附录 A　JavaMail 发送邮件

直接使用 JavaMail 编写邮件收发程序并不是一件轻松的事情，这归咎于 JavaMail 零散而复杂的 API 以及各种强制需要处理的检查型异常。Spring 对使用 JavaMail 发送邮件进行了很大程度的简化，它为 80%的需求提供了简单的处理方法，剩下的需求则可以通过直接调用 JavaMail API 完成。

本章主要内容：

- JavaMail 基础知识的快速学习
- Spring 对发送邮件的支持
- 使用 Spring 邮件支持类发送各种类型的邮件
- 在实际应用中发送邮件的经验

本章亮点：

- 对 JavaMail 的基础知识进行了简明扼要的讲解
- 总结了实际应用中发送邮件的受用经验

A.1 JavaMail 快速进阶

在 Java 编程领域，JavaMail 是最知名的邮件解决方案。可以使用 JavaMail 在各种协议环境下，收发各种或简单或复杂的电子邮件。但 JavaMail 本身的 API 比较难用，用户会发现直接使用 JavaMail 发送一封简单的电子邮件并不是一件轻松的事。幸而，Spring 对 JavaMail 施展了它一贯的化繁为简的魔法，在降低使用难度的同时保留 JavaMail 本身的强大功能。了解一下 JavaMail 的基础，有助于深刻理解 Spring 对 JavaMail 的支持。

A.1.1 JavaMail 概述

JavaMail 是由 Sun 定义的一套收发电子邮件的 API，不同的厂商可以提供自己的实现类。但它并没有包含在 JDK 中，而是作为 Java EE 的一部分。

厂商所提供的 JavaMail 服务程序可以有选择地实现某些邮件协议，常见的邮件协议包括：

- SMTP：简单邮件传输协议，用于发送电子邮件的传输协议；
- POP3：用于接收电子邮件的标准协议；
- IMAP：互联网消息访问协议，是 POP3 的替代协议。

这三种协议都有对应 SSL 加密传输的协议，分别是 SMTPS、POP3S 和 IMAPS。

除 JavaMail 的核心包之外，JavaMail 还需要 JAF（JavaBeans Activation Framework）来处理不是纯文本的邮件内容。这包括 MIME（多用途互联网邮件扩展）、URL 页面和文件附件等内容。图 A-1 描述了 JavaMail 的体系结构：

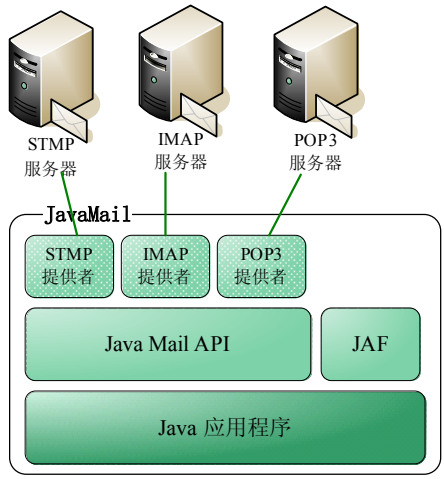

图 A-1 JavaMail 体系结构

- mail.jar：此 JAR 文件包含 JavaMail API 和 Sun 提供的 SMTP、IMAP 和 POP3 服务提供程序；
- activation.jar：此 JAR 文件包含 JAF API 和 Sun 的实现。

用户可以在 Spring 第三方依赖包 Spring/lib/j2ee 下找到对应的文件。

A.1.2　JavaMail 的关键对象

JavaMail 对收发邮件进行了高级抽象，形成了一些关键的接口和类，它们构成了邮件程序的基础，下面我们分别来了解一下这些核心的对象。

Properties：属性对象

第一个我们要认识的就是属性对象 java.util.Properties，由于 JavaMail 需要和邮件服务器进行通信，这就要求程序提供许多诸如服务器地址、端口、用户名、密码等信息，JavaMail 通过 Properties 对象封装这些属性信息。如下面的代码封装了两个属性信息：

```
Properties props = new Properties ();
props.put("mail.smtp.host", "smtp.sina.com.cn");
props.put("mail.smtp.auth", "true");
```

针对不同的邮件协议，JavaMail 规定了服务提供者必须支持一系列属性，表 A-1 是针对 SMTP 协议的一些常见属性（属性值都以 String 类型进行设置，属性类型列仅表示属性最终是如何被解析的）：

表 A-1　JavaMail 属性

属　性　名	属性类型	说　　明
mail.smtp.host	String	SMTP 服务器地址，如 smtp.sina.com.cn
mail.smtp.port	int	SMTP 服务器端口号，默认为 25
mail.smtp.auth	boolean	SMTP 服务器是否需要用户认证，默认为 false
mail.smtp.user	String	SMTP 默认的登录用户名
mail.smtp.from	String	默认的邮件发送源地址
mail.smtp.socketFactory.class	String	socket 工厂类类名，通过设置该属性可以覆盖提供者默认的实现。必须实现 javax.net.SocketFactory 接口
mail.smtp.socketFactory.port	int	指定 socket 工厂类所用的端口号，如果没有设定，则使用默认的端口
mail.smtp.socketFactory.fallback	boolean	设置为 true 时，当使用指定的 socket 类创建 socket 失败后，将使用 java.net.Socket 创建 socket。默认为 true
mail.smtp.timeout	int	I/O 连接超时时间，单位为毫秒，默认为永不超时

其他几个协议也有一系列类似的属性，如 POP3 的 mail.pop3.host、mail.pop3.port 以及 IMAP 的 mail.imap.host、mail.imap.port 等。更详细的信息请查看 com.sun.mail.smtp、com.sun.mail.pop3 和 com.sun.mail.imap 这三个包的 Javadoc：http://javamail.kenai.com/nonav/javadocs/index.html。

Session：会话对象

Session 是一个很容易被误解的类，这归咎于混淆视听的类名。乍一看，我们会以为 Session 像 HttpSession 一样代表真实的交互会话，但实际上创建 Session 对象时，并没有对应的物理连接，它只不过是一堆配置信息的集合。Session 的主要作用包括两个方面：

1）接收各种配置属性信息：通过 Properties 对象设置的属性信息；

2）初始化 JavaMail 环境：根据 JavaMail 的配置文件，初始化 JavaMail 环境，以便通过 Session 对象创建其他重要类的实例。

所以，如果把 Session 更名为 Configure 也许更容易理解一些。JavaMail 提供者在 JAR 包的 META-INF 目录下，通过以下文件提供了基本配置信息，以便 Session 能够根据这个配置文件加载提供者的实现类：

- javamail.providers 和 javamail.default.providers；
- javamail.address.map 和 javamail.default.address.map。

下面是 Sun 提供者 javamail.default.providers 文件的配置信息（位于 mail.jar 包中）：

```
# JavaMail IMAP provider Sun Microsystems, Inc
protocol=imap; type=store; class=com.sun.mail.imap.IMAPStore; vendor=Sun Microsystems, Inc;
protocol=imaps; type=store; class=com.sun.mail.imap.IMAPSSLStore; vendor=Sun
Microsystems, Inc;
# JavaMail SMTP provider Sun Microsystems, Inc
protocol=smtp; type=transport; class=com.sun.mail.smtp.SMTPTransport; vendor=Sun
Microsystems, Inc;
protocol=smtps; type=transport; class=com.sun.mail.smtp.SMTPSSLTransport; vendor=Sun
Microsystems, Inc;
# JavaMail POP3 provider Sun Microsystems, Inc
protocol=pop3; type=store; class=com.sun.mail.pop3.POP3Store; vendor=Sun Microsystems,
Inc;
protocol=pop3s; type=store; class=com.sun.mail.pop3.POP3SSLStore; vendor=Sun
Microsystems, Inc;
```

