課題実験報告書

～JavaMail を用いたメール送受信サーブレット～

4年　情報工学科　16 番

鈴木　雄太

【目的】

JavaMail を用いて、インターネットのホームページ上でメール送受信をするサーブレットを作成する。メールにはファイルも添付できるようにする。

【概要】

データの流れの図を図 1 に示す。

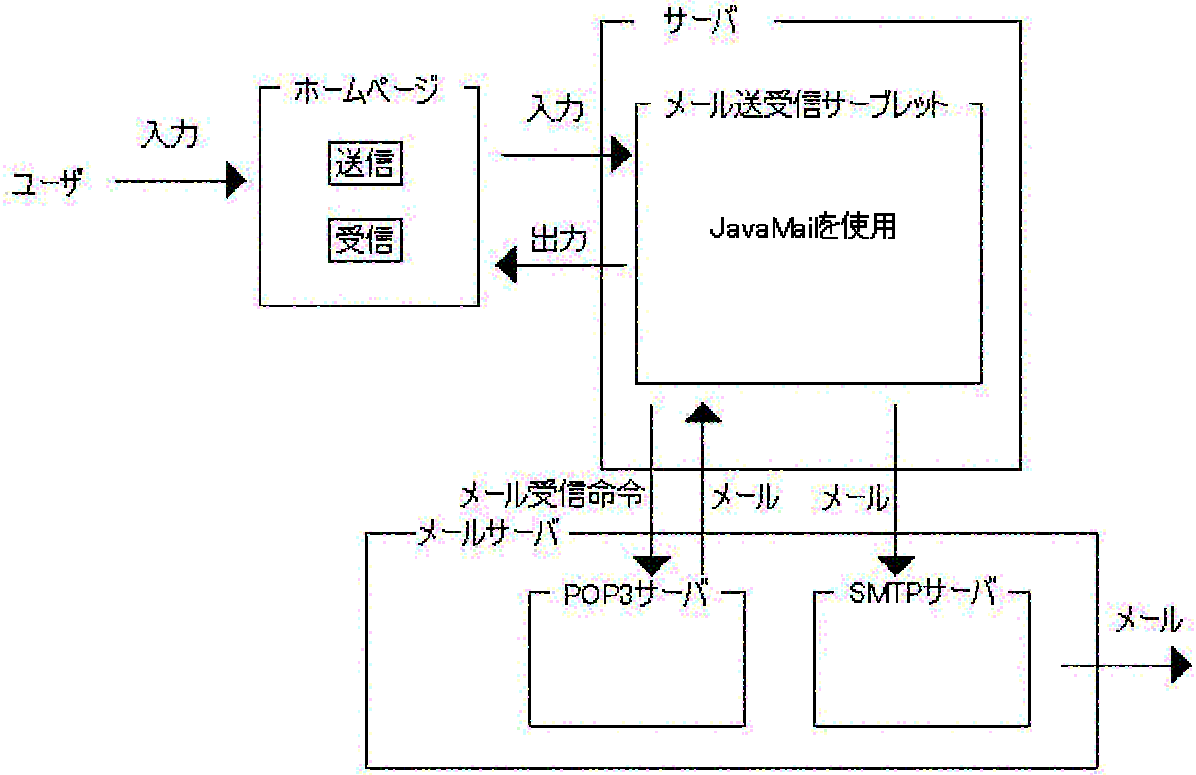


図１　データの流れ

動作としては

①ユーザがサーブレットへデータ送(受)信用の HTML にデータを入力。

②送(受)信ボタンを押す。

③WEB サーバに置いてあるサーブレットに、必要なデータが送信される。

④サーブレットはメールの処理を実行する。

⑤送信時は SMTP サーバへメールを送り、そこから宛先へメールを送信する。受信時は POP サーバへメール受信命令を送り、メールを受信する。

⑥結果ページを HTML に表示する。

○JavaMail とは

JavaMail は Sun が提供している Java 関連製品の一つで、Java を用いてメールの送受信を行うための API パッケージである。SMTP、POP3、IMAP4 などがサポートされている。

