

# JavaMail 简易教程

Alex Zhu 2011.3 Version 0.1 Beta

JavaMail API 简介	- 2 -
了解相关协议	- 2 -
JavaMail 的安装	- 4 -
安装 JavaBeans Activation Framework	- 4 -
使用 Java EE 企业版	- 4 -
发送邮件的例子	- 5 -
JavaMail 核心类	- 7 -
java.util.Properties 类	- 7 -
javax.mail.Session 类	- 9 -
javax.mail.Authenticator 类	- 10 -
javax.mail.Message 类	- 12 -
javax.mail.Address 类	- 15 -
javax.mail.Transport 类	- 16 -
javax.mail.Store 和 javax.mail.Folder 类	- 17 -
使用 JavaMail API	- 19 -
发送消息	- 19 -
获取消息	- 21 -
删除消息和标志	- 23 -
自我验证	- 24 -
回复消息	- 25 -
转发消息	- 26 -
操作附件	- 27 -
发送附件	- 27 -
获取附件	- 28 -
处理 HTML 消息	- 29 -
发送 HTML 消息	- 29 -
在消息中包含图片	- 29 -
用 SearchTerm 搜索	- 31 -
常见问答	- 32 -
JavaMail 的常用类速查	- 35 -
示例代码	- 38 -
Gmail 收发信	- 38 -
JavaMail 收取邮件属性配置（包括 Gmail、hotmail 等）	- 41 -
JavaMail 收取邮件 IMAP	- 44 -
JavaMail 发送邮件 [代码] MailSender.java	- 52 -
JavaMail 收取邮件 POP3	- 57 -
GmailFetch 收取 Gmail 邮件	- 62 -
Gmail Sender 发送 Gmail 邮件	- 63 -

## JavaMail API 简介

JavaMail API 是一种可选的、能用于读取、编写和发送电子消息的包（标准扩展），可使用其创建**邮件用户代理**（Mail User Agent，MUA）类型的程序，类似于 Eudora、Pine 及 Microsoft Outlook 邮件程序。其主要目的**不是**像发送邮件或其他**邮件传输代理**（Mail Transfer Agent，MTA）类型的程序那样用于传输、发送和转发消息。换句话说，用户可以与 MUA 类型的程序交互，以阅读和撰写电子邮件。MUA 依靠 MTA 处理实际的发送任务。

JavaMail API 的设计是，为收发信息提供与协议无关的访问。方式是把该 API 划分成两个部分：

- 第一个部分 基本上是如何发送和接收独立于提供程序/协议的消息。
- 第二个部分 使用特定的协议语言，如：SMTP、POP、IMAP 和 NNTP。如果要让 JavaMail API 与服务器通信，就需要为之提供协议。*Sun* 公司对特定协议提供程序有充分的介绍，用户可以免费获取。

### 了解相关协议

在学习 JavaMail API 的深层知识之前，让我们来看一看在该 API 中使用的协议，通常有 4 种人们常用的协议：

- **SMTP**
- **POP**
- **IMAP**
- **MIME**

还需要了解 NNTP 及其他一些协议。理解这些协议的基本原理有助于理解如何使用 JavaMail API。而该 API 的设计要与协议无关，所以不能克服这些基础协议的限制。如果选用的协议不支持某种功能，那么 JavaMail API 也无法在其上添加这种功能。（如在操作 POP 协议时，常常会碰到这种问题）

#### SMTP

简单邮件传输协议（Simple Mail Transfer Protocol）是用于传送电子邮件的机制。在 JavaMail API 环境中，基于 JavaMail 的程序将与公司或 Internet 服务提供商（ISP）的 SMTP 服务器通信。SMTP 服务器将会把消息转发给用作接收消息的 SMTP 服务器，最后用户可通过 POP 或 IMAP 协议获取该消息。由于支持身份验证，不需要 SMTP 服务器是一种开放的转发器，但需要确保 SMTP 服务器配置正确。JavaMail API 中没有集成用于处理诸如配置服务器以转发消息或添加/删除电子邮件帐户这一类任务的功能。参阅 RFC 821。

#### POP

POP 的含义是邮局协议（Post Office Protocol），当前的版本为 3，也称作 POP3，该协议是在 RFC 1939 中定义的。POP 是 Internet 上的经常用来接收邮件的机制。它为每个用户的每个邮箱定义支持，这是它所做的全部工作，也是大多数问题的根源。在使用 POP 协议时，人们熟悉的很多功能，如查看收到了多少新邮件消息的功能，POP 根本不支持。这些功能都内置到诸如 Eudora 或 Microsoft Outlook 之类的邮件程序中，能记住接收的上一封邮

件，以及计算有多少新邮件这类信息。因此，使用 JavaMail API 时，想获取这类信息，将需要由自己进行计算。

### IMAP

IMAP 用于接收消息的更高级的协议，在 RFC 2060 中定义。IMAP 的含义是“Internet Message Access Protocol”，当前版本是第 4 版，也称作 IMAP4。使用 IMAP 时，邮件服务器必须支持该协议。不能简单地把支持 POP 的程序用于 IMAP 协议，就指望能支持 IMAP 中的一切。如果邮件服务器支持 IMAP，那么基于 JavaMail 的程序就可访问在服务器上拥有的多个文件夹，并且这些文件夹可以被多个用户共享。

既然 IMAP 协议具有更高级的功能，那么 IMAP 应该被所有人使用？！事实不是这样！因为 IMAP 会加重邮件服务器的负荷，它需要服务器接收新消息，发送消息给请求的用户，并在多个文件夹中为每个用户维护这些消息。而这要集中备份，因而长期下去用户的文件夹会变得越来越大，当磁盘空间用光了时，每个人都会遭受损失。而使用 POP 协议时，已保存消息可以解除服务器的重负。

### MIME

MIME 的含义是 多用途的网际邮件扩充协议 (Multipurpose Internet Mail Extension)。它不是一种邮件传输协议，相反，它定义传输的内容：消息的格式、附件等。许多文档都定义了 MIME 协议，包含：RFC 822、RFC 2045、RFC 2046 和 RFC 2047。作为 JavaMail API 的用户，一般不需要担心这些格式。但是，这些格式确实存在，并为您的程序所用。

### NNTP 和其他协议

由于 JavaMail API 分开了提供程序和其他部分，可以轻松地为附加协议添加支持。Sun 公司提供第 3 方提供程序清单，这些提供程序要利用 Sun 公司不支持的少见的协议。在这份清单中，您将会看到对 NNTP（网络新闻传输协议）[新闻组]、S/MIME（安全多用途的网际邮件扩充协议）及其他协议的提供支持的第 3 方提供程序。

## JavaMail 的安装

目前 JavaMail API 最高版本 1.4.4，可以到 [Oracle 网站](#)查看最新情况。

下载地址：<http://java.sun.com/products/javamail/downloads/index.html>

解压 JavaMail 安装包： **javamail11\_4\_4.zip**

把其中的 mail.jar 文件添加到 CLASSPATH 路径下，或直接拷贝到 Tomcat 的 lib 目录下。

在 JavaMail 安装包的解压文件夹下，有 demo 演示目录可以看到许多示例程序[建议浏览]。

### 安装 JavaBeans Activation Framework

JavaMail API 的所有版本都需要 JavaBeans Activation Framework (JavaBeans 激活框架)，这种框架提供了对输入任意数据块的支持，并能相应地对其进行处理。

Java SE 6 以上版本中已经包含了最新的 JAF，如果安装低版本的 Java SE 则需要单独[下载 JAF 框架](#)。下载该框架后，解压缩 **jaf-1\_1\_1.zip** 文件，并将 activation.jar 文件添加到典型安装路径下。

### 使用 Java EE 企业版

如果使用的是 Java EE，则在使用基本 JavaMail API 时，不需要做什么特殊的工作；Java EE API 中包含有 JavaMail。只要确保 j2ee.jar 文件位于典型安装路径下，并完成了所有的设置工作，也不需要安装 JavaBeans Activation Framework。[推荐安装 GlassFish]

### 练习

设置 JavaMail 环境。

## 发送邮件的例子

```
import java.util.Properties;

import javax.mail.Authenticator;
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 Sender {
    private String receiver = "hui.zz@163.com";
    private String subject = "Hello! My Friend! Sending best wishes!";
    private String cc = "hui.zz@hytc.edu.cn"; // (Blind) Carbon Copy
    private String mailContent = "Hello! Frodo! peril is approaching! go to Minas Tirith now!";
    private Session session; // session 没有子类, 可以被共享, 来自javax.mail包
    private Message msg; // 放内容, 实现接口part, 有子类MimeMessage, 来自javax.mail

    public void sendNow() {
        Properties props = new Properties(); // dictionary-hashtable-properties
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.host", "smtp.163.com"); // 没有开外网, 程序就跑不动了
                                                // 呵呵运行这程序的时候必须有smtp server
        session = Session.getDefaultInstance(props, new Authenticator() {
            public PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication("courses4public", "hytczzh");
            }
        });
        session.setDebug(true); // 允许调试, 因此可以用getDebug 取调试信息, 消息多得吓人。哼哼
        try {
            msg = new MimeMessage(session);
            msg.setFrom(new InternetAddress("courses4public@163.com"));
            InternetAddress toAddress = new InternetAddress(receiver); // 收件人
            msg.addRecipient(Message.RecipientType.TO, toAddress); // 加收件人
            InternetAddress ccAddress = new InternetAddress(cc); // 收件人
            msg.addRecipient(Message.RecipientType.CC, ccAddress); // 加收件人
            msg.setSubject(subject);
            msg.setText(mailContent);
            Transport.send(msg);
        }
```

```
        } catch (MessagingException ex) {
            while ((ex = (MessagingException) ex.getNextException()) != null)
                ex.printStackTrace();
        }
    } // sendNow end

public static void main(String[] args){
    new Sender().sendNow();
}
}
```

## JavaMail 核心类

首先浏览一下构成 JavaMail API 的核心类：**会话、消息、地址、验证程序、传输，存储和文件夹**。所有这些类都可以在 JavaMail API 即 javax.mail 的顶层包中找到，但也会发现自己使用的具体子类是在 javax.mail.internet 包中找到的。

### java.util.Properties 类

JavaMail 需要 Properties 来创建一个 session 对象。它将寻找字符串"mail.smtp.host"属性值就是发送邮件的主机。

典型用法:

```
Properties props = new Properties(); // System.getProperties();
props.put("mail.smtp.host", "smtp.163.com");//可换上你的 smtp 主机名
或 props.setProperty("mail.smtp.host", "smtp.163.com");
```

<code>String</code>	<b><a href="#">getProperty (String key)</a></b> 用指定的键在此属性列表中搜索属性。
<code>String</code>	<b><a href="#">getProperty (String key, String defaultValue)</a></b> 用指定的键在属性列表中搜索属性。
<code>void</code>	<b><a href="#">list (PrintStream out)</a></b> 将属性列表输出到指定的输出流。
<code>void</code>	<b><a href="#">list (PrintWriter out)</a></b> 将属性列表输出到指定的输出流。
<code>void</code>	<b><a href="#">load (InputStream inStream)</a></b> 从输入流中读取属性列表（键和元素对）。
<code>void</code>	<b><a href="#">load (Reader reader)</a></b> 按简单的面向行的格式从输入字符流中读取属性列表（键和元素对）。
<code>void</code>	<b><a href="#">loadFromXML (InputStream in)</a></b> 将指定输入流中由 XML 文档所表示的所有属性加载到此属性表中。
<code>Enumeration&lt;?&gt;</code>	<b><a href="#">propertyNames ()</a></b> 返回属性列表中所有键的枚举，如果在主属性列表中未找到同名的键，则包括默认属性列表中不同的键。
<code>void</code>	<b><a href="#">save (OutputStream out, String comments)</a></b> <b>已过时。</b> 如果在保存属性列表时发生 I/O 错误，则此方法不抛出 IOException。保存属性列表的首选方法是通过 <code>store (OutputStream out, String comments)</code> 方法或 <code>storeToXML (OutputStream os, String comment)</code> 方法来进行。

<u>Object</u>	<b><u>setProperty</u></b> ( <u>String</u> key, <u>String</u> value) 调用 Hashtable 的方法 put。
void	<b><u>store</u></b> ( <u>OutputStream</u> out, <u>String</u> comments) 以适合使用 <u>load(InputStream)</u> 方法加载到 Properties 表中的格式，将此 Properties 表中的属性列表（键和元素对）写入输出流。
void	<b><u>store</u></b> ( <u>Writer</u> writer, <u>String</u> comments) 以适合使用 <u>load(Reader)</u> 方法的格式，将此 Properties 表中的属性列表（键和元素对）写入输出字符。
void	<b><u>storeToXML</u></b> ( <u>OutputStream</u> os, <u>String</u> comment) 发出一个表示此表中包含的所有属性的 XML 文档。
void	<b><u>storeToXML</u></b> ( <u>OutputStream</u> os, <u>String</u> comment, <u>String</u> encoding) 使用指定的编码发出一个表示此表中包含的所有属性的 XML 文档。
<u>Set&lt;String&gt;</u>	<b><u>stringPropertyNames</u></b> () 返回此属性列表中的键集，其中该键及其对应值是字符串，如果在主属性列表中未找到同名的键，则还包括默认属性列表中不同的键。

