目 录

1 前言

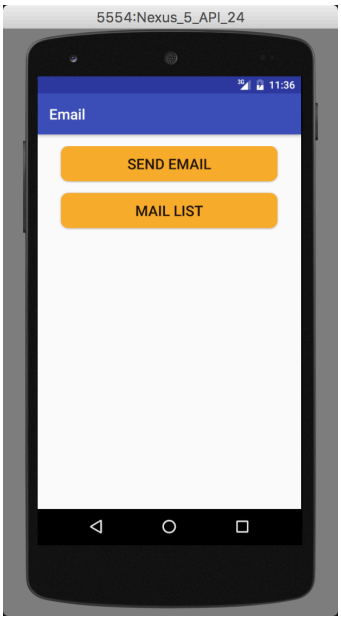
这篇报告是用来讲述自己实现的 android 简单邮件客户端有什么功能。我觉得与其把它当作一篇报告，不如把它当作一篇技术文档。阅读者如果有一定的 Java 基础和 android 基础，就可以根据这篇文章写出一个简单邮件客户端。

2 功能介绍

2.1 邮箱登陆

我实现的客户端登录界面如下图所示：

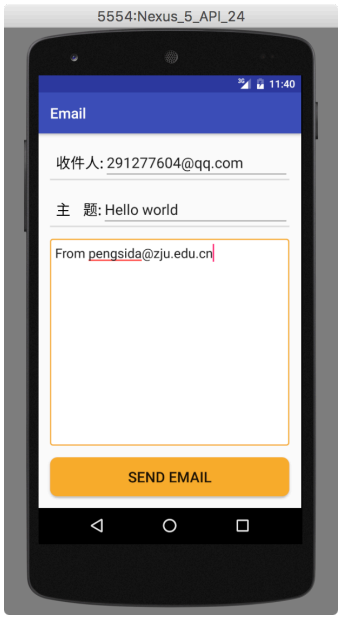
如果输入密码错误，将如左图所示，如果输入密码正确，将进入右图所示的界面：



2.2 发邮件

在上述视图中，点击“SEND EMAIL”的按钮，就可以进入发邮件的视图，如下图所示：

比如发送一封文本邮件，发给“291277604@qq.com”，主题为“Hello world”，内容为“From pengsida@zju.edu.cn”，随后我的 QQ 邮箱就收到了这封邮件，如下图所示：



2.3 收邮件

在主界面中，点击“MAIL LIST”的按钮，就可以查看邮件列表，如下图所示：

我的邮件客户端可以查看文本格式的邮件和多图文格式的邮件，文本格式的邮件和多图文格式的邮件如下所示：

3 功能的实现

因为代码比较多，有两千多行，接下来我采取的策略是，讲述自己实现相应功能的方法并贴出关键的代码。整个文档看下来，只会了解大体的实现框架，如果想了解细节，需要去看附件中完整的代码。

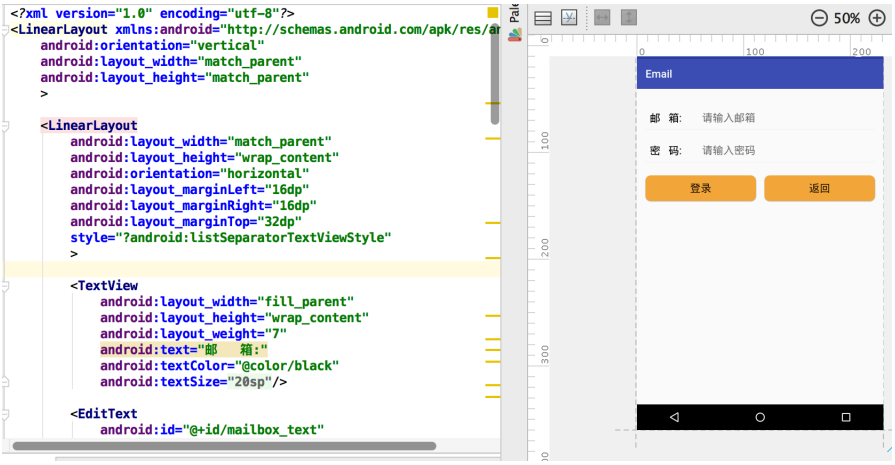
3.1 验证邮箱密码的实现

实现登陆邮箱的功能有如下步骤：

1. 写一个登录界面。
2. 通过 JavaMail 来进行邮箱密码的验证。
3. 编写一个控制层，从而可以让用户可以交互使用，其中关键的组件是两个输入框和一个按钮。

