Các thành viên trong nhóm: Lưu Quang Nghĩa, Đào Quang Trường, Phạm Đức Thắng.

Báo cáo nhóm 4 JavaMailAPI

Khái niệm

Email là ứng dụng đầu tiên của Internet và vẫn tạo ra nhiều lưu lượng truy cập Internet hơn bất kỳ giao thức nào ngoại trừ HTTP. Một trong những câu hỏi thường gặp nhất về Java là cách gửi email từ một applet hoặc ứng dụng Java. Mặc dù chắc chắn có thể viết một chương trình Java sử dụng các ổ cắm để giao tiếp với các máy chủ thư, thực hiện việc này đối với nhiều ứng dụng tầm thường nhất đòi hỏi kiến ​​thức chi tiết về một sốcác giao thức khá phức tạp, chẳng hạn như SMTP, POP và IMAP. Cũng giống như lớp URL làm cho việc tương tác với máy chủ HTTP đơn giản hơn rất nhiều so với tương tác với các socket thô, vì vậy, thư viện lớp dành riêng cho việc xử lý email cũng có thể làm cho việc viết email cho khách hàn đơn giản hơn.

19.1 JavaMail API là gì?

API JavaMail là một đại diện cấp khá cao của các thành phần cơ bản của bất kỳ hệ thống thư điện tử. Các thành phần được đại diện bởi các lớp trừu tượng trong javax.mail gói hàng. Ví dụ, lớp trừu tượng đại diện cho một email thông điệp. Nó khai báo các phương thức trừu tượng để lấy và thiết lập các thông tin phong bì khác nhau cho tin nhắn, chẳng hạn như người gửi và người nhận, ngày gửi và chủ đề của thông điệp. Lớp trừu tượng javax.mail.Folder đại diện cho một vùng chứa thông điệp. Nó khai báo các phương thức trừu tượng để lấy thư từ một thư mục, để di chuyển thư giữa thư mục và để xóa thư khỏi một thư mục

19.2 Gửi Email

Nhu cầu email cơ bản nhất của một chương trình Java là gửi tin nhắn. Trong khi gửi email cho khách hàng như Eudora và trình quản lý danh sách gửi thư như listproc là những chương trình phổ biến duy nhất nhận tin nhắn, tất cả các loại chương trình gửi tin nhắn. Ví dụ, web trình duyệt có thể gửi biểu mẫu HTML qua email. Các công cụ quét bảo mật như Satan có thể chạy trong nền và gửi kết quả của họ qua email cho quản trị viên khi hoàn tất.

Khi chương trình Unix cron phát hiện một tệp crontab được định cấu hình sai, nó sẽ gửi lỗi qua email cho chủ sở hữu. Books & Writers chạy một dịch vụ rất phổ biến với các tác giả để theo dõi xếp hạng bán sách của họ trên Amazon.com và thông báo cho họ định kỳ qua email.

(Xem http://www.booksandwriters.com/rank.html.) Một phép tính song song lớn như dự án SETI @ home có thể gửi kết quả cá nhân qua email. Một số các trò chơi nhiều người chơi như cờ vua có thể được chơi trên mạng bằng cách gửi các nước đi qua email qua lại (mặc dù lược đồ này sẽ không hoạt động đối với các trò chơi chuyển động nhanh hơn như Quake hoặc thậm chí đối với cờ vua tốc độ). Và đây chỉ là một vài trong số các loại

chương trình gửi email. Trong thế giới có dây ngày nay, cách đơn giản nhất để thông báo người dùng của một sự kiện khi anh ta hiện không ngồi trước máy tính mà chương trình đang chạy là để gửi email cho anh ta.

API JavaMail cung cấp mọi thứ mà chương trình của bạn cần để gửi email. Để gửi một thông báo, một chương trình chỉ làm theo tám bước sau:

1. đặt ra email.host thuộc tính trỏ đến máy chủ thư cục bộ
2. Start a mail session with the Session.getInstance( ) phương pháp
3. tạo một cái mới Message đối tượng, có thể bằng cách khởi tạo một trong những vật cụ thể của nó các lớp con.
4. 4. Đặt địa chỉ Từ: của tin nhắn.
5. 5. Đặt địa chỉ To: của tin nhắn.
6. 6. Đặt Chủ đề của tin nhắn:.
7. 7. Đặt nội dung của tin nhắn
8. gửi tin nhắn với Transport.send( ) phương pháp

Thứ tự của các bước này không đặc biệt cứng nhắc. Ví dụ: các bước từ 4 đến 7 có thể thực hiện theo thứ tự bất kỳ. Hơn nữa, mỗi bước trong số này khá đơn giản.

Bước đầu tiên là thiết lập các thuộc tính cho phiên thư. Tài sản duy nhất bạn phải thiết lập để gửi thư là mail.host Điều này được định cấu hình là java.util.Properties object rather than an environment variable. For example, this code fragment sets the email.host property to mail.cloud9.net :

Properties props = new Properties( ); props.put("mail.host", "mail.cloud9.net");

Các chương trình của bạn tất nhiên sẽ phải đặt thuộc tính này thành tên của thư riêng của bạn

người phục vụ. Các thuộc tính này được sử dụng để truy xuất một Session đối tượng từ Session.getInstance( ) phương pháp nhà máy như thế này:

Session mailConnection = Session.getInstance(props, null);

các session đối tượng được sử dụng để xây dựng một message obiect:

Message msg = new MimeMessage(mailConnection);

Tôi chỉ định cụ thể lớp MimeMessage vì tôi biết mình đang gửi email qua Internet.

Tuy nhiên, đây là nơi mà tôi chọn định dạng cho email một cách rõ ràng

thông điệp. Trong một số trường hợp, điều này có thể không cần thiết nếu tôi có thể sao chép tin nhắn đến

định dạng thay thế.

Bây giờ tôi có một đối tượng Message, tôi cần thiết lập các trường và nội dung của nó. Từ: địa chỉ và Đến: mỗi địa chỉ sẽ là javax.mail.internet.InternetAddress

các đối tượng. Bạn có thể cung cấp địa chỉ email một mình hoặc địa chỉ email và thực

Tên. Ví dụ: Address bill = new InternetAddress("god@microsoft.com", "Bill Gates"); Address elliotte = new InternetAddress("elharo@metalab.unc.edu");

Thông báo setFrom () cho phép chúng tôi biết ai đang gửi tin nhắn bằng cách đặt

Từ: tiêu đề. Không có biện pháp bảo vệ chống lại sự giả mạo ở đây. Nó khá dễ dàng cho tôi

giả trang thành Bill Gates tại một địa chỉ email (có thể là) hư cấu:

msg.setFrom(bill); Phương thức setRecipient () phức tạp hơn một chút. Bạn không chỉ phải

chỉ định địa chỉ mà thư sẽ được gửi đến, nhưng địa chỉ đó được sử dụng như thế nào; cái đó

là, as a To: field, a Cc: field, or a Bcc Những điều này được biểu thị bằng ba ký ức hằng số của Message.RecipientType lớp :

Message.RecipientType.TO

Message.RecipientType.CC

Message.RecipientType.BCC

ví dụ

msg.setRecipient(Message.RecipientType.TO, elliotte);

Chủ đề được đặt dưới dạng một chuỗi văn bản đơn giản. Ví dụ:

msg.setSubject("You must comply.");

Phần nội dung cũng được đặt dưới dạng một chuỗi văn bản. Tuy nhiên, cũng như văn bản đó bạn cần để cung cấp kiểu MIME của văn bản. Loại phổ biến nhất là văn bản / thuần túy. Đối với

thí dụ:

msg.setContent("Resistance is futile. You will be assimilated!", "text/plain");

Cuối cùng, phương thức Transport.send () tĩnh kết nối với máy chủ thư được chỉ định bởi thuộc tính mail.host và gửi tin nhắn theo cách của nó

Transport.send(msg);

Ví dụ 19.1 đặt tất cả các bước này lại với nhau thành một chương trình độc lập gửi

tin nhắn sau:

Date: Mon, 29 Nov 1999 15:55:42 -0500 (EST)

From: Bill Gates <god@microsoft.com>

To: elharo@metalab.unc.edu Subject: You must comply. Resistance is futile. You will be assimilated!

Tôi đã hiển thị thông báo này ở đây ở định dạng RFC 822 tiêu chuẩn được sử dụng cho email Internet.

Tuy nhiên, điều đó là không cần thiết. Điểm chính là bạn cần biết người nhận (elharo@metalab.unc.edu), người gửi (god@microsoft.com), chủ đề và nội dung thư.

Ví dụ 19.1. Gửi một tin nhắn thư rất đơn giản

import javax.mail.\*;

import javax.mail.internet.\*; import java.util.\*;

public class Assimilator

{ public static void main(String[] args)

{ try { Properties props = new Properties( ); props.put("mail.host", "mail.cloud9.net");

Session mailConnection = Session.getInstance(props, null);

Message msg = new MimeMessage(mailConnection); Address bill = new InternetAddress("god@microsoft.com", "Bill Gates"); Address elliotte = new InternetAddress("elharo@metalab.unc.edu"); msg.setContent("Resistance is futile. You will be assimilated!", "text/plain"); msg.setFrom(bill); msg.setRecipient(Message.RecipientType.TO, elliotte); msg.setSubject("You must comply."); Transport.send(msg); } catch (Exception e) { e.printStackTrace( ); }

}

}

19.2.1 Gửi email từ một ứng dụng

Ví dụ 19.1 là một ứng dụng đơn giản gửi một tin nhắn cố định đến một địa chỉ đã biết

với một chủ đề xác định. Sau khi bạn thấy cách thực hiện việc này, bạn có thể dễ dàng thay thế

các chuỗi cung cấp địa chỉ, chủ đề và nội dung thư với dữ liệu được đọc từ

dòng lệnh, GUI, cơ sở dữ liệu hoặc một số nguồn khác. Ví dụ, ví dụ 19.2 là

một GUI rất đơn giản để gửi email. Hình 19.1 cho thấy chương trình đang chạy. Các tất cả mã thư đều được gắn trong phương thức actionPerformed () và trông rất giống với phương thức main () của Ví dụ 19.1. Sự khác biệt lớn là bây giờ máy chủ, chủ đề, from: address, to: address và văn bản của tin nhắn đều được đọc từ GUI các thành phần trong thời gian chạy thay vì được mã hóa cứng dưới dạng chuỗi ký tự trong nguồn mã. Phần còn lại của mã liên quan đến việc thiết lập GUI và ít liên quan đến API JavaMail.

Example 19.2. A Graphical SMTP Client

import javax.mail.\*;

import javax.mail.internet.\*;

import java.util.\*;

import javax.swing.\*;

import java.awt.event.\*;

import java.awt.\*;

public class SMTPClient extends JFrame { private JButton sendButton = new JButton("Send Message"); private JLabel fromLabel = new JLabel("From: ");

private JLabel toLabel = new JLabel("To: ");

private JLabel hostLabel = new JLabel("SMTP Server: ");

private JLabel subjectLabel = new JLabel("Subject: ");

private JTextField fromField = new JTextField(40);

private JTextField toField = new JTextField(40);

private JTextField hostField = new JTextField(40);

private JTextField subjectField = new JTextField(40);

private JTextArea message = new JTextArea(40, 72);

private JScrollPane jsp = new JScrollPane(message);

public SMTPClient( ) { super("SMTP Client");

Container contentPane = this.getContentPane( ); contentPane.setLayout(new BorderLayout( ));

JPanel labels = new JPanel( ); labels.setLayout(new GridLayout(4, 1)); labels.add(hostLabel);

JPanel fields = new JPanel( );

fields.setLayout(new GridLayout(4, 1)); String host = System.getProperty("mail.host", "");

hostField.setText(host); fields.add(hostField);

labels.add(toLabel); fields.add(toField); String from = System.getProperty("mail.from", "");

fromField.setText(from); labels.add(fromLabel);

fields.add(fromField);

labels.add(subjectLabel);

fields.add(subjectField);

Box north = Box.createHorizontalBox( );

north.add(labels);

north.add(fields);

contentPane.add(north, BorderLayout.NORTH);

message.setFont(new Font("Monospaced", Font.PLAIN, 12)); contentPane.add(jsp, BorderLayout.CENTER);

JPanel south = new JPanel( );

south.setLayout(new FlowLayout(FlowLayout.CENTER)); south.add(sendButton);

sendButton.addActionListener(new SendAction( ));

contentPane.add(south, BorderLayout.SOUTH); this.pack( ); }

class SendAction implements ActionListener { public void actionPerformed(ActionEvent evt) { try { Properties props = new Properties( ); props.put("mail.host", hostField.getText( )); Session mailConnection = Session.getInstance(props, null); final Message msg = new MimeMessage(mailConnection); Address to = new InternetAddress(toField.getText( )); Address from = new InternetAddress(fromField.getText( )); msg.setContent(message.getText( ), "text/plain"); msg.setFrom(from); msg.setRecipient(Message.RecipientType.TO, to); msg.setSubject(subjectField.getText( )); // This can take a non-trivial amount of time so // spawn a thread to handle it. Runnable r = new Runnable( ) { public void run( ) { try { Transport.send(msg); } catch (Exception e) { e.printStackTrace( ); } } }; Thread t = new Thread(r); t.start( ); message.setText(""); } catch (Exception e) { // We should really bring up a more specific error dialog here. e.printStackTrace( ); } } } public static void main(String[] args) { SMTPClient client = new SMTPClient( ); // Next line requires Java 1.3. We want to set up the // exit behavior here rather than in the constructor since // other programs that use this class may not want to exit // the application when the SMTPClient window closes. client.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); client.show( ); } }

Đây không phải là một chương trình lý tưởng. GUI có thể được tách biệt rõ ràng hơn mã gửi thư. Và sẽ tốt hơn nếu hiển thị hộp thoại lỗi nếu có vấn đề xảy ra sai hơn là chỉ in dấu vết ngăn xếp của ngoại lệ trên System.err. Tuy nhiên, vì không có điều nào trong số đó sẽ dạy chúng tôi bất cứ điều gì về JavaMail API, tôi rời khỏi tất cả những điều đó như một bài tập cho người đọc quan tâm.

Hình 19.1. Một chương trình thư GUI đơn giản

19.2.2 Gửi Email từ Applet Về mặt GUI và API JavaMail, không có sự khác biệt giữa việc gửi email từ một applet và một ứng dụng. Tuy nhiên, trình quản lý bảo mật của trình duyệt có thể truy cập cách của bạn. Giống như mọi thứ khác trong cuốn sách này, API JavaMail không thể vượt qua hạn chế thông thường đối với kết nối mạng từ các applet. Một applet muốn gửi email vẫn chỉ có thể nói chuyện với máy chủ lưu trữ mà chính applet đến từ đó. Tuy nhiên, may mắn thay, hầu hết các máy chủ chạy máy chủ web cũng chạy máy chủ SMTP. [1] Nếu điều này trong trường hợp này, thì việc tạo một applet gửi email là điều khá đơn giản. Các API JavaMail và Khung kích hoạt Java mà nó phụ thuộc vào không được bao gồm với hầu hết các trình duyệt, nhưng vì chúng được triển khai bằng Java thuần túy trong gói javax, trình duyệt có thể tải xuống các lớp cần thiết từ máy chủ. Ví dụ, điều này Phần tử APPLICET không chỉ tham chiếu đến mã riêng của applet mà còn tham chiếu đến mail.jar và các tệp activation.jar cho JavaMail API và Java Activation Framework, tương ứng:

[1] This is perhaps a little more likely to be true of a Unix web server than a Mac or Windows web server, since most Mac and Windows servers don't ship with SMTP servers by default.

**Ví dụ 19.3** là một applet đơn giản để gửi email. Địa chỉ để gửi email và chủ đề được đọc từ các thẻ PARAM. Địa chỉ để gửi email cũng được đọc từ Thẻ PARAM, nhưng người dùng có tùy chọn để thay đổi nó. Văn bản cần gửi được nhập thành văn bản khu vực của người dùng. Cuối cùng, máy chủ được xác định bằng cách xem codebase của applet. Ví dụ 19.3. Applet gửi email

import java.applet.\*; import javax.mail.\*; import javax.mail.internet.\*; import java.util.Properties; import java.awt.event.\*; import java.awt.\*; public class SMTPApplet extends Applet { private Button sendButton = new Button("Send Message"); private Label fromLabel = new Label("From: "); private Label subjectLabel = new Label("Subject: "); private TextField fromField = new TextField(40); private TextField subjectField = new TextField(40); private TextArea message = new TextArea(30, 60); private String toAddress = ""; public SMTPApplet( ) { this.setLayout(new BorderLayout( )); Panel north = new Panel( ); north.setLayout(new GridLayout(3, 1)); Panel n1 = new Panel( ); n1.add(fromLabel); n1.add(fromField); north.add(n1); Panel n2 = new Panel( ); n2.add(subjectLabel); n2.add(subjectField); north.add(n2); this.add(north, BorderLayout.NORTH); message.setFont(new Font("Monospaced", Font.PLAIN, 12)); this.add(message, BorderLayout.CENTER); Panel south = new Panel( ); south.setLayout(new FlowLayout(FlowLayout.CENTER)); south.add(sendButton); sendButton.addActionListener(new SendAction( )); this.add(south, BorderLayout.SOUTH); } public void init( ) { String subject = this.getParameter("subject"); if (subject == null) subject = ""; subjectField.setText(subject); toAddress = this.getParameter("to"); if (toAddress == null) toAddress = ""; String fromAddress = this.getParameter("from"); if (fromAddress == null) fromAddress = ""; fromField.setText(fromAddress); } class SendAction implements ActionListener { public void actionPerformed(ActionEvent evt) { try { Properties props = new Properties( ); props.put("mail.host", getCodeBase().getHost( )); Session mailConnection = Session.getInstance(props, null); final Message msg = new MimeMessage(mailConnection); Address to = new InternetAddress(toAddress); Address from = new InternetAddress(fromField.getText( )); msg.setContent(message.getText( ), "text/plain"); msg.setFrom(from); msg.setRecipient(Message.RecipientType.TO, to); msg.setSubject(subjectField.getText( )); // This can take a non-trivial amount of time so // spawn a thread to handle it. Runnable r = new Runnable( ) { public void run( ) { try { Transport.send(msg); } catch (Exception e) { e.printStackTrace( ); } } }; Thread t = new Thread(r); t.start( ); message.setText(""); } catch (Exception e) { // We should really bring up a more specific error dialog here. e.printStackTrace( ); } } } }

Hình 19.2 cho thấy applet này đang chạy trong Internet Explorer 4.0.1 trên Macintosh. Tôi đã cẩn thận chỉ sử dụng các phương thức và lớp có sẵn trong Java 1.1 để điều này applet sẽ chạy trên hầu hết các trình duyệt web có thể. Tôi cũng tránh sử dụng Swing nên rằng sẽ có một tệp JAR nhỏ hơn để tải xuống. Như nó vốn có, mail.jar và Các tệp kích hoạt.jar mà applet này yêu cầu chiếm gần 300K, nhiều hơn tôi thoải mái nhưng có thể quản lý được trên kết nối nhanh.

