, đăng ký vào Remoting Chanel Services đều cần bắt lỗi.
- Có thể đưa Remote Object vào Windows Services.

Chương 6: Lập trình phân tán với WEB SERVICE

Lập trình phân tán với WEB SERVICE

Tổng quan web service

Vai trò của web service

Web service ra đời đã mở ra một hướng mới cho việc phát triển các ứng dụng trên Internet. Web services tạm dịch là các dịch vụ web. Web services kết hợp sử dụng nhiều công nghệ khác nhau cho phép các ứng dụng được viết bằng các ngôn ngữ khác nhau, chạy trên các hệ nền khác nhau có thể trao đổi được với nhau thông qua môi trường mạng Internet. Tuy nhiên những công nghệ sử dụng ở đây không nhất thiết phải là những công nghệ mới. Đây là điểm khác biệt của web services so với các công nghệ khác, đó chính là khả năng kết hợp các công nghệ đã có như là XML, SOAP, WSDL, UDDI để tạo ra các service, đặc điểm này làm nổi bật vai trò của web services.**Đặc điểm web service**

- Web service cho phép client và server tương tác được với nhau mặc dù trong những môi trường khác nhau (ngôn ngữ lập trình khác nhau, hệ nền khác nhau).

- Web service được thiết kế mở và dựa vào các chuẩn: XML và HTTP là nền tảng kỹ thuật cho web service. Bởi vậy, chúng độc lập với ngôn ngữ lập trình và hệ nền .

- Web service rất linh động: Vì với UDDI và WSDL, thì việc mô tả và phát triển web service có thể được tự động hóa.

-Web service được xây dựng trên nền tảng những công nghệ đã được chấp nhận

-Web service có dạng module

- Web service có thể công bố (publish) và gọi thực hiện qua mạng.-…Ngày nay web services được sử dụng rất nhiều trong những lĩnh vực khác nhau của cuộc sống , như :

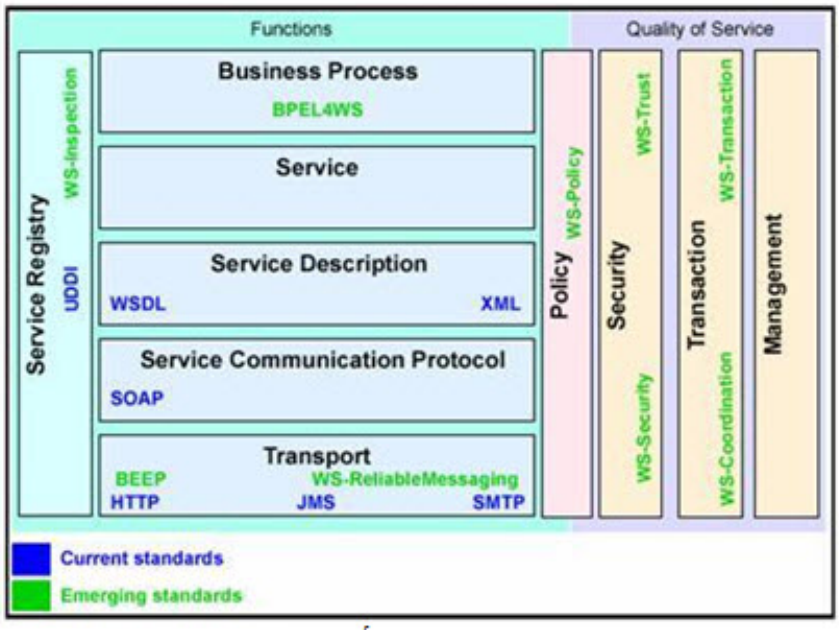
1. Dịch vụ chọn lọc và phân loại tin tức : là những hệ thống thư viện kết nối đến các web portal để tìm kiếm các thông tin từ các nhà xuất bản có chứa những từ khóa muốn tìm. Dịch vụ hiển thị danh sách đĩa nhạc dành cho các công ty thu thanh.Ứng dụng đại lý du lịch có nhiều giá vé đi du lịch khác nhau do có chọn

lựa phục vụ của nhiều hãng hàng không. Bảng tính toán chính sách bảo hiểm dùng công nghệ Excel/COM với giao diện web

2. Thông tin thương mại bao gồm nhiều nội dung, nhiều mục tin như: dự báo thời tiết, thông tin sức khoẻ, lịch bay, tỷ giá cổ phiếu,…
3. Những giao dịch trực tuyến cho cả B2B và B2C như: đặt vé máy bay, làm giao kèo thuê xe.
4. Hệ thống thông tin dùng Java để tính toán tỷ giá chuyển đổi giữa các loại tiền tệ. Hệ thống này sẽ được các ứng dụng khác dùng như một web service
5. …..