3.1.1 首先画一个登录界面

android 画一个界面还是比较简单的，只要编辑相应的代码，就能得到相应的界面。源码和相应的效果图如下所示，因为考虑到源码较多，这里只截取了一部分，如果想看完整的代码，可以查看附件：



3.1.2 进行邮箱密码的验证

对于这一功能，我的基本想法是通过邮箱和密码连接 smtp 服务器，如果连接成功，就说明密码正确并登入主界面，如果不能连接成功，就说明密码错误并提示错误信息。

我使用了 JavaMail 中的 Session 类、Transport 类来验证邮箱。

Session 类是 JavaMail API 的主要类，它不创建子类。Session 对象充当连接工厂的 JavaMail API，它可以同时处理配置设置和身份验证。创建 Session 对象的一种方法是基于编程方法，可以在其中使用的 java.util.Properties 对象来覆盖一些默认信息，如邮件服务器名，用户名，密码，那可以是其他信息整个应用程序共享。如：Session session = Session.getDefaultInstance(new Properties());

Transport 类用来作为消息传输机制。这个类通常使用 SMTP 协议来发送消息。你可以通过只调用静态的 send () 方法使用该类的默认版本：Transport.send(message);

实现这项功能的关键代码如下：

```
1    Session session = Session.getDefaultInstance(new Properties());
2
3    try
4    {
5        Transport transport = session.getTransport("smtp");
6        String mailboxKey = resolve(mailbox);
7
8        // smtpServerList是我的邮件客户端可以支持的邮件服务器
9        // 目前可以支持qq邮箱和浙大邮箱
10       // 要支持一个邮箱，只需要在smtpServerList中添加相应的smtp服务器即可
```

```
11          transport.connect(smtpServerList.get(mailboxKey), Integer.parseInt(portList
                .get(mailboxKey)), mailbox, passwd);
12          return "right";
13      }
14      catch (AuthenticationFailedException ea)
15      {
16          return "authenticate failed";
17      }
18      catch (MessagingException em)
19      {
20          return "wrong";
21      }
```

3.1.3 编写一个控制层

这里定义了视图层的三个组件：邮箱文本框、密码文本框和登录按钮。邮箱文本框用于输入邮箱账号，密码文本框用于输入密码，登陆按钮用于测试密码是否正确。

实现邮箱文本框和密码文本框的关键代码如下：

```
1   mailboxEditText.addTextChangedListener(new TextWatcher() {
2       ...
3
4       @Override
5       public void onTextChanged(CharSequence s, int start, int before, int count)
6       {
7           mailbox = s.toString();
8       }
9
10      ...
11  });
12
13  passwdEditText.addTextChangedListener(new TextWatcher() {
14      ...
15
16      @Override
17      public void onTextChanged(CharSequence s, int start, int before, int count)
18      {
19          passwd = s.toString();
20      }
21
22      ...
23  });
```

实现登录按钮的关键代码如下：

```
1   loginButton.setOnClickListener(new View.OnClickListener() {
2       @Override
3       public void onClick(View v)
4       {
5           ...
6           new VerifyPasswd().execute();
7       }
```

```
    });
```

上面的 VerifyPasswd 类是一个异步线程类，因为 android 不允许主线程使用网络，所以只能另开一个异步线程，如下所示：

```
    private class VerifyPasswd extends AsyncTask<Void, Void, String>
    {
        @Override
        protected String doInBackground(Void... params)
        {
            Session session = Session.getDefaultInstance(new Properties());

            try
            {
                Transport transport = session.getTransport("smtp");
                String mailboxKey = resolve(mailbox);
                transport.connect(smtpServerList.get(mailboxKey), Integer.parseInt(
                    portList.get(mailboxKey)), mailbox, passwd);
                return "right";
            }
            catch (AuthenticationFailedException ea)
            {
                return "authenticate failed";
            }
            catch (MessagingException em)
            {
                return "wrong";
            }
        }

        @Override
        protected void onPostExecute(String text)
        {
            if (text.equals("right"))
            {
                Map<String, String> keys = new HashMap<String, String>();
                putIntentData(keys);
                Intent i = MainActivity.newIntent(LoginActivity.this, keys);
                startActivity(i);
            }
            else
                Toast.makeText(LoginActivity.this, text, Toast.LENGTH_SHORT).show()
                    ;
        }
    }
```