JavaMail 本体はインターフェースの定義が主で実際に通信を行うプロトコルの実装は「プロバイダー」と呼ばれる部分で実装を行う。従ってプロバイダーを交換してやる事で新しいプロトコルに対応する事が可能。その際、アプリケーション側は使用するプロバイダーを変える記述をする以外の変更をする必要はない。

○通信プロトコル

SMTP ： インターネットやイントラネットで電子メールを送信するためのプロトコル。サーバ間でメールのやり取りをしたり、クライアントがサーバにメールを送信する際に用いられる。

POP3 ： インターネットやイントラネット上で、電子メールを保存しているサーバからメールを受信するためのプロトコル。現在最も広く普及している。電子メールの送信に使われる SMTP とセットで利用される。ユーザがタイトルや発信者を確認する前に、クライアントが全メールを受信してしまうため、発信者やタイトルの一覧を見てから受信するかどうか決められる IMAP を POP の代わりに利用する場合もある。

IMAP ： インターネットやイントラネット上で、電子メールを保存しているサーバからメールを受信するためのプロトコル。POP と違って、メールはサーバ上のメールボックスで管理され、タイトルや発信者を見て受信するかどうかを決めることができる。

IMAP と POP の比較

機能	POP	IMAP
メールの保管場所	基本的に、クライアント側で保管する。 サーバ側に保持して置くことも出来るが、フォルダ毎に分類などはできない。	サーバ側で保管する。 フォルダもサーバ側に作成する。
メールの検索	クライアント側で行う。	サーバ側で行う。

〇サーブレットとは

　サーブレットはサーバー側で動作する Java プログラムである。クライアントのリクエストをサーバー側で処理して HTML を動的に生成して表示する。CGI 等とよく似た目的で使用する。

　サーブレットは起動後、サーバにロードされて処理を行うが、処理終了後も破棄されずに待機状態になり、メモリに常駐する。

　メリットとしては

・処理終了後に破棄されないので、リクエスト毎にプロセスを起動する CGI より速度が速い。

・豊富な JavaAPI を利用できる。

　デメリットとしては

・Java Server API に対応した Web サーバを使っているプロバイダが少ない。

【環境】

・Java 開発環境　：　Java 2 sdk, Standart Edition 1.4.1
・JavaMail　　：　JavaMail 1.2(mail.jar)
・　WEB サーバのスペック
　OS　　：　RedHat Linux
　CPU　：300MHz
　Memory：128MB
　HDD　：6GB

【必要なもの】

・Java 開発環境（Java 2 sdk）
・mail.jar（JavaMail 本体）
・activation.jar（JAF:JavaBeans Activation Framework）
・pop3.jar（POP3 プロバイダー）

} Sun Microsystems 社のホームページから無料でダウンロードできる。

・メールサーバ（POP3 サーバ、STMP サーバ）
・Java Server API に対応した Web サーバ

※activation.jar(JAF)はデータの種類によって、適切な処理をしてくれる API。

【実験日程予定】

一週目　：　テーマの決定
二週目　：　送信プログラム及び送信用 HTML の作成
三週目　：　受信プログラム及び受信用 HTML の作成
四週目　：　発表の準備
五週目　：　室内発表

また、遅れている場合、作業は実験のない日も行う。

【プログラムの簡単な流れ】

・メール送信プログラム
SMTP サーバーのアドレスを指定
↓
送信元メールアドレスと送信者名を指定
↓
送信先メールアドレスを指定
↓
メールのタイトルを指定
↓
メールの内容を指定
↓
メールの形式を指定
↓
送信

・メール受信プログラム
ホストアドレス、アカウント、パスワードを指定
↓
接続
↓
フォルダを開く
↓
フォルダーにあるメッセージの数を取得
↓
メッセージを取得
↓
メッセージを表示
↓
フォルダーを閉じる

