



JavaMail API

Contenido

API de JavaMail	3
Arquitectura	4
Configuración del entorno	5
servidor SMTP	6
Clases principales	6
Envío de correos electrónicos sencillos	7
Crear clase de Java	8
Compilar y ejecutar	10
Verificar salida	10
Envío de correo electrónico con archivo adjunto	11
Crear clase de Java	12
Compilar y ejecutar	15
Verificar salida	15
Envío de un correo electrónico HTML	16
Crear clase de Java	16
Compilar y ejecutar	19
Verificar salida	19
Envío de correo electrónico con imágenes en línea	20
Crear clase de Java	21



Compilar y ejecutar	24
Verificar salida	25
Comprobación de correos electrónicos	25
Crear clase de Java	26
Compilar y ejecutar	29
Verificar salida	29
Obteniendo correos electrónicos	30
Crear clase de Java	30
Compilar y ejecutar	38
Verificar salida	38
Autenticación.....	41
Compilar y ejecutar	44
Verificar salida	44
Responder correos electrónicos	45
Crear clase de Java	45
Compilar y ejecutar	50
Verificar salida	50
Reenvío de correos electrónicos.....	51
Crear clase de Java	51
Compilar y ejecutar	56
Verificar salida	56
Eliminar correos electrónicos.....	57
Crear clase de Java	58
Compilar y ejecutar	61
Verificar salida	61
Servidor SMTP de Gmail.....	62
Crear clase de Java	62
Compilar y ejecutar	65
Verificar salida	65
Gestión de carpetas	65
Abrir una carpeta	65
Información básica de la carpeta	66
Administrar carpeta	67
Administrar mensajes en carpetas	67

Listado del contenido de una carpeta	68
Comprobación de correo	69
Obtener mensajes de carpetas	69
Carpetas de búsqueda	70
Banderas.....	71
Gestión de cuotas.....	72
Crear clase de Java	72
Compilar y ejecutar	74
Verificar salida	75
Mensajes rebotados.....	75
Crear clase de Java	75
Compilar y ejecutar	77
Verificar salida	78
Servidores SMTP	78
Servidores IMAP	80
Servidores POP3.....	81

API de JavaMail

La API de JavaMail proporciona un marco de trabajo independiente de la plataforma y del protocolo para crear aplicaciones de mensajería y correo. La API de JavaMail proporciona un conjunto de clases abstractas que definen los objetos que componen un sistema de correo. Es un paquete opcional (extensión estándar) para leer, redactar y enviar mensajes electrónicos.

JavaMail proporciona elementos que se utilizan para construir una interfaz para un sistema de mensajería, incluidos los componentes y las interfaces del sistema. Si bien esta especificación no define ninguna implementación específica, JavaMail incluye varias clases que implementan los estándares de mensajería de Internet RFC822 y MIME. Estas clases se entregan como parte del paquete de clases JavaMail.

A continuación, se muestran algunos de los protocolos compatibles con la API de JavaMail:

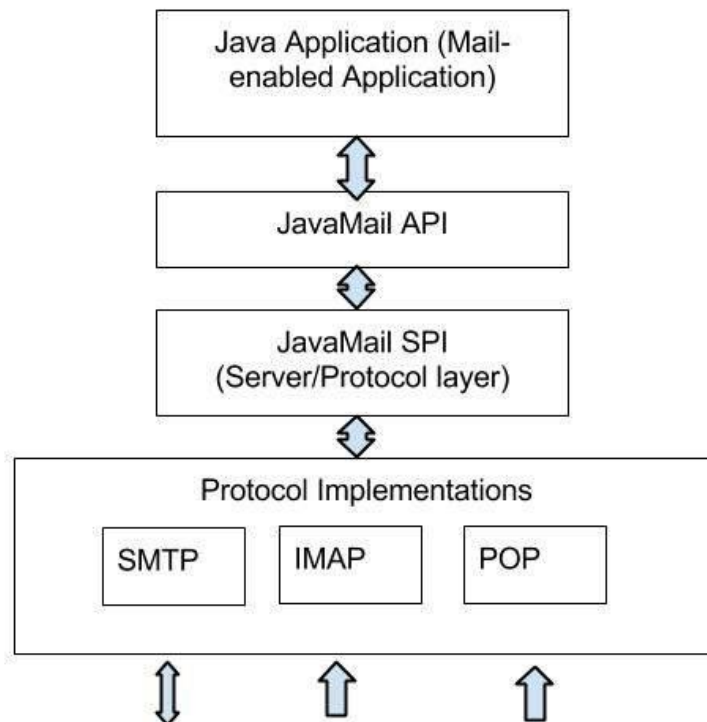
- **SMTP** : Acrónimo de **Simple Mail Transfer Protocol (Protocolo simple de transferencia de correo)** . Proporciona un mecanismo para enviar correo electrónico.

- **POP** : Acrónimo de **Post Office Protocol** . POP es el mecanismo que utiliza la mayoría de las personas en Internet para obtener su correo. Define la compatibilidad con un solo buzón de correo para cada usuario. RFC 1939 define este protocolo.
- **IMAP** : acrónimo de **Internet Message Access Protocol (Protocolo de acceso a mensajes de Internet)** . Es un protocolo avanzado para recibir mensajes. Proporciona soporte para varios buzones de correo para cada usuario, además de que varios usuarios pueden compartir el buzón de correo. Está definido en RFC 2060.
- **MIME** : Acrónimo de **Extensiones multipropósito de correo de Internet** . . No es un protocolo de transferencia de correo. En cambio, define el contenido de lo que se transfiere: el formato de los mensajes, los archivos adjuntos, etc. Hay muchos documentos diferentes que entran en vigor aquí: RFC 822, RFC 2045, RFC 2046 y RFC 2047. Como usuario de la API de JavaMail, normalmente no necesita preocuparse por estos formatos. Sin embargo, estos formatos existen y son utilizados por sus programas.
- **NNTP y otros** : hay muchos protocolos proporcionados por proveedores externos. Algunos de ellos son Protocolo de transferencia de noticias en red (NNTP), Extensiones seguras de correo de Internet multipropósito (S / MIME), etc.

Los detalles de estos se tratarán en los capítulos siguientes.

Arquitectura

Como se dijo anteriormente, la aplicación Java utiliza la API de JavaMail para redactar, enviar y recibir correos electrónicos. La siguiente figura ilustra la arquitectura de JavaMail:



El mecanismo abstracto de la API de JavaMail es similar a otras API de J2EE, como JDBC, JNDI y JMS. Como se ve en el diagrama de arquitectura anterior, la API de JavaMail se divide en dos partes principales:

- Una parte independiente de la aplicación: los componentes de la aplicación utilizan una interfaz de programación de aplicaciones (API) para enviar y recibir mensajes de correo, independientemente del proveedor o protocolo subyacente utilizado.
- Una parte que depende del servicio: una interfaz de proveedor de servicios (SPI) habla los idiomas específicos del protocolo, como SMTP, POP, IMAP y Network News Transfer Protocol (NNTP). Se utiliza para conectar un proveedor de un servicio de correo electrónico a la plataforma J2EE.

Configuración del entorno

Enviar un correo electrónico usando su aplicación Java es bastante simple, pero para empezar debe tener la **API de JavaMail** y el **Marco de activación de Java (JAF)** instalados en su máquina.

Necesitará la extensión **JavaBeans Activation Framework (JAF)** que proporciona el paquete *javax.activation* solo cuando no esté utilizando Java SE 6 o una versión posterior.

- Puede descargar la última versión de JavaMail (Versión 1.5.0) desde el sitio web estándar de Java.
- Puede descargar la última versión de JAF (Versión 1.1.1) del sitio web estándar de Java.

Descargue y descomprima estos archivos, en los directorios de nivel superior recién creados encontrará una serie de archivos jar para ambas aplicaciones. Necesita agregar archivos **mail.jar** y **activación.jar** en su CLASSPATH.

servidor SMTP

Para enviar correos electrónicos, debe tener un servidor SMTP que se encargue de enviar correos. Puede utilizar una de las siguientes técnicas para obtener el servidor SMTP:

- Instale y use cualquier servidor SMTP como el servidor Postfix (para Ubuntu), el servidor Apache James (Java Apache Mail Enterprise Server), etc. (o)
- Utilice el servidor SMTP proporcionado por el proveedor de alojamiento para, por ejemplo: SMTP libre de proporcionar por JangoSMTP sitio es *relay.jangosmtp.net* (o)
- Utilice el servidor SMTP proporcionado por empresas, por ejemplo, gmail, yahoo, etc.

En los ejemplos de los capítulos siguientes, hemos utilizado el servidor gratuito JangoSMTP para enviar correo electrónico. Puede crear una cuenta visitando este sitio y configurar su dirección de correo electrónico.

Clases principales

La API de JavaMail consta de algunas interfaces y clases que se utilizan para enviar, leer y eliminar mensajes de correo electrónico. Aunque hay muchos paquetes en la API de JavaMail, cubrirá los dos paquetes principales que se utilizan con frecuencia en la API de Java Mail: el paquete *javax.mail* y *javax.mail.internet*. Estos paquetes contienen todas las clases principales de JavaMail. Son:

Clase	Descripción
<u>javax.mail.Session</u>	La clase de clave de la API. Un objeto multiproceso representa la fábrica de conexiones.
<u>javax.mail.Message</u>	Una clase abstracta que modela un mensaje de correo electrónico. Las subclases proporcionan las implementaciones reales.

<u>javax.mail.Address</u>	Una clase abstracta que modela las direcciones (desde y hacia direcciones) en un mensaje. Las subclases proporcionan las implementaciones específicas.
<u>javax.mail.Authenticator</u>	Una clase abstracta que se utiliza para proteger los recursos de correo en el servidor de correo.
<u>javax.mail.Transport</u>	Una clase abstracta que modela un mecanismo de transporte de mensajes para enviar un mensaje de correo electrónico.
<u>javax.mail.Store</u>	Una clase abstracta que modela un almacén de mensajes y su protocolo de acceso, para almacenar y recuperar mensajes. Una tienda se divide en carpetas.
<u>javax.mail.Folder</u>	Una clase abstracta que representa una carpeta de mensajes de correo. Puede contener subcarpetas.
javax.mail.internet. MimeMensaje	El mensaje es una clase abstracta, por lo tanto, debe funcionar con una subclase; en la mayoría de los casos, usará un <code>MimeMessage</code> . Un <code>MimeMessage</code> es un mensaje de correo electrónico que comprende los tipos y encabezados MIME.
javax.mail.internet. InternetAddress	Esta clase representa una dirección de correo electrónico de Internet que utiliza la sintaxis de RFC822. La sintaxis de dirección típica tiene el formato <i>usuario@host.domain</i> o <i>Nombre personal <usuario@host.domain></i> .

Envío de correos electrónicos sencillos

A continuación se muestra un ejemplo para enviar un correo electrónico simple. Aquí hemos utilizado el servidor JangoSMTP a través del cual se envían correos electrónicos a nuestra dirección de correo electrónico de destino. La configuración se explica en el capítulo Configuración del entorno.

Para enviar un correo electrónico sencillo, se siguen los siguientes pasos:

- Obtener una sesión
- Cree un objeto `MimeMessage` predeterminado y establezca *From*, *To*, *Subject* en el mensaje.
- Establezca el mensaje real como:

```
message.setText("your text goes here");
```

- Envíe el mensaje utilizando el objeto Transporte.