Hoạt động thích hợp của applet này phụ thuộc vào một số yếu tố bên ngoài: • Trình duyệt phải hỗ trợ ít nhất Java 1.1 với mô hình bảo mật không nghiêm ngặt hơn so với mặc định. • Các tệp mail.jar và activation.jar phải có sẵn trong codebase của applet. • Máy chủ web cung cấp applet cũng phải là máy chủ SMTP sẵn sàng chuyển tiếp thư từ hệ thống máy khách đến hệ thống máy thu. Ngày nay, mở nhất Các rơle SMTP đã bị tắt để tránh bị những kẻ gửi thư rác lạm dụng, vì vậy điều này có thể một điểm dính. Nếu đúng như vậy, bạn sẽ nhận được một ngoại lệ như sau:

19.3 Nhận thư Nhận thư phức tạp hơn nhiều so với gửi. Ví dụ, trong đó lệnh HELLO đơn giản là đủ để truy cập hầu hết các máy chủ SMTP (thực tế là nguồn của nhiều email giả mạo và thư rác), việc truy xuất email thường yêu cầu cung cấp cả tên người dùng và mật khẩu. SMTP chỉ sử dụng 14 lệnh khác nhau và ứng dụng email khách đơn giản có thể được triển khai chỉ với năm trong số chúng. POP3, tuy nhiên, có 12 hầu như tất cả các lệnh mà máy khách phải có thể xử lý và IMAP4 có 24 các lệnh khác nhau. API JavaMail được thiết kế dựa trên ý tưởng rằng bạn đang truy xuất thư từ một IMAP hoặc có thể là một máy chủ NNTP. Có nghĩa là, nó giả định rằng máy chủ có thể trả về tiêu đề tách biệt với các thư mà chúng thuộc về, mà nó có thể tìm kiếm qua hộp thư, nó cung cấp bộ nhớ cho các thư chứ không phải máy khách, và vì vậy ra ngoài. API JavaMail cung cấp ít hơn những gì bạn cần cho thư hướng khách hàng giao thức truy cập, chẳng hạn như POP3, giả sử máy khách lưu trữ và quản lý thư lưu trữ, nhưng nó vẫn cung cấp cho bạn các công cụ để tải xuống thư từ máy chủ. Bạn chỉ phải triển khai hệ thống lưu trữ của riêng bạn trên máy khách. Ngày nay, nhiều khách hàng hơn sử dụng POP thay vì IMAP để truy cập thư của họ (đặc biệt là tại các ISP không muốn sử dụng dung lượng đĩa để lưu trữ hộp thư của người dùng), vì vậy chúng ta sẽ bắt đầu với giao thức POP đơn giản hơn, sau đó chuyển sang IMAP. Theo quan điểm của JavaMail, IMAP có thể được xem phần lớn là POP cộng với một số lệnh để thao tác các thư mục. Đối với các chương trình đơn giản chỉ hoạt động trên thư mục INBOX, ứng dụng khách POP và IMAP ít nhiều đều giống nhau. Việc triển khai JavaMail của Sun không bao gồm POP3 nhà cung cấp dịch vụ, vì vậy để kết nối với máy chủ POP, bạn sẽ phải cài đặt một máy chủ. Một số POP các nhà cung cấp được liệt kê trong Bảng 19.1. Đối với các ví dụ trong chương này, Sun's POP3 nhà cung cấp sẽ là đủ, mặc dù Nhà cung cấp POP3 của Phần mềm Liên kết cho JavaMail 1.1 là tính năng hoàn chỉnh hơn. Cả hai đều bao gồm tệp .jar mà bạn thêm vào CLASSPATH biến môi trường hoặc đặt trong thư mục ext của bạn. Có khoảng 12 bước để đọc một hộp thư từ xa. (Con số chính xác có thể thay đổi ít, vì một số bước là tùy chọn hoặc có thể được kết hợp với hoặc thay thế bằng những bước khác.) 1. Thiết lập các thuộc tính bạn sẽ sử dụng cho kết nối. 2. Xây dựng Authenticator mà bạn sẽ sử dụng cho kết nối. 3. Lấy một đối tượng Session với Session.getDefaultInstance (). 4. Sử dụng phương thức getStore () của phiên để trả về một Cửa hàng. 5. Kết nối với cửa hàng. 6. Lấy thư mục INBOX từ cửa hàng bằng phương thức getFolder (). 7. Mở thư mục INBOX. 8. Mở thư mục bạn muốn bên trong thư mục INBOX. Lặp lại nhiều lần như cần thiết để truy cập thư mục bạn đang tìm kiếm. 9. Nhận thư từ thư mục dưới dạng một mảng các đối tượng Thư. 10. Lặp đi lặp lại qua các mảng thông báo, xử lý lần lượt từng thông báo bằng cách sử dụng các phương thức của lớp Message. Ví dụ: bạn có thể in ra từng tin nhắn hoặc chỉ hiển thị người gửi, chủ đề và thông tin quan trọng khác trong GUI cho người dùng để chọn, như trong Hình 19.3. 11. Đóng thư mục. 12. Đóng cửa hàng. Hình 1

Mỗi bước này khá đơn giản. Đầu tiên là thiết lập các thuộc tính cho phiên thư. Các thuộc tính bạn có thể muốn đặt bao gồm mail.host, mail.store.protocol, mail.user, mail.pop3.user và mail.pop3.host. Tuy nhiên, bạn không nhất thiết phải thiết lập bất kỳ điều nào trong số này. Nếu Phiên sẽ được sử dụng chỉ để truy xuất thư, thì một đối tượng Thuộc tính trống sẽ là đủ. Ví dụ: Properties props = new Properties (); Tiếp theo, bạn sẽ muốn tạo một phiên bản của lớp javax.mail.Authenticator (đúng hơn, một ví dụ của một lớp con cụ thể của Trình xác thực trừu tượng lớp) có thể yêu cầu người dùng nhập mật khẩu của cô ấy. Hiện tại, chúng tôi chỉ đơn giản là mã hóa những và chuyển giá trị null thay vì một Trình xác thực thực tế. Tuy nhiên, chúng tôi sẽ sửa lỗi này sau khi chúng ta thảo luận về xác thực: Trình xác thực a = null; Sau đó, sử dụng các đối tượng Thuộc tính và Trình xác thực của bạn để có được một phiên bản Phiên như thế này: Phiên phiên = Session.getDefaultInstance (props, a); Tiếp theo yêu cầu phiên cho một cửa hàng cho nhà cung cấp. Ở đây, chúng tôi muốn một nhà cung cấp cho POP3: Cửa hàng lưu trữ = session.getStore ("POP3"); Cuối cùng, bạn đã sẵn sàng thực sự kết nối với cửa hàng bằng phương thức connect (). Bạn sẽ cần cung cấp máy chủ lưu trữ để kết nối và tên người dùng và mật khẩu để sử dụng: store.connect ("mail.cloud9.net", "elharo", "my\_password"); Bạn có thể chuyển null cho mật khẩu để chỉ ra rằng Trình xác thực phải được truy vấn cho mật khẩu. Bây giờ cửa hàng đã được kết nối, bạn đã sẵn sàng để mở một thư mục trong cửa hàng. Bước này là thực sự hướng đến IMAP hơn POP, vì máy chủ POP không theo dõi các thư mục khác nhau. Họ chỉ cung cấp tất cả thư đến của người dùng dưới dạng một hỗn hống không phân biệt. Đối với các mục đích của JavaMail API, các nhà cung cấp POP3 sử dụng tên thư mục INBOX: Hộp thư đến của thư mục = store.getFolder ("INBOX"); Thư mục được đóng khi bạn nhận được nó. Bạn có thể thực hiện một số hoạt động trên một thư mục bao gồm xóa hoặc đổi tên nó, nhưng bạn không thể lấy thư ra khỏi hộp đã đóng thư mục. Đầu tiên bạn phải mở nó. Bạn có thể mở một thư mục để đọc quyền truy cập bằng cách chuyển mnemonic Constant Folder.READ\_ONLY vào phương thức open () để truy cập đọc, hoặc Folder.READ\_WRITE để truy cập đọc / ghi: inbox.open (Thư mục.READ\_ONLY); Bây giờ bạn đã sẵn sàng để tải xuống các tin nhắn. Làm điều này với getMessages () , phương thức này trả về một mảng chứa tất cả các thư trong thư mục: Tin nhắn [] messages = inbox.getMessages (); (Nếu bạn đang sử dụng IMAP thay vì POP, bước này sẽ không thực sự tải xuống tin nhắn. Mỗi cái sẽ ở trên máy chủ cho đến khi bạn truy cập cụ thể vào nó. Bạn sẽ chỉ nhận được một con trỏ đến thông báo thực tế.) Lớp Message cung cấp nhiều phương thức để làm việc với các thư riêng lẻ. Nó có các phương thức để lấy các trường tiêu đề khác nhau của thư, để lấy nội dung của để trả lời tin nhắn và hơn thế nữa. Chúng ta sẽ sớm thảo luận về những điều này, khi chúng ta nói chuyện về các lớp Message và MimeMessage. Hiện tại, chúng tôi sẽ chỉ làm về điều đơn giản nhất có thể tưởng tượng, in từng tin nhắn trên System.out bằng cách sử dụng phương thức writeTo ():

for (int i = 0; i < messages.length; i++) { System.out.println("------------ Message " + (i+1) + " ------------"); messages[i].writeTo(System.out); }

Sau khi hoàn thành các thư, bạn nên đóng thư mục, sau đó đóng lưu trữ tin nhắn với các phương thức close () được đặt tên phù hợp:

inbox.close(false); store.close( );

Đối số sai đối với phương thức close () của thư mục chỉ ra rằng chúng tôi không muốn máy chủ để thực sự xóa mọi thư đã xóa trong thư mục. Chúng tôi chỉ đơn giản là muốn ngắt kết nối của chúng tôi với thư mục này. Ví dụ 19.4 đặt tất cả điều này cùng với một chương trình đơn giản tải xuống và in ra nội dung của một hộp thư POP được chỉ định. Tin nhắn chỉ đơn giản là đổ vào System.out trong bảng mã mặc định. Máy chủ, tên người dùng, v.v. mã cứng. Tuy nhiên, điều này nhanh chóng chứng tỏ hầu hết các điểm chính của việc nhận thư bằng JavaMail API. Một chương trình nâng cao hơn sẽ bao gồm một GUI

Example 19.4

. POP3Client import javax.mail.\*;

import javax.mail.internet.\*; import java.util.\*; import java.io.\*; public class POP3Client { public static void main(String[] args) { Properties props = new Properties( ); String host = "utopia.poly.edu"; String username = "eharold"; String password = "mypassword"; String provider = "pop3"; try { // Connect to the POP3 server Session session = Session.getDefaultInstance(props, null); Store store = session.getStore(provider); store.connect(host, username, password); // Open the folder Folder inbox = store.getFolder("INBOX"); if (inbox == null) { System.out.println("No INBOX"); System.exit(1); } inbox.open(Folder.READ\_ONLY); // Get the messages from the server Message[] messages = inbox.getMessages( ); for (int i = 0; i < messages.length; i++) { System.out.println("------------ Message " + (i+1) + " ------------"); messages[i].writeTo(System.out); } // Close the connection // but don't remove the messages from the server inbox.close(false); store.close( ); } catch (Exception e) { e.printStackTrace( ); } } } Here's some sample output I got when I pointed it at an account I don't use much: D:\JAVA\JNP2\examples\18>java POP3Client ------------ Message 1 ------------ Received: (from eharold@localhost) by utopia.poly.edu (8.8.8/8.8.8) id QAA05728 for eharold; Tue, 30 Nov 1999 16:14:29 -0500 (EST) Date: Tue, 30 Nov 1999 16:14:29 -0500 (EST) From: Elliotte Harold Message-Id: To: eharold@utopia.poly.edu Subject: test Content-Type: text X-UIDL: 87e3f1ba71738c8f772b15e3933241f0 Status: RO hello you ------------ Message 2 ------------ Received: from russian.cloud9.net (russian.cloud9.net [ .4]) by utopia.poly.edu (8.8.8/8.8.8) with ESMTP id OAA28428 for ; Wed, 1 Dec 1999 14:05:06 -0500 ( Received: from [168.100.203.234] (macfaq.dialup.cloud9.net [168.100.203 by russian.cloud9.net (Postfix) with ESMTP id 24B93764F for ; Wed, 1 Dec 1999 14:02:50 - 0500 Mime-Version: 1.0 X-Sender: macfaq@mail.cloud9.net Message-Id: Date: Wed, 1 Dec 1999 13:55:40 -0500 To: eharold@utopia.poly.edu From: Elliotte Rusty Harold Subject: New system Content-Type: text/plain; charset="us-ascii" ; format="flowed" X-UIDL: 01fd5cbcf1768fc6c28f9c8f934534b5 Just thought you'd be happy to know that now that I've got my desk moved over from my old apartment, I've finally ordered the Windows NT system I've been promising for months. -- David About the only change you'd need to make to port this program to IMAP would be setting the provider variable to imap instead of pop3.

19.4 Xác thực mật khẩu Mã hóa cứng mật khẩu trong mã nguồn như Ví dụ 19.4, nói một cách tối thiểu, rất ý kiến tồi. Nếu mật khẩu được yêu cầu, bạn nên hỏi người dùng về mật khẩu đó trong thời gian chạy. Hơn nữa, khi người dùng nhập mật khẩu, mật khẩu sẽ không được hiển thị trên màn. Tốt nhất, nó thậm chí không nên được truyền dưới dạng văn bản rõ ràng trên mạng, mặc dù trên thực tế, nhiều máy khách và máy chủ POP hiện tại làm chính xác điều đó. (IMAP có xu hướng an toàn hơn một chút.) Khi bạn mở một kết nối với kho tin nhắn, API JavaMail cho phép bạn cung cấp một đối tượng javax.mail.Authenticator mà nó có thể sử dụng để lấy tên người dùng và mật khẩu. Authenticator là một lớp trừu tượng: công khai lớp trừu tượng Authenticator mở rộng đối tượng Khi nhà cung cấp cần biết tên người dùng hoặc mật khẩu, nhà cung cấp sẽ gọi lại phương thức getPasswordAuthentication () trong lớp con do người dùng xác định của Trình xác thực. Điều này trả về một đối tượng PasswordAuthentication chứa cái này thông tin:

Để thêm xác thực mật khẩu thời gian chạy vào chương trình của bạn, bạn phân lớp Authenticator và ghi đè getPasswordAuthentication () bằng một phương thức biết cách yêu cầu người dùng nhập mật khẩu một cách an toàn. Một công cụ hữu ích cho quá trình này là thành phần JPasswordField từ Swing. Ví dụ 19.5 minh họa một lớp con Swingbased Authenticator mang đến một hộp thoại để hỏi người dùng về tên người dùng và mật khẩu.

Example 19.5.

Hầu hết mã này chỉ để xử lý GUI. Hình 19.4 cho thấy khá đơn giản hộp thoại này tạo ra.

Điều thú vị là JPasswordField mất nhiều công sức hơn để bảo mật Xác thực mật khẩu. JPasswordField lưu trữ mật khẩu dưới dạng một mảng ký tự để khi bạn hoàn thành mật khẩu, bạn có thể ghi đè nó bằng null. Điều này có nghĩa là mật khẩu tồn tại trong bộ nhớ trong thời gian ngắn hơn và ít có khả năng bị vô tình được đổi sang đĩa và để lại ở đó trong hệ thống bộ nhớ ảo. Tuy nhiên, PasswordAuthentication lưu trữ mật khẩu dưới dạng chuỗi, không thay đổi được và do đó có thể vô tình được lưu trữ trên đĩa. Việc sửa đổi ứng dụng khách POP để hỗ trợ kiểu xác thực này rất đơn giản, như Ví dụ 19.6 chứng minh. Chúng tôi thay thế tên người dùng và mật khẩu được mã hóa cứng bằng null và chuyển một phiên bản của MailAuthenticator làm đối số thứ hai cho kết nối( ). Thay đổi duy nhất khác là chúng tôi gọi System.exit () ở cuối phương thức main (), vì chương trình sẽ không còn thoát khi phương thức main () trả về sau khi chuỗi AWT đã được bắt đầu.

Example 19.6.