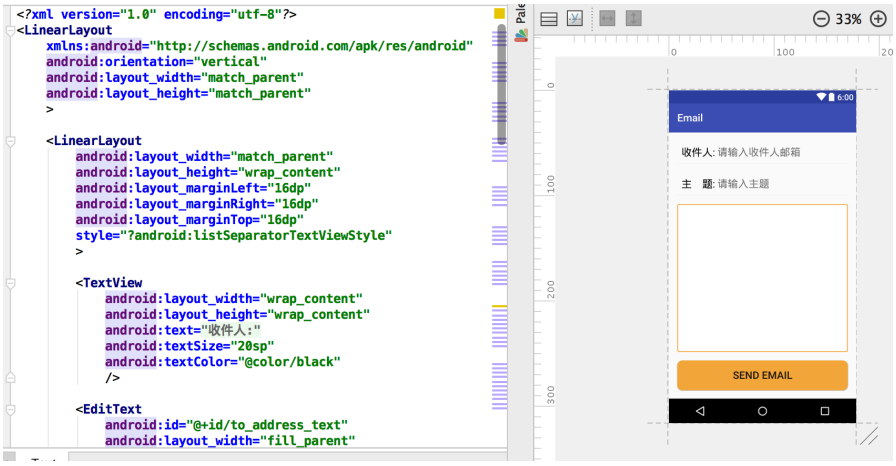
3.2 发邮件的实现

实现发邮件的功能有如下步骤：

1. 写一个发邮件的界面。
2. 通过 JavaMail 来进行邮件的发送。
3. 编写一个控制层，从而可以让用户可以交互使用，其中关键的组件是三个输入框和一个按钮。

3.2.1 发邮件界面的实现

部分代码和界面如下图所示：



3.2.2 通过 JavaMail 来实现发送邮件的功能

我使用了 JavaMail 中的 Session 类、Authenticator 类、Message 类、Address 类和 Transport 类来实现发送邮件的功能。

Authenticator 类表示懂得如何获得认证的网络连接的对象。它是一个抽象类。您可以创建一个子类 PasswordAuthentication，通过用户名和密码给它的构造。代码实现如下：

```
1   public class MyAuthenticator extends Authenticator
2   {
3       private String userName;
4       private String password;
5
6       public MyAuthenticator(String userName, String password)
7       {
8           this.userName = userName;
9           this.password = password;
10      }
```

```
11
12          @Override
13          protected PasswordAuthentication getPasswordAuthentication()
14          {
15              // 创建一个子类PasswordAuthentication，通过用户名和密码给它的构造
16              return new PasswordAuthentication(userName, password);
17          }
18
19          ...
20      }
```

我们通过 Authenticator 类来构造一个 Session 对象：

```
1       // MailSenderInfo是我自定义的一个类
2       // 它用于存放发送邮箱所需要的信息
3       public static boolean sendTextMail(MailSenderInfo mailInfo)
4       {
5           MyAuthenticator authenticator = null;
6           if(mailInfo.isValidate())
7               authenticator = new MyAuthenticator(mailInfo.getUserName(), mailInfo.
                    getPassword());
8
9           Session sendMailSession = Session.getInstance(mailInfo.getProperties(),
                authenticator);
10          ...
11      }
```

Address 类是一个抽象类。因此，它的子类 javax.mail.internet.InternetAddress 类大多被使用，它用于表示邮件发送中的地址对象。Address 可以通过电子邮件地址来创建：

```
1       // 创建收件箱和发件箱的地址对象
2       Address from = new InternetAddress(mailInfo.getFromAddress());
3       Address to = new InternetAddress(mailInfo.getToAddress());
```