Kiến trúc web service

Kiến trúc của web service bao gồm các tầng như sau :



Kiến trúc web service

Trong đó bao gồm các tầng :

- Tầng vận chuyển với những công nghệ chuẩn là HTTP , SMTP và JMS- Tầng giao thức tương tác dịch vụ (Service Communication Protocol) với công nghệ chuẩn là SOAP . SOAP là giao thức nằm giữa tầng vận chuyển và tầng mô tả thông tin về dịch vụ, SOAP cho phép người dùng triệu gọi một service từ xa thông qua một message XML.

- Tầng mô tả dịch vụ (Service Description) với công nghệ chuẩn là WSDL và XML . WSDL là một ngôn ngữ mô tả giao tiếp và thực thi dựa trên XML. Web service sử dụng ngôn ngữ WSDL để truyền các tham số và các loại dữ liệu cho các thao tác , các chức năng mà web service cung cấp.

- Tầng dịch vụ (Service) :cung cấp các chức năng của service.- Tầng đăng ký dịch vụ (Service Registry) với công nghệ chuẩn là UDDI .UDDI dùng cho cả người dùng và SOAP server, nó cho phép đăng ký dịch vụ để người dùng có thể gọi thực hiện service từ xa qua mạng , hay nói cách khác một service cần phải được đăng ký để cho phép các client có thể gọi thực hiện

- Bên cạnh đó để cho các service có tính an toàn , toàn vẹn và bảo mật thông tin trong kiến trúc web service chúng ta có thêm các tầng Policy , Security , Transaction , Management giúp tăng cường tính bảo mật , an toàn và toàn vẹn thông tin khi sử dụng service.

Các thành phần trong web service

XML – Extensible Markup Language

XML do W3C đề ra và được phát triển từ SGML .XML là một ngôn ngữ mô tả văn bản với cấu trúc do người sử dụng định nghĩa .Về hình thức XML có ký pháp tựa như HTML nhưng không tuân theo một đặc tả quy ước như HTML.Người sử dụng hay các chương trình có thể quy ước định dạng các tag XML để giao tiếp với nhau.Thông tin cần truyền tải được chứa trong các tag XML,ngoài ra không chứa bất cứ thông tin nào khác về cách sử dụng hay hiển thị những thông tin ấy.

Do web service là sự kết hợp của nhiều thành phần khác nhau , do đó web services sử dụng các tính năng và đặc trưng của các thành phần này để giao tiếp với nhau.Vì vậy XML là một công cụ chính yếu để giải quyết vấn đề này .Từ kết qủa này , các ứng dụng tích hợp vĩ mô tăng cường sử dụng XML.Nhờ có khả năng tổng hợp này mà XML đã trở thành kiến trúc nền tảng cho việc xạy dựng web service.

Web services tận dụng khả năng giải quyết vấn đề của các ứng dụng lớn trên các hệ điều hành khác nhau cho chúng giao tiếp với nhau .Yêu cầu này được đáp ứng với lập trình với Java , một ngôn ngữ viết một lần sử dụng mọi nơi là một chọn lựa thích hợp cho phát triển web services

WSDL -Web Services Description Language

WSDL định nghĩa cách mô tả web service theo cú pháp tổng quát XML, bao gồm các thông tin

- Tên service

- Giao thức và kiểu mã hóa sẽ được sử dụng khi gọi các hàm của web service.- Loại thông tin: những thao tác, những tham số, và những kiểu dữ liệu gồm có giao diện của web service, cộng với tên cho giao diện này.