19.5 Địa chỉ Lớp javax.mail.Address rất đơn giản. Đó là một lớp trừu tượng tồn tại chủ yếu được phân lớp bởi các lớp địa chỉ giao thức cụ thể khác: lớp trừu tượng công khai Địa chỉ mở rộng Đối tượng Có hai trong số các lớp con này trong API JavaMail tiêu chuẩn: InternetAddress cho email SMTP và NewsAddress cho nhóm tin Usenet: lớp công cộng InternetAddress mở rộng Địa chỉ public class NewsAddress mở rộng Địa chỉ Các nhà cung cấp giao thức thư khác cũng sẽ phân lớp Địa chỉ với các lớp đại diện cho phong cách địa chỉ của họ. 19.5.1 Lớp địa chỉ Bản thân lớp Address cực kỳ đơn giản. Nó chỉ có ba phương thức, tất cả đều trừu tượng và hai trong số đó là các phương thức tiện ích đơn giản ghi đè lên các phương thức tương ứng

in java.lang.Object: public abstract String getType( ) public abstract String toString( ) public abstract boolean equals(Object o)

Vì cả ba phương pháp này đều trừu tượng, không có bất kỳ đảm bảo nào ở đây về ngữ nghĩa của các phương thức, vì tất cả phải được ghi đè trong các lớp con. Tuy nhiên, điều này không yêu cầu rằng các lớp con cung cấp các triển khai của chính chúng đối với các hàm bằng () và toString () thay vì dựa vào các triển khai khá chung chung có sẵn từ java.lang.Object. Nói chung, phương thức getType () sẽ trả về một chuỗi chẳng hạn như "rfc822" hoặc "news" cho biết đây là loại đối tượng Địa chỉ. 19.5.2 Lớp InternetAddress Đối tượng InternetAddress đại diện cho một địa chỉ email kiểu RFC 822. Đây là địa chỉ email kiểu Internet chuẩn đang nhanh chóng thay thế tất cả các địa chỉ độc quyền khác các định dạng. Nó giống như elharo@metalab.unc.edu hoặc ask\_tim@oreilly.com. Tuy nhiên, nó cũng có thể chứa tên — ví dụ: ask\_tim@oreilly.com (Tim O'Reilly). Trạng thái của một đối tượng InternetAddress được duy trì bởi ba trường được bảo vệ:

protected String address protected String personal protected String encodedPersonal

Trường địa chỉ là địa chỉ email thực — ví dụ: ask\_tim@oreilly.com. Trường cá nhân là tên — ví dụ, Tim O'Reilly. Mặc dù chuỗi Java là Unicode thuần túy có thể biểu thị các tên như Erwin Schrödinger hoặc , các chuỗi được sử dụng trong tiêu đề thư phải thuần túy ASCII để chuyển qua hầu hết các phần mềm thư hiện có. Do đó, Java's Các chuỗi Unicode cần được chuyển đổi thành ASCII thuần túy bằng cách sử dụng một loại thập lục phân bỏ trốn. Chi tiết của việc chuyển đổi này được mô tả trong RFC 2047, MIME (Tiện ích mở rộng thư Internet đa năng) Phần thứ ba: Tiện ích mở rộng tiêu đề thư cho Văn bản không phải ASCII. Chuỗi mã hóa được đặt trong trường encodedPersonal. Tất cả các trường này ban đầu sẽ được đặt trong hàm tạo. Có bốn quá tải các hàm tạo cho các đối tượng InternetAddress:

Phương thức setAddress () đặt trường địa chỉ của đối tượng thành giá trị được chỉ định. Phương thức setPersonal () đặt trường cá nhân và trường cá nhân được mã hóa thành giá trị được chỉ định (sau khi mã hóa nó khi cần thiết). GetAddress () và Các phương thức getPersonal () trả về các giá trị của địa chỉ và cá nhân hoặc được giải mã Các trường cá nhân được mã hóa, tương ứng. Cuối cùng, phương thức getType () trả về chuỗi "rfc822". Phương thức toString () trả về một địa chỉ email phù hợp để sử dụng trong To: hoặc From: trường của một email RFC 822. Phương thức equals () và hashCode () có ngữ nghĩa thông thường của chúng. Ngoài ra còn có năm phương thức tiện ích tĩnh, bốn trong số đó chuyển đổi địa chỉ đến và đi dây:

Các phương thức InternetAddress.toString () chuyển đổi một mảng các đối tượng Địa chỉ thành danh sách địa chỉ được phân tách bằng dấu phẩy được mã hóa bằng ASCII thuần túy, có thể được xếp vào nhiều dòng. Đối số được sử dụng tùy chọn cung cấp số ký tự sẽ đặt trước chuỗi này trong trường tiêu đề, chẳng hạn như Tới: hoặc Cc :, mà chuỗi này sẽ là được chèn vào. Điều này cho phép toString () quyết định nơi nó cần ngắt dòng. A ClassCastException được ném nếu bất kỳ đối tượng Địa chỉ nào trong mảng không cụ thể hơn là các đối tượng InternetAddress. Hai phương thức parse () thực hiện thao tác này ngược lại, chuyển đổi một Chuỗi địa chỉ được phân tách bằng dấu phẩy thành một mảng các đối tượng InternetAddress. Đặt đối số chặt chẽ tùy chọn đối với sai thay đổi hành vi để các chuỗi sử dụng khoảng trắng thay vì dấu phẩy (hoặc khoảng trắng và dấu phẩy) để phân tách email địa chỉ cũng được hiểu. Tất cả bốn phương pháp này đều hữu ích cho tin nhắn trường tiêu đề chứa nhiều địa chỉ; ví dụ, một Cc: được hướng đến sáu Mọi người. Cuối cùng, phương thức getLocalAddress () kiểm tra một số thuộc tính hệ thống (mail.from, mail.user, mail.host và user.name) cũng như InetAddress.getLocalName () để xác định địa chỉ email của người dùng hiện tại: public static InternetAddress getLocalAddress (Phiên phiên) Ví dụ: đoạn mã này cố gắng sử dụng địa chỉ email của chính người dùng thay vì một mã được mã hóa cứng vào chương trình dưới dạng một chuỗi: msg.setFrom (InternetAddress.getLocalAddress ()); Tuy nhiên, không có gì đảm bảo rằng bất kỳ thuộc tính nào trong số này nhất thiết sẽ cung cấp cho địa chỉ thực của người dùng. 19.5.3 Lớp NewsAddress Có lẽ hơi ngạc nhiên, với một nhà cung cấp dịch vụ thích hợp, JavaMail API cũng có thể truy cập tin tức Usenet. API chủ yếu giống như để đọc POP hoặc Hộp thư IMAP. Tuy nhiên, thay vì sử dụng InternetAddress, bạn sử dụng NewsAddress: public class NewsAddress mở rộng Địa chỉ Đối tượng NewsAddress đại diện cho tên nhóm tin Usenet, chẳng hạn như comp.lang.java.machine. Nó cũng có thể bao gồm tên máy chủ cho máy chủ tin tức. Các trạng thái của một đối tượng NewsAddress được duy trì bởi hai trường được bảo vệ: nhóm tin String được bảo vệ máy chủ chuỗi được bảo vệ Trường nhóm tin chứa tên của nhóm tin — ví dụ: netscape.devs-java. Trường máy chủ lưu trữ là rỗng hoặc chứa tên máy chủ của tin tức máy chủ — ví dụ: secnews.netscape.com. Cả hai trường này đều được đặt trong constructor. Có ba hàm tạo được nạp chồng cho các đối tượng NewsAddress:

Phương thức setNewsgroup () và setHost () đặt trường nhóm tin và máy chủ lưu trữ của đối tượng đến các giá trị được chỉ định. Phương thức getNewsgroup () và getHost () trả về các giá trị của nhóm tin và các trường máy chủ lưu trữ. Cuối cùng, phương thức getType () trả về chuỗi "tin tức". Phương thức toString () trả về tên nhóm tin trong một biểu mẫu phù hợp với Nhóm tin: trường tiêu đề của một bài đăng trên Usenet. Equals () và hashCode () các phương thức có ngữ nghĩa thông thường của chúng. Ngoài ra còn có hai phương thức tiện ích tĩnh để chuyển đổi địa chỉ thành và từ chuỗi

public static String toString(Address[] addresses) throws ClassCastException public static NewsAddress[] parse(String newsgroups) throws AddressException

Phương thức toString () chuyển đổi một mảng các đối tượng Địa chỉ thành một danh sách tên nhóm tin được phân tách bằng dấu phẩy. Một ClassCastException được ném nếu bất kỳ Đối tượng địa chỉ trong mảng không phải là đối tượng NewsAddress cụ thể hơn. Các Phương thức parse () đảo ngược thao tác này, chuyển đổi một Chuỗi được phân tách bằng dấu phẩy của tên nhóm tin tức, chẳng hạn như "comp.lang.java.programmer, comp.lang.java.gui, comp.lang.java.help", thành một mảng của các đối tượng NewsAddress. Nó ném một AddressException nếu đối số nhóm tin không phải là danh sách tên nhóm tin được phân tách bằng dấu phẩy. Việc triển khai JavaMail API của Sun không có nhà cung cấp dịch vụ cho tin tức, Tuy nhiên; vì vậy, mặc dù bạn có thể tạo địa chỉ tin tức, trước khi bạn thực sự có thể đọc và đăng tin tức, bạn sẽ cần cài đặt một nhà cung cấp dịch vụ hỗ trợ nó. Bảng 19.1 danh sách một số nguồn có thể của các nhà cung cấp tin tức. Khi bạn đã có, đọc tin tức là đơn giản như nói chuyện với máy chủ IMAP. 19.6 Lớp URLName javax.mail.URLName đại diện cho tên của một URL; nghĩa là, nó coi một URL như một chuỗi, nhưng không cố gắng kết nối hoặc giải quyết bất kỳ phần nào của chuỗi. URL tên chủ yếu được sử dụng như một cách thuận tiện để xác định các thư mục và cửa hàng với URL không chuẩn, chẳng hạn như pop3: // elharo: mypassword@mail.metalab.unc.edu: 110 / INBOX, không có trình xử lý giao thức phù hợp: Đối tượng URLName lớp công khai Các phương thức của URLName rất giống với các phương thức của java.net.URL được thảo luận trong Chương 7, ngoại trừ việc tất cả những gì liên quan đến kết nối thực tế đã bị xóa. Những gì còn lại là một loạt các phương pháp để chia một chuỗi URL thành các phần thành phần của nó hoặc xây dựng một URL từ các mảnh. 19.6.1 Các nhà xây dựng Có ba hàm tạo URLName bị quá tải. Một người lấy các phần riêng lẻ của một URL dưới dạng đối số; cái khác lấy một đối tượng java.net.URL; và một phần ba lấy một chuỗi chứa một URL: public URLName (Giao thức chuỗi, Máy chủ chuỗi, cổng int, Tệp chuỗi, String userName, String password) public URLName (URL url) public URLName (String url) Việc tạo URLName không yêu cầu trình xử lý giao thức cho lược đồ phải có sẵn. Tất cả các hoạt động trên URLName diễn ra với chuỗi con đơn giản Thao tác. Điều này cho phép lớp URLName hỗ trợ các URL rất không chuẩn như pop3: // eharold: password@utopia.poly.edu/INBOX hoặc imap: //elharo@metalab.unc.edu/Spilities/SD99West. Các đối tượng URLName này có thể là được sử dụng để chỉ các thư mục cụ thể trên máy chủ. 19.6.2 Phương pháp phân tích cú pháp Bảy phương thức getter này trả về các phần riêng lẻ của URL là chính mục đích cho lớp nàyĐối với mục đích trình diễn, chương trình này bao gồm mật khẩu trong URL. Trong nói chung, tuy nhiên, đó là một rủi ro bảo mật rất lớn. Sẽ tốt hơn nhiều nếu sử dụng thời gian chạy Authenticator như Ví dụ 19.6 đã làm. Tất nhiên, cuối cùng thì nó rất đáng nghi vấn liệu đây có thực sự là một giao diện vượt trội so với Ví dụ 19.6 và ilk của nó hay không

19.6 Lớp URLName

javax.mail.URLName đại diện cho tên của một URL; nghĩa là, nó coi một URL như một

chuỗi, nhưng không cố gắng kết nối hoặc giải quyết bất kỳ phần nào của chuỗi. URL

tên chủ yếu được sử dụng như một cách thuận tiện để xác định các thư mục và cửa hàng với

URL không chuẩn, chẳng hạn như

pop3://elharo:mypassword@mail.metalab.unc.edu:110/INBOX,điều đó không có

trình xử lý giao thức phù hợp:

Đối tượng URLName lớp công khai

Các phương thức của URLName rất giống với các phương thức của java.net.URL được thảo luận trong

Chương 7, ngoại trừ việc tất cả những gì liên quan đến kết nối thực tế đã bị xóa.

Những gì còn lại là một loạt các phương pháp để ngắt một chuỗi URL thành các phần thành phần của nó

hoặc xây dựng một URL từ các mảnh.

19.6.1 Các nhà xây dựng

Có ba hàm tạo URLName bị quá tải. Một người lấy các phần riêng lẻ của một

URL dưới dạng đối số; khác lấy một đối tượng java.net.URL; và một phần ba lấy một chuỗi

chứa một URL:

public URLName(String protocol, String host, int port, String file,

String userName, String password)

public URLName(URL url)

public URLName(String url)

Việc tạo URLName không yêu cầu trình xử lý giao thức cho lược đồ phải

có sẵn. Tất cả các hoạt động trên URLName diễn ra với chuỗi con đơn giản

Thao tác. Điều này cho phép lớp URLName hỗ trợ các URL rất không chuẩn như

pop3://eharold:password@utopia.poly.edu/INBOX or imap://elharo@metalab.unc.edu/Speaking/SD99West.

Các đối tượng URLName này có thể là

được sử dụng để chỉ các thư mục cụ thể trên máy chủ.

19.6.2 Phương pháp phân tích cú pháp

Bảy phương thức getter này trả về các phần riêng lẻ của URL là chính

mục đích cho lớp này:

public int getPort ()

public String getProtocol ()

public String getFile ()

public String getRef ()

public String getHost ()

public String getUsername ()

public String getPassword ()

Tất cả những điều này có thể dễ dàng hiểu được bằng cách tương tự với các phương pháp được đặt tên tương tự trong

java.net.URL. Ngoại trừ getPort (), tất cả đều trả về null nếu thiếu mảnh.

getPort () trả về -1 nếu cổng không được đưa vào URL một cách rõ ràng.

Ngoài ra còn có một phương thức getURL () chuyển đổi URLName thành java.net.URL. Từ

làm như vậy yêu cầu Java phải có một trình xử lý giao thức cho lược đồ của URL, phương pháp này

có thể tạo ra một MalformedURLException:

URL công khai getURL () ném MalformedURLException

Cuối cùng, có ba phương thức tiện ích thông thường với ngữ nghĩa thông thường:

công khai boolean bằng (Đối tượng o)

public int hashCode ()

public String toString ()

Phương thức toString () chỉ trả về dạng chuỗi của URL.

Chúng tôi có thể sử dụng lớp URLName để cung cấp giao diện cho ứng dụng email

hoàn toàn không phụ thuộc vào giao thức. Tất cả thông tin về giao thức, máy chủ lưu trữ và các thông tin khác

chi tiết được cung cấp bởi một URL được đọc từ dòng lệnh. Ví dụ 19.7

biểu tình.

Ví dụ 19.7. Một ứng dụng thư độc lập với giao thức

fURLName làm cho mã nhỏ gọn hơn một chút vì nó di chuyển một số thông tin

từ mã nguồn đến dòng lệnh. Bên cạnh việc loại bỏ các biến rõ ràng

và chuỗi ký tự cho tên người dùng, máy chủ lưu trữ, v.v., chúng tôi đã quản lý để loại bỏ bất kỳ

tham chiếu trực tiếp đến lớp Cửa hàng. Một cuộc chạy điển hình bắt đầu như thế này:

% java MailClient pop3://eharold:mypassword@utopia.poly.edu/INBOX

------------ Message 1 ------------

Received: (from eharold@localhost)

by utopia.poly.edu (8.8.8/8.8.8) id QAA05728

for eharold; Tue, 30 Nov 1999 16:14:29 -0500 (EST)

Date: Tue, 30 Nov 1999 16:14:29 -0500 (EST)

From: Elliotte Harold <eharold@utopia.poly.edu>

Message-Id: <199911302114.QAA05728@utopia.poly.edu>

To: eharold@utopia.poly.edu

Subject: test

Content-Type: text

X-UIDL: 87e3f1ba71738c8f772b15e3933241f0

Status: RO

hello you

Đối với mục đích trình diễn, chương trình này bao gồm mật khẩu trong URL. Trong

nói chung, tuy nhiên, đó là một rủi ro bảo mật rất lớn. Sẽ tốt hơn nhiều nếu sử dụng thời gian chạy

Authenticator như Ví dụ 19.6 đã làm. Tất nhiên, cuối cùng thì nó rất đáng nghi vấn

liệu đây có thực sự là một giao diện vượt trội so với Ví dụ 19.6 và ilk của nó hay không.

19.7 Lớp thông báo

Lớp javax.mail.Message là lớp cha trừu tượng cho tất cả các email riêng lẻ,

các bài đăng tin tức và các tin nhắn tương tự:

lớp trừu tượng công khai Thông báo mở rộng Đối tượng triển khai Phần

Có một lớp con Message cụ thể trong API JavaMail tiêu chuẩn,

javax.mail.internet.MimeMessage. Điều này được sử dụng cho cả email và tin tức Usenet

tin nhắn. Các nhà cung cấp dịch vụ có thể tự do thêm các lớp cho các định dạng tin nhắn của riêng họ. Đối với

chẳng hạn, IBM có thể cung cấp một lớp NotesMessage cho Lotus Notes.

Lớp Message chủ yếu khai báo các phương thức getter và setter trừu tượng để xác định

thuộc tính chung của hầu hết các thông báo. Các thuộc tính này bao gồm các địa chỉ của

tin nhắn, người nhận tin nhắn, chủ đề và nội dung của tin nhắn, và

nhiều thuộc tính khác. Bạn có thể coi đây là các thuộc tính của phong bì

chứa thông điệp.

Hơn nữa, lớp Message thực hiện giao diện Part. Giao diện Part

chủ yếu là xử lý nội dung của một email. Nó khai báo các phương thức để nhận và

thiết lập loại nội dung của nội dung thư, nhận và thiết lập thông báo thực tế

nội dung nội dung, nhận và đặt các tiêu đề tùy ý từ thư và nhận

các luồng đầu vào được cung cấp bởi nội dung thư. Phần nội dung chính của thư có thể

chứa các bộ phận khác. Điều này được sử dụng để xử lý các tệp đính kèm, nội dung thư

có sẵn ở nhiều định dạng và các email nhiều phần khác. Vì lớp Message là

trừu tượng và cần được phân lớp bởi các lớp cụ thể như MimeMessage, hầu hết

những phương thức này không thực sự được khai báo lại trong Message nhưng có thể được gọi bởi bất kỳ

phiên bản thực tế của Message. Chúng ta sẽ bắt đầu bằng cách thảo luận về các phương pháp thực sự được khai báo

trong Tin nhắn, sau đó chuyển sang những người được khai báo trong Phần.

19.7.1 Tạo tin nhắn

Lớp Message có ba hàm tạo:

protected Message( )

protected Message(Folder folder, int messageNumber)

protected Message(Session session)

Vì tất cả các hàm tạo đều được bảo vệ, chúng chủ yếu dành cho việc sử dụng các lớp con

chẳng hạn như MimeMessage. Nếu bạn đang gửi một tin nhắn, bạn sẽ sử dụng một trong các hàm tạo

trong lớp con thay vào đó. Nếu bạn đang đọc thư, thì Thư mục hoặc Phiên bạn

đọc từ sẽ tạo các đối tượng Message và chuyển chúng cho bạn.

19.7.1.1 Trả lời tin nhắn

Nếu bạn đã có một đối tượng Message, một cách để tạo một đối tượng Message mới là

trả lời câu hỏi hiện có bằng cách sử dụng phương thức reply ():

public abstract Message reply(boolean replyToAll)

throws MessagingException

Phương thức này tạo một đối tượng Message mới có cùng chủ đề được đặt tiền tố là "Re:"

và gửi đến người gửi thư gốc. Nếu replyToAll là true, thì

tin nhắn được gửi tới tất cả những người nhận tin nhắn gốc đã biết. Nội dung của

tin nhắn trống. Nếu bạn muốn trích dẫn thông điệp gốc, bạn sẽ phải làm điều đó

bản thân bạn.