Message 类是一个抽象类。因此，它的子类 javax.mail.internet.MimeMessage 类大多使用，它用于表示邮件这个对象。我们要对其设置发件人、收件人、邮件主题、发件日期、邮件内容，代码如下：

```
1       Message mailMessage = new MimeMessage(sendMailSession);
2       Address from = new InternetAddress(mailInfo.getFromAddress());
3       // 设置发件人
4       mailMessage.setFrom(from);
5       Address to = new InternetAddress(mailInfo.getToAddress());
6       // 设置收件人
7       mailMessage.setRecipient(Message.RecipientType.TO, to);
8       // 设置邮件主题
9       mailMessage.setSubject(mailInfo.getSubject());
10      // 设置发件日期
11      mailMessage.setSentDate(new Date());
12      String mailContent = mailInfo.getContent();
13      // 设置邮件内容
14      mailMessage.setText(mailContent);
```

最后使用 Transport 类发送邮件：

```
Transport.send(mailMessage);
```

完整代码如下：

```
public static boolean sendTextMail(MailSenderInfo mailInfo)
{
    MyAuthenticator authenticator = null;
    if(mailInfo.isValidate())
        authenticator = new MyAuthenticator(mailInfo.getUserName(), mailInfo.
            getPassword());

    Session sendMailSession = Session.getInstance(mailInfo.getProperties(),
        authenticator);

    try {
        Message mailMessage = new MimeMessage(sendMailSession);
        Address from = new InternetAddress(mailInfo.getFromAddress());
        mailMessage.setFrom(from);
        Address to = new InternetAddress(mailInfo.getToAddress());
        mailMessage.setRecipient(Message.RecipientType.TO, to);
        mailMessage.setSubject(mailInfo.getSubject());
        mailMessage.setSentDate(new Date());
        String mailContent = mailInfo.getContent();
        mailMessage.setText(mailContent);

        Transport.send(mailMessage);
        return true;
    }
    catch (MessagingException ex)
    {
        ex.printStackTrace();
    }

    return false;
}
```

3.2.3 编写一个控制层

这里定义了视图层的四个组件：收件人文本框、主题文本框、内容文本框和发件按钮。

实现收件人文本框、主题文本框和内容文本框的关键代码如下：

```
toAddressEditText.addTextChangedListener(new TextWatcher() {
    ...
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count)
    {
        toAddress = s.toString();
    }
    ...
```

```
    });

    subjectEditText.addTextChangedListener(new TextWatcher() {
        ...
        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count)
        {
            subject = s.toString();
        }
        ...
    });

    contentEditText.addTextChangedListener(new TextWatcher() {
        ...
        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count)
        {
            content = s.toString();
        }
        ...
    });
```

发件按钮的实现如下：

```
    sendEmailButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v)
        {
            new Thread(sendEmail).start();
            finish();
        }
    });
```

其中 sendEmail 是一个用于发送邮件的线程：

```
    Runnable sendEmail = new Runnable() {
        @Override
        public void run() {
            Looper.prepare();
            MailSenderInfo mailSenderInfo = new MailSenderInfo();
            mailSenderInfo.setMailServerHost(smtpServer);
            mailSenderInfo.setMailServerPort("25");
            mailSenderInfo.setValidate(true);
            mailSenderInfo.setUserName(mailbox);
            mailSenderInfo.setPassword(passwd);
            mailSenderInfo.setFromAddress(mailbox);
            mailSenderInfo.setToAddress(toAddress);
            mailSenderInfo.setSubject(subject);
            mailSenderInfo.setContent(content);
            MailSender.sendTextMail(mailSenderInfo);
        }
    };
```