这个配置文件提供了以下四个方面的信息：

- protocol：协议名称；
- type：邮件操作类型；
- class：对应该操作类型的实现类；
- vendor：厂商名称。

Session 在加载配置文件时会按照以下优先级顺序进行：

1）首先使用<*JAVA_HOME*>/lib 中的 javamail.providers；

2）如果 1）不存在相应的配置文件，使用类路径下 mail.jar 中 META-INF 目录下的 javamail.providers；

3）如果 2）不存在相应的配置文件，使用类路径下 mail.jar 中 META-INF 目录下的 javamail.default.providers。

所以开发者可以在<*JAVA_HOME*>/lib 目录下提供配置文件覆盖 mail.jar/META-INF 目录中厂商的配置。但是，一般情况下，我们无须这样做。

Session 通过 JavaMail 配置文件以及程序中设置的 Properties 对象建构一个邮件处理环境，后续的处理将在 Session 基础上进行。Session 拥有多个静态工厂方法用于创建 Session 实例。

- static Session getDefaultInstance(Properties props, Authenticator authenticator)：当 JVM 中已经存在默认的 Session 实例时，直接返回这个实例，否则创建一个新的 Session 实例，并将其作为 JVM 中默认 Session 实例。这个 API 方法很诡秘，我们将对它进

行详细的讲解。由于这个默认 Session 实例可以被同一 JVM 所有的代码访问到，而 Session 中本身又可能包含密码、用户名等敏感信息在内的所有属性信息，所以后续调用也必须传入和第一次相同的 Authenticator 实例，否则将抛出 java.lang.SecurityException 异常。如果第一次调用时 Authenticator 入参为 null，则后续调用通过 null 的 Authenticator 入参或直接使用 getDefaultInstance(Properties props)即可返回这个默认的 Session 实例。值得一提的是，虽然后续调用也会传入 Properties，但新属性值并不会起作用，如果希望采用新的属性值，则可以通过 getInstance(Properties props)创建一个新的 Session 实例达到目的。Authenticator 在这里承当了两个功能：首先，对 JVM 中默认 Session 实例进行认证保护，后续调用执行 getDefaultInstance(Properties props, Authenticator authenticator)方法时必须和第一次一样；其次，在具体和邮件服务器交互时，又作为认证的信息；

- static Session getDefaultInstance(Properties props)：返回 JVM 中默认的 Session 实例，如果第一次创建 Session 未指定 Authenticator 入参，后续调用可以使用该访问获取 Session；
- static Session getInstance(Properties props, Authenticator authenticator)：创建一个新的 Session 实例，它不会在 JVM 中被作为默认实例共享；
- static Session getInstance(Properties props)：根据相关属性创建一个新的 Session 实例，未使用安全认证信息；

Session 是 JavaMail 提供者配置文件以及设置属性信息的“容器”，Session 本身不会和邮件服务器进行任何的通信。所以在一般情况下，我们仅需要通过 getDefaultInstance()获取一个共享的 Session 实例就可以了，下面的代码创建了一个 Session 实例：

```
Properties props = System.getProperties();
props.setProperty("mail.transport.protocol", "smtp ");
…
Session session = Session.getDefaultInstance(props);①
```

仅创建一个对象，并不会产生任何通信

Transport 和 Store：传输和存储

邮件的操作只有发送或接收两种处理方式，JavaMail 将这两种不同操作描述为传输（javax.mail.Transport）和存储（javax.mail.Store）两个类，传输对应邮件的发送，而存储对应邮件的接收（又是两个容易因命名被误解的类）。

Session 提供了几个用于创建 Transport 和 Store 实例的方法，在具体讲解这些方法之前，我们事先了解一下 Session 创建 Transport 和 Store 的内部机制。我们知道提供者在 javamail.providers 配置文件中为每一种支持的邮件协议定义了实现类，Session 根据协议类型（stmp、pop3 等）和邮件操作方式（传输和储存）这两个信息就可以定位到一个实例类上。比如，指定 stmp 协议和 transport 类型后，Session 就会使用 com.sun.mail.smtp.SMTPTransport 实现类创建一个 Transport 实例，而指定 pop3 协议和 store 类型时，则会使用 com.sun.mail.pop3.POP3Store 实例类创建一个 Store 实例。Session 提供了多个重载的 getTransport()和 getStore()方法，这些方法将根据 Session 中 Properties 属性设置情况进行工作，影响这两套方法工作的属性包括：

属 性 名	说 明
mail.transport.protocol	默认的传输邮件协议，例如：stmp。
mail.store.protocol	默认的存储邮件协议，例如：pop3。
mail.host	默认的邮件服务器地址，例如：192.168.67.1。
mail.user	默认的登录用户名，例如：masterspring。

下面，我们再回过头来了解 Session 的 getTransport()和 getStore()的重载方法。

- Transport getTransport()：当 Session 实例设置了 mail.transport.protocol 属性时，该方法返回对应的 Transport 实例，否则抛出 javax.mail.NoSuchProviderException。
- Transport getTransport(String protocol)：如果 Session 没有设置 mail.transport.protocol 属性，可以通过该方法返回指定类型的 Transport，如：transport = session.getTransport ("smtp")。

如果 Session 中未包含 Authenticator，以上两方法创建的 Transport 实例和邮件服务器交互时必须显式提供的用户名/密码的认证信息。如果 Authenticator 非空，则可以在和邮件服务器交互时被作为认证信息使用。除了以上两种提供认证信息的方式外，Session 还可以使用以下的方法为 Transport 提供认证信息。

- Transport getTransport(URLName url)：用户可以通过 URLName 入参指定邮件协议、邮件服务器、端口、用户名和密码信息，请看下面的代码：

```
URLName urln = new URLName("smtp"," smtp.sina.com.cn",25,null,"masterspring", "spring");
Transport transport = session.getTransport(urln);
```

这里，指定了邮件协议为 smtp，邮件服务器是 smtp.sina.com.cn，端口为 25，用户名/密码为 masterspring/spring。

getStore()也有和 getTransport()相似的重载方法：

- Store getStore()；
- Store getStore(String protocol)；
- Store getStore(URLName url)。

在从 Session 中获取 Transport 和 Store 时，也没有和邮件服务器发生任何通信，仅仅是创建一个内存实例而已，因此即使用户名和密码是错误的，也不会抛出运行异常。

Transport 和 Store 都拥有 connect()和 close()方法，只有调用 connect()方法时，程序才会和邮件服务器产生物理连接。服务器地址、端口以及用户名/密码等信息在此时才会真正被使用。

如果用户没有在构建 Session 或 Transport/Store 时提供认证信息，JavaMail 允许用户在调用 connect()方法时再提供连接信息：connect(String host, int port, String user, String password)。如果我们回顾一下 JDBC 的 API，就会发现 JavaMail 设置连接的认证信息确实是比较灵活的，存在三种方式：1)在创建 Session 时通过 Authenticator 设置；2)在创建 Transport/Store 时通过 URLName 提供；3)在实际连接时通过 connect()方法入参提供。但是这种过度的灵活性反而增强了理解的难度，造成使用上的不便。

Message：消息对象

javax.mail.Message 代表邮件信息，包含一系列的属性，其中 content 属性表示邮件的

内容。Message 是一个抽象类，子类提供具体的实现。其中最重要的一个子类是 javax.mail.internet.MimeMessage，它可以构造出复杂的邮件信息。下面的代码演示了创建一个简单邮件信息的过程：

```
Message msg = new MimeMessage(session);
msg.setSubject("Hello");
msg.setText("How are you");
msg.setSentDate(new Date());
```

上面介绍的 5 个对象是 JavaMail 最重要的基础类，从中我们可以看出，JavaMail 的整个体系比较复杂，甚至显得凌乱不堪，拥有太多属性需要设置（本该有默认的值），很多 API 的命名比较隐晦，没有直接地表现真实意图。这些缺点大大提高了 JavaMail 程序的使用难度。不过不能否认的一点是，JavaMail 的功能是非常强大的，用户可以使用它解决几乎任何的邮件问题。