19.7.1.2 Nhận tin nhắn từ các thư mục

Bạn đã thấy rằng khi bạn đang đọc email, JavaMail API sẽ tạo ra

Các đối tượng thông báo đại diện cho các thông báo mà nó tìm thấy trên máy chủ. Phương tiện chính

thực hiện điều này là các phương thức getMessage () và getMessages () trong Thư mục

lớp học:

public abstract Message getMessage(int messageNumber)

throws MessagingException

public Message[] getMessages(int start, int end)

throws MessagingException

public Message[] getMessages(int[] messageNumbers)

throws MessagingException

public Message[] getMessages( ) throws MessagingException

Ba phương thức đầu tiên cho phép người gọi chỉ định thông điệp nào nó muốn. Cuối cùng

chỉ cần trả lại tất cả các thư trong thư mục. Những gì thực sự được trả lại là những thứ đang giữ lại

vị trí của các thông điệp thực tế. Văn bản và tiêu đề của tin nhắn sẽ không

nhất thiết phải được truy xuất cho đến khi một số phương thức của lớp Message được gọi ra

yêu cầu thông tin này.

19.7.2 Thông tin tiêu đề cơ bản

Thông báo RFC 822 điển hình chứa tiêu đề trông giống như sau:

From levi@blazing.sunspot.noao.edu Fri Aug 5 10:57:08 1994

Date: Fri, 5 Aug 1994 10:57:04 +0700

From: levi@blazing.sunspot.noao.edu (Denise Levi)

To: volleyball@sunspot.noao.edu

Subject: Apologies

Content-Length: 517

Status: RO

X-Lines: 13

Các trường chính xác có thể khác nhau, nhưng hầu hết các thư đều chứa ít nhất trường Từ:, trường Đến:,

trường Ngày: và trường Chủ đề:. Các trường phổ biến khác bao gồm Cc: (bản sao carbon)

và Bcc: (bản carbon mù). Nói chung, những thứ này sẽ có thể truy cập được thông qua getter và

các phương pháp setter.

**19.7.2.1 Địa chỉ Từ**

Bốn phương pháp này cho phép bạn lấy và đặt trường Từ: của thư:

public abstract Address[] getFrom( ) throws MessagingException

public abstract void setFrom( ) throws MessagingException,

IllegalWriteException, IllegalStateException

public abstract void setFrom(Address address)

throws MessagingException, IllegalWriteException,

IllegalStateException

public abstract void addFrom(Address[] addresses)

throws MessagingException, IllegalWriteException,

IllegalStateException

Phương thức getFrom () trả về một mảng các đối tượng Địa chỉ, một đối tượng cho mỗi địa chỉ

được liệt kê trong tiêu đề Từ:. (Trong thực tế, rất hiếm khi có một tin nhắn từ nhiều hơn

một địa chỉ. Việc một tin nhắn được gửi tới nhiều hơn một người là điều khá phổ biến

địa chỉ.) Nó trả về null nếu tiêu đề From: không có trong thư. Nó ném một

MessagingException nếu tiêu đề From: không đúng định dạng theo một cách nào đó.

Các phương thức noargs setFrom () và addFrom () đặt và sửa đổi các tiêu đề From:

trong số các tin nhắn email gửi đi. Phương thức noargs setFrom () đặt tiêu đề thành

giá trị hiện tại của thuộc tính mail.user hoặc, dưới dạng dự phòng, thuộc tính user.name.

Phương thức setFrom () với các đối số đặt giá trị của tiêu đề From: thành

các địa chỉ được liệt kê. Phương thức addFrom () thêm các địa chỉ được liệt kê vào bất kỳ địa chỉ nào

đã tồn tại trong tiêu đề. Cả ba phương pháp này đều có thể tạo ra một

MessagingException nếu một trong những địa chỉ họ sử dụng không đúng định dạng. Họ

cũng có thể ném một IllegalWriteException nếu trường From: của Thông báo đã cho

không thể thay đổi đối tượng hoặc IllegalStateException nếu toàn bộ đối tượng Message

là chỉ đọc.

19.7.2.2 Địa chỉ trả lời

Một số thư chứa tiêu đề Reply-to: cho biết rằng bất kỳ thư trả lời nào sẽ được gửi đi

đến một địa chỉ khác với địa chỉ đã gửi tin nhắn. Có hai phương pháp để thiết lập

và lấy các địa chỉ sau:

public Address[] getReplyTo( ) throws MessagingException

public void setReplyTo(Address[] addresses) throws MessagingException,

MethodNotSupportedException, IllegalWriteException,

IllegalStateException

Ngữ nghĩa của các phương thức này giống như đối với getFrom () và

phương thức setFrom () — trên thực tế, việc triển khai mặc định của getReplyTo () chỉ đơn giản là

trả về getFrom () - với một cảnh báo duy nhất rằng một triển khai không

hỗ trợ Reply-to: địa chỉ riêng biệt có thể ném ra một MethodNotSupportedException

khi setReplyTo () được gọi.

19.7.2.3 Địa chỉ người nhận

Trong khi người gửi thư thường chỉ được tìm thấy trong tiêu đề From:,

người nhận thư thường được chia thành các trường Đến :, Cc :, và Bcc:. Hơn

ngoài việc cung cấp các phương pháp riêng biệt cho từng trường này, các

Các phương thức getRecipient () và setRecipient () dựa trên một

Đối số Message.RecipientType để xác định giá trị của trường nào được mong muốn.

RecipientType là một lớp bên trong công khai trong javax.mail.Message có lớp riêng tư

hàm tạo giới hạn nó trong chính xác ba đối tượng tĩnh sau:

Message.RecipientType.TO

Message.RecipientType.CC

Message.RecipientType.BCC

Có hai phương pháp để tìm người nhận của Thư:

public abstract Address[]getRecipients(Message.RecipientType type)

throws MessagingException

public Address[] getAllRecipients( ) throws MessagingException

Phương thức getRecipient () trả về một mảng các đối tượng Địa chỉ, mỗi đối tượng một

địa chỉ được liệt kê trong tiêu đề được chỉ định. Nó trả về null nếu tiêu đề được chỉ định không phải là

hiện trong tin nhắn. Nó ném ra một MessagingException nếu tiêu đề được chỉ định là

bị dị dạng theo một cách nào đó. Phương thức getAllRecipient () cũng làm như vậy ngoại trừ

rằng nó kết hợp nội dung của các tiêu đề To :, Cc :, và Bcc:.

Có hai phương pháp để đặt người nhận thư trong khi thay thế bất kỳ

người nhận trước đó và hai phương pháp để thêm người nhận vào thư:

public abstract void setRecipients(Message.RecipientType type,

Address[] addresses) throws MessagingException,

IllegalWriteException,

IllegalStateException

public void setRecipient(Message.RecipientType type, Address address)

throws MessagingException, IllegalWriteException

public abstract void addRecipients(Message.RecipientType type,

Address[] addresses) throws MessagingException,

IllegalWriteException, IllegalStateException

public void addRecipient(Message.RecipientType type, Address address)

throws MessagingException, IllegalWriteException

Tất cả bốn phương thức này đều có thể tạo ra một ngoại lệ MessagingException, thường là do một trong các

địa chỉ được sử dụng không đúng định dạng. Họ cũng có thể ném một

IllegalWriteException nếu không thể trường được chỉ định của đối tượng Message đã cho

đã thay đổi hoặc IllegalStateException nếu toàn bộ đối tượng Message ở chế độ chỉ đọc.

19.7.2.4 Chủ đề của thông điệp

Vì chủ đề chỉ đơn giản là một chuỗi văn bản, nên rất dễ dàng để đặt và lấy

hai phương pháp này:

public abstract String getSubject( ) throws MessagingException

public abstract void setSubject(String subject) throws

MessagingException, IllegalWriteException, IllegalStateException

Như với các phương thức setter trước đó, null được trả về nếu trường chủ đề không có trong

thông điệp. IllegalWriteException được ném ra nếu chương trình không được phép thiết lập

giá trị của trường Subject: và IllegalStateException được ném nếu

chương trình hoàn toàn không được phép thay đổi thông báo.

19.7.2.5 Ngày của tin nhắn

Tin nhắn cũng có ngày gửi và nhận. Ba phương pháp cho phép các chương trình truy cập

các trường này:

public abstract Date getSentDate( ) throws MessagingException

public abstract void setSentDate(Date date) throws MessagingException,

IllegalWriteException, IllegalStateException

public abstract Date getReceivedDate( ) throws MessagingException

Việc triển khai cơ bản chịu trách nhiệm chuyển đổi định dạng ngày văn bản

được tìm thấy trong tiêu đề thư như "Thứ sáu, ngày 5 tháng 8 năm 2000 10:57:04 +0700" tới

đối tượng java.util.Date. Như thường lệ, MessagingException chỉ ra một số vấn đề

với định dạng của thông báo bên dưới; một IllegalWriteException chỉ ra

rằng trường không thể thay đổi; và IllegalStateException chỉ ra rằng

toàn bộ tin nhắn không thể được thay đổi.

Ví dụ 19.8

là một chương trình ví dụ đơn giản tuân theo mẫu cơ bản của

một số chương trình đọc thư. Tuy nhiên, cái này không còn sử dụng writeTo ().

Thay vào đó, nó sử dụng các phương pháp trong phần này để chỉ in các tiêu đề. Hơn nữa, nó

in chúng theo một thứ tự cụ thể bất kể thứ tự của chúng trong thông điệp thực tế trên

người phục vụ. Cuối cùng, nó bỏ qua các tiêu đề ít quan trọng hơn như X-UIDL: và Trạng thái:. Các

Phương thức static InternetAddress.toString () chuyển đổi các mảng mà hầu hết các

các phương thức trả về thành các chuỗi đơn giản, được phân tách bằng dấu phẩy.

Ví dụ 19.8. Chương trình đọc tiêu đề thư

Đây là một số kết quả tiêu biểu. Một số chuỗi được yêu cầu là rỗng vì

chỉ đơn giản là các trường không có trong thư trong INBOX; ví dụ, Cc: và

Bcc:. HeaderClient kiểm tra điều đó và chỉ cần bỏ qua các trường nếu chúng không hiện diện.

% java HeaderClient pop3: //eharold@utopia.poly.edu/INBOX

% java HeaderClient pop3://eharold@utopia.poly.edu/INBOX

------------ Message 1 ------------

From: Elliotte Harold <eharold@utopia.poly.edu>

Reply-to: Elliotte Harold <eharold@utopia.poly.edu>

To: eharold@utopia.poly.edu

Subject: test

Sent: Tue Nov 30 13:14:29 PST 1999

------------ Message 2 ------------

From: Elliotte Rusty Harold <elharo@macfaq.com>

Reply-to: Elliotte Rusty Harold <elharo@macfaq.com>

To: eharold@utopia.poly.edu

Subject: New system

Sent: Wed Dec 01 10:55:40 PST 1999

------------ Message 3 ------------

From: Dr. Mickel <Greatsmiles@mail.com>

Reply-to: Dr. Mickel <Greatsmiles@mail.com>

To: eharold@utopia.poly.edu

Subject: Breath RX Products now available Online!

Sent: Thu Dec 02 03:45:52 PST 1999

Lưu ý rằng không có tin nhắn nào trong số này có ngày nhận được. Đó là bởi vì nhận được

thời gian không phải là một phần của phong bì thư. Nó phải được cung cấp bởi máy chủ và

Máy chủ POP không cung cấp nó. Một máy chủ IMAP sẽ có nhiều khả năng bao gồm một

ngày nhận được, như sẽ được hiển thị trong Ví dụ 19.9.

19.7.2.6 Lưu thay đổi

Khi bạn gọi một trong các bộ trước đó hoặc thêm phương thức, một số triển khai sẽ

lưu trữ các thay đổi ngay lập tức. Tuy nhiên, những người khác có thể không. SaveChanges ()

phương thức cam kết các thay đổi được thực hiện đối với đối tượng Message:

public abstract void saveChanges () ném MessagingException,

IllegalWriteException, IllegalStateException

Đây không phải là một sự tuôn ra. Các thay đổi thực tế có thể không được cam kết cho đĩa cho đến khi

thư mục chứa tin nhắn bị đóng. Tuy nhiên, phương pháp này đảm bảo rằng

các thay đổi được lưu trữ trong thư mục và chúng sẽ được lưu khi thư mục được lưu.

19.7.3 Cờ

Các chương trình thư có thể lưu thêm thông tin về các thư không phải là một phần của

tin nhắn của chính họ. Ví dụ: Pine cho tôi biết liệu tôi đã trả lời

tin nhắn, cho dù tôi đã đọc một tin nhắn, v.v. Như Hình 19.5 cho thấy, đây là

được biểu thị bằng các ký hiệu và chữ cái trong cột bên trái. D có nghĩa là một tin nhắn đã được

đã xóa; A có nghĩa là nó đã được trả lời; N là một tin nhắn mới chưa được đọc;

và kể từ đó trở đi. Trong JavaMail API, tất cả chúng đều được biểu diễn dưới dạng cờ. Một lá cờ là một

phiên bản của lớp javax.mail.Flags:

lớp công khai Cờ mở rộng Đối tượng triển khai Có thể sao chép

Bảy cờ được xác định trước như là các thể hiện của lớp bên trong tĩnh công khai Flags.Flag.

Đó là:

Flags.Flag.ANSWERED

Flags.Flag.DELETED

Flags.Flag.DRAFT

Flags.Flag.FLAGGED

Flags.Flag.RECENT

Flags.Flag.SEEN

Flags.Flag.USER

Ngoài ra, một số triển khai có thể cho phép các cờ do người dùng xác định tùy ý. Nếu vậy,

Cờ USER sẽ được đặt.

Hình 19.5.

Pine hiển thị cờ dưới dạng các chữ cái trong cột bên trái

Phương thức getFlags () trả về các cờ của một thông báo cụ thể:

public trừu tượng Flags getFlags () ném MessagingException

Phương thức isSet () kiểm tra xem một cờ cụ thể có được đặt cho thông báo đã cho hay không:

public boolean isSet (cờ Flags.Flag) ném MessagingException

Cuối cùng, các phương thức setFlags () và setFlag () được đặt hoặc không được đặt (tùy thuộc vào

đối số thứ hai) cờ được chỉ ra bởi đối số đầu tiên:

public abstract void setFlags(Flags flag, boolean set)

throws MessagingException, IllegalWriteException,

IllegalStateException

public void setFlag(Flags.Flag flag, boolean set) throws

MessagingException, IllegalWriteException, IllegalStateException

Bạn xóa tin nhắn bằng cách đặt cờ Flags.Flag.DELETED của chúng thành true. Ví dụ,

để xóa tin nhắn:

message.setFlag (Flags.Flag.DELETED, true);

Điều này chỉ đánh dấu tin nhắn là đã bị xóa. Nó không thực sự xóa nó khỏi tệp

trên máy chủ. Cho đến khi tin nhắn bị xóa, nó vẫn có thể bị xóa bằng cách cài đặt

Flags.Flag.DELETED trở lại false.

Ví dụ 19.9

là một sửa đổi nhỏ của Ví dụ 19.8, HeaderClient, in ra

cờ cũng vậy. Theo quy tắc chung, máy chủ POP sẽ không báo cáo cờ. Chỉ một giao thức mà

lưu trữ tin nhắn và chuyển tiếp chúng, chẳng hạn như IMAP hoặc mbox, sẽ báo cáo cờ.

Ví dụ 19.9. Chương trình đọc cờ hộp thư

Đây là một cuộc chạy mẫu. Tin nhắn đầu tiên đã được đọc và bị xóa. Thư hai

tin nhắn không có cờ thiết lập. Nó chưa được đọc, bị xóa hoặc được trả lời. Tin nhắn thứ ba

đã được đọc và trả lời nhưng không bị xóa. Lưu ý rằng tôi đang sử dụng máy chủ IMAP

thay vì máy chủ POP:

% java FlagsClient imap://elharo@mail.metalab.unc.edu/INBOX

------------ Message 1 ------------

From: Mike Hall <mikehall@spacestar.com>

Reply-to: Mike Hall <mikehall@spacestar.com>

To: mrj-dev@public.lists.apple.com

Subject: Re: dialog box, parents & X-platform

Sent: Mon Dec 13 05:24:38 PST 1999

Received: Mon Dec 13 06:33:00 PST 1999

Deleted

Read

------------ Message 2 ------------

From: Kapil Madan <kapil.madan@MIT-MISYS.COM>

Reply-to: XML-INTEREST@JAVA.SUN.COM

To: XML-INTEREST@JAVA.SUN.COM

Subject: Re: first mail to the list!

Sent: Mon Dec 13 06:19:46 PST 1999

Received: Mon Dec 13 06:40:00 PST 1999

------------ Message 3 ------------

From: Jim Jackl-Mochel <jmochel@foliage.com>

Reply-to: Jim Jackl-Mochel <jmochel@foliage.com>

To: elharo@metalab.unc.edu

Subject: CPreProcessorStream

Sent: Mon Dec 13 07:14:00 PST 1999

Received: Mon Dec 13 07:08:00 PST 1999

Answered

Read

19.7.4 Thư mục

Tin nhắn nhận được từ mạng (trái ngược với được gửi đến mạng) sẽ

thường thuộc về một số Thư mục. Phương thức getFolder () trả về một tham chiếu đến

đối tượng Thư mục có chứa Thông báo này:

public Folder getFolder ()

Nó trả về null nếu thư không được chứa trong một thư mục.

Trong một thư mục, các thư được sắp xếp từ đầu tiên (thư 1) đến cuối cùng. Các

Phương thức getMessageNumber () trả về vị trí tương đối của Thông báo này trong

Thư mục:

public int getMessageNumber ()

Tin nhắn không nằm trong bất kỳ thư mục nào có số 0. Số tin nhắn có thể thay đổi

trong khi một chương trình đang chạy nếu các thư khác được thêm vào hoặc xóa khỏi một thư mục.

