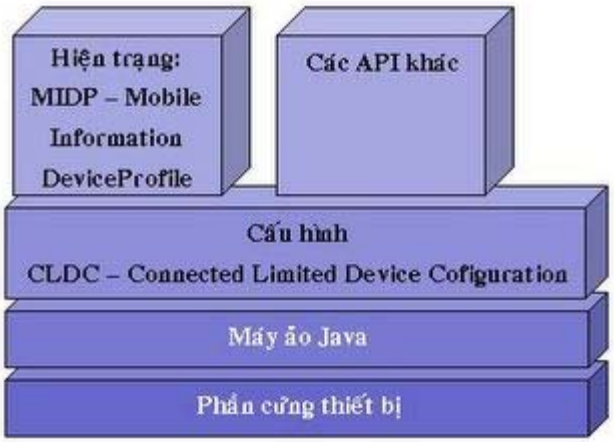
Lời giới thiệu:
Công nghệ Java cho công nghiệp di động (Java Technology Wireless Industry - JTWI) ngày càng phát triển và thu hút sự quan tâm của nhiều người. Nhằm đáp ứng nhu cầu này, TinCNTT mở chuyên mục J2ME Tutorial cố gắng đề cập đầy đủ nhiều khía cạnh của công nghệ Java cho di động. Để bắt đầu loạt bài, chúng ta sẽ cùng khảo sát các lớp và khái niệm quan trọng của J2ME.

Bài 1: Khái quát các lớp J2ME

Mục tiêu của J2ME là cho phép người lập trình viết các ứng dụng độc lập với thiết bị di động, không cần quan tâm đến phần cứng thật sự. Để đạt được mục tiêu này, J2ME được xây dựng bằng các tầng (layer) khác nhau để giấu đi việc thực hiện phần cứng khỏi nhà phát triển. Sau đây là các tầng của J2ME được xây dựng trên CLDC:



Hình 1. Các tầng của CLDC J2ME

Mỗi tầng ở trên tầng hardware là tầng trừu tượng hơn cung cấp cho lập trình viên nhiều giao diện lập trình ứng dụng (API-Application Program Interface) thân thiện hơn.

Từ dưới lên trên:

Tầng phần cứng thiết bị (Device Hardware Layer)

Đây chính là thiết bị di động thật sự với cấu hình phần cứng của nó về bộ nhớ và tốc độ xử lý. Dĩ nhiên thật ra nó không phải là một phần của J2ME nhưng nó là nơi xuất phát. Các thiết bị di động khác nhau có thể có các bộ vi xử lý khác nhau với các tập mã lệnh khác nhau. Mục tiêu của J2ME là cung cấp một chuẩn cho tất cả các loại thiết bị di động khác nhau.

Tầng máy ảo Java (Java Virtual Machine Layer)

Khi mã nguồn Java được biên dịch nó được chuyển đổi thành mã bytecode. Mã bytecode này sau đó được chuyển thành mã ngôn ngữ máy của thiết bị di động. Tầng máy ảo Java bao gồm KVM (K Virtual Machine) là bộ biên dịch mã bytecode có nhiệm vụ chuyển mã bytecode của chương trình Java thành ngôn ngữ máy để chạy trên thiết bị di động. Tầng này cung cấp một sự chuẩn hóa cho các thiết bị di động để ứng dụng J2ME sau khi đã biên dịch có thể hoạt động trên bất kỳ thiết bị di động nào có J2ME KVM.

Tầng cấu hình (Configuration Layer)

Tầng cấu hình của CLDC định nghĩa giao diện ngôn ngữ Java (Java language interface) cơ bản để cho phép chương trình Java chạy trên thiết bị di động. Đây là một tập các API định nghĩa lõi của ngôn ngữ J2ME. Lập trình viên có thể sử dụng các lớp và phương thức của các API này tuy nhiên tập các API hữu dụng hơn được chứa trong tầng hiện trạng (profile layer).

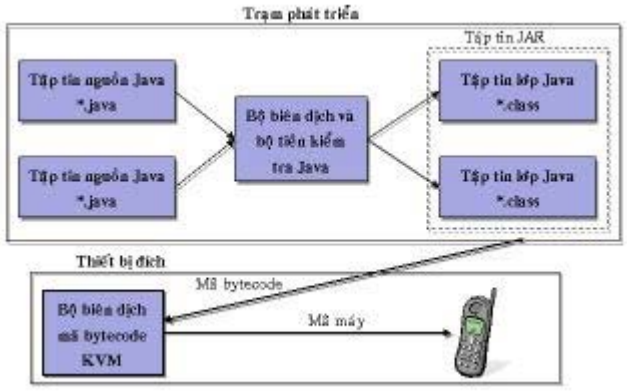
Tầng hiện trạng (Profile Layer)

Tầng hiện trạng hay MIDP (Hiện trạng thiết bị thông tin di động-Mobile Information

Device Profile) cung cấp tập các API hữu dụng hơn cho lập trình viên. Mục đích của hiện trạng là xây dựng trên lớp cấu hình và cung cấp nhiều thư viện ứng dụng hơn. MIDP định nghĩa các API riêng biệt cho thiết bị di động. Cũng có thể có các hiện trạng và các API khác ngoài MIDP được dùng cho ứng dụng. Ví dụ, có thể có hiện trạng PDA định nghĩa các lớp và phương thức hữu dụng cho việc tạo các ứng dụng PDA (lịch, sổ hẹn, sổ địa chỉ,…). Cũng có thể có một hiện trạng định nghĩa các API cho việc tạo các ứng dụng Bluetooth. Thực tế, các hiện trạng kể trên và tập các API đang được xây dựng. Chuẩn hiện trạng PDA là đặc tả JSR - 75 và chuẩn bluetooth API là đặc tả JSR - 82 với JSR là viết tắt của Java Specification Request.

1 Máy ảo Java (hay KVM)

Vai trò của máy ảo Java hay KVM là dịch mã bytecode được sinh ra từ chương trình Java đã biên dịch sang ngôn ngữ máy. Chính KVM sẽ chuẩn hóa output của các chương trình Java cho các thiết bị di động khác nhau có thể có bộ vi xử lý và tập lệnh khác nhau. Không có KVM, các chương trình Java phải được biên dịch thành tập lệnh cho mỗi thiết bị di động. Như vậy lập trình viên phải xây dựng nhiều đích cho mỗi loại thiết bị di động. Hình 2 đây biểu diễn tiến trình xây dựng ứng dụng MIDlet hoàn chỉnh và vai trò của KVM.



Hình 2. Tiến trình xây dựng MIDlet

Quá trình phát triển ứng dụng MIDlet với IDE (Môi trường phát triển tích hợp- Intergrated Development Environment):

Lập trình viên: Tạo các tập tin nguồn Java
Bước đầu tiên là lập trình viên phải tạo mã nguồn Java, có thể có nhiều tập tin (*.java).
Trên IDE: Bộ biên dịch Java (Java Compiler): Biên dịch mã nguồn thành mã bytecode

Bộ biên dịch Java sẽ biên dịch mã nguồn thành mã bytecode. Mã bytecode này sẽ được KVM dịch thành mã máy. Mã bytecode đã biên dịch sẽ được lưu trong các tập tin *.class và sẽ có một tập tin *.class sinh ra cho mỗi lớp Java.

Trên IDE: Bộ tiền kiểm tra (Preverifier): Kiểm tra tính hợp lệ của mã bytecode
Một trong những yêu cầu an toàn của J2ME là bảo đảm mã bytecode chuyển cho KVM là hợp lệ và không truy xuất các lớp hay bộ nhớ ngoài giới hạn của chúng. Do

đó tất cả các lớp đều phải được tiền kiểm tra trước khi chúng có thể được download về thiết bị di động. Việc tiền kiểm tra được xem là một phần của môi trường phát triển làm cho KVM có thể được thu nhỏ hơn. Bộ tiền kiểm tra sẽ gán nhãn lớp bằng một thuộc tính (attribute) đặc biệt chỉ rằng lớp đó đã được tiền kiểm tra. Thuộc tính này tăng thêm khoảng 5% kích thước của lớp và sẽ được kiểm tra bởi bộ kiểm tra trên thiết bị di động.

Trên IDE: Tạo tập tin JAR
IDE sẽ tạo một tập tin JAR chứa:

* Tất cả các tập tin *.class
* Các hình ảnh của ứng dụng. Hiện tại chỉ hỗ trợ tập tin *.png
* Các tập tin dữ liệu có thể được yêu cầu bởi ứng dụng
* Một tập tin kê khai (manifest.mf) cung cấp mô tả về ứng dụng cho bộ quản lý ứng dụng (application manager) trên thiết bị di động.
* Tập tin JAR được bán hoặc được phân phối đến người dùng đầu cuối

Sau khi đã gỡ rối và kiểm tra mã lệnh trên trình giả lập (simulator), mã lệnh đã sẵn sàng được kiểm tra trên điện thoại di động và sau đó được phân phối cho người dùng.

Người dùng: Download ứng dụng về thiết bị di động
Người dùng sau đó download tập tin JAR chứa ứng dụng về thiết bị di động. Trong hầu hết các điện thoại di động, có ba cách để download ứng dụng:

* Kết nối cáp dữ liệu từ PC sang cổng dữ liệu của điện thoại di động:
Việc này yêu cầu người dùng phải có tập tin JAR thật sự và phần mềm truyền thông để download ứng dụng sang thiết bị thông qua cáp dữ liệu.
* Cổng hồng ngoại IR (Infra Red) Port:
Việc này yêu cầu người dùng phải có tập tin JAR thật sự và phần mềm truyền thông để download ứng dụng sang thiết bị thông qua cổng hồng ngoại.
* OTA (Over the Air):
Sử dụng phương thức này, người dùng phải biết địa chỉ URL chỉ đến tập tin JAR

Trên thiết bị di động:

Bộ tiền kiểm tra: Kiểm tra mã bytecode
Bộ tiền kiểm tra kiểm tra tất cả các lớp đều có một thuộc tính hợp lệ đã được thêm vào bởi bộ tiền kiểm tra trên trạm phát triển ứng dụng. Nếu tiến trình tiền kiểm tra thất bại thì ứng dụng sẽ không được download về thiết bị di động.
Bộ quản lý ứng dụng: Lưu trữ chương trình
Bộ quản lý ứng dụng trên thiết bị di động sẽ lưu trữ chương trình trên thiết bị di động. Bộ quản lý ứng dụng cũng điều khiển trạng thái của ứng dụng trong thời gian thực thi và có thể tạm dừng ứng dụng khi có cuộc gọi hoặc tin nhắn đến.

Người dùng: Thực thi ứng dụng
Bộ quản lý ứng dụng sẽ chuyển ứng dụng cho KVM để chạy trên thiết bị di động.

KVM: Thực thi mã bytecode khi chương trình chạy.
KVM dịch mã bytecode sang ngôn ngữ máy của thiết bị di động để chạy.

2 Tầng CLDC (Connected Limited Device Configuration)

Tầng J2ME kế trên tầng KVM là CLDC hay Cấu hình thiết bị kết nối giới hạn. Mục đích của tầng này là cung cấp một tập tối thiểu các thư viện cho phép một ứng dụng Java chạy trên thiết bị di động. Nó cung cấp cơ sở cho tầng Hiện trạng, tầng này sẽ chứa nhiều API chuyên biệt hơn.
Các CLDC API được định nghĩa với sự hợp tác với 18 công ty là bộ phận của JCP (Java Community Process). Nhóm này giúp bảo đảm rằng các API được định nghĩa sẽ hữu dụng và thiết thực cho cả nhà phát triển lẫn nhà sản xuất thiết bị di động. Các đặc tả của JCP được gán các số JSR (Java Specification Request). Quy định CLDC phiên bản 1.0 được gán số JSR - 30.

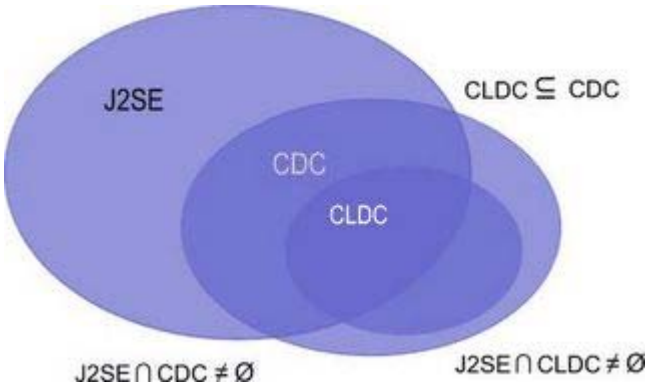
2.a CLDC – Connected Limited Device Configuration
Phạm vi: Định nghĩa các thư viện tối thiểu và các API.
Định nghĩa:

* Tương thích ngôn ngữ JVM
* Các thư viện lõi
* I/O
* Mạng
* Bảo mật
* Quốc tế hóa

Không định nghĩa:

* Chu kỳ sống ứng dụng
* Giao diện người dùng
* Quản lý sự kiện
* Giao diện ứng dụng và người dùng

Các lớp lõi Java cơ bản, input/output, mạng, và bảo mật được định nghĩa trong CLDC. Các API hữu dụng hơn như giao diện người dùng và quản lý sự kiện được dành cho hiện trạng MIDP.
J2ME là một phiên bản thu nhỏ của J2SE, sử dụng ít bộ nhớ hơn để nó có thể thích hợp với các thiết bị di động bị giới hạn bộ nhớ. Mục tiêu của J2ME là một tập con 100% tương thích của J2SE.



Hình 3 biểu diễn mối liên hệ giữa J2SE và J2ME (CDC, và CLDC).

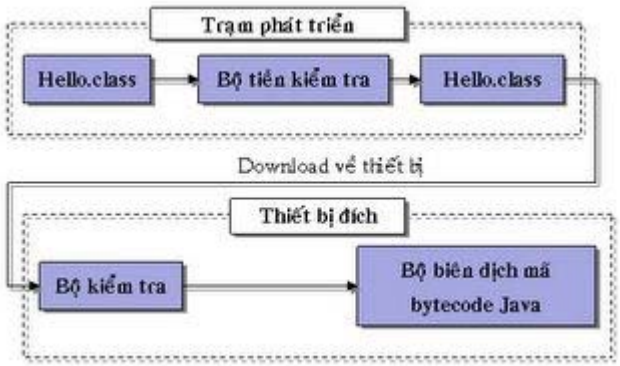
2.b Sự khác nhau giữa J2ME và J2SE.

Các điểm khác nhau là do một trong hai lý do. Do lớp Java đã bị bỏ đi để giảm kích

thước của J2ME hoặc do lớp bị bỏ bởi vì nó ảnh hưởng đến sự an toàn, bảo mật của thiết bị di động hay của các ứng dụng khác trên thiết bị di động (có thể dẫn đến phát triển virus).

Điểm khác biệt chính là không có phép toán số thực. Không có JNI (JavaNative Interface Support) do đó bạn không thể truy xuất các chương trình khác được viết bằng ngôn ngữ của thiết bị (như C hay C++). Tuyến đoạn (thread) được cho phép nhưng không có các nhóm tuyến đoạn (thread group) và các daemon thread.

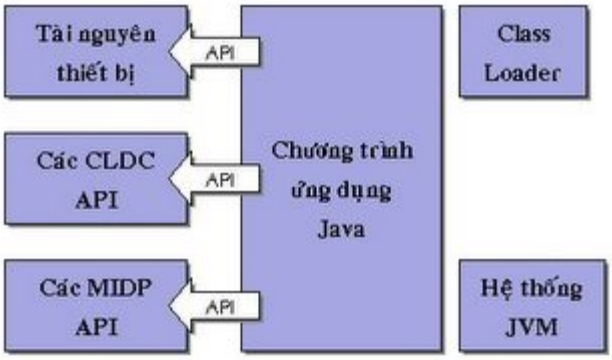
CLDC định nghĩa một mô hình an toàn, bảo mật được thiết kế để bảo vệ thiết bị di động, KVM, và các ứng dụng khác khỏi các mã phá hoại. Hai bộ phận được định nghĩa bởi CLDC này là bộ tiền kiểm tra và mô hình sandbox.



Hình 4 biểu diễn cách mà bộ tiền kiểm tra và bộ kiểm tra làm việc với nhau để kiểm tra mã chương trình Java trước khi chuyển nó cho KVM.

Như đã đề cập trước đây, các tập tin lớp được gán nhãn bằng một thuộc tính trên máy trạm của nhà phát triển. Thuộc tính này sau đó được kiểm tra bởi bộ tiền kiểm tra trước khi mã chương trình được giao cho KVM hay bộ biên dịch mã bytecode.

Một bộ phận khác của bảo mật trong CLDC là mô hình sandbox.