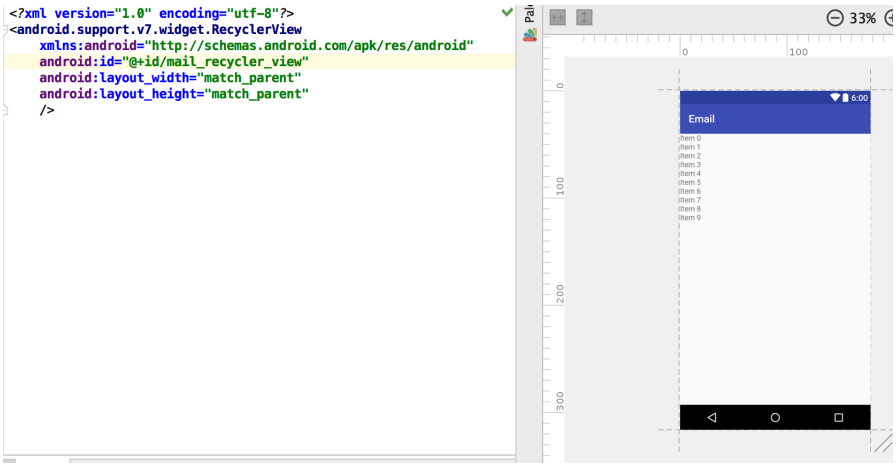
3.3 收邮件的实现

实现收邮件的功能有如下步骤：

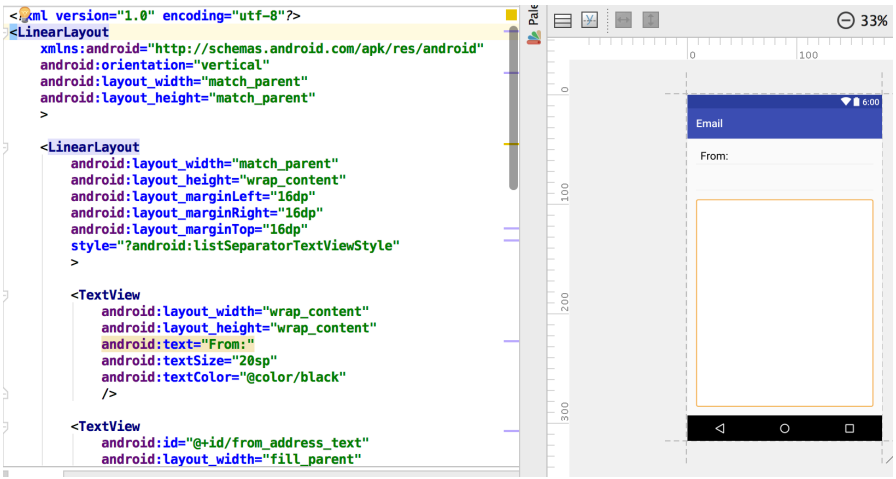
1. 写一个邮件列表的界面和一个邮件明细界面。
2. 通过 JavaMail 来进行邮件的接收。
3. 编写一个控制层，让用户可以看到邮件列表和查看特定的邮件。

3.3.1 写一个邮件列表的界面和一个邮件明细界面

邮件列表的界面是用 RecyclerView 实现的，这是一个比较高级的组件，有兴趣的读者可以深入去了解一下。它的功能实现会在控制层交代，这里只是画一个界面：



邮件明细界面就相对简单一些，如下所示：



3.3.2 通过 JavaMail 来进行邮件的接收

我使用了 JavaMail 中的 Folder 类、Store 类、Message 类。这里的 Folder 类和 Store 类主要用于打开收件箱，为了完成邮件的接收和显示，我们还需要解析 Message 对象，这是这一部分的难点。

Store 类是一个抽象类，模型信息存储和访问协议，用于存储和检索信息。子类提供实际的实现。存储扩展服务类，它提供命名商店，连接到存储，并听取连接事件很多常见的方法。客户获得通过获得它实现了数据库访问协议的 Store 对象访问消息存储。大多数邮件存储需要进行身份验证，才允许访问的用户。connect 方法进行身份验证。

