Internet Week '97 チュートリアル

DNS & mail

1997年12月17日(水)

中村 素典(京都大学)

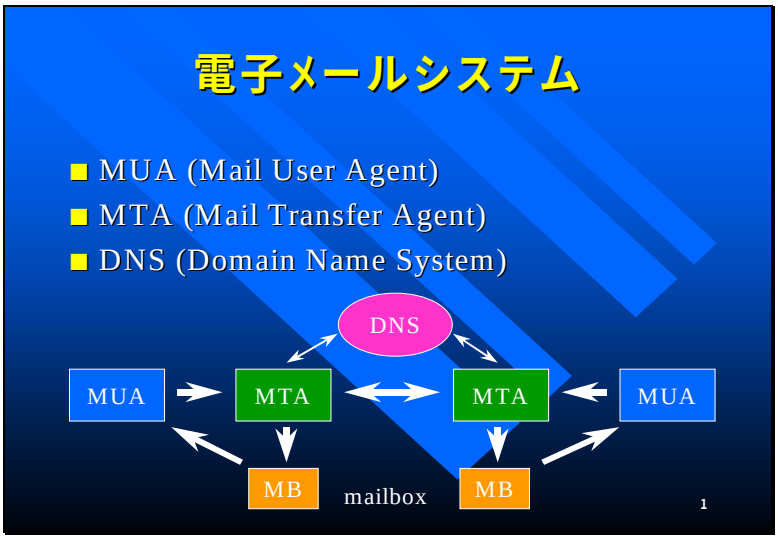
目的

メールシステムの基本的な概念を押さえていただき、どんなメールの送受信の設定もできるようになっていただけることを目的に説明致します。

1. **インターネットメールの基礎知識**
2. **メールの受信設定の基本**
3. **メールの配信設定の基本**
4. **Sendmailの概要**
5. **メールシステムのデザイン (CF)**
6. **Sendmailシステム管理**
7. **DNSの仕組みと管理**

1. インターネットメールの基礎知識

電子メールシステムの機能及び仕組みをまとめると下図のようになります。



両端にそれぞれの人が使うMUA(Mail User Agent)があります。実際にメールをやり取りする部分がMTA(Mail Transfer Agent)です。以下では「MTA」を中心に説明します。

このような機能分けをした場合に、外から見えるものとして、「DNS(Domain Name System)」と呼ばれる、メールの経路制御を行い受信者が希望する送り先を外部的に教えるものと、ユーザに対して届いたメールを保管しておいて後でユーザにメールを渡す「MB(mailbox)」があります。

インターネットで通信の話をするとプロトコルが出てきますが、メールのやり取りでは「SMTP(Simple Mail Transfer Protocol)」が使われます。SMTPに限らず、UUCPの上で送る場合もありまが、インターネットではSMTPの話が中心になります。

mailboxからMUAに取って来るところで、いろいろな方式がありますが、POPなどが使われます。

MUA (Mail User Agent):

ユーザが直接使用し、ユーザに対してサービスを行うユーザ・アプリケーションで、メールを読む、メールを書く、メールを保存/検索する、を行うものです。

例えばMUA Applicationsには以下のようなものがあります。

- UNIX – ucbmail, RMAIL, mush, MH (mh-e), mew,....
- Windows – OutLook, Netscape Mail, Eudora,....
- Macintosh – Eudora,....

MTA (Mail Transfer Agent):

MTAが行うことは、先ず、MUAあるいは別のMTAからメールを受け取ります。受け取ったら宛先を見て、配信先を決めます。送り先にはいくつかの選択肢があり、隣りのホスト(リモートへ)や自分自身(ローカルへ)、あるいは宛先が存在しない(拒否)、があります。

これらのデータの処理、すなわち、受け取ったら、一旦、自分のところに保管して、そのあと次のところに送りますが、この方式を「Store and Forward」と呼びます。
送り先が無かったときは、エラー通知を返送します。

MTAプログラムには以下のものがあります。どのMTAも基本機能は同じですので、基本的な概念をしっかり理解しておくと大抵のプログラムに対応できます。

- Sendmail　http://www.sendmail.org/　最も有名です。
- qmail　http://www.qmail.org/　最近、広く使われ始めています。
- SMAIL (GNU)
- MMDF (Multi-channel Memo Distribution, CSNET)
- exim　http://www.exim.org/
- VMail　http://wzv.win.tue.nl/vmail/
- LSMTP　http://www.lsoft.com/LSMTP.html
- PP (X.400)

DNS (Domain Name System):
メールのやり取りを行うとき、広域分散ディレクトリ・サービスが使われるようになってきました。
昔、日本ではJUNETというUUCPネットワークがあった時代は、メールの宛先を見たらそれがどの方向にあるのかを（少なくとも下流方向については）管理者がいちいち設定していましたが、最近のインターネットの世界では、DNSを正しく設定しておけば送信側ホストでは何も設定すること無く、DNSを参照するだけで、宛先へ届くようになっています。つまり、UUCPの時代には、いろんなホストで設定しなければいけなかったことがDNSに集約されて、DNSさえきちんと設定しておけばメールが届くようになりました。
逆に言えばDNSの設定がちゃんとしていなければメールが届かないので、最近のメールの設定においてはDNSの知識が重要になってきました。

DNSの基本的な役割は、ドメイン名やホスト名をIPアドレスに変換することです。昔は、1つのテーブル(etc/hosts)で全世界の知っているホストを定義(ホスト名とIPアドレスの関連を定義)して、メールが届くようになっていました。
基本機能はホスト名からIPアドレスを調べることですが、その仕組みをメールアドレスにも対応付けして、メールアドレスからホスト名を調べて、さらにIPアドレスを調べ、そこにメールを送り届けます。

本チュートリアルで使う用語を、以下に定義します。

・ 配信 ： メールを渡す動作

メールの宛先が自分のホストの場合をローカルに配信と表現し、自分のメールボックスにメールを追加します。

メールの宛先が別のホストの場合をリモートに配信と表現し、別のMTAへ渡します。

・ 転送 ： リモートに配信の別の表現

・ 受理 ： ローカルに配信の別の表現 (一般的な用語ではないかもしれません)

・ 受信 ： 別のMTAからの配信

配信、転送、受理は一旦、受け取ったものをどうするかという表現ですが、受信はどこ宛かはわかりませんが別のMTAから送ってきたときの動作の表現です。

メールアドレス:

メールアドレスがどのような形になっているか確認しておきます。メールの宛先、あるいは発信者を特定するための情報で、例えばmotonori@wide.ad.jpのように表現し、これは「ユーザ部@ドメイン部」の形式からなっています。この形式はドメイン形式といいます。メールの世界でドメイン部といった場合は、「@」から後ろの部分を指し、これがドメイン名であれホスト名であれドメイン部ということにします。

ドメイン形式以外にも、%-Hack、Route Address、UUCP addressingの形式のメールアドレスがあります。

%-Hack:

計算機を指定して、メールを直接送るのではなく、一旦、別のホストを中継させて送ることを明示的に指定したいときに使用するものですが、昔、直接メールが届かなくて、あるホストを経由すると届く場合に利用されました。関連するRFCは、RFC1123(S)です(SはStandardレベルを表す)。

人がわざわざ入れて使用することは少ないですが、インターネットと他のネットワークとのゲートウェイを経由する時に、そのホストを経由しないと返事が送り返せないといった場合にゲートウェイがこの形式のアドレスに変換してメールを転送することもあります。

user % host @ relay

senderからrelayを中継してユーザがいるhostに送る場合の記述 (sender → relay → host)

user % host % relay2 @ relay1

senderからrelay1、relay2を中継してhostに送る場合の記述 (sender → relay1 → relay2 → host)

Route Address:

%-Hackと同じように使用、関連RFCはRFC822(S)です。

@relay: user @ host

sender → relay → host

@relay1, @relay2: user @ host
sender → relay1 → relay2 → host

UUCP addressing:
目的地が右に記述されます。
host ! user
relay ! host ! user

host ! user @ domain の解釈では、インターネットとUUCPで以下のように異なります。
– host ! user @ domain (Internet的)
» sender → domain → host
– host ! user @ domain (UUCP的)
» sender → host → domain

コメント形式:
メールの設定でフィルタを記述したりする場合にコメント形式が重要になります。

例: Full Name <user@domain>
user@domain (Full Name)
user(User Name)@domain(Company Name)
( )のコメントはどこに入ってもよいので、MTAを作る場合は、このようなコメントが在っても正しく処理できるように注意する必要があります。

ドメイン部:
- Fully Qualified Domain Name (FQDN)
インターネットドメイン形式の完全な(省略のない)ホスト/ドメイン名
- Fully Qualified Mail Address
FQDNをドメイン部として持つメールアドレス(@の右がFQDN)
例 user@mailhost.wide.ad.jp
- Not Qualified Mail Address
ローカルにメールを送る場合のメールアドレス(ドメイン部が無い)
例 user
- Generic Address
ドメイン部にホスト名を含まないメールアドレス。ホストが変わってもuserにメールが届く表現。
例 user@wide.ad.jp

ヘッダ(header)と本文(body):

やり取りされるデータは、メールあるいはメッセージと呼ばれ、RFC822(S)(Standard for the format of arpa internet text messages)に形式が規定されていますが、その構造を整理します。

配送の制御に使う情報が上に並んでいて、途中に空行があり、その下からメールの本文が始まる形式になっています。空行の下には制御の情報は含まれません。

上の部分をヘッダと呼び、下の部分をボディまたは本文と呼びます。

```
From: announce@nic.ad.jp
To: motonori@wide.ad.jp
Subject: Internet Week 97
    ← 空行 (空白もなし)
Internet Week 97 のお知らせ
```

発信者と受信者:

From フィールドの定義によると、発信者(Sender)を複数書いてもよいことになっていますが、実質的な発信者を考えると通常は1人です。

受信者(Recipient)は、 1人または複数人です。

ヘッダとエンベロープ:

エンベロープ(envelope)という概念がメールシステムを理解するときの1つのメイントピックになります。エンベロープは、ヘッダ及びボディとは別にあり、配信処理と密接に関連があります。

SMTP(Simple mail transfer protocol)はRFC821(S)に規定されています。それ以外にUUCPの場合もありますが、共にメイルを配信するときにヘッダとは別の情報を扱います。

一般の郵便を考えた場合の封書に似ています。ヘッダやボディは、エンベロープの中に入る手紙であり、エンベロープは手紙を送り届けるために使われます。エンベロープは、最終的にメールを読む人にとっては重要ではないという位置付けになります。

例えば、社長Aと社長Bが手紙をやり取りするとき、社長と秘書を考えた場合、社長Aが本文を書き、社長Bに送るように秘書に指示すると、秘書はその手紙を封筒の中に入れて封筒の宛先Bを記入し、投函します。宛名が間違っていて、届かなかった場合は、その封筒は会社Aに戻ってきますが、住所が間違っただけの場合は秘書だけの判断で住所を訂正して送り直されます。これと同様に、送り直しなどの実務的な作業を行う人が、封筒の情報を書いて、封筒の発信人となります。

「ヘッダ」は、手紙の中に書かれている宛先及び発信者など、手紙を実際に書いた人や読んで欲しい人がそこに入ります。

これに対して、「エンベロープ」は、その封書を送りたい宛先で、直接社長ではなく、まず秘書に届いて開封されるように、実際作業する人に届くという違いがあります。

このように違いがありますが、一般的にはヘッダとエンベロープの送信者/受信者は一緒で、MUAに対してユーザがメールの宛先を指定する場合は、普通ヘッダに対して宛先を指定し、そこからエンベロープの情報を抽出して実際に送り届ける形になりますので、通常は封書の中に書かれている宛先が封書の外の住所としてコピーされます。個人宛ての場合がそうです。

異なる場合としてはメーリングリストなどがありますが、この場合はヘッダとエンベロープが異なります。メールがエラーになった場合の処理や、メールがメーリングリストで無限ループになった場合に断ち切る設定を行うためなど、管理の都合上、故意に変える場合があります。

SMTPの場合のヘッダとエンベロープの実例

つながっているSMTPのセッションで、最初に、このメールの封筒の発信人の情報にあたるものが送られます。それに対してアドレスが解釈できた場合は、sender okという返事が返ってきて、次に宛先のメールアドレスを相手に渡すと、recipient okが返ってきます。

次にDATAコマンドを送り、その下の行からがメールとして送られる内容となります。その中には、ヘッダ、空行、本文があります。最後に".”(ピリオド)でメールの全体が終わります。

これがSMTPのセッションの1回分で、このように、最初にエンベロープ、次にヘッダ、そして本文が渡され、メールのやり取りが行われます。

この例は、Internet Week 97のお知らせが、メーリングリストで送られてきたものです。

エンベロープは何時作るか:

ユーザがメールを出すときは、ヘッダとエンベロープの情報を区別して書いていませんので、どこかでエンベロープの情報を作る必要があります。

MUAが直接SMTPを実行する場合は、MUA自身がヘッダから情報を取り出しますし、unixのMUAであればsendmailが引き出します。

配信のときにはヘッダでなくエンベロープが使われますから、大学から会社に就職したり、社内で部署を変わったりしたときに、メールの転送の設定をしたときは、ヘッダはそのままでエンベロープの情報だけが書き換わることになります。

返信に利用するアドレス:

基本的に、配信にはエンベロープを使用しますが、送られてきたものを送り返す、あるいは返事をするなどのリアクションをする場合に、2つの方法があります。1つは自動的に行われる場合で、もう1つは人が介在する場合です。

自動的に送り返される場合は、手紙の中を見ること無しに、エンベロープの発信者に送り返されます。これは、主に宛先に間違いがあった場合に行われ、配信上あるいはMTA上のエラーにおいて返信が必要になった場合ですが、使われるアドレスはエンベロープのもの

となります。

人が介在する場合は、一旦、配信処理は完了していますので、エンベロープ情報は不要になります（実際に、宛先に届いた後ですから、少なくともエンベロープの受信者情報を残しておくことは無意味です。エンベロープの発信者情報はヘッダのReturn-Path:に残されることがあります）。ユーザが見た場合、残された情報はヘッダの情報だけですので、返信はFrom:, Reply-To:で書かれたヘッダのアドレスに対して行うことになります。

従って、メーリングリストの設定をするときに、よく考えないといけないのは、人が返事をする場合にどこにメールが飛んで来て欲しいか、エラーでメールが返って来る場合にどこに飛んで来て欲しいかを区別しておくことです。

この点が十分考えられていないMTAでは、自動返信のときにヘッダの内容を見て返信してしまい、メーリングリストでエラーメールのループが発生することあります。また、例えば、ユーザがvacationという自動応答のプログラムを書くとき、この点が十分考えられていないと、不在ですのでごめんなさい、といったメールがメーリングリストでばらまかれることになります。

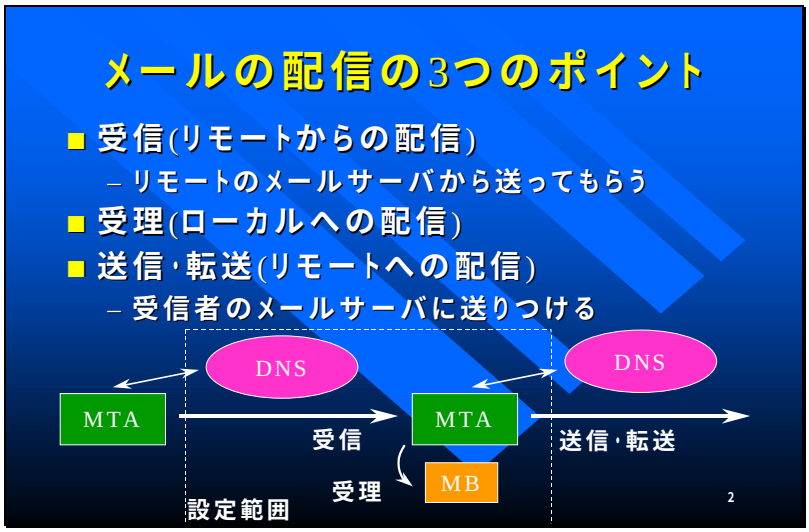
リモートへの配信手段：

- SMTP (Simple Mail Transfer Protocol)
  - TCP ポート25
- UUCP (Unix to Unix CoPy)

メールボックスから MUA へ渡す手段：以下のものがあります。

- ローカル・メールボックス
  - UNIX など
- POP
- IMAP

メールの配信の3つのポイント:



受信、受理、送信の3つを区別して理解してください。

受理のための設定しようとするのが右のMTAだとします。このMTAにメールを送って欲しいわけです。DNSは2つに分かれて描いてありますが実際は1つのシステムです。左のMTAが参照するDNSの情報を正しく設定することが、受信のために先ず必要になります。この設定ができていれば、送ってきたメールが自分宛てだと解釈する、あるいはFirewallがあって組織の中に転送する場合に正しく受理する設定ができているかどうか、あるいは隣りの計算機に転送する設定ができているかどうか、などの3つの項目をチェックすると、sendmailに限らず、どんなMTAでも確実にメールが送り届けられます。

2. メールの受信設定の基本

メールの受信設定とは、メールを送ってもらうための設定です。自分の設定ではなく、メールを送ってくれる他のホストに対して、私はここにいるので、このアドレスに対してメールを送ってください、ということを知らせることです。

どうやって送り先を教えるか：

インターネットの場合は、直接メールを送ってもらうので、DNSに対して配信先を定義します。すなわち、自分のホスト名を定義します。

バケツリレー・システムのとき(UUCPなどJUNET時代)は、このような直接配信ができませんでしたので、周りのホストに私はここにいますのでよろしく、と伝えて経路を設定してもらっていました。そのため、mailconfというバケツリレー・システムのための、経路上の(すべての)ホストに配信先を設定するツールが活躍しました。