## javax.mail.Session 类

Session类定义了一个基本的邮件会话，建模同远程邮件系统服务器的交互应答过程。

Session对象利用java.util.Properties对象获取诸如邮件服务器、用户名、密码等信息，以及其他可在整个应用程序中共享的信息。一般情况下，Session对象在一个JVM里只有一个，所以可能有多个用户共享一个Session。然而Session 里用户名和密码极有可能存在，所以安全问题要注意。

Session 类没有Constructor (私有构造函数)，所以只能用

static Session getInstance(Properties prop)

返回一个独享的 Session 对象。

而

static Session getDefaultInstance(Properties prop)

返回一个共享的Session 对象。

后者在之后的调用时，返回同一个Session。如果反复进行类似的操作，那么用后者不必每次都去重设Properites，显得更方便一些。[[具体看自己的操作需求](#)]

Properties 对象放的是Session 所必须的参数，参数分两部份：

参数名，参数值

其内容既可以用put(key, value) 存放，也可以用System.getProperties()取得系统默认值。推荐使用 Properties.setProperty(String key, String value)

例如下面的语句：

```
props.put("mail.smtp.host", "127.0.0.1");
```

其中mail.smtp.host 是参数名，而127.0.0.1 是值。**参数名不可以乱写**，是JavaMail 体系文档里规定的。

除了mail.smtp.host 之外还有：

<b>mail.transport.protocol</b>	调用getTransport()时可以得到它。返回默认的传输协议
<b>mail.store.protocol</b>	调用getStore()时可以得到它，返回默认存储协议实现
<b>mail.host</b>	在没指定主机时，存储和传输时使用这个主机
<b>mail.user</b>	默认的用户 缺省值 user.name
<b>mail.from</b>	当前用户 缺省值 username@host
<b>mail.protocol.host</b>	缺省值 mail.host
<b>mail.protocol.user</b>	缺省值 mail.user
<b>mail.debug</b>	会话调试开关，缺省是false

### 练习

准备 Session 对象以备后用。

```
Properties props = new Properties(); //开一个新特性
props.put("mail.transport.protocol","smtp");
props.put("mail.smtp.host","mail.163.com");
props.put("mail.smtp.port","25");
Session mySession=Session.getInstance(props); //好了！ 创建成功！
如果用getDefaultInstance 有什么区别？？
```

## javax.mail.Authenticator 类

现在大多数邮件系统为了防止邮件乱发(spam)，设定了**smtp**身份认证功能。因此发送的时候需要提供用户名密码的情况越来越多了。

**Java** 用一个类来封装用户认证操作，这个类就是

**java.lang.Object**

|

+ -- **javax.mail.Authenticator** (英文发音，重音在 上) |

验证信息需要通过**Session** 传给邮件服务器，所以**getInstance** 有以下的变形

```
static Session getInstance(Properties prop, Authenticator auth);
static Session getDefaultInstance(Properties prop, Authenticator auth);
```

其中的**Authenticator** 负责密码校验。

如果不需要验证身份，就用 **null** 做第二个参数，或者干脆用单参数的**getInstance()**。

如果 **session** 是需要密码的，那么 **session** 会自动发出如下调用：

```
javax.mail.PasswordAuthentication getPasswordAuthentication()
```

**PasswordAuthentication** 是一个包装类，里面包了用户名和密码。

如果**smtp**需要身份验证，则程序中需要自定义一个类继承**Authenticatior**，然后在类中实现**getPasswordAuthentication()**方法。

```
class Auth extends Authenticator{
    Properties pwd;
    public Auth() {
        try {
            pwd= new Properties();
            pwd.put("Alex","123");
            pwd.put("Mary","456");
            pwd.put("Obama","110");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public PasswordAuthentication getPasswordAuthentication() {
        String str= getDefaultUserName(); // 取得当前的用户
        System.out.println("the user:"+str);
        if (pwd.containsKey(str)) {
            // 当前用户在密码表里，就取出密码，封装好返回
            return new PasswordAuthentication(str, (String) pwd.get(str));
        } else {
            // 若当前用户不在密码表里，就提供null, session 不负责密码验证
            return null;
        }
    }
} // end class Auth
```

在这里，构造方法建立一个用户名-密码列表，这个列表可以放在文件中，可以写死在 java 程序里，也可以从 jndi ldap 中去取。

这个方法必须提供，因为 **getInstance()** 执行时，如果系统要求验证，就得提供这个方法。

在使用的时候，

```
Auth au = new Auth(); // 建立验证类实例
session=Session.getDefaultInstance(props, au); //如果服务器要求验证,就提供用户名、
密码对。如果不需,这个类的方法也不会调用。
```



Method Summary	
protected java.lang.String	<a href="#">getUserName()</a>
protected <a href="#">PasswordAuthentication</a>	<a href="#">getPasswordAuthentication()</a> Called when password authentication is needed.
protected int	<a href="#">getRequestingPort()</a>
protected java.lang.String	<a href="#">getRequestingPrompt()</a>
protected java.lang.String	<a href="#">getRequestingProtocol()</a> Give the protocol that's requesting the connection.
protected java.net.InetAddress	<a href="#">getRequestingSite()</a>

## 练习

编制一个封装好的类，可以根据不同的构造参数发送邮件。不需要GUI界面，只需要发送成功。

在上面作业基础上，建立一个简单的GUI界面，用户可以编写邮件（没有附件），指定收件人，然后点击发送，可以发出邮件。

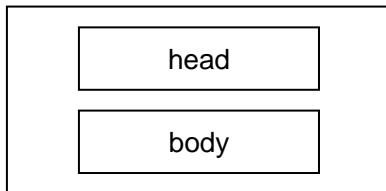
## javax.mail.Message 类

一旦创建了自己的 Session 对象，就可以发送消息了。这时要用到消息 Message 类型。由于 Message 是一个抽象类，使用时必须使用一个具体的子类型。大多数情况下，这个子类是 javax.mail.internet.MimeMessage。一个 MimeMessage 是封装了 MIME 类型数据和报头的消息。消息的报头严格限制为只能使用 US-ASCII 字符，尽管非 ASCII 字符可以被编码到某些报头字段中。

**MIME** 是什么东西？

邮件系统从一开始起就是为文本准备的，可是这世界变化快，非ASCII文件越来越多。**如何将二进制文件表示为ASCII文件以利传送呢？** MIME多用途Internet邮件扩充定义了这些方法。这些方法提供在**javax.mail** 包里，以API 形式提供：**javax.mail.Message**。

**Message** 的结构：



一个消息的头部包括主题，接收者，发送者发送日期等。实际的消息本身属于**body**部份。**MIME** 规范允许消息有多个部份，每个部份有自己的编码和属性。当然一个邮件可以有多个附件！

## 使用的API

### javax.mail.internet.MimeMessage

**Message** 是个抽象类，不能直接创建对象。它的子类**MimeMessage** 则可以生成对象。看一个构造方法

**public MimeMessage(Session session)**

这个方法根据**session** 生成一个消息。

另一个方法是

**public MimeMessage (MimeMessage msg)**

这个方法复制另一个**Message** 的实例，但效率比较差，一般用下面的

**public Message reply(boolean replyToAll)**

生成新的**Message**。

## 实例查看：一个Message 头部一般这个样子

```
Received: from njpcmesrv1.exclusives.nba.com([127.0.0.1]) by 21cn.com(AIMC2.9.5.6)
with SMTP id jm1803f550b5b; Wed, 03 Sep 2003 01:57:10 +0800
```

```
Received: from njpcmesrv1.exclusives.nba.com([12.18.6.135]) by 21cn.com(AIMC2.9.5.4)
with SMTP id AISP action; Wed, 03 Sep 2003 01:57:09 +0800
```

**To:** clydy@21cn.com

**Message-Id:** <20030902140122.59AA.241255-16845@exclusives.nba.com>

**Date:** Tue, 02 Sep 2003 14:01:22 -0400 (EDT)

**From:** NBA Daily <nba@exclusives.nba.com>

**Subject:** NBA Daily - Tuesday, September 2nd

**Content-Return:** allowed

```
Content-Type: text/plain
Content-Transfer-Encoding: 7bit
X-Mailer: CME-V5.0.5; L#8358-nba-p0
MIME-Version: 1.0
X-AIMC-AUTH: (null)
X-AIMC-MAILFROM: nba@exclusives.nba.com
```

分析一下头部的内容：

#### ■ From

指邮件的发出地，随便你填。取得发出地可以用

**Address[] getFrom()** 这是一个**Address** 对象数组。

**void setFrom()**

或者 **void setFrom(Javax.mail.Address fromAddress)**

或者 **void addFrom(Address[] moreAddress)**

不带参数的**setFrom**有默认属性可以取得。一般情况下设定多个**From**没意义。

#### ■ setRecipient

这个方法设置 **TO CC 和 BCC** 原型是这样的：

**void setRecipient(Message.RecipientType type, Address[] addresses);**

**void addRecipient(Message.RecipientType type, String address);**

其中的**Message.RecipientType type** 是常量，有：

**Message.RecipientType.TO**

**Message.RecipientType.CC**

**Message.RecipientType.BCC**

这些常量的意义不言自明。收件人可以是一个**Address[]**的数组或字符串。

#### ■ ReplyTo

指定回信字段。回信不一定回给发信人。用

**void setReplyTo(Address[] addresses)** 来设定。

**Address[] getReplyTo()** 可以取得回复地址信息。

#### ■ Subject 和 setSendDate(Date) 及 setSendDate()

用 **void setSubject(String ss)** 设定邮件标题

用 **String getSubject()** 取得标题

用 **setSendDate(Date dd)** 指定发送时间，用**不带参**的形式用当前时间发送。

#### ■ MessageID

电子邮件非常多的时候，ID可以用于索引和管理。如：标题内容都完全相同！！

**String getMessageID()** 取得这个编号。

#### ■ 可以在头部放其它东西以便用户灵活处理。

**void setHeader(String name, String value);**

**String getHeader(String name, String delimiter)** 可以取得头部的名-值对，**delimiter**是分界符，缺省是逗号。

## 动手操作

一旦创建了消息，就可以设置其各个部分。

```
MimeMessage mm = new MimeMessage(session);
```

Message(消息)实现 Part(部分)接口(MimeMessage 实现 MimePart)。设置内容的基本机制是 setContent()方法，它带有表示内容和 MIME 类型的参数：

```
message.setContent("Hello, my friend! ", "text/plain");
```

如果正在使用 MimeMessage，并且消息是纯文本，那么就可以使用 setText()方法。该方法只需要一个表示实际内容的参数，默认的 MIME 类型为纯文本：

```
message.setText("Hello, my friend! ");
```

对于纯文本消息，setText()方法更常常被用来设置内容。要发送其他类型的消息，如 HTML 消息，就要使用 setContent 方法()。现在用的更多的是 HTML 消息。

要设置主题，使用 setSubject()方法：

```
message.setSubject("First letter sent by JavaMail! ");
```

## javax.mail.Address 类

一旦创建了会话和消息，并为消息填充了内容，就需要用 Address 类为信件标上地址。同 Message 类一样，Address 类也是抽象类。可以使用 javax.mail.internet.InternetAddress 类。要创建只带有电子邮件地址的地址，可以把电子邮件地址传递给 Address 类的构造器：

```
Address address = new InternetAddress("president@whitehouse.gov");
```

如果想让一个名字出现在电子邮件地址后，也可以将其传递给构造器：

```
Address address = new InternetAddress("president@whitehouse.gov", "George Bush");
```

需要为消息的 from (发送者) 字段和 to (接收者) 字段创建地址对象。除非邮件服务器阻止这样做，否则要在发送的消息中注明该消息的发送者。

一旦创建好了地址，有两种方法可将地址与消息连接起来。为了鉴别发送者，可以使用 setFrom() 和 setReplyTo() 方法。

```
message.setFrom(address)
```

如果消息需要显示多个地址来源，则可以使用 addFrom() 方法：

```
Address address[] = ...;
```

```
message.addFrom(address);
```

为了鉴别消息接收者，可以使用 addRecipient() 方法。该方法除了需要一个地址参数外，还需要一个 Message.RecipientType 属性（消息的接收类型）。

```
message.addRecipient(type, address)
```

地址的 3 种预定义类型如下：

- Message.RecipientType.TO
- Message.RecipientType.CC
- Message.RecipientType.BCC