A.1.3　使用 JavaMail 发送邮件

下面，我们通过 JavaMail 发送一封简单的电子邮件，其代码如代码清单 A-1 所示：

代码清单 A-1　JavaMailSender

```
package com.baobaotao.basic.javamail;
import java.util.Date;
import java.util.Properties;
import javax.mail.*;
public class JavaMailSender {
    public static void main(String[] args) {
        Transport transport = null;
        try {
            Properties props = new Properties();                    ①
            props.put("mail.smtp.host", "smtp.163.com");
            props.put("mail.smtp.auth", "true");

            Authenticator auth = new Authenticator() {②
                protected PasswordAuthentication getPasswordAuthentication() {
                    return new PasswordAuthentication("masterspring", "spring");
                }
            };
            Session session = Session.getDefaultInstance(props); ③
            Message msg = new MimeMessage(session);
            msg.setFrom(new InternetAddress("masterspring@163.com"));③-1
            msg.setRecipient(Message.RecipientType.TO,                  ③-2
                    new InternetAddress("masterspring@sina.com"));
            msg.setSubject("Test Title");
            msg.setText("How are you!");
            msg.setSentDate(new Date());

            transport = session.getTransport("smtp");④
            transport.send(msg); ④-1
```

属性信息设置

创建认证匿名类实例，分别指定用户名和密码

构造邮件信息

源地址

目标地址

发送邮件信息

```
            System.out.println("邮件发送成功!");
        } catch (MessagingException m) {
            System.out.println("邮件发送失败!");
            m.printStackTrace();
        } finally {
            try {
              transport.close();⑤ ←—— 停止传输，关闭连接
            } catch (Exception e) {}
        }
    }
}
```

Message 除了包含邮件本身的标题、内容等信息外，还包括源地址、接收地址的信息，如③-1 和③-2 所示。用户可以同时将一封邮件发送到多个接收地址上，接收地址包括三种类型：主送地址、抄送地址和暗送地址，这些地址类型在 Message.RecipientType 类中定义：

- TO：主送地址；
- CC：抄送地址；
- BCC：暗送地址。

Message 提供了多个设置目标地址的方法：

- void addRecipient(Message.RecipientType type, Address address)：添加一个某一类型的接收地址；
- void addRecipients(Message.RecipientType type, Address[] addresses)：添加一批某一类型的接收地址；
- void setRecipient(Message.RecipientType type,Address address)：设置一个指定类型的接收地址，相同类型原有的地址将被覆盖；
- void addRecipients(Message.RecipientType type,Address[] addresses)：设置一批某类型的接收地址，相同类型的原有地址将被覆盖；

在④-1 处调用 Transport 的 send()方法时，在内部 Transport 会自动调用 connect()方法创建连接，然后再发送邮件消息。任何一个接收地址被探测为非法时，该方法就抛出 SendFailedException 异常，邮件是否还会被发送到其他合法的地址上，则仰仗于邮件服务器的具体实现。在程序结束后，需要关闭连接，如⑤所示。

轻松一刻

鸿雁是书信的代称，有时亦指代邮递员。何以“鸿雁”指代书信和邮递员？溯其源，来自于苏武牧羊。汉朝时，苏武出使匈奴，被单于流放北海（今俄罗斯贝加尔湖）牧羊。19 年后，汉朝与匈奴和亲，但单于仍不让苏武回汉。与苏武一起出使匈奴的常惠，把苏武的情况密告汉使，并设计让汉使对单于讲：汉朝皇帝打猎射得一雁，雁足上绑有书信，叙说苏武在北海牧羊。单于听后，以为天意不可违，只得让苏武回汉。后来，人们就用鸿雁比喻书信和传递书信的邮递员。

A.1.4　处理 SSL 加密传输协议邮件

Sun 提供的 JavaMail 实现还支持 SSL 加密的邮件协议，只要设置好属性信息，处理 SSL 加密协议的邮件和未加密协议的邮件并无多大区别。google 的 Gmail 采用 SSL 加密传输协议，下面的代码能够从 Gmail 服务器中获取某个邮箱内的信件信息：

代码清单 A-2　JavaMailSslReceiver

```
package com.baobaotao.basic.javamail;
import java.io.UnsupportedEncodingException;
import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.*;
public class JavaMailSslReceiver {
    public static void main(String argv[]) throws Exception {
        Properties props = new Properties();
        Session session = Session.getDefaultInstance(props);
        URLName urln = new URLName("pop3s", "pop.gmail.com", 995, null, ①
                "masterspring3", "masterspring");
        Store store = session.getStore(urln);
        Folder inbox = null;
        try {
            store.connect(); ②

            inbox = store.getFolder("INBOX");                    ③
            inbox.open(Folder.READ_ONLY);
            FetchProfile profile = new FetchProfile();
            profile.add(FetchProfile.Item.ENVELOPE);
            Message[] msgs = inbox.getMessages();
            inbox.fetch(msgs, profile);
            System.out.println("收件箱的邮件数：" + msgs.length);
            for (int i = 0; i < msgs.length; i++) {④

                String from = msgs[i].getFrom()[0].toString();
                InternetAddress ia = new InternetAddress(from);
                System.out.println("----------------------------");
                System.out.println("发送者:" + ia.getPersonal() + "/"+ ia.getAddress());
                System.out.println("标题:" + msgs[i].getSubject());
                System.out.println("大小:" + msgs[i].getSize());
                System.out.println("时间:" + msgs[i].getSentDate());
            }
        } finally {
            try {
                inbox.close(false); ⑤
            } catch (Exception e) {}
            try {
                store.close();⑥
            } catch (Exception e) {}
        }
    }
}
```

指定邮件协议为 pop3s，并指定邮件服务器，端口，用户名和密码的信息

连接邮件服务器

从邮件服务器中返回邮箱内的信息

显示邮件信息

关闭打开的 Folder

关闭打开的存储对象

在①处，我们通过 URLName 对象指定了连接邮件服务器必要的信息，未采用 SSL 加密传输协议的常用 POP3 协议默认端口为 110，使用 SSL 的 POP3S 的默认端口则为 995，而使用 SSL 的 SMTP 端口为 465。

在②处，Store 通过显式的 connect()方法建立和服务器的连接，如②所示，处理完成后，还必须关闭 Store，如⑥所示。JavaMail 通过类似于文件夹的方式操作邮箱中的信息，详细内容请参见 JavaMail 的 Javadoc。

运行以上代码，将输出以下的信息：

```
收件箱的邮件数：3
------------------------------
发送者:Gmail 小组/mail-noreply@google.com
标题:在手机上使用 Gmail
大小:2384
时间:Sat Jun 18 16:05:39 CST 2011
------------------------------
发送者:Gmail 小组/mail-noreply@google.com
标题:导入联系人和过去的电子邮件
大小:2981
时间:Sat Jun 18 16:05:39 CST 2011
------------------------------
发送者:Gmail 小组/mail-noreply@google.com
标题:使用颜色和主题自定义 Gmail
大小:2879
时间:Sat Jun 18 16:05:39 CST 2011
```

是不是觉得已经有点邮件管理系统的味道了，只要在 JavaMail 基础上添加展现层的设计，一个完善的电子邮局即可建立起来。

mail.jar 中 SMTPS 协议对应的实现类是 com.sun.mail.smtp.SMTPSSLTransport，但使用该类发送邮件时，在有些邮件服务器下不能正常工作。这时，用户可以通过 mail.smtp.socketFactory.class 属性显式指定 javax.net.ssl.SSLSocketFactory 作为 SSL Socket 的操作类。

A.2 Spring 的邮件支持

Spring 对发送电子邮件提供了一个抽象层，对接收电子邮件没有提供额外的支持。不过这对于大多数企业应用来说已经足够了。因为在一般情况下，只有专门的邮件管理系统才会有接收邮件的需求。JavaMail API 盘根错节，复杂凌乱，经过 Spring 大刀阔斧的整容，蓬头垢面的邋遢虫出落成清新淡雅的美少女。

A.2.1 邮件发送高级抽象层

在编写邮件发送程序时，有两个最重要的内容：邮件消息和邮件发送者。Spring 在 org.springframework.mail 包里通过 MailMessage 和 MailSender 这两个高层抽象接口描述了这两项内容。