ここでは、DNSの設定は既に行われていて、経験の少ない管理者が引き継ぐ場合のチェックポイントを説明します。

DNSレコードの基本型：1行毎に次のDNSのレコードができています。

```
name     [ttl]    IN    type    value...
<左辺>                          <右辺>
```

nameはインターネットで識別される名前で、これに対して各レコードが定義されます。
typeはレコードの種類でA、MX、CNAMEなどがあります。
各typeに対して、value(そのレコードの値)が定義されます。

メール配信時に参照されるtype：

typeは何種類かありますが、メールの配信に関しては、基本的なものは次の3つです。ネームサーバの管理に関してはさらに出てきますが、既に動いているシステムに対して設定を変更する場合はこの3つを覚えて下さい。

- A (Address) RR (Resource Record)
  インターネットのホストのIPアドレスを定義するレコードです。
- MX (Mail eXchanger) RR
  ある名前に対して、それがメールアドレスの場合に、そのメールの配信先(誰が受け取るか)を定義するレコードです。
- CNAME (Canonical NAME) RR
  ホストのエイリアス、すなわちホストの別名を定義するレコードです。

ホストに対するIPアドレスを定義(A RR):

形式	ホスト名	IN	A	IPアドレス
例	mail.x.co.jp.	IN	A	12.34.56.78

これはmail.x.co.jp. という名前のホストに対して 12.34.56.78 というIPアドレスを登録する場合のレコードの記述例です。

DNSができる前からメールの配信は行われていましたが、その時はホスト同士の通信でした。ホスト名同士の通信では、ホスト名に対してそのIPアドレスがわかって、そのIPアドレスに対してメールを投げることがメールの配信の基本でしたので、それがそのまま現在でも継続され、利用できるようになっています。

Aレコードとして、この例のように定義してあれば、user@mail.x.co.jp 宛てのメールの配信時に参照され、12.34.56.78のIPアドレスのホストにメールが送られます。

DNS登録状況の確認ツール:

自分で登録したものが正しく登録できているかを確認するためにいろいろなツールがありますが、bindに付属するツールは以下の通りです。DNSに関する知識をしっかり持っていればnslookupだけでもチェックすることができます。

- nslookup
- dig
- host
- dnsquery

nslookupでAレコードを確認する場合の例1(sh.wide.ad.jp.のAレコードを確認する場合)

% nslookup sh.wide.ad.jp. (注意:ドメイン名の補完を抑制するため最後のピリオドを補うこと)

```
Server:  localhost
Address:  127.0.0.1
```

この問い合わせに対し、結果が以下のように返ってきます。

```
Name:    sh.wide.ad.jp
Address:  203.178.137.73
```

複数のIPアドレスを持つホスト:

Aレコードの話でもう少し進んだトピックを紹介します。mail.x.co.jpというホストが複数のイーサーネットの足を持っていた場合は、以下のように2行目以降の後続する行は左辺を省略して右辺だけを記述し定義できます。

```
mail.x.co.jp      IN  A  12.34.56.78
                  IN  A  12.34.54.32
```

このような複数のアドレスを定義した場合、メールが来てアドレスが渡ったときに複数のアドレスがもらえますので、それを順番にトライしていずれかで配信が成功した場合にはメールが届き、もし全てに対して失敗したときはメールは届きません。例えば、1つ目に失敗し、2つ目に成功した場合は、2つ目のアドレスでメールが送られます。

実装依存で、この辺があまり考えられていないソフトの場合では、2つのアドレスが得られて

も1つ目のアドレスにしかトライしない場合があります。

最近のDNSではラウンドロビン機能がありますが、これは複数のアドレスを定義しておいた場合、問い合わせをしたときの返って来るアドレスの順番が毎回変わる機能です。例では、最初に問い合わせた場合、12.34.56.78、12.34.54.32の順でアドレスが返り、次に問い合わせた場合には12.34.54.32、12.34.56.78の順でアドレスが返るというものです。

これは、負荷分散のためにも使われますし、もし1つ目しかトライしないようなソフトが利用されていてもそのうち届く(?)ようになります。

nslookupでAレコードを確認する場合の例2（複数のIPアドレスを持つホストの場合）

```
% nslookup jp-gate.wide.ad.jp.
Server:  localhost
Address:  127.0.0.1
```

この問い合わせに対し、複数のアドレスが以下のように返ってきます。

```
Name:     jp-gate.wide.ad.jp.
Addresses:  203.178.137.17, 203.178.136.81, 203.178.137.75, 203.178.136.89
```

Aレコードの説明は以上で、次にMXレコードについて説明します。

Genericなメールアドレス:

MXレコードが必要になる1つの場合は、Genericなメールアドレスを定義する場合です。

Genericなメールアドレスとは、ドメイン部にホスト名部分が含まれないアドレスで、ホストに依存した配信になりませんので、ホストが故障してメールが届かなくなった場合、すぐ別の計算機に入れ替えて運用を続けることが簡単にできます。

MX (Mail eXchanger) RRの形式及び例は以下の通りです。MXはMail eXchangerの略でメール交換を行う計算機で、MX RRはこの計算機の定義を行います。

形式	メールアドレス	IN	MX	優先度	ホスト名
例	x.co.jp.	IN	MX	10	mail.x.co.jp.

メールアドレスと実際にメールを送って欲しいホストを定義します。この例では、user@x.co.jp 宛てのメールは指定されたホストmail.x.co.jp.に送られます。

ホスト名のIPアドレスではなく、計算機の名前を書くところがポイントです。この計算機の名前から、さらにAレコードに基づいてIPアドレスを拾ってそこにメールが送り届けらる、2段構えの処理となります。

nslookupでMXを確認する場合の例

nslookupはデフォルトではAレコードの問い合わせとなりますので、Aレコード以外を検索したい場合はオプションの指定が必要です。MXレコードの場合は-q=mxと明示的に指定して、それに続けて引きたいドメイン名を記述します。

```
% nslookup -q=mx wide.ad.jp.
Server:  localhost
Address:  127.0.0.1
```

この問い合わせに対し、結果が以下のように返ってきます。Aレコード的にホストのIPアドレスがadditional informationとしてわかります。この例では、wide.ad.jpのMXホスト名がsh.wide.ad.jpで、そのIPアドレスが203.178.137.73と定義されていることがわかります。

```
wide.ad.jp      preference = 10, mail exchanger = sh.wide.ad.jp
   :
sh.wide.ad.jp    internet address = 203.178.137.73          (additional information)
```

MXレコードを引くと、最初にホスト名が返ってきますが、そのホスト名に対して必ずIPアドレスを検索する処理が必要です。ここで再度検索に行くのは無駄なので、MXに関する検索を行った場合は、その名前に関連するIPアドレスは一緒に送り返されてきます。

中には、自分のドメインではなく、全く異なる組織に対してバックアップのメールサーバのサービスを行っている場合もありますが、その場合は一緒に返す情報をその時点で知らない場合もありますので、IPアドレスが一緒に返ってこないこともあります。

災害に備える(MX編):

メールのバックアップについて話します。自分のところのメールサーバが1つで、それがダウンしていた場合、メールが受け取れない状況が発生します。メールの発信ホストはメールが送れなくても、それぞれキューに保存して、30分とか1時間後にトライしますが、そうすると世界中で送れないメールがばらばらに溜まります。そうなると、メールサーバが回復してもすぐにはメールの再送処理をさせることができません。もし、自分がコントロールできる範囲内にバックアップのメールサーバを用意しておければ、自分のメインのプライマリ・メールサーバがダウンしていてメールが送れない場合に、そこにメールを溜めておくことができます。プライマリ・メールサーバが回復後には、バックアップのメールサーバからプライマリ・メールサーバに直ち送信させることができます。

そのようなことをしたい場合に、MXを使って、バックアップの仕組みを作ることができます。

メール受信の代行時のMX設定例：

メインのメールサーバをmail1、組織内のバックアップのメールサーバをmail2として設定します。場合によっては、自分の組織がつながっているプロバイダでバックアップサービスをしてくれる場合があり、そこを一緒に設定することもあります。

```
x.co.jp.        IN MX     10 mail1.x.co.jp.
                IN MX     50 mail2.x.co.jp.
                IN MX    100 mail.provider.ad.jp.
```

ここで10、50、100という数字が出てきますが、この数字をプレファレンス(コスト値)と呼び、数値が小さい方が優先度が高いことを意味し、送り側は成功するまで順にコストの大きなものへ配信を試みていきます。

この例では先ず、10のmail1にメールを送ろうとし、できなかった場合に50のmail2に、それができなかった場合は100のmail.providerにメールを送ろうとします。

このようにして、メインのメールサーバにトラブルがあった場合にバックアップのメールサーバにメールが溜まり、メインのメールサーバが回復したときにまとめてプライマリに向けて配信させることができます。

しばしば問題視されることですが、正月やお盆の休みにメールサーバを止めるような運用をしているところがあります。そのような場合に1週間メールサーバが止まったり、そうでなくてもトラブルで1週間メールサーバが使えなかったりした場合に、保存期限を越えて、本来受け取ってもらえているはずのメールが発信者に送り返されることがあります。バックアップサーバのメールの保存期間を十分長く取っておくことで、無意味にメールが送り返されないようにすることができ、これも管理上の注意点の1つです。

MXのプレファレンス (MX RRに指定するコスト値)：

最もプレファレンスの小さいメインのメールサーバのことを、以下のようにPrimaryやFirstという形容詞で表現します。

- Primary MX / Primary Mail Server
- First MX / First Mail Server

2番目にコストが小さいメールサーバを以下のように呼びます。

- Secondary MX / Secondary Mail Server

2番目より下のものを以下のように呼びます。

- Lower MX (優先度が低いという意味)

Lower MXの条件：

自分のところのメインのメールサーバを設定するときは、メールが正常に受信できて設定完了となりますが、バックアップのメールサーバを設定するときにはさらに以下の注意点があります。

自分のところのメールサーバがちゃんと動いているので、他の組織や他の部署のバックアップに使っても構わないとLowerMXに指定してもらうことがあります。プライマリ・メールサーバが安定して動作している場合はバックアップのメールサーバにメールが届くことはありませんが、

いざプライマリ・メールサーバにトラブルが発生したとき、突然バックアップのメールサーバにメールが届きはじめます。そのときバックアップとしての設定がしっかり行われていないとメールが落ちたり、返信されてしまい、バックアップとして機能しないどころか、問題を発生させる原因になってしまいます。もし年末年始やお盆のときにこのようなトラブルが発生すれば、連絡しようにも連絡がつかない事態となりますので注意して下さい。

MXが複数設定されていたときに、先ほどの例(mail1、mail2、mail.provider)で2番目のメールサーバmail2にメールが送られたとします。mail2はMXのリストに従ってプライマリのmail1へ受信したメールを転送しようとしますが、このときまだmail1がダウンしていた場合、定義の2番目が自分だと認識できていない場合は誤って自分自身につなぎに行ってしまいます。2番目以降について見れば自分が最上位のメールサーバとなるので、それ以下のメールサーバにメールを送る必要はありません（送るとメールの転送ループが発生するので送ってはいけません）。MTAは、MXリストに自分自身を見つけた場合は、それ以下のメールサーバにはメールを送らないようになっています。

Sendmailの場合、自分自身に接続してしまった場合は配信エラー(local configuration error)にしますから、そこでメールが発信者に返送されてしまいます。また、自分につながらずにthirdにメールが送られたとすると、今度はthirdは同様にMXのリストを見て、secondにメールを送ります。従って、secondとthirdの間でメールの転送が繰り返され、too many hopsエラーで発信者にメールが返送されます。このように、バックアップの設定が不完全だと、firstがダウンしているときにメールが届かないどころかエラーになって発信者に返ってしまうという問題が出てきます。

正常な動作のためには、secondaryがMXのリストの中に自分の名前があってそれがsecondaryであることを知ることが重要になります。知っていれば、自分より上のfirstに対してだけ転送を試みます。同様にthirdの場合は、secondより上に転送を試みます。

ポイントは、メール・ループを防ぐために、名前をちゃんとチェックできるように設定することです。

MX RR の右辺ある自分の名前を認識し、自分自身への接続の防止するには、sendmailの場合は、クラスwに定義されているホスト名が、先程のMXのリストにある名前と一致しているかどうかチェックすることで行われます。テストモードで、$=wというコマンドを入力すると、ホスト名のリストが出てきますので、MXの右辺に指定されている名前があるかどうかを確認することが重要なポイントになります。

それがきちんとできていれば、自分の名前に対応するMX RRのプレファレンスよりもコストの高いRR(Resource Record)を全て捨てる処理をしてくれますので、この名前を確認することがポイントになります。

負荷分散：

先ほど、Aレコードに関する負荷分散を説明しましたが、MXでも同じプレファレンスを指定しておくと同じように負荷分散ができます。但し、firstと同じプレファレンスを付けておく場合には両方に受理の処理をするような設定が必要になります。

```
x.co.jp.        IN MX   10 mail1.x.co.jp.
                IN MX   10 mail2.x.co.jp.
```

このように同じコストの場合は送り側が毎回乱数で試行順序を決定します。

またA RRのラウンドロビン機構を用いて複数のホストによる負荷分散を行うこともできます。この場合、一つのA RRに複数のホストのIPアドレスを記述することになりますが、その場合、spool full などプロトコル的に拒否された場合の動作がMX RRによる負荷分散の場合と異なります。どちらの場合であっても、最終的に一つのメールボックスへ集めなければなりませんから、受信側にそれなりの仕掛けが必要になります。

ホストにもMXレコードを：

災害に備えるためにバックアップを設定する場合は、負荷分散とは異なり順位付けが必要になります。Aレコードによる方法では順位付けできませんから、Secondary MXの指定ができません。順位付けをするような設定をホストに対して行いたいが、genericなアドレスばかりではなく、ホスト名が明示的に表れたアドレスもありますから、そういった場合はAレコードでメールを送ってもらうのではなく、MXレコードでメールを送ってもらうようにします。これが、MXレコードをホストに付ける必要性です。

さらにいうと、インターネットのトラフィックを押さえ、検索の効率化を行う、あるいはメールの配信にかかる時間を減らすためにも、ホストに対してMXレコードを設定すべきです(後述)。

ワイルドカードMX ：

最近はあまり必要になることはありませんし、必要がなければワイルドカードは定義しない方がよいです。ワイルドカードとは「*」にあたるところには何が来てもいい、どんな名前が来てもそれに対するレコードが存在するとしてネームサーバが答えるというものです。

これが必要になるのはFirewallがある場合(直接通信ない場合)です。企業の中にいろいろな部署があって、それぞれ別々のドメイン部を持っているとき、外に対して内側のホストの情報を見せたくない場合にワイルドカードMXを使ったりします。そうすると内側のホストを細かく指定する代わりに次の例のように、1個所をワイルドカードMXで指定しておくことで全てmail.x.co.jp.にメールが届きます。

*.x.co.jp. IN MX 10 mail.x.co.jp.

ここではx.co.jpより左側部分にピリオドが1つしかありませんが(x.co.jp.のすぐ左にホスト名がきますが)、nohost.x.co.jp(明示的にホストが定義されていない)だけでなく、host.nosubdom.x.co.jp (間にサブドメインが含まれている場合)にもマッチします。

ここで注意しておきたいことは、ワイルドカードMXを使うときによく次のような勘違いをされることです。

ワイルドカードの部分は、specificなホスト名が存在する場合は機能しません。以下の例のように1行目と2行目にそれぞれAレコード、ワイルドカードMXが定義されていたとき、MXレコードはいろいろな名前に対してワイルドカード・マッチしてくれますが、この例では、唯一nsと書いてあるものに関してはexactにレコードが存在しますので、このワイルドカードMXはマッチしません。すなわち、ワイルドカードMXは、ns.x.co.jpに対する検索には利用されないので、nsに関してMXを設定したい場合は必ずワイルドカードMXと並べて3行目に示すようなexactなレコードを設定する必要があります。

```
ns.x.co.jp.     IN A    12.34.56.78
*.x.co.jp.      IN MX 10 mail.x.co.jp.
ns.x.co.jp.     IN MX 10 mail.x.co.jp. (必要)
```

ns.xと書いたときに自動的にco.jpを補うことを補完と呼びますが、ネームサーバを引くことによって補完を行わせる設定をした場合に、このワイルドカードが邪魔をする場合があります。

CNAME (Canonical NAME) RR:

CNAME RRは、あるホストの名前に対してエイリアス(別名)を定義します。

例えば、archie.wide.ad.jpというサービス名があり、実際にはsun3.tokyo.wide.ad.jpという計算機がarchieのサービスをしている場合に、archieをsun3の別名として定義するのがCNAMEの機能です。

例えば、host1.x.co.jp.がpopのサービスをしていて、ユーザに対してpopという名前を見せたい場合に以下のように記述します。

```
pop.x.co.jp.  IN  CNAME  host1.x.co.jp.
```

このように定義しておくと、サービスするホストが後で替わったときにユーザは何も設定の変更をすること無く、管理者の都合だけでサービスする計算機を変更することができます。

CNAMEを定義するときのポイントは、CNAMEと同時に別のRRを定義してはいけないことです。すなわちpop.x.co.jp.に対してCNAMEが定義されているのに、さらにMXとかAとかを定義してはいけませんので、注意して下さい。

CNAMEは基本的にはホストに対するエイリアスを定義するものなので、MXに対するエイリアスとして使ってよいかという議論がありますが、色んな実装で動くようです。

nslookupでCNAMEを確認する場合の例:

```
% nslookup -q=cname archie.wide.ad.jp.
Server:  localhost
Address:  127.0.0.1
```

この問い合わせに対し、結果が以下のように返ってきます。

```
archie.wide.ad.jp  canonical name = sun3.tokyo.wide.ad.jp
```

DNS検索の手順:

DNSを使ってメールが送られて来る場合の話をしていますが、そのときのDNSの検索の手順をまとめてみました。

1. CNAMEを解決

   相手がメールを自分に対して送ってくる場合、どのように検索が行われるかというと、先ず、メールアドレスがCNAMEでないかどうか調べます。

   CNAMEであった場合は、そのネームが指している本名を引いて、それに置き換えて行くことを繰り返します。RFCでは、CNAMEに対するCNAMEはすべきではない、と規定されていますが、実装はそれがたどれるように作らないといけないとなっています。従って、事実上は、CNAMEのCNAMEのCNAMEもできます。但し、8段階とか10段階とかの制限は設けられていますので、あまり深くすることはできません。