因此，如果一条消息将发送给副总统，同时还将发送该消息的副本给第一夫人，则采用下面的代码：

```
Address toAddress = new InternetAddress("vice.president@whitehouse.gov");
Address ccAddress = new InternetAddress("first.lady@whitehouse.gov");
message.addRecipient(Message.RecipientType.TO, toAddress);
message.addRecipient(Message.RecipientType.CC, ccAddress);
```

JavaMail API 没有提供检查电子邮件地址有效性的机制。可以自己编写支持扫描有效字符(在 RFC 822 文档中所定义的)的程序或检验 MX(邮件交换)记录，这些都超越了 JavaMail API 的范围。

## javax.mail.Transport 类

发送消息的最后一步操作是使用 Transport 类。该类使用特定协议（通常是 SMTP）语言来发送消息。它是一个抽象类，其操作与 Session 类有些相似。可以通过只调用静态的 send() 方法来使用该类的默认版本：

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

或者，可以从协议的会话中获取一个特定的实例，然后传递用户名和密码（不必要时可以为空）并发送消息，最后关闭连接：

```
message.saveChanges(); // implicit with send()
Transport transport = session.getTransport("smtp");
transport.connect(host, username, password);
transport.sendMessage(message, message.getAllRecipients());
transport.close();
```

当需要发送多个消息时，建议采用[后一种方法](#)，因为它[将保持消息间活动服务器的连接](#)。而基本的 send() 机制会为每一个方法调用都建立一条独立的连接。

注意：要查看经过邮件服务器邮件命令，可以用 `session.setDebug(true)` 方法设置调试标志。

### Method Summary

void	<a href="#"><u>addTransportListener(TransportListener l)</u></a> Add a listener for Transport events.
protected void	<a href="#"><u>notifyTransportListeners(int type, Address[] validSent, Address[] validUnsent, Address[] invalid, Message msg)</u></a> Notify all TransportListeners.
void	<a href="#"><u>removeTransportListener(TransportListener l)</u></a> Remove a listener for Transport events.
static void	<a href="#"><u>send(Message msg)</u></a> Send a message.
static void	<a href="#"><u>send(Message msg, Address[] addresses)</u></a> Send the message to the specified addresses, ignoring any recipients specified in the message itself.
abstract void	<a href="#"><u>sendMessage(Message msg, Address[] addresses)</u></a> Send the Message to the specified list of addresses.

## javax.mail.Store 和 javax.mail.Folder 类

使用 Session 类来获取消息，开始时与发送消息很相似。但是，在获取会话后，很有可能使用用户名和密码或 Authenticator 类来连接 Store 类。与 Transport 类一样，要告诉 Store 类将使用什么协议：

```
// Store store = session.getStore("imap");
Store store = session.getStore("pop3");
store.connect(host, username, password);
```

在连接 Store 类后，就可以获取一个 Folder 类，**在读取其中的消息前必须先打开该类。**

```
Folder folder = store.getFolder("INBOX");
folder.open(Folder.READ_ONLY);
Message[] messages = folder.getMessages();
```

对于 POP3 协议，唯一可用的文件夹是 INBOX。如果使用的是 IMAP 协议，则可以使用其他的文件夹。

注意：Sun 公司的提供程序本来想提供方便。而 Message message[] = folder.getMessages(); 这条语句却是一种从服务器逐条读取消息的缓慢操作，所以仅当您确实需要获取消息部分（该内容是所检索消息的内容）时可以使用这条语句。

一旦读取消息，就可以使用 getContent()方法获取其内容，或使用 writeTo()方法将其内容写到一个流中。getContent()方法只获取消息内容，而 writeTo()方法则还会输出报头。

```
System.out.println(((MimeMessage)message).getContent());
```

一旦阅读完邮件，就可以关闭对文件夹和存储的连接。

```
folder.close(aBoolean);
store.close();
```

传递给文件夹的 close()方法的布尔变量指定了是否通过清除已删除的消息来更新文件夹。

### Method Summary

void	<a href="#"><b>addFolderListener(FolderListener l)</b></a>	Add a listener for Folder events on any Folder object obtained from this Store.
void	<a href="#"><b>addStoreListener(StoreListener l)</b></a>	Add a listener for StoreEvents on this Store.
abstract <a href="#"><b>Folder</b></a>	<a href="#"><b>getDefaultFolder()</b></a>	Returns a Folder object that represents the 'root' of the default namespace presented to the user by the Store.
abstract <a href="#"><b>Folder</b></a>	<a href="#"><b>getFolder(java.lang.String name)</b></a>	Return the Folder object corresponding to the given name.
abstract <a href="#"><b>Folder</b></a>	<a href="#"><b>getFolder(URLName url)</b></a>	Return a closed Folder object, corresponding to the given URLName.
<a href="#"><b>Folder[]</b></a>	<a href="#"><b>getPersonalNamespaces()</b></a>	Return a set of folders representing the <i>personal</i> namespaces for the current user.

<u>Folder[]</u>	<u><a href="#">getSharedNamespaces()</a></u> Return a set of folders representing the <i>shared</i> namespaces.
<u>Folder[]</u>	<u><a href="#">getUserNamespaces(java.lang.String user)</a></u> Return a set of folders representing the namespaces for user.
protected void	<u><a href="#">notifyFolderListeners(int type, Folder folder)</a></u> Notify all FolderListeners.
protected void	<u><a href="#">notifyFolderRenamedListeners(Folder oldF, Folder newF)</a></u> Notify all FolderListeners about the renaming of a folder.
protected void	<u><a href="#">notifyStoreListeners(int type, java.lang.String message)</a></u> Notify all StoreListeners.
void	<u><a href="#">removeFolderListener(FolderListener l)</a></u> Remove a listener for Folder events.
void	<u><a href="#">removeStoreListener(StoreListener l)</a></u> Remove a listener for Store events.

实际上，理解使用这 7 个类的方式，是使用 JavaMail API 处理几乎所有事情所需全部内容。用这 7 个类以外的方式构建的 JavaMail API，其大多数功能都是以几乎完全相同或特定的方式来执行任务的，就好像内容是附件。特定的任务，如：搜索、隔离等将在后面进行介绍。

## 使用 JavaMail API

前面已经看到了如何操作 JavaMail API 的核心部分。下面将学习如何连接几个部分以执行特定的任务。

### 发送消息

#### 发送 Email 几大步骤：

1	把应该import 的 javax.mail,java.util,java.io 导入
2	创建一个Properties 对象，存放邮件服务器的信息
3	给Properties 对象赋值
4	生成Session 对象
5	从这个session 中创建一条Message
6	给这条Message 设定From TO CC BCC 和Subject
7	给邮件正文设定内容
8	用Transport.send()发送邮件

发送电子邮件消息涉及到获取会话、创建和填充消息并发送消息这些操作。可以在获取 Session 时，通过为要传递的 Properties 对象设置 mail.smtp.host 属性来指定 SMTP 服务器。

```

String host = "smtp.163.com";
String from = "courses4public@163.com";
String to = "courses4public@163.com";
// Get system properties
Properties props = System.getProperties();
// Setup mail server
props.put("mail.smtp.host", host);
props.put("mail.smtp.auth", "true");
// Get session
Session session = Session.getDefaultInstance(props, null);
session.setDebug(true);
// Define message
MimeMessage message = new MimeMessage(session);
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO, new
    InternetAddress(to));
message.setSubject("Hello JavaMail");
message.setText("Welcome to JavaMail");
message.saveChanges();
// Send message
Transport trans = session.getTransport("smtp");
trans.connect(host, "courses4public", "hytczzh");

```

```
trans.sendMessage(message, message.getAllRecipients());  
trans.close();
```

应该在 try-catch 块中编写代码，以在创建消息并发送它时可以抛出一个异常。

```
throws AddressException, MessagingException
```

### 练习

给你的朋友发送一条消息，告知他信件是自己用 JavaMail 编写并发送的！

## 获取消息

获取 Email 几大步骤:

1	把应该 import 的 javax.mail, java.util, java.io 导入
2	创建一个Properties 对象, 存放邮件服务器的信息
3	给Properties 对象赋值
4	生成Session 对象
5	从这个session 中创建Store
6	从Store中获取Folder
7	从Folder中读取邮件内容

对于阅读邮件来说, 首先要获取一个会话, 然后获取并连接到一个相应的收件箱的存储上, 接着打开相应的文件夹, 再获取消息。同时, 不要忘记了操作完成后关闭连接。

```

String host = "pop3.163.com";
String username = "courses4public";
String password = "hytczzh";
// Create empty properties
Properties props = new Properties();
//props.put("mail.pop3.auth", "true");
// Get session
Session session = Session.getDefaultInstance(props, null);
// Get the store
Store store = session.getStore("pop3");
store.connect(host, username, password);
// Get folder
Folder folder = store.getFolder("INBOX");
folder.open(Folder.READ_ONLY);
// Get directory
Message message[] = folder.getMessages();
for (int i = 0, n = message.length; i < n; i++) {
    System.out.println(i + ":" + message[i].getFrom()[0] + "\t"
        + message[i].getSubject());
}
// Close connection
folder.close(false);
store.close();

```

每一条消息执行何种操作取决于自己决定。上面的代码块只是显示了消息的发送者和主题。从技术上讲, 发送者地址列表可以为空, 此时 getFrom()[0] 调用会抛出一个异常。

为了显示整条消息, 可以提示用户在看完消息的发送者和主题字段后, 想看到消息的内容, 可以再调用消息的 writeTo()方法。

```

BufferedReader reader = new BufferedReader(new
    InputStreamReader(System.in));
// Get directory
Message message[] = folder.getMessages();
for (int i = 0, n = message.length; i < n; i++) {

```

```
System.out.println(i + ":" + message[i].getFrom()[0]
    + "\t" + message[i].getSubject());
System.out.println("Do you want to read message? "
    + "[YES to read/QUIT to end]");
String line = reader.readLine();
if ("YES".equals(line)) {
    message[i].writeTo(System.out);
} else if ("QUIT".equals(line)) {
    break;
}
}
```

## 练习

检查邮件

## 删除消息和标志

删除消息涉及操作与消息关联的标志。对不同的状态有不同的标志，有些标志是系统定义的，有些则是由用户定义的。预定义的标志都是在内部类 `Flags.Flag` 中定义的，如下所示：

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

`Flags` 中定义了这些标志，并不表示所有的邮件服务器/提供程序都支持。例如，除了删除消息外，POP 协议对它们都不支持。检查新邮件不是 POP 的任务，但它已内置到邮件客户程序中。要搞清楚什么标志受到支持，可以使用 `getPermanentFlags()` 方法来询问文件夹。

要删除消息，需要为消息设置 `DELETE` 标志：

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

第一次以 `READ_WRITE`（读-写）模式打开文件夹：

```
folder.open(Folder.READ_WRITE);
```

然后，处理完了所有的消息，关闭文件夹，并传递 `true` 值以擦去删除的消息。

```
folder.close(true);
```

用户可使用 `Folder` 类的 `expunge()` 方法来删除消息。但该方法对 Sun 公司的 POP3 提供程序不起作用。其他提供程序不确定也实现了此功能，适用于 IMAP 提供程序。由于 POP 只支持对收件箱的简单访问，使用 Sun 公司的提供程序时，您将不得不关闭文件夹以删除消息。

要移去标志，只需传递一个 `false` 值给 `setFlag()` 方法。要看看是否设置了某个标志，可以使用 `isSet()` 进行检查。

### Flag Method Summary

abstract <code>Flags</code>	<a href="#"><code>getFlags()</code></a>
	Returns a <code>Flags</code> object containing the flags for this message.
boolean	<a href="#"><code>isSet(Flags.Flag flag)</code></a>
	Check whether the flag specified in the flag argument is set in this message.
protected void	<a href="#"><code>setExpunged(boolean expunged)</code></a>
	Sets the expunged flag for this Message.
void	<a href="#"><code>setFlag(Flags.Flag flag, boolean set)</code></a>
	Set the specified flag on this message to the specified value.
abstract void	<a href="#"><code>setFlags(Flags flag, boolean set)</code></a>
	Set the specified flags on this message to the specified value.