MailMessage：抽象的邮件消息

MailMessage 接口描述了邮件消息的通用模型，允许开发者通过多个简洁的属性设置方法填充邮件消息的各项内容。

- void setTo(String to) ：设置一个主送地址，如需设置多个地址，可以使用 setTo(String[] to)方法；
- void setFrom(String from)：设置发送地址；
- void setReplyTo(String replyTo)：设置回复地址；
- void setCc(String cc)：设置抄送地址，如需设置多个地址，可以使用 setCc(String[] cc)；
- void setBcc(String bcc)：设置暗送地址，如需设置多个地址，可以使用 setBcc(String[] bcc)；
- void setSentDate(Date sentDate)：设置发送邮件的时间；
- void setSubject(String subject)：设置邮件标题；
- void setText(String text)：设置邮件内容。

相对于 JavaMail Message 接口的 addRecipients()和 setRecipient()方法，MailMessage 接口中定义的方法显得更加简洁明了，Spring 通过明确的属性设置方法规避 JavaMail 接口中晦涩难懂的设计。

MailMessage 有两个实现类：SimpleMailMessage 和 MimeMailMessage，前者都是完全符合 Bean 风格的实现类，后者通过内置 JavaMail 的 MimeMessage 提供实现。Spring 之所以需要提供这两个实现类，是因为 Spring 的高级抽象层并不局限在某一特定的实现上，在 Spring 2.1 版本之前，Spring 同时支持 JavaMail 和 COS 两种邮件 API，不过在 Spring 2.1 中已经移除对 COS 的支持。在发送简单文本型邮件时，SimpleMailMessage 就可以满足要求了，如果发送复杂的邮件，则可以利用 MimeMailMessage 或直接使用 MimeMessage。

MailSender：抽象的邮件发送者

邮件发送者负责将邮件发送到指定的地址上，该接口只用于发送简单的邮件。不同的邮件系统可以提供各自的实现。如果需要发送复杂的邮件，则需要使用 JavaMailSender 子接口。MailSender 接口很简单，只有两个方法。

- void send(SimpleMailMessage simpleMessage) throws MailException：发送一个邮件消息，注意入参是 SimpleMailMessage 而非 MailMessage！为什么是这么一个奇怪的入参类型呢？原因是 MailSender 只能发送简单的邮件类型，它和邮件实现者无关，并不局限于 JavaMail。该接口根据邮件发送时的具体情况会抛出相应的异常，这些异常也是 Spring 高级邮件抽象的一部分：
  - MailParseException：当发生邮件消息解析错误时抛出；
  - MailAuthenticationException：当认证失败时抛出；
  - MailSendException：当发送邮件消息失败时抛出。
- void send(SimpleMailMessage[] simpleMessages) throws MailException：一次性发送多个邮件消息。

Spring 的邮件异常体系

Spring 定义了和邮件系统无关的邮件异常体系，Spring 为 JavaMail 提供的支持类会将邮件系统相关的异常转化为 Spring 的邮件异常。Spring 提供的邮件异常都是运行期异常，因此，用户可以有针对性地捕捉感兴趣的异常，而不像 JavaMail 中的检查型异常需要强制编写捕捉代码。一般情况下，像认证失败、邮件消息解析错误等异常都是不可恢复的错误，无须捕捉。在实际业务系统中，发送失败的异常可能会被关注，因为发送失败后，可能需要再次尝试发送，或者发送到接收者的备用邮箱中。Spring 的邮件异常体系，如图 A-2 所示：

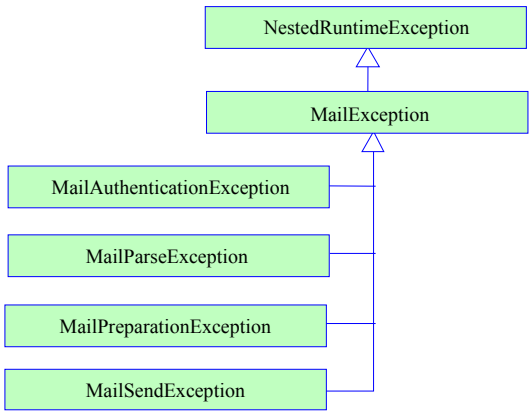

图 A-2 Spring 的邮件异常体系

MailException 是邮件异常的根，4 个子异常类分别概括了认证失败（MailAuthenticationException）、解析错误（MailParseException）、发送失败（MailSendException）和准备错误（MailPreparationException）这 4 种典型的错误。由于 MailException 继承于 Spring 的 NestedRuntimeException，所以它们都是运行期异常。

MailPreparationException 异常一般在用户代码中抛出，表示准备邮件消息时发生了错误。Spring 为 Velocity 和 FreeMarker 模板框架所提供的工具类会抛出 MailPreparationException 异常：

- VelocityEngineUtils#mergeTemplateIntoString()；
- FreeMarkerTemplateUtils#processTemplateIntoString()。

A.2.2 Spring 对 JavaMail 的支持

Spring 在 org.springframework.mail.javamail 包下提供了对 JavaMail 邮件系统的支持，通过这些支持类，编写 JavaMail 邮件发送程序变得轻松易行。体现在两个方面：首先，通过 JavaMailSenderImpl 可以很方便地创建 JavaMail 环境；其次，通过 MimeMessageHelper 很容易构造出 MimeMessage 对象。

JavaMailSender 及其实现类

Spring 通过 MailSender 的 JavaMailSender 子接口，提供了发送 JavaMail 复杂邮件的功

能。下面我们来了解一下 JavaMailSender 接口的方法：

- void send(MimeMessage mimeMessage)：发送一封 MimeMessage 类型的邮件，用户可以通过 send(MimeMessage[] mimeMessages) 批量发送多封邮件；
- void send(MimeMessagePreparator mimeMessagePreparator)：通过 MimeMessagePreparator 回调接口发送 MimeMessage 类型的邮件，可以通过 send(MimeMessagePreparator[] mimeMessagePreparators)进行批量发送；
- MimeMessage createMimeMessage()：我们知道使用 JavaMail 原生 API 创建一个 MimeMessage 是比较麻烦的，用户必须获得 JavaMail 的 Session 实例。该接口方法为调用者屏蔽了底层的复杂性，直接创建一个 MimeMessage 对象；
- MimeMessage createMimeMessage(InputStream contentStream)throws MailException：根据一个 InputStream 创建 MimeMessage，当发生消息解析错误时，抛出 MailParseException 异常。

JavaMailSenderImpl 是 JavaMailSender 的实现类，它同时支持 JavaMail 的 MimeMessage 和 Spring 的 MailMessage。通过在 Spring 配置文件中设置一些简单的属性就可以轻易地搭建起 JavaMail 环境。此外，还可以通过 JNDI 获取 Java EE 服务器中的 JavaMail Session 实例构建 JavaMail 环境。

下面代码配置了一个 JavaMailSenderImpl Bean：

代码清单 A-3　配置 JavaMailSender

```
<bean id="sender"
    class="org.springframework.mail.javamail.JavaMailSenderImpl"
    p:host="smtp.sina.com.cn"   ①
    p:username="masterspring"  ②
    p:password="spring">
    <property name="javaMailProperties">③
        <props>
            <prop key="mail.smtp.auth">true</prop>
        </props>
    </property>
</bean>
```

邮件服务器地址

用户名及密码

其他的 JavaMail 属性

JavaMailSenderImpl 将 JavaMail 中的属性通过 JavaBean 的方式开放出来方便设置，使我们不必关心 JavaMail 复杂的属性名，JavaMailSenderImpl 提供的属性包括：

- host：邮件服务器地址；
- port：邮件服务器端口，如果邮件服务器采用默认的端口号，就无须显式设置；
- protocol：协议类型，默认为 smtp；
- username：用户名；
- password：密码；
- defaultEncoding：在创建 MimeMessage 时，采用的默认编码；
- session：用户可以通过 session 指定一个已经存在的 Session 实例，如从 JNDI 中获取的 Java EE 服务器的 JavaMail Session。