Một WSDL hợp lệ gồm có hai phần :

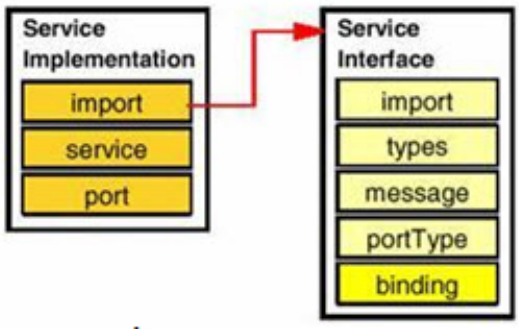
1. Phần giao diện mô tả giao diện và giao thức kết nối.

2. Phần thi hành mô tả thông tin để truy xuất service

Cả 2 phần trên sẽ được lưu trong 2 tập tin XML , bao gồm:

- Tập tin giao diện service (cho phần 1).

- Tập tin thi hành service (cho phần 2).



Cấu trúc WSDL

Tập tin giao diện - Service Interface

WSDL mô tả 5 loại thông tin chính bao gồm : import , types , message , portType , binding.

Types :WSDL định nghĩa các kiểu dữ liệu của thông điệp gửi.

<definitions .... >

<types><xsd:schema .... />*

</types></definitions>

Thông điệp (message) : mô tả thông điệp được gửi giữa client và server.

<definitions .... >

<message name="nmtoken"> *

<part name="nmtoken" element="qname"? type="qname"?/> *

</message></definitions>Những định nghĩa message được sử dụng bởi phần tử thi hành service . Nhiều thao tác có thể tham chiếu tới cùng định nghĩa message.

Thao tác và những message được mô hình riêng rẽ để hỗ trợ tính linh hoạt và đơn giản hóa việc tái sử dụng lại. Chẳng hạn, hai thao tác với cùng tham số có thể chia sẻ một định nghĩa message.

c.Kiểu cổng (port type):WSDL mô tả cách gửi và nhận thông điệp.<wsdl:definitions .... >

<wsdl: portType name="nmtoken">

<wsdl: operation name="nmtoken" .... /> *

</wsdl: portType>

</wsdl: definitions>

WSDL định nghĩa bốn kiểu thao tác mà một cổng có thể hỗ trợ:

- One-way : cổng nhận một message, message đó là message nhập.

- Request-response : cổng nhận một message và gửi một message phản hồi

- Solicit-response: cổng gửi một message và nhận về một message

- Notification: cổng gửi một message, message đó là message xuất.

Mỗi kiểu thao tác có cú pháp biến đổi tùy theo: thứ tự của các message nhập, xuất và lỗi.Ví dụ :

<wsdl:definitions .... >

<wsdl: portType .... > *

<wsdl: operation name="nmtoken" parameterOrder="nmtokens">

<wsdl: input name="nmtoken"? message="qname"/>

<wsdl: output name="nmtoken"? message="qname"/>

<wsdl:fault name="nmtoken" message="qname"/>*

</wsdl: operation>

</wsdl: portType >

</wsdl: definitions>

Kết hợp (Binding) :định nghĩa cách các web services kết hợp với nhau.

Một kết hợp bao gồm :

- Những giao thức mở rộng cho những giao tác và những message bao gồm thông tin URN và mã hóa cho SOAP.

- Mỗi một kết hợp tham chiếu đến một loại cổng; một kiểu cổng (portType) có thể được sử dụng trong nhiều mối kết hợp. Tất cả các thao tác định nghĩa bên trong kiểu cổng phải nằm trong phạm vi mối kết hợp.

Tập tin thi hành - Service Implementation

WSDL mô tả 2 loại thông tin chính bao gồm : service và port .

Dịch vụ (Service) : Nó sẽ thực hiện những gì đã được định nghĩa trong tập tin giao diện và cách gọi web services theo thủ tục và phương thức nào :

<wsdl:definitions .... >

<wsdl:service name="nmtoken">

<wsdl: port .... />

</wsdl: service>

</wsdl: definitions>

Port là một cổng đầu cuối, nó định nghĩa như một tập hợp của binding và một địa chỉ mạng.

<wsdl:definitions .... >

<wsdl: service .... >

<wsdl: port name="nmtoken" binding="qname">

</wsdl: port>

</wsdl: service>

</wsdl:definitions>Ở đây chúng ta thấy rằng thuộc tính kết hợp tên là qname. Nó tham chiếu tới một mối kết hợp. Một cổng chứa đựng chính xác một địa chỉ mạng; Bất kỳ cổng nào trong phần thi hành phải tương ứng chính xác với một tham chiếu trong phần giao diện.