Crear clase de Java

Cree un archivo de clase Java **SendEmail** , cuyo contenido es el siguiente:

```
package com.codigodata;

import java.util.Properties;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class SendEmail {
    public static void main(String[] args) {
        // Recipient's email ID needs to be mentioned.
        String to = "destinationemail@gmail.com";

        // Sender's email ID needs to be mentioned
        String from = "fromemail@gmail.com";
        final String username = "manishaspatil";//change accordingly
        final String password = "*****";//change accordingly

        // Assuming you are sending email through relay.jangosmtp.net
        String host = "relay.jangosmtp.net";
```



```
Properties props = new Properties();
props.put("mail.smtp.auth", "true");
props.put("mail.smtp.starttls.enable", "true");
props.put("mail.smtp.host", host);
props.put("mail.smtp.port", "25");

// Get the Session object.
Session session = Session.getInstance(props,
    new javax.mail.Authenticator() {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(username, password);
        }
    });

try {
    // Create a default MimeMessage object.
    Message message = new MimeMessage(session);

    // Set From: header field of the header.
    message.setFrom(new InternetAddress(from));

    // Set To: header field of the header.
    message.setRecipients(Message.RecipientType.TO,
        InternetAddress.parse(to));

    // Set Subject: header field
    message.setSubject("Testing Subject");
```

```
// Now set the actual message

message.setText("Hello, this is sample for to check send " +
               "email using JavaMailAPI ");

// Send message

Transport.send(message);

System.out.println("Sent message successfully....");

} catch (MessagingException e) {
    throw new RuntimeException(e);
}
}
```

Como usamos el servidor SMTP proporcionado por el proveedor de host JangoSMTP, necesitamos autenticar el nombre de usuario y la contraseña. La clase *javax.mail.PasswordAuthentication* se utiliza para autenticar la contraseña.

Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase `SendEmail.java` en el directorio: / **home** / **manisha** / **JavaMailAPIExercise** . Necesitaríamos los `jars javax.mail.jar` y `activation.jar` en el classpath. Ejecute el siguiente comando para compilar la clase (ambos frascos se colocan en el directorio / home / manisha /) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: SendEmail.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: SendEmail
```

Verificar salida

Debería ver el siguiente mensaje en la consola de comandos:

```
Sent message successfully....
```

Como envío un correo electrónico a mi dirección de Gmail a través de JangoSMTP, se recibiría el siguiente correo en la bandeja de entrada de mi cuenta de Gmail:



Envío de correo electrónico con archivo adjunto

Aquí hay un ejemplo para enviar un correo electrónico con un archivo adjunto desde su máquina. El archivo en la máquina local es **file.txt** ubicado en `/home/manisha/`. Aquí hemos utilizado el servidor JangoSMTP a través del cual se envían correos electrónicos a nuestra dirección de correo electrónico de destino. La configuración se explica en el capítulo configuración del entorno.

Para enviar un correo electrónico con una imagen en línea, los pasos que se siguen son:

- Obtener una sesión
- Cree un objeto `MimeMessage` predeterminado y establezca *From*, *To*, *Subject* en el mensaje.
- Configure el mensaje real como se muestra a continuación:

```
messageBodyPart.setText("This is message body");
```

- Cree un objeto `MimeMultipart`. Agregue el `messageBodyPart` anterior con el mensaje real establecido en él, a este objeto multiparte.
- A continuación, agregue el archivo adjunto creando un `DataHandler` de la siguiente manera:

```
messageBodyPart = new MimeBodyPart();  
  
String filename = "/home/manisha/file.txt";  
  
DataSource source = new FileDataSource(filename);  
  
messageBodyPart.setDataHandler(new DataHandler(source));  
  
messageBodyPart.setFileName(filename);  
  
multipart.addBodyPart(messageBodyPart);
```

- A continuación, configure el multipart en el mensaje de la siguiente manera:

```
message.setContent(multipart);
```

- Envíe el mensaje utilizando el objeto Transport.

Crear clase de Java

Cree un archivo de clase Java **SendAttachmentInEmail** , cuyo contenido es el siguiente:

```
package com.codigodata;

import java.util.Properties;

import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.mail.BodyPart;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;

public class SendAttachmentInEmail {

    public static void main(String[] args) {

        // Recipient's email ID needs to be mentioned.
```

```
String to = "destinationemail@gmail.com";

// Sender's email ID needs to be mentioned
String from = "fromemail@gmail.com";

final String username = "manishaspatil";//change accordingly
final String password = "*****";//change accordingly

// Assuming you are sending email through relay.jangosmtp.net
String host = "relay.jangosmtp.net";

Properties props = new Properties();
props.put("mail.smtp.auth", "true");
props.put("mail.smtp.starttls.enable", "true");
props.put("mail.smtp.host", host);
props.put("mail.smtp.port", "25");

// Get the Session object.
Session session = Session.getInstance(props,
    new javax.mail.Authenticator() {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(username, password);
        }
    });

try {
    // Create a default MimeMessage object.
    Message message = new MimeMessage(session);
```

```
// Set From: header field of the header.
message.setFrom(new InternetAddress(from));

// Set To: header field of the header.
message.setRecipients(Message.RecipientType.TO,
    InternetAddress.parse(to));

// Set Subject: header field
message.setSubject("Testing Subject");

// Create the message part
BodyPart messageBodyPart = new MimeBodyPart();

// Now set the actual message
messageBodyPart.setText("This is message body");

// Create a multipart message
Multipart multipart = new MimeMultipart();

// Set text message part
multipart.addBodyPart(messageBodyPart);

// Part two is attachment
messageBodyPart = new MimeBodyPart();
String filename = "/home/manisha/file.txt";
DataSource source = new FileDataSource(filename);
messageBodyPart.setDataHandler(new DataHandler(source));
messageBodyPart.setFileName(filename);
multipart.addBodyPart(messageBodyPart);
```

```
// Send the complete message parts
message.setContent(multipart);

// Send message
Transport.send(message);

System.out.println("Sent message successfully....");

} catch (MessagingException e) {
    throw new RuntimeException(e);
}
}
```

Como usamos el servidor SMTP proporcionado por el proveedor de host JangoSMTP, necesitamos autenticar el nombre de usuario y la contraseña. La clase *javax.mail.PasswordAuthentication* se utiliza para autenticar la contraseña.

Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase *SendAttachmentInEmail.java* en el directorio: / **home** / **manisha** / **JavaMailAPIExercise** . Necesitaríamos los jars *javax.mail.jar* y *activation.jar* en el classpath. Ejecute el siguiente comando para compilar la clase (ambos frascos se colocan en el directorio / home / manisha /) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar:
SendAttachmentInEmail.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar:
SendAttachmentInEmail
```

Verificar salida

Debería ver el siguiente mensaje en la consola de comandos:

Sent message successfully....

Como envío un correo electrónico a mi dirección de Gmail a través de JangoSMTP, se recibiría el siguiente correo en la bandeja de entrada de mi cuenta de Gmail:



Envío de un correo electrónico HTML

Aquí hay un ejemplo para enviar un correo electrónico HTML desde su máquina. Aquí hemos utilizado el servidor JangoSMTP a través del cual se envían correos electrónicos a nuestra dirección de correo electrónico de destino. La configuración se explica en el capítulo configuración del entorno .

Este ejemplo es muy similar al envío de correo electrónico simple, excepto que, aquí estamos usando el método `setContent ()` para establecer contenido cuyo segundo argumento es "texto / html" para especificar que el contenido HTML está incluido en el mensaje. Con este ejemplo, puede enviar contenido HTML tan grande como desee.

Para enviar un correo electrónico con contenido HTML, los pasos a seguir son:

- Obtener una sesión
- Cree un objeto `MimeMessage` predeterminado y establezca *From*, *To*, *Subject* en el mensaje.
- Configure el mensaje real usando el método `setContent ()` como se muestra a continuación:

```
message.setContent("<h1>This is actual message embedded in  
HTML tags</h1>", "text/html");
```

- Envíe el mensaje utilizando el objeto Transporte.

Crear clase de Java

Cree un archivo de clase Java **SendHTMLEmail** , cuyo contenido es el siguiente:


```
package com.codigodata;

import java.util.Properties;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class SendHTMLEmail {

    public static void main(String[] args) {

        // Recipient's email ID needs to be mentioned.
        String to = "destinationemail@gmail.com";

        // Sender's email ID needs to be mentioned
        String from = "fromemail@gmail.com";
        final String username = "manishaspatil";//change accordingly
        final String password = "*****";//change accordingly

        // Assuming you are sending email through relay.jangosmtp.net
        String host = "relay.jangosmtp.net";

        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", host);
```

```
props.put("mail.smtp.port", "25");

// Get the Session object.
Session session = Session.getInstance(props,
    new javax.mail.Authenticator() {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(username, password);
        }
    });

try {
    // Create a default MimeMessage object.
    Message message = new MimeMessage(session);

    // Set From: header field of the header.
    message.setFrom(new InternetAddress(from));

    // Set To: header field of the header.
    message.setRecipients(Message.RecipientType.TO,
        InternetAddress.parse(to));

    // Set Subject: header field
    message.setSubject("Testing Subject");

    // Send the actual HTML message, as big as you like
    message.setContent(
        "<h1>This is actual message embedded in HTML tags</h1>",
        "text/html");
}
```

```
// Send message

Transport.send(message);

System.out.println("Sent message successfully....");

} catch (MessagingException e) {

    e.printStackTrace();

    throw new RuntimeException(e);

}

}
```

Como usamos el servidor SMTP proporcionado por el proveedor de host JangoSMTP, necesitamos autenticar el nombre de usuario y la contraseña. La clase *javax.mail.PasswordAuthentication* se utiliza para autenticar la contraseña.

Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase *SendHTMLEmail.java* en el directorio: / **home** / **manisha** / **JavaMailAPIExercise** . Necesitaríamos los jars *javax.mail.jar* y *activation.jar* en el classpath. Ejecute el siguiente comando para compilar la clase (ambos jar se colocan en el directorio / home / manisha /) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar:
SendHTMLEmail.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: SendHTMLEmail
```

Verificar salida

Debería ver el siguiente mensaje en la consola de comandos:

```
Sent message successfully....
```

Como envío un correo electrónico a mi dirección de Gmail a través de JangoSMTP, se recibiría el siguiente correo en la bandeja de entrada de mi cuenta de Gmail:

Testing Subject



Inbox x



[Redacted]@gmail.com via jangomail.com

Oct 10

to me



Images are not displayed. [Display images below](#)

This is actual message embedded in HTML tags

Envío de correo electrónico con imágenes en línea

Aquí hay un ejemplo para enviar un correo electrónico HTML desde su máquina con una imagen en línea. Aquí hemos utilizado el servidor JangoSMTP a través del cual se envían correos electrónicos a nuestra dirección de correo electrónico de destino. La configuración se explica en el capítulo Configuración del entorno.

Para enviar un correo electrónico con una imagen en línea, los pasos que se siguen son:

- Obtener una sesión
- Cree un objeto `MimeMessage` predeterminado y establezca *From*, *To*, *Subject* en el mensaje.
- Cree un objeto `MimeMultipart`.
- En nuestro ejemplo, tendremos una parte HTML y una imagen en el correo electrónico. Entonces, primero cree el contenido HTML y configúrelo en el objeto multiparte como:

```
// first part (the html)

BodyPart messageBodyPart = new MimeBodyPart();

String htmlText = "<H1>Hello</H1><img src=\"cid:image\">";

messageBodyPart.setContent(htmlText, "text/html");

// add it

multipart.addBodyPart(messageBodyPart);
```

- A continuación, agregue la imagen creando un `DataHandler` de la siguiente manera:

```
// second part (the image)
```



```
messageBodyPart = new MimeBodyPart();

DataSource fds = new FileDataSource(
    "/home/manisha/javamail-mini-logo.png");

messageBodyPart.setDataHandler(new DataHandler(fds));
messageBodyPart.setHeader("Content-ID", "<image>");
```

- A continuación, configure el multiparte en el mensaje de la siguiente manera:

```
message.setContent(multipart);
```

- Envíe el mensaje utilizando el objeto Transport.

Crear clase de Java

Cree un archivo de clase Java **SendInlinelImagesInEmail** , cuyo contenido es el siguiente:

```
package com.codigodata;

import java.util.Properties;
import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.mail.BodyPart;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
```

```
public class SendInlinImagesInEmail {  
    public static void main(String[] args) {  
        // Recipient's email ID needs to be mentioned.  
        String to = "destinationemail@gmail.com";  
  
        // Sender's email ID needs to be mentioned  
        String from = "fromemail@gmail.com";  
        final String username = "manishaspatil";//change accordingly  
        final String password = "*****";//change accordingly  
  
        // Assuming you are sending email through relay.jangosmtp.net  
        String host = "relay.jangosmtp.net";  
  
        Properties props = new Properties();  
        props.put("mail.smtp.auth", "true");  
        props.put("mail.smtp.starttls.enable", "true");  
        props.put("mail.smtp.host", host);  
        props.put("mail.smtp.port", "25");  
  
        Session session = Session.getInstance(props,  
            new javax.mail.Authenticator() {  
                protected PasswordAuthentication getPasswordAuthentication() {  
                    return new PasswordAuthentication(username, password);  
                }  
            });  
  
        try {
```

```
// Create a default MimeMessage object.
Message message = new MimeMessage(session);

// Set From: header field of the header.
message.setFrom(new InternetAddress(from));

// Set To: header field of the header.
message.setRecipients(Message.RecipientType.TO,
    InternetAddress.parse(to));

// Set Subject: header field
message.setSubject("Testing Subject");

// This mail has 2 part, the BODY and the embedded image
MimeMultipart multipart = new MimeMultipart("related");

// first part (the html)
BodyPart messageBodyPart = new MimeBodyPart();
String htmlText = "<H1>Hello</H1><img src=\"cid:image\">";
messageBodyPart.setContent(htmlText, "text/html");
// add it
multipart.addBodyPart(messageBodyPart);

// second part (the image)
messageBodyPart = new MimeBodyPart();
DataSource fds = new FileDataSource(
    "/home/manisha/javamail-mini-logo.png");

messageBodyPart.setDataHandler(new DataHandler(fds));
```

```
messageBodyPart.setHeader("Content-ID", "<image>");

// add image to the multipart
multipart.addBodyPart(messageBodyPart);

// put everything together
message.setContent(multipart);

// Send message
Transport.send(message);

System.out.println("Sent message successfully....");

} catch (MessagingException e) {
    throw new RuntimeException(e);
}
}
```

Como usamos el servidor SMTP proporcionado por el proveedor de host JangoSMTP, necesitamos autenticar el nombre de usuario y la contraseña. La clase *javax.mail.PasswordAuthentication* se utiliza para autenticar la contraseña.

Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase *SendInlinelImagesInEmail.java* en el directorio: **/ home / manisha / JavaMailAPIExercise** . Necesitaríamos los *jars javax.mail.jar* y *activation.jar* en el classpath. Ejecute el siguiente comando para compilar la clase (ambos frascos se colocan en el directorio **/ home / manisha /**) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar:
SendInlinelImagesInEmail.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:


```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar:
```

[SendInlinedImagesInEmail](#)

Verificar salida

Debería ver el siguiente mensaje en la consola de comandos:

[Sent](#) message successfully....

Como envió un correo electrónico a mi dirección de Gmail a través de JangoSMTP, se recibiría el siguiente correo en la bandeja de entrada de mi cuenta de Gmail:



Comprobación de correos electrónicos

Hay dos aspectos que es necesario comprender antes de continuar con este capítulo. Son **Check** and **Fetch**.

- **Checking** un correo electrónico en JavaMail es un proceso en el que abrimos la carpeta respectiva en el buzón y recibimos cada mensaje. Aquí sólo comprobar el encabezado de cada mensaje, es decir, el *De*, *A*, *sujeto*. No se lee el contenido.
- **Fetching** un correo electrónico en JavaMail es un proceso en el que abrimos la carpeta respectiva en el buzón y obtenemos cada mensaje. Junto con el encabezado, también leemos el contenido reconociendo el tipo de contenido.

Para verificar o recuperar un correo electrónico usando la API de JavaMail, necesitaríamos servidores POP o IMAP. Para comprobar y recuperar los correos electrónicos, se necesitan las clases Carpeta y Tienda. Aquí hemos utilizado el servidor POP3 de GMAIL (pop.gmail.com). En este capítulo aprenderá a revisar correos electrónicos usando la API de JavaMail. La recuperación se tratará en los capítulos siguientes. Para revisar correos electrónicos:

- Obtener una sesión
- Cree el objeto pop3 Store y conéctese con el servidor pop.
- Crear objeto de carpeta. Abra la carpeta correspondiente en su buzón.
- Recibe tus mensajes.
- Cierre los objetos Store y Folder.

Crear clase de Java

Cree un archivo de clase java **CheckingMails** , cuyo contenido es el siguiente:

```
package com.codigodata;

import java.util.Properties;

import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.NoSuchProviderException;
import javax.mail.Session;
import javax.mail.Store;

public class CheckingMails {

    public static void check(String host, String storeType, String user,
        String password)
    {
        try {
```

```
//create properties field
Properties properties = new Properties();

properties.put("mail.pop3.host", host);
properties.put("mail.pop3.port", "995");
properties.put("mail.pop3.starttls.enable", "true");
Session emailSession = Session.getDefaultInstance(properties);

//create the POP3 store object and connect with the pop server
Store store = emailSession.getStore("pop3s");

store.connect(host, user, password);

//create the folder object and open it
Folder emailFolder = store.getFolder("INBOX");
emailFolder.open(Folder.READ_ONLY);

// retrieve the messages from the folder in an array and print it
Message[] messages = emailFolder.getMessages();
System.out.println("messages.length---" + messages.length);

for (int i = 0, n = messages.length; i < n; i++) {
    Message message = messages[i];
    System.out.println("-----");
    System.out.println("Email Number " + (i + 1));
    System.out.println("Subject: " + message.getSubject());
    System.out.println("From: " + message.getFrom()[0]);
    System.out.println("Text: " + message.getContent().toString());
}
```

```
}

//close the store and folder objects
emailFolder.close(false);
store.close();

} catch (NoSuchProviderException e) {
    e.printStackTrace();
} catch (MessagingException e) {
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
}
}

public static void main(String[] args) {

    String host = "pop.gmail.com";// change accordingly
    String mailStoreType = "pop3";
    String username = "yourmail@gmail.com";// change accordingly
    String password = "*****";// change accordingly

    check(host, mailStoreType, username, password);

}

}
```



Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase `CheckingMails.java` en el directorio: `/ home / manisha / JavaMailAPIExercise`. Necesitaríamos los `javax.mail.jar` y `activation.jar` en el classpath. Ejecute el siguiente comando para compilar la clase (ambos frascos se colocan en el directorio `/ home / manisha /`) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: CheckingMails.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: CheckingMails
```

Verificar salida

Debería ver el siguiente mensaje en la consola de comandos:

```
messages.length---4
-----
Email Number 1
Subject: Test Mail--Fetch
From: <abcd@gmail.com>
Text: javax.mail.internet.MimeMultipart@327a5b7f
-----
Email Number 2
Subject: testing ----checking simple email
From: <abcd@gmail.com>
Text: javax.mail.internet.MimeMultipart@7f0d08bc
-----
Email Number 3
Subject: Email with attachment
From: <abcd@gmail.com>
Text: javax.mail.internet.MimeMultipart@30b8afce
-----
Email Number 4
```



Subject: Email with Inline image

From: <abcd@gmail.com>

Text: javax.mail.internet.MimeMultipart@2d1e165f

Aquí hemos impreso el número de mensajes en el INBOX que es 4 en este caso. También hemos impreso Asunto, Dirección de remitente y Texto para cada mensaje de correo electrónico.

Obteniendo correos electrónicos

En el capítulo anterior aprendimos cómo revisar los correos electrónicos. Ahora veamos cómo recuperar cada correo electrónico y leer su contenido. Escribamos una clase de Java **FetchingEmail** que leerá los siguientes tipos de correos electrónicos:

- Correo electrónico simple
- Correo electrónico con archivo adjunto
- Correo electrónico con imagen en línea

Los pasos básicos seguidos en el código son los siguientes:

- Obtenga el objeto Session.
- Cree un objeto de tienda POP3 y conéctese a la tienda.
- Cree el objeto Carpeta y abra la carpeta correspondiente en su buzón.
- Recuperar mensajes.
- Cierre la carpeta y almacene los objetos respectivamente.

