さらに進んだ インターネット・セキュリティ

サーバー構築/運用術

最終回 メール(後編)

前々回はMTA間のメール配送について，前回はその前段階であるユーザーがメールを送信してからMTAに届くまでについて説明しました。メール3回目の今回は，メール配送の最終段階であるメールがあて先MTAに届いてから受信者が読むまでについて説明します。

(ケイ・ラボラトリー 仙石 浩明)

2000年4月号から始まり，2年間続いたこの連載*1ですが，今回が最終回となります。長らくのご愛読ありがとうございました。私が1989年ごろから10年余りインターネットに関わり続けてきた経験を元に，思いつくままにサーバー構築に必須と思われる技術について説明してきましたがいかがだったでしょうか。

サーバー構築に関する記事というと，インストールの方法から設定方法までを解説するのが一般的ですが，今回の連載ではあえてインストールに関しては一切触れず，ツールの使用方法や設定方法についての説明も最小限にとどめました。使用方法や設定方法などのいわゆるハウツー(howto)に関しては，分かりやすく説明した本や記事がちまたにあふれていますから，いまさら書いても仕方がありませんし，個々のソフトウエアの設定方法にまで言及していては，本質が見えにくくなってしまうと考えたからです。

目まぐるしい勢いで進化し続けるインターネットですが，サーバー構築や運用に限れば，最先端の技術は必ずしも必要ではなく，10年前の技術で足りてしまうことがほとんどだと思います。ドッグ・イヤーと呼ばれるこの世界で10年前というと絶望的なくらい昔のように思えますが，TCP/IPが生まれたのは20年以上前のことですし，現在のインターネットを支える主要なプロトコルのほとんどは，10年前ぐらいまでにほとんどが確立しています。

最近規定された新しいプロトコルも，そのほとんどが昔からあるプロトコルを拡張しただけのものにすぎません。

重要なことは，表層的な「新技術」に踊らされないことと，そのために長年変わらない普遍的で本質的な技術を確実に身に付けることではないでしょうか。その上で，何が変わったのか，なぜ変わったのか理解するよう努めれば，ハウツー本など読まなくても新技術を使いこなすことができるようになるはずです。

今回の連載では，なるべくインターネットの歴史や技術の変遷に触れるよう努めました。古きを温めれば現在の技術の本質が見えてくるだけでなく，今後どのような方向へ発展していくかのヒントをつかめることでしょう。

そして，連載の副題にもなっている「インターネット・セキュリティ」ですが，サーバーのセキュリティで一番重要なの

*1 現在の前の連載「実践で学ぶ，一歩進んだサーバー構築・運用術」も含みます。

```
/bin/mail -d sengoku < mail.txt↵
```

図1　/bin/mailを使ってメールをメール・ボックスへ配送　　↵このマークで改行

```
qmail-local sengoku /home/sengoku sengoku '' '' 'gcd.org' '' ./Mailbox < mail.txt↵
```

図2　qmail-localを使ってメールをメール・ボックスへ配送

はサーバーやネットワークの本質を理解することです。ハウツーをいくら積み上げても，その場限りの対策にすぎません。遠回りのように思えるかも知れませんが，本質を理解することこそが，どのような事態にも対応できる技術者になる唯一の道と言えるでしょう。この連載が，本質を理解する際の一助となったとしたら幸いです。

メール・ボックス

本連載2001年12月号で，MTA（Message Transfer Agent，メール配送エージェント）間のメール配送について説明し，2002年1月号でその前段階であるユーザーがメールを送信してからMTAに届くまでについて説明しました。3回目の今回は，メール配送の最終段階であるメールがあて先MTAへ届いてからユーザーが読むまでについて説明します。

メールが届いたとき，その直後にユーザーがメールを読むとは限りませんから，ユーザーがメールを読む操作を行うまで，メールを保存しておく必要があります。この保存しておくための場所をメール・ボックスと呼びます。

sendmailやqmailなどの今日最もよく使われているMTAでは，MTAが直接メール・ボックスにメールを配送するのではなく，メール・ボックスへ配送するプログラムを呼び出す方式を採用しています。例えばsendmailでは，メール・ボックスへ配送するプログラムとして，/bin/mailプログラム＊2がよく使われます。ユーザーsengokuのメール・ボックスに，メールmail.txtを配送する実行例を**図1**に示します。