【実験方法】

(1)準備

・JavaMail、JAF のアーカイブファイルをダウンロード

Sun の JavaMail のホームページと JAF のホームページからダウンロードする。

・ダウンロードした圧縮ファイルを展開

UNIX の unzip コマンドを使う。

・CLASSPATH の設定

使用する mail.jar、activation.jar、pop3.jar の CLASSPATH の設定をしておく。

CLASSPATH の役割は、作成したクラスの中で利用している他のクラスがどこにあるかを、javac や java コマンドなどに教えること。

(2)プログラム及びデータ送信用 HTML の作成

（２－１）テスト作成

①データを受け取って、表示するだけのサーブレットを作成。

②データを送る HTML の作成。

（２－２）メール送信プログラムの作成

①メール送信プログラムを作成する。

②コンパイル

③テスト実行

④デバッグ

⑤データ送信用 HTML の作成。

⑥プログラムをサーブレットにする。

⑦再度デバッグ

（２－３）メール受信プログラムの作成

①メール受信プログラムを作成する。

②コンパイル

③テスト実行

④デバッグ

⑤データ受信用 HTML の作成。

⑥プログラムをサーブレットにする。

⑦再度デバッグ

【結果】

WEB 上でメールを送信するプログラムが完成した。目的では、ファイルの添付もするとしていたが、達成できなかった。

受信側は作成しなかった。

【考察】

(1)今回のプログラムのメリット

サーブレットで作ったので、WEB 上で実行できるため、インターネットに接続できる環境ならどのような OS で、どのようなブラウザを使用していても実行できる。また、サーバ側で処理するため、クライアント側にプログラムを置く必要がない。

プログラム作成側は、JavaMail を用いたため、SMTP などの通信プロトコルを気にしないでプログラミングできる。

(2)今回のプログラムのデメリット

実験で作成したメール送信プログラムは、送信元のメールアドレスが固定されてしまっているので、誰が使っても送信元が同じになってしまう。そのために匿名性が出てしまい、悪用される恐れがある。

これを解決するには、ユーザ認証の機能をつければよい。

(3)将来性

今回のプログラムを応用して、送信にいろいろな機能(ファイル添付、Ｃｃ・Bcc など)を付け、受信プログラムも作成すると、フリーメールサービスのようなものも作れる。

【実験日程】

１週目 ： テーマを考えた。JavaMail を使うことに決定。

２週目 ： mail.jar、activation.jar、pop3.jar の CLASSPATH を設定。JavaMail のパッケージに入っていた DEMO プログラム(msgsendsample.java)を実行。

３週目 ： メール送信プログラムにメールの内容(宛先、タイトル、本文)を渡す HTML を作成。その HTML からデータを受け取って、そのまま表示するサーブレットを作成。

４週目 ： メール送信プログラムを作成。送り先メールアドレスがメールアドレスとしてありえない形（@がない等）だったら、エラーを表示するようにした。

５週目 ： 室内発表をした。全体発表に向けての準備をした。

【質問への回答】
室内発表・全体発表で出た質問と回答を下に記す。

○室内発表
[Q1]．CGI とは何か？
[A]．　Common Gateway Interface の略。Web サーバー上に実行可能なスクリプトやプログラムがあり、ブラウザから何らかの入力があった時に動く。必要な場合は、その他のプログラムを呼び出す。CGI は、CERN や NCSA などの Web サーバが使っている、サーバとプログラムとの間のやり取りの方法。この CGI や SSI (Server Side Include) などを、ゲートウェイスクリプトと呼んでいる。

[Q2]．API とは何か？
[A]．　Application Program Interface の略。　API は、OS がアプリケーションに対して公開しているプログラムインターフェイスで、アプリケーションは、基本的にすべての処理をこの API を経由して行なう。現在一般的な OS の API は関数の形式をとっており、アプリケーションからは、適当なパラメータ（引数）を指定して、API の関数を呼び出す。

[Q3]．JavaMail は実際に使われているか？
[A]．　自動的にメールを送信するようにしたりして、実際に WEB 上で使われていると思われる。

[Q4]．悪用できるか？
[A]．　匿名でいたずらメールを送るなど、悪用は出来る。今回作った送信プログラムは、送信元メールアドレスが固定されているので、悪用される可能性がある。ユーザ認証機能をつければ防げると思う。

○　全体発表
[Q1]．フリーメールとは何か？
[A]．　フリーメールはその電子メールアドレスを無料で得ることのできるサービスのこと。通常、電子メールの送受信にはメールソフトが必要だが、フリーメールのサービスではインターネット上でメールを読み書きできるので、それが必要ではない。有名なところでは、YAHOO などがこのサービスを行っている。