Ngoài ra còn có một phương thức setMessageNumber () được bảo vệ, nhưng nó chỉ dành cho dịch vụ

nhà cung cấp, không dành cho mã người dùng:

void setMessageNumber được bảo vệ (số int)

Chúng ta sẽ nói thêm về các thư mục và những gì chúng có thể làm ở cuối chương này. Một trong

những điều bạn có thể làm với một thư mục là xóa các thư khỏi nó. Vật lý này

xóa tin nhắn nếu nó đã được đánh dấu là đã xóa. (Một tin nhắn đơn thuần đã bị xóa

có thể là "không bị xóa", trong khi không thể xóa thư.) Nếu một thư là

bị xóa, vẫn có thể có một đối tượng Message trỏ đến tin nhắn nhưng hầu như tất cả

các phương thức trên tin nhắn sẽ ném ra một ngoại lệ MessagingException. Vì vậy, nó có thể được

quan trọng là kiểm tra xem một thư đã được xóa trước khi làm việc với nó hay chưa. Các

Phương thức isExpunged () thực hiện điều đó:

public boolean isExpunged ()

Ngoài ra còn có một phương thức setExpunged () được bảo vệ, nhưng nó chỉ dành cho các nhà cung cấp dịch vụ,

không dành cho mã người dùng:

void setExpunged được bảo vệ (boolean bị xóa)

19.7.5 Tìm kiếm

Phương thức cuối cùng còn lại trong lớp Message là match (). Phương thức match () được sử dụng

để xác định xem một Thông báo có đáp ứng các tiêu chí tìm kiếm cụ thể hay không. Chúng ta sẽ thảo luận về điều này

chi tiết hơn khi chúng ta nói về tìm kiếm thư mục:

đối sánh boolean công khai (cụm từ SearchTerm) ném MessagingException

19.8 Giao diện bộ phận

Giao diện Part được thực hiện bởi cả Message và BodyPart. Mỗi tin nhắn là

riêng biệt. Tuy nhiên, một số bộ phận có thể chứa các bộ phận khác. Giao diện Part khai báo

ba loại phương pháp:

• Các phương pháp lấy và thiết lập các thuộc tính của bộ phận

• Các phương pháp lấy và thiết lập các tiêu đề của phần

• Các phương pháp lấy và thiết lập nội dung của phần

Các thuộc tính của phần là những thứ như kích thước của thư hoặc ngày tháng

nhận. Những điều này không được chỉ định rõ ràng trong tiêu đề của thư. Các tiêu đề, bởi

tương phản, là các cặp tên-giá trị được bao gồm ở phía trước của phần. Cuối cùng, nội dung của

phần là dữ liệu thực tế mà thông báo đang cố gắng truyền.

19.8.1 Thuộc tính

API JavaMail xác định năm thuộc tính cho các bộ phận. Đó là:

Kích thước

Số byte gần đúng trong phần

Số dòng

Số dòng trong phần

Bố trí

Phần này có phải là phần đính kèm hay không nên được hiển thị nội dòng

Sự miêu tả

Tóm tắt văn bản ngắn gọn của phần

Tên tệp

Tên của tệp mà tệp đính kèm đến từ

Không phải tất cả các bộ phận đều có tất cả các thuộc tính. Ví dụ: một phần không đại diện cho một

tệp không chắc có thuộc tính tên tệp. Mỗi thuộc tính được ánh xạ tới một getter

phương pháp:

public int getSize( ) throws MessagingException

public int getLineCount( ) throws MessagingException

public String getDisposition( ) throws MessagingException

public String getDescription( ) throws MessagingException

public String getFileName( ) throws MessagingException,

ParseException

Nói chung, phương thức getter trả về null hoặc -1 nếu một phần không sở hữu

thuộc tính. Nó ném ra một MessagingException nếu có một số vấn đề khi truy xuất

thông điệp; ví dụ: nếu kết nối bị gián đoạn trong khi tin nhắn đang được

lấy lại.

Phương thức getSize () trả về số byte gần đúng trong phần.

Tùy thuộc vào máy chủ và giao thức, điều này có thể giải thích cho những thay đổi trong

kích thước do các hoạt động như mã hóa dữ liệu Base64 gây ra.

Phương thức getLineCount () trả về số dòng gần đúng trong nội dung

của bộ phận hoặc -1 nếu số dòng không xác định. Một lần nữa, số được trả lại có thể hoặc

có thể không tính đến những thay đổi về kích thước của bộ phận do mã hóa của bộ phận đó gây ra.

Phương thức getDisposition () trả về một chuỗi cho biết nội dung

nên được trình bày nội dòng hoặc dưới dạng tệp đính kèm. Giá trị trả về phải là

null (không xác định được vị trí) hoặc một trong hai hằng số có tên Part.INLINE hoặc

Part.ATTACHMENT:

public static final String ATTACHMENT = "attachment";

public static final String INLINE = "inline";

Nếu định vị là Part.ATTACHMENT, thì phương thức getFileName () sẽ

trả về tên của tệp để lưu tệp đính kèm vào. Nếu không, getFileName () sẽ

có thể trả về null. Tuy nhiên, một số ứng dụng email, bao gồm Netscape 4.5 cho

Windows, không đặt đúng tiêu đề Định vị nội dung cho tệp đính kèm.

Do đó, khi nhận được tin nhắn có tệp đính kèm được gửi bởi Navigator,

bạn sẽ thường nhận được bố cục rỗng nhưng tên tệp không phải null. Trong thực tế, nó có vẻ nhiều hơn

đáng tin cậy để giả định rằng bất kỳ phần cơ thể nào có tên tệp không rỗng đều là tệp đính kèm

bất kể tiêu đề Định vị nội dung và bất kỳ phần nội dung nào không có tên tệp và

không có tiêu đề Định vị nội dung nào được hiển thị nội dòng nếu có thể. Nếu nó không phải

có thể — ví dụ: nếu bạn không thể xử lý kiểu MIME — thì bạn có thể yêu cầu

người dùng cho tên tệp hoặc chọn một số mặc định hợp lý, chẳng hạn như tệp đính kèm1.tif.

Thông thường, tên tệp chỉ bao gồm tên thực của tệp nhưng không bao gồm bất kỳ

thư mục chứa tệp đó. Ứng dụng nhận thông báo tới

quyết định nơi đặt tệp tin đến. Ví dụ, Eudora thường lưu trữ

tệp đính kèm trong thư mục Tệp đính kèm bên trong thư mục Eudora. Tuy nhiên, người dùng có

một tùy chọn để chọn một vị trí khác. Vì không có gì lạ khi nhận được nhiều

các tệp đính kèm có cùng tên theo thời gian (vcard.vcf là một

đính kèm), hãy luôn kiểm tra xem tệp có tên tệp đính kèm đã

tồn tại trước khi viết ra tệp đính kèm. Nếu tồn tại một tệp có tên tương tự, bạn sẽ

phải đổi tên tệp đính kèm theo một số cách hợp lý; ví dụ, bằng cách bổ sung

một 1 hoặc một 2 cho nó; ví dụ: vcard1.vcf, vcard2.vcf, v.v.

Các thuộc tính mô tả, bố cục và tên tệp cũng có các phương thức setter.

Tuy nhiên, các thuộc tính kích thước và số dòng được xác định bởi nội dung của phần

chứ không phải là một phương thức setter:

public void setDisposition(String disposition) throws

MessagingException, IllegalWriteException, IllegalStateException

public void setFileName(String filename) throws MessagingException,

IllegalWriteException, IllegalStateException

public void setDescription(String description) throws

MessagingException, IllegalWriteException, IllegalStateException

Tất cả các phương thức setter đều ném ra một ngoại lệ MessagingException nếu có một số vấn đề trong khi

thay đổi tin nhắn. Họ cũng có thể ném IllegalWriteException nếu

thuộc tính liên quan của bộ phận không được phép sửa đổi hoặc

IllegalStateException nếu phần thuộc về thư mục chỉ đọc.

Phương thức setDisposition () xác định xem phần có được xem nội tuyến hay không

hoặc dưới dạng tệp đính kèm. Mặc dù nó được tuyên bố lấy một Chuỗi làm đối số, điều này

Chuỗi phải là một trong hai hằng số có tên Part.INLINE hoặc Part.ATTACHMENT.

Các phần là tệp đính kèm nói chung sẽ có tên tệp được bao gồm trong siêu thông tin của chúng. Tên này có thể được đặt bằng phương thức setFileName (). cuối cùng

setDescriptionMethod () có thể lấy bất kỳ Chuỗi nào để thêm mô tả vào một phần.

Ví dụ 19.10

là một chương trình đơn giản kết nối với máy chủ thư và đọc

thuộc tính của thư trong hộp thư. Vì mỗi tin nhắn tự nó là một phần (ngay cả khi nó

chứa các phần khác), chúng ta có thể gọi các phương thức này trên toàn bộ thông báo.

Ví dụ 19.10. Chương trình đọc thuộc tính thư

Đây là một số kết quả tiêu biểu. Tôi đã sử dụng máy chủ IMAP vì hầu hết các phương pháp này

gần như không hoạt động tốt với máy chủ POP. Máy chủ IMAP có thể cung cấp cho bạn các thuộc tính

của một thư mà không yêu cầu bạn tải xuống toàn bộ thư, nhưng các máy chủ POP

không phức tạp như vậy:

% java AttributeClient imap://elharo@mail.sunsite.unc.edu/INBOX

------------ Message 1 ------------

From: "Richman, Jeremy" <jrichman@hq.ileaf.com>

To: 'xsl-list' <XSL-List@mulberrytech.com>

Subject: Re: New twist: eliminating nodes with duplicate content

Sent: Mon Dec 06 08:37:51 PST 1999

This message is approximately 3391 bytes long.

This message has approximately 87 lines.

------------ Message 2 ------------

From: schererm@us.ibm.com

To: Unicode List <unicode@unicode.org>

Subject: Re: Number ordering

Sent: Mon Dec 06 11:00:28 PST 1999

This message is approximately 1554 bytes long.

This message has approximately 18 lines.

------------ Message 3 ------------

From: John Posner <jjp@connix.com>

To: 'Nakita Watson' <nakita@oreilly.com>

Subject: RE: Another conference Call

Sent: Mon Dec 06 11:16:38 PST 1999

This message is approximately 1398 bytes long.

This message has approximately 19 lines.

19.8.2 Tiêu đề

Các lớp triển khai giao diện Part — ví dụ: Message — thường khai báo

các phương thức để trả về các tiêu đề cụ thể như To: hoặc From :. Giao diện Part, bởi

Ngược lại, khai báo các phương thức để lấy và đặt các tiêu đề tùy ý bất kể tên.

Phương thức getHeader () nhận giá trị của tất cả các tiêu đề có tên

khớp với đối số tên. Một số tiêu đề như Đã nhận: có thể có nhiều

và có thể được đưa vào một thông báo nhiều lần, vì vậy phương thức này trả về

các giá trị dưới dạng một mảng chuỗi. Nó trả về null nếu không có tiêu đề nào có tên đó hiện diện trong

phần này:

public String [] getHeader (String name) ném MessagingException

Phương thức setHeader () thêm tiêu đề mới vào thư gửi đi:

public void setHeader (String name, String value) ném

MessagingException, IllegalWriteException, IllegalStateException

Nếu đã có một tiêu đề với tên này, thì tiêu đề đó sẽ bị xóa và một tiêu đề mới

được chèn vào vị trí của nó — trừ khi thư mục chứa thư ở đó chỉ được đọc, trong

trường hợp nào thì một IllegalStateException được ném ra.

Ngược lại, phương thức addHeader () thêm tiêu đề với tên được chỉ định nhưng không

không thay thế bất kỳ tồn tại:

public void addHeader (String name, String value) ném

MessagingException, IllegalWriteException, IllegalStateException

Phương thức removeHeader () xóa tất cả các trường hợp của tiêu đề được đặt tên khỏi phương thức này

Phần:

public void removeHeader (String name) ném MessagingException,

IllegalWriteException, IllegalStateException

Phương thức getAllHeaders () trả về một đối tượng java.util.Enumeration

chứa tất cả các tiêu đề trong thư này:

public Enumeration getAllHeaders () ném MessagingException

Enumeration chứa một đối tượng javax.mail.Header cho mỗi tiêu đề trong

thông điệp.

lớp công khai Header mở rộng đối tượng

Lớp Header rất đơn giản chỉ với một hàm tạo để đặt tên và giá trị của

header và các phương thức getName () và getValue () để trả về chúng:

public Header(String name, String value)

public String getName( )

public String getValue( )

Cuối cùng, phương thức getMatchingHeaders () trả về một Enumeration chứa tất cả

tiêu đề trong thư này có tên là một trong các chuỗi trong tên đối số

mảng. Phương thức getNonMatchingHeaders () trả về một Enumeration chứa

tất cả các tiêu đề trong thư này có tên không phải là một trong các chuỗi trong đối số

mảng tên. Một lần nữa, Enumeration chứa các đối tượng Header:

public Enumeration getMatchingHeaders(String[] names)

throws MessagingException

public Enumeration getNonMatchingHeaders(String[] names)

throws MessagingException

Bạn có thể nhớ rằng Ví dụ 19.8, HeaderClient, chỉ được in một số

các tiêu đề, chẳng hạn như Tới: và Từ:. Với các phương thức của giao diện Part (Thông báo đó

triển khai), thật dễ dàng để mở rộng điều này để bao gồm tất cả các tiêu đề trong thư, cho dù

biết trước hay không. Ví dụ 19.11 chứng minh. Điều này quan trọng bởi vì

Email Internet có thể chứa các tiêu đề tùy ý. Nó không chỉ giới hạn ở một vài tiêu đề

được đề cập trong các RFC có liên quan. Ví dụ: một số ứng dụng thư đồ họa cho X

Windows sử dụng tiêu đề X-Face: hoàn toàn không chuẩn có giá trị là 48 pixel ×

Ảnh bitmap 48 pixel, đen trắng, không mã hóa sắc thái của người gửi. Khác

khách hàng sử dụng tiêu đề tùy chỉnh cho các mục đích nghiêm trọng hơn và ngớ ngẩn hơn.

Ví dụ 19.11. Chương trình đọc tiêu đề thư

Here's a typical run:

% java AllHeaderClient pop3://eharold@utopia.poly.edu/INBOX

------------ Message 1 ------------

Received: (from eharold@localhost)

by utopia.poly.edu (8.8.8/8.8.8) id QAA05728

for eharold; Tue, 30 Nov 1999 16:14:29 -0500 (EST)

Date: Tue, 30 Nov 1999 16:14:29 -0500 (EST)

From: Elliotte Harold <eharold@utopia.poly.edu>

Message-Id: <199911302114.QAA05728@utopia.poly.edu>

To: eharold@utopia.poly.edu

Subject: test

Content-Type: text

X-UIDL: 87e3f1ba71738c8f772b15e3933241f0

Status: RO

------------ Message 2 ------------

Received: from russian.cloud9.net (russian.cloud9.net [168.100.1.4])

by utopia.poly.edu (8.8.8/8.8.8) with ESMTP id OAA28428

for <eharold@utopia.poly.edu>; Wed, 1 Dec 1999 14:05:06 -0500

(EST)

Received: from [168.100.203.234] (macfaq.dialup.cloud9.net

[168.100.203.234])

by russian.cloud9.net (Postfix) with ESMTP id 24B93764F8

for <eharold@utopia.poly.edu>; Wed, 1 Dec 1999 14:02:50 -

0500 (EST)

Mime-Version: 1.0

X-Sender: macfaq@mail.cloud9.net

Message-Id: <v04210100b46b1f97969d@[168.100.203.234]>

Date: Wed, 1 Dec 1999 13:55:40 -0500

To: eharold@utopia.poly.edu

From: Elliotte Rusty Harold <elharo@macfaq.com>

Subject: New system

Content-Type: text/plain; charset="us-ascii" ; format="flowed"

X-UIDL: 01fd5cbcf1768fc6c28f9c8f934534b5

Status: RO

------------ Message 3 ------------

Received: from russian.cloud9.net (russian.cloud9.net [168.100.1.4])

by utopia.poly.edu (8.8.8/8.8.8) with ESMTP id HAA17345

for <eharold@utopia.poly.edu>; Thu, 2 Dec 1999 07:55:04 -0500

(EST)

Received: from [168.100.203.234] (macfaq.dialup.cloud9.net

[168.100.203.234])

by russian.cloud9.net (Postfix) with ESMTP id C036A7630E

for <eharold@utopia.poly.edu>; Thu, 2 Dec 1999 07:54:58 -

0500 (EST)

Mime-Version: 1.0

X-Sender: elharo@luna.oit.unc.edu

Message-Id: <v04210100b46c0c686ecc@[168.100.203.234]>

Date: Thu, 2 Dec 1999 06:45:52 -0500

To: eharold@utopia.poly.edu

From: "Dr. Mickel" <Greatsmiles@mail.com>(by way of Elliotte Rusty

Harold)

Subject: Breath RX Products now available Online!

Sender: elharo@metalab.unc.edu

Content-Type: text/plain; charset="us-ascii" ; format="flowed"

X-UIDL: 40fa8af2aca1a8c11994f4c56b792720

Status: RO

19.8.3 Nội dung

Mỗi phần có một nội dung nhất định có thể được biểu diễn dưới dạng một chuỗi các byte. Đối với

ví dụ, trong một phần đó là một thông điệp email đơn giản, nội dung là phần thân của

thông điệp. Tuy nhiên, trong tin nhắn nhiều phần, nội dung này có thể chứa các phần khác.

Nội dung của mỗi phần này có thể được biểu diễn dưới dạng một chuỗi các byte.

Hơn nữa, chuỗi byte này có thể đại diện cho một số loại nội dung cụ thể hơn,

chẳng hạn như ảnh GIF chưa mã hóa hoặc clip âm thanh WAV được mã hóa Base64.

19.8.3.1 Đọc nội dung của phần

Giao diện Phần khai báo hai phương pháp để xác định kiểu nội dung MIME của một phần.

Phương thức getContentType () trả về kiểu nội dung MIME của phần này dưới dạng

chuỗi; ví dụ, văn bản / thuần túy; charset = "us-ascii"; format = "flow". Nó

trả về null nếu không xác định được loại nội dung:

public String getContentType () ném MessagingException

Phương thức isMimeType () trả về true nếu phần này có kiểu MIME được chỉ định và

kiểu con. Các tham số bổ sung, chẳng hạn như bộ ký tự, bị bỏ qua:

public boolean isMimeType (String mimeType) ném MessagingException

Giao diện Part cũng khai báo một số phương thức trả về nội dung dưới dạng

các đối tượng Java khác nhau bao gồm InputStream, String, DataHandler, v.v. Các

Phương thức getInputStream () trả về một InputStream mà từ đó nội dung của phần

có thể được đọc:

public InputStream getInputStream () ném IOException,

MessagingException

Nếu nội dung của phần đó đã được mã hóa theo một cách nào đó — ví dụ: bởi Base64

mã hóa nó — sau đó InputStream đọc nội dung được giải mã. API JavaMail

hỗ trợ tất cả các mã hóa phổ biến ngoại trừ định dạng BinHex được sử dụng cho Macintosh

các tập tin. Nếu nó gặp phải tệp đính kèm được mã hóa BinHex, thì nó sẽ loại bỏ các tiêu đề MIME

nhưng nếu không sẽ để dữ liệu BinHex không bị ảnh hưởng. Tài liệu BinHex rất khó xử lý

trên hầu hết các nền tảng vì tính chất hai nhánh bất thường của tệp Mac. Trừ khi

bạn là một chuyên gia Mac thực sự, có lẽ bạn nên sử dụng tiện ích của bên thứ ba, chẳng hạn như

StuffIt Expander (http://www.aladdinsys.com/) để giải mã tệp.