2. MXで検索

   CNAMEがたどれたら、次はMXを検索します。メールアドレスのドメイン部に対してMXレコードが存在するかどうかを調べに行きます。存在すれば、プレファレンスを使って順位付けをします。得られたMXの右辺については、それぞれAレコードの検索を行います。MXの検索に応答したサーバがMXの右辺に指定されているホストに関するAレコードを知っていた場合は、MXの応答と同時にそのAレコードに関する情報も返されます。

3. Aで検索

   MXレコードが得られなかった場合は、Aレコードを探しに行って、メールアドレスのドメイ

ン部に対応するIPアドレスを見つけます。

今まで出てきたDNSの3種類のリソースレコード、RRに関して、このように検索が行われて、メールの送り先が決定されます。

そこで、注意点として、もしメールアドレスのドメイン部に対応するレコードが、Aレコードにしか定義されていなければ、検索処理は2回必要になります。すなわち、MXを聞きに行って無いとわかって、Aを聞きに行って答えが得られます。このように、2往復のトラフィックが発生します。

もし、最初にMXの検索で情報が見つかった場合は、多くの場合、その情報とさらにそのMXレコードの右辺に指定されたホストのIPアドレスが一緒に返って来ますので、MXが存在すると問い合わせは1往復で済みます。Aレコードしか定義されていない場合、メールは届くことは届きますが、余計なトラフィックが発生していますので、通信トラフィックを削減するためには、メールを受信するホストにもMXレコードを定義すべきとなります。

この辺をしっかり把握しておけば、2往復が1往復で済むために通信トラフィックが減り、メールを送る前の無駄な時間が省かれます。

その他の注意事項:

メールを自分のホストに送ってもらうための設定でメールサーバに関する説明をしましたが、その他の注意点としては以下のものがあります。

- ホスト名に利用できる文字
- MX RR の右辺と CNAME
- メールアドレスと CNAME
- CNAME のチェイン

「ホスト名」に利用できる文字:

基本は、アルファベット(A-Z, a-z)と数字(0-9)とハイフン(-)ですが、最近問題になっている注意すべき文字としてアンダースコア(__)があります。

ホスト名にアンダースコアを使っているところがたまにありますが、RFC1035(S)やRFC1123(S)では許していません。それに対応して、新しい(4.9.4以降の)bindのresolverを利用する場合、アンダースコアを含むホスト名は無視されてしまいます。従って、アンダースコアをホスト名に入れていると、それに対するIPアドレスが定義されていてもresolverで引いたときに、それが見つからないということになり、そのホスト宛てのメールが届きません。このような現象が、最近、ちらほら報告されていますので、アンダースコアを使ったホスト名は使わないようにして下さい。

ネームサーバに対する定義が誤っていると、最近のネームサーバは間違っていることをメッセージでsyslogに残してくれますので、syslogをしっかりチェックしていれば自分の設定のミスも見つかると思います。

メールアドレスとCNAME:

メールアドレスとしてCNAMEに定義されているアドレスに対してメールを送っても一応メールは送られて来ます。ところが、RFC1123(S)を見ると、メールアドレスがもしCNEMEであれば

それは別名でなく本名に書き換えなくてはならないと規定されています。

すなわち、本当はエンベロープの情報に対してCNAMEを本名に書き直さないといけないということになっていますが、多くの(古い)sendmailはヘッダにあるメールドレスに関しても別名を本名に書き換えてしまいます。そこで、1つのホストに対して複数のサービスを用意してCNAMEで別名を与えてメールアドレスを分けていたとします。すなわち、沢山サービスを用意していて、それぞれにドメイン部を分けたメールアドレスを用意し、そのメールアドレスによって受信時の動作を変えたり、人間が読んだときにどのサービスに対するクレームなのか、をわかるようにしたいと思っていても、このようにメールのヘッダに対する書き換えが行われてしまうと、CNAMEでサービス毎に名前を分けていてもそれらが区別できなくなってしまいます。

これに関しては、自分のホストの設定だけではいかんともし難く、世界中で動作しているMTAにおけるヘッダに対する書き換えを止めない限り、CNAMEでメールアドレスを用意しておいてもそれが区別できなくなってしまいます。

そこで、CNAMEによる書き換えが行われたら困る場合は、CNAMEを使用する代わりにMXとAを使って設定をする必要があります。

CNAMEによる書き換えはあまり意味が無いという認識があるのか、最近のIETFではそういう書き換えは必要ないので止めましょう、ということになりつつあるようです。

CNAMEのチェイン:

以下のように、エイリアスを二重に参照すること、すなわちCNAME RRの右辺がさらに別のCNAME RRの左辺を指すような設定は、すべきでないとなっていますが、実際はできてしまいます。

```
alias1  IN   CNAME   alias2
alias2  IN   CNAME   real-name
```

DNS管理入門 - DNSデータの追加作業 –

データベースに対して新しいレコードを追加するときの処理をざっと紹介しておきます。ゾーンは後述しますが、ここでは、自分が管理しているネームサーバのデータベースのことを指していると思って下さい。

ゾーン・ファイルを探す:

稼動しているネームサーバのバージョンによって、探しかたが代わります。BIND 4.x の場合は、/etc/named.boot というファイルが存在するはずなので、まずその内容を確認します。BIND 8.x の場合は、/etc/named.conf の内容を確認します。BIND 4.x の場合、directory 定義と、自分のドメインに対応するファイル名を指定する primary 定義から、ファイル名とその置かれている場所を特定します。

・ /etc/named.boot (bind 4.x)

```
directory /etc/namedb
primary     my.domain.jp   my.domain.zone
```

→ /etc/namedb/my.domain.zone

レコードの追加:

ファイル内のレコードの記述で、レコードの左辺を省略した場合は直前のレコードの左辺が指定されたものと仮定されます。

```
; $ORIGIN my.domain.jp. (説明のため意図的にコメントです)
@       IN SOA ...
        IN NS
; my.domain.jp. に対する定義
a       IN A    ...
        IN MX   ...
; a.my.domain.jp. に対する定義
```

MX/CNAME RR は右辺にもホスト名を指定しますが、その場合末尾の . に注意してください。末尾に . をつけなかった場合は、$ORIGIN にあたるドメイン名が後ろに補われます。なお、明示的な $ORIGIN の指定がない場合は、named.{boot,conf} で指定されるゾーン名が自動的に設定されます。また、@ は指定されたゾーンのドメイン名自身に対する定義を意味します。

シリアルを増やす:

シリアルはデータベースのバージョンを定義するフィールドですが、データベースを変更した場合は必ずこのシリアルを増やして下さい。シリアルを増やして、内容を書き換えたら、ネームサーバに対してハングアップのシグナルを送ります。最近のネームサーバであれば、コントロールプログラムが付いていますので、ndc を使ってreloadと入力するとデータベースが更新されます。

セカンダリ・ネームサーバの更新:

シグナルで送ってデータベースを更新したときに、セカンダリ・ネームサーバが存在する場合はそちらも更新する必要があります。普通はしばらく時間が経つと更新されますが、急いでいる場合は通知して手動で更新してもらったり、自動通知(データベースを更新すると自動的に関係するネームサーバに更新情報が伝わる)を使います。

手動更新:

すぐに更新が必要な場合は、バックアップファイルを消してからnamedを再起動する必要があります。

3. メールの配信設定の基本

最初にメールの設定に関しての3つのポイントを紹介しました。これまでは、インターネットのいろんなホストからメイルを送ってもらうための設定について説明しました。ここからは自分のところの話になります。メイルが自分のところに届いたが、それをどうするか、自分のメールサーバのことについて説明します。

ここでは、リモートへの配信ということで、届いたメイルをさらにどこかに送る、あるい自分のところからメイルを送る場合の話、すなわち、MTAがあって、ここから出て行くというところの話になります。

配信方法のバリエーション:

- DNSのMX RR参照による配信
- ホスト名のみによる配送
- 固定ルールによる配信

メイルを自分のMTAから送り出すことを考えた場合、方法としては、ここにあげたくらいのものがあります。

まず、DNSを利用する場合ですが、自分が他のところにメイルを送るということに関しても、他のところで設定しているDNSの情報を見て自分がメイルを送り出すということになります。

2つ目のホスト名に関してですが、DNS以外の、たとえば/etc/hostとかを参照して送るとか、DNSを見てもMXを見ずに単にホスト名に対応するAレコードだけを見て配信するということです。このような設定は可能ですが、あまり行われません。

3つ目としては、固定ルールによる配信で、こういうメイルアドレスがあった場合は、このホストに送ってねというのを固定的に書くというものです。UUCPなどの場合にはこの方法が用いられます。

リモートへの配信についてはこれくらいの選択肢がありますが、インターネットに対してメールをやり取りする場合は、DNSを参照しないと話になりませんから、その話を中心に進めます。

DNS参照のための基本設定:

DNSを参照してメールを送り出すためには、 /etc/resolv.conf やサービススイッチファイルなどの、メールサーバとして使おうとしている計算機がDNSを参照できるような設定をするところからはじまります。

従来は/etc/resolv.conf だけを設定すればよかったのですが、最近のSolarisやUltrix等ではさらにサービススイッチファイルと呼ばれるファイルが存在するようになり、それによってホスト名からIPアドレスへの検索の際にどのデータベース(ホストとIPアドレスの関係を保存している)を使うのか、例えばDNSなのかNISなのか/etc/hostを使うのか、を細かく選択することができるようになっています。

この辺の説明は、最近たくさん出ている、ネットワーク管理の本に出ているでしょう。

- /etc/resolv.conf
  - ネームサーバの指定(どのアドレスがネームサーバかを指定)

    nameserver 0.0.0.0　(localhost - 127.0.0.1 と解釈)

    nameserver 12.34.56.78

nameserver 12.34.56.79

– 3つまで (MAXNS in resolv.h)

• いくつのサーバを指定しても全体的なタイムアウト時間は変わらない (75秒)

• 補完する自分のドメイン名の定義

domain sub.x.co.jp

search sub1.x.co.jp sub2.x.co.jp x.co.jp

— ドメインを省略したとき、自分のドメイン名が補完されますが、その際に補う名前を定義します。例えば、Aを検索しに行ったときに、ただAを探すのではなくA.sub.x.co.jpというように後ろに補ってくれる機能で使用されます。複数のサブドメインを頻繁に指定するような場合に、それらの複数のドメイン空間で探したい、この例ではsub1、sub2のサブドメインや自分の直下のドメインで見つけて欲しい場合に、domainの代りにsearchを使用します。これらの定義はsendmailのメールアドレスの補完にも利用されます。

・ サービススイッチファイル

Solarisの場合は /etc/nsswitch.conf というファイルがあり、そこのhostsにfiles dnsとあれば、先ず/etc/hostsファイルを見に行って、そこに無かったらDNSを見に行くという動作をします。

この辺はプラットフォーム毎にファイル名が違います(DECだと/etc/svc.conf)。

最近のsendmailは、ServiceSwitchFileでsendmail向けのservice.switchというファイルが参照できるようになっています。

既にこのようなサービススイッチファイルが用意されているプラットフォームであれば、それが参照されますが、それ以外のプラットフォームであればsendmail独自の同等なファイルの中にこのように記述をしておくことで、ホスト名のデータベースの検索の順序を細かく定義することができるようになっています。

DNSのMXを参照する場合:

DNSを参照してメールを送り出したい場合は、先ず、MXを参照して送り先を決定できるMTAを用意する必要があります。そのような機能を持つsendmailをsendmail.mxと呼びますが(SunOSにおいてそのような名前で提供されていたことに由来します)、新しいバージョンのsendmailは普通にコンパイルするとこのMXを参照するsendmailができあがります。MXを参照するにはlibresolv.aというライブラリをリンクする必要がありますが、これはBindのパッケージに付いてきますし、普通のunixのプラットフォームであれば初めから付いています。

sendmailであれば、このようなMTAを用意すると共に、sendmail.cfがMXを参照するようにできあがっている必要があります。

CFでsendmail.cfを作成する場合には、MX_SENDMAIL=yesにしておく必要がありますが、これはデフォルト値ですので普通に作るとMX対応のsendmail.cfができます。

固定ルールによる配信:

DNSを見るのではなく、sendmailに、直接このアドレスであればこのホストにふって欲しい(例えばUUCPに投げて欲しいとか)という設定をする場合は、sendmail.cfに固定ルールを書くことになります。これは、昔のmailconfが非常に得意な分野でしたが、CFの場合はSTATIC_ROUTE_FILEを用意することで同じことができるようになっています。

詳しい設定は、それぞれのドキュメントを参照して下さい。

配信先の確認:

sendmailの場合に、上記のように設定したときに、正しく設定できているかどうかを確認する方法を紹介します。最近のsendemailで新たに追加された機能を中心に話します。

ここでチェックするのはDNSを使って配信する場合の確認で、以下の2つの方法があります。

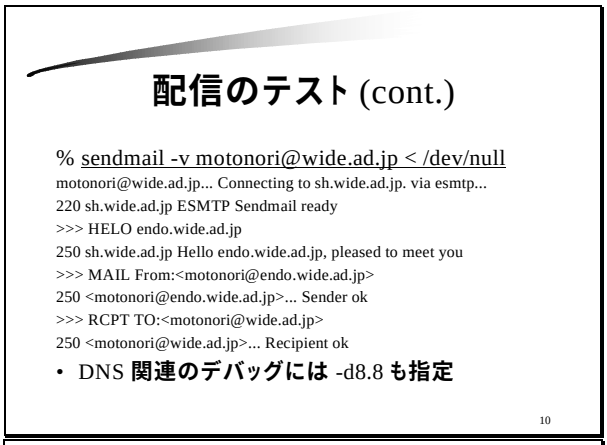
- アドレスの解釈が正しいか。
  sendmail.cfでちゃんとDNSを参照するような動作を行うか。
  – sendmail -bv や sendmail -bt の /parse
- MX が正常に検索できているか。
  実際にDNSを引きに行って期待通りの答えが得られているか。
  – sendmail -bt で /mx コマンド

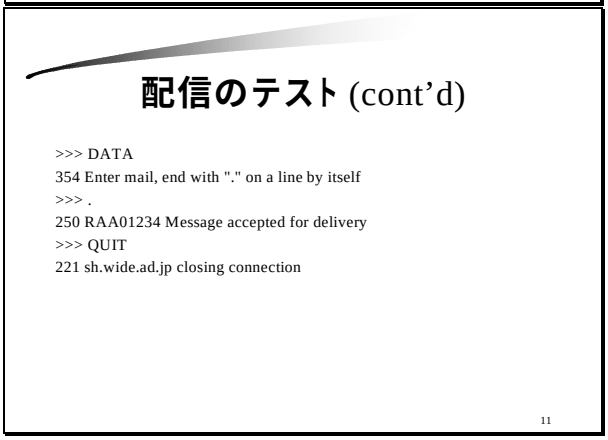
・ MX 検索の確認

これは、mslookupを使って確認することもできますが、実際にはsendmailがMXを引いて配信するので、sendmail自体から正しく引けているかどうかを確認した方が良いでしょう。

新しいsendmail(バージョン8.6以降)では/mx wide.ad.jpと入力すると、それに対するMXレコードが出てきます。ここではjpの後のピリオドは不要です。実際にMXが正しく引けていることが確認できたら、次は配信テストを行います。

・配信のテスト





これらはsendmailコマンドを実行した例で、コマンドとそれに対する結果です。

SMTPでエンベロープが送られる部分が確認できますが、ここでチェックするところは、メールアドレスのドメイン部に対して正しく期待通りのホストにつなぎに行っているか、を確認することです。

DNS関係で正しくデータが引けているかどうかを確認するためには、デバッグ用の-dオプションを使って調べるとよいでしょう(この例では指定されていません)。

メールを送り出すために、DNSを参照して送り出すことを前提に話しましたが、この場合ホストがちゃんとDNSを参照できるかどうかを確認できさえすれば、それによってメールがちゃんと送り出されます。次に、それを詳しく説明します。

4. sendmail の概要

メールをもらうのはDNSの設定の話でしたが、受け取ったメールをどうするかというのは、MTA毎の細かい設定の話になります。基本的な概念はどのMTAでも同じですので、概念的に同じ部分を中心に説明します。

Sendmailの紹介からはじめます。

sendmailはMTAの一つで、メールを受信し、配信先を決定し、実際に配信するプログラムです。

Sendmailの利点 :

世の中には、いろいろのMTAがありますが、sendmailの利点は全ての設定がsendmail.cfで行われるということです。このファイルさえ調べれば、どんな動作をするかを把握できます。

柔軟性があり、機能が豊富で、アドレスの解釈の部分をsendmail.cfの中で行っていますので、インターネットのドメイン形式以外のいろんな新しい体系のメールアドレスに対しても対応できます。また、メールゲートウェイとして、インターネット以外の様々な形式のアドレスを持っているネットワークとの間で橋渡しをさせることができ、非常に強力です。

バイナリが1つですので、例えばqmailのように沢山のバイナリに分かれていてユーザ登録をたくさん行うなどもなく、インストール/バージョンアップが簡単です。

歴史が長く、広く利用されているので、Known Howが蓄積されておりいろいろな活かし方ができており、それらの情報が簡単に得られます。また、管理技術者が豊富(?)、バグが枯れてる(?)、開発が続いている、などがあります。

Sendmailの欠点 :

強力であることの反面、sendmail.cfが難解です。本来、普通の管理者にとっては関係ない部分も書かれていますので、その部分を細かく理解しようとすると非常に難解になってしまいます。$という記号をメタとしていろいろ使っていますので、はじめて見たときはは戸惑うこともあるかもしれません。ルールセットという方式を使っていますので、メールアドレスの処理の流れをたどって行ったとき、プログラムの経験がないと解釈が難しいかもしれません。