Hình 5 biểu diễn khái niệm mô hình sandbox

Hình trên cho thấy ứng dụng J2ME đặt trong một sandbox có nghĩa là nó bị giới hạn

truy xuất đến tài nguyên của thiết bị và không được truy xuất đến Máy ảo Java hay bộ nạp chương trình. Ứng dụng được truy xuất đến các API của CLDC và MIDP. Ứng dụng được truy xuất tài nguyên của thiết bị di động (các cổng, âm thanh, bộ rung, các báo hiệu,…) chỉ khi nhà sản xuất điện thoại di động cung cấp các API tương ứng. Tuy nhiên, các API này không phải là một phần của J2ME.

Thế hệ kế tiếp của CLDC là đặc tả JSR - 139 và được gọi là CLDC thế hệ kế tiếp (Next Generation). Nó sẽ nhắm đến các vấn đề như nâng cao việc quản lý lỗi và có thể phép toán số thực.

3 MIDP (Mobile Information Device Profile)

Tầng J2ME cao nhất là tầng hiện trạng và mục đích của nó là định nghĩa các API cho các thiết bị di động. Một thiết bị di động có thể hỗ trợ nhiều hiện trạng. Một hiện trạng có thể áp đặt thêm các giới hạn trên các loại thiết bị di động (như nhiều bộ nhớ hơn hay độ phân giải màn hình cao hơn). Hiện trạng là tập các API hữu dụng hơn cho các ứng dụng cụ thể. Lập trình viên có thể viết một ứng dụng cho một hiện trạng cụ thể và không cần quan tâm đến nó chạy trên thiết bị nào.

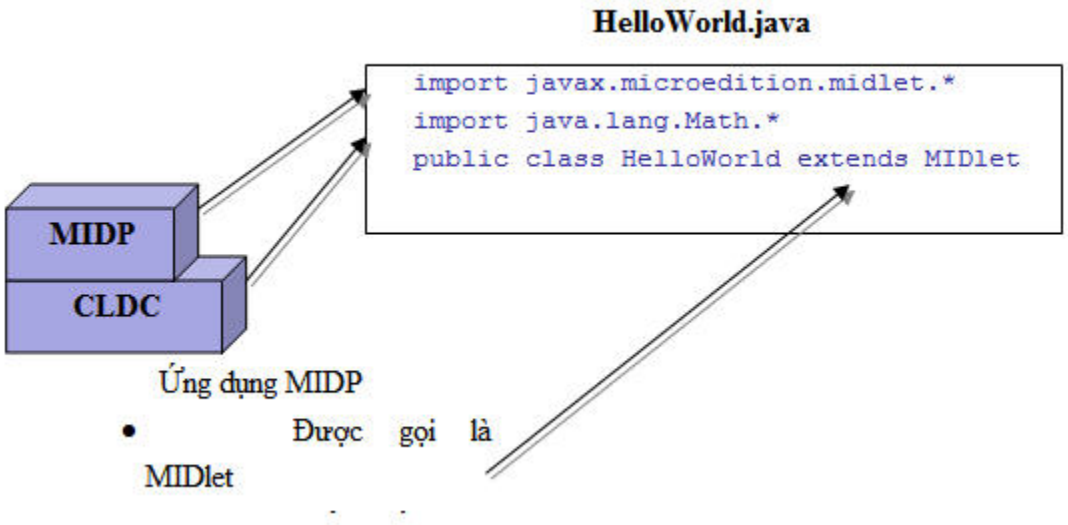
Hiện tại hiện trạng được công bố là MIDP (Mobile Information Profile) với đặc tả JSR - 37. Có 22 công ty là thành viên của nhóm chuyên gia tạo ra chuẩn MIDP.

MIDP cung cấp các API cho phép thay đổi trạng thái chu kỳ sống ứng dụng, đồ họa (mức cao và mức thấp), tuyến đoạn, timer, lưu trữ bền vững (persistent storage), và mạng.

Nó không định nghĩa cách mà ứng dụng được nạp trong thiết bị di động. Đó là trách nhiệm của nhà sản xuất. Nó cũng không định nghĩa bất kỳ loại mô hình bảo mật end-to-end nào, vốn cần thiết cho ứng dụng kinh doanh nhận số thẻ tín dụng của người dùng. Nó cũng không bắt buộc nhà sản xuất cách mà lớp MIDP được thực hiện.

Từng bước lập trình cho điện thoại di động J2ME - Phần 2

1/ MIDlet
Các ứng dụng J2ME được gọi là MIDlet (Mobile Information Device applet).

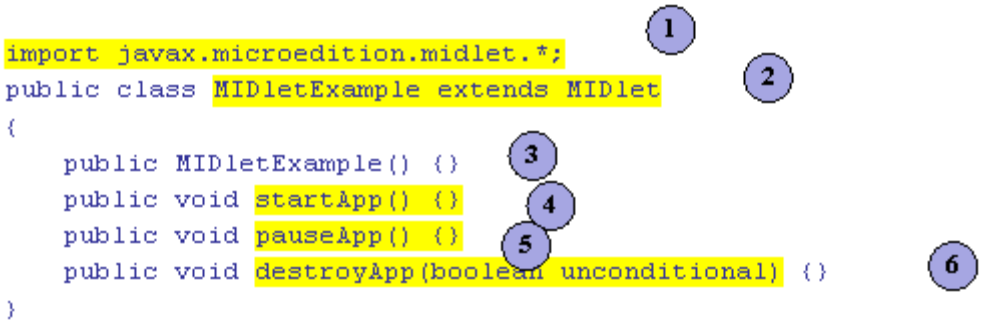


Hình 1. MIDlet

Thông báo import dùng để truy xuất các lớp của CLDC và MIDP.

Lớp chính của ứng dụng được định nghĩa là lớp kế thừa lớp MIDlet của MIDP. Có thể chỉ có một lớp trong ứng dụng kế thừa lớp này. Lớp MIDlet được trình quản lý ứng dụng trên điện thoại di động dùng để khởi động, dừng, và tạm dừng MIDlet (ví dụ, trong trường hợp có cuộc gọi đến).
1.1 Bộ khung MIDlet (MIDlet Skeleton)

Một MIDlet là một lớp Java kế thừa (extend) của lớp trừu tượng java.microedition.midlet.MIDlet và thực thi (implement) các phương thức startApp(), pauseApp(), và destroyApp().



Hình 2 biểu diễn bộ khung yêu cầu tối thiểu cho một ứng dụng MIDlet

1) Phát biểu import

Các phát biểu import được dùng để include các lớp cần thiết từ các thư viện CLDC và MIDP.

2) Phần chính của MIDlet

MIDlet được định nghĩa như một lớp kế thừa lớp MIDlet. Trong ví dụ này MIDletExample là bắt đầu của ứng dụng.

3) Hàm tạo (Constructor)

Hàm tạo chỉ được thực thi một lần khi MIDlet được khởi tạo lần đầu tiên. Hàm tạo sẽ không được gọi lại trừ phi MIDlet thoát và sau đó khởi động lại.

4) startApp()

Phương thức startApp() được gọi bởi bộ quản lý ứng dụng khi MIDlet được khởi tạo, và mỗi khi MIDlet trở về từ trạng thái tạm dừng. Nói chung, các biến toàn cục sẽ được khởi tạo lại trừ hàm tạo bởi vì các biến đã được giải phóng trong hàm pauseApp(). Nếu không thì chúng sẽ không được khởi tạo lại bởi ứng dụng.

5) pauseApp()

Phương thức pauseApp() được gọi bởi bộ quản lý ứng dụng mỗi khi ứng dụng cần được tạm dừng (ví dụ, trong trường hợp có cuộc gọi hoặc tin nhắn đến). Cách thích hợp để sử dụng pauseApp() là giải phóng tài nguyên và các biến để dành cho các chức năng khác trong điện thoại trong khi MIDlet được tạm dừng. Cần chú ý rằng khi nhận cuộc gọi đến hệ điều hành trên điện thoại di động có thể dừng KVM thay vì dừng MIDlet. Việc này không được đề cập trong MIDP mà đó là do nhà sản xuất quyết định sẽ chọn cách nào.

6) destroyApp()

Phương thức destroyApp() được gọi khi thoát MIDlet. (ví dụ khi nhấn nút exit trong ứng dụng). Nó chỉ đơn thuần là thoát MIDlet. Nó không thật sự xóa ứng dụng khỏi điện thoại di động. Phương thức destroyApp() chỉ nhận một tham số Boolean. Nếu tham số này là true, MIDlet được tắt vô điều kiện. Nếu tham số là false, MIDlet có thêm tùy chọn từ chối thoát bằng cách ném ra một ngoại lệ MIDletStateChangeException.

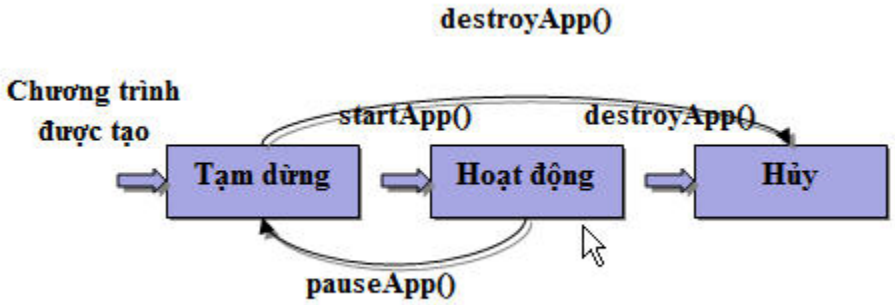
Tóm tắt các trạng thái khác nhau của MIDlet:

Tạo (Created) ð Hàm tạo MIDletExample() được gọi một một lần

Hoạt động (Active) ð Phương thức startApp() được gọi khi chương trình bắt đầu hay sau khi tạm dừng

Tạm dừng (Paused) ð Phương thức pauseApp() được gọi. Có thể nhận các sự kiện timer.

Hủy (Destroyed) ð Phương thức destroy() được gọi.
1.2 Chu kỳ sống của MIDlet (MIDlet lifecycle)



Hình 3 biểu diễn chu kỳ sống của MIDlet

Khi người dùng yêu cầu khởi động ứng dụng MIDlet, bộ quản lý ứng dụng sẽ thực thi MIDlet (thông qua lớp MIDlet). Khi ứng dụng thực thi, nó sẽ được xem là đang ở trạng thái tạm dừng. Bộ quản lý ứng dụng gọi hàm tạo và hàm startApp(). Hàm startApp() có thể được gọi nhiều lần trong suốt chu kỳ sống của ứng dụng. Hàm destroyApp() chỉ có thể gọi từ trạng thái hoạt động hay tạm dừng.

Lập trình viên cũng có thể điều khiển trạng thái của MIDlet.

Các phương thức dùng để điều khiển các trạng thái của MIDlet:

resumeRequest(): Yêu cầu vào chế độ hoạt động

Ví dụ: Khi MIDlet tạm dừng, và một sự kiện timer xuất hiện.

notifyPaused(): Cho biết MIDlet tự nguyện chuyển sang trạng thái tạm dừng

Ví dụ: Khi đợi một sự kiện timer.

notifyDestroyed(): Sẵn sàng để hủy

Ví dụ: Xử lý nút nhấn Exit

Lập trình viên có thể yêu cầu tạm dừng MIDlet trong khi đợi một sự kiện timer hết hạn. Trong trường hợp này, phương thức notifyPaused() sẽ được dùng để yêu cầu bộ quản lý ứng dụng chuyển ứng dụng sang trạng thái tạm dừng.

1.3 Tập tin JAR

Các lớp đã biên dịch của ứng dụng MIDlet được đóng gói trong một tập tin JAR (Java Archive File). Đây chính là tập tin JAR được download xuống điện thoại di động.

Tập tin JAR chứa tất cả các tập tin class từ một hay nhiều MIDlet, cũng như các tài nguyên cần thiết. Hiện tại, MIDP chỉ hỗ trợ định dạng hình .png (Portable Network Graphics). Tập tin JAR cũng chứa tập tin kê khai (manifest file) mô tả nội dung của MIDlet cho bộ quản lý ứng dụng. Nó cũng phải chứa các tập tin dữ liệu mà MIDlet cần. Tập tin JAR là toàn bộ ứng dụng MIDlet. MIDlet có thể load và triệu gọi các phương thức từ bất kỳ lớp nào trong tập tin JAR, trong MIDP, hay CLDC. Nó không thể truy xuất các lớp không phải là bộ phận của tập tin JAR hay vùng dùng chung của thiết bị di động.

1.4 Tập tin kê khai (manifest) và tập tin JAD

Tập tin kê khai (manifest.mf) và tập tin JAD (Java Application Descriptor) mô tả các đặc điểm của MIDlet. Sự khác biệt của hai tập tin này là tập tin kê khai là một phần của tập tin JAR còn tập tin JAD không thuộc tập tin JAR. Ưu điểm của tập tin JAD là các đặc điểm của MIDlet có thể được xác định trước khi download tập tin JAR. Nói chung, cần ít thời gian để download một tập tin văn bản nhỏ hơn là download một tập tin JAR. Như vậy, nếu người dùng muốn download một ứng dụng không được thiết bị di động hỗ trợ (ví dụ, MIDP 2.0), thì quá trình download sẽ bị hủy bỏ thay vì phải đợi download hết toàn bộ tập tin JAR.

Mô tả nội dung của tập tin JAR:

Các trường yêu cầu

· Manifest-Version // Phiên bản tập tin Manifest

· MIDlet-Name // Tên bộ MIDlet (MIDlet suite)

· MIDlet-Version // Phiên bản bộ MIDlet

· MIDlet-Vendor // Nhà sản xuất MIDlet

· MIDlet- for each MIDlet // Tên của MIDlet

· MicroEdtion-Profile // Phiên bản hiện trạng

· MicroEdtion-Configuration // Phiên bản cấu hình

Ví dụ một tập tin manifest.mf:

MIDlet-Name: CardGames

MIDlet-Version: 1.0.0

MIDlet-Vendor: Sony Ericsson

MIDlet-Description: Set of Card Games

MIDlet-Info-URL: http://www.semc.com/games

MIDlet-Jar-URL: http://www.semc.com/j2me/games

MIDlet-Jar-Size: 1063

MicroEdtion-Profile: MIDP-1.0

MicroEdtion-Configuration: CLDC-1.0

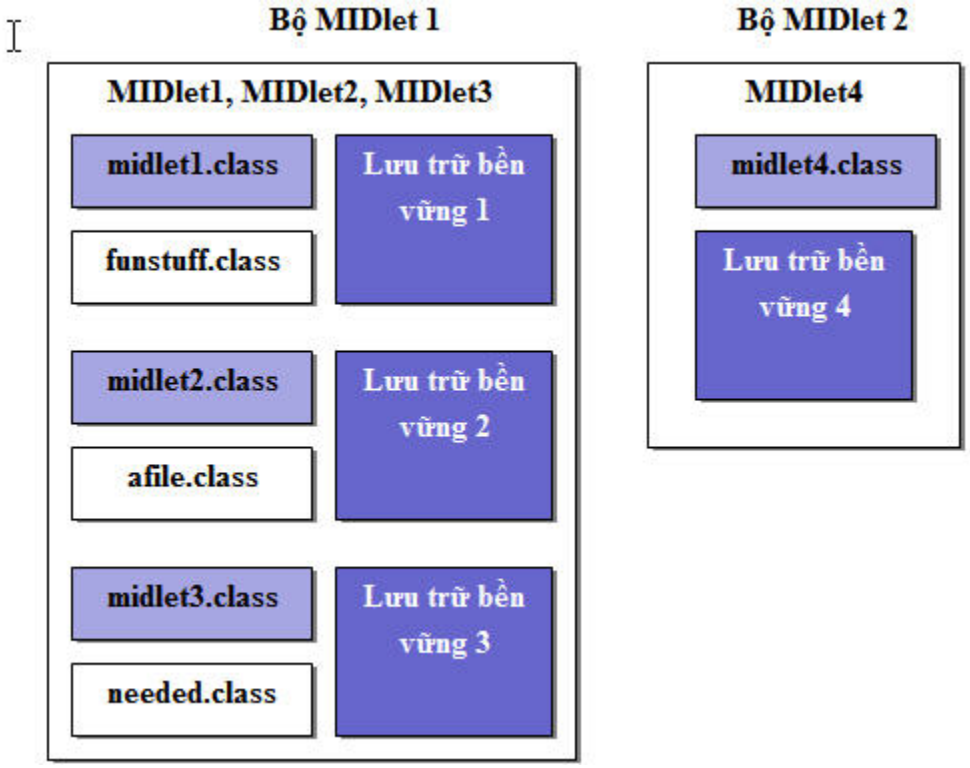
MIDlet-1: Solitaire, /Sol.png, com.semc.Solitaire

MIDlet-2: BlackJack, /Blkjk.png, com.semc.BlackJack

Tập tin JAD chứa cùng thông tin như tập tin manifest. Nhưng nó nằm ngoài tập tin JAR.