Một khả năng khác là yêu cầu một DataHandler cho nội dung có

phương thức getDataHandler (). Lớp DataHandler đến từ Java

Khung kích hoạt. Nó khai báo các phương pháp để giúp bạn quyết định phải làm gì với

Nội dung; ví dụ: bằng cách tìm ứng dụng Java hoặc ứng dụng trợ giúp phù hợp để hiển thị

Nội dung:

public javax.activation.DataHandler getDataHandler ()

ném MessagingException

Khả năng thứ ba là yêu cầu nội dung dưới dạng một đối tượng Java không xác định bằng cách sử dụng

phương thức getContent ():

public Object getContent () ném IOException, MessagingException

Điều này gợi nhớ đến phương thức getContent () của java.net.URL. Tuy nhiên, thay

hơn là dựa vào cơ chế xử lý nội dung được thiết kế kém, getContent () này

phương thức sử dụng Khung kích hoạt Java, vì vậy hành vi rõ ràng hơn một chút

được chỉ định. Hầu hết thời gian, nếu loại nội dung là văn bản / thuần túy, thì một Chuỗi sẽ là

trả lại. Nếu loại nội dung là nhiều phần, thì bất kể loại phụ, a

đối tượng javax.mail.Multipart được trả về. Nếu loại nội dung là một số loại khác

được nhận dạng bởi DataHandler bên dưới, khi đó một đối tượng Java thích hợp là

trả lại. Cuối cùng, nếu kiểu không được nhận dạng, thì InputStream sẽ được trả về.

Bạn có thể thay đổi đối tượng nào được trả về cho loại nội dung nào bằng cách cung cấp

DataHandler riêng. Điều này sẽ được cài đặt bằng phương thức setDataHandler ():

public void setDataHandler (javax.activation.DataHandler dh) ném

MessagingException, IllegalWriteException, IllegalStateException

Mặc dù phương thức này được khai báo để ném nhóm ngoại lệ thông thường, nhưng có lẽ nó là một

ít có khả năng thực sự làm như vậy, vì việc đặt DataHandler chỉ ảnh hưởng đến

Đối tượng tin nhắn chứ không phải là tin nhắn thực sự được lưu trữ trên máy chủ.

19.8.3.2 Viết nội dung của phần

Khi gửi tin nhắn, đương nhiên bạn phải cài đặt nội dung của tin nhắn. Kể từ email

tin nhắn là văn bản, cách đơn giản nhất là chỉ cung cấp văn bản của phần

với setText ():

public void setText (String text) ném MessagingException,

IllegalWriteException, IllegalStateException

Phương thức setText () đặt kiểu MIME thành văn bản / thuần túy. Các đối tượng khác có thể được thực hiện

vào nội dung, miễn là phần đó có DataHandler hiểu cách

chuyển đổi chúng thành văn bản được mã hóa. Điều này được thực hiện với phương thức setContent ():

public void setContent (Object o, String type) ném

MessagingException, IllegalWriteException, IllegalStateException

Một cách khác để viết nội dung của một phần là sử dụng OutputStream. Các

Phương thức writeTo () ghi nội dung của Part vào một OutputStream. Nếu

cần thiết, nó sẽ mã hóa nội dung bằng Base64, có thể in được trích dẫn hoặc một số

định dạng như được chỉ định bởi DataHandler:

public void writeTo (OutputStream out) ném IOException,

MessagingException

Trên thực tế, điều này không chỉ viết nội dung của Phần này; nó cũng viết các thuộc tính và

tiêu đề của phần. Ví dụ 19.4 đã sử dụng điều này để cung cấp một cách đơn giản để nhận được một

toàn bộ tin nhắn email trong một lần bị rơi. Tuy nhiên, nó thuận tiện nhất khi bạn muốn

để gửi toàn bộ tin nhắn đến máy chủ SMTP trong một lệnh gọi phương thức.

Cuối cùng, nhiều phần có thể được thêm vào một phần bằng cách gói chúng trong một đối tượng Multipart

và chuyển nó tới setContent ():

public void setContent (Multipart mp) ném MessagingException,

IllegalWriteException, IllegalStateException

Trong trường hợp này, toàn bộ thư thường sẽ có kiểu nội dung như

nhiều phần / hỗn hợp, nhiều phần / có dấu, hoặc nhiều phần / thay thế. Các phần riêng lẻ của

tất cả thư đều được đựng trong một phong bì, nhưng mỗi phần của thư đều có

loại nội dung, mã hóa nội dung và dữ liệu. Nhiều phần có thể được sử dụng để trình bày

các dạng khác nhau của cùng một tài liệu (ví dụ: HTML và thư văn bản thuần túy), một tài liệu

và siêu thông tin về tài liệu (ví dụ: một thông báo và thông báo MD5 của

tin nhắn), hoặc một số tài liệu khác nhau (ví dụ: một tin nhắn và một số tệp đính kèm).

Phần tiếp theo mở rộng về quá trình này.

19.9 Multipart Messages và File Attachments

Cách tất cả các loại tệp văn bản và tệp nhị phân khác nhau được mã hóa thành văn bản thô có thể

được chuyển qua cổng email 7-bit khá khéo léo và khá chi tiết.

May mắn thay, API JavaMail bảo vệ bạn khỏi những chi tiết đó, thú vị như chúng vốn có.

Để gửi một tin nhắn nhiều phần bằng cách sử dụng JavaMail API, tất cả những gì bạn phải làm là thêm

các bộ phận của một đối tượng MimeMultipart, sau đó chuyển đối tượng đó vào

phương thức setContent (). Để nhận một tin nhắn nhiều phần, bạn chỉ cần xử lý từng

các bộ phận riêng lẻ.

Hầu hết các phương pháp bạn sử dụng để xây dựng và giải cấu trúc các thông báo nhiều phần là trong

lớp trừu tượng javax.mail.Multipart:

public abstract class Multipart mở rộng đối tượng

Tuy nhiên, vì lớp này là trừu tượng, bạn thường bắt đầu với

Thay vào đó, đối tượng javax.mail.internet.MimeMultipart:

public class MimeMultipart mở rộng Multipart

Mỗi phần bạn thêm vào Multipart là một thể hiện của phần tóm tắt

Lớp javax.mail.BodyPart triển khai giao diện Phần của phần cuối cùng:

công khai lớp trừu tượng BodyPart mở rộng Đối tượng triển khai Phần

Trong email Internet, lớp con cụ thể của BodyPart mà bạn sẽ sử dụng là

javax.mail.internet.MimeBodyPart:

lớp công khai MimeBodyPart mở rộng BodyPart triển khai MimePart

Hầu hết các phương pháp bạn cần trong các lớp MimeBodyPart và BodyPart là

những cái bạn đã quen thuộc từ giao diện Phần, các phương pháp như

setContent () và setDataHandler (). Ngoài ra còn có ba phương pháp để đọc

nội dung của đối tượng Multipart:

public String getContentType( )

public int getCount( ) throws MessagingException

public BodyPart getBodyPart(int index)

throws IndexOutOfBoundsException, MessagingException

Phương thức getContentType () trả về kiểu MIME của toàn bộ Phần nhân,

thường là một cái gì đó như nhiều phần / hỗn hợp hoặc nhiều phần / thay thế. Đây không phải là

giống như kiểu MIME của các bộ phận riêng lẻ, giống như

văn bản / thuần túy hoặc hình ảnh / gif.

Phương thức getCount () trả về số phần trong Multipart này. Các

Phương thức getBodyPart () trả về một phần cụ thể. Các bộ phận được đánh số bắt đầu từ như

các thành phần của một mảng. Phần nhân không phân cấp. Multipart có thể không

chứa Multipart khác. Điều này làm cho việc viết chương trình để phân tích cú pháp các tin nhắn nhiều phần

khá đơn giản

. Ví dụ 19.12 rất giống với Ví dụ 19.11,

AllHeaderClient. Tuy nhiên, điều này thêm mã cần thiết để xử lý phần thân của

thông điệp. Nếu tin nhắn là một tin nhắn gồm một phần, thì nó chỉ được in trên

System.out. Tuy nhiên, nếu thư có nhiều phần, thì mỗi phần sẽ được xử lý

riêng biệt. Nếu phần không có tên tệp, không có sự sắp xếp

Part.ATTACHMENT và có loại MIME văn bản / thuần túy, sau đó nó được giả định là nội tuyến

và được in trên System.out. Nếu không, nó được giả định là một tệp đính kèm

và được lưu vào một tệp thích hợp. Nếu cần, tệp tĩnh File.createTempFile ()

phương thức được giới thiệu trong Java 1.2 được sử dụng để tạo tên hợp lý cho tệp.

Ví dụ 19.12.

Ứng dụng thư khách xử lý thư đa phần với tệp đính kèm

import javax.mail.\*;

import javax.mail.internet.\*;

import java.util.\*;

import java.io.\*;

public class AllPartsClient {

public static void main(String[] args) {

if (args.length == 0) {

System.err.println(

"Usage: java AllPartsClient

protocol://username@host:port/foldername");

return;

}

URLName server = new URLName(args[0]);

try {

Session session = Session.getDefaultInstance(new Properties( ),

new MailAuthenticator(server.getUsername( )));

// Connect to the server and open the folder

Folder folder = session.getFolder(server);

if (folder == null) {

System.out.println("Folder " + server.getFile( ) + " not

found.");

System.exit(1);

}

folder.open(Folder.READ\_ONLY);

// Get the messages from the server

Message[] messages = folder.getMessages( );

for (int i = 0; i < messages.length; i++) {

System.out.println("------------ Message " + (i+1)

+ " ------------");

// Print message headers

Enumeration headers = messages[i].getAllHeaders( );

while (headers.hasMoreElements( )) {

Header h = (Header) headers.nextElement( );

System.out.println(h.getName() + ": " + h.getValue( ));

}

System.out.println( );

// Enumerate parts

Object body = messages[i].getContent( );

if (body instanceof Multipart) {

processMultipart((Multipart) body);

}

else { // ordinary message

processPart(messages[i]);

}

System.out.println( );

}

// Close the connection

// but don't remove the messages from the server

folder.close(false);

}

catch (Exception e) {

e.printStackTrace( );

}

// Since we may have brought up a GUI to authenticate,

// we can't rely on returning from main( ) to exit

System.exit(0);

}

public static void processMultipart(Multipart mp)

throws MessagingException {

for (int i = 0; i < mp.getCount( ); i++) {

processPart(mp.getBodyPart(i));

}

}

public static void processPart(Part p) {

try {

String fileName = p.getFileName( );

String disposition = p.getDisposition( );

String contentType = p.getContentType( );

if (fileName == null && (disposition.equals(Part.ATTACHMENT)

|| !contentType.equals("text/plain"))) {

// pick a random file name. This requires Java 1.2 or later.

fileName = File.createTempFile("attachment",

".txt").getName( );

}

if (fileName == null) { // likely inline

p.writeTo(System.out);

}

else {

File f = new File(fileName);

// find a version that does not yet exist

for (int i = 1; f.exists( ); i++) {

String newName = fileName + " " + i;

f = new File(newName);

}

FileOutputStream out = new FileOutputStream(f);

// We can't just use p.writeTo( ) here because it doesn't

// decode the attachment. Instead we copy the input stream

// onto the output stream which does automatically decode

// Base-64, quoted printable, and a variety of other formats.

InputStream in = new

BufferedInputStream(p.getInputStream( ));

int b;

while ((b = in.read( )) != -1) out.write(b);

out.flush( );

out.close( );

in.close( );

}

}

catch (Exception e) {

System.err.println(e);

e.printStackTrace( );

}

}

}

Bạn cũng có thể lấy một phần từ thông báo nhiều phần bằng cách chuyển một Dòng đầu ra đến

phương thức writeTo () của part:

public abstract void writeTo (OutputStream out)

throws IOException, MessagingException

Tuy nhiên, điều này khác với cách tiếp cận được lấy trong Ví dụ 19.12 ở chỗ nó không

giải mã phần trước khi viết nó. Nó để lại bất kỳ mã hóa Base64 hoặc BinHex hoặc có thể trích dẫn nào mà người gửi chỉ áp dụng cho tệp đính kèm. Thay vào đó, nó chỉ đơn giản là

ghi dữ liệu thô.

Việc đính kèm tệp (hoặc tài liệu khác) vào thư bạn gửi sẽ phức tạp hơn. Đến

đính kèm tệp vào thư, trước tiên bạn phải bọc dữ liệu trong đối tượng BodyPart và

thêm nó vào Multipart bằng một trong hai phương thức addBodyPart ():

public void addBodyPart(BodyPart part)

throws IllegalWriteException, MessagingException

public void addBodyPart(BodyPart part, int index)

throws IllegalWriteException, MessagingException

Biến thể đầu tiên chỉ cần thêm phần vào cuối tin nhắn. Thư hai

biến thể thêm phần đã cho ở vị trí được chỉ định. Nếu vị trí lớn hơn

số phần trong tin nhắn, sau đó nó chỉ được thêm vào cuối. Nếu nó được thêm vào

ở đâu đó ở giữa, điều này có thể khiến vị trí của các bộ phận khác thay đổi. Nếu

không thể thay đổi thông báo, sau đó một IllegalWriteException sẽ được ném ra.

Phần khó là tạo đối tượng BodyPart. Để làm điều này, trước tiên bạn cần phải đoán

loại nội dung MIME hợp lý cho tệp (văn bản / thuần túy và ứng dụng / octet-stream là

các loại phổ biến nhất). Tiếp theo, bạn sẽ cần đọc tệp và chuyển đổi nó thành một số

lớp của đối tượng Java. Sau đó, bạn sẽ cài đặt một lớp javax.activation.DataHandler

biết cách chuyển đổi lớp dữ liệu của bạn theo kiểu MIME bạn đã chọn. Một lần

bạn đã làm tất cả những điều này, bạn có thể tạo một đối tượng MimeBodyPart mới và sử dụng các

các phương thức của giao diện Part để đặt các thuộc tính như tên tệp và nội dung

bố trí.

Ngoài ra còn có hai phương thức removeBodyPart () xóa một phần được chỉ định khỏi

, mặc dù chúng không được sử dụng phổ biến:

public boolean removeBodyPart(BodyPart part)

throws IllegalWriteException, MessagingException

public void removeBodyPart(int index)

throws IndexOutOfBoundsException, MessagingException

Nếu không thể thay đổi thông báo, thì một IllegalWriteException sẽ được ném ra. Nếu

chỉ mục được chỉ định không xác định một phần, khi đó IndexOutOfBoundsException là

ném. Nếu phần được chỉ định không xuất hiện trong tin nhắn, thì MessagingException

được ném.

19.10 Tin nhắn MIME

MIME được thiết kế chủ yếu cho email Internet và nó được tổ chức đặc biệt để

rằng nó sẽ tương thích ngược với các giao thức và phần mềm hiện có. Vì thế,

một email thông thường trên Internet trên thực tế là một tin nhắn MIME. Bê tông duy nhất

lớp con của Message trong JavaMail API là javax.mail.internet.MimeMessage:

public class MimeMessage mở rộng Message triển khai MimePart

Lớp này khai báo gần bảy mươi phương thức công khai và được bảo vệ. Tuy nhiên, với

ngoại lệ tự nhiên của các hàm tạo, hầu như tất cả các phương thức này đều ghi đè các phương thức từ

lớp cha Thông báo hoặc thực hiện các phương thức được khai báo bởi giao diện Phần. Các

chỉ có các phương pháp mới được khai báo trong giao diện MimePart,

giao diện con của Phần:

giao diện công cộng MimePart mở rộng Phần

Hầu hết các phương pháp này rất giống với các phương pháp trong Phần hoặc các phương pháp trong

Thông điệp. Tuy nhiên, chúng có các tính năng khó có thể tìm thấy trong không phải MIME

tin nhắn. Ví dụ: một phần MIME có thể có thông báo MD5, sẽ là

được mã hóa như một tiêu đề phụ bên trong phần. Do đó, giao diện MimePart tuyên bố và

lớp MimeMessage triển khai hai phương thức để thiết lập và nhận thông báo này:

public String getContentMD5( ) throws MessagingException

public void setContentMD5(String md5) throws MessagingException,

IllegalWriteException, IllegalStateException

Phương thức addHeaderLine () thêm một chuỗi văn bản vào tiêu đề của thư. nó là

tùy thuộc vào bạn để đảm bảo rằng chuỗi này thực sự có ý nghĩa trong tiêu đề:

ném public void addHeaderLine (String line)

MessagingException, IllegalWriteException, IllegalStateException

Phương thức getHeader () trả về giá trị của mọi tiêu đề trong thư với

Tên. Nếu có nhiều tiêu đề với tên này, thì chuỗi phân tách

giá trị của các tiêu đề khác nhau với chuỗi dấu phân cách được chỉ định:

public String getHeader (Tên chuỗi, Dấu phân cách chuỗi)

throws MessagingException

Phương thức getAllHeaderLines () trả về một java.util.Enumeration chứa

mọi tiêu đề trong thư. Enumeration chứa các đối tượng Chuỗi, mỗi đối tượng

đầu trang. Mỗi Chuỗi chứa đầy đủ tên và giá trị; ví dụ: "Chủ đề: Re:

Hỗ trợ Java 2 ". Nó không được chia thành một tên và giá trị riêng biệt:

public Enumeration getAllHeaderLines ()

throws MessagingException

Phương thức getMatchingHeaderLines () trả về tất cả các dòng tiêu đề có tên là

được đưa ra trong mảng đối số tên. Phương thức getNonMatchingHeaderLines ()

thực hiện điều tương tự ngoại trừ việc nó trả về tất cả các dòng tiêu đề đó với tên không phải

được đề cập trong đối số tên:

public Enumeration getMatchingHeaderLines(String[] names)

throws MessagingException

public Enumeration getNonMatchingHeaderLines(String[] names)

throws MessagingException

Phương thức getEncoding () trả về mã hóa của phần MIME này dưới dạng một Chuỗi như

được cung cấp bởi tiêu đề Content-transfer-encoding:. Mã hóa điển hình cho văn bản thuần túy

email là 7-bit hoặc có lẽ 8-bit hoặc có thể in được trích dẫn. Mã hóa điển hình cho một tệp

tệp đính kèm là Base64:

public String getEncoding () throws MessagingException

Phương thức getContentID () trả về một chuỗi xác định duy nhất phần này là

được cung cấp bởi trường Content-ID: của phần. Một ID điển hình trông giống như

<Pine.LNX.4.10.9912290930220.8058@akbar.nevex.com>. Nó trả về null nếu

phần không có ID nội dung:

public String getContentID () ném MessagingException

IllegalWriteException, IllegalStateException

Phương thức getContentLanguage () trả về giá trị của ngôn ngữ Content:

đầu trang. Đây là danh sách được phân tách bằng dấu phẩy gồm hai (hoặc nhiều) chữ cái viết tắt cho

theo định nghĩa của RFC 1766. Ví dụ: tiếng Anh là "en" và tiếng Pháp là "fr". Nó

trả về null nếu phần không có tiêu đề Content-language:.

public String [] getContentLanguage () throws MessagingException

Ngoài ra còn có một phương thức setContentLanguage () mà bạn có thể sử dụng khi gửi

thông điệp:

ném public void setContentLanguage (String [] ngôn ngữ)

MessagingException, IllegalWriteException, IllegalStateException

Cuối cùng, hai phương thức setText () thiết lập nội dung của phần có kiểu MIME

văn bản / thuần túy. Phương thức setText () thứ hai cũng cho phép bạn chỉ định bộ ký tự; cho

ví dụ, us-ascii hoặc ISO 8859-1:

public void setText(String text) throws MessagingException

public void setText(String text, String charset)

throws MessagingException

19.11 Thư mục

Cho đến nay, chúng tôi đã làm việc chủ yếu với thư mục INBOX. Đây là thư mục mặc định, nơi

hầu hết các thư đều nằm cho đến khi người dùng lọc hoặc lưu vào một số thư mục khác. Một chút

hệ thống, nó thực sự có thể nằm trong một tệp được gọi là INBOX. Trên các hệ thống khác, nó có thể

được gọi là một cái gì đó khác biệt. Tuy nhiên, bạn luôn có thể truy cập nó từ JavaMail

API sử dụng tên INBOX.