一般情况下，这些属性已经可以满足要求，如果需要对 JavaMail 其他属性进行设置，

则可以通过 javaMailProperties 属性达到目的。在③处，我们就通过 javaMailProperties 定义了一个名为 mail.smtp.auth 的 JavaMail 属性。由于几乎所有的邮件服务器都要求进行身份认证才可访问，所以需要将 mail.smtp.auth 属性设置为 true。

MimeMessageHelper

MimeMessageHelper 是 MimeMessage 的封装类，它隐藏了构造 MimeMessage 对象的复杂性，使 MimeMessage 实例的构建变得简单易行。该帮助类提供了设置 HTML 邮件内容、内嵌的文件（如图片文件、音频文件等）以及邮件附件的方法。

MimeMessageHelper 通过构造函数接受一个 MimeMessage，让我们来了解一下 MimeMessageHelper 的几个常用构造函数：

- MimeMessageHelper(MimeMessage mimeMessage)：封装 MimeMessage 对象，默认为简单非 multipart 的邮件消息，采用默认的编码；
- MimeMessageHelper(MimeMessage mimeMessage, boolean multipart)：封装 MimeMessage 对象时，同时指定是否是 multipart 邮件消息，采用默认的编码；
- MimeMessageHelper(MimeMessage mimeMessage, boolean multipart, String encoding)：采用指定编码的 MimeMessage。

MimeMessageHelper 所提供多个操作方法可以分为两类：1)指定邮件各种地址（主送、抄送、暗送等）的方法，如 setFrom()、setTo()、setCc()、addTo、addBcc()等；2)设置邮件消息内容的方法，不但包括设置标题、文本内容，还包括添加附件、添加内嵌文件的方法。这里，我们着重了解一下添加附件和内嵌文件的方法。

首先来认识一下添加附件的方法。

- void addAttachment(String attachmentFilename, File file)：添加一个文件作为附件。
- void addAttachment(String attachmentFilename, InputStreamSource inputStreamSource)：将 org.springframework.core.InputStreamResource 添加为附件，Resource 是 Spring 对资源访问的高级抽象，参见 3.3.1 节。InputStreamSource 所对应的 MIME 类型通过 attachmentFilename 指定的文件名进行判断，attachmentFilename 表示邮件中显示的附件文件名。
- void addAttachment(String attachmentFilename, InputStreamSource inputStreamSource, String contentType)：该方法可以显式指定附件的 MIME 类型。

其次，是添加内嵌文件的方法。

- void addInline(String contentId, File file)：将一个文件内嵌到邮件中，文件的 MIME 类型通过文件名判断。contentId 标识这个内嵌的文件，以便邮件中的 HTML 代码可以通过 src="cid:contentId "引用内嵌文件。举一个例子，假设使用 addInline("**img01**",file)添加了一个内嵌图片文件，邮件的 HTML 代码为 <img src="**cid:img01**"></img> ，邮件客户端软件将会把这个图片文件显示在对应的 HTML 中。
- void addInline(String contentId, InputStreamSource inputStreamSource, String contentType)：将 InputStreamSource 作为内嵌文件添加到邮件中，通过 contentType 指定内嵌文件的 MIME 类型。

- void addInline(String contentId, Resource resource)：将 Resource 作为内嵌文件添加到邮件中，内嵌文件对应的 MIME 类型通过 Resource 对应的文件名判断。

向邮件中添加附件或内嵌文件时，文件对应的 MIME 类型是一个很重要的信息，因为 Outlook、Foxmail 等邮件客户端软件必须根据 MIME 类型决定如何处理邮件中的内嵌文件。文件扩展名和 MIME 类型的对应关联在 activation.jar/META-INF 目录下的 mimetypes.default 文件中定义：

```
text/html          html htm HTML HTM
text/plain         txt text TXT TEXT
image/gif          gif GIF
image/ief          ief
image/jpeg             jpeg jpg jpe JPG
image/tiff         tiff tif
image/x-xwindowdump    xwd
application/postscript     ai eps ps
application/rtf        rtf
application/x-tex      tex
application/x-texinfo  texinfo texi
application/x-troff    t tr roff
audio/basic            au
audio/midi         midi mid
audio/x-aifc           aifc
audio/x-aiff               aif aiff
audio/x-mpeg           mpeg mpg
audio/x-wav                wav
video/mpeg             mpeg mpg mpe
video/quicktime        qt mov
video/x-msvideo            avi
```

第一列表示 MIME 类型的规范名，第二列表示对应文件的扩展名。正因为文件扩展名是确定 MIME 类型的依据，所以在添加附件或内嵌文件时，必须注意文件扩展名的正确性。如果需要操作的文件的扩展名没有在 mimetypes.default 中定义，而且它确实属于某类 MIME 类型，用户可以通过 Spring 所提供的 ConfigurableMimeFileTypeMap 类显式指定，该类扩展了 javax.activation.FileTypeMap 接口，可以手工指定文件扩展名和 MIME 类型的映射关系。JavaMailSenderImpl 提供了一个 defaultFileTypeMap 属性用以设置 ConfigurableMimeFileTypeMap 实例。

在 http://www.w3school.com.cn/media/media_mimeref.asp 可以找到所有 MIME 类型的信息。

A.3　发送各种形式的邮件

在学习完 JavaMail 以及 Spring 对 JavaMail 支持的知识后，我们将开始使用 Spring 的 JavaMailSender 发送各种类型的邮件：从最简单的纯文本邮件到带内嵌文件的邮件。

A.3.1 发送纯文本邮件

纯文本邮件是最简单的邮件，邮件内容由简单的文本组成，任何邮件客户端软件都支持查看纯文本的邮件。在 Spring 里，用户可以使用 JavaMailSenderImp 发送纯文本邮件。下面，我们使用 JavaMailSenderImp 发送一封纯文本邮件，这是一封用户在论坛上注册成功后的通知邮件：

代码清单 A-4 MailService# sendSimpleMail()

```
package com.baobaotao.service;
import org.springframework.stereotype.Service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;

@Service
public class MailService {

    @Autowired
    private JavaMailSender sender;

    public void sendSimpleMail(int userId) { ①
        SimpleMailMessage msg = new SimpleMailMessage();
        msg.setFrom("masterspring@163.com");
        msg.setTo("masterspring3@gmail.com");
        msg.setReplyTo("masterspring@163.com");
        msg.setCc("masterspring@sina.com");
        msg.setSubject("注册成功");      ②
        msg.setText("恭喜，您在宝宝淘论坛已经注册成功!您的用户ID为：" + userId); ③
        sender.send(msg); ④

    }
}
```

通过 SimpleMailMessage 发送简单邮件

邮件标题

邮件体内容

发送邮件

如果某个邮箱地址是非法的，④处的 send()在发送邮件过程中将抛出 MailSendException，外部调用者可以捕获这个异常，进行必须的处理。下面我在 Spring 配置文件为 MailService 提供配置：

```
...
    <context:component-scan base-package="com.baobaotao"/>

    <bean id="sender"
        class="org.springframework.mail.javamail.JavaMailSenderImpl"
        p:host="smtp.163.com"
        p:username="masterspring"
        p:password="spring">
        <property name="javaMailProperties">
            <props>
```

```
                <prop key="mail.smtp.auth">true</prop>
            </props>
        </property>
    </bean>
```