また，qmail付属のqmail-localプログラム＊3は，**図2**のように実行することにより同様のことができます。/bin/mailの場合に比べて引数が多いのですが，これはプログラムの単純化に一役買っています。例えば第1，2引数はそれぞれ「ユーザーID」と「ユーザーのホーム・ディレクトリ」ですが，どちらもプログラム内部で生成可能なので，引数として渡す必要は無いのですが，呼び出し側であるMTAから渡してもらうことにより生成のためのコードを省略して単純化しているわけです。

なお，第3〜6引数はメールのあて先アドレス（図2の例では「sengoku@gcd.org」）を分割して表現したものであり，第7引数は送信者のアドレスを，第8引数はデフォルトのメール・ボックスを，それぞれ指定します。

このように，MTAとメール・ボックスへ配送するプログラムを分離することにより，後者のプログラムを変更するだけで異なる形式のメール・ボックスに対応することが可能になります。

メール・ボックスの形式には，mbox形式やmaildir形式などがあります。

mbox形式

前述した/bin/mailプログラムで用いられている形式です。メール・ボックス全体が1つのファイルになります。複数のメールを1つのファイルにまとめるために，それぞれのメールの間に区切り

＊2　メール・ボックスへの配送だけでなく，メール・ボックスを直接アクセスする方式のMUAとしての機能も持っています。最近のsendmailには，メール・ボックスへの配送に機能を絞ったmail.localプログラムが付属しています。また，RedHatなどLinuxの主要ディストリビューションでは，メール・ボックスへの配送用プログラムとしてprocmailを利用しているようです。

＊3　qmail-localは，メール・ボックスへの配送だけでなく，後述するように~/.qmailで指定したプログラムを呼び出すこともできます。

```
From postmaster@gcd.org Fri, 31 Dec 15:01:02 1999
Received: from localhost (HELO asao.gcd.org) (sengoku@127.0.0.1)
  by localhost with SMTP; 1 Jan 2000 00:01:02 +0900
Date: Sat, 01 Jan 2000 00:01:02 +0900
From: postmaster@gcd.org
To: sengoku@gcd.org
Date: Sat, 01 Jan 2000 00:01:02 +0900

仙石です。これは SMTP 説明用のテストメールです。

#5403.                                                    仙石 浩明
http://www.gcd.org/sengoku/            Hiroaki Sengoku <sengoku@gcd.org>

From test@gcd.org Sat, 10 Nov 08:22:07 2001
Received: from unknown (HELO server.aes.nt) (212.72.XX.51)
  by asaogw.gcd.org with SMTP; 10 Nov 2001 17:22:07 +0900
Received: from asao.gcd.org ([210.156.250.240]) by server.aes.nt with Microsoft SMTPSVC(5.0.2195.2966);
         Sat, 10 Nov 2001 12:12:52 +0400
From: test@gcd.org
To: postmaster@gcd.org
Subject: test
Return-Path: test@gcd.org
Message-ID: <SERVERyF1BcI3Em6kwV00000903@server.aes.nt>
Date: 10 Nov 2001 12:13:41 +0400

This is a test mail, sorry.

From sengoku@gcd.org Sun, 06 Jan 07:08:41 2002
Received: from localhost (HELO asao.gcd.org) (sengoku@127.0.0.1)
  by localhost with SMTP; 6 Jan 2002 16:08:41 +0900
Date: Sun, 06 Jan 2002 16:08:41 +0900
Message-ID: <vpwuywj6rq.wl@asao.gcd.org>
From: sengoku@gcd.org
To: sengoku@gcd.org

>From のテスト
```

図3　mbox形式のメール・ボックス

が必要になりますが, この形式では区切りとして空行とそれに続く「From 」(Fromの後に空白文字)で始まる行を用います。例えば**図3**に示すようなファイルになります。

注意しなければならないのは, メールの本文に「From 」で始まる行がある場合です。その前の行が空行であれば, メールの区切りと判断されて2通のメールとして扱われてしまいます。そこでmbox形式では,「From 」で始まる行がある場合は, その前に「>」を追加した

```
Content-Length: 20
```