Các thuộc tính MIDlet-Name, MIDlet-Version, và MIDlet-Vendor phải được lặp lại trong tập tin JAD và JAR. Các thuộc tính khác không cần phải lặp lại. Giá trị trong tập tin mô tả sẽ đè giá trị của tập tin manifest.
1.5 Bộ MIDlet (MIDlet Suite)

Một tập các MIDlet trong cùng một tập tin JAR được gọi là một bộ MIDlet (MIDlet suite). Các MIDlet trong một bộ MIDlet chia sẻ các lớp, các hình ảnh, và dữ liệu lưu trữ bền vững. Để cập nhật một MIDlet, toàn bộ tập tin JAR phải được cập nhật.



Hình 4 biểu diễn hai bộ MIDlet

Trong hình trên, một bộ MIDlet chứa MIDlet1, MIDlet2, và MIDlet3. Bộ kia chỉ chứa MIDlet4. Ba MIDlet trong bộ đầu tiên truy xuất các lớp và dữ liệu của nhau nhưng không truy xuất đến các lớp hay dữ liệu của MIDlet4. Ngược lại, MIDlet4 cũng không truy xuất được các lớp, hình ảnh, và dữ liệu của chúng.

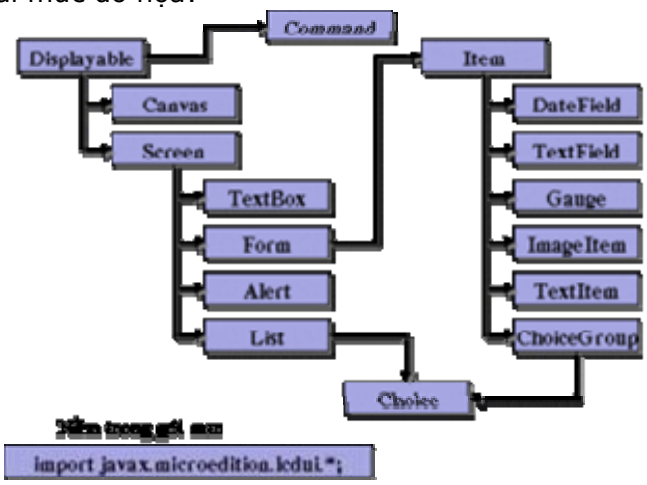
Từng bước lập trình cho điện thoại di động J2ME - Phần 3

Bài 3 - Đồ họa trong J2ME

1 Đồ họa (Graphic)

1.1 Đồ họa mức thấp (low level) và mức cao (high level)
Các lớp MIDP cung cấp hai mức đồ họa: đồ họa mức thấp và đồ họa mức cao. Đồ họa mức cao dùng cho văn bản hay form. Đồ họa mức thấp dùng cho các ứng dụng trò chơi yêu phải vẽ lên màn hình.

Hình 1 biểu diễn hai mức đồ họa:



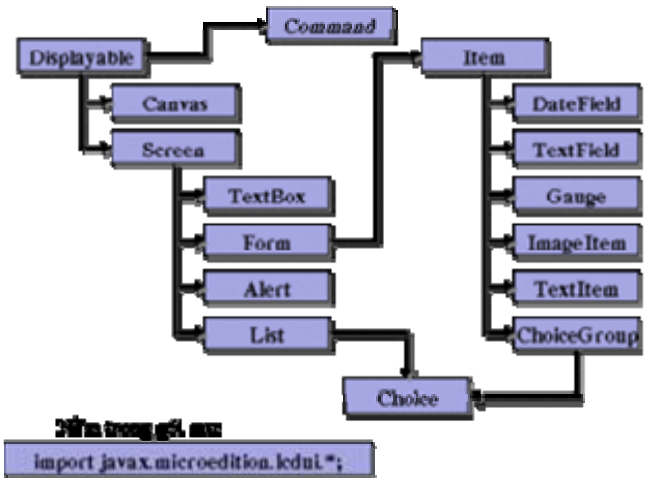
Hình 1 . Hai mức đồ họa

Cả hai lớp đồ họa mức thấp và mức cao đều là lớp con của lớp Displayble. Trong MIDP, chỉ có thể có một lớp displayable trên màn hình tại một thời điểm. Có thể định nghĩa nhiều màn hình nhưng một lần chỉ hiển thị được một màn hình.

1.1.a Đồ họa mức cao (High Level Graphics) (Lớp Screen) Đồ họa mức cao là lớp con của lớp Screen. Nó cung cấp các thành phần như text box, form, list, và alert. Ta ít điều khiển sắp xếp các thành phần trên màn hình. Việc sắp xếp thật sự phụ thuộc vào nhà sản xuất.

1.1.b Đồ họa mức thấp (Lớp Canvas)Đồ họa mức thấp là lớp con của lớp Canvas. Lớp này cung cấp các phương thức đồ họa cho phép vẽ lên màn hình hay vào một bộ đệm hình cùng với các phương thức xử lý sự kiện bàn phím. Lớp này dùng cho các ứng dụng trò chơi cần điều khiển nhiều về màn hình.

Hình 2 biểu diễn phân cấp lớp đồ họa:



Hình 2 . Phân cấp lớp đồ họa

Form có thể là kiểu đồ họa hữu dụng nhất của các lớp Screen vì nó cho phép chứa nhiều item khác nhau. Nếu sử dụng các lớp khác (TextBox, List) thì chỉ có một item được hiển thị bởi vì chúng đều là đối tượng Displayable và do chỉ có thể có một đối tượng Displayable được hiển thị tại một thời điểm. Form cho phép chứa nhiều item khác nhau (DateField, TextField, Gauge, ImageItem, TextItem, ChoiceGroup).

1.2 Đồ họa mức cao
Là các đối tượng của lớp Screen

1.2.a TextBox
Lớp TextBox cho phép người dùng nhập và soạn thảo văn bản. Lập trình viên có thể định nghĩa số ký tự tối đa, giới hạn loại dữ liệu nhập (số học, mật khẩu, email,…) và hiệu chỉnh nội dung của textbox. Kích thước thật sự của textbox có thể nhỏ hơn yêu cầu khi thực hiện thực tế (do giới hạn của thiết bị). Kích thước thật sự của textbox có thể lấy bằng phương thức getMaxSize().

1.2.b Form
Form là lớp hữu dụng nhất của các lớp Screen bởi vì nó cho phép chứa nhiều item trên cùng một màn hình. Các item có thể là DateField, TextField, ImageItem, TextItem, ChoiceGroup.

1.2.c List
Lớp List là một Screen chứa danh sách các lựa chọn chẳng hạn như các radio button. Người dùng có thể tương tác với list và chọn một hay nhiều item.

1.2.d Alert
Alert hiển thị một màn hình pop-up trong một khoảng thời gian. Nói chung nó dùng để cảnh báo hay báo lỗi. Thời gian hiển thị có thể được thiết lập bởi ứng dụng. Alert có thể được gán các kiểu khác nhau (alarm, confirmation, error, info, warning), các âm thanh tương ứng sẽ được phát ra.

1.3 Form và các Form Item
Sử dụng form cho phép nhiều item khác nhau trong cùng một màn hình. Lập trình viên không điều khiển sự sắp xếp các item trên màn hình. Sau khi đã định nghĩa đối tượng Form, sau đó sẽ thêm vào các item.

Mỗi item là một lớp con của lớp Item.

1.3.a String Item
Public class StringItem extends Item

StringItem chỉ là một chuỗi hiển thị mà người dùng không thể hiệu chỉnh. Tuy nhiên, cả nhãn và nội dung củaStringItem có thể được hiệu chỉnh bởi ứng dụng.

1.3.b Image Item
public class ImageItem extends Item

ImageItem cho phép thêm vào hình form. ImageItem chứa tham chiếu đến một đối tượng Image phải được tạo trước đó.

1.3.c Text Field
public class TextField extends Item

TextField cho phép người dùng nhập văn bản. Nó có thể có giá trị khởi tạo, kích thước tối đa, và ràng buộc nhập liệu. Kích thước thật sự có thể nhỏ hơn yêu cầu do giới hạn của thiết bị di động.

1.3.d Date Field
public class DateField extends Item

DateField cho phép người dùng nhập thông tin ngày tháng và thời gian. Có thể xác định giá trị khởi tạo và chế độ nhập ngày tháng (DATE), thời gian (TIME), hoặc cả hai.

1.3.e Choice Group
public class ChoiceGroup extends Item Implements Choice

ChoiceGroup cung cấp một nhóm các radio-button hay checkbox cho phép lựa chọn đơn hay lựa chọn nhiều.

1.3.f Gauge
public class Gauge extends Item

Lớp Gauge cung cấp một hiển thị thanh (bar display) của một giá trị số học. Gauge có thể có tính tương tác hoặc không. Nếu một gauge là tương tác thì người dùng có thể thay đổi giá trị của tham số qua gauge. Gauge không tương tác chỉ đơn thuần là để hiển thị.

1.4 Ticker
Một màn hình có thể có một ticker là một chuỗi văn bản chạy liên tục trên màn hình. Hướng và tốc độ là do thực tế qui định. Nhiều màn hình có thể chia sẻ cùng một ticker.

Ví dụ:

```
Ticker myTicker = new Ticker("Useful Information");
MainScreen = new Form("Main Screen");
MainScreen.setTicker(myTicker);
```

Ticker(String str)

public class Ticker extends Object

Từng bước lập trình cho điện thoại di động J2ME - Phần 4

1 Lưu trữ bản ghi (Record Store)

Lưu trữ bản ghi cho phép lưu dữ liệu khi ứng dụng thoát, khởi động lại và khi thiết bị di động tắt hay thay pin. Dữ liệu lưu trữ bản ghi sẽ tồn tại trên thiết bị di động cho đến khi ứng dụng thật sự được xóa khỏi thiết bị di động. Khi một MIDlet bị xóa, tất cả các lưu trữ bản ghi của nó cũng bị xóa.

Hình 1 minh họa dữ liệu lưu trữ bản ghi với MIDlet

Như trong hình, các MIDlet có thể có nhiều hơn một tập lưu trữ bản ghi, chúng chỉ có thể truy xuất dữ liệu lưu trữ bản ghi chứa trong bộ MIDlet của chúng. Do đó, MIDlet 1 và MIDlet 2 có thể truy xuất dữ liệu trong Record Store 1 và Record Store 2 nhưng chúng không thể truy xuất dữ liệu trong Record Store3. Ngược lại, MIDlet 3 chỉ có thể truy xuất dữ liệu trong Record Store 3 và không thể truy xuất dữ liệu dữ liệu trong Record Store 1 và Record Store 2. Tên của các lưu trữ bản ghi phải là duy nhất trong một bộ MIDlet nhưng các bộ khác nhau có thể dùng trùng tên.

Các bản ghi trong một lưu trữ bản ghi được sắp xếp thành các mảng byte. Các mảng byte không có cùng chiều dài và mỗi mảng byte được gán một số ID bản ghi.

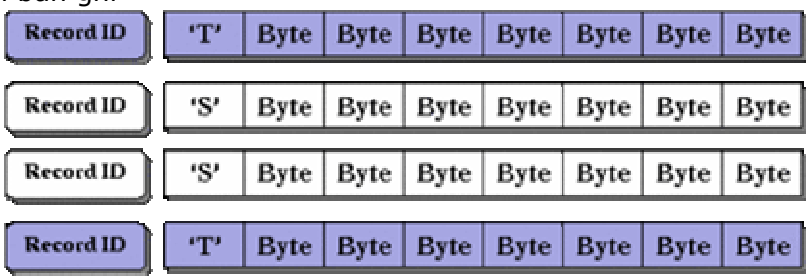
Các bản ghi được định danh bằng một số ID bản ghi (record ID) duy nhất. Các số ID bản ghi được gán theo thứ tự bắt đầu từ 1. Các số sẽ không được dùng lại khi một bản ghi bị xóa do đó sẽ tồn tại các khoảng trống trong các ID bản ghi. Đặc tả MIDP không định nghĩa chuyện gì xảy ra khi đạt đến số ID bản ghi tối đa, điều này phụ thuộc vào ứng dụng.

1.1 Định dạng (Format), Thêm (Add) và Xóa (Delete) các bản ghi
Thêm bản ghi gồm hai bước. Bước đầu tiên là định dạng bản ghi theo định dạng yêu cầu và bước tiếp theo là thêm bản ghi đã định dạng vào lưu trữ bản ghi. Sự tuần tự hóa (serialization) dữ liệu lưu trữ bản ghi không được hỗ trợ, do đó lập trình viên phải định định dạng các mảng byte để xây dựng dữ liệu lưu trữ bản ghi

Sau đây là ví dụ của việc định dạng dữ liệu bản ghi, mở một lưu trữ bản ghi và sau đó thêm dữ liệu bản ghi vào lưu trữ bản ghi

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
DataOutputStream outputStream = new DataOutputStream(baos);
outputStream.writeByte('T'); // byte [0] Thẻ chỉ loại bản ghi
outputStream.writeInt(score); // byte [1] đến [4]
outputStream.writeUTF(name); // byte [5] đến 2 + name.length
byte[] theRecord = boas.toByteArray();
recordStore rs = null;
rs = RecordStore.openRecordStore("RecordStoreName", CreateIfNoExist);
int RecordID = rs.addRecord(theRecord, 0, theRecord.length);
```

Hình 2. Thêm bản ghi



1.1.a Định dạng dữ liệu bản ghi
Trong ví dụ trên, hai dòng đầu tạo một luồng xuất để giữ dữ liệu bản ghi. Sử dụng đối tượng DataOutputStream (bọc mảng byte) cho phép các bản ghi dễ dàng được định dạng theo các kiểu chuẩn của Java (long, int, string,…) mà không phải quan tâm đến tách nó thành dữ liệu byte. Phương thức writeByte(), writeInt(), và writeUTF() định dạng dữ liệu như trong hình (tag, score, name). Sử dụng thẻ (tag) làm byte đầu tiên có ích để xác định loại bản ghi sau này. Phương thức toByteArray() chép dữ liệu trong luồng xuất thành một mảng byte chứa bản ghi để lưu trữ. Biến theRecord là tham chiếu đến dữ liệu đã định dạng.

1.1.b Thêm dữ bản ghi đã định dạng vào lưu trữ bản ghi
Khi dữ liệu đã được định dạng, nó có thể được thêm vào lưu trữ bản ghi. Phát biểu openRecordStore() tạo và mở một lưu trữ bản ghi với tên là RecordStoreName. Phát biểu addRecord() thêm bản khi (bắt đầu bằng byte 0 của theRecord) và trả về ID bản ghi gắn với record này.

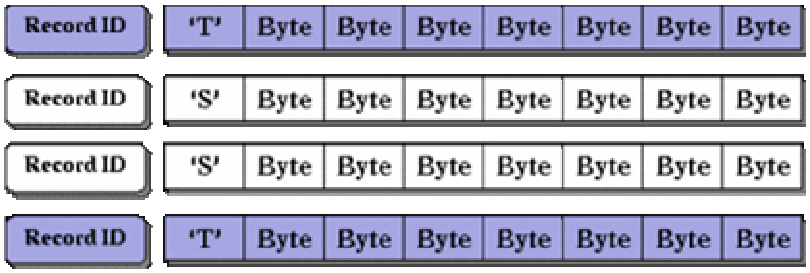
1.1.c Xóa bản ghi

Bản ghi được xóa bằng cách chuyển số ID bản ghi cho phương thức deleteRecord() của đối tượng RecordStore.
Ví dụ, bản ghi 7 bị xóa bằng phương thức deleteRecord(), nếu một bản ghi khác được thêm vào thì số ID bản ghi sẽ là 8 và ID bản ghi 7 sẽ không được dùng lại.

1.2 Lọc các bản ghi (Filtering Records)

Giao diện RecordFilter cung cấp một cách thuận tiện để lọc các bản ghi theo tiêu chuẩn của lập trình viên. RecordEnumeration có thể được dùng để duyệt qua các bản ghi và chỉ trả về các record phù hợp với tiêu chuẩn xác định. Giao diện RecordFilter có phương thức matches() dùng để xác định tiêu chuẩn phù hợp. Phương thức matches() có một tham số đầu vào là mảng byte biểu diễn một bản ghi. Phương thức phải trả về true nếu bản ghi này phù hợp với tiêu chuẩn đã định nghĩa.

Hình 3 minh họa ví dụ cách sử dụng giao diện RecordFilter



Hình 3. Lọc bản ghi

```
class IntegerFilter implements RecordFilter {
public boolean matches(byte[] candidate) throws IlleegalArgumentException {
return(candidate[0] == 'T');
}
```

Trong ví dụ trên, lớp IntegerFilter được dùng để lọc ra tất cả các bản ghi có 'T' ở byte đầu tiên. Nhớ rằng các bản ghi không phải có cùng định dạng. Do đó có byte đầu tiên làm thẻ (tag) rất có ích. Phương thức matches() chỉ trả về true nếu byte đầu tiên là 'T'.

1.3 Sắp xếp các bản ghi

Các bản ghi trong một lưu trữ bản ghi có thể được sắp xếp theo thứ tự do lập trình viên định nghĩa. Việc sắp xếp được thực hiện thông qua giao diện RecordComparator. Duyệt kê qua các bản ghi sẽ trả về các bản ghi theo thứ tự sắp xếp đã định nghĩa. Giao diện RecordComparator có phương thức compare() phải được implement để định nghĩa cách hai bản ghi so sánh theo thứ tự. Các tham số đầu vào là hai mảng byte biểu diễn hai bản ghi. Phương thức compare() phải trả về một trong ba giá trị:

EQUIVALENT: Hai bản khi được xem là giống nhau
FOLLOWS: Bản ghi đầu tiên có thứ tự theo sau bản khi thứ hai.
PRECEDES: Bản ghi đầu tiên có thứ tự đứng trước bản ghi thứ hai.

Ví dụ sắp xếp các bản ghi sử dụng giao diện RecordComparator