[Q2]．いろいろなブラウザで試したか？
[A]．　Mozilla、Internet Explorer、Netscape Communicator 等で確認した。

【感想】

今回の課題実験は、初めて Java という言語を使い、実験前までは聞いたこともなかった JavaMail を用いて、サーブレットを作成するという、基礎知識がほとんどないところから始まり、使用した OS は Linux だった。初めは不安だったが、プログラムがうまく動いたときの喜びと、「やってみたらなんとかなる」という自信がついてよかったと思う。

また、全体発表の経験は来年の卒研の発表に役立てたいと思う。

ソフトウェアドキュメント

―メール送信サーブレット―

開発環境：Java 2 sdk, Standart Edition 1.4.1

使用 OS　：　RedHat Linux

開発者　：　４年　情報工学科　１６番

鈴木　雄太

【目的】

WEB 上でメールを送信するプログラムを作成。

【プログラムの概要】

作成したプログラムはメール送信サーブレットである。

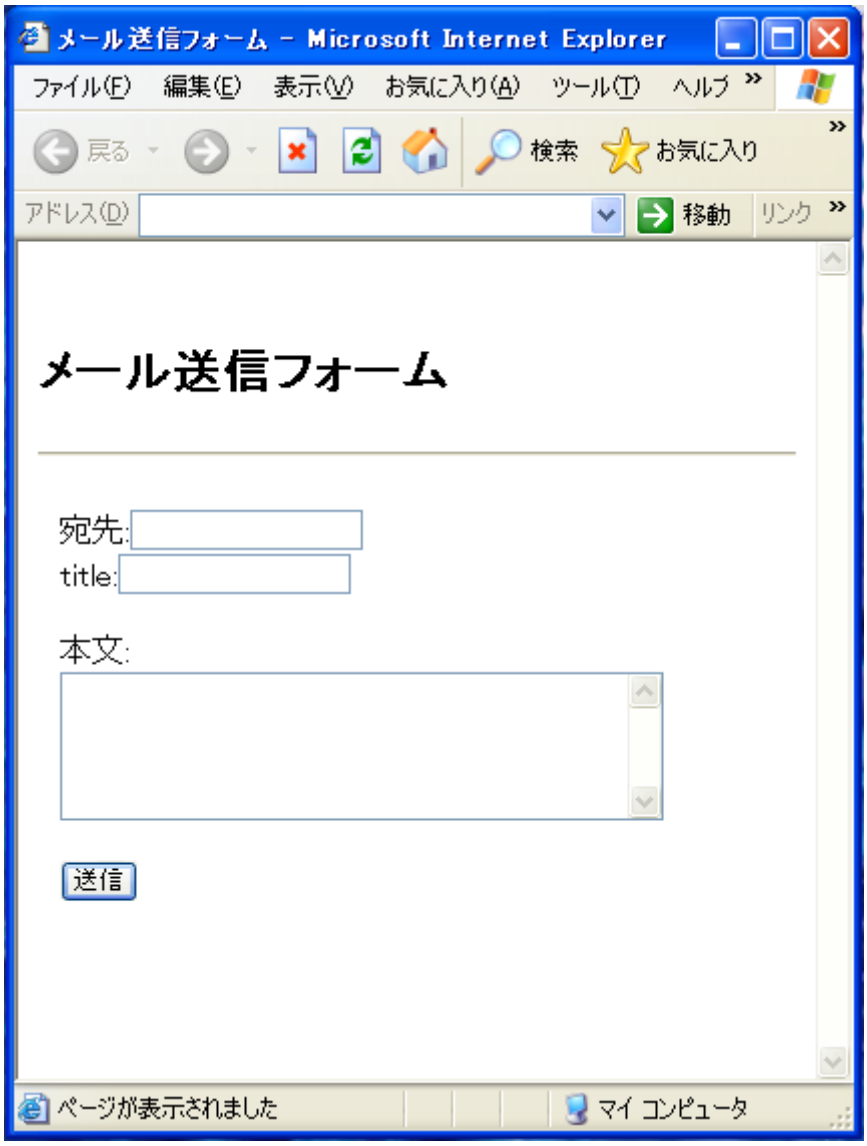
プログラム内では、

・SMTP サーバーのアドレスを指定
・送信元メールアドレスと送信者名を指定
・送信先メールアドレスを指定
・メールのタイトルを指定
・メールの内容を指定
・メールの形式を指定
・送信