図4　ヘッダーに追加されるフィールド

上でメール・ボックスに入れます。例えば図3の3通目のメールは本文に「>From のテスト」という行がありますが，この行はMTAに届いた時点では「From のテスト」という行であり，/bin/mailが「>」を行頭に追加した上でメール・ボックスのファイルに追加したのだろうと推測できます。

ところが，元々「>From のテスト」という内容のメールを送信者が送った可能性ももちろん有り得ます。このままではどちらであるか判別する方法はないので，mboxの改良版が提案されました。

●「From 」で始まる行だけでなく，その前に「>」が複数個ついている行の前にも，「>」を追加する方法

つまり「 >From 」という行は「>>From 」に変換されてメール・ボックスに入れられます。メール・ボックスからメールを読み出すときは，行頭から「>」が1個以上あって，続いて「From 」がある行の最初の「>」を取り除けば，MTA が受け取ったときと同一のメール本文を復元することができます。

●メール本文の長さ(バイト数)を値とする，「Content-Length:」フィールドをメール・ヘッダーに追加する方法

例えば図3の3通目のメールは本文「From のテスト」が20バイト＊4なので，**図4**に示すフィールドがヘッダーに追加されます。この形式のメール・ボックスからメールを読み出すときは，「From 」区切りに続いてヘッダーを読み込んだ後，「Content-Length:」フィールドで指定されたバイト数だけ本文を読み込めば，次の「From 」区切りを見つけることができます。本文は一括して読み込むので，「From 」で始まる行があっても，そこを区切りと誤認識することは避けられます。

メールの状態

メール・ボックス内のメールは，MUA (Message User Agent，いわゆるメーラー)を介して＊5ユーザーが読むことになりますが，それぞれのメールが既に読んだメールか，あるいはまだ読んでないメールか，それとも新しく届いたメールなのか，という状態を記録しておく必要があります。mbox形式のメール・ボックスでは，このような状態を記録するために，それぞれのメールのヘッダーに「Status: 」フィールドを追加します。例えば新しく届いたメールの場合，「Status: 」フィールドが無いのでMUAは新着メールとして扱い，MUAは

```
Status: O
```

というフィールドをメールのヘッダーに追加します。そして，ユーザーがこのメールを読むと，MUAはこのフィールドを

```
Status: RO
```

に書き換えて，既に読んだメールとして扱います。また，ユーザーがメールを削除する操作を行った場合は，MUAは該当メールの部分を削除したファイルでメール・ボックスを上書きすることになります。

排他制御

いずれの場合も，メール・ボックスの

＊4　「のテスト」はJIS コードだと14バイトなので，「From 」の5バイトと，改行コードの1バイトを合計して，20バイトになります。
＊5　MUAがメール・ボックスと異なるマシン上で動いている場合など，後述するようにメール・ボックスにアクセスするのはPOPサーバー等になります。ここでは分かりやすいように，単にMUAがメール・ボックスにアクセスするなどと記述することにします。

ファイルに対する書き込みが必要になりますが，メールが新たに届いた場合もメール・ボックスの末尾に新着メールを書き込む必要があります。つまり，MTAからの書き込み*6とMUAからの書き込みが非同期で発生しますから，排他制御を行って書き込みが同時に発生しないようにすることが必須です。

排他制御の方法としては，flock等のシステム・コールを用いる方法と，ロック・ファイルを作る方法とがありますが，MTA側とMUA側で同じ方法を用いる必要があります。

システム・コールを用いる方法では，NFS（Network File System）環境で利用できなかったり，実装に問題があったりするので，ロック・ファイルを作る方法が望ましいのですが，どのディレクトリにロック・ファイルを作るのか，という問題があります。

伝統的な /bin/mailプログラムでは，すべてのユーザーのメール・ボックスを1つのディレクトリ，例えば /usr/spool/mailに格納します。ユーザーsengokuのメール・ボックスであれば，ファイル/usr/spool/mail/sengokuを使う，といった具合です。

もし，ロック・ファイルを同じディレクトリに作る場合*7，ディレクトリのパーミッションが問題になります。もしMUAが各ユーザーの権限でメール・ボックスにアクセスする場合は，各ユーザーがディレクトリに自由に書き込めるようにしなければならず，セキュリティ上の問題が発生する可能性があるでしょう。