WSDL API

WSDL4J là một dự án nguồn mở , hiện tại có một WSDL Java API được gọi là WSDL4J. Bộ WSDL4J cung cấp cho chúng ta các hàm API để thực hiện việc tạo WSDL dễ dàng hơn so với cách sử dụng trực tiếp cú pháp theo dạng tag như trên.Tên gói chứa các API này là javax.wsdl.

UDDI - Universal Description , Discovery and Intergration

Để có thể sữ dụng các dịch vụ , trước tiên client phải tìm dịch vụ , ghi nhận thông tin về cách sử dụng dịch vụ và biết được đối tượng cung cấp dịch vụ .UDDI định nghĩa một số thành phần cho biết trước các thông tin này để cho phép các client truy tìm và nhận lại những thông tin yêu cầu sử dụng web services.

Cấu trúc UDDI :

Cấu trúc UDDI gồm các thành phần :

- Trang trắng -White pages : chứa thông tin liên hệ và các định dạng chính yếu của web services , chẳng hạn tên giao dịch , địa chỉ ,… Những thông tin này cho phép các đối tượng khác xác định được service.

- Trang vàng -Yellow pages : chứa thông tin mô tả web services theo những chủng loại khác nhau .Những thông tin này cho phép các đối tượng thấy web services theo từng chủng loại của nó.

- Trang xanh -Green pages: chứa thông tin kỹ thuật mô tả các hành vi và các chức năng của web services .Các đối tượng dựa vào đặc điểm của web services để tìm kiếm.- Loại dịch vụ - tModel : chứa các thông tin về loại dịch vụ sử dụng.Những UDDI registry hiện có :

- UDDI Business Registry: bộ đăng ký được bảo trì bởi Microsoft, IBM đặc điểm của bộ đăng ký này là nó phân tán về mặt vật lý.

- IBM Test Registry: bộ đăng ký cho những người phát triển để thử nghiệm công nghệ và kiểm tra những service của họ.

- Private registries IBM ships: bộ đăng ký UDDI cá nhân.

SOAP - Simple Object Accesss Protocol

Đến đây chúng ta đã hiểu được web services là như thế nào , nó được công bố và truy xuất ở đâu.Nhưng chúng ta vẫn còn một vấn đề khá quan trọng đó là : làm thế nào chúng ta truy xuất dịch vụ khi tìm thấy ? Câu trả lời là web servicves có thể truy xuất bằng một giao thức là Simple Object Access Protocol – SOAP . Nói cách khác chúng ta có thể truy xuất đến UDDI registry bằng các lệnh gọi hoàn toàn theo kiểu SOAP.SOAP là một giao thức giao tiếp có cấu trúc như XML và mã hóa thành định dạng chung cho các ứng dụng trao đổi với nhau .Ý tưởng bắt đầu từ Microsoft và phần mềm Userland , trải qua nhiều lần thay đổi , hiện tại là phiên bản SOAP 1.2 với nhiều ưu điểm vượt trội hơn bản SOAP 1.1. SOAP được xem như là cấu trúc xương sống của các ứng dụng phân tán xây dựng từ nhiều ngôn ngữ , hệ điều hành khác nhau.Đặc trưng SOAP

SOAP có những đặc trưng sau :

- SOAP được thiết kế đơn giản và dễ mở rộng.

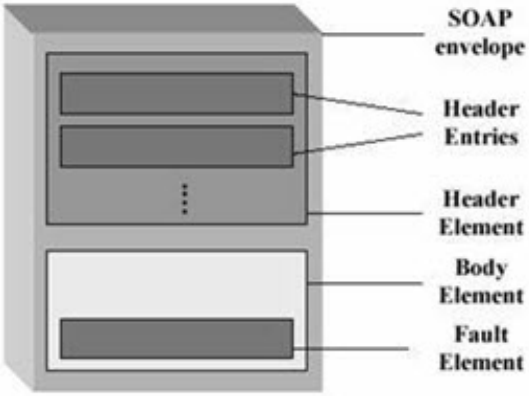
- Tất cả các message SOAP đều được mã hóa sử dụng XML.