セキュリティホールがいろいろ報告されていてsendmailは危ないと言われていますが、デーモンとして常駐して、外からコネクションを受けてサービスするプログラムでは、他のいろいろなプログラムでもそのようなセキュリティホールが報告されていますので、仕方がないことかもしれません。もちろん、qmailのように、セキュリティホールの可能性を極力減らすこともできますが、現在のsendmailはそのようなアプローチをとっていません。また、root権限を持って処理を行っているところも1つの問題点ですが、基本的な部分をきっちり管理して設定しておけば、特に問題はないはずなので、その辺を注意しましょう。

また様々な機能拡張のため、バイナリが肥大化していますが、最近の計算機の高機能化やメモリの増大によって救われているところがあります。配信処理が遅いことも、最近、顕著な問題となっていますが、後で対応策を説明します。

Sendmailのバージョン:

歴史が長く、以下のように発展してきています。

- Delivermail (4.0, 4.1BSD)
- Sendmail 4.12 (4.2BSD)
- Sendmail 5.52 (4.3BSD)
- Sendmail 5.67 (R5 final)
- Sendmail R6 project 開始 (92/12)
- Sendmail R8 (4.4BSD, 93/6)
- Sendmail 8.6.x (こうもり本1st, 93/10)
- Sendmail 8.8.8 (最新バージョン, 97/10)
- IDA(UIUC, KJS)版、WIDE版
- OSベンダによる独自拡張

Sendmail R8の新機能:

以下の通りです。重要なものは後述します。

- 8Bit クリーン
- MX RR レベルの相乗り配送判定
- コネクション・キャッシュ
- Ident (識別)プロトコル (133/TCP)
- ESMTP (Extended SMTP)
  - DSN (Delivery Status Notification)
- 未配送通知
- Berkeley NewDB の採用
- 外部データベース

Sendmail R8.7 の新機能:

- 8Bit → 7Bit (MIME形式)
- SIGHUP による再起動
- テストモードの拡張　　前述のテストモードの拡張は8.7で行われました。
- Apparently-To:の事実上の廃止
- Return-Receipt-To:の廃止(DSNへ)
- LongName マクロ/クラスの採用
- 外部データベースの拡充
  - サービス・スイッチ・ファイル、NIS+

Sendmail R8.8の新機能:

最近、問題が顕著になって来ているSPAM、余計なメールの中継処理(関係ないホストを中継拠点としていろんなごみメールを投げ続ける)問題を防ぐ手段をcheck以降のセットで提供されています。

セキュリティについて、ローカルファイルのパーミッションに関しても厳重にチェックするようになったので、それによって簡単にバージョンアップすると、これまでゆるかったので今まで動

いていたメーリングリストが動かなくなる等の話が出てきています。その場合、この辺をしっかりチェックすると問題なく移行できます(後述)。

- PersistentHostStatus
- SingleThreadDelivery
  - 配送状況情報の活用
- 7Bit → 8Bit デコード
- check_* ルールセット
- ETRN (外部から sendmail -q)
- ファイルのパーミッションチェックが厳密(8.8.6〜)
  - group writable, symlink
  - NFS, chown check

Sendmailのインストール

メールサーバでメールを送り出す、あるいはメールを受け取って送り出す場合に、何かMTAを使う必要がありますが、ここではsendmailのインストール方法を簡単に紹介します。

MX版とNOMX版:

sendmailにはMX版とNOMX版がありますが、最新のsendmailの場合はデフォルトでMX版ができます。

MX版を作るときはBindのライブラリをリンクする必要がありますが、バージョンによってリンクするライブラリが変わりますので、注意が必要です。

NOMX版を作るときは、makefileの中で-DNAMED_BIND=0としておく必要があります。

普通のインターネットとお話するsendmailを作る場合はMX版でいいですが、firewall上やfirewallの中でDNSを見ずに使いたい場合にNOMX版のsendmailを作り、使っているところがあります。

- MX版 (sendmail.mx)
  - -DNAMED_BIND=1
  - DNSのMX RR参照機能つき
  - -lresolv -l44bsd (bind 4.9.6)
  - -lbind (bind 8.1.1)
- NOMX版 (sendmail.nomx)
  - -DNAMED_BIND=0
  - DNSのMX RR参照機能なし
  - Firewall内用
  - メールサーバ従属クライアント用

データベース･ライブラリの組込：

sendmailを作るときに、今のところconfigureのようなコマンド1つで最後までチェックして思いのものができるようにはなっていませんので、自分が使いたいsendmailに、どんな機能が必要か、どんなfeatureが必要かを整理して、それに従ってmakefileを手直しする必要があります。

ここで言っているデータベースは、エイリアス関係のもので、高速に検索するためのものとして何を使うか、を指定します。

最初の2つ(DNDBM、DNEWDB)は、ローカルでエイリアスのデータベースを管理するためのものです。ちょっと前までのunixの場合は、普通NDBMが基本でしたので、それを使ってデータベースを管理しました。

これを高性能化して、例えば1行1024文字の制限とかを取り払った、新しいものとしてBerkeley NewDBが出てきています。最近のBSD 4.4とかであれば、NewDBがはじめから入っていますので、それを使うようにコンパイルすればよいです。

但し、今基本として使われているのはdb1.8だったと思いますが、それに加えて最近、db2が出てきていますのでこちらを使う手もあります。現在のsendmailのソースがdb2に完全対応していませんので、db2を使うときに手直しが何ヵ所か必要です。

後は、NIS経由で、ネットワーク経由でエイリアスのデータベースを引きたい場合は、NISの定義をする必要があります。

- -DNDBM - (aliases.{dir,pag})
  - 旧来の ndbm
- -DNEWDB - (aliaes.db)
  - Berkeley NewDB (db1.xx, db2.xx)
    - http://www.sleepycat.com/ (db2)
      --enable-compat185
  - -ldb
- -DNIS, -DNISPLUS
  - Sun's Network Information System

コンパイル：

makefileの変更を行うわけですが、それを行うタイミングが難しいです。sendmailの場合は、configureを使う代りに、makesendmailというスクリプトが用意されています。これは、1回実行してしまうと、コンパイル環境を作って完成させてしまうスクリプトですので、途中でmakefileを変更したい場合は中断させる必要があります。

- sh makesendmail
  - コンパイル環境の構築 (obj.*/)
  - 自動的に Makefile が選択される

コンパイル環境が作られていって、例えば最初のailiases.cとかのコンパイルが始まったら、おもむろに中断させてmakefileを変更します。変更点は、以下を参照下さい。

- 一時中断して obj.*/Makefile の変更

- BIND 関係
  - sendmail.nomx は -DNAMED_BIND=0
  - bind 4: -lresolv, -l44bsd (必要なときだけ)
  - bind 8: -lbind
  - BIND の lib / include パスの指定
- NDBM / NEWDB / NIS
  - NEWDB
    - -ldb
    - lib / include の指定
  - NDBM と NEWDB の同時指定には注意
    - NEWDB は NDBM コンパチ・インタフェースを持つ
    - aliases.{dir,pag} 形式を利用する場合はNEWDB からコンパチ対応部分を除去しておく

      NISのサーバになっているが、ローカルではNEWDBを使いたい、あるいは、NDBM形式のファイルも作りたいし、NEWDB形式のファイルも作りたい場合は、両方を混ぜて作る必要があります。ところが、NEWDBにはNDBMのコンパチライブラリが含まれていますので、NISのサーバの場合に必要となるdirやpagのファイル形式を作るときは、そのコンパチライブラリを外しておく必要があります。

・ obj.* ディレクトリで make を継続

・ groff がないとき

groffを要求してくる場合がありますが、マニュアルのフォーマットをするだけですので、groffをコンパイルするのは無駄でしょう。

動作テスト:

sendmailが正しく動いてることを確認した上で、実際に今まで動いていたsendmailと入れ替える方法を以下に説明します。

既にsendmailが動いているシステムがあった場合に、コンパイル終了後にいきなりmake installなどで入れ替えを行うと、コンパイルしたsendmailがちゃんと動かなかった場合に元に戻すのが大変です。そこで、新たに作ったものに入れ替える前に、それがちゃんと動くことを確認した上で入れ替える手順を踏む必要があります。特に、多くのユーザがいるメール環境でバージョンアップを行う場合は、慎重に行う必要があります。

先ず、sendmailの場合は、2つのファイル(sendmailのバイナリファイルとsendmail.cf)に関してチェックする必要があります。sendmailのバイナリがバージョンアップされる場合、sendmail.cfがそれにちゃんと対応しているかどうか、も重要なチェックポイントです。特に、sendmailを大きくバージョンアップする場合(例えばver.5からver.8にアップするなど)、あるいはベンダーが作っているsendmail.cfを使っていたがこれを新しいバージョンに置き換えたい場合に、以下の項目の対応のチェックが必要になります。

先ず、% ./sendmail -bt でsendmail.cfにエラーがないことを確認します。確認ができたら、次に実際に配信してみます。ローカル(自分のホスト)及びリモート(別のホスト)にそれぞれ送られるか確認します。-vは、詳しいメッセージを出してくれるオプションで、SMTPの状況などを出してくれます。-vが指定されていると-odi(バックグラウンドに落ちずに、そのまま画面に張り付い

て実行するオプション)は省略できると思います。
ここまで確認できると、次はインストールです。

(バイナリのみの置き換えの場合)
- sendmail.cfの対応チェック
  – /etc/sendmail.cf (場所の統一)
  % ./sendmail -bt でエラーがないことを確認
- ローカル配信のテスト
  # ./sendmail -v -odi myname < file
- リモート配信のテスト
  # ./sendmail -v -odi name@remote < file

インストール:
インストールするときはrootの権限を持たせますが、以下のコマンドでsendmailをrootの持ちものにします。
何故rootの権限が要るかといいますと、mailqがrootでないと読み書きできないようになっていたり、aliasesがrootだけで管理できるようになっていたり、マシンの負荷が高くなってくるとメールの配信を止めることができますので、そのためのカーネル内のload averageを読み出すための権限とか、あるいは25番という小さい数字のポートがrootでないと扱えない特権ポートであるなどの理由からです。
mailqやmailstatusというコマンドがありますので、それに対してリンクを張っておきます。ハードリンクで張ってある場合が多いですが、バージョンアップ時に面倒ですのでシンボリックにしておく方が後々楽です。

- SetUID root
  # chown root sendmail ; chmod 4755 sendmail
    - mqueue の管理
    - aliases の管理
    - kmem の参照 (負荷値など)
    - 特権ポート(25)の利用
- mailq, mailstatus にリンク
  # ln -s /.../sendmail mailq
  # ln -s /.../sendmail mailstatus

daemonの再起動:
インストールができたらデーモンの再起動を行います。デーモンモードで起動するには-bdを指定しますが、ここで新しいsendmailの場合は、起動をフルパスで行うことになっています。何故かといいますと、SIGHUP(ハングアップ)の信号を送ったときに最近のsendmailは自動的に再起動するようになっており、その場合に再起動の方法として自分が起動されたときのパスを覚えておいて、それで再度実行するからです。
再起動のための情報はsendmail.pid に記録されています。

- /usr/lib/sendmail -bd -q1h
  - 起動はフルパスで
    - SIGHUP による再起動のため
    - パスはシステムによって異なる
    - /etc/rc* なども確認
      - ブート時の起動
  - 起動時オプションは sendmail.pid に記憶
    - SIGHUP でもオプションは引き継がれる

SMTPの動作確認:

今度は、サーバとしてちゃんと動いているかどうかの確認です。

telnetでポート25番につないでみます。ポート25番で、メールを受信しようと待ち受けているかどうかをチェックします。入力するところはアンダーラインが引いてある部分です。

先ず、HELOで自分の名前を名乗ります。エンベロープの情報をMAIL FROMとRCPT TOで渡します。次に、DATAの後ろにメールの本文(ヘッダとボディ)を書いて、最後にピリオドを打って閉じます。そして、メッセージを受け取ったとくるので、QUITして終わります。

ここまで入力して、SMTPでメールがちゃんと送られれば、デーモンとして正常に動いていることが確認できます。

Sendmailの配信の高速化:

複数の宛先に対してメールを送るときに、sendmailは上から順番に1人ずつ処理を行いますので、最初の方の受信者で時間がかかってしまうと後続の受信者の処理開始時間に大きな影響が出るという問題があります。それを高速化するために、いろいろな手法があります。

- WIDE版パッチ2.7W(beta)

  ネームサーバへの問い合わせやメーリングリストのように沢山の宛先があるときに、それぞれのドメインに対するAやMXを順番に引きに行くと時間がかかりますので、それを同時に聞きに行ってかかる時間を1回に圧縮することと、さらに実際にメールを送り始めるときのSMTPコネクションの接続を同時にすることで反応のないものは放っておいて、反応のあるものから先に処理して行くことによって、配信を高速化するパッチです。

- smtpfeed + WIDE版パッチ3.0W

  最近では、パッチだけでの対応では破綻をきたしたので、その部分だけをsmtpfeedというパッケージに分離して、コネクション自体の処理、すなわちSMTP自体の処理も複数の間で処理をすることを実現しています。

  現在、smtpfeedはβ版で配布しています。WIDE版パッチ3.0Wは、前の版から配信の高速化の機能の部分を削ってsmtpfeedに対応したものを付け足したものです。試してみて下さい。

5. メールシステムのデザインとsendmail.cf

実際にsendmailを使ってどのような設定をして行ったらいいのか、を説明します。3つのポイント、メールを送ってもらう、メールを受理する、メールを転送するのうち受理と転送の説明をします。

sendmail.cf とは：

sendmailバイナリとsendmail.cfのファイルの2本立で、その他のファイルも存在しますが、それらはすべてsendmail.cfの中に記述されています。従って、sendmail.cfがどこにあるかがわかれば、他の設定ファイルがどこにあるのかは全てわかる仕掛けになっています。

どういうアドレスを受理するかと、どのアドレスをどこへ送信・転送するか、が記述されています。

よくある勘違いとして、ネームサーバで自分のアドレスを設定して、このメールサーバに送ってよと書いたら、それだけで受理できると思っている人が割といるようですが、メールをそのホストに送ってもらうという設定と、そのホストが自分宛てのメールだと解釈することは別のことです。

sendmailはSMTP以外のいろいろな方式で配信することができますから、何を使って配信するかということもsenmail.cfの中で指定されます。

- sendmail が参照するファイルの位置を指定
- 配信経路の決定
  - 受理(ローカルへの配信)
  - 送信・転送(リモートへの配信)
    - どのホストに送ろうか
- 配信方法の定義
  - SMTP
  - UUCP

標準的なsendmail.cf ：

sendmailがOSに標準で付いている場合は、sendmail.cfも標準的なものが付いています。

標準的なsendmail.cfでは、大抵は、その計算機に対して定義されているホスト名宛てのメールは受理するよう、設定されています。また、それ以外に関しては、MXを参照してメールを送り出すように設定されています。

ホスト名だけでメールを受ける場合は、標準的なsendmailを使えば何もする必要はありません。

変更が必要になる場合は、メールのドメインマスターのようなgenericなメールアドレスを受け取るとか、firewall上でその組織内のメールサーバにさらに転送する場合です。

他には、メールを送ってもらうときに注意するポイントとして言いましたが、MXの右辺に自分の名前が出てくる場合にそれをちゃんと正しく自分の名前であることを認識してくれる必要があります。そうでなければ、プライマリがダウンしたときにバックアップのメールサーバのメールが落ちますので、この辺の設定がちゃんと行われているかのチェックが必要になります。

- OS にも標準で添付
  - sendmail に合ったバージョンの sendmail.cf を利用する

・ 基本機能
  – 当該ホスト名宛てのメールを受理
  – MX を参照して配信
    ・ 特に何も設定を変更する必要はない
・ 変更が必要な場合
  – メール・ドメインマスタ
  – Lower MX として MX RR の右辺に指定

sendmail.cf のバージョン:

あまり気にする必要はありませんが、各sendmailに対応するバージョンは以下の通りです。但し、sendmailのバージョンに対してsendmail.cfのバージョンが対応していなければならないわけではありません。新しいsendmail.cfの代りに、それより昔のsendmail.cfであればどれでも使える形になっています。従って、極端な話としては、sendmailのバイナリだけバージョンアップして、V1のsendmail.cfを使い続けることも可能です。

但し、新しいsendmailで拡張されている機能を使用するためには対応する新しいバージョンのsendmail.cfが必要になります。

・ V1 : sendmail 5 以前 (無指定は V1)
・ V2,V3 : sendmail 6.x
・ V4 : sendmail 8.5 まで
・ V5 : sendmail 8.6.x
・ V6 : sendmail 8.7.x
・ V7 : sendmail 8.8.x

・ mqueueファイル(qf)のバージョン

送ることができなかったメールを一定期間保存しておくための場所としてmqueueというディレクトリがありますが、その中にできるファイル(一時保存されているメールに関する情報が保存されている)にもバージョンがあります。新しいsendmailで生成されたqfは、古いsendmailでは解釈できませんので、一旦新しいsendmailをインストールしてバージョンアップして、しばらく運用した後に何か不都合が生じて古いsendmailに戻す場合は、mqueueの中に新しいsendmailが関与してできあがったqfが残っていた場合はエラーが起こり、そのメールは捨てられますので、注意が必要です。

古いsendmail.cfは使えるの? :

古いsendmail.cfを新しいバージョンのsendmailで使う話を前項でしましたが、もう少し細かい話をします。いくつか問題があります。

・ sendmail R8 でも基本的に利用可能
・ SunOS による拡張($%yなど)には非対応

バージョンだけに由来するものではありませんが、1つはsunのOSなどベンダーが提供するsendmail.cfがあった場合、ベンダー独自の拡張があります。それらは純粋なsendmailではサポートされていませんので、そのようなsendmail.cfをそのまま使うことはできません。

・ Null address (<>、エラーメールの発信アドレス)に対応しているか?