## 自我验证

先前使用 Authenticator 类，以在需要时提示输入用户名和密码，而不是以字符串的形式传入它们。可以充分地使用验证。

不需使用主机、用户名和密码连接到 Store，可以配置 Properties 带有主机，并告诉 Session 关于自定义的 Authenticator 实例，如下所示：

```
// Setup properties
Properties props = System.getProperties();
props.put("mail.pop3.host", host);
// Setup authentication, get session
Authenticator auth = new PopupAuthenticator();
Session session = Session.getDefaultInstance(props, auth);
// Get the store
Store store = session.getStore("pop3");
store.connect();
```

然后使用 Authenticator 类的子类，并通过 getPasswordAuthentication()方法返回一个 PasswordAuthentication 对象。下面是这种实现的一个例子，其中一个字段同时适用于两部分内容。它不是一个 Project Swing 指南，只是在一个字段中输入了两部分内容，它们是用逗号隔开的。

```
public class PopupAuthenticator extends Authenticator {
    public PasswordAuthentication getPasswordAuthentication() {
        String username, password;
        String result = JOptionPane
            .showInputDialog("Enter 'username,password'");
        StringTokenizer st = new StringTokenizer(result, ",");
        username = st.nextToken();
        password = st.nextToken();
        return new PasswordAuthentication(username, password);
    }
}
```

由于 PopupAuthenticator 依赖于 Swing，会启动 AWT 事件处理线程。这在本质上要求代码中添加一个对 System.exit()的调用，以终止程序的执行。

## 回复消息

Message 类包含一个 reply()方法，以用正确的接收者和主题（添加“Re::”，如果没有的话）配置一条新消息。该方法不会为消息添加任何内容，只是为新的接收者复制发送者或回复到的报头。该方法使用一个布尔型参数，提示是否只回复给发送者（false）或回复给所有人(true)。

```
// 只回复给发送者false 或回复给所有人true
MimeMessage reply = (MimeMessage) message.reply(false);
reply.setFrom(new InternetAddress("president@whitehouse.gov"));
reply.setText("Thanks");
Transport.send(reply);
```

在发送消息时，要配置回复地址，可使用 setReplyTo()方法。

### 练习

回复邮件

## 转发消息

转发消息涉及的内容要稍微多一点，没有一个专门用于转发消息的方法，可以通过处理组成消息的各个部分来创建要转发的消息。

一条邮件消息可由多个部分组成，每一部分是一个 BodyPart(报文部分)，在操作 MIME 消息时则是 MimeBodyPart。不同的报文部分组合到一个称为 Multipart 的容器中，或一个 MimeMultipart 容器。

要转发消息，要创建一个用于消息文本的部分，和用于要转发的消息的第二个部分，并将这两个部分组合成一个 multipart(多个部分)。然后可以把这个 multipart 添加到一个合适的注明地址的消息中并发送它。

这就是转发消息的本质。要把一条消息的内容复制给另一条消息，只需通过它的 DataHandler 类复制即可，它是出自于 JavaBeans Activation Framework 的一个类。

```
// Create the message to forward
Message forward = new MimeMessage(session);
// Fill in header
forward.setSubject("Fwd: " + message.getSubject());
forward.setFrom(new InternetAddress(from));
forward.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
// Create your new message part
BodyPart messageBodyPart = new MimeBodyPart();
messageBodyPart.setText("Here you go with the original message:\n\n");
// Create a multi-part to combine the parts
Multipart multipart = new MimeMultipart();
multipart.addBodyPart(messageBodyPart);
// Create and fill part for the forwarded content
messageBodyPart = new MimeBodyPart();
messageBodyPart.setDataHandler(message.getDataHandler());
// Add part to multi part
multipart.addBodyPart(messageBodyPart);
// Associate multi-part with message
forward.setContent(multipart);
// Send message
Transport.send(forward);
```

## 操作附件

附件是与邮件消息关联的资源，通常保存在消息之外，如：一个文本文件，电子表格或图片。对于像 Eudora 和 Pine 之类的常用邮件程序，可以通过 JavaMail API 把资源附加到邮件消息上，并在您接收消息时获取附件。

### 发送附件

发送附件与转发消息非常相似，要创建组成完整消息的各个部分。在创建好第一个部分即消息文本之后，添加用 DataHandler 类处理的其他部分就是您的附件。

从一个文件读取附件时，附件的数据资源是 FileDataSource；从 URL 读取时，则是 URLDataSource。一旦有了自己的 DataSource，将其通过 setDataHandler()方法最终附加到 BodyPart 上之前，只需将其传递给 DataHandler 类的构造器即可。假定想保留附件的原始文件名，要做的最后一件事就是用 BodyPart 类的 setFileName()方法设置与附件关联的文件名。

所有这些操作如下所示：

```
// Define message
Message message = new MimeMessage(session);
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
message.setSubject("Hello JavaMail Attachment");

// Create the message part
BodyPart messageBodyPart = new MimeBodyPart();
// Fill the message
messageBodyPart.setText("Pardon Ideas");
Multipart multipart = new MimeMultipart();
multipart.addBodyPart(messageBodyPart);

// Part two is attachment
messageBodyPart = new MimeBodyPart();
DataSource source = new FileDataSource(filename);
messageBodyPart.setDataHandler(new DataHandler(source));
messageBodyPart.setFileName(filename);
multipart.addBodyPart(messageBodyPart);

// Put parts in message
message.setContent(multipart);

// Send the message
Transport.send(message);
```

在消息中包含附件时，如果程序是一个 servlet，用户就必须上传附件，上传的每一个文件都可以用一个表单来处理，该表单是以 multipart/表单数据 (form-data) 来编码的。

```
<form enctype="multipart/form-data" method="post" action="/myservlet">
    <input type="file" name="thefile">
    <input type="submit" value="upload">
</form>
```

注意：消息的大小受 SMTP 服务器的限制，而不是由 JavaMail API 限制的。如果出现了问题，可以通过设置 ms 和 mx 参数来考虑增加 Java 堆区的空间尺寸。

## 获取附件

从消息中取出附件比发送附件涉及的操作要稍微多一点，而 MIME 没有简单的附件概念。当消息带有附件时，消息的内容就是一个 Multipart 对象。然后需要处理各个部分，以获取主要内容和附件。通过 part.getDisposition()方法标记上 Part.ATTACHMENT 配置的部分就是附件。同时，附件也可以不带有配置（和非文本 MIME 类型）或 Part.INLINE 配置。当配置是 Part.ATTACHMENT 或 Part.INLINE 时，可以脱离该消息部分的内容将其保存起来。只需通过 getFileName()方法获取原始文件名，并通过 getInputStream()方法获取输入流即可。

```
Multipart mp = (Multipart) message.getContent();
for (int i = 0, n = multipart.getCount(); i < n; i++) {
    Part part = multipart.getBodyPart(i);
    String disposition = part.getDisposition();
    if (disposition != null
        && ((disposition.equals(Part.ATTACHMENT)
            || (disposition.equals(Part.INLINE))))) {
        saveFile(part.getFileName(), part.getInputStream());
    }
}
```

saveFile()方法用于根据文件名创建一个文件，并从输入流中读取字节，将它们写入到文件中去。如果文件已存在，就在文件名后添加一个编号，直到找到一个可用的文件名为止。

```
// from saveFile()
File file = new File(filename);
for (int i = 0; file.exists(); i++) {
    file = new File(filename + i);
}
```

上面的代码介绍了消息的各个部分被标上相应的标志的一个最简单的例子。要想包含所有的情况，还要对 disposition 值为 null 及消息部分为 MIME 类型的情况作相应处理。

```
if (disposition == null) {
    // Check if plain
    MimeBodyPart mbp = (MimeBodyPart) part;
    if (mbp.isMimeType("text/plain")) {
        // Handle plain
    } else {
        // Special non-attachment cases here of image/gif,
        // text/html, ...
    }
    // more code to process
}
```

## 处理 HTML 消息

发送基于 HTML 的消息比发送纯文本消息要稍微复杂一点，尽管它不需要做大量的工作。

### 发送 HTML 消息

如果发送一个 HTML 文件作为消息，并让邮件阅读者取出任何嵌入的图片或相关片段，那么就可以使用消息的 `setContent()`方法，以字符串形式传递消息内容，并把**内容类型**设置为 `text/html`。

```
String htmlText = "<H1>Hello</H1>"  
+ "<img src=\"http://www.jguru.com/images/logo.gif\">";  
message.setContent(htmlText, "text/html");
```

在接收端，如果 JavaMail API 获取消息，在该 API 中没有内置任何用于以 HTML 格式显示消息的功能。JavaMail API 只以字节流的形式来查看消息。要以 HTML 格式显示消息，必须使用 Swing JEditorPane 或某些第 3 方 HTML 阅读器组件。

```
if (message.getContentType().equals("text/html")) {  
    String content = (String) message.getContent();  
    JFrame frame = new JFrame();  
    JEditorPane text = new JEditorPane("text/html", content);  
    text.setEditable(false);  
    JScrollPane pane = new JScrollPane(text);  
    frame.getContentPane().add(pane);  
    frame.setSize(300, 300);  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.setVisible(true);  
}
```

### 在消息中包含图片

如果 HTML 消息中嵌入了作为消息一部分的图片，并且想保持消息内容的完整，就必须把图片看作附件，并用特殊的通信标识符 URL 引用该图片，该通信标识符引用的是图片附件的内容 ID 报文。

嵌入图片的处理与附加一个文件到消息上非常相似，**唯一不同之处在于：必须区分 MimeType 中，哪些部分是在构造器（或通过 setSubType()方法）通过设置其子类型而使之相关的，以及将图片的内容 ID 报头设置成任意字符串，它将在 img 标记中用作图片的源路径。**

下面显示了一个完整的示例：

```
String file = "";  
// Create the message  
Message message = new MimeMessage(session);  
// Fill its headers  
message.setSubject("Embedded Image");
```

```
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO,
                     new InternetAddress(to));

// Create your new message part
BodyPart messageBodyPart = new MimeBodyPart();
String htmlText = "<H1>Hello</H1>" + "<img src=\"cid:memememe\">";
messageBodyPart.setContent(htmlText, "text/html");
// Create a related multi-part to combine the parts
MimeMultipart multipart = new MimeMultipart("related");
multipart.addBodyPart(messageBodyPart);

// Create part for the image
messageBodyPart = new MimeBodyPart();
// Fetch the image and associate to part
DataSource fds = new FileDataSource(file);
messageBodyPart.setDataHandler(new DataHandler(fds));
messageBodyPart.setHeader("Content-ID", "memememe");
// Add part to multi-part
multipart.addBodyPart(messageBodyPart);
// Associate multi-part with message
message.setContent(multipart);
```

## 练习

发送带有图片的 HTML 消息

## 用 SearchTerm 搜索

JavaMail API 包含一种可用于创建 SearchTerm（搜索条件）的筛选机制，可以在 javax.mail.search 包中找到。一旦创建了 SearchTerm，就可以询问某个文件夹匹配的消息，并检索出消息对象数组：

```
SearchTerm st = ...;  
Message[] msgs = folder.search(st);
```

有 22 种不同的类可用于帮助创建搜索条件。

- AND 条件(AndTerm 类)
- OR 条件(OrTerm 类)
- NOT 条件(NotTerm 类)
- SENT DATE 条件(SentDateTerm 类)
- CONTENT 条件(BodyTerm 类)
- HEADER 条件(FromTerm / FromStringTerm, RecipientTerm / RecipientStringTerm, SubjectTerm, etc.)

本质上，可以为匹配的消息创建一个逻辑表达式，然后进行搜索。例如，下面显示了一条消息的条件搜索示例，该消息带有（部分带有）一个 AD 主题字符串，其发送者字段为 friend@public.com。具体使用中可能考虑定期运行该查询，并自动删除任何返回的消息。

```
SearchTree st = new OrTerm(new SubjectTerm("AD:") ,  
                           new FromStringTerm(  
                               "friend@public.com"));  
  
Message[] msgs = folder.search(st);
```

## 常见问答

问. JavaMail 是否包括所有必要的邮件服务器?

答: 不是, JavaMail API 包不包括任何邮件服务器。为了使用 JavaMail API 包, 你将需要访问 IMAP 或 POP3 邮件服务器(用于阅读邮件)和/或 SMTP 邮件服务器(用于发送邮件)。这些邮件服务器通常由 Internet 服务提供商提供, 或者作为组织网络基础结构的一部分。

问. 从哪里可以获得必要的邮件服务器?

答: Sun Java System Messaging Server 可用于 Solaris 和 Windows 平台。华盛顿大学的 IMAP 服务器支持多种平台(UNIX、32 位 Windows 等)。可从如下地址获取源代码:  
<ftp://ftp.cac.washington.edu/imap/imap.tar.Z>。其他的许多供应商提供了支持 Internet 标准的邮件服务器。可以从 IMAP Connection 和 Internet Mail Consortium 获得更多信息。

问. 我应该使用什么主机名、用户名或密码?

答: 我们不提供邮件服务器让你使用。你必须使用自己的邮件服务器, 或者使用 Internet 服务提供商或你所工作的公司提供的邮件服务器。网络管理员可能给你一些必要的信息用于配置 JavaMail, 以便同邮件服务器一起工作。

问. 我如何配置 JavaMail 通过代理服务器工作?