- SOAP sử dùng giao thức truyền dữ liệu riêng.

- Không có garbage collection phân tán , và cũng không có cơ chế tham chiếu.Vì thế SOAP client không giữ bất kỳ một tham chiếu đầy đủ nào về các đối tượng ở xa.- SOAP không bị ràng buộc bởi bất kỳ ngôn ngữ lập trình nào hoặc công nghệ nào.Vì những đặc trưng này, nó không quan tâm đến công nghệ gì được sử dụng để thực hiện miễn là người dùng sử dụng các message theo định dạng XML. Tương tự, service có thể được thực hiện trong bất kỳ ngôn ngữ nào, miễn là nó có thể xử lý được những message theo định dạng XML.

Cấu trúc một message theo dạng SOAP

Cấu trúc một message theo dạng SOAP được mô tả như hình dưới đây:



Cấu trúc message SOAP

Message theo dạng SOAP là một văn bản XML bình thường bao gồm các phần tử sau:- Phần tử gốc - envelop : phần từ bao trùm nội dung message , khai báo văn bản XML như là một thông điệp SOAP.

- Phần tử đầu trang – header : chứa các thông tin tiêu đề cho trang , phần tử này không bắt buộc khai báo trong văn bản .Những đầu mục còn có thể mang những dữ liệu chứng thực, những chữ ký số hóa, và thông tin mã hóa, hoặc những cài đặt cho giao tác.

- Phần tử khai báo nội dung chính trong thông điệp - body , chứa các thông tin yêu cầu và phản hồi.

- Phần tử phát sinh lỗi (Fault) cung cấp thông tin lỗi xảy ra trong qúa trình xử lý thông điệp.

Trong trường hợp đơn giản nhất, phần thân của SOAP message gồm có:

- Tên của message

- Một tham khảo tới một thể hiện service.

- Một hoặc nhiều tham số mang các giá trị và mang các tham chiếu. Có 3 kiểu thông báo:

+ Request messages : với các tham số gọi thực thi một service

+ Response messages với các tham số trả về, được sử dụng khi đáp ứng yêu cầu.+ Fault messages báo tình trạng lỗi.

Những kiểu truyền thông

SOAP hỗ trợ hai kiểu truyền thông khác nhau :

- Remote procedure call (RPC): cho phép gọi hàm hoặc thủ tục qua mạng. Kiểu này được khai thác bởi nhiều web service và có nhiều trợ giúp.

- Document: được biết như kiểu hướng message : kiểu này cung cấp một lớp thấp của sự trừu tượng hóa, và yêu cầu người lập trình nhiều hơn khi làm việc.

Các định dạng message, tham số, và lời gọi đến các API thì tương ứng trong RPC và document là khác nhau. Nên việc quyết định chọn cái nào tùy thuộc vào thời gian xây dựng và sự phù hợp của service cần xây dựng.

Mô hình dữ liệu

Mục đích của mô hình dữ liệu SOAP là cung cấp những một sự trừu tượng hóa độc lập ngôn ngữ cho kiểu ngôn ngữ lập trình chung. Nó gồm có :

- Những kiểu XSD đơn giản như những kiểu dữ liệu cơ bản trong đa số các ngôn ngữ lập trình như int, string, date, …

- Những kiểu phức tạp, có hai loại là struct và array.

Tất cả các phần tử và những định danh có trong mô hình dữ liệu SOAP thì được định nghĩa bằng namespace SOAP-ENC

Mã hóa

Trong những môi trường tính toán phân tán, mã hóa định nghĩa làm sao giá trị của dữ liệu trong ứng dụng có thể được dịch từ khuôn dạng nghi thức. Khuôn dạng nghi thức cho những web service là XML, giả sử ở đây chúng ta giả thiết rằng service requestor và service provider phát triển trong Java. Vì vậy, mã hóa SOAP là trong môi trường thực thi để làm thế nào chuyển đổi từ cấu trúc dữ liệu Java sang SOAP XML và ngược lại. Một ánh xạ định nghĩa là mối quan hệ giữa một phần tử XML, một lớp Java, và một trong những loại mã hóa giới thiệu ở trên.