```
class IntegerCompare implements RecordComparator {
public int compare(byte[] b1, byte[] b2) {
```

```
DataInputStream is1 = new DataInputStream(new ByteArrayInputStream(b1));
DataInputStream is2 = new DataInputStream(new ByteArrayInputStream(b2));
is1.skip(1);
is2.skip(2);
int i1 = is1.readInt();
int i2 = is2.readInt();
if (i1 > i2) return RecordComparator.FOLLOWS;
if (i1 < i2) return RecordComparator.PRECEDES;
return RecordComparator.EQUIVALENT;
}
}
```

Trong ví dụ trên, các bản ghi được sắp xếp dựa trên giá trị số nguyên chứa trong 4 byte sau byte thẻ đầu tiên. Tham số b1 và b2 biểu diễn hai bản ghi được chuyển cho phương thức compare(). Sử dụng phương thức DataInputStream() cho phép sử dụng các kiểu dữ liệu chính của Java (int, long, String) thay vì phải thao tác trực tiếp với dữ liệu byte. Phương thức skip() bỏ qua byte thẻ đầu tiên trong mỗi luồng. Phương thức readInt() đọc số nguyên trực tiếp từ luồng nhập. Dòng cuối cùng so sánh các số nguyên và trả về giá trị (FOLLOWS, PRECEDES, và EQUIVALENT). Như vậy thứ tự sắp xếp của toàn bộ bản ghi sẽ được xác định bởi giá trị của các số nguyên.

1.4 Liệt kê (Enumerate) các bản ghi

Liệt kê qua các bản ghi trong lưu trữ bản ghi được thực hiện bằng cách dùng giao diện RecordEnumeration kết hợp với các lớp RecordFilter và RecordComparator. Lớp RecordEnumerator giữ thứ tự luận lý của các bản ghi. Lớp RecordFilter định nghĩa tập con của các bản ghi từ lưu trữ bản ghi sẽ được sắp xếp. RecordComparator định nghĩa thứ tự sắp xếp của các bản ghi. Nếu RecordFilter không được dùng thì tất cả các bản ghi trong lưu trữ bản ghi sẽ được dùng. Nếu RecordComparator không được dùng thì các bản ghi sẽ được trả về theo thứ tự ngẫu nhiên.

Bộ liệt kê có thể được thiết lập cập nhật khi các bản ghi thay đổi hoặc nó có thể được thiết lập bỏ qua các thay đổi và được cập nhật thủ công sau. Nếu sự liệt kê được cập nhật tự động mỗi khi thêm hoặc xóa bản ghi, thì nó có thể làm chậm hiệu suất của ứng dụng. Tuy nhiên, nếu các bản ghi bị xóa thì bộ liệt kê có thể trả về các bản ghi không hợp lệ nếu nó chưa được cập nhật. Giải pháp là đặt cờ các bản ghi đang được thay đổi và sau đó gọi phương thức rebuilt() để xây dựng lại bộ liệt kê một cách thủ công.

Các bản ghi duyệt bằng cách dùng phương thức nextRecord(). Lần đầu tiên được gọi nó sẽ trả về bản ghi đầu tiên trong tập liệt kê. Lần gọi kế tiếp nó sẽ trả về bản ghi kế tiếp theo thứ tự sắp xếp luận lý.

Ví dụ biểu diễn quá trình liệt kê bản ghi

```
IntegerFilter iFilt = new IntegerFilter();
IntegerCompare iCompare = new IntegerCompare();
RecordEnumeration intRecEnum = null;
intRecEnum = recordStore.enumerateRecords((RecordFilter)iFilt,
(RecordComparator)iCompare, false);
while (intRecEnum.hasNextElement()) {
byte b[] = intRecEnum.nextRecord();
}
```

```
// intRecEnum = recordStore(null, null, false);
```

Trong ví dụ trên, một đối tượng IntegerFilter và IntegerCompare được tạo ra. IntegerFilter sẽ chỉ trả về các bản ghi chứa trường số nguyên. IntegerCompare sẽ sắp xếp các bản ghi theo thứ tự số học.

Bộ liệt kê bản ghi được định nghĩa và được khởi tạo bằng output của phương thức enumerateRecords() của lớp RecordStore.

Phương thức enumerateRecords() có ba tham số. Tham số đầu tiên là tham chiếu đối tượng lọc (iFilt). Tham số thứ hai là tham chiếu đến đối tượng sắp xếp (iCompare). Tham số cuối cùng là một giá trị boolean xác định bộ liệt kê có được cập nhật khi các bản ghi thay đổi, thêm, xóa hay không.
Vòng lặp while() chỉ cách duyệt các bản ghi theo thứ tự yêu cầu. Vòng lặp while() sẽ tiếp tục miễn là bộ liệt kê còn chứa một bản ghi.

Dòng cuối cùng biểu diễn ví dụ cách duyệt tất cả bản ghi theo thứ tự ngẫu nhiên. Như ta thấy, các hai tham số lọc và so sánh đều được đặt là null.

Từng bước lập trình cho điện thoại di động J2ME - Phần 5

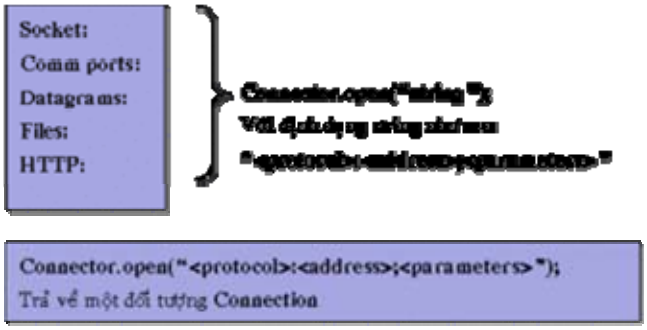
1 Lập trình mạng

1.1 Khung mạng CLDC tổng quát (Generic CLDC Networking Framework)

Mạng cho phép client di động gởi và nhận dữ liệu đến server. Nó cho phép thiết bị di động sử dụng các ứng dụng như tìm kiếm cơ sở dữ liệu, trò chơi trực tuyến… Trong J2ME, mạng được chia làm hai phần. Phần đầu tiên là khung được cung cấp bởi CLDC và phần hai là các giao thức thật sự được định nghĩa trong các hiện trạng.

CLDC cung cấp một khung tổng quát để thiết lập kết nối mạng. Ý tưởng là nó là đưa ra một khung mà các hiện trạng khác nhau sẽ sử dụng. Khung CLDC không định nghĩa giao thức thật sự. Các giao thức sẽ được định nghĩa trong các hiện trạng.

Hình 1 biểu diễn cách mà khung CLDC làm việc:



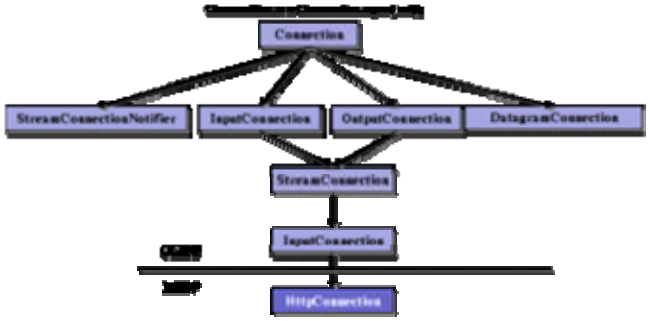
Hình 1. Khung mạng CLDC tổng quát
Kết nối mạng được xây dựng bằng phương thức open() của lớp Connector trong CLDC. Phương thức open() nhận một tham số đầu vào là chuỗi. Chuỗi này dùng để xác định giao thức. Định dạng của chuỗi là:
protocol:address;parameters

CLDC chỉ xác định tham số là một chuỗi nhưng nó không định nghĩa bất kỳ giao thức thật sự nào. Các hiện trạng có thể định nghĩa các giao thức kết nối như HTTP, socket, cổng truyền thông, datagram,… Phương thức open() trả về một đối tượng Connector. Đối tượng này sau đó có thể đóng vai trò là một giao thức xác định được định nghĩa trong hiện trạng.
Connector.open(":
;");
Một số giao thức ví dụ (nhưng không được hỗ trợ bởi CLDC hay MIDP):
Socket: Connector.open("socket://199.3.122.21:1511");
Comm port: Connector.open("comm:0;baudrate=9600");
Datagram: Connector.open("Datagram://19.3.12.21:1511");
Files: Connector.open("file:/filename.txt");
MIDP hỗ trợ giao thức HTTP:

HTTP: Connector.open("http://www.sonyericsson.com");
Trả về một đối tượng Connection
Ví dụ trên minh họa kết nối socket, cổng truyền thông, datagram, file và HTTP. Tất cả các kết nối mạng đều có cùng định dạng, không quan tâm đến giao thức thật sự. Nó chỉ khác nhau ở chuỗi chuyển cho phương thức open(). Phương thức open() sẽ trả về một đối tượng Connection đóng vai trò là lớp giao thức (ví dụ. HttpConnection) để có thể sử dụng các phương thức cho giao thức đó. J2ME chỉ định nghĩa một kết nối là kết nối HTTP trong MIDP.

1.2 Các lớp giao diện kết nối (Connection Interface Class)

Dẫn xuất từ lớp Connection là nhiều lớp giao diện con cung cấp khung kết nối mạng. Các giao diện khác nhau để hỗ trợ các loại thiết bị di động khác nhau.



Hình 2 . Các lớp kết nối
Sau đây là mô tả các giao diện kết nối được định nghĩa trong CLDC
StreamConnectionNotifier

Giao diệnStreamConnectionNotifier được dùng khi đợi một kết nối phía server được thiết lập. Phương thức acceptAndOpen() bị chặn cho đến khi client thiết lập kết nối.

Giao diện DatagramConnection
Kết nối datagram cung cấp kiểu truyền thông gói không chứng thực. Datagram chứa gói dữ liệu và địa chỉ. Chuỗi địa chỉ có định dạng sau:
datagram:[//{host}]:{port}
Nếu tham số host được xác định, thì datagram mở kết nối ở chế độ client. Nếu tham số host không được xác định, thì datagram được mở ở chế độ server
c = Connector.open("datagram://192.365.789.100:1234"); // Chế độ client
c = Connector.open("datagram://:1234"); // Chế độ server
Giao diện InputConnection
Giao diện InputConnection dùng để thực hiện một luồng nhập tuần tự dữ liệu chỉ đọc.
Giao diện OutputConnection
Giao diện OutputConnection dùng để thực hiện một luồng xuất dữ liệu chỉ viết.
Giao diện StreamConnection
Giao diện StreamConnection là kết hợp của cả hai giao diện InputConnection và OutputConnection. Nó dùng cho các thiết bị di động có truyền thông hai chiều.

Giao diện ContentConnection
Giao diện ContentConnection kế thừa giao diện StreamConnection và thêm vào các phương thức getType(), getEncoding(), và getLength(). Nó cung cấp cơ sở cho giao diện HttpConnection của MIDP.
Giao diện HttpConnection
Giao diện HttpConnection được định nghĩa trong MIDP và kế thừa giao diện ContentConnection của CLDC. Giao diện này cung cấp các phương thức thiết lập một kết nối HTTP.

1.3 Kết nối HTTP

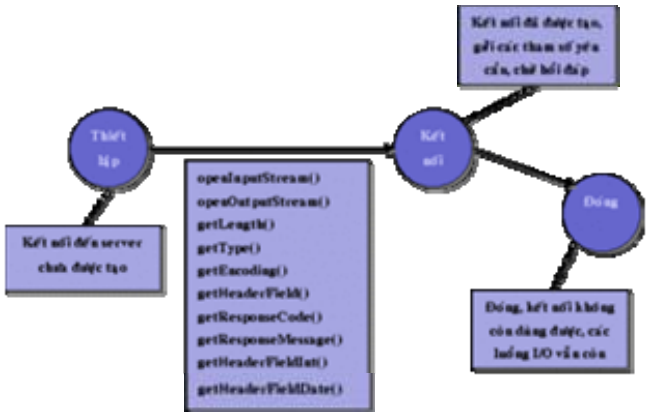
Hiện trạng MIDP hỗ trợ kết nối HTTP phiên bản 1.1 thông qua giao diện HttpConnection. Hỗ trợ GET, POST, HEAD của HTTP. Yêu cầu GET (GET request) được dùng để lấy dữ liệu từ server và đây là phương thức mặc định. Yêu cầu POST dùng để gởi dữ liệu đến server. Yêu cầu HEAD tương tự như GET nhưng không có dữ liệu trả về từ server. Nó có thể dùng để kiểm tra tính hợp lệ của một địa chỉ URL.

Phương thức open() của lớp Connector dùng để mở kết nối. Phương thức open() trả về một đối tượng Connection sau đó có thể đóng vai trò là một HttpConnection cho

phép dùng tất cả các phương thức của HttpConnection.
Một kết nối HTTP có thể ở một trong ba trạng thái khác nhau: Thiết lập (Setup), Kết nối (Connectd), hay Đóng (Close).

Trong trạng thái Thiết lập, kết nối chưa được tạo. Phương thức setRequestMethod() và setRequestProperty() chỉ có thể được dùng trong trạng thái thiết lập. Chúng được dùng để thiết lập phương thức yêu cầu (GET, POST, HEAD) và thiết lập thuộc tính HTTP (ví dụ. User-Agent). Khi sử dụng một phương thức yêu cầu gởi dữ liệu đến hay nhận dữ liệu về từ server sẽ làm cho kết nối chuyển sang trạng thái Kết nối. Gọi phương thức close() sẽ làm cho kết nối chuyển sang trạng thái Đóng.

Hình 3 minh họa các trạng thái kết nối khác nhau:



Hình 3 . Các trạng thái kết nối HTTP
Lưu ý rằng gọi bất kì phương thức nào liệt kê ở trên (ví dụ. openInputStream(), getLenght()) cũng sẽ làm cho kết nối chuyển sang trạng thái Kết nối.

1.4 Ví dụ HTTP GET

Phương thức HTTP GET cho phép lấy dữ liệu từ server và là phương thức mặc định nếu không xác định phương thức trong trạng thái Thiết lập.
Ví dụ thực hiện một kết nối HTTP GET cơ bản:

```
void getViaHttpConnection(String url) throws IOException {
HttpConnection c = null; InputStream is = null;
try {
c = (HttpConnection)Connector.open(url); // Mở kết nối HTTP
is = c.openInputStream(); // Mở Input Stream, mặc định GET
type = c.getType();
int len = (int)c.getLength();
if (len > 0) {
byte[] data = new byte[len];
int numBytes = is.read[data]; // Nếu biết chiều dài
processData(data);
} else {
int ch;
while ((ch = is.read()) != -1) { // đọc đến khi nào gặp -1
stringBuffer.append((char)ch);
}
processBuffer(stringBuffer);
```

```
}
} finally {
if (is != null) is.close();
if (c != null) c.close();
}
}
```

getViaHttpConnection() nhận một chuỗi là tham số đầu vào, đó là địa chỉ địa chỉ URL chuyển cho phương thức open() của lớp Connection. Phương thức open() trả về một đối tượng Connection đóng vai trò là một lớp HttpConnection. Phương thức openInputStream() sẽ làm cho kết nối chuyển sang trạng thái Kết nối. Vì không có yêu cầu phương thức nào, kết nối sẽ mặc định là một kết nối HTTP GET.
Phương thức getLength() sẽ trả về chiều dài của dữ liệu gởi từ server. Nếu biết được chiều dài, thì biến len sẽ chứa chiều dài dữ liệu và ta có thể đọc toàn bộ khối dữ liệu. Nếu không thì len sẽ chứa giá trị -1 và dữ liệu phải được đọc từng ký tự một cho đến khi gặp đánh dấu cuối file (-1). Phương thức processData() và processBuffer() xử lý dữ liệu đến từ server. Khối lệnh cuối cùng sẽ đóng tất cả các kết nối không quan tâm đến có lỗi từ khối lệnh try ở trước hay không.

1.5 Ví dụ HTTP POST

HTTP POST cho phép gởi dữ liệu đến server. Dữ liệu gởi đến server qua phương thức GET chỉ giới hạn là dữ liệu chứa địa chỉ URL. Phương thức POST cho phép gởi một luồng byte đến server. Phương thức HTTP POST thực hiện theo cách tương tự với phương thức HTTP GET.
Ví dụ thực hiện một kết nối HTTP POST:

```
void getViaHttpConnection(String url) throws IOException {
HttpConnection c = null; InputStream is = null;
OutputStream os;
try {
c = (HttpConnection)Connector.open(url); // Mở kết nối
// Thiết lập phương thức POST
// trong khi vẫn ở trạng thái Thiết lập
c.setRequestMethod(HttpConnection.POST);
// Mở luồng output stream và chuyển sang trạng thái Kết nối
os = c.openOutputStream();
// Chuyển đổi dữ liệu thành luồng byte
// và gởi đến server
os.write("Data Sent to Server\n".getBytes());
int status = c.getResponseCode();
// Kiểm tra status
if (status != HttpConnection.HTTP_OK) throw new IOException("not OK");
int len = (int)c.getLength();
// Giống như ví dụ HTTP GET:
// Kiểm tra length và xử lý tương ứng
} finally {
// Đóng kết nối giống như ví dụ HTTP GET
}
}
```

Như ví dụ trước, phương thức postViaHttpConnection() nhận tham số đầu vào là một chuỗi là địa chỉ URL được chuyển đến phương thức open() của lớp Connection. Phương thức open() trả về một đối tượng Connection đóng vai trò là một lớp HttpConnection.

Kết nối bây giờ ở trong trạng thái thiết lập và phương thức yêu cầu được đặt là POST bằng phương thức setRequestMethod(). Tất cả các thuộc tính khác phải được thiết lập trong trạng thái này.

Phương thức openOutputStream() sẽ làm cho kết nối chuyển sang trạng thái Kết nối. Phương thức write() và flush() sẽ gởi dữ liệu đến server.
Đoạn mã còn lại giống như phương thức GET. Luồng input được mở, chiều dài của dữ liệu được kiểm tra, và dữ liệu được đọc toàn bộ khối hay từng ký tự một tùy vào chiều dài được trả về. Khối lệnh cuối cùng sẽ đóng kết nối.

1.6 Triệu gọi CGI script

Cả hai phương thức GET và POST có thể được dùng để triệu gọi CGI script (Common Gateway Interface script) và cung cấp dữ liệu nhập. Ví dụ, một MIDlet có một form cho người dùng điền dữ liệu, sau đó có thể gởi dữ liệu kết quả cho server để CGI script xử lý. CGI script có thể được triệu gọi giống như phương thức GET và POST. Tên của CGI script và dữ liệu tham số nhập có thể chuyển trong địa chỉ URL. Nếu cần gởi thêm dữ liệu cho server, thì có thể dùng phương thức POST.

Ví dụ các tham số được gởi là một phần của URL:

url = http://www.asite.com/cgi-bin/getloca...=abc&zip=12345

Trong ví dụ trên, địa chỉ URL có thể được chuyển như là một tham số giống như phương thức getViaHttpConnection() ở ví dụ trước.

1.7 HTTP Request Header

Như ta đã nói trước, HTTP request header phải được thiết lập ở trạng thái Thiết lập bằng phương thức setRequestMethod() và setRequestProperty(). Phương thức setRequestMethod() dùng để thiết lập các phương thức GET, POST, hoặc HEAD. Phương thức setRequestProperty() dùng để thiết lập các trường trong request header. Ví dụ có thể là "Accept-Language", "If-Modified-Since", "User-Agent". Phương thức getRequestMethod() và getRequestProperty() có thể được dùng để lấy các thuộc tính trên.

2 Wireless Messaging API

J2ME chứa hầu hết các cấu hình và hiện trạng, kết hợp với nhau để định nghĩa môi trường thực thi Java hoàn chỉnh cho các thiết bị có tài nguyên giới hạn.

Tuy nhiên, đôi khi, cần phải có gói giao diện lập trình ứng dụng (Application Programming Interface – API), có thể chi xẻ bởi các ứng dụng chạy trên các hiện trạng khác nhau. J2ME định nghĩa API như vậy là các gói tùy chọn (optional package), là một tập các lớp và các tài nguyên khác có thể được dùng kết hợp với hiện trạng.

Cũng giống như các thành phần của J2ME, các gói tùy chọn được định nghĩa là yêu cầu đặc tả Java (Java Specification Request – JSR) thông qua Java Community Process. Một trong những gói tùy chọn đầu tiên cho J2ME là JSR 120, bộ API nhắn tin không dây (Wireless Messaging API – WMA), dùng để gởi và nhận các tin nhắn văn bản hoặc nhị phân ngắn trên kết nối không dây.

WMA dựa trên khung kết nối mạng tổng quát (GCF).

Các tin nhắn được gởi và nhận với WMA được gởi trên các mạng không dây của điện thoại di động và các thiết bị tương tự khác, có thể là GSM hay CDMA. WMA hỗ trợ Short Message Service (SMS) và Cell Broadcast Short Message Service (CBS). Mặc dù tin nhắn WMA tương tự như datagram, WMA không sử dụng giao diện datagram được định nghĩa bởi GCF, giao diện này dùng cho kết nối UDP. Thay vào đó, WMA định nghĩa một tập giao diện mới trong gói java.wireless.messaging.

Để gởi hoặc nhận tin nhắn, ứng dụng trước hết phải tạo một instance của giao diện MessageConnection, sử dụng GCF connection factory. Địa chỉ URL chuyển cho phương thức java.microedition.io.Connector.open() chỉ định giao thức sử dụng (SMS hoặc CBS), và số điện thoại đích, cổng, hoặc cả hai. Ví dụ, đây là những URL hợp lệ:

sms://+417034967891

sms://+417034967891:5678
sms://:5678
cbs://:5678

URL trong hai dạng đầu tiên mở kết nối client, ứng dụng kết nối đến một server với địa chỉ thiết bị và cổng chỉ định. Nếu cổng không chỉ định, sẽ dùng cổng nhắn tin mặc định của ứng dụng. Dạng URL thứ ba mở một kết nối server trên thiết bị, cho phép ứng dụng đợi và hồi đáp tin nhắn đến từ các thiết bị khác. Dạng cuối cùng cho phép ứng dụng lắng nghi tin nhắn broadcast từ người điều hành mạng.
Sau đây là một ví dụ đơn giản tạo một kết nối SMS client:

```
import java.microedition.io.*;
import java.wireless.messaging.*;
.....
MessageConnection conn = null;
String url = "sms://+417034967891";
try {
conn = (MessageConnection) Connector.open( url );
// thực hiện công việc gì đó
}
catch( Exception e ){
// xử lý lỗi
}
finally {
if( conn != null ){
try { conn.close(); } catch( Exception e ){}
}
}
```

Để gởi tin nhắn, sử dụng phương thức MessageConnection.newMessage() để tạo một tin nhắn rỗng, thiết lập payload của nó (dữ liệu văn bản hoặc nhị phân để gởi), và triệu gọi phương thức MessageConnection.send():

```
public void sendText( MessageConnection conn, String text )
throws IOException, InterruptedIOException {
TextMessage msg = conn.newMessage( conn.TEXT_MESSAGE );
msg.setPayloadText( text );
conn.send( msg );
}
```

Gởi dữ liệu nhị phân cũng hoàn toàn tương tự:

```
public void sendBinary( MessageConnection conn, byte[] data )
throws IOException, InterruptedIOException {
BinaryMessage msg =conn.newMessage( conn.BINARY_MESSAGE );
msg.setPayloadData( data );
conn.send( msg );
}
```

Dĩ nhiên, có giới hạn lượng dữ liệu có thể gởi trong một tin nhắn. Thông thường, tin nhắn văn bản SMS bị giới hạn đến 160 hoặc 70 ký tự, tin nhắn nhị phân bị giới hạn đến 140 bytes.
Nhận tin nhắn thậm chí còn đơn giản hơn: Sau khi mở một kết nối server, ứng dụng gọi phương thức receive() của kết nối, phương thức này sẽ trả về tin nhắn có trong cổng đã xác định. Nếu không có tin nhắn, phương thức sẽ đứng (block) cho đến khi có tin nhắn, hoặc cho đến khi có một thread khác đóng kết nối:

```
import java.io.*;
import java.microedition.io.*;
import java.wireless.messaging.*;
MessageConnection conn = null;
String url = "sms://:5678"; // không có số điện thoại!
try {
conn = (MessageConnection) Connector.open( url );
while( true ){
Message msg = conn.receive(); // blocks
if( msg instanceof BinaryMessage ){
byte[] data =((BinaryMessage) msg).getPayloadData();
// thực hiện công việc gì đó
} else {
String text =((TextMessage) msg).getPayloadText();
//thực hiện công việc gì đó
}
}
}
catch( Exception e ){
//xử lý lỗi
}
finally {
if( conn != null ){
try { conn.close(); } catch( Exception e ){}
}
}
```

Để bảo đảm tính ổn định của chương trình, việc gởi và nhận thông điệp nên giao cho một thread riêng đảm nhận.

Từng bước lập trình cho điện thoại di động J2ME - Phần 6

Lĩnh vực Ứng dụng không dây với công nghệ Java

Khái quát

Các ứng dụng Java cho các thiết bị không dây nhỏ ("MIDlet") sẽ đóng một vai trò – có thể là nhỏ, cũng có thể là lớn – trong các hệ thống phần mềm phân tán. Khi đó,

nó sẽ sinh ra một dạng phần mềm client mới. Chúng rất thích hợp với khái niệm thin-client, nhưng do chúng quá nhỏ, yêu cầu phải có thêm sự phối hợp làm việc hiệu quả với các thông tin được cung cấp bởi các servlet và JSP, và có thể là EJB ở đằng sau.

Ta sẽ xem xét các công nghệ Java chủ chốt để phát triển ứng dụng không dây trong hệ thống doanh nghiệp. Ta cũng sẽ xét đến các kiến trúc hỗ trợ client không dây trong các hệ thống doanh nghiệp.

Trong lúc này, dịch vụ Web (Web service), có thể sẽ trở thành một phương tiện vượt trội để hỗ trợ cho phần mềm client không dây trong một vài năm tới.

Các phiên bản Java 2

Nền tảng Java 2 được chia thành ba phiên bản, mỗi phiên bản hỗ trợ một dạng phần mềm trên các hệ thống khác nhau.

Phiên bản chuẩn, hay J2SE (Java 2 platform, Standard Edition), là phiên bản cũ nhất và thông dụng nhất. Nó hỗ trợ các ứng dụng Java, applet, lập trình desktop và các hệ thống lớn hơn – chủ yếu là cho PC - có thể có nối mạng hoặc không nối mạng. Người ta thông thường sử dụng J2SE cho các ứng dụng GUI đơn và console, các thành phần middleware và các dịch vụ RMI.

Phiên bản doanh nghiệp, hay J2EE (Java 2 platform, Enterprise Edition), mở rộng phiên bản chuẩn với các API có các "tính năng doanh nghiệp" (enterprise features). J2EE hỗ trợ Web service thông qua các servlet và JSP, dữ liệu bằng JDBC, và các hệ thống giao tác lớn thông qua EJB – đây là một vài công nghệ chính của J2EE. Các thành phần J2EE gắn chặt với phía server của các hệ thống lớn: khả năng xử lý mạnh, bộ nhớ và không gian lưu trữ lớn và có khả năng mở rộng.

Phiên bản mới nhất trong ba phiên bản là phiên bản thu nhỏ, hay J2ME (Java 2 platform, Micro Edition). Nó hỗ trợ các thiết bị "micro" đa dạng, mà J2ME gọi là các "hiện trạng" (profile) nhưng tất cả chúng đều kém khả năng hơn so với máy tính cá nhân. Trong J2ME, sức mạnh CPU, bộ nhớ, lưu trữ và khả năng kết nối đều bị hạn chế, có thể là rất nghiêm ngặt.

Sự cần thiết của J2ME

Thế giới của các thiết bị di động và các thiết bị "sub-PC" không có các đặc tính giống như trong lĩnh vực PC và server.

Ngoài ra, không phải mọi thiết bị trong lĩnh vực này đều cùng làm một việc. Sự khác nhau về thiết kế và mục đích giữa PDA, điện thoại, và máy nhắn tin là rất đáng kể.

Bất kể nó mang lại sự đổi mới gì cho thị trường, thì tính đa dạng của các thiết bị này là một ác mộng đối với các lập trình viên. Nếu tôi muốn xây dựng một ứng dụng cho điện thoại di động, tôi có phải viết mã lại, xây dựng lại, và kiểm tra lại cho mọi thiết bị hay không? Nếu tôi muốn xây dựng một client có kết nối mạng, tôi phải xét đến các công nghệ kết nối nào? v.v...

J2ME ra đời nhằm mục đích chính là thiết lập một chuẩn đơn mà thông qua đó các nhà phát triển có thể tạo nên các phần mềm có tính khả chuyển (portable) cho các thiết bị micro. Ngôn ngữ Java là sự lựa chọn đương nhiên cho lĩnh vực này, bởi vì về

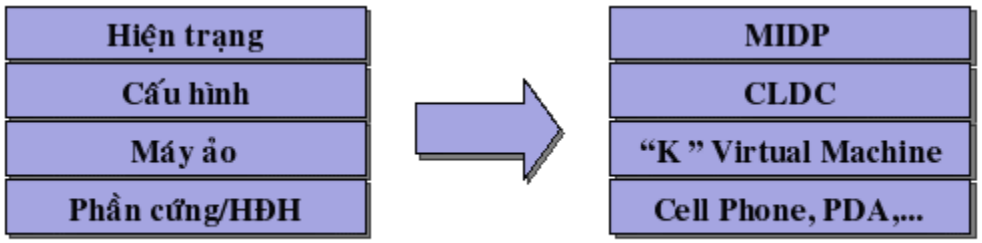
cơ bản nó đã hướng nhiều về tính khả chuyển. Bằng cách này, Sun đã đảm nhận bài toán lớn về tính đa dạng của thiết bị ở một mức tổng quát, do đó các nhà phát triển không phải quan tâm đến vấn đề này nữa. Nếu mọi nhà cung cấp PDA, điện thoại và máy nhắn tin đều thực hiện J2ME cho thiết bị của họ, thì chúng ta có khả năng viết chương trình "viết một lần, chạy mọi nơi" (write once, run anywhere) trong lĩnh vực micro, cũng giống như ta đã quen với khái niệm này ở các hệ thống máy lớn.
Hiện trạng thiết bị thông tin di động (Mobile Information Device Profile)

Mặc dù không phải chỉ có một hướng kiến trúc J2ME, nhưng các thiết bị di động không dây dường như dần dần càng quan tâm đến J2ME. Bao gồm:

* Điện thoại di động
* Trợ tá cá nhân số (Personal Digital Assistant-PDA)
* Máy nhắn tin
* Thiết bị đọc sách điện tử
* Các thiết bị point-of-sale

J2ME được tổ chức thành các mức, mỗi mức xác định một định nghĩa tăng dần của các thiết bị đích. Có nhiều lựa chọn kiến trúc tồn tại ở mỗi mức, và ràng buộc tùy chọn ở các mức cao hơn. Lập trình viên chỉ cần quan tâm đến hiện trạng (profile), định nghĩa các API; các nhà thực hiện J2ME cho thiết bị cần tập trung đến mức VM (Virtual Machine).

Hình 1. Các mức tổ chức J2ME

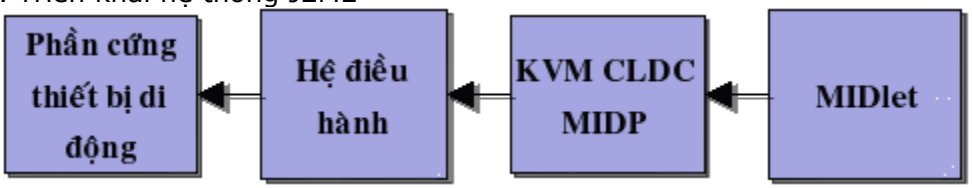


Các đặc tả cho các thiết bị không dây là Connected Limited Device Configuration hay CLDC, và Mobile Information Device Profile hay MIDP. MIDP định nghĩa các đặc tính tối thiểu của thiết bị như sau:

* Bộ nhớ không bay hơi có dung lượng 128K (nghĩa là, bộ nhớ có trạng thái được giữ lại khi thiết bị đã tắt) dành cho các thành phần MIDP, bao gồm KVM, Core API và chương trình MIDP.
* 8K bộ nhớ không bay hơi dành cho dữ liệu bền vững của ứng dụng.
* 32K bộ nhớ bay hơi cho bộ nhớ của chương trình.
* Màn hình hiển thị ít nhất là 96x54 pixel, có thể chỉ là một bit màu hay hỗ trợ nhiều màu hoặc màu mức xám.
* Cơ chế nhập liệu hỗ trợ ít nhất một bộ phím số, hoặc một màn hình cảm ứng có khả năng cấu hình hỗ trợ nhập liệu số.
* Khả năng kết nối mạng không dây hai chiều, với băng thông hạn chế và thông thường là không liên tục.

Như vậy các thiết bị hỗ trợ MIDP cung cấp một nền tảng chuẩn cho các phần mềm Java:

Hình 2. Triển khai hệ thống J2ME



Các kiểu ứng dụng MIDP
Các ứng dụng MIDP được gọi là các MIDlet. Hầu hết các MIDlet đều ở một trong hai dạng sau:

1. Ứng dụng đơn (standalone application) được nạp hoàn toàn vào thiết bị và có thể chạy bất kỳ lúc nào thiết bị được mở, không yêu cầu tài nguyên bên ngoài. Loại này bao gồm:

* Các ứng dụng PDA và ứng dụng organizer như sổ địa chỉ, danh sách công việc và lịch hẹn.
* Các công cụ đơn giản như máy làm tính (calculator)
* Trò chơi

2. Ứng dụng nối mạng (networked application) được chia thành ít nhất hai thành phần, một thành phần là client được triển khai trên thiết bị di động. Thành phần này sẽ ít được dùng nếu không có kết nối đến ít nhất một server trên hệ thống. Server thường là được đặt trong môi trường J2EE, và phục vụ bằng Web hoặc các giao thức Internet khác.

Ở đây, ta hãy xét kỹ thuật ngữ client. Ta không gọi một MIDlet là một client chỉ đơn giản là vì nó sử dụng kết nối mạng MIDP và liên lạc đến các thành phần khác. Câu hỏi là phần luận lý lõi của ứng dụng đặt ở đâu? MIDlet có đảm nhận hầu hết việc "suy nghĩ" và chỉ quan tâm đến mạng hay không? Đó không phải là client, thật vậy – ít nhất là không theo đúng nghĩa trong ngữ cảnh của hệ thống enterprise. Một client là một MIDlet dựa vào một server để suy nghĩ, lưu trữ, tải, xử lý, hay nói cách khác là làm việc thay cho nó.

Java 2 Enterprise Edition
Các MIDlet client không yêu cầu phải kết nối đến các server chạy Java. Một MIDlet có thể được viết để tạo HTTP request đến một trang web đã có từ trước, và nó không cần quan tâm là trang web đó được hỗ trợ bởi ASP trên IIS, hay servlet trên Apache/Tomcat,... Tuy nhiên, trên thực tế, khi toàn bộ hệ thống phân tán được phát triển mới, thì Java nên được dùng ở mọi mức.

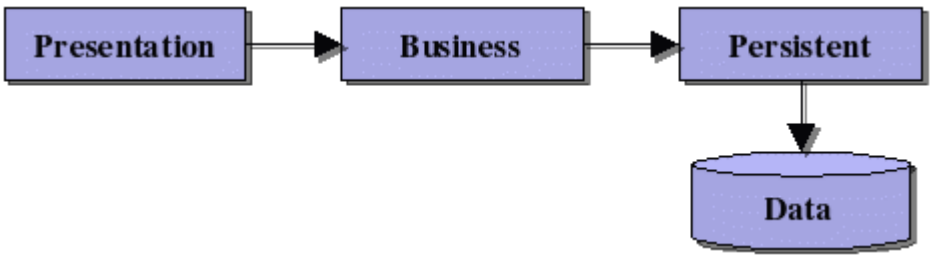
Phiên bản Java doanh nghiệp, Java 2 Enterprise Edition, hay J2EE – là một tập các chuẩn để áp dụng công nghệ Java cho các hoạt động "loại doanh nghiệp (enterprise-class)", ví dụ như:

* Dịch vụ HTTP, bao gồm ứng dụng Web và dịch vụ Web (Web service)
* Lưu trữ và lấy dữ liệu từ cơ sở dữ liệu quan hệ
* Xử lý giao tác trực tuyến
* Thực hiện đối tượng phân tán (bằng CORBA)
* Truyền thông điệp tin cậy giữa server và các tiến trình
* Xử lý tài liệu XML

Ta xét thuật ngữ Enterprise software (phần mềm doanh nghiệp). Đây là một thuật ngữ được định nghĩa không chặt. Nói chung, ta định nghĩa các hệ thống mức doanh nghiệp bằng các yêu cầu và nhu cầu khi thực thi.

* Trong bất kỳ lĩnh vực và mức nào, các hệ thống doanh nghiệp thường phải chịu áp lực rất cao: xử lý hay lưu trữ nhiều dữ liệu, xử lý nhiều yêu cầu, thường là thường xuyên, nhiều công việc phải làm cho client. Hệ thống phải có khả năng nâng cấp, và phải hoạt động có hiệu quả dưới áp lực cao.
* Hệ thống phải có tính sẵn sàng (available).
* Quản lý dữ liệu ứng dụng phải thỏa mãn tất cả tính chất của giao tác ACID: atomicity (tính nguyên tử), consistency (tính toàn vẹn), isolation (tính tách biệt), và durability (tính bền vững). Nói chung, điều này có nghĩa là server phải hỗ trợ một chuẩn tin cậy rất cao trong việc xử lý dữ liệu.
* Các chức năng dữ liệu và ứng dụng phải an toàn (secure): điều này bao gồm cần phải có xác thực, và chính sách cấp quyền.
* Truyền thông điệp giữa các thành phần phải đáng tin cậy (reliable) – điều này cũng giống như tính ACID của giao tác, nhưng ở đây ta áp dụng cho các thông điệp của ứng dụng.

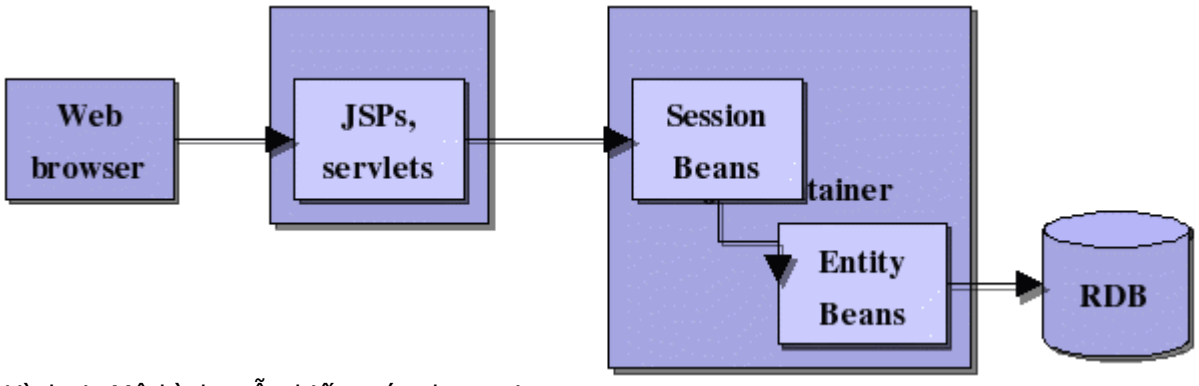
Kiến trúc Ba-tầng (Three-tier)
Một ứng dụng J2EE nên thực hiện theo kiến trúc ba tầng (three-tier architecture), bởi vì nó sẽ phân chia rõ ràng trách nhiệm cho từng tầng khác nhau trong mô hình ứng dụng.



Hình 3. Kiến trúc three-tier

* Tầng trình diễn (presentation tier) chỉ đảm nhận phần biểu diễn thông tin đến server và thu thập dữ liệu nhập của người dùng. Nó không biết hoặc không quan tâm đến cách mà thông tin được phát sinh, mặc dù nó biết một số điều về "hình dạng (shape)" của thông tin.
* Tầng luận lý nghiệp vụ (business logic tier) (hay đôi khi còn gọi là "domain", hay đơn giản là "tầng giữa (middle tier)" đảm nhận chức năng lõi của ứng dụng: các tính năng và các hàm để biên dịch hay thay đổi dữ liệu, các luật phải được áp dụng cho dữ liệu khi nó thay đổi. Tầng này cung cấp cho tầng trình diễn trước nó, và cũng là phương tiện cho việc lưu trữ và nhận dữ liệu của tầng sau nó.
* Tầng persistent quản lý lưu trữ bền vững và lấy dữ liệu ứng dụng. Tầng này có thể bao gồm mã chương trình cộng với hệ quản trị cơ sở dữ liệu quan hệ.

Mô hình mẫu có thể biểu diễn như Hình 4:

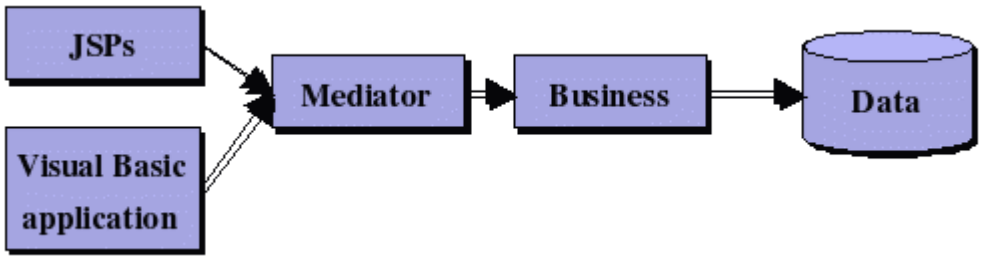


Hình 4. Mô hình mẫu kiến trúc three-tier

* JavaServer page và servlet, quản lý bởi một Web server J2EE, xác định tầng trình diễn – đây là giao diện do server quản lý.
* Một lớp xác định của Enterprise JavaBean được gọi là session bean thực hiện logic nghiệp vụ.
* JDBC là một loại khác của EJB, entity bean, quản lý dữ liệu trên các RDBMS.

Tuy nhiên client không dây (wireless client) là một dạng client đặc biệt. Nó cần phải được server phục vụ đặc biệt: dữ liệu phải được xử lý đặc biệt cho loại client này. Hỗ trợ các thiết bị MIDP thông qua tầng môi giới (Mediation)

Việc chuẩn bị đặc biệt dữ liệu từ tầng giữa để cho một dạng trình diễn đặc biệt được gọi là sự môi giới (mediation). Tầng môi giới (mediator tier) là một tính năng thông thường của các hệ thống N-tầng, thường được triển khai để hỗ trợ việc dùng nhiều khung (framework) trình diễn khác nhau cho cùng một tầng domain.



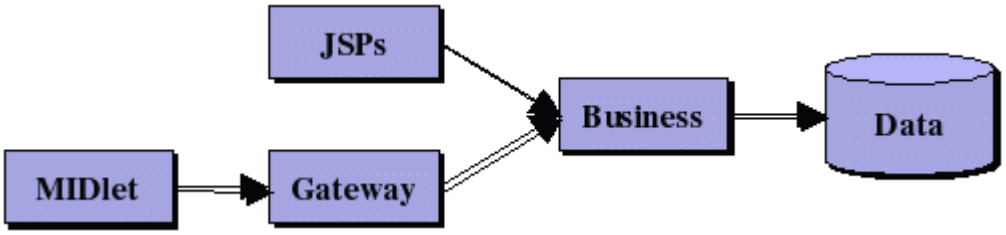
Hình 5. Vị trí của tầng môi giới

Đối với các MIDP client, sự môi giới thường là ở dạng một gateway, biên dịch nội dung mức PC sang nội dung mức micro, và có thể xử lý chuyển đổi giao thức, ví dụ như:

* Nội dung HTML có thể được biên dịch thành Wireless Markup Language, hay WML
* Giao thức cơ bản có thể chuyển từ HTTP sang Wireless Application Protocol hay WAP
* Các datagram sẽ không được cung cấp bằng User Datagram Protocol (UDP) mà bằng

Wireless Datagram Protocol hay WDP.
Kiến trúc cuối cùng sẽ là một trong hai biến thể của kiến trúc N-tầng của kiến trúc J2EE mà ta đã thấy ở trên.