– 対応していないと @@host になることがある

メールがエラーになって返送されて来て、さらに返送されたエラーメールがエラーになったとき、さらにエラーメールに対するエラーのエラーが発生すると意味が無く、無駄ですので、そのようなことが起こらないようにエラーメールに関しては、エンベロープの発信者アドレスは中身が空のNull address(<>)が付くことになっています。Null addressがエンベロープの発信者として送られてきたとき、そらが正しく解釈されるか必要がありますが、古いsendmail.cfはその辺の解釈が甘くて@@hostになることがあります。だからといって、これに対して返信するとエラーになります。

- list:; 形式に対応しているか?
  – 対応していないと list:;@host になる

あまり見る機会はありませんが、IETFのメーリングリストなどに入って行くとこのような形式のアドレスがヘッダに付いていることがあります。リスト形式のアドレス表記に対応していないsendmail.cfを使っていると、これをローカルパートと思って @自分のホスト を付加したりしますので、このようなアドレスを見たらsendmeil.cfが古いと考えられます。

- %-hack の解釈は正しいか?

sendmail.cfの準備:

sendmail.cfの作り方として、生成のためのパッケージがあります。sendmail自体に付いてくる標準的な生成ツールとしてはm4で書かれたものがあります。mailconfというJUNET時代に活躍したものもあります。だいたいCFが使われていますので、CFの話が中心に出てきます。

- OS 添付のものを修正して利用
- m4 cf 作成ツール
  – sendmail に添付
- mailconf
  – JUNET 時代に活躍
  – 静的配送ルールの生成が容易
- CF
  – sendmail R8 にも対応

CFによるsendmail.cfの作成:

新しいsendmailに対応する部分は標準的に設定されていますので、あとは自分のメールサーバがどのように動いて欲しいのか、どのアドレスをどのように解釈して欲しいのか、という部分を必要に応じて定義するだけで、意図どおりのsendmail.cfができます。

- パッケージの入手
  – make cleantools; make tools が必要なときも
    - sed を使うとき、perl のパスが違うとき
- sendmail.def を記述
  – Standards/* をベースにする
- make sendmail.cf
  – sendmail.def から sendmail.cf を生成

・ sendmail.cf のテストとインストール

.def の最低限の設定:

一般的なsendmail.cfの機能が最も簡単に作れるようになっています。

先ず、hostnameリターンで出てくるホスト名宛てのメールが受理できるものと、DNSを参照してメールを送り出すことができるものが基本として作られます。そのための最小限の設定として、どのバージョンのcfを作りたいのか、自分の使おうとしているunixのプラットフォームは何か、インターネット的にBITNET境界の提供する方式でBITNETに対するメールを送りたいか、を設定します。あとは、これに必要なものを加えて行きます。

・ 当該ホスト名宛てのメールを受理
・ MX を参照して配信
・ CF_TYPE=R8V7
  – sendmail.8.8 用 V7 形式
・ OS_TYPE=bsdos3.0
  – 使用する OS を指定
  – ファイルのパス、mailer の設定など
・ BITNET=mx
  – user@node.BITNET への配信

ドメインマスタの設定:

最も簡単なところから説明しますと、genericなアドレスというのを紹介しましたが、特にgenericなアドレスでなくても別の計算機宛てのアドレスを代りに受け取る場合もあります。そのような自分が受理したいアドレスをACCEPT_ADDRSの部分に並べて書きます。つまり、ネームサーバの設定に従っていろんな計算機が自分のホストにメールを送ってきた場合に、それを自分のローカルスプールメールボックスに送ること(ローカルに配信)をしたいわけですが、その自分のローカルメールボックスに送りたいドメイン部を並べます。

そのホストからのメールの発信者のアドレスとしてgenericなドメイン部を使いたいときは、FROM_ADDRESSにデフォルトで利用してほしいドメイン部を記述します。genericなアドレスは、普通のユーザには便利ですが、管理的なアドレスに関してはできるだけgenericなものではなくてホスト名が特定されるアドレスで発信された方が便利です。どの計算機のルートから送られてきたメールなのかわからないといろいろなシステムのメインテナンス用のメールが来たときに判断が付かないので、そのような管理用のアドレスの関してはホスト名が付くようになっています。

・ACCEPT_ADDRS='x.co.jp'
  – 受理すべきアドレス部 (これがポイント!)
・FROM_ADDRESS='x.co.jp'
  – 送信時のデフォルトのドメイン部
  – 管理用アドレスにはホスト名が付与される
    ・ root, daemon, postmaster,...
・ 複数ドメイン宛を受理
  – ACCEPT_ADDRS='sub1.co.jp sub2.x.co.jp'

NULL Client：

何も設定を触りたくない、メールスプールも持たない、直接メールを他のホストに送らない、一旦メールサーバを経由してメールを送る、といった最小限の動作しかしないようなsendmailを作りたいときはNULL Clientという方式を使います。これを使えば、発信時のドメイン部の設定とかも全部メールサーバ任せとなりますので、メールサーバがどれかを指定しておくだけですぐに使えるようになりますし、ドメイン名が変わったりしても変更は要りません。この場合はCF_TYPEとSPOOL_HOSTだけを設定します。

・ スプールを持たない
・ 全てのメールをメールサーバへ
  – メールサーバのアドレスの定義が必要

CF_TYPE=R8V7-null
SPOOL_HOST=mail.x.co.jp

・ メールサーバにだけ届くアドレスを記述
  – [] で囲む (lower MX があるときに A RR を参照)
  – IP アドレスを記述

PPPクライアント：

最近は、ダイアルアップでアクセスする場合が増えていますが、その場合にfree BSDとか入れてsendmailを使って、メールをやり取りすることが割と行われています。その場合に設定することは以下の通りです。

FROM_ADDRESSでプロバイダのメールサーバのアドレスを指定します。

直接メールは送らずに、すべてそのプロバイダのメールサーバに対してメールを押し付けて、それがさらにインターネットに配信してくれるようにするために、2行目の直接送らないことの指定と3行目のメールサーバの指定をし、メールを送り出す設定をします。

最後の2行で、ダイアルアップ環境で毎回メールを出す度に接続しに行くとメールを出す度にお金がかかるので、できるだけメールを書き溜めてまとめて送り出し節約するために、必ずmqueueに溜めるように設定します。これは何をしているかといいますと、SMTPのメーラに対して、接続のためのコストが高いことを知らせ(EXPはEXPENSIVEの意)、1回目は直接行かずに一旦mqueueに入れてqueueの掃除のときにつなぎに行きなさいというわけです。expensiveなメーラに対してすぐにつなぎに行くなということをグローバルに指定するためにこのCONNECTION EXPENSIVEをTrue(コストが高いメーラにはすぐに送らないようにという指定)にします。そうするとメールは必ずmqueueに溜まって行きますので、あとはsendmail –qでダイアルアップ回線がつながっているときにまとめて配信するようにできます。

FROM_ADDRESS=po.provider.ne.jp
DIRECT_DELIVER_DOMAINS=none
DEFAULT_RELAY=mail.provider.ne.jp
CON_EXP=True
SMTP_MAILER_FLAG_ADD=e

・ 必ず mqueue にためる
・ 接続時に sendmail -q でまとめて配信

PPPクライアントでの考慮点:

- 発信者アドレスの書き換え
  - ローカルのユーザ名と契約のユーザ名
  - userdb, usertable の利用

    プロバイダに登録し契約している名前と自分や家で使っているユーザ名が違う場合に、メールを発信するときに書き換えなければいけませんが、そのためにuserdbやusertableを使います。
- 契約していないアドレスからの発信の抑制
  - check_compatルールセットの利用

    契約している名前でメールが発信されるのはいいのですが、その他に一般ユーザを登録していた場合に、勝手にインターネットに出て行くとまずいので、そういったメールが出て行かないようにcheck_compatを使ってメールの発信を抑制します。
- 自動ダイアルアップ時のタイムアウト

  O DialDelay=15s

  人間が手動でつなぐのではなくmqueueの掃除のタイミングで自動的につなぐ設定をしている場合、ダイヤルアップの時間が割とかかりますので、その時間を考慮したDelayの設定をします。
- 受信は POP 等で

Firewallとメールサーバ:

Firewallが出てくるとメールサーバだけではなく、いろんなサービスがFirewallのところで一旦切れますので、そこを抜けて行く方法を考える必要があります。

Firewallを境にして、外向き用のNS(NameServer)と内向き用のNSの2つを用意するのが基本的な方法です。それを1台で頑張ることもできなくはないですが、どれだけ組織内でＮＳを使うかにも依存します。NSとメールサーバの構成の関係によって、いくつかのやり方がありますが、3つの方法が出てきます。

ネームサーバとメールサーバの構成

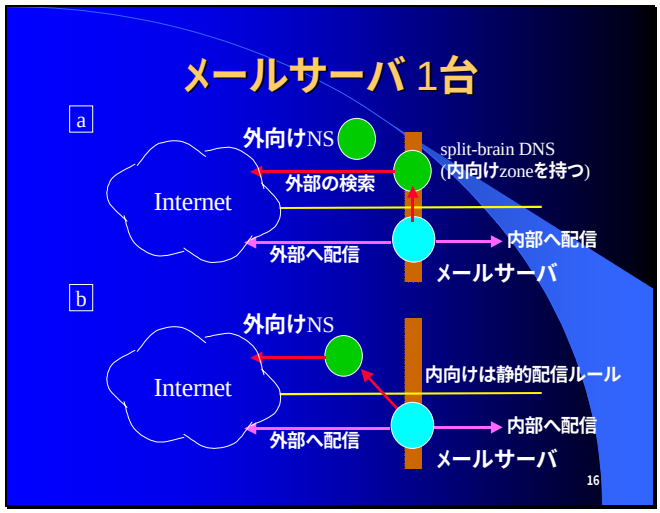
メールサーバは1台:

メールサーバ1台で頑張る場合、2つの方法があります。1つ目は、マージ系列という形で外向けのNSの内容と内向けのNSの内容の両方を見られるものを用意する方法と、もう一つは内向けのNSを用意せずに内側に関しては固定ルールで配信する方法です。あまり内側に対してホスト毎に細かく設定せずに、2つか3つのサブドメインしかない場合は簡単に方式bはできます。

方式a 内向けzoneを持った外部検索用NSを参照

・ split-brain DNS

方式b 内部は固定ルールで配信



メールサーバ 1台の構成

メールサーバを2台:

メールサーバ1台で構成すると制限が出てきますので、設定する面で一番楽な方法はメールサーバを2台用意する方法です。すなわち、役割を2つに分け、外部DNSを参照し外にメールを出すメールサーバと、内部DNSを参照して組織内にメールを配信するメールサーバの2台を用意する方法です。

この場合にその2つの間をどうするか、には2つの間を静的に設定する方法と、全く独立にする方法があります。

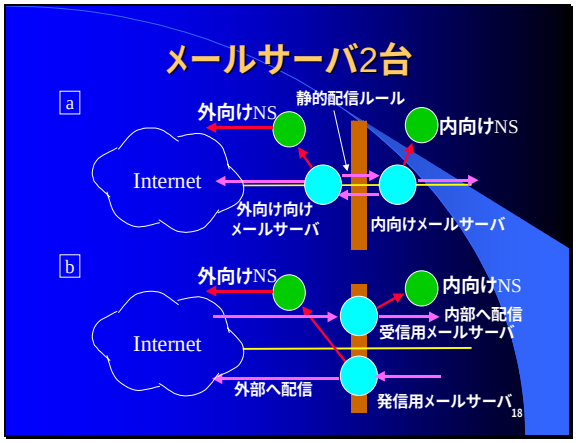
・ 方式 a

– 両者間は静的経路設定

・ 方式 b

– 受信専用メールサーバ

– 発信専用メールサーバ



メールサーバ2台の構成

メールサーバとNSを1台で構成する場合、例えばフィルタリング型のFirewallを用意しているのであればあまり考える必要がありませんが、そうでない場合は内側のアドレスに関して下にさらに内側向けの特別な階層を作って、内側のドメインに関しては特別な階層の下にもう1回アドレスを定義しなおして、そこを見て配信させる、NSに混ぜておく方法があります。

そういった設定をする場合に内向けのメールサーバの設定がどうなるかを考えてみます。

内部向けメールサーバ:

外向けと内向けの2つのメールサーバを用意するわけですが、その場合、内側に向けて配信するものは、例えばCFで定義するとDIRECT_DELIVER_DOMAINSに自分の組織のアドレスを書いておきます。そうすると、自分の組織に属するアドレス、ドメイン部に関しては直接NSを見て配信してくれますが、それ以外のものに関してはDEFAULT_RELAYということで外向けのメールサーバのアドレスを書いておくと、そこに送ってくれます。

・ 外部への静的配信ルール

DIRECT_DELIVER_DOMAINS=x.co.jp

DEFAULT_RELAY=external.x.co.jp

– 外部向けメールサーバ

外部向けメールサーバ:

基本的に全部DNSを参照して送り出すわけですが、自分の組織宛てだけはDNSを見てはいけない、DNSを見ると自分を指しているので、自分に接続しようとしてループメールになりメールが落ちますので、STATIC_ROUTE_FILEを使って定義します。例えば、下の例では、x.co.jpというドメインに属するアドレスに関しては、[12.34.56.78]というIPアドレスを持つゲートウェイを指定しています。

このようにして、外に対してはNSを見て配信できますし、内側のこのドメインに対しては、こ

のゲートウェイ(内向けのメールサーバ)に投げます。

・内部への静的配信ルール

```
STATIC_ROUTE_FILE=x.static
x.static:
  GW        [12.34.56.78]
  #         (internal.x.co.jp)
  DOM   x.co.jp
```

このような2段構えのメールサーバの構成をする場合は、これで設定はほとんど終了です。あとは、Firewallを挟んで、外向けと内向けのメールサーバの間で通信ができるように設定をしておけば終了です。このような構成を取っておけば設定は割と楽です。

メールサーバは、external.x.co.jpやinternal.x.co.jpのようにメールサーバのホスト名がありますのでそれぞれへの宛先のメールは、デフォールトで受理されます。

ネームサーバを1台で構成する場合の問題点を説明します。

1台構成の場合は、外からと中からの両方のNSの問い合わせに対して答えなければならので、両方から見える場所にNSを置く必要があります。Firewallを設けているのに、内側の情報を外から見られると困る状況が起こります。そこで、allow-queryの設定がNSまわりであり、問い合わせをしてもよいアドレスが定義できますが、これだけではうまくいかないことがあります。

外と内で直接通信できない、フィルタ型でない場合を考えたとき、NSは両方から見えるところに置かなければなりませんが、設定上で注意しなければならない問題として、直接通信できないホストがMXレコードに書いてあったり、外から見えないはずのレコードがAレコードに書いてあることがあります。

first MXとして組織内のホストが書いてあり、セカンダリとしてゲートウェイ(firewall上のメールサーバ)が書いてあると、外からメールが飛んできた場合はゲートウェイが受け、ゲートウェイは内側のホストに投げたい、それを1つのNSで行いたい、そうするとゲートウェイとしては配信先のさらに内側のホストがありますので、ゲートウェイは必然的にセカンダリになります。外から送って来るときは、先ずプライマリを試そうとしますが、Firewallがあって通信できませんから、あきらめて次に行きます。そうするとセカンダリが受け取って、それがfirstに送りますので、メールの通信としてはちゃんと動きますが、余計なタイムアウトが発生します。そうすると、メーリングリストで遅延の原因になったり、いろいろな通信が遅れる原因になってしまい、良くないので、この方法は不可能ではありませんが、もう少し考えた方法での設定が必要になります。

その1つの方法は、firstMXをゲートウェイにする方法です。ゲートウェイは自分のfirstなので、その転送をどうするかをsendmail.cf的にちゃんと設定する必要がありますが、3通りの方法があります。

1つ目は、静的な経路設定で、STATIC_ROUTE_FILEで内側のホストに関する転送先を設定するsendmail.cf的に頑張る方法と、もう1つNSのAレコードで頑張る方法があります。これもNSを1つで頑張るということなので、外からAレコードが見えてしまうのでallow-queryを設定したりするわけですが、Aレコードを定義してさらに、自分がfirstMXだった場合にどうするかという動作の設定を切り換えることができます(ここがポイント)。

どのように切り換えるかといいますと、自分がfirstMXだった場合に、もしAレコードが同じ名

前に対して定義されているとそちらを優先することができます。基本はMXがあったらAレコードは見ないということですが、MXが自分だった場合は普通local configuration errorという形ではじきます、つまり静的に自分が受理するあるいは静的な受け入れ定義をしておいて他のところにCF的に(明示的に)送りますが、Aレコードを優先する設定を行うことによって、MXもAもある場合、先ずMXを見てメールが飛んできて自分がfirstMXだった場合だけAレコードを見てそちらに転送することができます。

それをするためには、sendmail.cfで、CFの場合はこういう定義をする、sendmail.cfに直接書く場合はこういう定義をすることによって、そういう動作になります。

こうするとタイムアウトの待ちはなくなりますので、だいぶましになります。

Firewallとメールサーバ:

- どちらも1台でなんとかする
  a. 内側にfirst MXを向けておく
  inner-host IN MX 10 inner-host
  IN MX 20 gw
  - 1st-MX と直接通信できず、タイムアウトが発生
  b. 内側は A RR を参照して配信
  inner-host IN A 12.34.56.78
  IN MX 10 gw
  c. 内側を別の枝にマップ
  - inner.domain.jp → inner.domain.jp.local
    – sendmail.cf でアドレス変換
    – STATIC_ROUTE_FILE の MAP 行 (CF)
  d. 1台に複数のデーモンを起動する
  - IP アドレスにバインド
    – 外向け named と 内向け named
    – 外向け sendmail と内向け sendmail
      • O DaemonPortOptions=Address=12.34.56.78

NSを1台で:

- a, b 方式の問題
  – 内外の直接の通信は不可なのに
  – 外から内部ホストの情報が見える
    - bind8のallow-queryだけでは役不足
- a 方式の問題
  – 直接通信不可のホストに外部から接続を試みさせるべきではない
    - 1st MX が内側ホスト
    - 2nd MX がゲートウェイ
      – 余計なタイムアウト待ちの発生
- b 方式の具体的設定
  – ゲートウェイから内部への配信
    - 静的経路定義

・A RR を見る
– 1st-MX が自分だった場合の挙動の変更
TRY_NULL_MX_LIST=True (CF)
O TryNullMXList=True (sendmail.cf)
– 正しく設定できていないと
local configuration error

ゲートウェイ内クライアント:
ゲートウェイの中のクライアントの設定として、いくつかの方式があります。

・なんでもGWへ
DIRECT_DELIVER_DOMAINS=none
DEFAULT_RELAY=internal.x.co.jp
Null Clientとよく似た方法で、全てゲートウェイに送る、但し、自分はスプールを持っている場合、すなわち、自分宛てのメールは、自分で受理してスプールに落しますが、自分から送るメールは全てゲートウェイに送る場合の設定です。
・内部直接配送
DIRECT_DELIVER_DOMAINS=x.co.jp
DEFAULT_RELAY=internal.x.co.jp
全部ゲートウェイに送ると、内側のマシンが沢山になった場合に、何でもゲートウェイを通過して行くのでゲートウェイの負荷が増大し大変なので、内側のメールに対しては直接配信したい場合は、DIRECT_DELIVER_DOMAINSをnoneではなく、組織内のドメイン名にします。
・スプールなし
– NULL Client

バックアップメールサーバ:
1st-MXの障害発生時に2nd-MXが直接配信する、あるいはプレファレンスを一緒にした場合にどうするか、という話です。
前半で、Aレコードを沢山書く場合の負荷分散あるいはバックアップと、MXレコードで同じプレファレンスで沢山書く方式のバックアップ2種類を説明しましたが全く同じに動くわけではなく、前者の場合はその名前に対して1回失敗したらその名前に対しては失敗した情報が記録されてしまいますので、例えばつながって何か通信したが相手のメールサーバのスプールが一杯だったとかロードが高くて拒否されたとか、相手のメールサーバが何か言ってきた場合においてはその名前に対して情報が保存されますので別のAレコードのアドレスに対してトライされなくなります。Aレコードの場合にいいことは、最初のSMTPの接続をするときにホストが落ちていたとか、unreachだった場合に次のアドレスを順番にトライしてくれる点で負荷分散ができることです。
1st-MXと2nd-MXがあって、使用している1st-MXに何か障害が発生し止まってしまったが2nd-MXを経由してメールを受けたい、例えば1st-MX自体にメールボックスがある人はメールを読むことができませんが、内側に中継して行く方式では1st-MXに一旦中継されてそれがさらに組織の内側のメールサーバにメールを中継しますが、そのような構成の場合はさらに

2nd-MXにおいても1st-MXに一旦送ってさらに内側のメールボックスに中継するのではなくて2nd-MXに届いたものもいきなりそこから送って来ることができると少しはフェイルセーフというか、バックアップの機能が効果的になります。

それで、昔わりと行われていた2つの方法があります。基本的に、バックアップをするとき、同じアドレスに対して受理をします。但し、受理をした場合に同じメールボックスを持っていればいいのですが、普通ホストが違うとメールボックスは別れますので、メールボックスに落とすのはどちらかに統一しておいて、1stにメールボックスがあるのであれば2ndはそれが回復するまで待ち、他の計算機がメールボックスだった場合はそこに直接転送するようにします。

どちらの場合もありますが、ACCEPT_ADDRSで両方同じ名前を登録しておいて、メールボックスが片方によっている人は.forwardとかaliasesの設定で記述することができます。

1st-MXと2nd-MXがさらに別のメールボックスのホストに転送することを設定するときに、できるだけ両方で同じ転送先を管理したいので、例えば、aliasesをNISを使って共有できるとよいわけです。但し、すべてのaliasesが全く一緒であればいいのですが、管理者関係のaliasesが微妙に違ったりしますので、最近のsendmailはaliasesを複数使えるようになっています。例えば、ホストに固有のaliasesを書いてあるファイルと両方で共有するファイルを分けます。

メールサーバの上でメーリングリストをやっている場合に問題になるのは、単にばらけさせるだけならいいのですが、メーリングリストのアーカイブを取っているとか、シーケンシャル番号をふっている場合に2つのメールサーバでバックアップ体制にした場合、どちらにアーカイブを残すのか、両方に残すのか、あるいは連番を両方で統一してふるにはどうするか、などの問題がありますが、この辺は未だ解決されていない問題です。

- 1st-MXの障害発生時に代わりに受信
  - 2nd-MX
- 2nd-MX から直接配信
  - aliases を共有
  - 同じアドレスを受理
    - 全てのユーザ
      ACCEPT_ADDRS=
    - 特定のユーザ
      SECONDARY_*=
      - 指定したもの以外は、1st-MX の回復を待つ
- aliases を共有
  - NIS などの共有手段の利用
  - ローカル aliases と共有 aliases に分離
  - OA/etc/aliases, nis: mail.aliases
- ML のバックアップ
  - アーカイブ問題
  - 連番問題

バーチャル・ドメイン:

1台のホストで複数のアドレスを扱いたい、但し、アドレス毎に別のユーザがメールを受け取りたい、つまり同じAという人がA at domain1という人とA at domain2という人を違う人にしたいと

きに、どうするかという話です。

・ ユーザ空間の共有
　ユーザ空間を分ける方法と分けないでやる方法がありますが、共有の場合もdomain1の場合はetc/map1ファイルを参照してその中にある宛先によってさらにforwardする形で分けます。但し、そのホストの中でAがloginする場合はその人しかいませんから、きちっと分割しているわけではありませんが、さらに転送する場合はある程度のことができます。

```
USERTABLE_MAPS='domain1=hash:/etc/map1 ¥
                domain2=hash:/etc/map2'
```

・ ユーザ空間の分離(1)
　さらにきっちり分けたいときは、最近の計算機では1つのインターフェイスに対して複数のIPアドレスをふることができますので、そのアドレス毎に別々のsendmailを起動して振り分けをします。あとは、chrootで環境を分ければ、完全なユーザ環境の分割ができます。

– 1つのホストに複数のIPアドレス
– アドレスごとにsendmailを起動
　O DaemonPortOptions=Address=1.2.3.4
– chroot で環境を分離

・ ユーザ空間の分離(2)
　sendmail.cfで頑張る方法もあり、送られてきたメールアドレスのドメイン部を見て、それによってlocal mailer の部分を切り替えると、popを使ってドメイン毎に別々のpopサーバが起きる、あるいはpopサーバでドメイン名を付けたままのユーザ名として指定すると、それによってスプールを切り換えることができるようにしてあれば、うまくできます。この辺の技術を使って、プロバイダでバーチャルドメインのサービスをしているようです。

– sendmail.cf でがんばる
　・ アドレスごとに local mailer を切り替え
– /etc/passwd とは別のデータベースを利用
– POP 等専用

sendmail.cf のテスト:
　色々なチェック項目がありますが、先ずちゃんと受理するか、別のホストに配信･転送するか、を確認するために-bvオプションを付けた方法と-btオプションを付けた方法があります。これらで、先ず、期待通りに正しく動いているかをチェックして、実際に配信させて、それが完了したら最後に置き換えます。いつでも言えることですが、sendmailもsendmail.cfも古いものは残しておきます。

・ 配信先の簡単なチェック
　– sendmail -bv
・ ルールセットによるアドレス書換のチェック
　– sendmail -bt

- 配信のテスト
- /etc/sendmail.cf の置き換え
  - 古いものは残しておくこと

配信先の簡単なチェック:

bvとbtの2つの方式があります。bv方式はあまり知られていませんが、bvはわりと簡単に解析ができています。DNSで配信する場合はあまりチェックすることがありませんが、静的に経路を設定した場合は、この辺がちゃんと設定した方向のホストに向いたIPアドレスになっているか、などを簡単にチェックできるので有効です。

今使っているsendmailで、新しく作ったsendmail.cfはCで指定するとか、あるいはルート権限無しにチェックしたい場合はqueue directoryでmqueueの場所、ルートが持っているもの以外の場所、例えばtempのdirectoryとか、そいったところに指定しないと実行できないなどエラーがでますので、この辺を注意すればいいです。

- sendmail -bv
  - 一般ユーザでチェックする場合はmqueueの場所を適当に指定(permission)

```
% sendmail -C new.cf -oQ/tmp -bv user@domain
motonori@wide.ad.jp... deliverable: mailer smtp, host wide.ad.jp., user motonori@wide.ad.jp
```

ルールセット・テストのポイント:

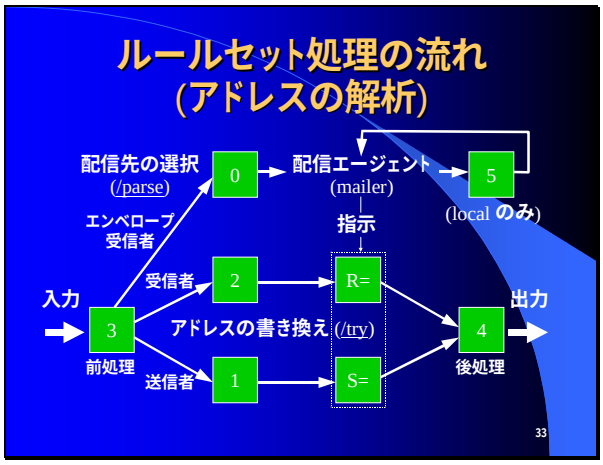
先ず、メールを送るときにリモートへの配信とローカルへの配信がありますが、それぞれちゃんと動いているかを確認します。

リモートへの配信に関しては「3,0」や「/parse」を使用します。その結果が、例えば前出の例のようにSMTPメーラーになっていると別のホストに配信するというわけです。ここがローカルになっていると自分で受理しますので、それぞれのアドレスがリモートに配信されるか受理されるか確認することができます。あとはfromアドレスみたいにドメイン部が自動的に付与されるときに期待通りに正しくなっているかを見ます。この辺をチェックします。

- リモートのメールアドレス
  - 3,0 あるいは /parse、さらに 5
  - smtp 等のリモート配信用 mailer に渡されるか
- ローカルのメールアドレス
  - 3,0 あるいは /parse
  - local mailer に渡されるか (受理の確認)
- 期待するドメイン部が付与されるか
  - /try コマンド (ヘッダ、エンベロープのチェック)

ルールセット処理の流れ:

下図は、sendmail関係の本を見ると出てくる図です。



ルールセット処理の流れ(アドレスの解析)

sendmail.cfを作るとルールセットという用語が出てきます。sendmailのアドレスの処理はルールセットの単位で順番に行われていきます。ルールセットは頭にS1とかS0とか書いているところから始まりますが、それぞれ処理の順番があり、既にヘッダとエンベロープの説明はしましたが、基本的にsendmailの配信処理はエンベロープに対して行われますので、重要な部分は図の上の部分です。

アドレスが入力されてきますと、先ずルールセットの3番で前処理が行われて、それが0番に渡されて配信エージェント、例えば先ほどのSMTPやローカルが選ばれます。ローカルの場合のみ、新しいsendmailからサポートされた5番を通して、さらにSMTPとか他のものにふり直され

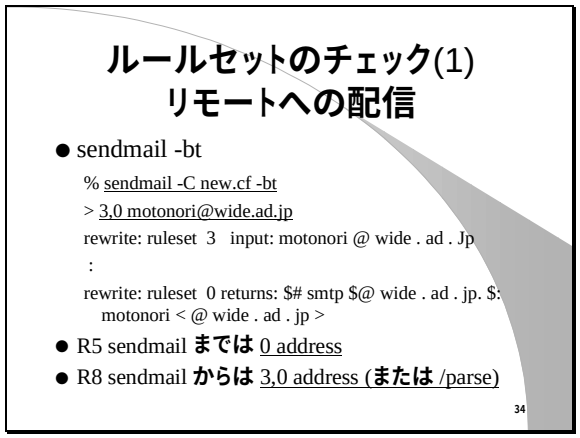
た場合は再度ここに戻ってきます。このように、3番、0番、5番をチェックすることでメールが期待するところに正しく届くか確認できます。

さらに、sendmailはヘッダの書き換えも行うことができますので、その辺の設定、つまり自分の組織の内と外でアドレスの表現を変えたいとか、いろいろな注文がありますが、それらのヘッダに見える(人に見える)アドレスの部分に手を加えたい場合は、3番から2番を通ってmailerに指定されるRのところを通じ、例えばRだから21とかを通って、さらに4番への手順で処理が行われますので、その辺をチェックします。

ルールセットのチェック(1) リモートへの配信

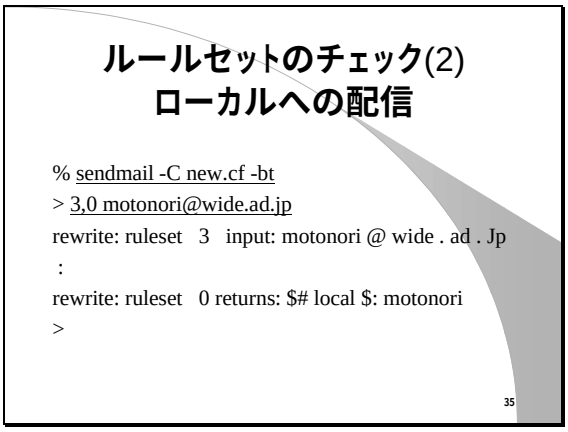
配信先が正しく解釈されているかをチェックしたいときはsenmail –btでチェックできます。その場合は、ルートセット処理の図の3番と0番を使うことになっていますので、3と0を指定して後ろにメールアドレスを書きます。そうするとこれが解釈され、最後に出力が得られます。基本は$#と$@の部分です。この部分に注目して、$#のところがちゃんとSMTPという、自分のホストから別のホストにメールを配信するためのものが選ばれているかということと、その時に配信先として何が選ばれているかをちゃんと確認できればいいわけです。

sendmailのバージョン5(R5)とバージョン8(R8)では、テストのやり方が違いますので注意が必要です。昔は、いきなり0と書いてアドレスを書けばよかったのですが(3は自動的に実行されていた)、R8のときは最初に3を実行することを明示的に指定して実行するようになりました。3,0の代りに/parseを書いてもよいです。



ルールセットのチェック(2) ローカルへの配信

(1)は隣りのホストに転送する場合の出力結果でしたが、これは自分のホストが受理する場合にどうなるかです。3,0の後に自分のアドレスを書いた場合に、最終結果が、$#の後ろがローカルでその先にユーザ名が書かれると、ローカルに届くことが確認できます。



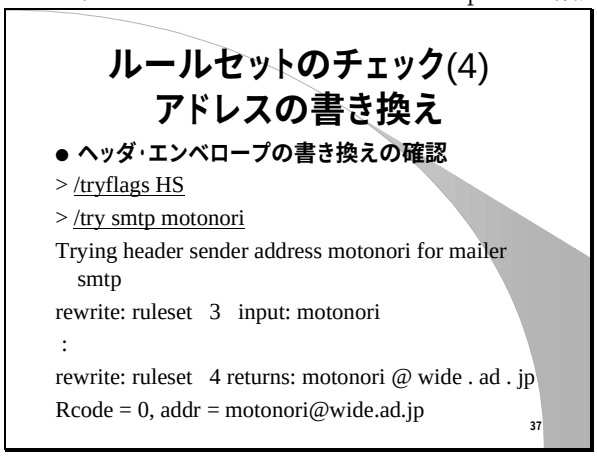
ルールセットのチェック(3) ルールセット5

ローカルに届いた場合、先ほどさらにルールセット5がありましたが、それに対しさらに自分の最後に得られたアドレスを指定することによって、例えばスプールホストに再転送するような設定が行われていた場合は、一旦ローカルになりますが、さらにSMTPでスプールホストに転送することになったりします。ですから、スプールホストに転送するような設定をする場合は、ルールセットの3と0だけではなく、5もチェックして下さい。

ルールセットのチェック(4) アドレスの書き換え

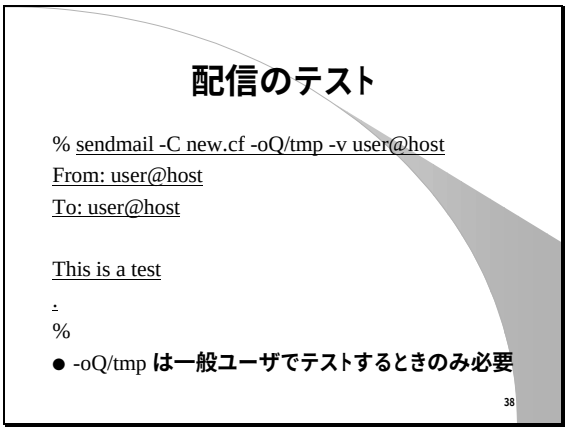
ヘッダのチェックで、ルートセット処理の図の下側です。下側は、自分でどのルールセットが使われるかを確かめて書いていけばいいわけですが、それは面倒なので、最新のsendmailの書き方を書いておきます。ヘッダの処理の方法を確認する場合、ヘッダにおける発信者のチェックをしたいことを指定して、あとでtryで発信者の様子(ここではsmtp motonori)を指定しますと、ずーっとアドレスの書き換えが行われて、最終的に付くアドレスがわかります。

HSはヘッダのsenderで、ERであればエンベロープのrecipientで指定できます。



配信のテスト:

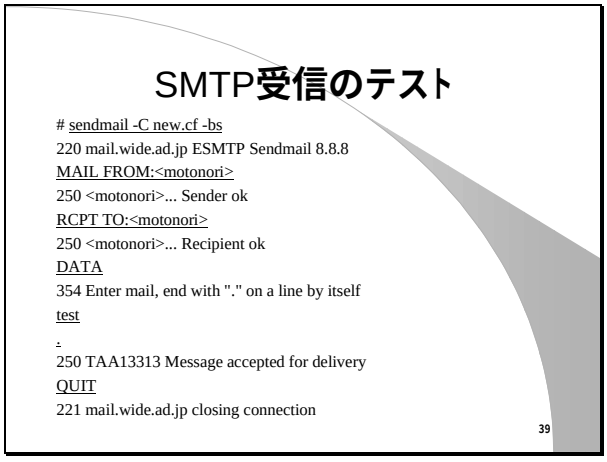
以上の確認が終わったら、実際にメールを送ってみます。



SMTP受信のテスト:

前半の説名で25番ポートをつないで動作確認をする方法を説明しましたが、それ以外に直接コンソールへ動作確認をする方法があります。

こちらは、後ろの-bsがポイントで、これを指定すると端末に対してSMTPをしゃべるモードになります。あとは、SMTPのコマンドを入力して行くと正しく動作するかどうかが確認できます。



sendmail.cf の入れ替え:

前半の説明のsendmailの入れ替えと同じで、sendmail.cfを入れ替えて、あとはSIGHUPを送ります。

あと1つポイントとしては、メールを受信しない、例えばNull Clientのようにスプールを持たない場合はメールが飛んでこないので、メールサーバとしてsendmailを動かしておく必要がないので、bdとしてデーモンとして動きなさいというコマンドを省略してメールの発信のときのmqueueの処理だけを要求するように起動するとよいでしょう。

メールを受信しないホストということでメールを受け取らない設定にはなるわけですが、本当に飛んでこないかどうかは保証できませんので、ネームサーバにおいて、こういう設定をするホスト宛てのメールは別のメールサーバにメールが飛んでいくようなMXの設定をしておく方が確実です。

・ /etc/sendmail.cf にコピー
(もしもの時のために古いものは残しておく)
・ daemon sendmail の再起動
ps aux ¦ grep sendmail
72 ?? .... accepting connections on port 25 (sendmail)
195 ?? .... sendmail: OAA12345 mail.y.co.jp.: user open
kill -HUP 72
・ SIGHUP が利用できるのは R8 sendmail から
・ R5 sendmail は殺して再起動
・ 動作していなかった場合
/usr/lib/sendmail -bd -q1h
・ -bd
– デーモン･モード(受信用)
・ -q1h (1時間) -q30m (30分)
– mqueue に溜まったメールの配信再試行間隔
・ メールを受信しないホスト
– 受信用デーモンは不要
/usr/lib/sendmail -q1h

Sendmail.cf のまとめ:

sendmail.cfができたときに、どういうポイントを重点的にチェックして動作確認をするか、をここで説明しました。

・ どのアドレスを受理するか
・ アドレスごとの配信方法の選択
– 配信先を静的に定義
– ネームサーバ (MX) を参照
・ まずメール･システムのデザインを決める

sendmailを設定するときの注意点をあまり詳しくは説明できていませんが、sendmailのバージョンアップに関しては今ドキュメントを公開できるように書いているところで、そのうち公開できると思いますので、それも参考にして下さい。

6. sendmailシステム管理

sendmailなどのMTAが動き出したという前提で、管理者としては日頃からメールがちゃんと届いているかを管理・監視することが仕事になってきますが、そのときどういうポイントをチェックしていったらよいか、どういうトラブルがよく起こるのか、を重点的に説明して行きます。

メールボックス:

ローカルへの配信に注目した場合に、ローカルへ配信するプログラムがあります。

sendmailが何でもするわけではなく、メールを受け取って送り出すのはSMTPです。インターネットのSMTPに関しては内蔵された機能が入っていますが、それ以外の配信に関しては別のプログラムを呼び出す形で実現しています。UUCPの場合もUUXとかを呼び出します。

ローカルへの配信の場合も、ローカルへ配信するプログラムが既にOSに用意されているので、それを使って配信します。例えばbinmailやmail.localが最近使われています。

この辺の配信プログラムにしろ、UUXなどのUUCPの配信にしろ、設定の方法や使い方はシステムによって違っているので、OSに付いてくるsendmail.cfの該当する部分(local mailer定義)のMlocal行を注目して、その部分に関してはできるだけ一緒になるように注意する必要があります。但し、sendmail.cfのバージョンによってはこの辺の書き方が微妙に変わっており、mailer flagなどの部分に関しては同じにするとまずいことが起こったりしますので注意が必要です。

mail.localに関しては、できるだけ最新のものを使う方が安全です。

何が問題かというと、メールボックスの置いてあるファイルシステムでそこのディスクが溢れた場合、普通は送れなかったということでエラーが返るのではなくて、その溢れが収まるまでしばらく待って、収まったあとでメールをローカルスプールのメールボックスに入れますので、しばらく待つことが必要になります。

正しいローカル配信用のプログラムであれば、TEMPFAILというリターンコード(プログラムの終了コードで75)が返ってくればsendmailはすぐにエラーとしてメールを送り返さずに、一時的なトラブルが収まるのを待って配信を試みます。これらがちゃんとできていることや、あるいはメールを配信するときにtmpのdirectoryを使ったりする、特にSunOSの4.?などの古いものではこのような問題がありますが、tmpが溢れているときにメールを受け取ってしまうと、ちゃんとtmpを経由してメールが欠けて無いのにちゃんと受け取れたような動作になって、メールを失ってしまうことがあります。従って、このような一時的に使うようなところであっても溢れているかどうかをちゃんとチェックすることと、もし溢れていたらしばらく待つように75という終了コードを返すことが重要です。

これらがちゃんとできていない、OSに付いているローカル配信用プログラムがありますから、このような危険性が予知される場合は新しいものに入れ替えた方がよい。

溢れるだけではなく、quotaで制限しているユーザ当たりのメールの容量がありますが、これを何も考えずに設定しておくとローカル配信用のプログラムは同時に複数の宛先に対してメールを送り届けることができるようになっていますので、効率はいいのですがquotaで誰か1人が制限を越えているとメールを配信するプログラムは送れなかったことをsendmailに返事しなくてはいけないので、駄目だよと言うわけですが、そのときに終了コードは1つしか数字が返せないので、同時に複数の相手に対してメールを送った場合にもし他に正常にメールが送れている人があった場合に、そういう終了コード、例えば75という終了コードを返

してしまうとすべての人が問題なく受信できるまで配信が繰り返されますので、ちゃんと届いている人には何通も同じメールが届いてしまうという問題があります。

こういう問題があるといけないので、mailer flagということで、Mlocalの行を見ていきますと、F=というところがあり、その中に小文字のmがあった場合はそのmを外すと、1人毎にlocal mailer(配信プログラム)を起動するようになりますので、誰宛てのメールがエラーになっても正しい人には1回しか届かないようになります。この辺を効率的にするためにLMTPも提案されていますが、未だLMTPを使ったlocal mailerは未だ出てきていません。

- ローカル配信用プログラム
  - binmail, mail.local
  - OS 添付のsendmail.cfを参考にする
    - local mailer定義 (Mlocal行)
  - 最新のmail.localを利用するのが安全
- ディスクの溢れ
  - TEMPFAIL(75)が返ること
  - /tmp/ の溢れも検出できること (使うなら)
- quota制限時の注意
  - 複数同時配信の抑制
    - 一人が制限を越えると他人に何通も届く
    - 誰宛の配信がエラーになったのかが不明なため
      - LMTP (RFC2033(I)) へ
  - mailer flag F=...m... のmをはずす
    - local mailer (Mlocalの行)
    - 一人ごとにmailerを起動

受信の一時拒否:

この受信拒否というのはSpamです。余計な転送ではなく、システムの標準的な問題で、負荷が高いときにさらに負荷を上げないようにするための防衛策などの話です。受信拒否をする項目にはいくつかあり、負荷が高いときにさらに負荷が上がらないようにしばらく待たせます。例えば、O RefuseLA=12という設定がsendmail.cfにあった場合、ロードアベレージが12を超えた場合は受信処理を拒否し、メールを送って来ても受け付けません。新しいsendmailの場合はconnection refusedが返って、すぐにMXとか次のアドレスを試してくれます。

負荷が高いときと並んで禁止したいときは、mqueueが一杯のときです。メールサーバのディスクは普通一杯ありますが、sendmailが管理しているのはmqueueだけです。先ほどのメールボックスとかテンポラリディレクトリとか、そういったものはそれぞれそういったディレクトリを使う別の専用のプログラムが管理すべきで、sendmailはmqueueしか使いませんから、mqueueに関して残り容量が少ないときは受信を拒否する動作になります。

受信を拒否するとき、ここで1つ問題があり、先ほど言いましたようにconnection refusedというような動作をするのが理想なんですが、古いsendmailの場合は負荷が高いとか、ディスクが一杯だった場合にアクセプトせずに、ずっと黙ったまましらばっくれる実装になっています。そうすると、つなぎに来た方はそれが返事するかタイムアウトしてconnectionを切るか

するまで、そのままずっと待っていますから、それがさらに遅延の原因になってしまいます。この辺もできるだけ新しいsendmailにして、何か問題があって受信拒否した場合は次のメールサーバに行ってくれるようにすることが重要なポイントになります。

- 高負荷時
  O RefuseLA=12
- mqueue の空きが少ないとき
  O MinFreeBlocks=100
- 従来の非応答処理
  – accept せずにほったらかし(従来)
    • 遅延の原因になり、よくない
  – socket を close
    • connection refused、すぐに次の MX へ

メールのサイズ制限:

メールのサイズも最近、非常に問題になって来ているトピックです。最近、ユーザインターフェイスが優れたソフトが出てきて、大きな絵とかを、何もサイズを考えずにやり取りする状況になって来ています。そうすると困るのは管理者で、あまり大きなメールは転送させたくない、受け取りたくないというわけです。

そのときチェックする点は、sendmail.cfにO MaxMessageSize=1000000と書いておくと、1メガを超えるサイズは受け取らないようになります。最近のメールは、SMTPを拡張したESMTPというプロトコルが使われていますが、それの賢いところは、今までのSMTPはエンベロープの情報を出して最後にメールの本文を一度に送り、受け取る方は受け取って自分の持っているスプールのサイズを超えたり、ディスクが足りないと思った時点で拒否しますが、EMTPの場合は最初にエンベロープの情報を送るときにこれから送ろうとするメールはこれだけのサイズがありますよ、と渡すようになっています。そのサイズの情報を見て、今手元にどれだけのサイズの残りがある、あるいはここでどれだけの大きさのメールが受け取れないかの制限があるというのを判断して、その大きさのメールが駄目であることをこの時点で言うことができるよになっています。これによって、ネットワーク的に無駄のない、効率の良い転送ができるようになっています。こういう点からもできるだけESMTPをサポートしているMTAを使うようにして下さい。

全体的なサイズの制限に加えて、mailer毎のサイズの制限もあります。SMTPで飛ばすときにはこれだけの制限ですよとか、インターネットに出すときはこれくらいのメッセージの大きさの制限で組織内はこれだけのメッセージサイズの制限ですよとか、あるいはユーザ毎に例えば先生だったらこれくらいのサイズは受け取れますが、学生だったらもうちょっと小さいサイズしか駄目ですよとか、そういった制限をしたり、細かく制限をかけたいときに、mailerのところでM=サイズを書いて制限することができます。

これもよくネットワークニュースで質問が出るのですが、8.8.5くらいのバージョンですとこの辺の設定を書いても一応拒否はされますが、拒否したよとか、メールが大きすぎて送れなかったよ、とかのエラーメールがちゃんと送り返されない問題があったので、これよりも新しいsendmailのバージョンを使うようにして下さい。

- O MaxMessageSize=1000000

– ESMTP で受信前にチェック
  • RFC1870(S)
  • MAIL FROM:<addr> SIZE=1234
・ mailer 定義の M=式
– Msmtp,... M=1000000
– sendmail 8.8.5 (古い!) はうまく機能しない

配信処理効率の向上:

配信効率に関して、前半の中でメーリングリストなど大量に配信するときの配信効率の問題を解決するパッチの紹介をしましたが、それ以外の配信処理効率を向上させる方法・機能がいくつか実装されています。

・ コネクション・キャッシュ

メールが送れずにqueueに沢山保存されていて、それらが同じホスト宛だった場合、送り先のホストが回復したときにqueueに溜まったメールを送る必要がありますが、そのとき(run-queue) に同じ宛先のホストに対しては一本のSMTPコネクションを張ってそこに沢山のメッセージを順番に送り付けることができます。これはSMTPの仕様で、1回SMTPのコネクションを張った場合はその上で何通のメールも送れるとなっていますので、1回1回コネクションを切らずに同じコネクションで送り続けることができるというものです。同じホスト宛てのメールが溜まったときに効果的になります。

・ 配信状況情報の活用

8.8からできたものですが、配信状況情報がハードディスクに保存されるようになりました。これも設定が必要で、PersistentHostStatusという項目にdirectry名を書いて、このdirectryの名前に当たるものをvar/spool/mqueueの中に作っておき、宛先のホスト毎にファイルを作ります。宛先のホスト毎に、いつつなぎに行ったか、そのとき成功したかどうか、5時間前から駄目になっている、などの以前メールを送ったときの情報が記録されますので、今度もう一回同じ宛先に送信しようと思ったときに、送れたと記録されていた場合はそのまま送りに行きますし、例えば20秒前に駄目だったとわかると回復していないと判断してその配信はスキップします。これで、送れなかったところへ何回も試みて無駄な時間を費やすことを回避できるようになりました。

例で20秒といいましたが、その時間は設定ができ、配信失敗から30分以内は再試行しない設定はTimeout.hoststatus 項目に30mと記述します。

・O PersistentHostStatus=.hoststat
– mqueue に情報を保存
mkdir /var/spool/mqueue/.hoststat
  • .hoststat/jp./ac./kyoto-u./ に情報を保存
・O Timeout.hoststatus=30m
– 配信失敗から30分以内は再試行しない

・ 相手ホストに対する考慮

自分のところの問題ではなく相手のところを考慮する場合、例えば、メールの授業をしていて、一斉にメールを送ってもらったとき、40人とか80人で授業をしているとそれだけのメールが同時にメールサーバに届きますが、その一瞬メールサーバの負荷が上昇しますが、そのような場合にうまくメールをさばく方法として、SingleThreadDeliveryオプションを設定しておきますと、同じホストに対して複数のコネクションを張らないで順番に試しに行くような動作になりますので、相手に対して一度に負荷をかけるような配信をしないようにできます。

既に処理中であった場合は、mqueueに保存されてしまうので、この場合はrun-queueの間隔をわりと短くしておいて、残ったものは次のqueueの処理でまとめて送るようにしないといけません。

mqueue 管理:

- 頻繁な処理の抑制

  できるだけ効果的にメールを送ることを考え、HostStatusのように駄目だったところに関してはすぐにはもう一回無駄な試みをしないようにしましたが、それをqueueに関しても適用するものがMinQueueAgeです。MinQueueAge=30mは、mqueueに対して過去30分以内に1回試みて送れなかったものに対しては30分以上経たないと次の試みができないことを定義しています。そうするとrun-queueのプロセスで何回も最初の方に溜まっているメッセージでrun-queueのプログラムが引っ掛かって後ろの方に溜まっているメールがなかなかはけない問題がありますが、そういった問題が回避されます。

- 指定したメッセージだけ配信

  メールサーバのバックアップを作っていた場合、そのバックアップに溜まっているメールをメインのメールサーバが回復したので転送したいときに、その指定した宛先だけにqueueの処理をすることができます。

  指定の方法がいくつかあり、メールを受け取ったときに、その1つ1つのメール毎にidが付きますがそのid(queue-id)を指定してメールをはかせたり、発信者を指定してはかせたり、受信者を指定してはかせるなどの指定が使えます。これはreceiverのsubstringですから、ユーザ名の一部でもいいですし、ドメイン部の一部でも構いません。

  – sendmail -qIqid
  – sendmail -qSsender-substr
  – sendmail -qRrceiver-substr

- ETRN (リモートから sendmail -q)

  メールのrun-queueの処理をリモートから行うための拡張として最近のsendmailに実装されたものです。これを使って、ダイヤルアップのところにpopではなく、ダイヤルアップした自分のところにSMTPでメールを送って欲しいときにメールサーバ側に、つまりダイヤルアップの向こう側にあるサーバに、今つながっているからメールを送るようにSMPT経由で言うことができます。

  – RFC1985(PS): Remote Message Queue Starting

- 処理の順序

  sendmailの管理をしているときに問題になるのが、前述のバックアップの話に関連しますが、queueに溜まっているメールがあったときにそのqueueに溜まっているメールの処理の順

番がぐちゃぐちゃになることがあります。今までのsendmailでデフォルトはpriorityという、例えばメッセージのサイズが大きかったらpriorityが低いとか受信者が沢山いるとpriorityが低いとか古いメールはpriorityが低いとか、そういうことによって処理の順位付けがされて送られて来るので、届いたときには時間順ではないということが気になります。そこで、timeをQueueSortOrderに定義するとpriorityでソートするのではなく、届いた順番でqueueの処理を行う動作に変わりますので、ユーザにとっては時間順に届くので読み易くなります。

・ mqueue の集約

バックアップみたいな感じはなく、組織内に沢山ホストがあった場合にそれぞれが直接インターネットに対してメールを送り出すような仕掛け、firewallをしているところではあまりそのような状況はありませんが、大学とかでそれぞれの計算機がバラバラに動いているのを一括して管理する必要がある場合に、それぞれのホストに送れなかったメールが溜まって行くという状況になりますのでそれが管理上面倒になりますが、そのような場合にFallbackMXhostという定義があり、そこにメールサーバのアドレスを1つ定義しておくとDNSから得られたMXを次々に試みて行って全てのMXに対して送れなかったときにそのFallbackMXhostに対してメールを転送する動作になります。

これを設定しておくと、その組織の中で沢山直接インターネットにメールを送り出す設定になっていた場合にそれらの沢山のホストにメールが溜まる代りに1つの指定したfall.back.hostにメールを集めて一括してそこのqueueを管理すればよくなりますので、管理効率が良くなり楽になります。