代码中对 Store 对象的获取如下所示：

```
1   public void connectToServer() throws MessagingException
2   {
3       MyAuthenticator authenticator = null;
4       if(this.receiverInfo.isValidate())
5           authenticator = new MyAuthenticator(this.receiverInfo.getUserName(),
                this.receiverInfo.getPassword());
6
7       Session session = Session.getInstance(this.receiverInfo.getProperties(),
            authenticator);
8
9       try
10      {
11          // 获得Store对象
12          this.store = session.getStore(this.receiverInfo.getProtocal());
13      }
14      catch (NoSuchProviderException e)
15      {
16          e.printStackTrace();
17          throw new MessagingException("连接服务器失败!");
18      }
19
20      System.out.println("connecting");
21
22      try
23      {
24          // 连接到服务器
25          this.store.connect();
26      }
27      catch (MessagingException e)
28      {
29          throw new MessagingException("连接服务器失败!");
30      }
31
32      System.out.println("连接服务器成功");
33  }
```

Folder 类是表示一个文件夹的邮件消息的抽象类。子类实现协议的具体文件夹。文件夹可以包含子文件夹，以及消息，从而提供了一种分层结构。连接到存储后，您就可以得到一个文件夹，必须先打开，然后才能从中读取消息。我们在代码中打开收件箱的操作如下：

```
public void openInBoxFolder() throws MessagingException
{
    try {
        this.folder = this.store.getFolder("INBOX");
        folder.open(Folder.READ_ONLY);
    }
    catch (MessagingException e)
    {
        throw new MessagingException("打开收件箱失败！");
    }
}
```

接下来就是获取 Message 对象并解析 Message 对象。获取 Message 对象如下所示：

```
Message[] messages = this.folder.getMessages();
```

但是，这里获得的 messages 都是乱序的，不是按照日期排序的，我们还需要对其进行排序。这里使用了 Java 中的 Collections 类进行排序：

```
// 获得邮件对象
messages = myMessage.getMessages();

// 这里的MailItem是我定义的一个类，用于存放邮件的一些信息
// 稍后在控制层中可以看到，mailItemList会和recyclerview配合使用
ArrayList<MailItem> mailItemList = new ArrayList<MailItem>();

for (int index = 0; index < messages.length; ++index)
{
    try {
        MessageResolver messageResolver = new MessageResolver(messages[index]);
        mailItemList.add(new MailItem(messages[index], messageResolver.
            getSubject(), messageResolver.getSentDate()));
    }
    catch (MessagingException e)
    {
        e.printStackTrace();
    }
}
// MailItem是Comparable的一个子类，所以可以用Collections排序
// 具体实现见附件中的代码
Collections.sort(mailItemList);
```

解析 Message 对象的算法如下：

1. 首先判断 Message 对象是不是一个附件，如果是，就直接保存它的内容。
2. 随后判断 Message 对象的类型。
3. 如果 Message 对象是具体一个文件类型的邮件，就直接得到它的内容。
4. 如果 Message 对象的类型是 multipart，就循环得到它的各个 part，然后再递归解析各个 part 对象。

实现解析功能的代码如下：

```
1   private void saveEveryPartOfMessage(String dirName, Part part) throws
        IOException, MessagingException
2   {
3       String disposition = part.getDisposition();
4       String contentType = part.getContentType();
5
6       // 判断Message对象是不是一个附件
7       if (disposition != null)
8       {
9           String filename = getFileName(dirName, part);
10          System.out.println("邮件附件的存储路径:" + filename);
11          saveFile(part, filename);
12          return;
13      }
14
15      // 判断Message对象的类型
16      if (contentType.contains("multipart"))
17      {
18          DataSource source = new ByteArrayDataSource(part.getInputStream(), "
                multipart/*");
19          Multipart mp = new MimeMultipart(source);
20
21          for (int index = 0; index < mp.getCount(); ++index)
22              saveEveryPartOfMessage(dirName, mp.getBodyPart(index));
23      }
24      else
25      {
26          if (contentType.contains("text/plain"))
27          {
28              String filename = getFileName(dirName, part);
29              System.out.println("邮件txt文件的存储路径:" + filename);
30              saveFile(part, filename);
31          }
32          else if(contentType.contains("text/html"))
33          {
34              String filename = getFileName(dirName, part);
35              System.out.println("邮件html文件的存储路径:" + filename);
36              saveFile(part, filename);
37          }
38      }
39  }
```

收邮件的功能其实到这里就结束了，我们可以得到邮件的各个部分。但是需要知道的是，我们是想实现一个 android 应用，不可能只是把邮件的各部分简单地保存到本地，我们还需要把它的内容显示出来，这在控制层会讲如何实现。