* Mediation của domain:



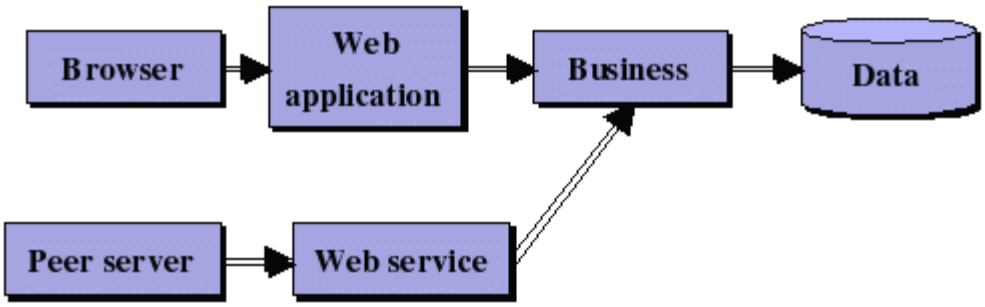
Hình 6. Môi giới của tầng domain

* Mediation/Translation của tầng trình diễn:

Hình 7. Môi giới của tầng trình diễn

MIDP client sẽ dựa nhiều vào phần mềm J2EE và các gateway hay tầng môi giới để đơn giản hóa hay định dạng nội dung cho việc trình diễn và xử lý ở người dùng di động.
Các dịch vụ Web (Web service)

Ứng dụng Web dần dần đang chia sẻ với dịch vụ Web, là một thành phần cung cấp truy xuất programmatic trực tiếp đến tầng business/domain, nhưng vẫn sử dụng các giao thức Web để chấp nhận và phục vụ yêu cầu.



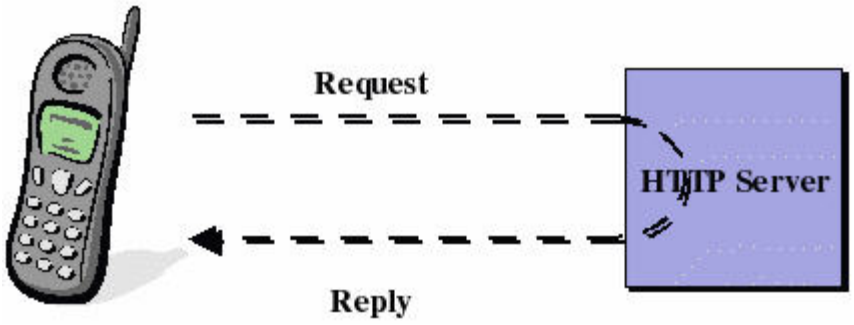
Hình 8. Dịch vụ Web

Các MIDlet có thể là các client của các dịch vụ Web, nhưng vẫn cần phải có sự môi giới.

Có hai nỗ lực đế hỗ trợ MIDP truy xuất các dịch vụ Web:

* Một tập con của Java API for XML Processing đang được đưa vào MIDP 2.0 API
* Một đặc tả một Web-service gateway đang được phát triển, sẽ tránh việc xử lý XML trong MIDlet.

Lập trình Web Service với MIDP
Lập trình mạng MIDP trên HTTP client
Khái quát
Đặc tả MIDP 1.0 phát biểu rằng các triển khai của MIDP trên thiết bị di động bắt buộc phải hỗ trợ ít nhất là kết nối HTTP 1.1 sử dụng khung kết nối chung (GCF – Generic Connection Framework). Sử dụng kết nối client HTTP 1.1 nghĩa là thiết bị gởi một yêu cầu (request) và server gởi về một hồi đáp (response) tương ứng.



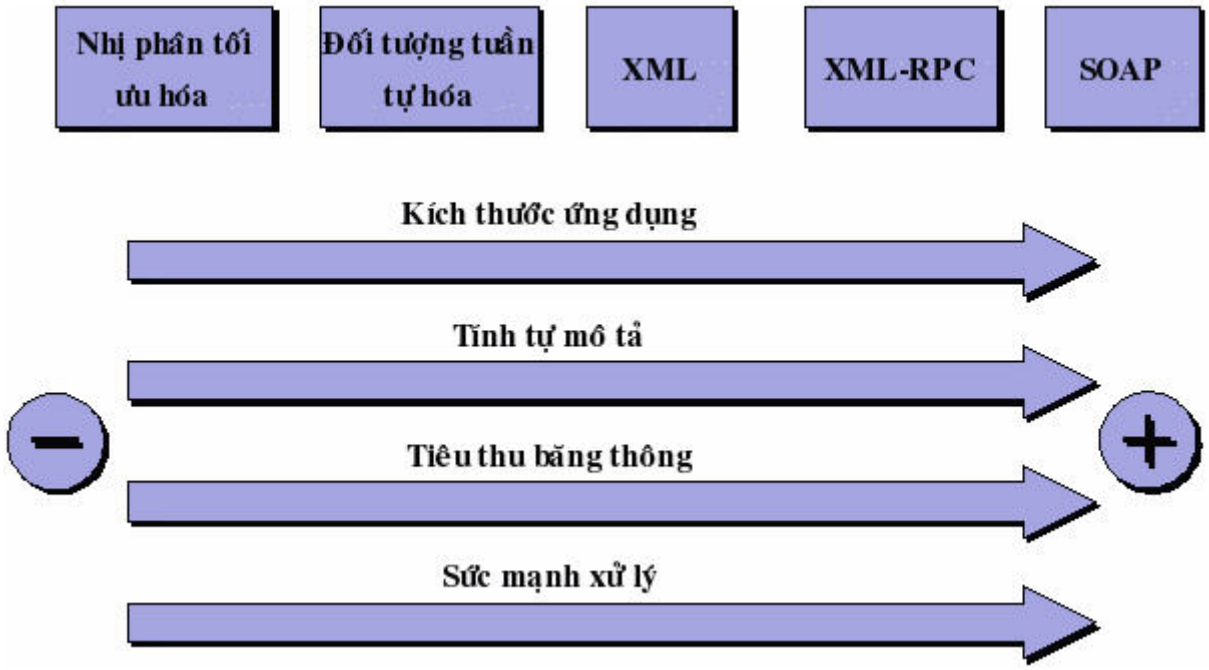
Hình 1. HTTP client request-response
Bằng cách chỉ dùng kết nối HTTP client nghĩa là server không thể thiết lập liên lạc với thiết bị ngoại trừ bằng cách hồi đáp một request. Một MIDlet HTTP client thông thường sẽ dùng cả hai phương thức HTTP GET và POST.
Đặc tả MIDP 2.0 phát biểu rằng cả HTTP và HTTPS bắt buộc phải được hỗ trợ.
Thân của thông điệp HTTP
Thông tin gởi trong thân thông điệp HTTP request và response đơn giản là một luồng byte. MIDlet và servlet chọn kiểu định dạng thông tin để mã hóa các byte này.
Thân của thông điệp SOAP/HTTP
Các điểm cuối dịch vụ Web dựa trên SOAP trao đổi các thông điệp SOAP với nhau. HTTP là một cơ chế mặc định dùng để truyền thông điệp SOAP. Thông điệp SOAP chứa dữ liệu theo định dạng XML. Thông điệp XML có thể dùng cả UTF-8 hay UTF-16 để làm bảng mã và mã hóa.
Khái quát về dịch vụ Web (Web service), SOAP và WSDL
Thuật ngữ "Dịch vụ Web" (Web service) nói đến truyền thông ứng dụng-đến-ứng dụng (application-to-application). Một dịch vụ Web đơn giản là một dịch vụ trên Internet có khả năng được truy xuất thông qua giao diện theo khuôn dạng sử dụng các giao thức Internet chuẩn như HTTP.
World Wide Web Consortium (W3C) định nghĩa dịch vụ Web như sau:
Một dịch vụ Web là một hệ thống phần mềm được nhận dạng bằng một URI (Uniform Resource Identifier), mà các giao diện chung và sự gắn kết của nó được định nghĩa và mô tả bằng XML. Định nghĩa của nó có thể được nhận ra bằng các hệ thống phần mềm khác. Các hệ thống này sau đó có thể tương tác với dịch vụ Web theo phương cách được mô tả trong định nghĩa của nó, sử dụng các thông điệp theo XML được chuyển bằng các giao thức Internet.
Hai đặc tả quan trọng về dịch vụ Web là Ngôn ngữ mô tả dịch vụ Web (Web Services Description Language – WSDL) và Giao thức truy xuất đối tượng đơn giản (Simple Object Access Protocol – SOAP). WSDL được dùng để mô tả một dịch vụ Web đã được triển khai. SOAP được dùng để định nghĩa định dạng của thông điệp được trao đổi giữa các điểm cuối (thí dụ như client và server) của dịch vụ Web trong suốt quá trình hoạt động của dịch vụ Web đó. Một dịch vụ Web có thể tự đăng ký ở một nơi đăng ký thích hợp (ví dụ bằng cách cung cấp mô tả WSDL của nó) để client có thể nhận ra nó. Các tiến trình này được gọi là quá trình đăng ký và nhận biết dịch vụ.
Java, Web service và SOAP
Lĩnh vực dịch vụ Web đang phát triển nhanh chóng. Tại thời điểm này Ủy ban công

nghệ Java (Java Techonology Community) đã xây dựng phiên bản đầu tiên của Java API cho RPC dựa trên XML (Java API for XML-based RPC – JAXRPC) cho J2SE. Một gói tùy chọn cho dịch vụ Web trên J2ME cũng đang được xây dựng.
Đặc tả MIDP 1.0 và MIDP 2.0 không xác định bất kỳ hỗ trợ nào cho XML hay SOAP. Các nhà phát triển MIDP muốn sử dụng XML hay SOAP thường phải sử dụng các thư viện bên ngoài. Điều này rất bất lợi vì mỗi MIDlet phải chứa các thư viện này. Các thư viện như vậy thường khoảng 25 đến 50 KB (kích thước file .class). Điều này có khả năng sẽ làm giảm không gian cho ứng dụng MIDlet.
Luận án này được phát triển bằng các thư viện mở KXML và KSOAP. Một vài thư viện XML và SOAP khác nhắm đến thiết bị J2ME cũng có thể dễ dàng được tìm thấy, và có thể được sử dụng theo phương cách tương tự.
Tối ưu hóa truyền thông Client/Server cho các ứng dụng di động
Ứng dụng di động client/server
Ngoài các ứng dụng chạy đơn trên thiết bị di động không cần tương tác với tài nguyên bên ngoài, còn có nhu cầu một môi trường phân tán với client có nhu cầu liên lạc với server sử dụng kết nối IP. Ta sẽ xét một số vấn đề điển hình về liên lạc client/server có thể phát sinh trong quá trình kết nối giữa Java 2 Platform, Enterprise Edition (J2EE), nền tảng server và MIDlet. Tiếp theo sẽ so sánh các giao thức khác nhau, có thể được dùng để phát triển các loại ứng dụng phân tán này.
Ngoài ra, lập trình viên có thể sử dụng thêm các tầng trừu tượng giữa giao thức chuyển vận, dựa trên HTTP, và chính ứng dụng để xây dựng một kiến trúc linh động có thể được tối ưu hóa. Với cách tiếp cận này, giao thức chuyển vận được chọn có thể được chuyển đổi tương đối dễ dàng mà không cần phải hiệu chỉnh logic của ứng dụng.
Ở đây ta sẽ dùng một proxy servlet để có thể nâng cao hiệu quả của các ứng dụng di động client/server.
Trên thực tế, vô số ứng dụng Mobile Information Device Profile (MIDP) không chỉ chạy trên các thiết bị di động, mà cũng có truy xuất đến server, và do đó thể hiện một ứng dụng phân tán. Nhiều ứng dụng di động chỉ thật sự hoạt động khi kết nối đến server. Kết nối có thể "luôn luôn mở (always on)" hay chỉ mở khi ứng dụng cần liên lạc với server. Sử dụng cách tiếp cận phân tán, ứng dụng di động có thể truy xuất đến các cơ sở dữ liệu ngoại, vì những công việc quá phức tạp đối với khả năng hạn chế của thiết bị MIDP có thể được chuyển đến cho một server mạnh hơn. Do đó, lời giải cho ứng dụng di động doanh nghiệp chỉ có thể thực hiện thông qua tương tác giữa J2EE và Java 2 Platform, Micro Edition (J2ME). Tuy nhiên, trong quá trình trao đổi dữ liệu giữa server và client di động, cần phải quan tâm đến các vấn đề liên quan, đặc biệt là các vấn đề liên quan đến hiệu suất truyền tải và xử lý dữ liệu trên thiết bị.
Đối với giải pháp doanh nghiệp dựa trên công nghệ J2ME, cần phải quan tâm đến sự hạn chế của cả kết nối mạng và tài nguyên của thiết bị, không giống như môi trường thông thường của máy tính cá nhân với kết nối mạng cố định. Điều này có nghĩa là nhà phát triển nên lường trước được các khoảng thời gian trễ dài trên băng thông hạn chế. Hơn nữa, bất kỳ trong tình huống nào cũng không nên cho rằng thiết bị di động luôn luôn có kết nối. Về tài nguyên, ta phải đối mặt với vấn đề khả năng tính toán hạn chế cùng với khả năng lưu trữ tương đối của thiết bị. Do đó, trước khi phát triển một ứng dụng phân tán cho client di động, ta cần phải xem xét kỹ các yếu tố trước khi chọn giao thức, bởi vì quyết định này có thể có ảnh hưởng lớn đến hiệu suất của ứng dụng.
HTTP là một giao thức liên lạc client/server lý tưởng cho ứng dụng Java di động. Đối với mỗi đặc tả, thiết bị tương thích MIDP 1.0 phải hỗ trợ HTTP. Các giao thức khác như TCP hay UDP là tùy chọn. Bởi vì không phải tất cả thiết bị MIDP đều hỗ trợ truyền thông socket hay datagram, do đó triển khai HTTP trên thiết bị di động cho phép tối ưu khả năng chuyển đổi giữa các thiết bị từ các nhà sản xuất khác nhau.

Mặc dù một số thiết bị, như Nokia 6800 hỗ trợ kết nối socket, nhưng để tương thích tối đa, nên sử dụng HTTP làm giao thức trao đổi giữa client và server.
Một lợi điểm khác nữa là giao thức HTTP được hưởng truy xuất không lỗi (trouble-free access) thông qua tường lửa. Bởi vì server và client di động hầu như được tách biệt bằng firewall, HTTP không cần phải cấu hình thêm. Mặc dù vậy, ta cũng nên qua tâm đến các rủi ro bảo mật có thể có khi mở kết nối HTTP ra thế giới bên ngoài. Java cung cấp API lập trình mạng, hỗ trợ giao thức HTTP 1.1. Ta dễ dàng tạo ra các request GET, POST, và HEAD trong ứng dụng Java.

Các loại giao thức khác nhau

Bây giờ ta đã chọn HTTP làm giao thức chuyển vận, vai trò của người phát triển là phải quyết định định dạng thông điệp để trao đổi dữ liệu giữa server và client. Nền tảng J2ME không đưa ra các cơ chế đã được chuẩn hóa như Java Remote Method Invocation (RMI) và Java API for XML-based Remote Procedure Call (JAX-RPC) (vốn rất tốn tài nguyên), người phát triển phải tự mình định nghĩa định dạng và lớp truyền thông trên lớp chuyển vận HTTP. Có nhiều sự lựa chọn, ta sẽ xem xét chi tiết dưới đây.
Chủ yếu có hai cách định nghĩa định dạng thông điệp: Một là định dạng thuần nhị phân, được tối ưu hóa để bảo đảm hiệu suất cao nhất. Hai là định dạng phức tạp dựa trên XML, ví dụ như SOAP, cung cấp khả năng đọc và khả chuyển cao, nhưng hiệu suất rất kém, đặc biệt là với các thiết bị di động với băng thông và tốc độ xử lý hạn chế. Người phát triển phải đối mặt với thử thách là phải lựa chọn giải pháp tốt nhất cho ứng dụng. Về cơ bản, kích thước của giao thức tăng tương ứng với khả năng tự mô tả, do đó làm giảm hiệu quả truyền thông trên mạng điện thoại di động băng thông hẹp. Tăng khả năng human-readability, thì đồng thời cũng gia tăng các định dạng dựa trên XML, cũng như hiệu suất tính toán để phát sinh và phân tích các thông điệp đến.

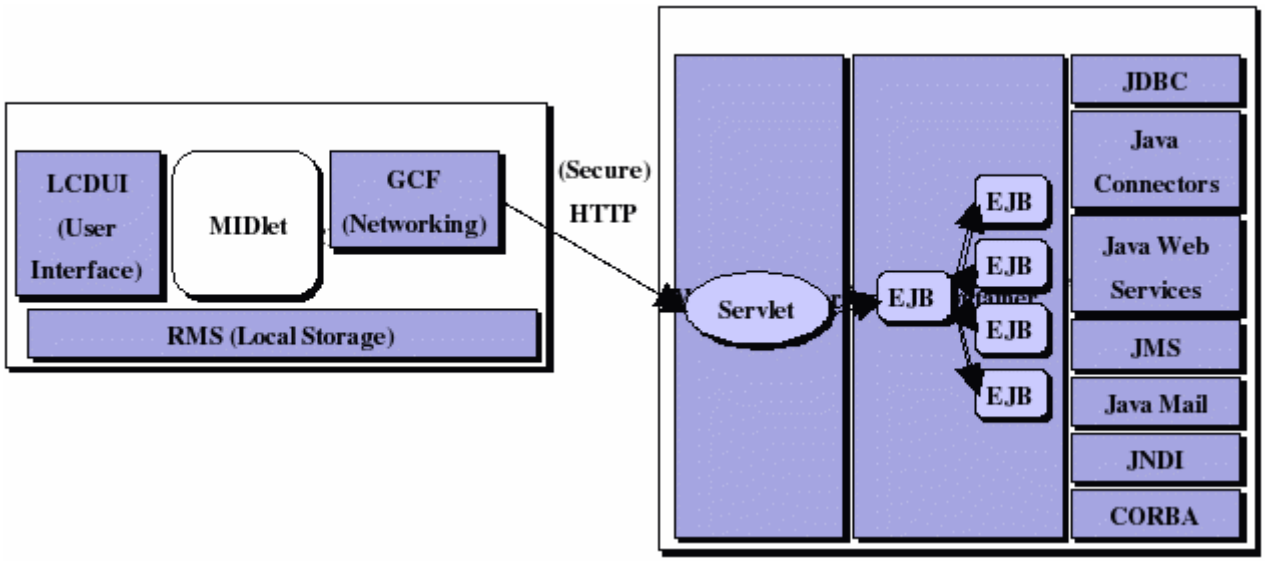


Hình 2. Biểu đồ so sánh các giao thức liên lạc khác nhau
Định dạng nhị phân độc quyền (Proprietary Binary Format)

Định dạng này có khả năng linh động cao và có thể được phát sinh một cách dễ dàng

Các vấn đề thiết kế ứng dụng Enterprise không dây áp dụng công nghệ Java
Mô hình lập trình cơ bản

Hình 1 biểu diễn cấu trúc tổng quát của một ứng dụng enterprise không dây điển hình, bao gồm một thiết bị J2ME và một server J2EE.



Hình 1. Kiến trúc mức cao của một ứng dụng enterprise Java không dây.

Kiến trúc của một ứng dụng enterprise phục vụ các client không dây cũng tương tự như của một ứng dụng J2EE chuẩn:

Một client ứng dụng sử dụng MIDP hay được gọi là MIDlet client, cung cấp giao diện người dùng trên thiết bị di động. MIDlet giao tiếp với một Java servlet, thường là thông qua HTTP, và trên một kênh truyền bảo mật khi cần thiết.

Servlet dịch yêu cầu từ MIDlet, và tới lượt nó, gởi yêu cầu đến các thành phần EJB. Khi các yêu cầu được thỏa mãn, servlet phát sinh một hồi đáp cho MIDlet.

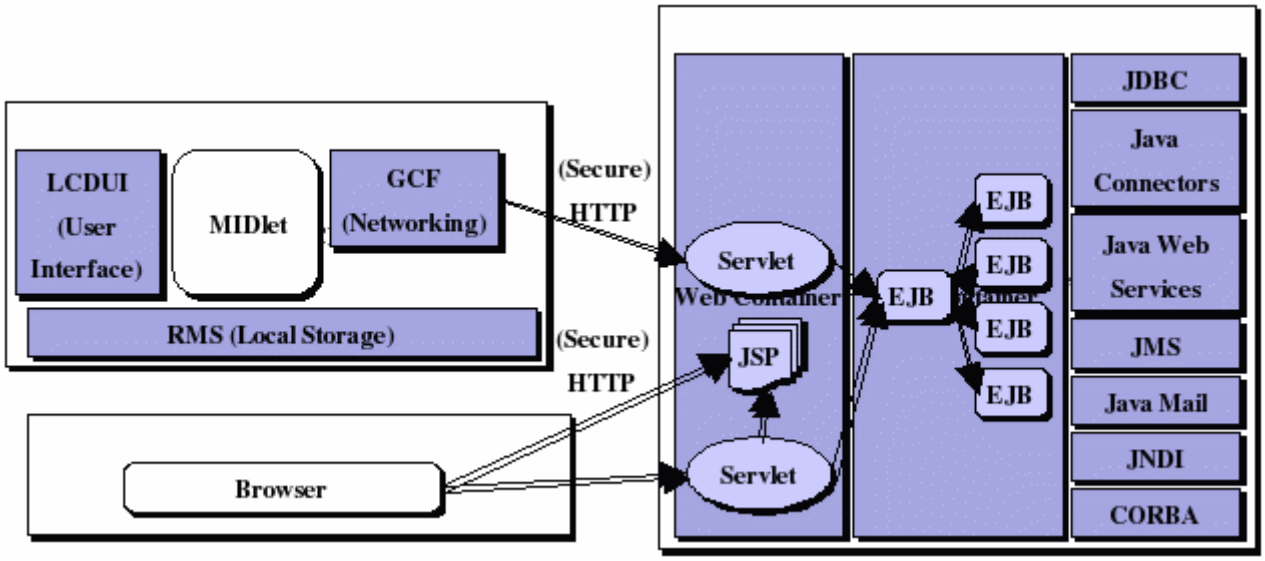
Các thành phần EJB, hay các enterprise beans, bao bọc logic nghiệp vụ của ứng dụng. Một trình chứa EJB cung cấp các dịch vụ chuẩn như giao tác, bảo mật, và quản lý tài nguyên để các nhà phát triển có thể tập trung vào việc thực hiện logic nghiệp

vụ.

Các thành phần servlet và EJB có thể sử dụng các API bổ sung để truy xuất dữ liệu và dịch vụ. Ví dụ, chúng có thể sử dụng JDBC API để truy xuất cơ sở dữ liệu quan hệ, hay JavaMail API để gởi e-mail cho người dùng.

Hỗ trợ nhiều loại client

Nền tảng J2EE nhấn mạnh vào các thành phần có thể tái sử dụng. Ứng dụng có thể dùng các thành phần này để hỗ trợ nhiều loại client mà không (hay ít) ảnh hưởng đến logic nghiệp vụ chính của ứng dụng. Hình 2 biểu diễn kiến trúc của một ứng dụng với client J2ME và client trình duyệt.



Hình 2. Kiến trúc mức cao của một ứng dụng J2EE hỗ trợ client J2ME và client trình duyệt

Các vấn đề khi thiết kế và thực hiện
Ta xem xét một số vấn đề khi thiết kế và thực hiện các ứng dụng enterprise không dây.

Xây dựng ứng dụng không dây có những ràng buộc đặc thù. Và khi thiết kế các ứng dụng không dây, ta sẽ gặp phải ba vấn đề sau: ràng buộc thiết kế (design

constraint), thông điệp (messaging), và trình diễn (presentation).

Ràng buộc thiết kế (Design Constraint)
Hạn chế của các thiết bị di động dẫn đến nhiều ràng buộc khi thiết kế các ứng dụng không dây. Các ứng dụng này phải cung cấp các giao diện có ích và tiện lợi trong khi phải đối mặt với kích thước màn hình, khả năng nhập liệu, sức mạnh xử lý, bộ nhớ, lưu trữ, và thời gian sử dụng nguồn pin bị hạn chế.