従って，全ユーザーのメール・ボックスを同じディレクトリに格納するのではなく，それぞれのユーザーのディレクトリ，例えば各ユーザーのホーム・ディレクトリに格納する方が望ましいと言えます。

maildir形式

mbox形式の欠点を解消する目的で設計されたのがmaildir形式です。mbox形式のように，メール・ボックス内の全メールを1つのファイルにまとめるのではなく，一つひとつのメールがそれぞれ別々のファイルとして保存されます。つまりmaildir形式では，メール・ボックスはディレクトリになります。

メールの状態

maildir形式のディレクトリには，「tmp」「new」「cur」の3つのサブディレクトリがあり，その中に各メールのファイルがタイム・スタンプ付きのユニークなファイル名で保存されます。例えば「Sun, 06 Jan 2002 16:08:41」に，メール・ボックスへ新着メールを書き込むとき，書き込むプロセスのIDが「28546」で，プロセスが実行されたホストの名前が「asao.gcd.org」であるならば，ファイル名は「1010300921.28546.asao.gcd.org」になります。ここで「1010300921」は，「Thu, 01 Jan 1970 09:00:00」*8からの通算秒数を意味していて，「Sun, 06 Jan 2002 16:08:41」を表わしています。

「new」は新着メールを入れておくディレクトリで，MUAがメール・ボックスにアクセスするとき，そのメールを「cur」ディレクトリへ移動します。この時，MUAはファイル名の末尾に「:」に続けてメールの状態を表わす文字列を追加します。前述した，ファイル名のユニークな部分はそのまま引き継ぎます。

例えば，「new」ディレクトリ下の「1010300921.28546.asao.gcd.org」というファイルは，「cur」ディレクトリに移されて「1010300921.28546.asao.gcd.org:2,」というファイル名になります。追加した文字列が「:2,」の場合はまだユーザーが見ていないメールを表わし，「:2,S」の場合は，ユーザーが見たメールを意味し

＊6　正確に言えば，MTAが直接メール・ボックスへ書き込みを行うのではなく，前述したようにメール・ボックスへ書き込むプログラムを呼び出すのですが，以下，簡単のためMTAが書き込むと記述します。
＊7　POPサーバーがメール・ボックスにアクセスできれば十分である場合は，POPサーバーが書き込めるようにしておくだけで済みます。
＊8　「Thu, 01 Jan 1970 00:00:00」をEpoch(紀元)と呼びます。Linuxをはじめとする数多くのOSで，Epochからの通算秒数で時刻を管理しています。Epoch はグリニッジ標準時GMTで00:00なので，日本標準時JSTでは09:00になります。

表1　フラグが意味するメールの状態

フラグ	意味	メールの状態
R	Replied	そのメールに対して返事を送信した
S	Seen	そのメールを見た (完全に読んだかどうかは不明)
T	Trashed	そのメールをゴミ箱に捨てた
D	Draft	下書き中という印をユーザーが付けたメール
F	Flagged	ユーザーが印を付けたメール (意味はユーザー次第)

```
./Maildir/
|/home/sengoku/bin/cmail-filter 090xxxxxxxx@cmail.ido.ne.jp
```

図5　~/.qmail の設定

```
+OK POP3 server ready
USER sengoku
+OK Password required for sengoku
PASS secret_password
+OK
LIST
+OK
1 259
2 462
3 459
.
RETR 1
+OK
Received: from localhost (HELO asao.gcd.org) (sengoku@127.0.0.1)
  by localhost with SMTP; 6 Jan 2002 16:08:41 +0900
Date: Sun, 06 Jan 2002 16:08:41 +0900
Message-ID: <vpwuywj6rq.wl@asao.gcd.org>
From: sengoku@gcd.org
To: sengoku@gcd.org

From のテスト

.
QUIT
+OK
```

図6　MUAとPOPサーバーとの通信

ます。「:2,」から始まる場合，続く文字の1文字1文字がフラグになっています。

それぞれのフラグの意味を**表1**に示します。ちなみに「:1,」の場合は実験用とされています。数字の部分が変わると「,」以下に続く文字列の解釈の方法も変わりますが，「:3」以降の数字については，今のところ定義されていません。