使用 MailService#sendSimpleMail(int userId)发送一封邮件，通过 Foxmail 查看这封邮件，其结果如图 A-3 所示。

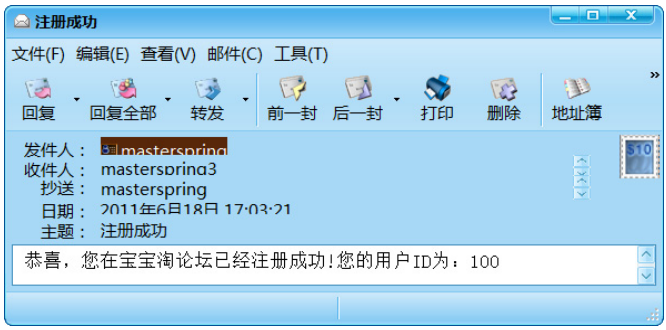

图 A-3　纯文本型邮件

仔细观察收信箱中的信息项，用户可以轻易找出这些信息项和 SimpleMailMessage 属性的对应关系，如“抄送地址”对应 setCc()，“收件人”对应 setTo()。当点击“回复”按钮后，Foxmail 自动将 setReplyTo()中指定的邮箱地址设置为回复信件的“收件人”。

A.3.2　发送 HTML 类型的邮件

发送 HTML 邮件和发送纯文本邮件差不多，不过用户必须使用 MimeMessage 创建邮件消息，通过 MimeMessageHelper 的帮助，创建并填充 MimeMessage 变得非常轻松：

代码清单 A-5　MailService#sendHtmlMail()

```
package com.baobaotao.service;
…
import org.springframework.mail.javamail.MimeMessageHelper;
@Service
public class MailService {
    …

    public void sendHtmlMail(int userId) throws MessagingException { ①
        MimeMessage msg = sender.createMimeMessage();②

        MimeMessageHelper helper = new MimeMessageHelper(msg,false,"utf-8"); ③
    helper.setFrom("masterspring@163.com")
;
        helper.setTo("masterspring3@gmail.com");
        helper.setSubject("注册成功");④
        String htmlText = "<html><head>"+
        "<meta http-equiv=\"content-type\" content=\"text/html; charset=utf-8\">"+ ⑤
                "</head><body>"+
```

MimeMessageHelper 大部分方法都会抛出 MessagingException 异常

创建 MimeMessage

指定编码为 utf-8,同时标识为非 marltipart 的消息

设置标题

构造 HTML 代码

```
                "恭喜，您在宝宝淘论坛已经注册成功!您的用户ID为："+
                "<font color='red' size='30'>" + userId+
                "</font>" + "</body></html>";
        helper.setText(htmlText,true); ⑥ 设置邮件内容，第二个参数为true，表示是 HTML 邮件
        sender.send(msg);
    }
}
```

首先，通过 JavaMailSender#createMimeMessage()方法创建 MimeMessage，如②所示。其次，通过 MimeMessageHelper 指定了邮件消息所采用 utf-8 的编码格式，如③所示。我们推荐使用 utf-8 编码，因为它可以避免各种烦人的中文乱码问题。邮件头只能包含 US-ASCII 字符(参见 RFC822，RFC2047)，如果发送程序使用 GBK 等编码格式，必须利用 javax.mail.internet.MimeUtility 工具类对中文内容进行处理，如：helper.setSubject(MimeUtility.encodeText(subject, "GBK","B"))，第二个入参指定源编码格式；第三个入参指定目标编码格式；B 代表 Base64 编码；Q 代表 Quoted-Printable 编码。

由于我们希望按照 UTF-8 编码格式，因此在构造 HTML 代码时，也必须指定相同的格式，如⑤所示。邮件发送成功后，用户将可以使用 HTML 格式查看邮件了，其效果如图 A-4 所示：

图 A-4　HTML 格式的邮件

由于我们测试所用的 STMP 邮件服务器使用免费的 smtp.163.com，它会对客户端访问的时间间隔进行控制，所以如果过于频繁地测试，邮件服务器将拒绝服务，测试程序将因此抛出异常。所以读者朋友在编写测试程序时，最好申请一个自己的邮箱账号，以免有很多其他的读者因为同时测试而发生相互干扰的问题。

A.3.3　发送带内嵌文件的邮件

用户不但可以使用 JavaMailSender 发送基于 HTML 的文本型邮件，还可以将一些 MIME 类型文件（如图片文件、音频文件）嵌入到邮件体中，以便构建出更加丰富的邮件内容。下面通过一个例子来学习发送带内嵌文件邮件的过程。

代码清单 A-6　MailService#sendInlineMail ()

```
package com.baobaotao.service;
```

```
...
public class MailService {
    ...
    public void sendInlineMail() throws MessagingException {
        MimeMessage msg = sender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(msg,true,"utf-8");①
        helper.setFrom("masterspring@163.com");
        helper.setTo("masterspring3@gmail.com");
        helper.setSubject("注册成功");
        String htmlText = "<html><head>"+
        "<meta http-equiv=\"content-type\" content=\"text/html; charset=utf-8\">"+
                "</head><body>"+
                   "欢迎访问宝宝淘论坛！</hr>"+
                   "<div><img src=\"cid:img01\"></img></div>"+ ②
                   "</body></html>";
        helper.setText(htmlText,true);
        ClassPathResource img = new ClassPathResource("bbt.gif");③
        helper.addInline("img01",img); ④
        sender.send(msg);
    }
}
```

和上一个例子不同，内嵌文件邮件是 multipart 类型，第二个入参需要设置为 true

引用内嵌文件的特殊标识

从类路径加载图片文件

添加内嵌文件

内嵌文件邮件既包含正常的邮件体文本又包含内嵌文件，所以是 multipart 类型的邮件，需要通过 MimeMessageHelper 指定，如①所示。可以通过 addInline()将多个文件内嵌到邮件中，如④所示。内嵌文件的 ID 在邮件 HTML 代码中以特定标志引用，这个特定标志的格式为：cid:<*内嵌文件 id*>，如②所示。

邮件发送成功后，在 Foxmail 中即可查看到如图 A-5 所示效果的邮件：

图 A-5　带内嵌文件的邮件

实战经验

对于一个 Web 应用程序来说，一般情况下，我们不推荐使用内嵌文件的邮件，用户大可将这些资源文件放在一台 Web 资源服务器上，然后简单地通过 URL 来引用这些文件。这带来了明显的好处：邮件体积缩小很多，并且提高了邮件的收发效率，同时邮件的展现效果并不会受到影响。

A.3.4　发送带附件的邮件

发送带附件的邮件和发送带内嵌文件的邮件很相似，它们都是 multipart 类型的邮件，两者都是在 MimeMessage 内容中建立多个 MIME 部分。主要区别在客户端软件里：内嵌文件显示在邮件体中，而邮件附件则显示在附件区中。下面的程序发送一封带有两个附件文件的邮件：

代码清单 A-7　MailService# sendAttachmentMail()

```
package com.baobaotao.service;
    …
public class MailService {
    …
    public void sendAttachmentMail() throws Exception {
        MimeMessage msg = sender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(msg,true,"utf-8");
        helper.setFrom("masterspring@163.com");
        helper.setTo("masterspring3@gmail.com");
        helper.setSubject("注册成功");
        helper.setText("欢迎访问宝宝淘论坛！");
        ClassPathResource file1 = new ClassPathResource("bbt.zip");
        helper.addAttachment("file01.zip",file1.getFile()); ①
        ClassPathResource file2 = new ClassPathResource("file.doc");
        helper.addAttachment("file02.doc",file2.getFile());②
        sender.send(msg);
    }
}
```

添加一个附件

再添加一个附件

在添加附件文件时，第一个入参是文件名，JavaMail 使用它的扩展名判断附件的 MIME 类型，这和内嵌文件不一样，后者是通过 Resource 的文件名确定的。用户可以使用 addAttachment()添加多个附件。邮件发送成功后，在 Foxmail 中将看到如图 A-6 所示的效果：