答: 大多数代理服务器只支持 HTTP 协议。JavaMail 没有使用 HTTP 协议来阅读或发送邮件。使用代理服务器的一个主要原因是为了允许企业网络中的 HTTP 请求通过企业防火墙。防火墙通常会阻止对 Internet 的大多数访问, 但允许来自代理服务器的请求通过。此外, 企业网络内部的邮件服务器将为邮件执行类似的功能, 通过 SMTP 接收消息, 然后将它们转发到 Internet 上的最终目的地, 以及接收传入的消息, 然后将它们发送到合适的内部邮件服务器。

如果你的代理服务器支持 SOCKS V4 或 V5 协议(<http://www.socks.nec.com/aboutsocks.html>, RFC1928), 并允许匿名连接, 可以告诉 Java 运行时把所有的 TCP socket 直接连接到 SOCKS 服务器。参阅 <http://java.sun.com/j2se/1.4/docs/guide/net/properties.html>, 获取 socksProxyHost 和 socksProxyPort 属性的最新文档。这些是系统级属性, 而不是 JavaMail 会话属性。当调用应用程序时, 它们可以从命令行中设置, 例如: `java -DsocksProxyHost=myproxy ...`。这个工具可用于指出从 JavaMail 到 SOCKS 代理服务器进行 SMTP、IMAP 和 POP3 通信。注意, 设置这些属性将告诉所有 TCP socket 连接到 SOCKS 代理, 在应用程序的其他方面上, 这可能会带来负面影响。

假如没有这样的 SOCKS 服务器, 如果想使用 JavaMail 来直接访问防火墙外部的邮件服务器, 那将需要配置防火墙来允许这一访问。一个简单的 HTTP 代理 Web 服务器是足够的。

问. 当试图在 Linux 中运行程序时, 得到了非常奇怪的错误消息, 而且程序运行失败了。  
错误在哪里?

答: 通常, 错误消息看起来像下面这样:

Exception in thread "main"

```
java.lang.VerifyError:(Class:com/sun/mail/pop3/POP3Store,
method: finalize Signature :()V)
Illegal use of nonvirtual function call
```

问题是由于在 Linux 上，使用的 `unzip` 命令是有 bug 的版本，这样解压缩 JavaMail 下载包时，`unzip` 命令破坏了 `mail.jar` 文件。获取更新版本的 `unzip` 命令或使用 JDK 的 `jar` 命令来解压缩下载包。

问. 在运行于 `SecurityManager` 下面的应用程序中，我如何使用 `JavaMail`；我必须授予应用程序和 `JavaMail` 什么权限？

答：在具有 `SecurityManager` 的 JDK 1.2（或更新版本）中，当使用 `JavaMail` 时，`JavaMail` 读取 `mail.jar` 文件中的配置文件有时会失败。在从 `activation.jar` 文件中读取配置文件时，`JavaBeans Activation Framework` 可能也有相同的问题。这些默认配置文件是作为“资源”文件存储的，并且存储在 `jar` 文件的 `META-INF` 目录中。

有许多调试技术可用于决定这是否是个问题。设置 `Session` 属性“`mail.debug`”为 `true`（或调用 `session.setDebug(true)`），将导致 `JavaMail` 在试图加载各个配置文件时打印调试消息。形如“`DEBUG: cant load default providers file`”（`DEBUG:` 不能加载默认提供程序文件）的消息指出这个问题可能存在。同样，设置 `System` 属性“`javax.activation.debug`”为“`true`”（例如，通过使用 `"java -Djavax.activation.debug=true ..."` 来运行程序），将导致 `JAF` 在试图加载各个资源文件时打印调试消息。最后，通过设置 `system` 属性“`java.security.debug`”为“`access:failure`”（例如，通过使用 `"java -Djava.security.debug=access:failure ..."` 来运行程序），`JDK` 可以产生有用的调试输出。

除了读取配置文件的必要权限外，应用程序（和 `JavaMail`）也将需要一定的权限才可以连接到它使用的邮件服务器。如果应用程序使用 `System` 属性来配置 `JavaMail`（例如，像许多 `JavaMail` 演示程序所做的那样，通过传递从 `System.getProperties()` 中返回的 `Properties` 对象到 `Session` 构造函数），它也将需要一定的权限才可以使用 `System Properties` 对象。另外，应用程序可以使用自己的 `Properties` 对象，以及确信设置“`mail.from`”属性或“`mail.user`”和“`mail.host`”属性（参见 `InternetAddress.getLocalAddress()` 方法）。

在 JDK 1.2 `SecurityManager` 中，为了使应用程序能够使用 `JavaMail`，应用程序、`JavaMail` 和 `JAF` 将需要某些权限，比如下面的一些权限（一定要使用适当的值替换主机名和路径名）；把这些权限添加到应用程序使用的安全策略文件中。

```
grant {
    // following two permissions allow
    // access to default config files
    permission java.io.FilePermission
    "/path/to/mail.jar", "read";
    permission java.io.FilePermission
    "/path/to/activation.jar", "read";
    // following to use SMTP
    permission java.net.SocketPermission
    "SMTPHOST:25", "connect,resolve";
    // following to use IMAP
    permission java.net.SocketPermission
    "IMAPHOST:143", "connect,resolve";
    // following to use POP3
```

```
    permission java.net.SocketPermission
    "POP3HOST:110", "connect,resolve";
    // following needed if System.getProperties() is used
    permission java.util.PropertyPermission
    "*", "read,write";
};
```

问. 如何配置 Web 服务器来运行 JavaMail 演示 servlet?

答: 针对以下 Web 服务器的指导说明可从这里获得:

Tomcat

Apache with JServ

iPlanet Web Server

Java Web Server

问. 当在 servlet 中使用 JavaMail 时, 未找到任何的 JavaMail 类。我已经在服务器的 CLASSPATH 中添加了 mail.jar?

答: 当改变 CLASSPATH 时, 通常有必要完全重启 Web 服务器。

问. 我的 servlet 可以找到 JavaMail 类, 但 JavaMail 抱怨它不能找到针对“smtp”或“imap”的服务提供程序或地址类型 “rfc822”。

答: 通常这是因为 JavaMail 无法访问 mail.jar 中的配置文件, 而这可能是由于安全权限问题造成的; 参见 本条目, 获取更多的细节。也保证你没有提取 mail.jar 内容; 在服务器的 CLASSPATH 中, 应该包括未更改的 mail.jar 文件。

问. 在哪里可以找到 jws.jar? 我已经安装了 Java Web Server 2.0, 并试图运行 JavaMailServlet。README 文件指示我在 CLASSPATH 中添加 jws.jar。

答: jws.jar 不再与 Java Web Server 一起发行 (在以前版本中, 它们是一起发行的), 因此不需要在 CLASSPATH 中添加它。只要在 CLASSPATH 中添加 mail.jar 和 activation.jar, 然后重启 Java Web Server

## JavaMail 的常用类速查

### (1) javax.mail.Properties 类

JavaMail 需要 Properties 来创建一个 session 对象。它将寻找字符串"mail.smtp.host"，属性值就是发送邮件的主机。

用法:

```
Properties props = new Properties ();
props.put("mail.smtp.host", "smtp.163.com"); //可以换上你的 smtp 主机名。
```

### (2) javax.mail.Session 类

这个 Session 类代表 JavaMail 中的一个邮件 session. 每一个基于 JavaMail 的应用程序至少有一个 session 但是可以有任意多的 session。在这个例子中, Session 对象需要知道用来处理邮件的 SMTP 服务器。

用法:

```
Session sendMailSession;
sendMailSession = Session.getInstance(props, null);
```

### (3) javax.mail.Transport 类

邮件是既可以被发送也可以被受到。JavaMail 使用了两个不同的类来完成这两个功能: Transport 和 Store. Transport 是用来发送信息的, 而 Store 用来收信。

用法:

```
Transport transport;
transport = sendMailSession.getTransport("smtp");
```

用 JavaMail Session 对象的 getTransport 方法来初始化 Transport。传过去的字符串申明了对象所要使用的协议, 如"smtp"。这将为我们省了很多时间。因为 JavaMail 以境内置了很多协议的实现方法。

注意: JavaMail 并不是绝对支持每一个协议, 目前支持 IMAP、 SMTP 和 POP3.

### (4) javax.mail.MimeMessage 类

Message 对象将存储我们实际发送的电子邮件信息, Message 对象被作为一个 MimeMessage 对象来创建并且需要知道应当选择哪一个 JavaMail session。

用法:

```
Message newMessage = new MimeMessage(sendMailSession);
```

#### (5) javax.mail.InternetAddress 类

一旦创建了 Session 和 Message，并将内容填入消息后，就可以用 Address 确定信件地址了。和 Message 一样，Address 也是个抽象类。您用的是 Javax.mail.internet.InternetAddress 类。

用法：

```
InternetAddress from=new InternetAddress("xxf@cafe.com");
```

#### (6) javax.mail.Store 类

Store 类实现特定邮件协议上的读、写、监视、查找等操作。通过 Javax.mail.Store 类可以访问 Javax.mail.Folder 类。

用法：

```
Store store=s.getSorte("pop3");//s 为一个邮件会话  
store.connect(popserver,username,password);//通过你提供的 pop 地址,用户名和密码登录  
你的邮箱
```

#### (7) javax.mail.Folder 类

Folder 类用于分级组织邮件，并提供照 Javax.mail.Message 格式访问 email 的能力。

用法：

```
Folder folder=store.getFolder("INBOX");  
folder.open(Folder.READ_ONLY);
```

#### (8) javax.mail.Internet.MimeMultipart

一般保存电子邮件内容的容器是 Multipart 抽象类，它定义了增加和删除及获得电子邮件不同部分内容的方法。由于 Multipart 是抽象类，必须为它使用一个具体的子类，JavaMail API 提供 javax.mail.Internet.MimeMultipart 类来使用 MimeMessage 对象。

用法：

```
MimeMultipart multipart=new MimeMultipart();
```

注：使用 MimeMultipart 对象的一个方法是 addBodyPart(), 它在电子邮件内容里添加 BodyPart(BodyPart 类在下面紧接着要介绍)对象，消息可以有很多部分，一个 BodyPart 可以代表一个部分。

#### (9) javax.mail.Internet.MimeBodyPart 类

MimeBodyPart 是 BodyPart 具体用于 mimeMessage 的一个子类。

MimeBodyPart 对象代表一个 MimeMessage 对象内容的一部分.每个 MimeBodyPart 被认为有两部分:

- ①一个 MIME 类型
- ②匹配这个类型的内容

用法:

```
MimeBodyPart mdp=new MimeBodyPart();
String text="Hello JavaMail!";
mdp.setContent(text,"text/plain");//定义 MIME 类型为 text/plain,并设置内容.
```

#### (10) javax.activation.DataHandler 类(包含在 JAF 中)

JavaMail API 不限制信息只为文本,任何形式的信息都可能作 MimeMessage 的一部分。除了文本信息,作为文件附件包含在电子邮件信息的一部分是很普遍的。JavaMail API 通过使用 DataHandler 对象,提供一个允许我们包含非文本 BodyPart 对象的简便方法.

用法:

```
DataHandler dh=new DataHandler(text,type);
mdp.setDatahandler(dh);//mdp 是一个 MimeBodyPart 对象
```

#### (11) javax.activation.FileDataSource 类(包含在 JAF 中)

一个 FileDataSource 对象可以表示本地文件和服务器可以直接访问的资源.一个本地文件可以通过创建一个新的 MimeBodyPart 对象附在一个 MimeMessage 对象上.