新着メール

「tmp」はMTAからの配送を確実にするためのディレクトリです。mbox形式では，MTAがメール・ボックスに新着メールを書き込みつつある瞬間にマシンがクラッシュすると，メールが途中で切れてしまった状態でメール・ボックスに書き込まれてしまうかも知れません。マシンが復旧した後，MTAがメールをメール・ボックスに再度書き込んだとしても，途中で切れてしまったメールはそのままメール・ボックスの中にゴミとして残ってしまいます*9。

そこでmaildir形式では，MTAが新着メールを保存するときは「tmp」ディレクトリに対してファイルを書き込みます。正常に書き込みが終了したときのみ，「tmp」ディレクトリに作成したファイルを「new」ディレクトリへ移動します。

＊9　ゴミが残る程度ならまだ良いのですが，後続のメールがくっついたり文字化けしたりすることも有り得ます。

もし「tmp」ディレクトリに書き込み中にマシンがクラッシュした場合，「tmp」ディレクトリにゴミが残りますが，36時間経過したものについてはMUA側で削除して構いません。

排他制御

maildir形式では，メールが常にユニークなファイル名で保存されるため，mbox形式で問題になったような排他制御の問題がありません＊10。MTAはいつでも新着メールを「tmp」ディレクトリ経由で「new」ディレクトリへ書き込めますし，MUAはいつでも「new」ディレクトリにあるメールを参照したり，状態を変更して「cur」ディレクトリへ移動したり，削除したりできます。

プログラムの起動

メールが届いたとき，特定のプログラムを起動できるとメールの応用範囲が広がります。例えば私は，届いたメールを携帯電話で読みやすいように加工＊11して携帯電話へ転送するプログラムcmail-filterを作って，**図5**に示すように~/.qmailに設定していました＊12。

前述したqmail-localプログラムは，~/.qmailが存在しないときは第8引数で指定したデフォルトのメール・ボックスへメールを保存するのですが，~/.qmailが存在するとメール・ボックスの形式や場所を変更することができます。図5の1行目のように，末尾に「/」を付けるとmaildir形式のメール・ボックスの指定になります。また図5の2行目のように行頭に「|」を付けると，メール・ボックスへ保存する代わりに，指定されたプログラムを起動します。そしてそのプログラムの標準入力にメールを与えます。

MUA

そして，最後はメール配送の最終段階，MUAです。

直接アクセス

MUAがメール・ボックスにアクセスする最も単純な方法が，メール・ボックスのあるファイル・システムを直接アクセスする方法です。メール・ボックスの存在するマシン上でMUAを実行する必要がありますが，NFSなどの方法でリモートのファイル・システムをマウントすれば，他のマシン上にあるメール・ボックスをアクセスすることもできます。

この方法は，MUAが直接メール・ボックスを操作できるため単純だし確実です＊13（ただし，メール・ボックスがmbox形式である場合は，前述したように排他制御の問題に注意する必要があります）。メール・ボックスのあるマシンにログインすることができて，かつそのマシン上で使い勝手の良いMUAが利用可能であるならば，この種のMUAを使うべきでしょう。

Post Office Protocol

1990年ごろから，MacintoshやWindowsなどのPC用OSにもTCP/IPが実装されるようになり，それまでUNIXマシンの独擅場だったLANにPCが接続されるようになってきました。始めはPCからtelnetなどを使ってUNIXマシンへリモート・ログインしてメールの読み書きをしていたユーザーたちは，すぐに使い慣れたPC上でメールを読み書きしたいと思うようになります。

リモートのメール・ボックスへアクセスするために考案されたのがポスト・オフィス（郵便局）・プロトコル（Post Office Protocol，POP）です。POPは1984年にRFC918として提案され，バージョンアップを経て現在はPOPバージョン3がRFC1939で規定されています。また，