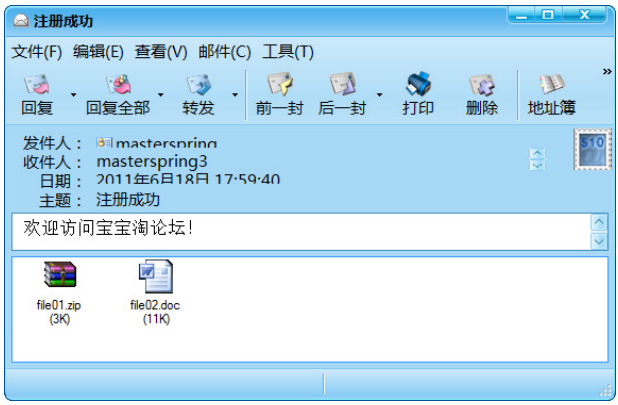

图 A-6　带附件的邮件

A.3.5　发送纯文本和 HTML 双版本的邮件

发送一般的邮件并不需要和 JavaMail 底层的 API 打交道，仅借助 MimeMessageHelper 帮助类就足以应付。如果需要创建更复杂的邮件，比如本节介绍的双版本邮件，那么就只得掀开底层 JavaMail API 的盖头了。

Spring 为直接访问 JavaMail 提供了 MimeMessagePreparator 回调接口，在该接口的 prepare(MimeMessage msg)方法中，用户可以毫无限制地访问所有 JavaMail 的 API。值得提醒的是：在回调接口方法中依然可以继续使用 MimeMessageHelper。

使用 MimeMessagePreparator 回调接口的好处是 Spring 会处理 JavaMail API 各种五花八门的检查型异常并将它们转换为 Spring Mail 的运行期异常，它使我们无须对各种烦琐无趣的 JavaMail 异常进行捕捉。

双版本邮件是指在邮件体中既提供纯文本版本的邮件，也提供 HTML 版本的邮件。这样，当用户使用不支持 HTML 的邮件客户端程序阅读邮件时，选用纯文本版本的邮件，如果客户端程序支持 HTML 则使用 HTML 版本的邮件。这种周全的设计理念越来越得到用户的推崇。以下实例展示了如何构建并发送一封双版本邮件：

代码清单 A-8　MailService：sendAlternativeMail()

```
package com.baobaotao.service;
import javax.mail.BodyPart;
import javax.mail.MessagingException;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
…
public class MailService {
    …
    public void sendAlternativeMail() throws Exception {
        MimeMessagePreparator mmp = new MimeMessagePreparator() {
            public void prepare(MimeMessage msg) throws Exception {
                MimeMessageHelper helper = new MimeMessageHelper(msg, true,"utf-8"); ①
                helper.setFrom("masterspring@163.com");
                helper.setTo("masterspring3@gmail.com");
                helper.setSubject("注册成功");

                MimeMultipart mmPart = new MimeMultipart("alternative"); ②
                msg.setContent(mmPart);

                BodyPart plainTextPart = new MimeBodyPart(); ③
                plainTextPart.setText("欢迎访问宝宝淘论坛！");
                mmPart.addBodyPart(plainTextPart);

                BodyPart htmlPart = new MimeBodyPart();④
                String htmlText = "<html><head>"
```

使用 MimeMessageHelper 设置常见的属性

创建双版本邮件内容

创建纯文本版本的邮件体

创建 HTML 版本的邮件体

```
                        + "<meta http-equiv=\"content-type\" “+
                        + "content=\"text/html; charset=utf-8\">"
                        + "</head><body><font size='20' size='30'>"
                        + "欢迎访问宝宝淘论坛</font>" + "</body></html>";
                htmlPart.setContent(htmlText, "text/html;charset=utf-8");   ④-1
                mmPart.addBodyPart(htmlPart);     必须指定编码格式，否
            }                                     则会产生中文乱码
        };
        sender.send(mmp); ⑤   使用 MimeMessagePreparator 回调接口发送邮件
    }
}
```

以上代码创建了一个 MimeMessagePreparator 的匿名类实例，JavaMailSender 会自动创建一个 MimeMessage 实例，并作为入参传给回调接口的 prepare(MimeMessage msg)方法。在回调接口中，我们可以对该 MimeMessage 进行各种内容填充操作。该匿名类乍一看挺复杂，通过下面的分析之后，相信可以很容易地掌握。

首先，我们通过 MimeMessageHelper 设置了一些常见的信息，如①所示。然后创建一个标识双版本邮件的 MimeMultipart，用于封装纯文本及 HTML 双版本的邮件内容，如②所示。接着，我们进行纯文本版本邮件部分的创建工作，并将其添加到 MimeMultipart 中，如③所示。最后，我们创建了 HTML 版本邮件部分并将其添加到 MimeMultipart 中，如④所示。至此，一个包含双版本的邮件消息就构建成功了。

在⑤处，我们使用 JavaMailSender#send(MimeMessagePreparator mimeMessagePreparator)方法将这个复杂的邮件消息发送出去。收取这个双版本的邮件，在 Foxmail 中，用户在默认情况下看到纯文件版本的邮件，可以通过点击工具栏的“HTML”按钮查看 HTML 版本的邮件，如图 A-7 所示。

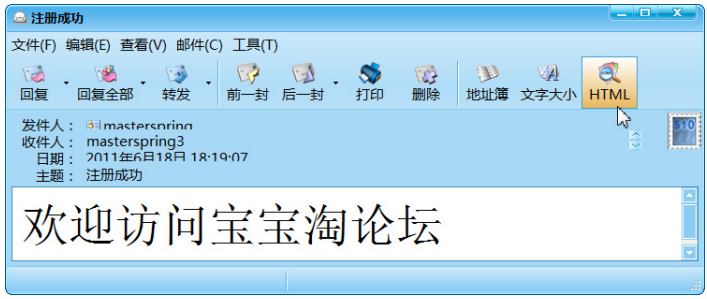

图 A-7　双版本邮件：HTML 版本的效果

A.4　在实际应用中发送邮件

前面我们编写的邮件发送程序直接在代码中构建邮件内容，对于一些简单的邮件来说，这种方式是可行的。但如果邮件体的内容很复杂，这种方式的弊端将马上暴露出来——回想一下直接使用 Servlet 构造复杂内容网页的痛苦经历吧，读者马上就会意识到直接在代码中构造复杂邮件内容是不妥当的。

对于 HTML 格式的邮件来说，除少部分内容外，大部分的 HTML 代码都是固定的，

模板技术最适合解决类似的问题。因此，在实际应用中，常采用的办法是制作好邮件模板，在发送邮件时，通过模板解析构建最终邮件内容。

发送邮件一般由具体业务流程中触发调用，如在论坛用户注册流程中，当完成用户注册后，紧接着向用户发送一封通知邮件。由于发送邮件相对来说是比较重量级的操作，它受限于邮件服务器和网络的性能，可能需要好几秒的时间。如果直接在业务流程的线程中采用同步的方式发送邮件，业务流程的响应速度将会受到很大的影响。为了降低这种影响，在实际的应用系统中，一般需要采用异步邮件发送方式，使用单独的线程发送邮件，甚至使用 JMS 消息提交邮件发送任务，由单独的邮件发送服务器负责实际邮件发送的任务。

A.4.1　使用邮件模板

在开源领域，Velocity 和 FreeMarker 是广泛使用的两个模板框架，由于 FreeMarker 的后发优势，受到的青睐度相对来说更高些。这里，我们就使用 Freemarker 模板框架来生成邮件的内容。Freemarker 的原理其实很简单，就是用动态的数据替换模板中的特殊标签，生成最终的内容。Freemarker 更多地被当成生成 Web 网页页面的模板技术，不过我们也完全可以在非 Web 的环境中使用 Freemarker。Spring 提供了对 Freemarker 的支持，我们将在第 17 章讲解这部分的知识。对于一些简单的 Freemarker 应用，相信本节所介绍的相关知识就可以满足要求了。

Spring 为 Freemarker 提供了一个 FreeMarkerConfigurer，通过这个类可以方便地创建 Freemarker 的基础设施，然后就可以在此基础上获取 Freemarker 的基础组件实例。Spring 还提供了一个降低 Freemarker 模板使用难度的 FreeMarkerTemplateUtils 工具类，通过工具类所提供的方法能够轻松地完成模板解析的任务。