用法:

```
MimeMultipart mm=new MimeMultipart();
MimeBodyPart mdp=new MimeBodyPart();
FileDataSource fds=new FileDataSource("c:/exam.txt");
mdp.setDataHandler(new DataHandler(fds)); //设置数据源
mm.addBodyPart(mdp); //为当前消息 MimeMultipart 对象增加 MimeBodyPart
```

#### (12) javax.activation.URLDataSource 类(包含在 JAF 中)

远程资源, URL 不会指向它们,由一个 URLDataSource 对象表示。一个远程资源可以通过创建一个新 mimeBodyPart 对象附在一个 mimeMessage 对象上(同 FileDataSource 差不多)

用法:

与 FileDataSource 唯一不同的是数据源的设置:

```
URLDataSource uds=new URLDataSource(http://www.cnjsp.com/logo.gif);
```

## 示例代码

### Gmail 收发信

Gmail 目前已经启用了 POP3 和 SMTP 服务，与其他邮箱不同的是 Gmail 提供的 POP3 和 SMTP 是使用安全套接字层 SSL 的，因此常规的 JavaMail 程序是无法收发邮件的，下面是使用 JavaMail 如何收取 Gmail 邮件以及发送邮件的代码：

#### 1. 邮件收取

```

package lius.javamail.ssl;
import java.io.UnsupportedEncodingException;
import java.security.*;
import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeUtility;
/**
 * 用于收取 Gmail 邮件
 */
public class GmailFetch {

    public static void main(String argv[]) throws Exception {
        Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
        final String SSL_FACTORY = "javax.net.ssl.SSLSocketFactory";
        // Get a Properties object
        Properties props = System.getProperties();
        props.setProperty("mail.pop3.socketFactory.class", SSL_FACTORY);
        props.setProperty("mail.pop3.socketFactory.fallback", "false");
        props.setProperty("mail.pop3.port", "995");
        props.setProperty("mail.pop3.socketFactory.port", "995");
        //以下步骤跟一般的 JavaMail 操作相同
        Session session = Session.getDefaultInstance(props,null);
        //请将红色部分对应替换成你的邮箱帐号和密码
        URLName urln = new URLName("pop3","pop.gmail.com",995,null,
        "[邮箱帐号]", "[邮箱密码]");
        Store store = session.getStore(urln);
        Folder inbox = null;
        try {
            store.connect();
            inbox = store.getFolder("INBOX");
            inbox.open(Folder.READ_ONLY);
            FetchProfile profile = new FetchProfile();
            profile.add(FetchProfile.Item.ENVELOPE);
            Message[] messages = inbox.getMessages();
            inbox.fetch(messages, profile);
            System.out.println("收件箱的邮件数: " + messages.length);
            for (int i = 0; i < messages.length; i++) {
                //邮件发送者
                String from = decodeText(messages[i].getFrom()[0].toString());
                InternetAddress ia = new InternetAddress(from);
                System.out.println("FROM:" + ia.getPersonal()+(+ia.getAddress())++);
                //邮件标题
                System.out.println("TITLE:" + messages[i].getSubject());
                //邮件大小
                System.out.println("SIZE:" + messages[i].getSize());
                //邮件发送时间
                System.out.println("DATE:" + messages[i].getSentDate());
            }
        }
    }
}

```

```

} finally {
try {
inbox.close(false);
} catch (Exception e) {}
try {
store.close();
} catch (Exception e) {}
}
}

protected static String decodeText(String text)
throws UnsupportedEncodingException {
if (text == null)
return null;
if (text.startsWith("=?GB") || text.startsWith("=?gb"))
text = MimeUtility.decodeText(text);
else
text = new String(text.getBytes("ISO8859_1"));
return text;
}
}

```

## 2. 发送邮件

```

package lius.javamail.ssl;
import java.security.Security;
import java.util.Date;
import java.util.Properties;
import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
/**
 * 使用 Gmail 发送邮件
 */
public class GmailSender {
public static void main(String[] args) throws AddressException, MessagingException {
Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
final String SSL_FACTORY = "javax.net.ssl.SSLSocketFactory";
// Get a Properties object
Properties props = System.getProperties();
props.setProperty("mail.smtp.host", "smtp.gmail.com");
props.setProperty("mail.smtp.socketFactory.class", SSL_FACTORY);
props.setProperty("mail.smtp.socketFactory.fallback", "false");
props.setProperty("mail.smtp.port", "465");
props.setProperty("mail.smtp.socketFactory.port", "465");
props.put("mail.smtp.auth", "true");
final String username = "[邮箱帐号]";
final String password = "[邮箱密码]";
Session session = Session.getDefaultInstance(props, new Authenticator() {
protected PasswordAuthentication getPasswordAuthentication() {
return new PasswordAuthentication(username, password);
}});
// -- Create a new message --

```

```
Message msg = new MimeMessage(session);
// -- Set the FROM and TO fields --
msg.setFrom(new InternetAddress(username + "@mo168.com"));
msg.setRecipients(Message.RecipientType.TO,
InternetAddress.parse("[收件人地址]",false));
msg.setSubject("Hello");
msg.setText("How are you");
msg.setSentDate(new Date());
Transport.send(msg);

System.out.println("Message sent.");
}
}
```

## JavaMail 收取邮件属性配置（包括 Gmail、hotmail 等）

```

package com.tom.inq.util;

import java.util.Properties;

/**
 * A configuration of properties,See the server
 *
 * @author dengluxiao
 * @version $Id: PropertyManager.java,v 1.4.1 10/08/2009 14:05:23 dengluxiao $
 */
public final class PropertyManager {

    //private static boolean ssl;
    //private static boolean tls;

    //private static final String PATH = "config/mail.properties";

    private static final String SMTP = "smtp";

    private static Properties prop_send;
    private static Properties prop_send_gmail;
    private static Properties prop_send_hotmail;

    private static Properties prop_receive;
    private static Properties prop_receive_gmail;
    private static Properties prop_receive_hotmail;

    //-----
    // init every property
    //-----

    static
    {
        if (prop_send == null) prop_send = new Properties();
        if (prop_send_gmail == null) prop_send_gmail = new Properties();
        if (prop_send_hotmail == null) prop_send_hotmail = new Properties();

        if (prop_receive == null) prop_receive = new Properties();
        if (prop_receive_gmail == null) prop_receive_gmail = new Properties();
        if (prop_receive_hotmail == null) prop_receive_hotmail = new Properties();

        prop_send.put("mail.smtp.port", "25"); // default smtp
        prop_send.put("mail.Transport.protocol", SMTP);

        prop_send_gmail.putAll(prop_send); // add default smtp
        prop_send_gmail.put("mail.smtp.socketFactory.port", "465");
        prop_send_gmail.put("mail.smtp.socketFactory.fallback", "false");
        prop_send_gmail.put("mail.smtp.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");

        prop_send_hotmail.putAll(prop_send); // add default smtp
        prop_send_hotmail.put("mail.smtp.starttls.enable", "true");

        prop_receive_gmail.put("mail.pop3.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");
        prop_receive_gmail.put("mail.pop3.socketFactory.fallback", "false");
    }
}

```

```

prop_receive_gmail.put("mail.pop3.port", "995");
prop_receive_gmail.put("mail.pop3.socketFactory.port", "995");

// gmail equals hotmail when receive mail
prop_receive_hotmail.putAll(prop_receive_gmail);
}

-----
//choose read property file
-----

-----
// static {
// try {
// prop = new Properties();
// under two read properties method
//
prop.load(PropertyManager.class.getClassLoader().getResourceAsStream("com/tom/inq/util/mail.properties"));
// FileInputStream fis = new FileInputStream("config/mail.properties");
// prop.load(fis);
// fis.close();
// } catch (FileNotFoundException e) {
// e.printStackTrace();
// } catch (IOException e) {
// e.printStackTrace();
// }
// }

-----
// get send properties by ext ( mail user ext)
-----


public static Properties getSendProperty(String ext,String host)
{
    Properties prop = null;
    ext = ext.toLowerCase(); // ***
    if(ext.trim().indexOf("@gmail.com")!= -1) prop = prop_send_gmail;
    else if(ext.trim().indexOf("@live.cn")!= -1) prop = prop_send_hotmail;
    else if(ext.trim().indexOf("@msn.com")!= -1) prop = prop_send_hotmail;
    else if(ext.trim().indexOf("@live.com")!= -1) prop = prop_send_hotmail;
    else if(ext.trim().indexOf("@hotmail.com")!= -1) prop = prop_send_hotmail;
    else prop = prop_send;

    prop.put("mail.smtp.host", host);
    return prop;
}

-----
// get receive properties by ext ( mail user ext)
-----


public static Properties getReceiveProperty(String ext,String host)
{
    Properties prop = null;

    ext = ext.toLowerCase();

```

```
if (ext.trim().indexOf("@gmail.com")!= -1) prop = prop_receive_gmail;
else if (ext.trim().indexOf("@live.cn")!= -1) prop = prop_receive_hotmail;
else if (ext.trim().indexOf("@msn.com")!= -1) prop = prop_receive_hotmail;
else if (ext.trim().indexOf("@live.com")!= -1) prop = prop_receive_hotmail;
else if (ext.trim().indexOf("@hotmail.com")!= -1) prop = prop_receive_hotmail;
else prop = prop_receive;

return prop;
}

//-----
//-----
//-----
```

```
public static void main(String[] args)
{
    String host = "xxx";
    String ext = "xxx@hotmail.com";
    System.out.println(getSendProperty(ext, host).size());
    System.err.println(getReceiveProperty(ext, host).size());

}
```

```
}
```

## JavaMail 收取邮件 IMAP

```
package com.tom.inq imap;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

import javax.mail.AuthenticationFailedException;
import javax.mail.FetchProfile;
import javax.mail.Flags;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessageRemovedException;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.UIDFolder;
import javax.mail.search.HeaderTerm;
import javax.mail.search.SearchTerm;

import com.sun.mail.imap.IMAPFolder;
import com.sun.mail.imap.IMAPStore;
import com.tom.inq.bo.MailInfo;
import com.tom.inq.dao.MailDaoImpl;
import com.tom.inq.mail.Connector;
import com.tom.inq.po.Mail;
import com.tom.inq.po.MailGroup;
import com.tom.inq.util.ConfigKeys;
import com.tom.inq.util.PropertyManager;

/**
 * Handles the connection to the IMAP server.
 *
 * @author dengluxiao
 * @version $Id: IMPConnector.java,v 1.4.1 13/08/2009 16:00:23 dengluxiao $
 */
public class IMPConnector extends Connector
{
    private Mail mail;

    private boolean debug;

    private Session session;

    private IMAPStore store;

    private IMAPFolder folder;

    private static final String IMAP = "imap";

    //private Set<Mail> set = new HashSet<Mail>();
```

```

//need to synchronize the state of users and their e-mail message
private static Map<String, Set<String>> userstatussyncmap = new HashMap<String,
Set<String>>();

//multiple user authentication failure
private static Map<String, ArrayList<Date>> userauthentfailmap = new HashMap<String,
ArrayList<Date>>();


/**
 * @param mail
 * the {@link Mail} instance with connect server
 */
public void setMail(Mail mail) {
    this.mail = mail;
}

/**
 * Lists all available messages uids in the server
 *
 * @return Array of {@link Message}'s
 * @throws MessagingException
 */
public List<Mail> getUids(MailInfo b) throws MessagingException
{
    List<Mail> l = new ArrayList<Mail>();

    Message[] messages = folder.getMessages();
    FetchProfile profile = new FetchProfile();
    profile.add(UIDFolder.FetchProfileItem.UID);
    try { folder.fetch(messages, profile);
    } catch (MessageRemovedException e) {}

    String uid = "";
    for (int i = 0; i < messages.length; i++) {
        uid = folder.getUID((Message) messages[i])+"";
        l.add(getMailEntity(b, uid));
    }

    return l;
}

/**
 * set the Mail values
 *
 */
protected Mail getMailEntity(MailInfo b,String uid)
{
    Mail p = new Mail();

    p.setSessionid(b.getSessionid());
    p.setMail(b.getMail());
    p.setUid(uid);
    p.setTbl(b.getTbl());

    return p;
}

```

```

/**
 * Lists all available new messages in the server
 *
 * @return Array of {@link Message}'s
 */
public Map<String,Message> getNewMessagesMap(List<Mail> uids)
{
    Map<String,Message> messagesMap = new HashMap<String, Message>();
    try {

        Message[] messages = folder.getMessages();
        FetchProfile profile = new FetchProfile();
        profile.add(UIDFolder.FetchProfileItem.UID);
        try { folder.fetch(messages, profile);
        } catch (MessageRemovedException e) {}

        String uid = "";
        Message msg = null;
        for (int i = 0; i < messages.length; i++)
        {
            boolean flag = false;
            msg = (Message)messages[i];
            uid = folder.getUID(msg)+"";

            if (uids != null && uids.size() >0)
            {
                for (Iterator<Mail> it = uids.iterator(); it.hasNext();)
                {
                    if (uid.trim().equals(it.next().getUid())) flag = true;
                }
            }

            if (!flag)
            if (!isRemoved(msg))    messagesMap.put(uid, msg);
        }
    }
    catch (Exception e) {}

    return messagesMap;
}

/**
 * Lists all available new messages in the server
 * @override
 * @return Array of {@link Message}'s
 */
public Map<String,Message> getNewMessagesMap(Mail mail)
{
    Map<String,Message> messagesMap = new HashMap<String, Message>();
    try {

        Message[] messages = folder.getMessages();
        if(messages == null || messages.length == 0) return messagesMap;

        FetchProfile profile = new FetchProfile();
        profile.add(UIDFolder.FetchProfileItem.UID);
    }
}

```

```

        try { folder.fetch(messages, profile);
    } catch (MessageRemovedException e) {}

        long uid = 0;
        long uidd = 0;
        Message msg = null;
        long uidmax = Long.parseLong(mail.getUid());
        uidd = folder.getUID((Message)messages[messages.length-1]);

        if (uidd > uidmax) {
            for (int i = 0; i < messages.length; i++) {
                msg = (Message)messages[i];
                uid = folder.getUID(msg);
                if (uid > uidmax) {
                    if (!isRemoved(msg))    messagesMap.put(uid+"", msg);
                }
            }
        }
        catch (Exception e) { e.printStackTrace(); }

        return messagesMap;
    }

    /**
     * Comparison of UID, and synchronization.
     * @override
     * @return Array of {@link String}'s
     */
    public List<String> getSyncMessageUid(List<String> uids)
    {
        List<String> slist = new ArrayList<String>();
        try {
            Message[] messages = folder.getMessages();
            if(messages == null) return slist;

            FetchProfile profile = new FetchProfile();
            profile.add(UIDFolder.FetchProfileItem.UID);
            try { folder.fetch(messages, profile);
            } catch (MessageRemovedException e) {}

            Iterator<String> it = uids.iterator();
            while (it.hasNext()) {
                String id = it.next();
                boolean b = false;
                for (int i = 0; i < messages.length; i++) {
                    String uid = folder.getUID((Message)messages[i])+"";
                    if(id.equals(uid)) b = true;
                }
                if(!b) slist.add(id);
            }
        }
        catch (Exception e) { e.printStackTrace(); }

        return slist;
    }

    /**