Một ánh xạ chỉ rõ làm cách nào, để khi đã mã hóa mà một phần tử XML đầu vào vẫn chuyển đổi được tới một lớp Java và ngược lại.Chúng ta quan tâm tới hai phương hướng ánh xạ là XML tới Java và Java tới XML. Bất kỳ môi trường thực thi SOAP nào cũng phải có một bảng chứa những mục ánh xạ, gọi là SOAPMappingRegistry.Nếu một kiểu dữ liệu được giả thiết sẽ được sử dụng dưới một loại mã hóa nhất định, thì một ánh xạ tương ứng phải tồn tại trong bộ đăng ký (registry) của môi trường thực thi SOAP đó. Đa số các kiểu Java chuẩn cũng như JavaBeans đều mặc định là được hỗ trợ.

Những kiểu dữ liệu không chuẩn (do tự định nghĩa) thì cần ánh xạ trên cả server và client.

Kết luận

Để tạo một web service chúng ta cần xây dựng các tầng cần thiết trong kiến trúc web service hay nói cách khác là xây dựng và thiết lập các thành phần trong các tầng đó , cụ thể là các thành phần SOAP , WSDL , UDDI , XML, trong đó:

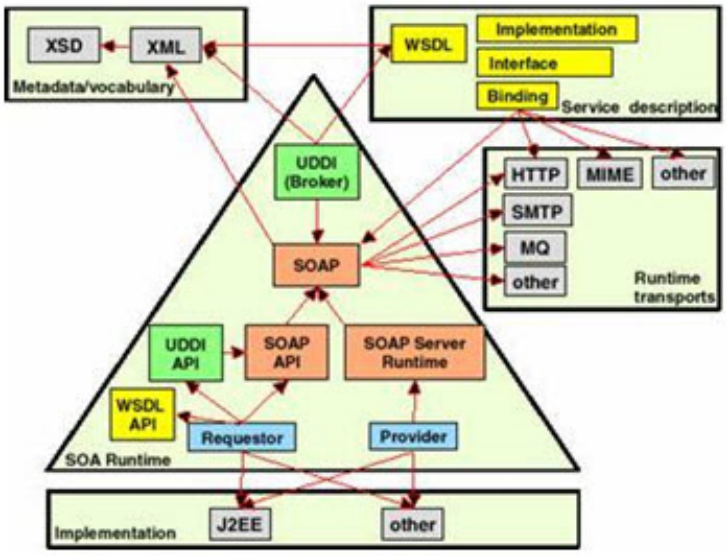
- SOAP là giao thức nằm giữa tầng vận chuyển và tầng mô tả thông tin về dịch vụ, SOAP cho phép người dùng triệu gọi một service từ xa thông qua một message XML.

- WSDL là một ngôn ngữ mô tả giao tiếp và thực thi dựa trên XML. Web service sử dụng ngôn ngữ WSDL để truyền các tham số và các loại dữ liệu cho các thao tác , các chức năng mà web service cung cấp.

- UDDI dùng cho cả người dùng và SOAP server, nó cho phép đăng ký dịch vụ để người dùng có thể gọi thực thi các hàm , các chức năng của web service hay nói cách khác một service cần phải được đăng ký để cho phép các client có thể gọi thực hiện

- Bên cạnh đó chúng ta cũng phải quan tâm đến việc làm sao để cho các service có tính an toàn , toàn vẹn và bảo mật thông tin trong web services nhất là các service liên quan đến giao dịch thương mại và tài chính.Chúng ta sẽ tìm hiểu nội dung này trong các phần tiếp theo.

Sơ đồ dưới đây cho chúng ta thấy rõ hơn về các thành phần cần thiết trong một web service và mối quan hệ giữa các thành phần



Các thành phần cần thiết trong một web service và mối quan hệ giữa các thành phần

Tham gia đóng góp

Tài liệu: Phát triển Hệ thống phân tán

Biên tập bởi: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://voer.edu.vn/c/8fddcc3d

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Các mô hình ứng dụng

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/95550ff5

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Lập trình phân tán với COM+/MTS

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/16bb8015

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: RMI và lập trình phân tán đối tượng

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/c8706f0b

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Cài đặt đối tượng phân tán

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/35adc9bb

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Chuyển tham số trong các lời gọi từ xa