Crear clase de Java

Cree un archivo de clase Java **FetchingEmail**, cuyo contenido es el siguiente:

```
package com.codigodata;

import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
```

```
import java.io.InputStreamReader;

import java.util.Date;

import java.util.Properties;


import javax.mail.Address;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.NoSuchProviderException;
import javax.mail.Part;
import javax.mail.Session;
import javax.mail.Store;


public class FetchingEmail {


    public static void fetch(String pop3Host, String storeType, String user,
        String password) {
        try {
            // create properties field
            Properties properties = new Properties();
            properties.put("mail.store.protocol", "pop3");
            properties.put("mail.pop3.host", pop3Host);
            properties.put("mail.pop3.port", "995");
            properties.put("mail.pop3.starttls.enable", "true");

            Session emailSession = Session.getDefaultInstance(properties);

            // emailSession.setDebug(true);


            // create the POP3 store object and connect with the pop server
```

```
Store store = emailSession.getStore("pop3s");

store.connect(pop3Host, user, password);

// create the folder object and open it
Folder emailFolder = store.getFolder("INBOX");
emailFolder.open(Folder.READ_ONLY);

BufferedReader reader = new BufferedReader(new InputStreamReader(
    System.in));

// retrieve the messages from the folder in an array and print it
Message[] messages = emailFolder.getMessages();
System.out.println("messages.length---" + messages.length);

for (int i = 0; i < messages.length; i++) {
    Message message = messages[i];
    System.out.println("-----");
    writePart(message);
    String line = reader.readLine();
    if ("YES".equals(line)) {
        message.writeTo(System.out);
    } else if ("QUIT".equals(line)) {
        break;
    }
}

// close the store and folder objects
emailFolder.close(false);
```



```
store.close();

} catch (NoSuchProviderException e) {
    e.printStackTrace();
} catch (MessagingException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
}
}

public static void main(String[] args) {

    String host = "pop.gmail.com";// change accordingly
    String mailStoreType = "pop3";
    String username =
        "abc@gmail.com";// change accordingly
    String password = "*****";// change accordingly

    //Call method fetch
    fetch(host, mailStoreType, username, password);

}

/*
 * This method checks for content-type
 * based on which, it processes and
```

```
* fetches the content of the message
*/
public static void writePart(Part p) throws Exception {
    if (p instanceof Message)
        //Call method writeEnvelope
        writeEnvelope((Message) p);

    System.out.println("-----");
    System.out.println("CONTENT-TYPE: " + p.getContentType());

    //check if the content is plain text
    if (p.isMimeType("text/plain")) {
        System.out.println("This is plain text");
        System.out.println("-----");
        System.out.println((String) p.getContent());
    }

    //check if the content has attachment
    else if (p.isMimeType("multipart/*")) {
        System.out.println("This is a Multipart");
        System.out.println("-----");
        Multipart mp = (Multipart) p.getContent();
        int count = mp.getCount();
        for (int i = 0; i < count; i++)
            writePart(mp.getBodyPart(i));
    }

    //check if the content is a nested message
    else if (p.isMimeType("message/rfc822")) {
        System.out.println("This is a Nested Message");
        System.out.println("-----");
    }
}
```

```
writePart((Part) p.getContent());
}

//check if the content is an inline image
else if (p.isMimeType("image/jpeg")) {
    System.out.println("-----> image/jpeg");
    Object o = p.getContent();

    InputStream x = (InputStream) o;
    // Construct the required byte array
    System.out.println("x.length = " + x.available());
    int i = 0;
    byte[] bArray = new byte[x.available()];

    while ((i = (int) ((InputStream) x).available()) > 0) {
        int result = (int) (((InputStream) x).read(bArray));
        if (result == -1)
            break;
    }

    FileOutputStream f2 = new FileOutputStream("/tmp/image.jpg");
    f2.write(bArray);
}

else if (p.getContentType().contains("image/")) {
    System.out.println("content type" + p.getContentType());
    File f = new File("image" + new Date().getTime() + ".jpg");
    DataOutputStream output = new DataOutputStream(
        new BufferedOutputStream(new FileOutputStream(f)));
    com.sun.mail.util.BASE64DecoderStream test =
        (com.sun.mail.util.BASE64DecoderStream) p
            .getContent();
```

```
byte[] buffer = new byte[1024];

int bytesRead;

while ((bytesRead = test.read(buffer)) != -1) {
    output.write(buffer, 0, bytesRead);
}

}

else {
    Object o = p.getContent();
    if (o instanceof String) {
        System.out.println("This is a string");
        System.out.println("-----");
        System.out.println((String) o);
    }
    else if (o instanceof InputStream) {
        System.out.println("This is just an input stream");
        System.out.println("-----");
        InputStream is = (InputStream) o;
        is = (InputStream) o;
        int c;
        while ((c = is.read()) != -1)
            System.out.write(c);
    }
    else {
        System.out.println("This is an unknown type");
        System.out.println("-----");
        System.out.println(o.toString());
    }
}

}
```

```
}  
  
/*  
 * This method would print FROM,TO and SUBJECT of the message  
 */  
public static void writeEnvelope(Message m) throws Exception {  
    System.out.println("This is the message envelope");  
    System.out.println("-----");  
    Address[] a;  
  
    // FROM  
    if ((a = m.getFrom()) != null) {  
        for (int j = 0; j < a.length; j++)  
            System.out.println("FROM: " + a[j].toString());  
    }  
  
    // TO  
    if ((a = m.getRecipients(Message.RecipientType.TO)) != null) {  
        for (int j = 0; j < a.length; j++)  
            System.out.println("TO: " + a[j].toString());  
    }  
  
    // SUBJECT  
    if (m.getSubject() != null)  
        System.out.println("SUBJECT: " + m.getSubject());  
  
    }  
  
}
```

Puede activar la depuración descomentando la declaración `emailSession.setDebug(true);`

Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase `FetchingEmail.java` en el directorio: `/ home / manisha / JavaMailAPIExercise`. Necesitaríamos los jars `javax.mail.jar` y `activation.jar` en el classpath. Ejecute el siguiente comando para compilar la clase (ambos frascos se colocan en el directorio `/ home / manisha /`) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: FetchingEmail.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: FetchingEmail
```

Verificar salida

Debería ver el siguiente mensaje en la consola de comandos:

```
messages.length---3
-----
This is the message envelope
-----
FROM: XYZ <xyz@gmail.com>
TO: ABC <abc@gmail.com>
SUBJECT: Simple Message
-----
CONTENT-TYPE: multipart/alternative; boundary=047d7b343d6ad3e4ea04e8ec6579
This is a Multipart
-----
-----
CONTENT-TYPE: text/plain; charset=ISO-8859-1
This is plain text
-----
Hi am a simple message string....
```



--

Regards

xyz

This is the message envelope

FROM: XYZ <xyz@gmail.com>

TO: ABC <abc@gmail.com>

SUBJECT: Attachement

CONTENT-TYPE: multipart/mixed; boundary=047d7b343d6a99180904e8ec6751

This is a Multipart

CONTENT-TYPE: text/plain; charset=ISO-8859-1

This is plain text

Hi I've an attachment.Please check

--

Regards

XYZ

CONTENT-TYPE: application/octet-stream; name=sample_attachement

This is just an input stream

Submit your Tutorials, White Papers and Articles into our Tutorials Directory. This is a tutorials database where we are keeping all the tutorials shared by the internet community for the benefit of others.

This is the message envelope

FROM: XYZ <xyz@gmail.com>

TO: ABC <abc@gmail.com>

SUBJECT: Inline Image

CONTENT-TYPE: multipart/related; boundary=f46d04182582be803504e8ece94b

This is a Multipart

CONTENT-TYPE: text/plain; charset=ISO-8859-1

This is plain text

Hi I've an [inline](#) image

[image: [Inline image 3](#)]

--

Regards

XYZ



CONTENT-TYPE: image/png; name="javamail-mini-logo.png"

content typeimage/png; name="javamail-mini-logo.png"

Aquí puede ver que hay tres correos electrónicos en nuestro buzón. Primero un simple correo con el mensaje "Hola, soy un mensaje simple". El segundo correo tiene un archivo adjunto. El contenido del adjunto también se imprime como se ve arriba. El tercer correo tiene una imagen en línea.

Autenticación

Modificaremos nuestro CheckingMails.java desde el capítulo Comprobación de correos electrónicos . Su contenido es el siguiente:

```
package com.codigodata;

import java.util.Properties;

import javax.mail.Authenticator;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.NoSuchProviderException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Store;

public class CheckingMails {

    public static void check(String host, String storeType, String user,
        String password)
    {
        try {

            // create properties field
```

```
Properties properties = new Properties();

properties.put("mail.pop3s.host", host);
properties.put("mail.pop3s.port", "995");
properties.put("mail.pop3s.starttls.enable", "true");

// Setup authentication, get session
Session emailSession = Session.getInstance(properties,
    new javax.mail.Authenticator() {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(
                "manishapatil3may@gmail.com", "manisha123");
        }
    });

// emailSession.setDebug(true);

// create the POP3 store object and connect with the pop server
Store store = emailSession.getStore("pop3s");

store.connect();

// create the folder object and open it
Folder emailFolder = store.getFolder("INBOX");
emailFolder.open(Folder.READ_ONLY);

// retrieve the messages from the folder in an array and print it
Message[] messages = emailFolder.getMessages();
System.out.println("messages.length---" + messages.length);
```

```
for (int i = 0, n = messages.length; i < n; i++) {  
    Message message = messages[i];  
    System.out.println("-----");  
    System.out.println("Email Number " + (i + 1));  
    System.out.println("Subject: " + message.getSubject());  
    System.out.println("From: " + message.getFrom()[0]);  
    System.out.println("Text: " + message.getContent().toString());  
}  
  
// close the store and folder objects  
emailFolder.close(false);  
store.close();  
  
} catch (NoSuchProviderException e) {  
    e.printStackTrace();  
} catch (MessagingException e) {  
    e.printStackTrace();  
} catch (Exception e) {  
    e.printStackTrace();  
}  
}  
  
public static void main(String[] args) {  
  
    String host = "pop.gmail.com";// change accordingly  
    String mailStoreType = "pop3";  
    String username = "abc@gmail.com";// change accordingly  
    String password = "*****";// change accordingly
```

```
        check(host, mailStoreType, username, password);

    }

}
```

Puede activar la depuración descomentando la declaración `emailSession.setDebug(true);`

Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase `CheckingMails.java` en el directorio: **/ home / manisha / JavaMailAPIExercise** . Necesitaríamos los jars `javax.mail.jar` y `activation.jar` en el classpath. Ejecute el siguiente comando para compilar la clase (ambos frascos se colocan en el directorio `/ home / manisha /`) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: CheckingMails.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: CheckingMails
```

Verificar salida

Puede ver un mensaje similar al siguiente en la consola de comandos:

```
messages.length---3
-----
Email Number 1
Subject: Today is a nice day
From: XYZ <xyz@gmail.com>
Text: javax.mail.internet.MimeMultipart@45f676cb
-----
Email Number 2
Subject: hiii...
From: XYZ <xyz@gmail.com>
Text: javax.mail.internet.MimeMultipart@37f12d4f
```

Email Number 3

Subject: helloo

From: XYZ <xyz@gmail.com>

Text: javax.mail.internet.MimeMultipart@3ad5ba3a

Responder correos electrónicos

En este capítulo veremos cómo responder a un correo electrónico utilizando la API de JavaMail. Los pasos básicos seguidos en el programa a continuación son:

- Obtenga el objeto Session con los detalles del servidor POP y SMTP en las propiedades. Necesitaríamos detalles de POP para recuperar mensajes y detalles de SMTP para enviar mensajes.
- Cree un objeto store POP3 y conéctese a store.
- Cree el objeto Carpeta y abra la carpeta correspondiente en su buzón.
- Recuperar mensajes.
- Repite los mensajes y escribe "Y" o "y" si quieres responder.
- Obtenga toda la información (Para, De, Asunto, Contenido) del mensaje.
- Genere el mensaje de respuesta utilizando el método Message.reply (). Este método configura un nuevo mensaje con el destinatario y el asunto adecuados. El método toma un parámetro booleano que indica si responder solo al remitente (falso) o responder a todos (verdadero).
- Establezca De, Texto y Responder a en el mensaje y envíelo a través de la instancia del objeto Transport.
- Cierre los objetos Transport, folder y store respectivamente.

Aquí hemos utilizado el servidor JangoSMTP a través del cual se envían correos electrónicos a nuestra dirección de correo electrónico de destino. La configuración se explica en el capítulo Configuración del entorno.