Hầu hết các chương trình thư đều cho phép bạn sắp xếp thư của mình vào các thư mục khác nhau.

Các thư mục này có thứ bậc; nghĩa là, một thư mục có thể chứa một thư mục khác. Trong

đặc biệt, trong giao thức IMAP, các máy chủ lưu trữ các thông báo trong các thư mục khác nhau, từ

khách hàng nào truy xuất và thao tác các thông báo khi cần thiết. Máy chủ POP, bởi

ngược lại, thường gửi tất cả tin nhắn cho người dùng khi người dùng kết nối, sau đó dựa vào

trên máy khách để lưu trữ và quản lý chúng. Ưu điểm chính của phương pháp IMAP

hơn POP là nó cho phép người dùng dễ dàng truy cập vào toàn bộ kho lưu trữ email của mình từ nhiều

máy khách.

API JavaMail đại diện cho các thư mục giống IMAP dưới dạng các bản sao của Thư mục trừu tượng

lớp học:

public abstract class Thư mục mở rộng đối tượng

Lớp này khai báo các phương thức để yêu cầu các thư mục được đặt tên từ máy chủ, xóa

thư từ các thư mục, tìm kiếm thư cụ thể trong các thư mục, liệt kê

thư trong một thư mục, v.v. Hầu hết các phương thức này được khai báo là trừu tượng. Khi nào

bạn yêu cầu một phiên, một cửa hàng hoặc một thư mục cung cấp cho bạn một trong những thư mục mà nó chứa, nó sẽ

cung cấp cho bạn một ví dụ về lớp con cụ thể phù hợp với giao thức đang sử dụng: IMAP,

POP, mbox hoặc bất cứ thứ gì. Việc triển khai tham chiếu của JavaMail API biết

cách thực hiện các thao tác này chỉ dành cho máy chủ IMAP. Tuy nhiên, một số bên thứ ba

triển khai cung cấp các hoạt động này trong các thư mục hộp thư cục bộ được lưu trữ trên

hệ thống tệp của khách hàng.

19.11.1 Mở thư mục

Bạn không thể tạo thư mục trực tiếp. Hàm tạo duy nhất được bảo vệ:

Thư mục được bảo vệ (Cửa hàng lưu trữ)

Thay vào đó, bạn nhận được một Thư mục từ một Phiên, một Cửa hàng hoặc một Thư mục khác như sau:

Thư mục outbox = containerner.getFolder ("đã gửi-mail");

Thực tế có ba phương thức getFolder (), mỗi phương thức trong Session, Store và

Các lớp thư mục. Tất cả chúng đều có cùng một chữ ký và hoạt động tương tự:

ném thư mục tóm tắt công khai getFolder (Tên chuỗi)

MessagingException

Các phương thức này có chung một kiểu riêng khó chịu với lớp Tệp. Nhận một thư mục

đối tượng không ngụ ý rằng Thư mục được đặt tên thực sự tồn tại trên máy chủ. Nói

liệu thư mục có thực sự hiện diện hay không, bạn phải kiểm tra nó bằng phương thức being ():

boolean công khai tồn tại () throws MessagingException

Khi bạn lần đầu tiên nhận được một thư mục, nó sẽ bị đóng. Trước khi bạn có thể đọc các tin nhắn trong nó,

bạn phải mở thư mục bằng phương thức open ():

public abstract void open (int mode)

throws FolderNotFoundException, MessagingException

Đối số mode phải là một trong hai hằng số được đặt tên là Folder.READ\_ONLY hoặc

Thư mục.READ\_WRITE. Một số nhưng không phải tất cả các triển khai đều cho phép bạn mở nhiều

kết nối chỉ đọc đến một thư mục thực bằng cách sử dụng nhiều đối tượng Thư mục. Tuy nhiên, tất cả

triển khai cho phép nhiều nhất một đối tượng Thư mục có quyền ghi vào một thư mục tại

một lần.

Một số thao tác được thảo luận trong phần này, chẳng hạn như tìm kiếm hoặc truy xuất tin nhắn

từ một thư mục chỉ có thể được thực hiện trên một thư mục đang mở. Những người khác chẳng hạn như xóa hoặc

thay đổi tên của một thư mục chỉ có thể được thực hiện trên một thư mục đóng. Các

Phương thức isOpen () trả về true nếu thư mục đang mở, false nếu nó bị đóng:

public trừu tượng boolean isOpen ()

Nói chung, cố gắng làm điều gì đó với một thư mục đóng yêu cầu thư mục đó phải

mở hoặc ngược lại sẽ ném ra một java.lang.IllegalStateException. Đây là một

ngoại lệ thời gian chạy, vì vậy nó không cần phải được bắt hoặc khai báo rõ ràng.

Khi bạn hoàn thành một thư mục, bạn nên đóng nó bằng phương thức close ():

public abstract void close (boolean expunge)

throws FolderNotFoundException, MessagingException

Nếu đối số xóa là đúng, thì mọi thư đã xóa trong thư mục sẽ bị xóa

từ tệp thực tế trên máy chủ. Nếu không, chúng chỉ được đánh dấu là đã xóa, nhưng

tin nhắn vẫn có thể được hoàn tác.

19.11.2 Thông tin thư mục cơ bản

Lớp Thư mục có tám phương thức trả về thông tin cơ bản về một thư mục:

public abstract String getName( )

public abstract String getFullName( )

public URLName getURLName( ) throws MessagingException

public abstract Folder getParent( ) throws MessagingException

public abstract int getType( ) throws MessagingException

public int getMode( ) throws IllegalStateException

public Store getStore( )

public abstract char getSeparator( )

throws FolderNotFoundException, MessagingException

Phương thức getName () trả về tên của thư mục, chẳng hạn như "Reader Mail",

trong khi phương thức getFullName () trả về tên phân cấp hoàn chỉnh từ

gốc, chẳng hạn như "books / JNP2E / Reader Mail". Phương thức getURLName () bao gồm

máy chủ; ví dụ: "imap: //elharo@mail.metalab.unc.edu/books/JNP2E/Reader

Thư ". Trong ví dụ này, ký tự gạch chéo được sử dụng làm dấu phân cách giữa các phần của

thư mục. Dấu phân cách có thể thay đổi tùy theo việc triển khai, nhưng

Phương thức getSeparator () luôn cho bạn biết nó là gì.

Phương thức getParent () trả về tên của thư mục chứa thư mục này;

ví dụ: "JNP2E" cho ví dụ Reader Mail trước đó.

Phương thức getType () trả về một int cho biết liệu thư mục có thể chứa

tin nhắn và / hoặc các thư mục khác. Nếu nó có thể chứa tin nhắn nhưng không chứa thư mục, thì

getType () trả về Thư mục hằng số được đặt tên.HOLDS\_MESSAGES. Nếu nó có thể chứa

thư mục chứ không phải thư, sau đó getType () trả về hằng số được đặt tên

Thư mục.HOLDS\_FOLDERS. Nếu nó có thể chứa cả thư mục và tin nhắn, thì getType ()

trả về Thư mục liên hợp bitwise.HOLDS\_FOLDERS & Thư mục.HOLDS \_MESSAGES.

Phương thức getMode () cho bạn biết liệu một thư mục có cho phép ghi hay không. Nó trả về một trong số

hai hằng số được đặt tên là Folder.READ\_ONLY hoặc Folder.READ\_WRITE hoặc -1 nếu

chế độ không xác định. Cuối cùng, phương thức getStore () trả về đối tượng Store từ

mà thư mục này đã được truy xuất.

19.11.3 Quản lý thư mục

Phương thức create () tạo một thư mục mới trong Cửa hàng của thư mục này:

public abstract boolean create (int type) throws MessagingException

Loại thư mục phải là một trong các hằng số được đặt tên

Thư mục.HOLDS\_MESSAGES hoặc Thư mục.HOLDS\_FOLDERS tùy thuộc vào việc nó sẽ

giữ các thư mục hoặc tin nhắn khác. Nó trả về true nếu tạo thành công, false nếu nó

không.

Phương thức delete () sẽ xóa thư mục này. Nó có thể làm điều đó chỉ khi thư mục được đóng.

Nếu không, IllegalStateException sẽ được ném ra.

ném xóa boolean trừu tượng công khai (boolean recurse) throws

IllegalStateException, FolderNotFoundException, MessagingException

Nếu có thư trong thư mục này, thì chúng sẽ bị xóa cùng với thư mục. Nếu

thư mục chứa các thư mục con, sau đó các thư mục con sẽ bị xóa nếu đối số đệ quy là

thật. Nếu đối số đệ quy không đúng, thì thư mục sẽ chỉ bị xóa nếu có

không chứa bất kỳ thư mục con nào. Nếu nó có chứa các thư mục con, thì việc xóa không thành công. Nếu

thư mục có chứa các thư mục con và cũng chứa các thư, sau đó nó phụ thuộc vào việc triển khai cho dù các thư có bị xóa hay không mặc dù chính thư mục đó

không phải. Nếu xóa thành công, thì phương thức trả về true; nếu không, nó trả về false.

Phương thức renameTo () thay đổi tên của thư mục này. Một thư mục phải được đóng lại

được đổi tên. Nếu không, IllegalStateException sẽ được ném ra. Phương thức này trả về

true nếu thư mục được đổi tên thành công, false nếu không phải:

public trừu tượng boolean renameTo (Folder f) throws

IllegalStateException, FolderNotFoundException, MessagingException

19.11.4 Quản lý tin nhắn trong thư mục

Đôi khi, bạn có thể thấy cần phải đặt một tin nhắn vào một thư mục. Chỉ có một

phương thức để thực hiện việc này, appendMessages ():

public abstract void appendMessages (Message [] tin nhắn)

throws FolderNotFoundException, MessagingException

Như tên của nó, các tin nhắn được đặt ở cuối thư mục này.

Phương thức copyMessages () sao chép thư vào thư mục này từ một

thư mục được cung cấp như một đối số:

public void copyMessages(Message[] messages, Folder destination)

throws

IllegalStateException, FolderNotFoundException, MessagingException

Các thư đã sao chép được nối vào thư mục đích. Chúng không bị xóa

từ thư mục nguồn. Để di chuyển một tin nhắn, bạn phải sao chép nó từ nguồn sang

đích, sau đó xóa nó khỏi thư mục nguồn, rồi cuối cùng xóa nguồn

thư mục.

Để xóa thư khỏi một thư mục, bạn đặt cờ Flags.Flag.DELETED của thư đó thành true. Đến

xóa vật lý các tin nhắn đã xóa khỏi một thư mục, bạn phải xóa nó bằng cách sử dụng

phương thức expunge ():

public abstract Message [] expunge () throws MessagingException,

IllegalStateException, FolderNotFoundException

Sau khi một tin nhắn đã bị xóa, vẫn có thể có các đối tượng Message tham chiếu đến nó.

Trong trường hợp này, hầu hết mọi phương thức đều gọi một đối tượng như vậy, ngoại trừ isExpunged () và

getMessageNumber (), sẽ đưa ra một ngoại lệ.

19.11.5 Đăng ký

Một số triển khai (mặc dù không phải là triển khai IMAP mặc định) cho phép bạn

đăng ký vào các thư mục cụ thể. Điều này sẽ thích hợp nhất cho một nhà cung cấp NNTP,

nơi một máy chủ điển hình cung cấp hàng nghìn nhóm tin, nhưng người dùng thông thường sẽ muốn

để lấy tối đa vài chục tin nhắn trong số này. Mỗi nhóm tin sẽ là

được biểu diễn dưới dạng một đối tượng Thư mục. Đăng ký vào Thư mục của nhóm tin tức cho biết

mà người dùng muốn truy xuất thông báo từ nhóm tin đó:

public boolean isSubscribed( )

public void setSubscribed(boolean subscribe)

throws FolderNotFoundException, MethodNotSupportedException,

MessagingException

Nếu nhà cung cấp không hỗ trợ đăng ký, thì setSubscribe () ném

MethodNotSupportedException và isSubscribe () trả về false.

19.11.6 Liệt kê nội dung của một thư mục

Các thư mục được phân cấp. Nghĩa là, một thư mục có thể chứa các thư mục khác. Có bốn

phương pháp liệt kê các thư mục mà một thư mục chứa. Đó là:

public Folder[] list( )

throws FolderNotFoundException, MessagingException

public Folder[] listSubscribed( )

throws FolderNotFoundException, MessagingException

public abstract Folder[] list(String pattern)

throws FolderNotFoundException, MessagingException

public Folder[] listSubscribed(String pattern)

throws FolderNotFoundException, MessagingException

Phương thức đầu tiên trả về một mảng chứa tất cả các thư mục mà thư mục này chứa.

Phương thức thứ hai trả về một mảng chứa tất cả các thư mục đã đăng ký mà

thư mục chứa.

Phương pháp thứ ba và thứ tư lặp lại hai phương pháp đầu tiên này ngoại trừ việc chúng cho phép bạn

chỉ định một mẫu. Chỉ những thư mục có tên đầy đủ phù hợp với mẫu sẽ nằm trong

mảng đã trả về. Mẫu là một chuỗi cung cấp tên của các thư mục phù hợp.

Tuy nhiên, chuỗi có thể chứa ký tự%, là ký tự đại diện khớp với bất kỳ

chuỗi ký tự không bao gồm dấu phân tách thứ bậc và dấu \* phù hợp với

bất kỳ chuỗi ký tự nào bao gồm cả dấu phân tách thứ bậc.

19.11.7 Kiểm tra thư

Phương thức getMessageCount () trả về số lượng thư trong thư mục này:

public abstract int getMessageCount ()

throws FolderNotFoundException, MessagingException

Phương thức này có thể được gọi trên một thư mục đang mở hoặc đã đóng. Tuy nhiên, trong trường hợp của một

thư mục đã đóng, phương thức này có thể (hoặc không) trả về -1 để chỉ ra rằng chính xác

số lượng tin nhắn không dễ dàng có sẵn.

Phương thức hasNewMessages () trả về true nếu thông báo mới đã được thêm vào

thư mục kể từ lần cuối cùng được mở (không phải kể từ lần cuối cùng bạn kiểm tra!):

public trừu tượng boolean hasNewMessages ()

throws FolderNotFoundException, MessagingException

Phương thức getNewMessageCount () sử dụng một cách tiếp cận hơi khác cho

xác định có bao nhiêu tin nhắn mới. Nó kiểm tra số lượng tin nhắn trong

thư mục có cờ RECENT được đặt:

public int getNewMessageCount ()

throws FolderNotFoundException, MessagingException

Không giống như hasNewMessages (), getNewMessageCount () có thể được gọi trên một

mở hoặc một thư mục đã đóng. Tuy nhiên, trong trường hợp một thư mục đóng,

getNewMessageCount () có thể trả về -1 để cho biết rằng câu trả lời thực cũng sẽ là

đắt tiền để có được.

Phương thức getUnreadMessageCount () tương tự nhưng trả về số

thư trong thư mục có cờ SEEN không được đặt:

public int getUnreadMessageCount ()

throws FolderNotFoundException, MessagingException

Giống như getNewMessageCount (), getUnreadMessageCount () có thể được gọi trên

hoặc một thư mục đang mở hoặc đã đóng. Tuy nhiên, trong trường hợp một thư mục đóng, nó có thể

trả về -1 để chỉ ra rằng câu trả lời thực sẽ quá đắt để có được.

19.11.8 Nhận tin nhắn từ thư mục

Lớp Thư mục cung cấp bốn phương pháp để lấy thư từ các thư mục đang mở.

Đó là:

public abstract Message getMessage(int messageNumber) throws

IndexOutOfBoundsException, FolderNotFoundException,

IllegalStateException, MessagingException

public Message[] getMessages( ) throws FolderNotFoundException,

IllegalStateException, MessagingException

public Message[] getMessages(int start, int end) throws

IndexOutOfBoundsException, FolderNotFoundException,

IllegalStateException, MessagingException

public Message[] getMessages(int[] messageNumbers) throws

IndexOutOfBoundsException, FolderNotFoundException,

IllegalStateException, MessagingException

Phương thức getMessage () trả về thông báo thứ n trong thư mục. Tin nhắn đầu tiên

trong thư mục là số 1 (không phải 0). Số tin nhắn có thể thay đổi khi có tin nhắn

bị xóa khỏi thư mục. Một IndexOutOfBoundsException được ném ra nếu bạn yêu cầu

thư n và có n-1 thư trở xuống trong thư mục.