```

```

* Opens a connection to the server. The method will try to retrieve the
* <i>INBOX</i> folder in <i>READ_WRITE</i> mode
*/
public boolean openConnection()
{
    boolean flag = false;
    try {

        Properties props = PropertyManager.getReceiveProperty(mail.getMail(),
mail.getHost());
        props.put("mail.imap.connectiontimeout", ConfigKeys.IMAP_CONNECTIONTIMEOUT);

        this.session = Session.getInstance(props, null);
        this.store = (IMAPStore) this.session.getStore(IMAP);
        this.store.connect(mail.getHost(), mail.getMail(), mail.getPassword());

        this.session.setDebug(debug);

        this.folder = (IMAPFolder) this.store.getFolder("INBOX");

        if (folder == null) throw new Exception("No Inbox");

        this.folder.open(Folder.READ_WRITE);

        flag = true;

        if ("0".equals(mail.getPasswordstate().trim()) && flag)
        {
            mail.setPasswordstate("1");
            new MailDaoImpl().updatePasswordState(mail);
            try {userauthentfailmap.remove(mail.getMail());} catch (Exception e) {}
        }

    } catch (AuthenticationFailedException e) {
        flag = false;
        this.doAuthentcationFailed();
    } catch (Exception e) {
        //e.printStackTrace();
        flag = false;
    }
    return flag;
}

/**
 * do AuthenticationFailedException
 */

protected void doAuthentcationFailed() {
    ArrayList<Date> authentList = null;
    try {
        authentList = userauthentfailmap.get(mail.getMail());
    } catch (NullPointerException e) {}

    if (authentList == null) {
        authentList = new ArrayList<Date>();
        authentList.add(new Date());
        userauthentfailmap.put(mail.getMail(), authentList);
    } else {
}

```

```

        authentList.add(new Date());
        if (authentList.size() >= 2){
            Date recorder = null;
            int count = 0;
            Date line = new Date(System.currentTimeMillis()-30*60*1000);
            for (Iterator<Date> it = authentList.iterator(); it.hasNext();)
            {
                recorder = it.next();
                if(line.before(recorder)) count++;
            }
            if(count >= 2)
            {
                mail.setPasswordstate("0");
                new MailDaoImpl().updatePasswordState(mail);
                userauthentfailmap.remove(mail.getMail());
            }
        }
    }

    public SearchTerm setSearchTerm(){
        return new HeaderTerm("X-Tmail-Type","IMAP");
    }

/*****
 * Search Term
*****/
public Message[] searchIMAPMessageByHeader() {
    try {
        SearchTerm term = new HeaderTerm("X-Tmail-Type","IMAP");
        Message[] msgs = this.folder.search(term);
        FetchProfile profile = new FetchProfile();
        profile.add(UIDFolder.FetchProfileItem.UID);
        try { folder.fetch(msgs, profile);
        } catch (MessageRemovedException e) {}

        return msgs;
    } catch (Exception exception) {}
    return null;
}

/*****
 * execute syncnization mail status
*****/
public MailGroup getNeedSyncStatusMailGroup(List<Mail> l){
    String oldid = "";
    MailGroup group = new MailGroup();
    Message[] messages = this.searchIMAPMessageByHeader();
    try {
        Iterator<Mail> it = l.iterator();
        while (it.hasNext()) {
            boolean b = false;
            Mail mail = (Mail) it.next();
            for (int i = 0; i < messages.length; i++) {
                Message message = messages[i];
                oldid = this.getHeaderValueByName(message, "X-Tmail-Uid");
                if (oldid.equals(mail.getHeader("X-Tmail-Uid")))
                    b = true;
            }
            if (!b)
                group.add(mail);
        }
    } catch (Exception exception) {}
    return group;
}

```

```

        if (mail.getUid().equals(olddid)) {
            b = true;
            String nowstatus = getChangedStatus(message, mail.getState());
            if (nowstatus.length() > 0) {
                Mail m = new Mail();
                setParam(m, mail, nowstatus);
                group.add(m);
            }
        }
    }
    if (!b) {
        Mail m = new Mail();
        setParam(m, mail, "D");
        group.add(m);
    }
}
} catch (Exception anyexception) {}
return group;
}

protected void setParam(Mail m1,Mail m2,String status)
{
    m1.setSessionid(m2.getSessionid());
    m1.setPassword(m2.getPassword());
    m1.setMail(m2.getMail());
    m1.setUid(m2.getUid());
    m1.setState(status);
    m1.setTbl(m2.getTbl());
}

protected boolean isChanged(Message message, String status) {
    String s = "";
    try {
        s = message.isSet(Flags.Flag.SEEN) ? "R" : "N";
    } catch (MessagingException e) {}

    return !s.equals(status);
}

protected String getChangedStatus(Message message, String status){
    String s = "";
    try {
        s = message.isSet(Flags.Flag.SEEN) ? "R" : "N";
    } catch (MessagingException e) {}

    if (s != null && !s.trim().equals(status)) return s;
    return "";
}

protected String getHeaderValueByName(Message message, String name) {
    String v = "";
    try {
        v = message.getHeader(name)[0];
    } catch (Exception e) { v = "";}
    return v;
}

/**

```

```
* Closes the connection to the server. Before finishing the
* communication channel, all messages are flagged for deletion.
*/
public void closeConnection()
{
    if (this.folder != null)
    {
        try { this.folder.close(true);
        } catch (Exception e) {}
    }
    if (this.store != null)
    {
        try { this.store.close();
        } catch (Exception e) {}
    }
}

@Override
public String toString()
{
    return new StringBuffer()
.append('[')
.append("com.tom.ing imap.IMPConnector-")
.append("sessionid=").append(this.mail.getSessionid())
.append("protocol=").append(IMAP)
.append(", mail=").append(this.mail.getMail())
.append(", password=").append(this.mail.getPassword())
.append(']')
.appendToString();
}
}
```

## JavaMail 发送邮件 [代码] MailSender.java

```
/*
 * 邮件发送组件类
 * 该类需要三个 jar 文件: mail.jar,activation.jar,htmlparser.jar
 * @version 1.0
 */
package lius.util.mail;

import java.io.File;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.Date;
import java.util.Properties;

import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.mail.Authenticator;
import javax.mail.BodyPart;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.PasswordAuthentication;
import javax.mail.SendFailedException;
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;

import org.htmlparser.Node;
import org.htmlparser.Parser;
import org.htmlparser.tags.ImageTag;
import org.htmlparser.util.ParserException;

/**
 * 邮件发送组件,具体的使用方法参照该类的 main 方法
 */
public abstract class MailSender extends Authenticator{

    private String username = null;          //邮件发送帐号用户名
    private String userpasswd = null; //邮件发送帐号用户口令
    protected BodyPart messageBodyPart = null;
    protected Multipart multipart = new MimeMultipart("related");
    protected MimeMessage mailMessage = null;
    protected Session mailSession = null;
    protected Properties mailProperties = System.getProperties();
    protected InternetAddress mailFromAddress = null;
    protected InternetAddress mailToAddress = null;
    protected Authenticator authenticator = null;
    protected String mailSubject = "";
    protected Date mailSendDate = null;

    /**
     * 构造函数
     * @param smtpHost
     */
```

```

* @param username
* @param password
*/
protected MailSender(String smtpHost, String username, String password) {
    this.username = username;
    this.userpasswd = password;
    mailProperties.put("mail.smtp.host", smtpHost);
    mailProperties.put("mail.smtp.auth", "true"); //设置 smtp 认证，很关键的一句
    mailSession = Session.getDefaultInstance(mailProperties, this);
    mailMessage = new MimeMessage(mailSession);
    messageBodyPart = new MimeBodyPart();
}
/**
 * 构造一个纯文本邮件发送实例
 * @param smtpHost
 * @param username
 * @param password
 * @return
*/
public static MailSender getTextMailSender(String smtpHost, String username, String password)
{
    return new MailSender(smtpHost,username,password) {
        public void setMailContent(String mailContent) throws MessagingException {
            messageBodyPart.setText(mailContent);
            multipart.addBodyPart(messageBodyPart);
        }
    };
}
/**
 * 构造一个超文本邮件发送实例
 * @param smtpHost
 * @param username
 * @param password
 * @return
*/
public static MailSender getHtmlMailSender(String smtpHost, String username, String password)
{
    return new MailSender(smtpHost,username,password) {
        private ArrayList arrayList1 = new ArrayList();
        private ArrayList arrayList2 = new ArrayList();

        public void setMailContent(String mailContent) throws MessagingException {
            String htmlContent = getContent("";
                messageBodyPart.setHeader("Content-ID", contentId);
                messageBodyPart.setFileName((String) arrayList1.get(i));
            }
        }
    };
}

```

```

        multipart.addBodyPart(messageBodyPart);
    }
}

//处理要发送的 html 文件，主要是针对 html 文件中的图片
private String getContent(String searchString, String mailContent) {
    try {
        Parser parser = Parser.createParser(new String(mailContent.getBytes(),
ISO8859_1));
        Node[] images = parser.extractAllNodesThatAre(ImageTag.class);
        for(int i=0;i<images.length;i++) {
            ImageTag imgTag = (ImageTag) images[i];
            if(!imgTag.getImageURL().toLowerCase().startsWith("http://"))
                arrayList1.add(imgTag.getImageURL());
        }
    } catch (UnsupportedEncodingException e1) {
    } catch (ParserException e) {}
    String afterReplaceStr = mailContent;
    //在 html 文件中用"cid:"+Content-ID 来替换原来的图片链接
    for (int m = 0; m < arrayList1.size(); m++) {
        arrayList2.add(createRandomStr());
        String addString = "cid:" + (String) arrayList2.get(m);
        afterReplaceStr = mailContent.replaceAll(
            (String) arrayList1.get(m), addString);
    }
    return afterReplaceStr;
}

//产生一个随机字符串，为了给图片设定 Content-ID 值
private String createRandomStr() {
    char[] randomChar = new char[8];
    for (int i = 0; i < 8; i++) {
        randomChar[i] = (char) (Math.random() * 26 + 'a');
    }
    String replaceStr = new String(randomChar);
    return replaceStr;
}

private final static String ISO8859_1 = "8859_1";
};

/**
 * 用于实现邮件发送用户验证
 * @see javax.mail.Authenticator#getPasswordAuthentication
 */
protected PasswordAuthentication getPasswordAuthentication() {
    return new PasswordAuthentication(username, userpasswd);
}

/**
 * 设置邮件标题
 * @param mailSubject
 * @throws MessagingException
 */
public void setSubject(String mailSubject) throws MessagingException {
    this.mailSubject = mailSubject;
    mailMessage.setSubject(mailSubject);
}

```

```

/**
 * 所有子类都需要实现的抽象方法，为了支持不同的邮件类型
 * @param mailContent
 * @throws MessagingException
 */
protected abstract void setMailContent(String mailContent) throws MessagingException;

/**
 * 设置邮件发送日期
 * @param sendDate
 * @throws MessagingException
 */
public void setSendDate(Date sendDate) throws MessagingException {
    this.mailSendDate = sendDate;
    mailMessage.setSentDate(sendDate);
}

/**
 * 设置邮件发送附件
 * @param attachmentName
 * @throws MessagingException
 */
public void setAttachments(String attachmentName) throws MessagingException {
    messageBodyPart = new MimeBodyPart();
    DataSource source = new FileDataSource(attachmentName);
    messageBodyPart.setDataHandler(new DataHandler(source));
    int index = attachmentName.lastIndexOf(File.separator);
    String attachmentRealName = attachmentName.substring(index + 1);
    messageBodyPart.setFileName(attachmentRealName);
    multipart.addBodyPart(messageBodyPart);
}

/**
 * 设置发件人地址
 * @param mailFrom
 * @throws MessagingException
 */
public void setMailFrom(String mailFrom) throws MessagingException {
    mailFromAddress = new InternetAddress(mailFrom);
    mailMessage.setFrom(mailFromAddress);
}

/**
 * 设置收件人地址，收件人类型为 to,cc,bcc(大小写不限)
 * @param mailTo 邮件接收者地址
 * @param mailType 值为 to,cc,bcc
 */
public void setMailTo(String[] mailTo, String mailType) throws Exception {
    for (int i = 0; i < mailTo.length; i++) {
        mailToAddress = new InternetAddress(mailTo[i]);
        if (mailType.equalsIgnoreCase("to")) {
            mailMessage.addRecipient(Message.RecipientType.TO, mailToAddress);
        } else if (mailType.equalsIgnoreCase("cc")) {
            mailMessage.addRecipient(Message.RecipientType.CC, mailToAddress);
        } else if (mailType.equalsIgnoreCase("bcc")) {
            mailMessage.addRecipient(Message.RecipientType.BCC, mailToAddress);
        } else {
}
}