Crear clase de Java

Cree un archivo de clase Java **ReplyToEmail** , cuyo contenido es el siguiente:

```
package com.codigodata;  
  
import java.io.BufferedReader;  
import java.io.InputStreamReader;
```

```
import java.util.Date;

import java.util.Properties;


import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Store;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;


public class ReplyToEmail {
    public static void main(String args[])
    {
        Date date = null;


        Properties properties = new Properties();
        properties.put("mail.store.protocol", "pop3");
        properties.put("mail.pop3s.host", "pop.gmail.com");
        properties.put("mail.pop3s.port", "995");
        properties.put("mail.pop3.starttls.enable", "true");
        properties.put("mail.smtp.auth", "true");
        properties.put("mail.smtp.starttls.enable", "true");
        properties.put("mail.smtp.host", "relay.jangosmtp.net");
        properties.put("mail.smtp.port", "25");

        Session session = Session.getDefaultInstance(properties);


        // session.setDebug(true);

        try
```

```
{  
  
    // Get a Store object and connect to the current host  
  
    Store store = session.getStore("pop3s");  
  
    store.connect("pop.gmail.com", "xyz@gmail.com",  
        "*****");//change the user and password accordingly  
  
    Folder folder = store.getFolder("inbox");  
  
    if (!folder.exists()) {  
        System.out.println("inbox not found");  
        System.exit(0);  
    }  
  
    folder.open(Folder.READ_ONLY);  
  
    BufferedReader reader = new BufferedReader(new InputStreamReader(  
        System.in));  
  
    Message[] messages = folder.getMessages();  
  
    if (messages.length != 0) {  
  
        for (int i = 0, n = messages.length; i < n; i++) {  
            Message message = messages[i];  
            date = message.getSentDate();  
  
            // Get all the information from the message  
            String from = InternetAddress.toString(message.getFrom());  
            if (from != null) {  
                System.out.println("From: " + from);  
            }  
  
            String replyTo = InternetAddress.toString(message  
                .getReplyTo());
```

```
if (replyTo != null) {  
    System.out.println("Reply-to: " + replyTo);  
}  
  
String to = InternetAddress.toString(message  
    .getRecipients(Message.RecipientType.TO));  
  
if (to != null) {  
    System.out.println("To: " + to);  
}  
  
  
String subject = message.getSubject();  
  
if (subject != null) {  
    System.out.println("Subject: " + subject);  
}  
  
Date sent = message.getSentDate();  
  
if (sent != null) {  
    System.out.println("Sent: " + sent);  
}  
  
  
System.out.print("Do you want to reply [y/n] : ");  
String ans = reader.readLine();  
  
if ("Y".equals(ans) || "y".equals(ans)) {  
  
    Message replyMessage = new MimeMessage(session);  
    replyMessage = (MimeMessage) message.reply(false);  
    replyMessage.setFrom(new InternetAddress(to));  
    replyMessage.setText("Thanks");  
    replyMessage.setReplyTo(message.getReplyTo());  
  
    // Send the message by authenticating the SMTP server
```



```
// Create a Transport instance and call the sendMessage
Transport t = session.getTransport("smtp");
try {
    //connect to the SMTP server using transport instance
    //change the user and password accordingly
    t.connect("abc", "*****");
    t.sendMessage(replyMessage,
        replyMessage.getAllRecipients());
} finally {
    t.close();
}
System.out.println("message replied successfully ....");

// close the store and folder objects
folder.close(false);
store.close();

} else if ("n".equals(ans)) {
    break;
}

} //end of for loop

} else {
    System.out.println("There is no msg....");
}

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

```
}
```

```
}
```

Puede activar la depuración descomentando la instrucción `session.setDebug (true);`

Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase `ReplyToEmail.java` en el directorio: **/ home / manisha / JavaMailAPIExercise** . Necesitaríamos los jars `javax.mail.jar` y `activation.jar` en el classpath. Ejecute el siguiente comando para compilar la clase (ambos jar se colocan en el directorio `/ home / manisha /`) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: ReplyToEmail.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: ReplyToEmail
```

Verificar salida

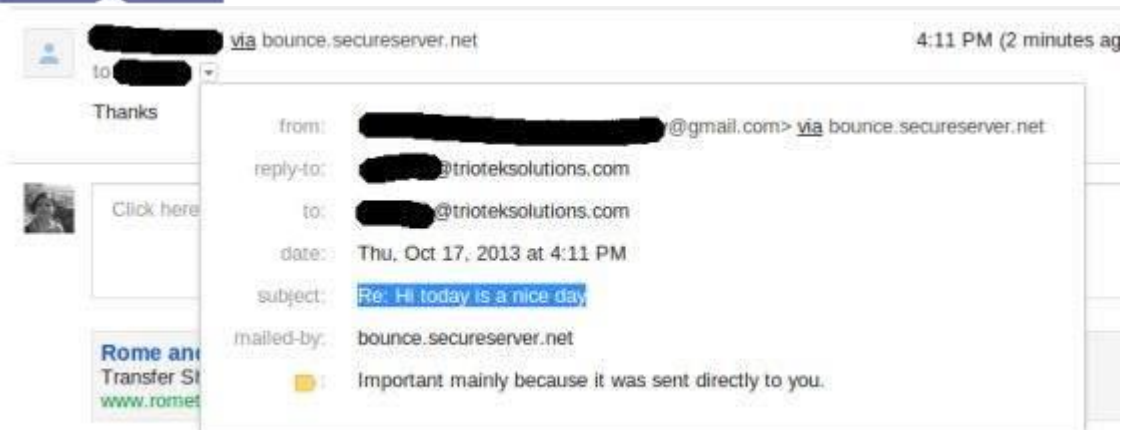
Debería ver el siguiente mensaje en la consola de comandos:

```
From: ABC <abc@gmail.com>
Reply-to: abc@trioteksolutions.com
To: XYZ <xyz@gmail.com>
Subject: Hi today is a nice day
Sent: Thu Oct 17 15:58:37 IST 2013
Do you want to reply [y/n] : y
message replied successfully ....
```

Compruebe la bandeja de entrada a la que se envió el correo. En nuestro caso, el mensaje recibido tiene el siguiente aspecto:



**INSTITUTO TÉCNICO
DE ESTUDIOS
PROFESIONALES**



Reenvío de correos electrónicos

En este capítulo veremos cómo reenviar un correo electrónico utilizando la API de JavaMail. Los pasos básicos seguidos en el programa a continuación son:

- Obtenga el objeto Session con los detalles del servidor POP y SMTP en las propiedades. Necesitaríamos detalles de POP para recuperar mensajes y detalles de SMTP para enviar mensajes.
- Cree un objeto de tienda POP3 y conéctese a store.
- Cree el objeto folder y abra la carpeta correspondiente en su buzón.
- Recuperar mensajes.
- Repita los mensajes y escriba "Y" o "y" si desea reenviar.
- Obtenga toda la información (Para, De, Asunto, Contenido) del mensaje.
- Construya el mensaje de reenvío trabajando con las partes que componen un mensaje. La primera parte sería el texto del mensaje y una segunda parte sería el mensaje a reenviar. Combine los dos en varias partes. Luego, agrega el multiparte a un mensaje con la dirección adecuada y lo envía.
- Cierre los objetos Transport, folder y store respectivamente.

Aquí hemos utilizado el servidor JangoSMTP a través del cual se envían correos electrónicos a nuestra dirección de correo electrónico de destino. La configuración se explica en el capítulo configuración del entorno .

Crear clase de Java

Cree un archivo de clase java **ForwardEmail** , cuyo contenido es el siguiente:

```
package com.codigodata;

import java.io.BufferedReader;
```



```
import java.io.InputStreamReader;

import java.util.Date;

import java.util.Properties;


import javax.mail.BodyPart;

import javax.mail.Folder;

import javax.mail.Message;

import javax.mail.Multipart;

import javax.mail.PasswordAuthentication;

import javax.mail.Session;

import javax.mail.Store;

import javax.mail.Transport;

import javax.mail.internet.InternetAddress;

import javax.mail.internet.MimeBodyPart;

import javax.mail.internet.MimeMessage;

import javax.mail.internet.MimeMultipart;


public class ForwardEmail {


    public static void main(String[] args) {

        Properties properties = new Properties();

        properties.put("mail.store.protocol", "pop3");

        properties.put("mail.pop3s.host", "pop.gmail.com");

        properties.put("mail.pop3s.port", "995");

        properties.put("mail.pop3.starttls.enable", "true");

        properties.put("mail.smtp.auth", "true");

        properties.put("mail.smtp.host", "relay.jangosmtp.net");

        properties.put("mail.smtp.port", "25");

        Session session = Session.getDefaultInstance(properties);
```

```
try {  
    // session.setDebug(true);  
    // Get a Store object and connect to the current host  
    Store store = session.getStore("pop3s");  
    store.connect("pop.gmail.com", "xyz@gmail.com",  
        "*****");//change the user and password accordingly  
  
    // Create a Folder object and open the folder  
    Folder folder = store.getFolder("inbox");  
    folder.open(Folder.READ_ONLY);  
    BufferedReader reader = new BufferedReader(new InputStreamReader(  
        System.in));  
    Message[] messages = folder.getMessages();  
    if (messages.length != 0) {  
  
        for (int i = 0, n = messages.length; i < n; i++) {  
            Message message = messages[i];  
            // Get all the information from the message  
            String from = InternetAddress.toString(message.getFrom());  
            if (from != null) {  
                System.out.println("From: " + from);  
            }  
            String replyTo = InternetAddress.toString(message  
                .getReplyTo());  
            if (replyTo != null) {  
                System.out.println("Reply-to: " + replyTo);  
            }  
            String to = InternetAddress.toString(message  
                .getRecipients(Message.RecipientType.TO));
```

```
if (to != null) {  
    System.out.println("To: " + to);  
}  
  
String subject = message.getSubject();  
if (subject != null) {  
    System.out.println("Subject: " + subject);  
}  
  
Date sent = message.getSentDate();  
if (sent != null) {  
    System.out.println("Sent: " + sent);  
}  
  
System.out.print("Do you want to reply [y/n] : ");  
String ans = reader.readLine();  
if ("Y".equals(ans) || "y".equals(ans)) {  
    Message forward = new MimeMessage(session);  
  
    // Fill in header  
    forward.setRecipients(Message.RecipientType.TO,  
        InternetAddress.parse(from));  
    forward.setSubject("Fwd: " + message.getSubject());  
    forward.setFrom(new InternetAddress(to));  
  
    // Create the message part  
    MimeBodyPart messageBodyPart = new MimeBodyPart();  
  
    // Create a multipart message  
    Multipart multipart = new MimeMultipart();  
  
    // set content  
    messageBodyPart.setContent(message, "message/rfc822");  
  
    // Add part to multi part
```

```
multipart.addBodyPart(messageBodyPart);

// Associate multi-part with message
forward.setContent(multipart);
forward.saveChanges();

// Send the message by authenticating the SMTP server
// Create a Transport instance and call the sendMessage
Transport t = session.getTransport("smtp");
try {
    //connect to the SMTP server using transport instance
    //change the user and password accordingly
    t.connect("abc", "*****");
    t.sendMessage(forward, forward.getAllRecipients());
} finally {
    t.close();
}

System.out.println("message forwarded successfully....");

// close the store and folder objects
folder.close(false);
store.close();
} // end if

} // end for
} // end if
} catch (Exception e) {
    e.printStackTrace();
}
```

```
}  
  
}
```

Puede activar la depuración descomentando la instrucción `session.setDebug (true);`

Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase `ForwardEmail.java` en el directorio: `/ home / manisha / JavaMailAPIExercise` . Necesitaríamos los jars `javax.mail.jar` y `activation.jar` en el classpath. Ejecute el siguiente comando para compilar la clase (ambos frascos se colocan en el directorio `/ home / manisha /`) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: ForwardEmail.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: ForwardEmail
```


Verificar salida

Debería ver el siguiente mensaje en la consola de comandos:

```
From: ABC <abc@gmail.com>  
Reply-to: abc@trioteksolutions.com  
To: XYZ <xyz@gmail.com>  
Subject: Hi today is a nice day  
Sent: Thu Oct 17 15:58:37 IST 2013  
Do you want to reply [y/n] : y  
message forwarded successfully....
```

Compruebe la bandeja de entrada a la que se envió el correo. En nuestro caso, el mensaje reenviado se vería a continuación:



Hi today is a nice day  Inbox x



Eliminar correos electrónicos

En este capítulo veremos cómo eliminar un correo electrónico usando la API de JavaMail. Eliminar mensajes implica trabajar con los indicadores asociados con los mensajes. Hay diferentes banderas para diferentes estados, algunos definidos por el sistema y otros definidos por el usuario. Las banderas predefinidas se definen en la clase interna `Flags.Flag` y se enumeran a continuación:

- `Flags.Flag.ANSWERED`
- `Flags.Flag.DELETED`
- `Flags.Flag.DRAFT`
- `Flags.Flag.FLAGGED`
- `Flags.Flag.RECENT`
- `Flags.Flag.SEEN`
- `Flags.Flag.USER`

El protocolo POP solo admite la eliminación de mensajes.

Los pasos básicos seguidos en el programa de eliminación son:

- Obtenga el objeto Session con los detalles del servidor POP y SMTP en las propiedades. Necesitaríamos detalles de POP para recuperar mensajes y detalles de SMTP para enviar mensajes.
- Cree un objeto store POP3 y conéctese a store.
- Cree el objeto folder y abra folder correspondiente en su buzón en modo READ_WRITE.
- Recupera mensajes de la carpeta de la bandeja de entrada.
- Repita los mensajes y escriba "Y" o "y" si desea eliminar el mensaje invocando el método setFlag (Flags.Flag.DELETED, true) en el objeto Message.
- Los mensajes marcados como ELIMINADOS no se eliminan realmente, hasta que llamamos al método expunge () en el objeto Folder, o cerramos la carpeta con el expunge establecido en true.
- Cierre el objeto store.

Crear clase de Java

Cree un archivo de clase java **ForwardEmail** , cuyo contenido es el siguiente:

```
package com.codigodata;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Properties;

import javax.mail.Flags;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.NoSuchProviderException;
import javax.mail.Session;
import javax.mail.Store;
```

```
public class DeleteEmail {

    public static void delete(String pop3Host, String storeType, String user,
        String password)
    {
        try
        {
            // get the session object
            Properties properties = new Properties();
            properties.put("mail.store.protocol", "pop3");
            properties.put("mail.pop3s.host", pop3Host);
            properties.put("mail.pop3s.port", "995");
            properties.put("mail.pop3.starttls.enable", "true");
            Session emailSession = Session.getDefaultInstance(properties);
            // emailSession.setDebug(true);

            // create the POP3 store object and connect with the pop server
            Store store = emailSessiongetStore("pop3s");

            store.connect(pop3Host, user, password);

            // create the folder object and open it
            Folder emailFolder = store.getFolder("INBOX");
            emailFolder.open(Folder.READ_WRITE);

            BufferedReader reader = new BufferedReader(new InputStreamReader(
                System.in));

            // retrieve the messages from the folder in an array and print it
```

```
Message[] messages = emailFolder.getMessages();

System.out.println("messages.length---" + messages.length);

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

    Message message = messages[i];

    System.out.println("-----");

    System.out.println("Email Number " + (i + 1));

    System.out.println("Subject: " + message.getSubject());

    System.out.println("From: " + message.getFrom()[0]);


    String subject = message.getSubject();

    System.out.print("Do you want to delete this message [y/n] ? ");

    String ans = reader.readLine();

    if ("Y".equals(ans) || "y".equals(ans)) {

        // set the DELETE flag to true

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

        System.out.println("Marked DELETE for message: " + subject);

    } else if ("n".equals(ans)) {

        break;

    }

}

// expunges the folder to remove messages which are marked deleted
emailFolder.close(true);

store.close();


} catch (NoSuchProviderException e) {

    e.printStackTrace();

} catch (MessagingException e) {

    e.printStackTrace();

} catch (IOException io) {
```

```
        io.printStackTrace();
    }
}

public static void main(String[] args) {

    String host = "pop.gmail.com";// change accordingly
    String mailStoreType = "pop3";
    String username = "abc@gmail.com";// change accordingly
    String password = "*****";// change accordingly

    delete(host, mailStoreType, username, password);

}

}
```

Puede activar la depuración descomentando la declaración `emailSession.setDebug(true);`

Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase `DeleteEmail.java` en el directorio: / **home** / **manisha** / **JavaMailAPIExercise** . Necesitaríamos los `jars javax.mail.jar` y `activation.jar` en el classpath. Ejecute el siguiente comando para compilar la clase (ambos jar se colocan en el directorio / home / manisha /) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: DeleteEmail.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: DeleteEmail
```

Verificar salida

Debería ver el siguiente mensaje en la consola de comandos:

```
messages.length---1
```

Email Number 1

Subject: Testing

From: ABC <abc@gmail.com>

Do you want to delete this message [y/n] ? y

Marked DELETE for message: Testing

Servidor SMTP de Gmail

En todos los capítulos anteriores usamos el servidor JangoSMTP para enviar correos electrónicos. En este capítulo, aprenderemos sobre el servidor SMTP proporcionado por Gmail. Gmail (entre otros) ofrece el uso de su servidor SMTP público de forma gratuita.

Los detalles del servidor SMTP de Gmail se pueden encontrar [aquí](#) . Como puede ver en los detalles, podemos usar una conexión TLS o SSL para enviar correo electrónico a través del servidor SMTP de Gmail.

El procedimiento para enviar correo electrónico utilizando el servidor SMTP de Gmail es similar al que se explica en el capítulo Envíos de correos electrónicos , excepto que cambiaríamos el servidor host. Como requisito previo, la dirección de correo electrónico del remitente debe ser una cuenta de Gmail activa. Probemos con un ejemplo.

Crear clase de Java

Cree un archivo Java **SendEmailUsingGMailSMTP** , cuyo contenido es el siguiente:

```
package com.codigodata;

import java.util.Properties;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
```

```
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class SendEmailUsingGMailSMTP {
    public static void main(String[] args) {
        // Recipient's email ID needs to be mentioned.
        String to = "xyz@gmail.com";//change accordingly

        // Sender's email ID needs to be mentioned
        String from = "abc@gmail.com";//change accordingly
        final String username = "abc";//change accordingly
        final String password = "*****";//change accordingly

        // Assuming you are sending email through relay.jangosmtp.net
        String host = "smtp.gmail.com";

        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", host);
        props.put("mail.smtp.port", "587");

        // Get the Session object.
        Session session = Session.getInstance(props,
        new javax.mail.Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username, password);
            }
        });
    }
}
```

```
try {  
    // Create a default MimeMessage object.  
    Message message = new MimeMessage(session);  
  
    // Set From: header field of the header.  
    message.setFrom(new InternetAddress(from));  
  
    // Set To: header field of the header.  
    message.setRecipients(Message.RecipientType.TO,  
        InternetAddress.parse(to));  
  
    // Set Subject: header field  
    message.setSubject("Testing Subject");  
  
    // Now set the actual message  
    message.setText("Hello, this is sample for to check send "  
        + "email using JavaMailAPI ");  
  
    // Send message  
    Transport.send(message);  
  
    System.out.println("Sent message successfully....");  
  
} catch (MessagingException e) {  
    throw new RuntimeException(e);  
}  
}
```


Aquí, el host está configurado como *smtp.gmail.com* y el puerto está configurado como 587. Aquí hemos habilitado la conexión TLS.

Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase `SendEmailUsingGMailSMTP.java` en el directorio: **/ home / manisha / JavaMailAPIExercise**. Necesitaríamos los jars *javax.mail.jar* y *activation.jar* en el classpath. Ejecute el siguiente comando para compilar la clase (ambos jar se colocan en el directorio / home / manisha /) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar:  
SendEmailUsingGMailSMTP.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar:  
SendEmailUsingGMailSMTP
```

Verificar salida

Debería ver el siguiente mensaje en la consola de comandos:

```
Sent message successfully....
```

Gestión de carpetas

Hasta ahora, hemos trabajado en nuestros capítulos anteriores principalmente con la carpeta INBOX. Esta es la carpeta predeterminada en la que reside la mayor parte del correo. Algunos sistemas pueden llamarlo como INBOX y otros pueden llamarlo por algún otro nombre. Pero siempre puede acceder a él desde la API de JavaMail usando el nombre INBOX.

La API de JavaMail representa carpetas como instancias de la clase *Carpeta* abstracta:

```
public abstract class Folder extends Object
```

Esta clase declara métodos para solicitar carpetas con nombre de los servidores, eliminar mensajes de carpetas, buscar mensajes particulares en carpetas, enumerar los mensajes en una carpeta, etc.

Abrir una carpeta

No podemos crear una carpeta directamente ya que el único constructor de la clase *Carpeta* está *protegido*. Podemos obtener una *carpeta* de:

- una session
- una store

- u otra folder

Todas las clases anteriores tienen un método `getFolder()` similar con una firma similar:

```
public abstract Folder getFolder(String name) throws MessagingException
```

Algunos de los métodos que ayudan a obtener el objeto *Folder* son:

Método	Descripción
boolean exists()	Comprueba si la carpeta realmente existe. Utilice este método antes de obtener el objeto Carpeta.
abstract void open(int mode)	Cuando obtienes una <i>carpeta</i> , se cierra. Utilice este método para abrirlo. <i>el modo</i> puede ser <code>Folder.READ_ONLY</code> o <code>Folder.READ_WRITE</code> .
abstract boolean isOpen()	Este método devuelve <i>verdadero</i> si la carpeta está abierta, <i>falso</i> si está cerrada
abstract void close(boolean expunge)	Cierra la carpeta. Si el argumento de <i>eliminación</i> es <i>verdadero</i> , todos los mensajes eliminados en la carpeta se eliminan del archivo real en el servidor. De lo contrario, simplemente se marcan como <i>eliminados</i> , pero los mensajes aún se pueden recuperar.

Información básica de la carpeta

A continuación se muestran algunos de los métodos de la clase Carpeta que devuelven información básica sobre una carpeta:

Método	Descripción
abstract String getName()	Devuelve el nombre de la carpeta, como "TutorialsPoint Mail"
abstract String getFullName()	Devuelve el nombre jerárquico completo de la raíz, como "libros / Manisha / TutorialsPoint Mail".
URLName getURLName()	Devuelve un <code>URLName</code> que represente esta carpeta.
abstract Folder getParent()	Devuelve el nombre de la carpeta que contiene esta carpeta, es decir, la carpeta

	principal. Por ejemplo, "Manisha" del ejemplo anterior "TutorialsPoint Mail".
abstract int <i>getType()</i>	Devuelve un int que indica si la carpeta puede contener mensajes y / u otras carpetas.
int <i>getMode ()</i>	Devuelve una de las dos constantes con nombre Folder.READ_ONLY o Folder.READ_WRITE o -1 cuando se desconoce el modo.
Store <i>getStore()</i>	Devuelve el objeto Store del que se recuperó esta carpeta.
abstract char <i>getSeparator()</i>	Devuelve el carácter delimitador que separa el nombre de ruta de esta carpeta de los nombres de las subcarpetas inmediatas.

Administrar carpeta

A continuación, se muestran algunos de los métodos que ayudan a administrar la carpeta:

Método	Descripción
abstract boolean <i>create(int type)</i>	Esto crea una nueva carpeta en la Tienda de esta carpeta. Donde el <i>tipo</i> sería: Carpeta.HOLDS_MESSAGES o Carpeta.HOLDS_FOLDERS. Devuelve <i>verdadero</i> si la carpeta se creó con éxito; de lo contrario, devuelve <i>falso</i> .
abstract boolean <i>delete(boolean recurse)</i>	Esto elimina la carpeta solo si la carpeta está cerrada. De lo contrario, arroja una <i>IllegalStateException</i> . Si <i>recurse</i> es <i>verdadero</i> , entonces se eliminan las subcarpetas.
abstract boolean <i>renameTo(Folder f)</i>	Esto cambia el nombre de esta carpeta. Se debe cerrar una carpeta para que se le cambie el nombre. De lo contrario, se lanza una <i>IllegalStateException</i> .

Administrar mensajes en carpetas

A continuación, se muestran algunos de los métodos que ayudan a administrar los mensajes en la carpeta:

Método	Descripción
abstract void <i>appendMessages</i> (Mensaje [] mensajes)	Como su nombre lo indica, los mensajes de la matriz se colocan al final de esta carpeta.
void <i>copyMessages</i> (Mensaje [] mensajes, destino de carpeta)	Esto copia los mensajes de esta carpeta en una carpeta específica dada como argumento.
abstract Message[] <i>expunge</i>()	Para eliminar un mensaje de una carpeta, establezca su indicador <code>Flags.Flag.DELETED</code> en verdadero. Para eliminar físicamente los mensajes eliminados de una carpeta, debe llamar a este método.

Listado del contenido de una carpeta

Hay cuatro métodos para enumerar las carpetas que contiene una carpeta:

Método	Descripción
Folder[] <i>list</i>()	Esto devuelve una matriz que enumera las carpetas que contiene esta carpeta.
Folder[] <i>listSubscribed</i>()	Esto devuelve una matriz que enumera todas las carpetas suscritas que contiene esta carpeta.
abstract Folder[] <i>list</i>(String pattern)	Es similar al método <i>list</i> () excepto que le permite especificar un patrón. El patrón es una cadena que da el nombre de las carpetas que coinciden.
Folder[] <i>listSubscribed</i>(String pattern)	Esto es similar al método <i>listSubscriptions</i> () excepto que le permite especificar un patrón. El

	patrón es una cadena que da el nombre de las carpetas que coinciden.
--	--

Comprobación de correo

Método	Descripción
abstract int <i>getMessageCount()</i>	Este método se puede invocar en una carpeta abierta o cerrada. Sin embargo, en el caso de una carpeta cerrada, este método puede (o no) devolver -1 para indicar que el número exacto de mensajes no está disponible fácilmente.
abstract boolean <i>hasNewMessages()</i>	Esto devuelve <i>verdadero</i> si se han agregado nuevos mensajes a la carpeta desde que se abrió por última vez.
int <i>getNewMessageCount ()</i>	Devuelve el recuento de mensajes nuevos comprobando los mensajes en la carpeta cuyo indicador RECIENTE está establecido.
int <i>getUnreadMessageCount ()</i>	Esto se puede invocar en una carpeta abierta o cerrada. Sin embargo, en el caso de una carpeta cerrada, puede devolver -1 para indicar que la respuesta real sería demasiado cara de obtener.

Obtener mensajes de carpetas

La clase Carpeta proporciona cuatro métodos para recuperar mensajes de carpetas abiertas:

Método	Descripción
abstract Message <i>getMessage(int messageNumber)</i>	Esto devuelve el enésimo mensaje de la carpeta. El primer mensaje de la carpeta es el número 1.
Message[] <i>getMessages()</i>	Esto devuelve una matriz de objetos de <i>mensaje</i> que representan todos los mensajes de esta carpeta.
Message[] <i>getMessages(int start, int end)</i>	Esto devuelve una matriz de objetos <i>Message</i> de la carpeta, comenzando con el

	inicio y terminando con el final, inclusive.
Message[] <i>getMessages</i>(int[] messageNumbers)	Esto devuelve una matriz que contiene solo aquellos mensajes identificados específicamente por número en la matriz <i>messageNumbers</i> .
void <i>fetch</i>(Message[] messages, FetchProfile fp)	Busque previamente los elementos especificados en FetchProfile para los mensajes dados. El argumento FetchProfile especifica qué encabezados de los mensajes se van a buscar previamente.

Carpetas de búsqueda

Si el servidor admite la búsqueda (como lo hacen muchos servidores IMAP y la mayoría de los servidores POP no), es fácil buscar en una carpeta los mensajes que cumplan ciertos criterios. Los criterios están codificados en objetos SearchTerm. A continuación se muestran los dos métodos de búsqueda:

Método	Descripción
Message[] <i>search</i>(SearchTerm term)	Busque en esta carpeta los mensajes que coincidan con el criterio de búsqueda especificado. Devuelve una matriz que contiene los mensajes coincidentes. Devuelve una matriz vacía si no se encontraron coincidencias.
Message[] <i>search</i>(SearchTerm term, Message[] messages)	Busque en la matriz dada de mensajes aquellos que coincidan con el criterio de búsqueda especificado. Devuelve una matriz que contiene los mensajes coincidentes. Devuelve una matriz vacía si no se encontraron

coincidencias. Los objetos de mensaje especificados deben pertenecer a esta carpeta.

Banderas

La modificación de banderas es útil cuando necesita cambiar las banderas para todo el conjunto de mensajes en una carpeta. A continuación se muestran los métodos proporcionados en la clase Carpeta:

Método	Descripción
<code>void setFlags(Message[] messages, Flags flag, boolean value)</code>	Establece las banderas especificadas en los mensajes especificados en la matriz.
<code>void setFlags(int start, int end, Flags flag, boolean value)</code>	Establece las banderas especificadas en los mensajes numerados de principio a fin, ambos de principio a fin inclusive.
<code>void setFlags(int[] messageNumbers, Flags flag, boolean value)</code>	Establece las banderas especificadas en los mensajes cuyos números de mensaje están en la matriz.
<code>abstract Flags getPermanentFlags()</code>	Devuelve las banderas que admite esta carpeta para todos los mensajes.

Gestión de cuotas

Una cuota en JavaMail es un número o cantidad limitada o fija de mensajes en una tienda de correo electrónico. Cada solicitud de servicio de correo cuenta para la cuota de llamadas a la API de JavaMail. Un servicio de correo electrónico puede aplicar el siguiente criterio de cuota:

- Tamaño máximo de los mensajes de correo salientes, incluidos los archivos adjuntos.
- Tamaño máximo de los mensajes de correo entrantes, incluidos los archivos adjuntos.
- Tamaño máximo de mensaje cuando un administrador es un destinatario

Para la gestión de cuotas, JavaMail tiene las siguientes clases:

Clase	Descripción
public class Quota	Esta clase representa un conjunto de cuotas para una raíz de cuota determinada. Cada raíz de cuota tiene un conjunto de recursos, representados por la clase <code>Quota.Resource</code> . Cada recurso tiene un nombre (por ejemplo, "ALMACENAMIENTO"), un uso actual y un límite de uso. Esto solo tiene un método <code>setResourceLimit</code> (<i>nombre de cadena, límite largo</i>).
public static class Quota.Resource	Representa un recurso individual en una raíz de cuota.
public interface QuotaAwareStore	Una interfaz implementada por tiendas que admiten cuotas. Los métodos <code>getQuota</code> y <code>setQuota</code> admiten el modelo de cuota definido por la extensión IMAP QUOTA. <i>GmailSSLStore</i> , <i>GmailStore</i> , <i>IMAPSSLStore</i> , <i>IMAPStore</i> son las clases de implementación conocidas de esta interfaz.

Veamos un ejemplo en las siguientes secciones que verifica el nombre de almacenamiento de correo, el límite y su uso.

Crear clase de Java

Cree un archivo de clase java **QuotaExample**, cuyo contenido es el siguiente:

```
package com.codigodata;

import java.util.Properties;
```



```
import javax.mail.Quota;

import javax.mail.Session;

import javax.mail.Store;


import com.sun.mail.imap.IMAPStore;


public class QuotaExample
{
    public static void main(String[] args)
    {
        try
        {
            Properties properties = new Properties();
            properties.put("mail.store.protocol", "imaps");
            properties.put("mail.imaps.port", "993");
            properties.put("mail.imaps.starttls.enable", "true");

            Session emailSession = Session.getDefaultInstance(properties);

            // emailSession.setDebug(true);


            // create the IMAP3 store object and connect with the pop server
            Store store = emailSession.getStore("imaps");


            //change the user and password accordingly
            store.connect("imap.gmail.com", "abc@gmail.com", "*****");

            IMAPStore imapStore = (IMAPStore) store;

            System.out.println("imapStore ---" + imapStore);


            //get quota
```

```
Quota[] quotas = imapStore.getQuota("INBOX");

//Iterate through the Quotas
for (Quota quota : quotas) {

    System.out.println(String.format("quotaRoot:%s",
        quota.quotaRoot));

    //Iterate through the Quota Resource
    for (Quota.Resource resource : quota.resources) {

        System.out.println(String.format(
            "name:%s', limit:%s', usage:%s'", resource.name,
            resource.limit, resource.usage));

    }

}

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

}
```

Aquí está la conexión al servicio de Gmail a través del servidor IMAP (imap.gmail.com), ya que IMAPStore implementa QuotaAwareStore. Una vez que obtenga el objeto Store, busque la matriz Quota, repita e imprima la información relevante.

Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase QuotaExample.java en el directorio: / **home** / **manisha** / **JavaMailAPIExercise** . Necesitaríamos los jars *javax.mail.jar* y *activation.jar* en el classpath. Ejecute el siguiente comando para compilar la clase (ambos jar se colocan en el directorio / home / manisha /) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar:
QuotaExample.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: QuotaExample
```

Verificar salida

Debería ver un mensaje similar en la consola de comandos:

```
imapStore --- imaps: //abc%40gmail.com@imap.gmail.com
```

```
quotaRoot: "
```

```
nombre: 'ALMACENAMIENTO', límite: '15728640', uso: '513'
```

Mensajes rebotados

Un mensaje puede rebotarse por varias razones. Este problema se analiza en profundidad en [rfc1211](#) . Solo un servidor puede determinar la existencia de un buzón o nombre de usuario en particular. Cuando el servidor detecta un error, devolverá un mensaje indicando el motivo del error al remitente del mensaje original.

Hay muchos estándares de Internet que cubren las notificaciones de estado de entrega, pero una gran cantidad de servidores no son compatibles con estos nuevos estándares, sino que utilizan técnicas ad hoc para devolver dichos mensajes de error. Por lo tanto, resulta muy difícil correlacionar el mensaje *devuelto* con el mensaje original que causó el problema.