＊10　ただし，「tmp」「new」「cur」が同じファイル・システム上のディレクトリである必要があります。
＊11　引用部分や空行を削除し，ショート・メールの文字数制限に合わせてメールを分割して転送します。
＊12　このあて先メールアドレスからも分かるように，KDDIのCメール（いわゆるショート・メール）を利用していたのですが，インターネットからCメールへのゲートウエイ・サービスは2001年12月20日に廃止になりました。KDDIの携帯電話あての迷惑メールのほとんどが，このサービスを利用していたため，とKDDIは説明していますが，迷惑メール対策であれば，他のキャリアが実施しているように，ドメインごとにメールの受信拒否/許可をユーザー側で指定できるようにすべきだったのではないでしょうか。
＊13　私が多くを学んだあるシステム管理者は，mbox形式のメール・ボックスをlessコマンドで読んでいました。確かにこれ以上単純で確実な方法は無いかも知れません。

```
+OK <22663.1010986863@asao.gcd.org>
APOP sengoku c147fa2c29a69d8b730b3dcb52eccfda
+OK
LIST
+OK
1 259
2 462
3 459
.
```

図7　APOPコマンドによる認証

RFC2449で拡張方法が定義されています。

POPサーバーをメール・ボックスがあるマシンで走らせ，MUAがPOPサーバーへポスト・オフィス・プロトコルでアクセスする形になります。通信の例を**図6**に示します。網掛けしている行がMUAからPOPサーバーへの送信，その他がPOPサーバーからの応答です。

POPの特徴は,その単純さにあります。MUAからの送信は，コマンドだけか，コマンドと引数1つだけ＊14というシンプルな形なので，サーバーの実装が簡略化できます。

また，サーバーからの応答は，成功の場合は「+OK 」，失敗の場合は「-ERR 」が必ず先頭に付き，その後に続くデータも，必要最小限のデータだけなのでMUAの実装も簡単です。例えば図中「LIST」コマンドに対する応答は，メールの番号とそのサイズを並べただけですし，「RETR メール番号」コマンドに対する応答は，引数で指定したメール全体になります。

後述するIMAPは多機能である半面，サーバーの実装が複雑になりがちで，セキュリティ上のリスクが高いと言えます。実際，IMAPサーバーには現在までに数多くのセキュリティ・ホールが報告されています。インターネットから直接アクセスできるサーバーを設置する際は，IMAPサーバーではなく POPサーバーを選ぶべきでしょう。

ただし，図6に示したPOPだとパスワード（図中では「secret_password」にしています）が平文でインターネットを流れることになり好ましくありません。認証は毎回同じパスワードを送るのではなく，サーバーが送った「チャレンジ」に対して，クライアントが「チャレンジ」とパスワードを元に算出した「レスポンス」を返す方式にすべきでしょう＊15。サーバーが毎回異なる「チャレンジ」を送ることにより，クライアントが送る「レスポンス」が毎回異なるようにできます。この方式の認証を行うためのコマンドがAPOP です。

APOPコマンドをサポートしているPOPサーバーでは，最初の応答で，「+OK 」の後に「<」と「>」で括ったチャレンジを送信します。図6に示した例では「POP3 server ready」とだけ送信しているのでAPOPコマンドに対応していません。APOPコマンドに対応しているPOPサーバーにおける通信例を**図7**に示します。

図7の1行目がPOPサーバーからの最初の応答で，チャレンジ「<22663.1010986863@asao.gcd.org>」を含んでいます。「22663」はPOPサーバーのプロセスのIDであり，「1010986863」は「Thu, 01 Jan 1970 09:00:00」からの通算秒数ですから，このチャレンジは毎回異なる文字列であることが保証されます＊16。

MUA は，このチャレンジにパスワードをつなげた文字列「<22663.1010986863@asao.gcd.org>secret_password」を，メッセージ・ダイジェスト5（MD5,

＊14　例外的にAPOP，TOPの二つのコマンドは引数を2つ取りますが，この場合も第1引数には空白文字が含まれないので，コマンド解析が容易です。
＊15　チャレンジとレスポンスによる認証方式は広く利用されています。例えば，2001年8月号の本連載「使い捨てパスワード」を参照してください。

詳しくはRFC1321を参照してください）と呼ばれるハッシュ関数に入力して，ハッシュ値「c147fa2c29a69d8b730b3dcb52eccfda」を算出します。そしてこのハッシュ値をレスポンスとしてAPOPコマンドの第2引数として送信します。なお第1引数はユーザーIDです。認証に成功すれば，図7のように「+OK 」が返ってくるので，後は図6に示した POPと同様です。