3.3.3 编写一个控制层

控制层主要想实现的功能是，能显示邮件列表，然后能显示邮件内容。邮件列表是 recyclerview 实现的，邮件内容的显示是通过三个 textview 实现的。其中两个 textview 是显示发件人和邮件主题，最后一个 textview 用于显示邮件内容。

3.3.4 recyclerView 的实现

首先来说 recyclerview，我们为了实现它，需要写一个 MailItemHolder 类和一个 MailItemAdapter 类。简单来说，MailItemHolder 类用于显示每个 MailItem，MailItemAdapter 类用于将 MailItemHolder 类和特定的 MailItem 绑定起来。

```
1   private class MailItemHolder extends RecyclerView.ViewHolder implements View.
        OnClickListener
2   {
3       public TextView subjectTextView;
4       public TextView dateTextView;
5       public Message message;
6
7       public MailItemHolder(View itemView)
8       {
9           super(itemView);
10          itemView.setOnClickListener(this);
11          subjectTextView = (TextView)itemView.findViewById(R.id.
                from_and_subject_info);
12          dateTextView = (TextView)itemView.findViewById(R.id.date_info);
13      }
14
15      @Override
16      public void onClick(View v)
17      {
18          myMessage.setMessage(this.message);
19          Intent i = MailDetailActivity.newIntent(MailListActivity.this);
20          startActivity(i);
21      }
22  }
23
24  private class MailItemAdapter extends RecyclerView.Adapter<MailItemHolder>
25  {
26      private List<MailItem> mailItemList;
27      private DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
            ;
28
29      public MailItemAdapter(List<MailItem> mailItemList)
30      {
31          this.mailItemList = mailItemList;
32      }
33
34      @Override
35      public MailItemHolder onCreateViewHolder(ViewGroup parent, int viewType)
36      {
37          LayoutInflater layoutInflater = LayoutInflater.from(MailListActivity.
```

```
                this);
38          View view = layoutInflater.inflate(R.layout.received_mail_item, parent,
                false);
39          return new MailItemHolder(view);
40      }
41
42      @Override
43      public void onBindViewHolder(MailItemHolder holder, int position)
44      {
45          if (mailItemList == null)
46              return;
47          holder.subjectTextView.setText(mailItemList.get(position).getSubject())
                ;
48          holder.dateTextView.setText(dateFormat.format(mailItemList.get(position
                ).getDate()));
49          holder.message = mailItemList.get(position).getMessage();
50      }
51
52      @Override
53      public int getItemCount()
54      {
55          if (mailItemList == null)
56              return 0;
57          return this.mailItemList.size();
58      }
59  }
```

实现这两个类以后，只要在通过如下两行代码，就能让 recyclerview 显示邮件列表：

```
1   mailItemAdapter = new MailItemAdapter(mailItemList);
2   recyclerView.setAdapter(mailItemAdapter);
```

3.3.5 显示邮件内容

前两个 textview 分别用于显示显示发件人和邮件主题，代码如下：

```
1   private class GetMessages extends AsyncTask<Void, Void, Void>
2   {
3       @Override
4       protected Void doInBackground(Void... params)
5       {
6
7           try
8           {
9               myMessage.openInBoxFolder();
10
11              // MessageResolver是我定义的一个类，用于解析Message对象
12              messageResolver = new MessageResolver(myMessage.getMessage());
13              fromAddress = messageResolver.getFrom();
14              subject = messageResolver.getSubject();
15              ...
16          }
17          catch (MessagingException e)
```

```
            {
                e.printStackTrace();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
            return null;
        }

        @Override
        protected void onPostExecute(Void params)
        {
            // 显示发件人和邮件主题
            fromAddressTextView.setText(fromAddress);
            subjectTextView.setText(subject);
            ...
        }
    }
```