Nhất là các ứng dụng enterprise không dây càng bị ràng buộc, bởi vì chúng dựa vào mạng. Các hạn chế do mạng di động ảnh hưởng đến ứng dụng di động nhiều hơn so với trình duyệt Web thông thường. Nói chung, các thiết bị di động sẽ gặp phải các vấn đề sau:

Độ trễ cao

Băng thông hạn chế

Kết nối không liên tục

Để giải quyết các ràng buộc này, client MIDP có thể sử dụng các cách sau:

Chỉ kết nối vào mạng khi cần thiết.

Chỉ sử dụng dữ liệu đúng mức cần thiết.

Phải có khả năng sử dụng khi đã ngắt kết nối.

Truyền thông điệp
Mặc dù MIDP không có các cơ chế truyền thông client/server phức tạp, như Java Remote Method Invocation (RMI) hay Java API for XML-based Remote Procedure Calls (JAX-RPC), các nhà phát triển vẫn có thể thiết kế một giao thức truyền thông điệp sử dụng định dạng và cách trao đổi theo ý mình.

Đối với hầu hết các ứng dụng, HTTP xứng đáng là một giao thức truyền thông điệp cơ bản, và nó được ưa chuộng hơn so với các phương thức truyền thông khác (ví dụ như dựa trên socket hay datagram) vì các lý do sau đây:

Tất cả các thiết bị MIDP phải hỗ trợ lập trình mạng MIDP. Do đó, các ứng dụng chỉ dựa vào HTTP sẽ có tính khả chuyển trên các thiết bị khác nhau. Mặt khác, không phải tất cả các thiết bị MIDP đều hỗ trợ truyền thông dựa trên packet hay datagram, do đó các ứng dụng sử dụng các phương thức này không bảo đảm tính khả chuyển.

HTTP có khả năng bảo mật tường lửa (firewall). Hầu hết server được tách biệt khỏi client di động bằng firewall, và HTTP là một trong số ít các giao thức mà hầu hết các firewall đều cho phép đi qua.

Các API lập trình mạng của Java làm cho lập trình HTTP dễ dàng hơn. MIDP hỗ trợ HTTP 1.1 và các API để phát sinh các GET, POST và HEAD request, các thao tác header cơ bản, và cơ chế luồng cho thông điệp. Trong khi đó, API cho Java servlet, cung cấp khả năng xử lý HTTP request và sinh các HTTP response khá mạnh.

Khi một MIDP client liên lạc với một Java servlet thì các sự việc sau xảy ra:

Client mã hóa application request và đóng gói nó vào một HTTP request. Các Content-Type và Content-Length header phải được thiết lập để bảo đảm các gateway trung gian xử lý request đúng đắn.

Servlet nhận HTTP request và giải mã application request. Server hay một thành phần khác (ví dụ như enterprise bean) thực hiện công việc xác định bởi application request.

Servlet mã hóa application response và đóng gói nó vào một HTTP response. Content-Type và Content-Length header cũng phải được thiết lập đúng để bảo đảm các gateway trung gian xử lý response đúng đắn.

Client nhận HTTP response và giải mã application response chứa trong đó. Client có thể thiết lập một hoặc nhiều đối tượng và thực hiện một số công việc trên các đối tượng cục bộ này.

Thiết kế định dạng thông điệp (Message Format)
Cách định dạng application request và response là tùy thuộc vào lập trình viên. Các lựa chọn rơi vào hai cách sau:

Một cách là dùng định dạng nhị phân. Các thông điệp nhị phân có thể được đọc và ghi sử dụng các lớp DataInputStream và DataOutputStream trong gói java.io.

Trên thực tế, sử dụng các thông điệp này đạt được hiệu quả trao đổi bởi vì tải được rút gọn. Chú ý rằng để tiết kiệm không gian, các thông điệp phải thỏa mãn tính tự miêu tả (self-descriptive). Do đó, định dạng của thông điệp phải được biết cả ở client và server, và do đó chúng gắn chặt với nhau.

Cách khác là sử dụng Extensible Markup Language (XML). Trong khi nền tảng J2EE cung cấp rất nhiều hỗ trợ cho XML (đặc biệt là trong Web service), thì đặc tả MIDP không yêu cầu hỗ trợ XML, mặc dù các nhà phát triển có thể thêm hỗ trợ XML vào ứng dụng MIDP bằng cách kết hợp các thư viện bổ sung.

Để phân tích và xử lý tài liệu XML, các nhà phát triển có thể lựa chọn nhiều cách thực hiện, bao gồm hai mô hình xử lý phổ biến, Document Object Model (DOM) và Simple API for XML (SAX). (SAX và các bộ phân tích dựa trên sự kiện khác thích hợp hơn DOM khi áp dụng cho các thiết bị di động với bộ nhớ và tốc độ xử lý bị hạn chế). Các thư viện RPC dựa trên XML cũng được cung cấp, bao gồm các bộ phân tích dựa trên đặc tả Simple Access Object Protocol (SOAP).

Tuy nhiên khi sử dụng định dạng XML, ngoài việc chi phí cho kích thước và băng thông, còn có chi phí không nhỏ về bộ nhớ, xử lý và lưu trữ.

Liên quan đến truyền thông điệp, ta có các vấn đề sau:

Liên lạc an toàn (Communicating Securely)
Các client MIDP có thể dựa vào một số cơ chế giống các cơ chế dùng để hỗ trợ liên lạc an toàn giữa các ứng dụng J2EE và các client trình duyệt Web.

Server ứng dụng J2EE và nhiều thiết bị MIDP hỗ trợ HTTP trên Secure Sockets Layer (SSL). Các thiết bị MIDP sử dụng secure HTTP để xác thực với server và tiến hành trao đổi an toàn với server đó. Khung kết nối tổng quát (Generic Connection Framework) trong MIDP cho phép người lập trình mở kết nối secure HTTP chỉ đơn

giản bằng cách gọi phương thức Connector.open() với URL bắt đầu bằng https.

Để xác thực ở phía client, các MIDP client dựa vào việc xác nhận do ứng dụng quản lý, có thể dựa vào cơ chế tự đăng ký. Nói cách khác, MIDP client gởi thông tin ủy nhiệm (ví dụ như tên đăng nhập và mật khẩu) đến ứng dụng J2EE, và ứng dụng sẽ xác nhận các thông tin này, có thể bằng cách sử dụng cơ sở dữ liệu.

Quản lý giao tác
Nền tảng J2EE hỗ trợ giao tác theo nhiều cách. Các nhà phát triển có thể quản lý giao tác một cách thủ công sử dụng Java Transaction API, hoặc dựa vào server J2EE để quản lý các giao tác đó một cách tự động. Enterprise bean thông thường có thực hiện giao tác, nhưng các thành phần nghiệp vụ trong tầng Web cũng có thể thực hiện giao tác.

Khi thiết kế một MIDP client, nhà phát triển nên quan tâm đến việc giao tác không thể trải qua nhiều HTTP request (Các client trình duyệt cũng bị ảnh hưởng bởi giới hạn này). Nếu một request xác định bất kỳ hoạt động nào yêu cầu một giao tác, thì các hoạt động được xử lý như một đơn vị nguyên tử; trước khi trả về response, tất cả các hoạt động đều đã được thực hiện, hoặc không có hoạt động nào được thực hiện.

Do đó, nếu một MIDP client muốn hoàn lại một request, thì nó không thể đưa ra một rollback trong request kế tiếp, bởi vì request kế tiếp sẽ nằm trong một ngữ cảnh giao tác khác. Thay vào đó, ứng dụng phải sử dụng một giao tác bù (compensating transaction) để hoàn lại request đó.

Quản lý lỗi
Khi server J2EE không thể thực hiện request cho MIDP client, nó cần phải thông báo điều này cho client. Mặc dù chương trình của server có thể sử dụng cơ chế quản lý ngoại lệ của Java để xử lý lỗi cục bộ, nó không thể sử dụng cơ chế này để thông báo lỗi cho MIDP client khi liên lạc trên mạng. Nói cách khác, lập trình viên không thể cài đặt một khối try-catch trong mã client để bắt trực tiếp ngoại lệ được ném ra từ server. Thay vào đó, họ phải tổ chức một cơ chế thông báo lỗi vào giao thức truyền thông điệp của họ.

Một cách là dành một phần cố định trong mỗi response của ứng dụng cho một mã trạng thái thể hiện là request của ứng dụng có thành công hay không. Ví dụ, khi sử dụng truyền thông điệp nhị phân, hai byte đầu tiên có thể dành cho mã trạng thái số nguyên. Khi sử dụng HTTP, các nhà phát triển ứng dụng có thể dùng mã trạng thái của HTTP response để thể hiện thành công hay thất bại ở mức truyền thông. Ví dụ, mã trạng thái của 200 (“OK”) có thể dùng để chỉ thành công, trong khi mã trạng thái 500 (“Internal Server Error”) có thể dùng để chỉ thất bại.

Các chiến lược về trình diễn (presentation)
Người dùng tương tác với ứng dụng càng tập trung và trực tiếp, thì người dùng càng có dễ dàng sử dụng. Điều này có ý nghĩa đặc biệt quyết định cho các ứng dụng không dây do màn hình hiển thị và khả năng nhập liệu của các thiết bị di động bị hạn chế.

Các nhà phát triển có thể sử dụng một vài chiến lược để làm cho các ứng dụng không dây có kết nối mạng tăng tính hữu dụng hơn: thực hiện kiểm tra ở phía client, cung cấp biểu thị diễn tiến, cho phép ngắt các hoạt động, và cá nhân hóa ứng dụng. Ta sẽ nghiêng cứu các chiến lược này.

Thực hiện kiểm tra ở phía client
Việc kiểm tra nhập liệu ở phía client là một phương thức tốt để giảm việc gọi đến server. Xét một form đặt hàng, có các trường thông tin thẻ tín dụng. Một MIDlet có thể không thể kiểm tra thông tin này một mình nó được, nhưng chắc chắn nó có thể áp đặt một số phương cách kiểm tra đơn giản để xác định thông tin đó có hợp lệ hay không. Ví dụ, nó có thể kiểm tra tên chủ thẻ không thể là null, hoặc chỉ số thẻ phải có đủ các con số. Nếu dữ liệu nhập được qua các bước này, client sẽ chuyển chúng đến server. Server có thể xử lý các công việc phức tạp hơn, ví dụ như kiểm tra số thẻ tín dụng đó có thật sự thuộc về chủ thẻ hay không hoặc chủ thẻ đó còn đủ tiền hay không.

Bằng cách thực hiện việc kiểm tra nhập liệu ở phía client, các MIDlet có thể tránh việc liên lạc không cần thiết đến server. Các MIDlet có thể chủ động hơn để tránh việc nhập liệu không hợp lệ. Ví dụ có thể giới hạn việc nhập số điện thoại bằng cách sử dụng trường nhập ràng buộc số, do đó các số điện thoại không phải là số sẽ không thể được gởi đến server.

Cung cấp biểu thị diễn tiến (process indicator)
Bởi vì các hoạt động kết nối mạng tốn nhiều thời gian, ứng dụng nên cung cấp cho người dùng một thông tin phản hồi về diễn tiến của hoạt động đó. Ví dụ có thể đưa ra một hoạt hình hoặc gauge để biểu thị diễn tiến. Biểu thị diễn tiến này dùng cho các hoạt động kéo dài, ví dụ như khi download danh sách các trường trên mạng.

Cho phép ngắt hoạt động
Cho phép người dùng có khả năng ngắt các hoạt động kéo dài giúp họ giữ việc điều khiển ứng dụng. Các biểu thị diễn tiến có thể có thêm một nút nhấn ngừng. Biểu thị này sẽ lắng nghe sự kiện của nút nhấn ngừng, và khi nhấn nút ngừng màn hình hiển thị sẽ ngay lập tức chuyển sang màn hình trước đây.

Cần chú ý rằng, không phải tất cả các hoạt động đều có thể dừng. Ví dụ, không nên dừng việc tạo một tài khoản người dùng trên server và lưu lại trên client. Hai công việc này nên thực hiện như là một hoạt động chung hoặc là không thực hiện cả hai. Nếu hoạt động bị ngắt, sẽ có thể dẫn đến sự không thống nhất giữa dữ liệu của client và server.

Cá nhân hóa ứng dụng
Khái niệm cá nhân hóa (personalization) chỉ khả năng một dịch vụ thích ứng với thông tin mà nó biết về người dùng. Thông thường, nhiều thông tin của người dùng, chẳng hạn như địa chỉ, mã ZIP, hay màu sắc ưa thích, sẽ không thay đổi từ phiên làm việc này sang phiên làm việc khác. Bởi vì các dữ liệu này là ổn định, ứng dụng có thể dùng nó để cá nhân hóa việc sử dụng của người dùng.

Việc cá nhân hóa một dịch vụ là có lợi vì hai lý do sau:

Nó giảm việc yêu cầu nhập liệu. Người dùng sẽ chán nếu phải nhập đi nhập lại các thông tin này mỗi lần sử dụng dịch vụ.

Nó sẽ rút ngắn dòng chảy công việc (workflow). Người dùng có thể nhập thông tin tài khoản vào lần đầu, và client sẽ giữ lại thông tin đăng nhập của họ. Trong các lần sử dụng sau đó, người dùng có thể lựa chọn tự động đăng nhập mà không phải qua màn hình đăng nhập.

Trong khi trạng thái phiên làm việc có thể xem là thông tin tạm thời, thì dữ liệu cá

nhân hóa sẽ có tính bền vững. Lưu dữ liệu bền vững này ở đâu là tùy vào người phát triển ứng dụng.

Khi quyết định lưu trữ dữ liệu cá nhân hóa, các nhà phát triển phải xem xét các câu hỏi sau:

Dữ liệu cá nhân hóa có thường xuyên ảnh hưởng đến client request không? Ví dụ, ứng dụng đặt vé sẽ liệt kê các rạp dựa vào mã vùng của người dùng. Server lưu trữ mã vùng này, do đó client không cần phải gởi lại mã vùng mỗi lần nó gởi yêu cầu. Tuy nhiên cũng nên cho phép người dùng có thể bỏ qua mã vùng này, ví dụ khi họ sang thăm một vùng khác.

Thông tin cá nhân hóa có khả năng sử dụng giữa nhiều loại client hay không? Ví dụ, người dùng sử dụng ứng dụng đặt vé trên điện thoại di động có thể muốn truy xuất cùng ứng dụng đó qua Web. Khi đó, họ có thể muốn dữ liệu cá nhân sẽ có sẵn để tránh việc nhập lại nó qua Web.

Quyết định nơi để lưu dữ liệu cá nhân hóa không phải luôn luôn là một quyết định lựa chọn một trong hai. Dữ liệu cá nhân hóa có thể được lưu chồng ở cả client và server. Khi dữ liệu cá nhân hóa được lưu chồng trên cả client và server, ứng dụng có thể cần phải có thêm một số tính năng để đồng bộ hóa dữ liệu này. Các nhà phát triển được khuyên là nên cân nhắc đến chi phí của việc thực hiện các tính năng này.