Internet Message Access Protocol

POPは基本的にはメールをメール・ボックスから取ってくるだけのプロトコルなので，複数のMUAを使いたい場合は不便と感じるかも知れません。例えば，職場のPCと自宅のPCの両方から職場のメール・ボックスを参照したい場合，どちらか一方でメールを取得すると他方では読めなくなってしまいます。もちろん，メールをメール・ボックスから消さなければ，どちらでも読めますが，1つのメール・ボックスに何百通，何千通ものメールを保存しておくことは現実的ではありません＊17。

そこで，1つのメール・ボックスだけではなく複数のメール・フォルダをサーバー側に作って，メールを整理できるようにしたプロトコルがInternet Message Access Protocol（IMAP）です。現在はRFC 2060で規定されたバージョン4rev1（以下，IMAP4と略記）が標準です。

メール・フォルダをたくさん作ってメールを整理するという最近のMUAならば必ず備えている機能が，IMAP4を使うとサーバー側で実現できます。複数のMUAから同一のメール・フォルダ群を操作したい場合には便利でしょう＊18。

また，IMAP4にはメール・ヘッダーの一部のみを一覧する機能もあります。出先から携帯電話等を使ってサーバーへアクセスするときなど，通信路の帯域が十分に確保できないときは，サイズの大きなメール＊19を受け取ろうとすると長い時間がかかってしまうので，あらかじめ指定したサイズ以上のメールは受け取らないように設定できるMUAが一般的だと思います。実際，図6に示したようにPOPでもLISTコマンドによって各メールのサイズが送られてくるので，大きなメールは受け取らないようにすることができます。

しかし，重要なメールならば多少サイズが大きくても受け取りたいでしょうし，不要不急のメールはサイズが小さくても受け取る時間が惜しい場合もあるでしょう。IMAP4では，メールのヘッダーだけ，あるいはヘッダー中の特定のフィールドのみを参照することができますから，「From: 」や「Subject: 」を見てそのメールが重要か否かを判断することが可能になります。

IMAP4を使って，新着メールそれぞれのサイズおよび「Date: 」「From: 」「Subject: 」フィールドを参照した例を**図8**に示します。MUAからIMAP4サーバーへの送信（網掛けしている部分）は，最初にタグを付けます。このタグは適当な文字列で構わないのですが，それぞれのコマンドごとに変えていくべきなので図中「A0001」のような1つづつ増えていく一連の数字を付けたタグの方が望ましいでしょう。

サーバーからの応答は，「＊」で始まるか，送信したコマンドに付けたタグと同じタグで始まります。タグで始まる行は，サーバーの応答の終わりを意味しているので，MUAは次のコマンドを送信することができます。

「SELECT INBOX」コマンドで対象となるメール・ボックスを選択し，「UID SEARCH RECENT」コマンドで新着メ

＊16　もし，1つのプロセスで複数のMUAからの接続を受け付けるPOPサーバーを実装する場合は，プロセスIDではなく別のユニークな数字を付けるようにすべきでしょう。
＊17　最近はディスクの値段が非常に安くなったので，何千通ものメールをため込んでいても容量的には問題がないのかも知れませんが，単一のメール・ボックスに何千通ものメールがあると必要なメールを検索するのに大変骨が折れます。
＊18　そこまでするなら，サーバーへログインしてサーバー上のMUAを使う方がシンプルで良いのではないか，と私は思いますけれど。サーバーへログインしてしまえば，どこからでも全く同じ環境が使えます。
＊19　最近は巨大なファイルを添付した10Mバイト単位のメールも来ることがあるので困ったものです。

```
* OK Courier-IMAP ready. Copyright 1998-2000 Double Precision, Inc.  See COPYING for distribution information.
A0001 CAPABILITY
* CAPABILITY IMAP4rev1 NAMESPACE THREAD-ORDEREDSUBJECT SORT
A0001 OK CAPABILITY completed
A0002 LOGIN sengoku secret_password
A0002 OK LOGIN Ok.
A0003 SELECT INBOX
* FLAGS (\Answered \Flagged \Deleted \Seen \Recent)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen)] Limited
* 3 EXISTS
* 3 RECENT
* OK [UIDVALIDITY 1010985213]
A0003 OK [READ-WRITE] Ok
A0004 UID SEARCH RECENT
* SEARCH 61 62 63
A0004 OK SEARCH done.
A0005 UID FETCH 1:100 (RFC822.SIZE BODY[HEADER.FIELDS (DATE FROM SUBJECT)])
* 1 FETCH (UID 61 RFC822.SIZE 470 BODY[HEADER.FIELDS ("DATE" "FROM" "SUBJECT")] {67}
Date: Sat, 01 Jan 2000 00:01:02 +0900
From: postmaster@gcd.org

)
* 1 FETCH (FLAGS (\Seen \Recent))
* 2 FETCH (UID 62 RFC822.SIZE 474 BODY[HEADER.FIELDS ("DATE" "FROM" "SUBJECT")] {71}
From: test@gcd.org
Subject: test
Date: 10 Nov 2001 12:13:41 +0400

)
* 2 FETCH (FLAGS (\Seen \Recent))
* 3 FETCH (UID 63 RFC822.SIZE 267 BODY[HEADER.FIELDS ("DATE" "FROM" "SUBJECT")] {64}
Date: Sun, 06 Jan 2002 16:08:41 +0900
From: sengoku@gcd.org

)
* 3 FETCH (FLAGS (\Seen \Recent))
A0005 OK FETCH completed.
```