下面，我们为用户注册成功通知邮件编写一个 Freemarker 格式的模板文件 registerUser.ftl，将其放到 src/mailTemplate 目录下：

```
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8">
    </head>
    <body>
        恭喜，您在宝宝淘论坛已经注册成功!您的用户ID为：
        <font size='20' size='30'>${userId}</font>①
    </body>
</html>
```

Freemarker 模板拥有各种丰富而强大的标签，在某种程度上可以将其看成一种编程语言。①处的${xxx}标签代表一个可被替换的属性变量。该标签支持级联属性，如${user.userId}将对应 user 对象的 userId 属性。下面，我们使用 FreeMarker 模板技术构造邮件的内容。

代码清单 A-9　MailService# sendTemplateMail()

```
package com.baobaotao.service;
```

```
import org.springframework.ui.freemarker.FreeMarkerTemplateUtils;
import org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer;
import freemarker.template.Template;
…
@Service
public class MailService {
    …                                                    FreeMarker 基础组件类
    @Autowired
    private FreeMarkerConfigurer freeMarkerConfigurer; ①

    private String getMailText(int userId){②   通过模板构造邮件内容
        String htmlText = null;

        try {                                  通过指定模板名获取 Freemarker 模板实例
          Template tpl=freeMarkerConfigurer.getConfiguration().getTemplate("registerUser.ftl");②-1
            Map map = new HashMap();②-2   通过 Map 传递动态数据
            map.put("userId", userId); ②-3
                                          注意动态数据的名字必须和模
                                          板标签中指定属性相匹配

            htmlText = FreeMarkerTemplateUtils.processTemplateIntoString(tpl,map); ②-4
        } catch (Exception e) {                  解析模板并替换动态数
            throw new RuntimeException(e);       据，产生最终的内容
        }
        return htmlText;
    }

    public void sendTemplateMail(int userId) throws MessagingException {
        MimeMessage msg = sender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(msg, false, "utf-8");
        helper.setFrom("masterspring@163.com");
        helper.setTo("masterspring3@gmail.com");
        helper.setSubject("注册成功:基于模板");
        String htmlText = getMailText(userId); ③
        helper.setText(htmlText, true);          使用模板产生
        sender.send(msg);                        HTML 邮件体内容
    }
}
```

我们在 MailService 中新增了一个 sendTemplateMail()方法，它和 sendHtmlMail()方法唯一的区别在于产生 HTML 邮件体内容的方式。sendTemplateMail()通过 Freemarker 模板技术产生 HTML 邮件体内容，这样程序代码中用于组织 HTML 内容的难看代码被完美地切除了。

当然，sendTemplateMail()正常工作的前提是必须拥有一个可用的 FreeMarkerConfigurer 实例，这通过 Spring 配置文件来实现：

```
<bean id="freeMarkerConfigurer"
    class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer"
    p:templateLoaderPath="classpath:mailTemplate/>①   指定模板目录
```

```
    <property name="freemarkerSettings">②          设置 Freemarker 环境属性      刷新模板的周
        <props>                                                                  期，单位为秒
            <prop key="template_update_delay">1800</prop>②-1
            <prop key="default_encoding">UTF-8</prop>②-2
            <prop key="locale">zh_CN</prop>②-3
        </props>                                    本地化设置
    </property>
</bean>
```

模板的编码格式

正是因为在①处设置了模板目录，所以我们在代码清单 A-8 的②-1 处才可以直接使用模板文件名定位到具体的模板文件上。模板文件使用 UTF-8 编码格式，以避免麻烦的中文乱码问题。通过 template_update_delay 属性的设置，可以让 Freemarker 定期刷新模板，这样我们就可以在应用程序不重启的情况下更新模板了。

A.4.2　异步发送邮件

在 13.5 节中，我们介绍了 Spring 提供的 TaskExecutor，它的很多实现类可以提供异步任务执行的功能。这里，我们使用 TaskExecutor 对上一节的 sendTemplateMail ()方法进行封装，提供异步执行的功能。其代码如代码清单 A-10 所示：

代码清单 A-10　MailService# sendAsyncMail()

```
package com.baobaotao.service;
import org.springframework.core.task.TaskExecutor;
…
import freemarker.template.Template;
@Service
public class MailService {                                拥有异步执行能
    …                                                     力的任务执行器
    @Autowired
    private TaskExecutor taskExecutor; ①

    public void sendAsyncMail(final int userId){ ②
        taskExecutor.execute(new Runnable(){          异步调用 sendTemplateMail ()方法
            public void run() {
                try {
                    sendTemplateMail(userId);
                    System.out.println("邮件发送成功！");
                } catch (Exception e) {
                    System.out.println("邮件发送失败！，异常信息：" + e.getMessage());
                }
            }
        });
    }
}
```

在②处，我们通过一个匿名类封装了 sendTemplateMail(userId)方法的调用，并通过 TaskExecutor 进行异步执行。我们使用基于 JDK 5.0 新特性的 ThreadPoolTaskExecutor 作为

任务执行器：

```
<bean id="taskExecutor"
      class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor"①
      p:corePoolSize="10"                                    声明一个基于线程池
      p:maxPoolSize="30"/>                                   的异步任务执行器
```

由于 ThreadPoolTaskExecutor 会在 Spring 容器关闭时自动关闭，如果测试 MailService#sendAsyncMail()的程序必须提供一段模拟的运行时间，由于异步程序的测试比较特别，为此我们提供了一个典型的测试实例，如代码清单 A-11 所示：

代码清单 A-11 测试异步邮件发送的代码

```
package com.baobaotao.service;

import org.junit.Ignore;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath:applicationContext.xml" })
public class MailServiceTest{
    @Autowired
    private MailService mailService;

    public void testSendAsyncMail() {
        try {
            mailService.sendAsyncMail(100); ①
            System.out.println("调用结束！ ");
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            Thread.sleep(15000); ②
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

这行代码是异步执行的

防止 Spring 容器关闭，模拟系统运行一段时间

由于采用了异步任务执行的方式，①处的代码将马上返回并迅速执行后面的代码。我们通过 Spring 的测试框架测试 sendAsyncMail()方法，需要特别注意的是在②处，我们让主线程睡眠一小段时间以模拟程序的运行，如果主线程结束后，Spring 容器马上就会关闭，进而导致 ThreadPoolTaskExecutor 过早停止，这样的话，异步发送邮件的任务就会被取消。关于 Spring 测试框架的更多知识，参见第 A 章的内容。

A.5　小结

发送邮件是最常见的业务需求之一，JavaMail 本身的 API 比较复杂，而且没有根据用户的需要进行适当的功能划分，所以直接使用 JavaMail 发送邮件不是一件令人愉快的差事，Spring 大幅简化邮件发送程序编写的难度。Spring 对 JavaMail 发送邮件的支持主要体现在两大类上：JavaMailSender 和 MimeMessageHelper，前者让我们可以方便快速地使用 Bean 风格搭建起 JavaMail 环境，而后者屏蔽了操作 MimeMessage 对象的难度。对于复杂的邮件，用户可以通过 MimeMessagePreparator 回调接口直接使用 JavaMail 底层的 API 完成。

虽然 Spring 屏蔽了大量 JavaMail 底层的 API，但是并不意味着我们就无须了解 JavaMail 的基础知识。在本章中，我们对 JavaMail 的基础知识进行了学习，掌握好 JavaMail 的基础知识对于开发基于 Spring 的邮件发送程序是非常有帮助的。

在本章中，我们讲解了如何发送各种类型的邮件，这包括简单的纯文本邮件、HTML 邮件、带内嵌文件的邮件、带附件的邮件以及同时拥有纯文本和 HTML 的双版本邮件。相信这些邮件类型已经可以满足大部分系统发送邮件的要求。

在实际应用系统中，如何构造邮件内容以及如何进行邮件发送是两个特别值得关注的问题，因为前者不但涉及程序编码的问题，也涉及邮件内容管理的问题。而后者的不当处理将严重影响系统的响应性能，导致不良的用户体验，最终直接影响用户对软件的整体评价。在实际系统中，一般通过模板技术和异步发送的方式解决这两个问题。