Phương thức getMessages () đầu tiên trả về một mảng các đối tượng Message đại diện cho

tất cả các thư trong thư mục này. Phương thức getMessages () thứ hai trả về một mảng

đối tượng Message từ thư mục bắt đầu bằng bắt đầu và kết thúc bằng cuối,

bao gồm. Phương thức getMessages () thứ ba trả về một mảng chỉ chứa những

thông báo được xác định cụ thể bằng số trong mảng messageNumbers.

Tất cả bốn phương pháp này chỉ tạo các đối tượng Message và điền vào

số trường trong các đối tượng đó. Văn bản thực tế và nội dung khác của tin nhắn

sẽ chỉ được tìm nạp từ máy chủ khi các phương thức của Thông báo sử dụng những thứ đó

được gọi. Ví dụ: điều này có nghĩa là bạn không thể nhận được tất cả các tin nhắn từ

máy chủ, sau đó ngắt kết nối PPP của bạn và làm việc với chúng ngoại tuyến. Có,

tuy nhiên, một phương thức fetch (), sẽ điền vào một số phần nhất định của các đối tượng Message với

dữ liệu thực tế từ máy chủ:

tìm nạp void công khai (Thông báo [] thông báo, FetchProfile fp)

throws IllegalStateException, MessagingException

Đối số thông báo là một mảng chứa các đối tượng Thông báo được tìm nạp trước.

Đối số FetchProfile chỉ định tiêu đề nào trong thư cần tìm nạp trước.

Tuy nhiên, đây vẫn chỉ là một gợi ý. Việc triển khai có thể miễn phí để bỏ qua yêu cầu này

và chỉ tìm nạp nội dung tin nhắn khi nó thực sự cần thiết.

Bạn có thể yêu cầu tìm nạp trước các tiêu đề riêng lẻ như Chủ đề: theo tên. Bạn có thể

cũng yêu cầu tìm nạp trước ba khối thông tin được xác định trước: phong bì

(về cơ bản là chủ đề và người nhận của thư), cờ của thư hoặc

thông tin nội dung của tin nhắn. Ba nhóm bạn có thể yêu cầu được đưa ra là

các đối tượng FetchProfile.Item không đổi. Chúng là FetchProfile.Item.ENVELOPE,

FetchProfile.Item.FLAGS và FetchProfile.Item.CONTENT\_INFO.

Lớp FetchProfile có một phương thức khởi tạo noargs đơn giản cũng như các phương thức cho

xây dựng một hồ sơ mới, để thêm các mục và tiêu đề cụ thể vào hồ sơ, và

để kiểm tra xem một mục cụ thể có phải là một phần của một cấu hình cụ thể hay không. Đó là:

public FetchProfile( )

public void add(FetchProfile.Item item)

public void add(String headerName)

public boolean contains(FetchProfile.Item item)

public boolean contains(String headerName)

public FetchProfile.Item[] getItems( )

public String[] getHeaderNames( )

Ví dụ: giả sử bạn chỉ muốn tải xuống các chủ đề, địa chỉ Tới:

và thông tin nội dung của một khối tin nhắn. Sau đó, bạn sẽ tìm nạp chúng như

điều này:

Message[] messages = folder.getMessages( );

FetchProfile fp = new FetchProfile( );

fp.add(FetchProfile.Item.CONTENT\_INFO);

fp.add("Subject");

fp.add("To");

19.11.9 Tìm kiếm thư mục

Nếu máy chủ hỗ trợ tìm kiếm (như nhiều máy chủ IMAP và hầu hết các máy chủ POP

không), thật dễ dàng để tìm kiếm một thư mục cho các thư đáp ứng các tiêu chí nhất định. Các

tiêu chí được mã hóa trong các đối tượng SearchTerm:

lớp trừu tượng công khai SearchTerm mở rộng đối tượng

Lớp SearchTerm là trừu tượng, nhưng API JavaMail cung cấp nhiều lớp con cho

thực hiện các tìm kiếm thông thường:

public abstract class AddressTerm extends SearchTerm

public abstract class FlagTerm extends SearchTerm

public abstract class StringTerm extends SearchTerm

public final class FromTerm extends AddressTerm

public final class FromStringTerm extends AddressStringTerm

public final class ReceipientTerm extends AddressTerm

public final class AddressStringTerm extends StringTerm

public final class BodyTerm extends StringTerm

public final class HeaderTerm extends StringTerm

public final class MessageIDTerm extends StringTerm

public final class SubjectTerm extends StringTerm

public abstract class DateTerm extends ComparisonTerm

public final class ReceivedDateTerm extends DateTerm

public final class SentDateTerm extends DateTerm

Nó cũng cung cấp một số lớp để kết hợp các tìm kiếm:

public Message[] search(SearchTerm term) throws SearchException,

FolderNotFoundException, IllegalStateException, MessagingException

public Message[] search(SearchTerm term, Message[] messages)

throws SearchException, FolderNotFoundException,

IllegalStateException, MessagingException

SearchException chỉ ra rằng cụm từ tìm kiếm phức tạp hơn

thực hiện có thể xử lý. Ví dụ: cụm từ tìm kiếm này tìm kiếm tất cả các thư từ

billg@microsoft.com:

Address billg = new InternetAddress("billg@microsoft.com");

SearchTerm term = new FromTerm(billg);

Cụm từ tìm kiếm này tìm kiếm tất cả các thư từ billg@microsoft.com sau năm 1999:

Address billg = new InternetAddress("billg@microsoft.com");

SearchTerm term1 = new FromTerm(billg);

Date millennium = Calendar.getInstance().set(2000, 0, 1).getTime( );

SearchTerm term2 = new SentDateTerm(ComparisonTerm.GE, millennium);

SearchTerm term = new AndTerm(term1, term2);

Ví dụ 19.13 là một biến thể đơn giản của chương trình MailClient của Ví dụ 19.7. Nó

cho phép người dùng liệt kê các địa chỉ email trên dòng lệnh sau URL ban đầu như

điều này:

% java SearchClient imap: //elharo@mail.metalab.unc.edu/INBOX

willis@nvx.com billg@microsoft.com

Chỉ những tin nhắn từ những người dùng được chỉ định sẽ được trả lại. Tuy nhiên, nếu không có email

địa chỉ được đưa ra, sau đó tất cả các tin nhắn sẽ được trả lại.

Ví dụ 19.13. Một ứng dụng thư điện tử tìm kiếm theo địa chỉ:

import javax.mail.\*;

import javax.mail.search.\*;

import javax.mail.internet.\*;

import java.util.\*;

import java.io.\*;

public class SearchClient {

public static void main(String[] args) {

if (args.length == 0) {

System.err.println(

"Usage: java SearchClient

protocol://username@host/foldername");

return;

}

URLName server = new URLName(args[0]);

try {

Session session = Session.getDefaultInstance(new Properties( ),

new MailAuthenticator(server.getUsername( )));

// Connect to the server and open the folder

Folder folder = session.getFolder(server);

if (folder == null) {

System.out.println("Folder " + server.getFile( ) + " not

found.");

System.exit(1);

}

folder.open(Folder.READ\_ONLY);

SearchTerm term = null;

if (args.length > 1) {

SearchTerm[] terms = new SearchTerm[args.length-1];

for (int i = 1; i < args.length; i++) {

Address a = new InternetAddress(args[i]);

terms[i-1] = new FromTerm(new InternetAddress(args[i]));

}

if (terms.length > 1) term = new OrTerm(terms);

else term = terms[0];

}

// Get the messages from the server

Message[] messages;

if (term == null) {

messages = folder.getMessages( );

}

else {

messages = folder.search(term);

}

for (int i = 0; i < messages.length; i++) {

System.out.println("------------ Message " + (i+1)

+ " ------------");

// Print message headers

Enumeration headers = messages[i].getAllHeaders( );

while (headers.hasMoreElements( )) {

Header h = (Header) headers.nextElement( );

System.out.println(h.getName() + ": " + h.getValue( ));

}

System.out.println( );

// Enumerate parts

Object body = messages[i].getContent( );

if (body instanceof Multipart) {

processMultipart((Multipart) body);

}

else { // ordinary message

processPart(messages[i]);

}

System.out.println( );

}

// Close the connection

// but don't remove the messages from the server

folder.close(false);

}

catch (Exception e) {

e.printStackTrace( );

}

// Since we may have brought up a GUI to authenticate,

// we can't rely on returning from main( ) to exit

System.exit(0);

}

public static void processMultipart(Multipart mp)

throws MessagingException {

for (int i = 0; i < mp.getCount( ); i++) {

processPart(mp.getBodyPart(i));

}

}

public static void processPart(Part p) {

try {

// I'd prefer to test the Content-Disposition header here.

// However, too many common email clients don't use it.

String fileName = p.getFileName( );

if (fileName == null) { // likely inline

p.writeTo(System.out);

}

else if (fileName != null) {

File f = new File(fileName);

// find a version that does not yet exist

for (int i = 1; f.exists( ); i++) {

String newName = fileName + " " + i;

f = new File(newName);

}

FileOutputStream out = new FileOutputStream(f);

// We can't just use p.writeTo( ) here because it doesn't

// decode the attachment. Instead we copy the input stream

// onto the output stream which does automatically decode

// Base-64, quoted printable, and a variety of other formats.

InputStream in = new

BufferedInputStream(p.getInputStream( ));

int b;

while ((b = in.read( )) != -1) out.write(b);

out.flush( );

out.close( );

in.close( );

}

}

catch (Exception e) {

System.err.println(e);

e.printStackTrace( );

}

}

}

19.11.10 Cờ

Đôi khi, rất hữu ích khi có thể thay đổi cờ cho toàn bộ nhóm thư tại

Một lần. Lớp Thư mục có hai phương thức để thực hiện việc này:

public void setFlags(Message[] messages, Flags flag, boolean value)

throws IllegalStateException, MessagingException

public void setFlags(int start, int end, Flags flag, boolean value)

throws IllegalStateException, MessagingException

public void setFlags(int[] messageNumbers, Flags flag, boolean value)

throws IndexOutOfBoundsException, IllegalStateException,

MessagingException

Cuối cùng, đây chỉ là những tiện ích. Bạn không thể làm gì với những thứ này

bạn không thể làm bằng cách đặt cờ trên từng thư riêng lẻ với setFlags ()

phương thức của lớp Message. Trên thực tế, việc triển khai mặc định chỉ đơn giản gọi rằng

trên mỗi thư trong khối thư được chỉ định.

Lớp Thư mục cũng có một phương thức getPermanentFlags () để trả về các cờ

thư mục này sẽ cung cấp cho tất cả các thư. Điều này bao gồm tất cả các cờ ngoại trừ các cờ do người dùng xác định, chỉ được áp dụng cho các thư cụ thể mà người dùng đã gắn cờ.

Ví dụ: không phải tất cả các triển khai thư mục đều có thể theo dõi xem các thư đã được

đã trả lời:

Công khai trừu tượng Cờ getPermanentFlags ()

19.11.11 Xử lý sự kiện

Nhiều chương trình email như Eudora và Pine có thể được cấu hình định kỳ

kiểm tra email đến trong nền. Một cách để cấu trúc một chương trình email

giống như một loạt các phản ứng đối với các sự kiện không thể đoán trước. Điều này giống như lập trình cho

giao diện người dùng đồ họa và thực sự là API JavaMail sử dụng các mẫu cơ bản giống nhau

để xử lý các sự kiện thư mà AWT và Swing sử dụng để xử lý các sự kiện GUI.

API JavaMail xác định sáu loại sự kiện thư khác nhau, tất cả đều nằm trong

gói javax.mail.event. Đây là tất cả các lớp con của MailEvent:

public abstract class MailEvent mở rộng EventObject

Sáu loại sự kiện thư cụ thể, bốn loại đầu tiên liên quan đến thư mục, là:

ConnectionEvent

Một Thư mục (hoặc Lưu trữ hoặc Vận chuyển) đã được mở, đóng hoặc ngắt kết nối.

FolderEvent

Một Thư mục đã được tạo, xóa hoặc đổi tên.

MessageChangedEvent

Phong bì hoặc cờ của thư đã thay đổi.

MessageCountEvent

Một thư đã được thêm vào hoặc xóa khỏi một Thư mục.

StoreEvent

Thông báo hoặc cảnh báo từ Cửa hàng.

TransportEvent

Một thông báo từ một Bộ phận vận chuyển rằng một tin nhắn đã được gửi, một phần

đã giao, hoặc không giao được.

Có các giao diện người nghe cho từng loại sự kiện trong số sáu loại sự kiện sau:

public interface ConnectionListener extends EventListener

public interface FolderListener extends EventListener

public interface MessageChangedListener extends EventListener

public interface MessageCountListener extends EventListener

public interface StoreListener extends EventListener

public interface TransportListener extends EventListener

Mỗi giao diện này khai báo một hoặc nhiều phương thức phải được cung cấp bởi

thực hiện các lớp. Ví dụ, lớp ConnectionListener khai báo những

ba phương pháp:

public void opened(ConnectionEvent e)

public void disconnected(ConnectionEvent e)

public void closed(ConnectionEvent e)

Giao diện FolderListener khai báo ba phương thức sau:

public void folderCreated(FolderEvent e)

public void folderDeleted(FolderEvent e)

public void folderRenamed(FolderEvent e)

Bốn trong số các sự kiện này có thể được kích hoạt bởi các thư mục. Do đó, có 14

addXXXListener (), removeXXXListener () và thông báoXXXListener ()

các phương thức trong lớp Thư mục:

public void addConnectionListener(ConnectionListener l)

public void removeConnectionListener(ConnectionListener l)

protected void notifyConnectionListeners(int type)

public void addFolderListener(FolderListener l)

public void removeFolderListener(FolderListener l)

protected void notifyFolderListeners(int type)

protected void notifyFolderRenamedListeners(Folder folder)

public void addMessageCountListener(MessageCountListener l)

public void removeMessageCountListener(MessageCountListener l)

protected void notifyMessageAddedListeners(Message[] messages)

protected void notifyMessageRemovedListeners(boolean removed,

Message[] messages)

public void addMessageChangedListener(MessageChangedListener l)

public void removeMessageChangedListener(MessageChangedListener l)

protected void notifyMessageChangedListeners(int type, Message

message)

Các phương thức addXXXListener () được gọi để thêm việc triển khai

giao diện cụ thể cho danh sách người nghe. Các phương thức removeXXXListener () là

được gọi để xóa một triển khai khỏi danh sách đó. Thông báo cho XXXListener ()

các phương pháp không được sử dụng trực tiếp. Thay vào đó, chúng được sử dụng bởi các phiên bản của Thư mục và

lớp con để thông báo cho người nghe đã đăng ký về các sự kiện cụ thể. Tất cả điều này hoạt động chính xác

như trong AWT và Swing, chỉ với các sự kiện khác nhau.

19.11.12 Phương pháp Tiện ích

Cuối cùng, vì lợi ích của sự hoàn chỉnh, tôi sẽ lưu ý rằng lớp Thư mục ghi đè hai phương thức

từ java.lang.Object, finalize () và toString ():

protected void finalize( )

throws Throwable public String toString( )

Cả hai điều này đều không đặc biệt quan trọng đối với lập trình viên.

Colophon

Giao diện của chúng tôi là kết quả của nhận xét của người đọc, thử nghiệm của chính chúng tôi và phản hồi

từ các kênh phân phối. Các bản cover khác biệt bổ sung cho cách tiếp cận đặc biệt của chúng tôi đối với

chủ đề kỹ thuật, thở tính cách và cuộc sống vào các chủ đề có thể khô khan.

Con vật trên trang bìa của Lập trình mạng Java ™, Ấn bản thứ hai, là một phương Bắc

Rái cá sông Mỹ (Lutra canadensis). Những loài ăn thịt nhỏ này được tìm thấy ở tất cả các loài chính

đường thủy của Hoa Kỳ và Canada, ở hầu hết mọi môi trường sống ngoại trừ lãnh nguyên

và các vùng khô nóng ở Tây Nam Hoa Kỳ. Chúng nặng khoảng 20 pound và

dài khoảng 2,5 feet và con cái có xu hướng nhỏ hơn khoảng một phần ba

hơn nam giới. Chế độ ăn uống của họ chủ yếu bao gồm các động vật sống dưới nước như cá và ếch, nhưng kể từ khi

chúng dành khoảng hai phần ba thời gian trên cạn, thỉnh thoảng chúng ăn chim hoặc

loài gặm nhấm. Hai lớp lông - lớp lông ngoài thô và lớp lông bên trong dày và dày - bảo vệ

rái cá sông khỏi cái lạnh, và trên thực tế, chúng có vẻ thích chơi trong băng tuyết. A

Nhịp tim của rái cá sông lặn xuống chỉ còn 20 nhịp mỗi phút so với mức bình thường 170 của nó,

bảo tồn oxy và cho phép rái cá ở dưới nước lâu hơn. Những con vật

hòa đồng và dễ thuần hóa, và ở Châu Âu, một loài có liên quan đã từng là

được huấn luyện để bắt cá cho người dân ăn.

Beverly Goldfarb là copyeditor cho Lập trình mạng Java ™, Thứ hai

Phiên bản. Deborah English là người hiệu đính. Jeffrey Holcomb, Sarah Jane

Shangraw và Claire Cloutier đã thực hiện các bài đánh giá kiểm soát chất lượng. Nancy Crumpton

đã viết chỉ mục. Thành phần nội thất được thực hiện bởi Claire Cloutier, Sarah Jane

Shangraw, Molly Shangraw và Joan McGaw.

Bìa được thiết kế bởi Emma Colby và sử dụng một loạt thiết kế của Edie Freeman. Các

ảnh bìa là bản khắc từ thế kỷ 19 từ Kho lưu trữ Báo ảnh Dover. Emma

Colby đã tạo bố cục bìa bằng QuarkXPress 4.1 sử dụng ITC Garamond của Adobe

nét chữ. Alicia Cech và David Futato thiết kế bố trí nội thất.

Văn bản được tạo trong FrameMaker 5.5.6 sử dụng mẫu do Mike triển khai

Sierra. Phông chữ tiêu đề là Bodoni BT; phông chữ văn bản là New Baskerville. Các

các hình minh họa xuất hiện trong sách được tạo trong Macromedia Freehand 8 và

Adobe Photoshop 5 của Robert Romano và Rhon Porter. Colophon này đã được viết

của Leanne Soylemez.