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/aa76d1b2

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Tìm hiểu RMIREGISTRY và các cách đăng kí đối tượng

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/649350f2

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Dùng một đối tượng sản sinh nhiều đối tượng (FACTORY OBJECT)

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/e5fa1e4f

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Kỹ thuật gọi đối tượng ở xa bằng phương thức động(DYNAMIC METHOD INVOKE)

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/dececca0

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Tự động kích hoạt các đối tượng từ xa (ACTIVATION)

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/a427e4df

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Trình mồi nạp ứng dụng từ xa

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/0ba439b1

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Ứng dụng RMI

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/2359b5eb

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Giới thiệu về Corba

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/9779bdaf

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Ánh xạ giữa ngôn ngữ IDL và Java

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/7d046cb0

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Corba và mô hình phân tán.

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/2ef24490

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Xây dựng Ứng dụng Corba khách/chủ trong môi trường java

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/339ab96b

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Tìm hiểu JNDI và dịch vụ quản lí trên CosNaming của Corba

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/c91a184a

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Lưu địa chỉ tham chiếu của đối tượng Corba ở dạng chuỗi

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/b80b3f34

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Tham chiếu ngược (Callback) của đối tượng Corba

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/2f4bffb7

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Gọi các phương thức động (Dynamic Method Call) của đối tượng Corba

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/f95e261d

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: RMI trên IIOP (RMI-OVER-IIOP)

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/de7c7a01

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Corba và trình môi giới VISIBROKER

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/da4e6dde

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Lập trình phân tán REMOTING trong .NET

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/cd0fc9de

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Module: Lập trình phân tán với WEB SERVICE

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: http://www.voer.edu.vn/m/79c2292d

Giấy phép: http://creativecommons.org/licenses/by/3.0/

Chương trình Thư viện Học liệu Mở Việt Nam

Chương trình Thư viện Học liệu Mở Việt Nam (Vietnam Open Educational Resources – VOER) được hỗ trợ bởi Quỹ Việt Nam. Mục tiêu của chương trình là xây dựng kho Tài nguyên giáo dục Mở miễn phí của người Việt và cho người Việt, có nội dung phong phú. Các nội dung đều tuân thủ Giấy phép Creative Commons Attribution (CC-by) 4.0 do đó các nội dung đều có thể được sử dụng, tái sử dụng và truy nhập miễn phí trước hết trong trong môi trường giảng dạy, học tập và nghiên cứu sau đó cho toàn xã hội.

Với sự hỗ trợ của Quỹ Việt Nam, Thư viện Học liệu Mở Việt Nam (VOER) đã trở thành một cổng thông tin chính cho các sinh viên và giảng viên trong và ngoài Việt Nam. Mỗi ngày có hàng chục nghìn lượt truy cập VOER (www.voer.edu.vn) để nghiên cứu, học tập và tải tài liệu giảng dạy về. Với hàng chục nghìn module kiến thức từ hàng nghìn tác giả khác nhau đóng góp, Thư Viện Học liệu Mở Việt Nam là một kho tàng tài liệu khổng lồ, nội dung phong phú phục vụ cho tất cả các nhu cầu học tập, nghiên cứu của độc giả.

Nguồn tài liệu mở phong phú có trên VOER có được là do sự chia sẻ tự nguyện của các tác giả trong và ngoài nước. Quá trình chia sẻ tài liệu trên VOER trở lên dễ dàng như đếm 1, 2, 3 nhờ vào sức mạnh của nền tảng Hanoi Spring.

Hanoi Spring là một nền tảng công nghệ tiên tiến được thiết kế cho phép công chúng dễ dàng chia sẻ tài liệu giảng dạy, học tập cũng như chủ động phát triển chương trình giảng dạy dựa trên khái niệm về học liệu mở (OCW) và tài nguyên giáo dục mở (OER) . Khái niệm chia sẻ tri thức có tính cách mạng đã được khởi xướng và phát triển tiên phong bởi Đại học MIT và Đại học Rice Hoa Kỳ trong vòng một thập kỷ qua. Kể từ đó, phong trào Tài nguyên Giáo dục Mở đã phát triển nhanh chóng, được UNESCO hỗ trợ và được chấp nhận như một chương trình chính thức ở nhiều nước trên thế giới.