```

```

        throw new Exception("Unknown mailType: " + mailType + "!");
    }
}
/**
 * 开始发送邮件
 * @throws MessagingException
 * @throws SendFailedException
 */
public void sendMail() throws MessagingException, SendFailedException {
    if (mailToAddress == null)
        throw new MessagingException("请你必须你填写收件人地址!");
    mailMessage.setContent(multipart);
    Transport.send(mailMessage);
}

/**
 * 邮件发送测试
 * @param args
 */
public static void main(String args[]) {
    String mailHost = "smtp.163.com"; //发送邮件服务器地址
    String mailUser = "user1"; //发送邮件服务器的用户帐号
    String mailPassword = "password1"; //发送邮件服务器的用户密码
    String[] toAddress = {"user1@163.com"};
    //使用超文本格式发送邮件
    MailSender sendmail = MailSender.getHtmlMailSender(mailHost, mailUser, mailPassword);
    //使用纯文本格式发送邮件
    //MailSender sendmail = MailSender.getTextMailSender(mailHost, mailUser, mailPassword);
    try {
        sendmail.setSubject("邮件发送测试");
        sendmail.setSendDate(new Date());
        String content = "<H1> 你 好 , 中 国 </H1><img src=\"http://www.javayou.com/images/logo.gif\">";
        //请注意如果是本地图片比如使用斜杠作为目录分隔符,如下所示
        content+="<img src=\"D:/EclipseM7/workspace/JDlog/dlog/images/rss200.png\"/>";
        sendmail.setMailContent(content);
        sendmail.setAttachments("E:\\TOOLS\\pm_sn.txt");
        sendmail.setMailFrom("user1@163.com");
        sendmail.setMailTo(toAddress, "to");
        //sendmail.setMailTo(toAddress, "cc");//设置抄送给...
        //开始发送邮件
        System.out.println("正在发送邮件, 请稍候.....");
        sendmail.sendMail();
        System.out.println("恭喜你, 邮件已经成功发送!");
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
}

```

## JavaMail 收取邮件 POP3

```
package com.tom.inq.pop;

import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Properties;

import javax.mail.AuthenticationFailedException;
import javax.mail.FetchProfile;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessageRemovedException;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.UIDFolder;

import com.sun.mail.pop3.POP3Folder;
import com.sun.mail.pop3.POP3Store;
import com.tom.inq.bo.MailInfo;
import com.tom.inq.dao.MailDaoImpl;
import com.tom.inq.mail.Connector;
import com.tom.inq.po.Mail;
import com.tom.inq.util.ConfigKeys;
import com.tom.inq.util.PropertyManager;

/**
 * Handles the connection to the POP3 server.
 *
 * @author dengluxiao
 * @version $Id: POPConnector.java,v 1.4.1 13/08/2009 16:00:23 dengluxiao $
 */
public class POPConnector extends Connector {

    private Mail mail;

    private boolean debug;

    private POP3Store store;

    private Session session;

    private POP3Folder folder;

    private static final String POP3 = "pop3";

    private static Map<String, ArrayList<Date>> authentFailMap = new HashMap<String,
ArrayList<Date>>();

    /**
     * @param mail
     * the {@link Mail} instance with connect server
     */
    public void setMail(Mail mail) {
```

```

        this.mail = mail;
    }

    /**
     * Lists all available messages uids on pop server
     *
     * @return Array of {@link Message}'s
     * @throws MessagingException
     */
    public List<Mail> getUids(MailInfo b) throws MessagingException
    {
        List<Mail> l = new ArrayList<Mail>();

        Message[] messages = folder.getMessages();
        FetchProfile profile = new FetchProfile();
        profile.add(UIDFolder.FetchProfileItem.UID);
        try { folder.fetch(messages, profile);
        } catch (MessageRemovedException e) {}

        String uid = "";
        for (int i = 0; i < messages.length; i++) {
            uid = folder.getUID((Message) messages[i]);
            l.add(getMailEntity(b, uid));
        }

        return l;
    }

    /**
     * set the Mail values
     * @return
     */
    protected Mail getMailEntity(MailInfo b, String uid)
    {
        Mail p = new Mail();

        p.setSessionid(b.getSessionid());
        p.setMail(b.getMail());
        p.setUid(uid);
        p.setTbl(b.getTbl());

        return p;
    }

    /**
     * Lists all available new messages in the pop server
     *
     * @return Array of {@link Message}'s
     */
    public Map<String,Message> getNewMessagesMap(List<Mail> uids)
    {
        Map<String,Message> messagesMap = new HashMap<String, Message>();
        try {

            Message[] messages = folder.getMessages();
            FetchProfile profile = new FetchProfile();
            profile.add(UIDFolder.FetchProfileItem.UID);
            try { folder.fetch(messages, profile);

```

```

        } catch (MessageRemovedException e) {}

        String uid = "";
        Message msg = null;
        for (int i = 0; i < messages.length; i++)
        {
            boolean flag = false;
            msg = (Message)messages[i];
            uid = folder.getUID(msg);
            if (uids != null && uids.size() >0)
            {
                for (Iterator<Mail> it = uids.iterator(); it.hasNext());
                {
                    if (uid.trim().equals(it.next().getUid())) flag = true;
                }
            }

            if (!flag)
                if (!isRemoved(msg))    messagesMap.put(uid, msg);
        }

    }
    catch (Exception e) {}

    return messagesMap;
}

/**
 * Opens a connection to the pop server. The method will try to retrieve the
 * <i>INBOX</i> folder in <i>READ_WRITE</i> mode
 */
public boolean openConnection()
{
    boolean flag = false;
    try {

        Properties props      = PropertyManager.getReceiveProperty(mail.getMail(),
mail.getHost());
        props.put("mail.pop3.connectiontimeout", ConfigKeys.POP3_CONNECTIONTIMEOUT);
        props.put("mail.pop3.timeout", ConfigKeys.POP3_TIMEOUT);

        this.session = Session.getInstance(props, null);
        this.store = (POP3Store) this.session.getStore(POP3);
        this.store.connect(mail.getHost(), mail.getMail(), mail.getPassword());

        this.session.setDebug(debug);

        this.folder = (POP3Folder) this.store.getFolder("INBOX");

        if (folder == null) throw new Exception("No Inbox");

        this.folder.open(Folder.READ_WRITE);

        flag = true;

        if ("0".equals(mail.getPasswordstate().trim()))
    {

```

```

        mail.setPasswordstate("1");
        new MailDaoImpl().updatePasswordState(mail);
        try {authentFailMap.remove(mail.getMail());} catch (Exception e) {}
    }

} catch (AuthenticationFailedException e) {
    flag = false;
    this.doAuthentcationFailed();
} catch (Exception e) {
    flag = false;
}

return flag;
}

/**
 * do AuthenticationFailedException
 *
 */

protected void doAuthentcationFailed() {
    ArrayList<Date> authentList = null;

    try {
        authentList = authentFailMap.get(mail.getMail());
    } catch (NullPointerException e) {}

    if (authentList == null) {
        authentList = new ArrayList<Date>();
        authentList.add(new Date());
        authentFailMap.put(mail.getMail(), authentList);
    } else {
        authentList.add(new Date());
        if (authentList.size() >= 2) {
            Date recorder = null;
            int count = 0;
            Date line =new Date(System.currentTimeMillis()-30*60*1000);
            for (Iterator<Date> it = authentList.iterator(); it.hasNext();) {
                recorder = it.next();
                if(line.before(recorder)) count++;
            }
            if(count >= 2) {
                mail.setPasswordstate("0");
                new MailDaoImpl().updatePasswordState(mail);
                authentFailMap.remove(mail.getMail());
            }
        }
    }
}

/**
 * Closes the connection to the pop server. Before finishing the
 * communication channel, all messages are flaged for deletion.
 */
public void closeConnection()
{
}

```

```
    if (this.folder != null)
    {
        try { this.folder.close(true);
        } catch (Exception e) {}
    }
    if (this.store != null)
    {
        try { this.store.close();
        } catch (Exception e) {}
    }
}

@Override
public String toString()
{
    return new StringBuffer()
.append('[')
.append("com.tom.inq.pop.POPConnector-")
.append("sessionid=").append(this.mail.getSessionid())
.append("protocol=").append(POP3)
.append(", mail=").append(this.mail.getMail())
.append(", password=").append(this.mail.getPassword())
.append(']')
.appendToString();
}
}
```

## GmailFetch 收取 Gmail 邮件

```

package lius.javamail.ssl;

import java.io.UnsupportedEncodingException;
import java.security.*;
import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeUtility;

/**
 * 用于收取 Gmail 邮件
 */
public class GmailFetch {

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

        Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
        final String SSL_FACTORY = "javax.net.ssl.SSLSocketFactory";

        // Get a Properties object
        Properties props = System.getProperties();
        props.setProperty("mail.pop3.socketFactory.class", SSL_FACTORY);
        props.setProperty("mail.pop3.socketFactory.fallback", "false");
        props.setProperty("mail.pop3.port", "995");
        props.setProperty("mail.pop3.socketFactory.port", "995");

        //以下步骤跟一般的 JavaMail 操作相同
        Session session = Session.getDefaultInstance(props,null);

        //请将红色部分对应替换成你的邮箱帐号和密码
        URLName urln = new URLName("pop3","pop.gmail.com",995,null,
            "[邮箱帐号]", "[邮箱密码]");
        Store store = session.getStore(urln);
        Folder inbox = null;
        try {
            store.connect();
            inbox = store.getFolder("INBOX");
            inbox.open(Folder.READ_ONLY);
            FetchProfile profile = new FetchProfile();
            profile.add(FetchProfile.Item.ENVELOPE);
            Message[] messages = inbox.getMessages();
            inbox.fetch(messages, profile);
            System.out.println("收件箱的邮件数: " + messages.length);
            for (int i = 0; i < messages.length; i++) {
                //邮件发送者
                String from = decodeText(messages[i].getFrom()[0].toString());
                InternetAddress ia = new InternetAddress(from);
                System.out.println("FROM:" + ia.getPersonal()+'('+ia.getAddress()+')');
                //邮件标题
                System.out.println("TITLE:" + messages[i].getSubject());
                //邮件大小
                System.out.println("SIZE:" + messages[i].getSize());
                //邮件发送时间
                System.out.println("DATE:" + messages[i].getSentDate());
            }
        } finally {
    }
}

```

```

        try {
            inbox.close(false);
        } catch (Exception e) {}
        try {
            store.close();
        } catch (Exception e) {}
    }
}

protected static String decodeText(String text)
throws UnsupportedEncodingException {
if (text == null)
    return null;
if (text.startsWith("=?GB") || text.startsWith("=?gb"))
    text = MimeUtility.decodeText(text);
else
    text = new String(text.getBytes("ISO8859_1"));
return text;
}
}

```

## Gmail Sender 发送 Gmail 邮件

```

package lius.javamail.ssl;

import java.security.Security;
import java.util.Date;
import java.util.Properties;

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

/**
 * 使用 Gmail 发送邮件
 */
public class GmailSender {

    public static void main(String[] args) throws AddressException, MessagingException {
        Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
        final String SSL_FACTORY = "javax.net.ssl.SSLSocketFactory";
        // Get a Properties object
        Properties props = System.getProperties();
        props.setProperty("mail.smtp.host", "smtp.gmail.com");
        props.setProperty("mail.smtp.socketFactory.class", SSL_FACTORY);
        props.setProperty("mail.smtp.socketFactory.fallback", "false");
        props.setProperty("mail.smtp.port", "465");
        props.setProperty("mail.smtp.socketFactory.port", "465");
        props.put("mail.smtp.auth", "true");
        final String username = "[邮箱帐号]";
        final String password = "[邮箱密码]";
    }
}

```

```
Session session = Session.getDefaultInstance(props, new Authenticator(){
    protected PasswordAuthentication getPasswordAuthentication() {
        return new PasswordAuthentication(username, password);
    }
};

// -- Create a new message --
Message msg = new MimeMessage(session);

// -- Set the FROM and TO fields --
msg.setFrom(new InternetAddress(username + "@mo168.com"));
msg.setRecipients(Message.RecipientType.TO,
    InternetAddress.parse("[收件人地址]",false));
msg.setSubject("Hello");
msg.setText("How are you");
msg.setSentDate(new Date());
Transport.send(msg);

System.out.println("Message sent.");
}
}
```