図8　MUAとIMAP4サーバーとの通信

ールの ID の一覧「61 62 63」を得ます。そして次の「UID FETCH 1:100 …」コマンドが，メールを取得するコマンドです。

「1:100」が取得すべきメールのIDの範囲が1から100であることを意味し，その次の引数が何を取得すべきか指定します。この例の場合でしたら，メールのサイズと「Date: 」「From: 」「Subject: 」の各フィールドということになります。

そしてユーザーが重要だと判断したメールのみを取得することになります。例えばIDが63のメールを取得する場合は，**図9**のように「UID FETCH 63 BODY[]」コマンドを送信します。

```
A0006 UID FETCH 63 BODY[]
* 3 FETCH (UID 63 BODY[] {267}
Received: from localhost (HELO asao.gcd.org) (sengoku@127.0.0.1)
  by localhost with SMTP; 6 Jan 2002 16:08:41 +0900
Date: Sun, 06 Jan 2002 16:08:41 +0900
Message-ID: <vpwuywj6rq.wl@asao.gcd.org>
From: sengoku@gcd.org
To: sengoku@gcd.org

From のテスト
)
A0006 OK FETCH completed.
A0007 LOGOUT
* BYE Courier-IMAP server shutting down
A0007 OK LOGOUT completed
```

図9 メールの取得

さて，IMAP4は使いやすいMUAを作る上で非常に便利なのですが，サーバーが複雑であるためにセキュリティ・ホールが残っている危険が高く，インターネットから直接アクセスできるようにする場合は，それなりの覚悟が必要です。

IMAP4サーバーに限った話ではありませんが，サーバーを直接インターネットからアクセスできる状態にさらす場合は，サーバーの開発元のバージョンアップ情報には常に目を通し，セキュリティ上の問題点が発見されたら速やかにバージョンアップする体制作りが最低限必要です。

そして，安全性を高めるには，サーバーへのアクセスを常に監視し，想定外のアクセス・パターンがないか監視することが重要です。

可能であればサーバーのソース・プログラムにも目を通し，セキュリティ上脅威になりそうな部分がないかチェックしたいところです。

もちろんたくさんのサーバーそれぞれについて，これだけのことをやっていては大変なので，一般論としてはインターネットから直接アクセスできるサーバーを減らすべし，ということになります。例えばsshサーバーだけをアクセスできるようにしておいて，他のサーバーはsshのポート・フォワード機能[*20]を使って利用するようにすれば，監視すべきサーバーはsshだけで，しかもIMAP4を含めたさまざまなサーバーが，セキュリティ上のリスクを負わずに利用可能になります。

IMAP4には，図8で示したLOGIN認証だけでなく，各種認証方式が利用可能ですが，認証方式が立派でも，サーバーの実装にセキュリティ・ホールがあれば元も子もありません。逆にパスワードを平文で流すLOGIN認証でも，sshのポート・フォワード機能と組み合わせて使えば，sshが通信を暗号化するので，セキュリティ上の問題は無いと言えます。

＊20 sshのポート・フォワード機能については，連載「実践で学ぶ，一歩進んだサーバー構築・運用術」第9回〜第11回の「ssh」を参照してください。