JavaMail incluye soporte para analizar notificaciones de estado de entrega. Hay una serie de técnicas y heurísticas para abordar este problema. Una de las técnicas son las rutas de retorno de envoltente variable. Puede establecer la ruta de retorno en el sobre como se muestra en el siguiente ejemplo. Esta es la dirección a la que se envían los correos electrónicos de devolución. Es posible que desee establecer esto en una dirección genérica, diferente del encabezado From:, para que pueda procesar rebotes remotos. Esto se hace estableciendo la propiedad *mail.smtp.from* en JavaMail.

Crear clase de Java

Cree un archivo de clase Java **SendEmail** , cuyo contenido es el siguiente:

```
import java.util.Properties;

import javax.mail.Message;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
```

```
import javax.mail.internet.MimeMessage;

public class SendEmail {

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

        String smtpServer = "smtp.gmail.com";

        int port = 587;

        final String userid = "youraddress";//change accordingly
        final String password = "*****";//change accordingly
        String contentType = "text/html";

        String subject = "test: bounce an email to a different address " +
                        "from the sender";

        String from = "youraddress@gmail.com";

        String to = "bouncer@fauxmail.com";//some invalid address
        String bounceAddr = "toaddress@gmail.com";//change accordingly
        String body = "Test: get message to bounce to a separate email address";

        Properties props = new Properties();

        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", smtpServer);
        props.put("mail.smtp.port", "587");
        props.put("mail.transport.protocol", "smtp");
        props.put("mail.smtp.from", bounceAddr);

        Session mailSession = Session.getInstance(props,
            new javax.mail.Authenticator() {
                protected PasswordAuthentication getPasswordAuthentication() {
                    return new PasswordAuthentication(userid, password);
                }
            }
        );
```

```
}  
});  
  
MimeMessage message = new MimeMessage(mailSession);  
message.addFrom(InternetAddress.parse(from));  
message.setRecipients(Message.RecipientType.TO, to);  
message.setSubject(subject);  
message.setContent(body, contentType);  
  
Transport transport = mailSession.getTransport();  
try {  
    System.out.println("Sending ....");  
    transport.connect(smtpServer, port, userid, password);  
    transport.sendMessage(message,  
        message.getRecipients(Message.RecipientType.TO));  
    System.out.println("Sending done ...");  
} catch (Exception e) {  
    System.err.println("Error Sending: ");  
    e.printStackTrace();  
  
}  
transport.close();  
} // end function main()  
}
```

Aquí podemos ver que la propiedad *mail.smtp.from* se fija diferente de la de dirección.

Compilar y ejecutar

Ahora que nuestra clase está lista, compilemos la clase anterior. He guardado la clase `SendEmail.java` en el directorio: / **home** / **manisha** / **JavaMailAPIExercise** . Necesitaríamos los

jars *javax.mail.jar* y *activation.jar* en el classpath. Ejecute el siguiente comando para compilar la clase (ambos jar se colocan en el directorio / home / manisha /) desde el símbolo del sistema:

```
javac -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: SendEmail.java
```

Ahora que la clase está compilada, ejecute el siguiente comando para ejecutar:

```
java -cp /home/manisha/activation.jar:/home/manisha/javax.mail.jar: SendEmail
```

Verificar salida

Debería ver el siguiente mensaje en la consola de comandos:

Sending

Sending done ...

Servidores SMTP

SMTP es un acrónimo de **Protocolo simple de transferencia de correo** . Es un estándar de Internet para la transmisión de correo electrónico (correo electrónico) a través de redes de Protocolo de Internet (IP). SMTP utiliza el puerto TCP 25. Las conexiones SMTP protegidas por SSL se conocen por la abreviatura SMTPS, aunque SMTPS no es un protocolo en sí mismo.

La API de JavaMail tiene el paquete **com.sun.mail.smtp** que actúa como proveedor de protocolo SMTP para acceder a un servidor SMTP. La siguiente tabla enumera las clases incluidas en este paquete:

Clase	Descripción
SMTPMessage	Esta clase es una especialización de la clase <code>MimeMessage</code> que le permite especificar varias opciones y parámetros de SMTP que se utilizarán cuando este mensaje se envíe a través de SMTP.
SMTPSSLTransport	Esta clase implementa la clase abstracta de transporte utilizando SMTP sobre SSL para el envío y transporte de mensajes.
SMTPTransport	Esta clase implementa la clase abstracta de transporte utilizando SMTP para el envío y transporte de mensajes.

La siguiente tabla enumera las excepciones lanzadas:

Excepción	Descripción
SMTPAddressFailedException	Esta excepción se lanza cuando el mensaje no se puede enviar.
SMTPAddressSucceededException	Esta excepción se encadena a una <code>SendFailedException</code> cuando la propiedad <code>mail.smtp.reportsuccess</code> es verdadera.
SMTPSenderFailedException	Esta excepción se lanza cuando el mensaje no se puede enviar.
SMTPSendFailedException	Esta excepción se produce cuando el mensaje no se puede enviar. La excepción incluye la dirección del remitente, que el servidor de correo rechazó.

El proveedor **com.sun.mail.smtp** usa la autenticación SMTP opcionalmente. Para usar la autenticación SMTP, deberá configurar la propiedad `mail.smtp.auth` o proporcionar al Transporte SMTP un nombre de usuario y una contraseña cuando se conecte al servidor SMTP. Puede hacer esto usando uno de los siguientes enfoques:

- Proporcione un objeto Autenticador al crear su sesión de correo y proporcione la información de nombre de usuario y contraseña durante la devolución de llamada del Autenticador. La propiedad `mail.smtp.user` se puede configurar para proporcionar un nombre de usuario predeterminado para la devolución de llamada, pero la contraseña aún deberá proporcionarse explícitamente. Este enfoque le permite utilizar el método de envío de transporte estático para enviar mensajes. Por ejemplo:

```
Transport.send(message);
```

- Llame al método de conexión de transporte explícitamente con argumentos de nombre de usuario y contraseña. Por ejemplo:

```
Transport tr = session.getTransport("smtp");
tr.connect(smtpHost, username, password);
msg.saveChanges();
tr.sendMessage(msg, msg.getAllRecipients());
tr.close();
```

El proveedor del protocolo SMTP admite las siguientes propiedades, que se pueden configurar en el objeto Sesión JavaMail. Las propiedades siempre se establecen como cadenas. Por ejemplo:

```
props.put("mail.smtp.port", "587");
```

Servidores IMAP

IMAP es el acrónimo de **Protocolo de acceso a mensajes de Internet**. Es un protocolo de Internet de capa de aplicación que permite a un cliente de correo electrónico acceder al correo electrónico en un servidor de correo remoto. Un servidor IMAP normalmente escucha en el conocido puerto 143. IMAP sobre SSL (IMAPS) se asigna al puerto número 993.

IMAP admite modos de funcionamiento en línea y fuera de línea. Los clientes de correo electrónico que utilizan IMAP generalmente dejan mensajes en el servidor hasta que el usuario los elimina explícitamente.

El paquete **com.sun.mail.imap** es un proveedor de protocolo IMAP para la API de JavaMail que proporciona acceso a un almacén de mensajes IMAP. La siguiente tabla enumera la interfaz y las clases de este proveedor:

Clase / Interfaz	Descripción
IMAPFolder.ProtocolCommand	Esta es una <i>interfaz</i> simple para comandos de protocolo IMAP definidos por el usuario.
ACL	Esta es una clase. Una entrada de la lista de control de acceso para un identificador de autenticación particular (usuario o grupo).
IMAPFolder	Esta clase implementa una carpeta IMAP.
IMAPFolder.FetchProfileItem	Esta es una clase para buscar encabezados.
IMAPMessage	Esta clase implementa un objeto ReadableMime.
IMAPMessage.FetchProfileCondition	Esta clase implementa la prueba a realizar en cada mensaje de la carpeta.
IMAPSSLStore	Esta clase proporciona acceso a un almacén de mensajes IMAP a través de SSL.
IMAPStore	Esta clase proporciona acceso a un almacén de mensajes IMAP.

Rights	Esta clase representa el conjunto de derechos para un identificador de autenticación (por ejemplo, un usuario o un grupo).
Rights.Right	Esta clase interna representa un derecho individual.
SortTerm	Un criterio de clasificación particular, según lo definido por RFC 5256.

Algunos puntos que deben tenerse en cuenta por encima de este proveedor:

- Este proveedor admite los protocolos IMAP4 e IMAP4rev1.
- Un IMAPStore conectado mantiene un grupo de objetos de protocolo IMAP para usar en la comunicación con el servidor IMAP. A medida que se abren carpetas y se necesitan nuevos objetos de protocolo IMAP, IMAPStore los proporcionará desde el grupo de conexiones o los creará si no hay ninguno disponible. Cuando se cierra una carpeta, su objeto de protocolo IMAP se devuelve al grupo de conexiones si el grupo.
- El objeto IMAPStore conectado puede mantener o no un objeto de protocolo IMAP separado que proporciona a la tienda una conexión dedicada al servidor IMAP.

Servidores POP3

El Protocolo de oficina de correos (POP) es un protocolo estándar de Internet de nivel de aplicación utilizado por los clientes de correo electrónico locales para recuperar el correo electrónico de un servidor remoto a través de una conexión TCP / IP. POP admite requisitos simples de descarga y eliminación para acceder a buzones de correo remotos. Un servidor POP3 escucha en el conocido puerto 110.

El paquete **com.sun.mail.pop3** es un proveedor de protocolo POP3 para la API de JavaMail que proporciona acceso a un almacén de mensajes POP3. La siguiente tabla enumera las clases de este paquete:

Nombre	Descripción
POP3Folder	Una carpeta POP3 (solo puede ser "INBOX").
POP3Message	Un mensaje POP3.
POP3SSLStore	Un almacén de mensajes POP3 que utiliza SSL.



POP3Store

Una tienda de mensajes POP3.

Algunos puntos que deben tenerse en cuenta por encima de este proveedor:

- El proveedor de POP3 admite solo una carpeta llamada **INBOX** . Debido a las limitaciones del protocolo POP3, muchas de las capacidades de la API de JavaMail como notificación de eventos, administración de carpetas, administración de banderas, etc. no están permitidas.
- Se accede al proveedor de POP3 a través de las API de JavaMail mediante el nombre de protocolo *pop3* o una URL con el formato *pop3:// usuario: contraseña @ host: puerto / INBOX "* .
- POP3 no admite banderas permanentes. Por ejemplo, el indicador *Flags.Flag.RECENT* nunca se establecerá para mensajes POP3. Depende de la aplicación determinar qué mensajes de un buzón POP3 son *nuevos* .
- POP3 no admite el método *Folder.expunge ()*. Para eliminar y eliminar mensajes, configure el indicador *Flags.Flag.DELETED* en los mensajes y cierre la carpeta con el método *Folder.close (true)*.
- POP3 no proporciona una *fecha de recepción* , por lo que el método *getReceivedDate* devolverá un valor nulo.
- Cuando se accede a los encabezados de un mensaje POP3, el proveedor de POP3 utiliza el comando TOP para recuperar todos los encabezados, que luego se almacenan en caché.
- Cuando se accede al contenido de un mensaje POP3, el proveedor de POP3 utiliza el comando RETR para recuperar el mensaje completo.
- El método *POP3Message.invalidate* se puede utilizar para invalidar los datos almacenados en caché sin cerrar la carpeta.