最后一个 textview 用于显示邮件内容，算法如下：

1. 获得邮件的类型。
2. 如果邮件是“text/plain”类型的，就直接显示邮件内容。
3. 如果邮件是“text/html”类型的，就将邮件内容转码以后显示。
4. 如果邮件是“multipart/related”类型的，说明邮件是多图文的，需要将附件中的图片存入本地，然后将 html 文本中 img 标签的 src 路径改为本地路径，从而在 textview 显示出来。
5. 如果邮件是其它类型的，就分别得到其中的“HTML”内容，如果该内容为空，就得到其中的“PLAIN”内容。

实现代码如下：

```
    private class GetMessages extends AsyncTask<Void, Void, Void>
    {
        @Override
        protected Void doInBackground(Void... params)
        {
            try
            {
                myMessage.openInBoxFolder();
                messageResolver = new MessageResolver(myMessage.getMessage());
                ...
                contentType = messageResolver.getContentType();
                if (contentType.contains("text/plain") || contentType.contains("
                    text/html"))
                    content = myMessage.getMessage().getContent().toString();
                else if (contentType.contains("multipart/related"))
```

```
15                  {
16                      File externalFilesDir = MailDetailActivity.this.
                            getExternalFilesDir(Environment.DIRECTORY_PICTURES);
17                      if(externalFilesDir == null)
18                      {
19                          Toast.makeText(MailDetailActivity.this, "Wrong", Toast.
                                LENGTH_SHORT).show();
20                          return null;
21                      }
22                      Log.d("OPEN", externalFilesDir.toString());
23                      dirName = externalFilesDir.toString();
24
25                      content = messageResolver.getHtmlContent(dirName);
26                      if (content.contains("<style>"))
27                      {
28                          int start = content.indexOf("<style>");
29                          int end = content.indexOf("</style>");
30                          content = content.substring(0, start) + content.substring(
                                end+8);
31                      }
32
33                      text_or_html = false;
34                  }
35                  else if (contentType.contains("multipart"))
36                  {
37                      content = messageResolver.getHtmlContent("NULL");
38                      if (content == null || content.equals(""))
39                      {
40                          content = messageResolver.getPlainContent();
41                          text_or_html = true;
42                      }
43                      else
44                      {
45                          if (content.contains("<style>"))
46                          {
47                              int start = content.indexOf("<style>");
48                              int end = content.indexOf("</style>");
49                              content = content.substring(0, start) + content.
                                    substring(end+8);
50                          }
51                          text_or_html = false;
52                      }
53                  }
54                  else
55                      content = "";
56              }
57              catch (MessagingException e)
58              {
59                  e.printStackTrace();
60              }
61              catch (IOException e)
62              {
63                  e.printStackTrace();
64              }
65              return null;
66          }
```

```
67
68          @Override
69          protected void onPostExecute(Void params)
70          {
71              ...
72              String temp = "<img src=\"/storage/emulated/0/Android/data/psd.email/
                    files/Pictures/test.jpg\" />";
73              if (contentType.contains("multipart") && content != null)
74              {
75                  if (contentType.contains("multipart/related"))
76                  {
77                      content = FormatHtml.stringFormat(content);
78                      contentView.setText(Html.fromHtml(content, imageGetter, null));
79                  }
80                  else
81                  {
82                      if (text_or_html)
83                          contentView.setText(content);
84                      else
85                          contentView.setText(Html.fromHtml(content));
86                  }
87              }
88              else if (contentType.contains("text/plain"))
89                  contentView.setText(Html.fromHtml(content));
90              else
91                  contentView.setText(content);
92          }
93      }
```

其中 FormatHtml.stringFormat() 函数是我用来处理 html 文本的，代码如下：

```
1       public class FormatHtml
2       {
3           public static String stringFormat(String html)
4           {
5               Pattern pattern = Pattern.compile("<img src=\"cid.*?>");
6               Matcher matcher = pattern.matcher(html);
7               System.out.println(matcher.replaceFirst("<img src=\"/storage/emulated
                    /0/Android/data/psd.email/files/Pictures/test.jpg\" />"));
8               String s = matcher.replaceFirst("<img src=\"/storage/emulated/0/Android
                    /data/psd.email/files/Pictures/test.jpg\" />");
9               pattern = Pattern.compile("<img src=\"http.*?>");
10              matcher = pattern.matcher(s);
11              return matcher.replaceFirst("");
12          }
13          ...
14      }
```