という操作をしている。

また、サーブレットにメールのタイトル、宛先アドレス、本文を送信する HTML も作成した。

【プログラムの実行方法】



①上図の宛先に送信先アドレスを入力。
②title にメールのタイトルを入力。
③本文にメールの本文を入力。
④送信ボタンを押す。
これでメールが送信される。

【PAD 図】


出力する形式の設定
送信先メールアドレス、
タイトル、本文の取得
送信元メールアドレス、
送信元メールサーバの設定
送信内容の出力
（HTML）
新しいプロパティ生成
SMTPメールサーバを表す
プロパティを設定
デフォルトのメール処理セッション
を取得
MIMEメッセージオブジェクト生成
送信元メールアドレス指定
送信先メールアドレス指定
タイトル指定
日付指定
本文指定
メール送信
送信内容の出力
もしエラーがあったら、表示

【プログラムリスト】

```
public class testSend extends HttpServlet {
        public void doPost(HttpServletRequest req,HttpServletResponse res)
                throws ServletException, IOException {

                res.setContentType("text/html; charset=EUC-jp");
                PrintWriter out = res.getWriter();

                String to = req.getParameter("inAtesaki");          //宛先
                String subject = req.getParameter("inTitle");       //タイトル
                String honbun = req.getParameter("honbun");         //メール本文
                String from = "j99317@hakodate-ct.ac.jp";           //送信元
                String host = "10.30.2.27";                         //ホストアドレス


                out.println("<html><head>");
                out.println("<title>送信メール");
                out.println("</title>");
                out.println("</head>");
                out.println("");
                out.println("<h1>送信メール</h1>");
                out.println("<h2>");
                out.println(subject + "<br>");
                out.println(honbun);
                out.println("</h2>");


                Properties props = new Properties();
        //SMTP サーバのアドレスを指定
                props.put("mail.smtp.host", host);

        //タイムアウト
                props.put("mail.smtp.connectiontimeout",new Integer(10000));
                props.put("mail.smtp.timeout",new Integer(10000));
```

```
                Session session = Session.getDefaultInstance(props, null);


                try {

                        Message msg = new MimeMessage(session);
        //送信元メールアドレスの指定
                        msg.setFrom(new InternetAddress(from));
                        InternetAddress[] address = {new InternetAddress(to)};
        //送信先メールアドレスの指定
                        msg.setRecipients(Message.RecipientType.TO, address);
        //メールタイトルの指定
                        msg.setSubject(req.getParameter("inTitle"));
        //日付を指定
                        msg.setSentDate(new Date());
        //メールの内容(本文)を指定
                        msg.setText(req.getParameter("honbun"));

                        Transport.send(msg);
                        out.println("<p></p>");
                        out.println("<pre>");
                      ByteArrayOutputStream bos = new ByteArrayOutputStream();
                        msg.writeTo(bos);
                        bos.close();
                        out.print(bos.toString("EUC-jp"));
                        out.println("</pre>");

                } catch(MessagingException ex) {
                        out.println("<p>送信できませんでした。");
                       ex.printStackTrace(out);
                }

                out.println("</body></html>");
        }
}
```

【データ送信用 HTML のソース】

```
<html>
<head>
	<title>メール送信フォーム</title>
	<meta http-equiv="Content-Type" content="text/html; charset=EUC-jp">
</head>
<body>
	<h2>メール送信フォーム/h2>
	<form method=post action="http://10.30.2.30/testpage/testsend">

	<p>
		<label>宛先：<input type="text" name="inAtesaki"></label><br>
		<label>title:<input type="text" name="inTitle"></label><br>
	</p>
	<p>
		<label>本文:<br>
		<textarea name="honbun" rows="5" cols="40"></textarea>
	</p>
	<p>
		<input type="submit" value="送信">
	</p>
	</form>
</body>
</html>
```