Identとfirewall：

IdentはRFC1413(PS)に定義されているIdentify Protocolで、MTAが2つあるときにsenderの方からSMTPでコネクションを張りに行ってメールを送りますが、そのときに相手が誰かを調べる機能が提供されています。

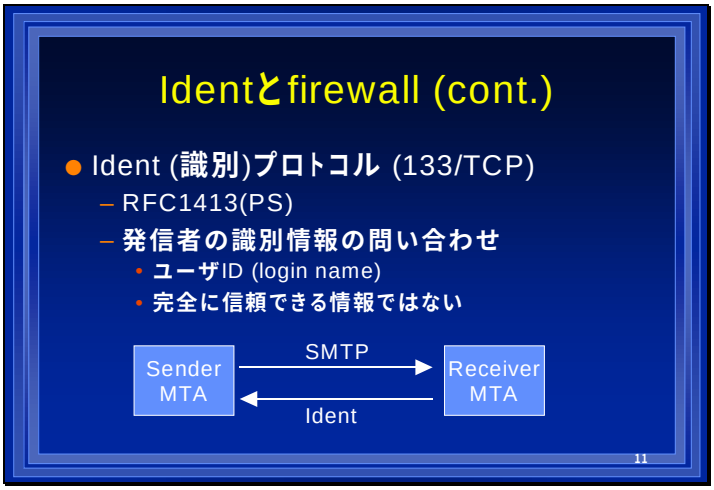
unixの場合であればユーザという概念があって、コネクションを握っているユーザは誰かをコネクション毎に特定できますが、そのコネクション毎のユーザを特定するためにIdentというサービスがあり、そこにつなぎ返しに行って、今SMTPでこういうコネクションが張られて来たがこのコネクションは誰が持っているのかを聞くとそれに対してユーザidを返してくれる機能です。これによって、認証情報が得られ、相手が悪戯でSMTPを張ってメールを送って来たときに誰が悪戯をしているのかがわかるヒントになります。実際にメールを送ろうとするホストがそのIdentの問い合わせに対して誰であるかを答えるわけですから、そのメールサーバが悪戯している人の意のままに動いているとすればIdentの問い合わせに対する返事も信用できませんのでヒントにしかすぎません。

Firewallとからんで、問題が1つあります。MTAの間にfirewallがあり、SMTPは通すが他のプロトコルは通さないというフィルタリングとか、ＳＭＴＰは両方通すがIdentなどは自分から出てつなぎに行く方向しか許さないようなfirewallの設定をしていると、Identのリクエストが来たときにそれはフィルタリングで落とされます。落とすのはいいのですが、host unreachというICMPのレスポンスで返って来るとホストに対するreachabilityが無くなってしまった、つまり今張っているSMTPも同じホストなのでつながらないと思ってそちらも一緒に切ってしまうことが起こります。そうするとせっかくSMTPをつないだのにIdentが返って来て、それが駄目だと言われたときに同時にSMTPも切れてしまうので、SMTPが張れているにもかかわらずメールが送れないという

問題が出ます。

そういうことでIdentとfirewallに関して、メールが送れそうで送れないという問題があるときはこの辺をチェックする必要があります。

対策としては、host unreachではなくport unreachを返すとか、静かに何も言わずに捨てる(時間がかかるという問題は生じます)などの対策が必要になります。



・ フィルタが Ident に対して host unreach
– SMTP も一緒に切れてしまう
→ メールが送れない
→ 静かに捨てる
– タイムアウト待ちが発生するが、届く
→ port unreach を返す

エラー通知:

メールが届かなかったときにそれに対して返事を出すということです。これは宛先が実際に見つからなかった場合に送り返すのと、queueに1回溜めたが長期休業などで送れなかった場合に、この辺を設定しておくとtimeoutして返送されるまでの時間を指定できます。例えば、5dと指定すると5日間我慢してそれでも送れなかった場合は返送します。そのときにメーリングリストなどを運営していて、全部本文付きでメールが返って来るとうっとうしいので、例えばメールのヘッダに付くPreferenceでbulkとかlistとかになっていると本文が返って来ません。さらにエラー通知に対しては、queueには溜まるがそれが送れなかった場合にそれをさらに返送することは、発信者がnullなのでできません。

実際、送れなかったので諦めたというのが先ほどの話ですが、その他に、送れないみたいだよ、あるいは、実際に送り返すのではなくもう少し試みるのでもう少し待ってね、と教えてあげる機能も新しいsendmailには入りました。その設定はqueuewarnというwarningのところで、例えば4時間queueに溜まっていたらそういう通知を出して、ということを設定しておくとそれによって返事が返ります。

sendmail.cfでは、負のPreferenceはPで始まる行で定義されています。

mqueue に入ってからの経過時間に基づく

- 返送するまでの時間
  - O Timeout.queuereturn=5d
    - 長期休業の際に注意
      - Lower-MX での設定
    - Preference: が負なら本文を返さない
    - エラー通知(発信者が<>)に返送しない
- 未配信(まだ届いていないこと)を通知
  - O Timeout.queuewarn=4h
    - Preference: が負なら通知しない
    - エラー通知(発信者が<>)に通知しない
- 負のPreference:
  - list, bulk, junk
  - sendmail.cfのP行で定義
    - Plist = -30

エラーのモニタ:

いろいろなエラーが発生しますが、それが自分が管理しているシステムにとって致命的なものかどうか、致命的なエラーが発生した場合に管理者としてできるだけ早く気が付きたいので、そういう情報を送ってもらうことを考える必要があります。

sendmailの場合には、2つのエラーがあります。先ず、普通にユーザがメールを送ったが相手先が間違っていたとか、ユーザがいなかったとかでエラーになる場合がありますが、そのエラーを通知する機能があります。PostMasterCopyという定義に、例えばpostmasterと書いておくと、エラーが発生する原因となったメールのヘッダ部分がpostmasterに送られてきます。昔はメール全体が送られてきたのでプライバシー的に非常に問題がありましたが、今はヘッダの部分だけがpostmasterに届くようになっています。但し、subjectがヘッダの一部として入っている情報なので、そこに他人に見られて困る情報が入っているとそれがpostmasterに渡ってしまうので、この辺は注意する必要があります。

さらに、エラーが発生して送り返そうとしたが、元のメールの発信者のアドレスが間違っていた場合はダブルエラーという形になりますが、その場合のエラーの送り先はDoubleBounceAddressで定義します。これについては、本文も含め全て丸ごと返って来ます。発信者の設定ミスが悪いという考えもありますが、これも本文全部が返って来ることは問題であるとも考えられますので、ヘッダだけにする必要があるかもしれません。

- エラー発生時にpostmasterにも通知
  - O PostMasterCopy=postmaster
    - ヘッダ部分のみ
      - Subject: は見えてしまうので注意
- ダブル・エラー
  - 発信者への返送不能
  - O DoubleBounceAddress=postmaster
    - 本文を含めて通知

- オプションにすべき(?)

Return-Receipt-To:の現在:

sendmailのバージョンのところで出てきましたが、現在のReturn-Receipt-Toは既に廃止の方向に向かっています。このような自動配信に関する制御はヘッダに書くべきでなくエンベロープの仕事だという意見があり、Delivery Status NotificationというRFCでヘッダではなくエンベロープでその機能を実現することが提案されて、sendmail8.7からはその方式でReturn-Receipt-Toに相当する機能が実装されています。

具体的には、エンベロープのところにアドレスがくるわけですが、宛先のところにさらにNOTIFYというオプションを付けて=successと書くと、配信が成功したときに返事が来ることになります。ここにはいくつか種類があって、failureと書くと普通の配信が失敗したときに返事が来ますし、メーリングリストとかでエラーになっても返事がいらない場合はneverと書くとエラーがあっても返事が来ない動作になります。

Return-Receipt-Toのときは返事が欲しい人のアドレスを書きましたが、エンベロープでDSNで実装された場合はエンベロープの発信者に返事が返って来ます。

実際にエンベロープにこの情報を入れるかといいますと、sendmailを直接起動して使う場合はsendmail -N successと書くとDSNにこの情報が入って配信が成功したときに結果が返って来ます。問題は、sendmail8.7以降とそれ以前のものが混在していた場合にどうするかですが、現時点ではReturn-Receipt-Toの付いたメールが8.7に渡された瞬間にReturn-Receipt-Toの情報は無視され、それがDSNに変換されることも今のところ行われていませんので、古いものから新しいsendmailへ行く場合はReturn-Receipt-Toが書いてあったとしても途中で8.7になってしまうとその情報は途切れてしまいます。8.7以降のsendmailが並んでいればDNSは正しく伝わりますが、途中に古いsendmailがあるとそこで途切れてしまいます。その場合、途切れた瞬間に、途切れる次の古いsendmailが次にあることがわかった場合は、それ以上はDSNの機能は伝わらないので、ここまではちゃんと届いていたことを返事することは行われるようになっています。

セキュリティ的に、Return-Receipt-Toのようにどこに最終的に届いたかという情報が返って来るとまずいという話もあり、プライバシー的に問題があるだろうということで送り返さない設定ができるようになっていますが、この辺を注意していないと知られたくない情報が勝手に拾われてしまうことが起こりえます。

- RFC1891(PS): DSN (Delivery Status Notification)
  - sendmail 8.7 以降
  - 古い sendmail が途中にあるとダメ
- Return-Receipt-To: ヘッダの廃止
- エンベロープで指定
  - RCPT TO:<addr> NOTIFY=success
- エンベロープの発信者に通知
- sendmail -N success

プライバシーを守る:

firewallの技術と並んで、内側の自分の身近な情報はできるだけ外には隠しておきたいとい

う風潮に変わりつつあります。そのためのオプションとしてPrivacyOptionsがあります。

デフォールトはauthwarningsとなっており、不審な操作をするとX-Authentication-Warningsというヘッダが付いて、このような動作をしてこのメールが発信されましたと、例えば-bsというオプションでメールが発信されましたとか、HELOのところで実際名乗った名前が登録された名前と違ったとか、をヘッダとして残してくれます。

noexpnやnovrfyは、SMTPを張ってこの人のフォワード先がどこかをSMTPのEXPNで確認できますが、そういう操作を禁止する設定です。

外から確認するものはそういうものですが、内側に対して、例えば誰がどういうメールを送ったのか知られたくない、沢山のユーザが1つの計算機を使っているが互いのプライバシーを守りたい場合にメールキューに残っているメールの宛先関係を見ると誰が誰にメールを出してやり取りしているかがわかりますが、そういうmailqコマンドを信頼できるユーザだけに制限するとか、Return-Receiptを抑制(送り返さない)する設定があります。

- O PrivacyOptions= (, で区切って複数指定)
  - authwarnings
    - 不審な挙動に X-Authentication-Warnings: を付与
  - noexpn, novrfy
    - EXPN (転送先の確認) の禁止
  - restrictmailq
    - 一般ユーザの mailq コマンド利用規制
  - noreceipts
    - Return-Receipt の抑制

MIME形式の利用:

時代はMIMEということで、エラー通知に関してもMIMEでエラーが返って来るようになっています。

- MIME 形式のエラー通知
  O SendMimeErrors=True (MIME形式にするかの設定)
- 7Bitしか通らないところへの配信のため(普通の配信に関して8ビット目の立っているメールの扱いの設定)
  O SevenBitInput=False
  O EightBitMode=pass8
    - 8Bit → 7Bit (base64/quoted-printable encode)
    - Content-Type: unknown-8bit がついてしまう
- 7Bit → 8Bit デコード
  - mailer flag F=9 の指定の場合

配信統計:

どのくらいの数のメールが流れているかを入手する方法にはいくつかあり、syslogやmailstatsから拾うことができます。mailstatusのクリアの特別のコマンドはなく、単にnullのファイルをコピーしてクリアします。

・ syslogから収集
・ mailstatsコマンド
O StatusFile=/var/log/sendmail.st

M	msgsfr	bytes_from	msgsto	bytes_to	Mailer
3	1308	6475K	4072	14975K	local
4	3450	15797K	1819	5681K	smtp
T	4758	22272K	5891	20656K	

・ 情報のクリア
– cp /dev/null sendmail.st

ありがちな落とし穴:

メールサーバを管理しているとき、そのメールサーバと他のメールソフトあるいはメールクライアントがあったときに、その間の関係でうまくメールがやり取りできないことがいくつかあります。

・ 2行greetingに対応していない相手

sendmail 8.6の頃は、SMTPで接続したときに、2行のgreetingの行が出てくるようになっていました。そうするとメールのクライアントが1行だけしか返ってこないと思って実装されているとSMTPがうまく対応できないということになりますので、このような場合はsendmailを新しくする、あるいは8.6で1行だけしか返事をしないように手を加えることが必要になります。

– sendmail 8.6までの問題

220-mail.x.co.jp Sendmail 8.6.13
220 ESMPT spoken here

・ SMTPコネクションを再利用できない

SMTPのコネクションは基本的に複数のメッセージが1つのコネクションで送ることができるようになっているはずなんですが、そういう形でメールを受け付けてくれない相手も中にはありますので、その場合はコネクションキャッシュサイズを0にすることで回避することができます。

・ アドレスにコメントがついていると受け取れない

パソコン系のメールサーバに対してメールを転送するときに、アドレスにコメント部分が付いているとうまく受け取れない問題があることがあり、その場合はmailer flag、すなわちMSMTPで始まる行のFのところにCを足します。

– mailer flag に F=C を足す

・ 同時に2人以上の宛先で受信できない

同時にSMTPで複数の宛先に対してメールを送ろうとしても受け取ってくれない場合は、SMTPの部分に関してもF=mを削除することが必要になります。

– mailer flag から F=m を削除

・古い(?) TIS FWTKのsmapd

TIS FWTKを使っていろいろ設定していたときに、これはメールの中継をしてくれるのですが、基本的にSMTPのメールのやり取りにおいて5で始まるのはパーマネントなエラー、つまり回復不能なエラーですからキューに保存するのではなくその場でエラーとしてメッセージを送り返す必要があるのにもかかわらずキューに保存する動作をするので、エラーメールが何通も来るという問題がありました。

日常管理:

以下のようなものがあります。

- アドレスの廃止
- NFS関連
- 停電からの回復
- システムリプレース

アドレスの廃止:

アドレスがなくなる人がいたときに、どういう風にどこまで面倒を見てあげるかという話で、単に消したり、単に転送するだけではなく、この人はいなくなったのでこのアドレスに送り直して下さいといったアドレスの変更通知を行う機能も用意されています。

- 異動/転勤/卒業/除籍
  - 単に削除
  - アドレス変更通知
    oldaddr: newaddr.redirect (/etc/aliases)
    エラーメッセージ: User has moved; please try <newaddr>
  - 新アドレスに転送
  - 通知つき転送
    oldaddr: newaddr.forward, newaddr
    エラーメッセージ: User has moved; your mail has been forwarded; next time please try to <newaddr>

NFS はトラブルのもと:

NFSをメールサーバで使わないように設計しましょうという話です。

例えばメールボックスの共有をしますと、lockがNFSを介して不完全ですから、新しいメールが追加される瞬間にメールを読んでいたりしますとメールを失う可能性があります。また、forwardをチェックするときにautomountをしているとmountに時間が掛かってなかなかメールが送れないなどの問題があります。forwardするときのプログラムがNFS先であると、もしmountに失敗していたらプログラムが存在しないといってメールを送れないという問題もあります。設定ファイルに関してもNFSの向こうに関してパーミッションのチェックが完全にできないので、セキュリティ的に甘いということでうまく動作しないことがあります。

このようにNFSを考えていては駄目であるという状況で、設計する必要があります。

- メールボックス
  - lock の不完全問題
    - メールを失う可能性
- ホーム・ディレクトリ
  - mountに時間がかかる(automount)
  - .forwardの無視(マウントに失敗したとき)
    - .forwardの置き場所をローカルに

O ForwardPath=$z/.forward:/var/forward/$u/.forward

- プログラム共有
  - プログラムが見つからずエラーで返送
    - マウントに失敗したとき
      - 停電後の起動の順序問題
    - vacation, slocal,...
- 設定ファイル
  - セキュリティチェック
  - ファイルの所有者の権限で実行されない
    - :include:

停電からの回復:

日頃の管理の1つとして停電を考えた場合に、停電から回復するときにどのような順番で回復させるかという話しです。メールサーバに関しては、必ずDNSすなわちネームサーバを復活させてから、メールサーバを復活させます。マウントの順番をちゃんと考えて行います。

- NFSマウントの順番
- ネームサーバを先に
  - メールサーバは後
  - メールサーバで named を動かす
    - Secondary にする
  - 自分の FQDN が引けない
    - アドレスの認識を誤る恐れがある
    - sendmail が常駐しない (メールが届かない)

システムリプレース(OSのバージョンアップ):

システムのリプレースをするとき注意することがあります。デフォールトでsendmailが動くように設定されているとOSのバージョンアップをするときに、使っていたsendmail.cfを残しておくことは重要ですし、OSをインストールして立ち上がったときメールが届いてアドレスの設定が正しくないのでメールが落ちてしまうことを避けるためにネットワークを外しておき、sendmailが正しくなってからネットワークにつなぐ、などをきちんとチェックする必要があります。

- 未設定のsendmailが動くと悲惨
  回復後に届いたメールがエラーになる！
  - 使っていたsendmail.cfを残しておく
  - 未処理メールの確認 (mqueue)
    - run-queueプロセスが動くとまずい
  - ネットワークをはずしておく
  - sendmail を殺してから接続
    - リブートの場合は /etc/rc から起動されないように

メールサーバとセキュリティ:

セキュリティに関しては最近はドキュメントが充実してきていますので十分にチェックして設定して下さい。

- CERT report をチェック (www.cert.org)
- 最新の sendmail にする
- パッケージの PGP シグネチャの確認
  - トロイの木馬対策
- 各種ファイルのパーミッション
- SafeFileEnvironment option の活用
- smrsh の利用
- 発信者認証
  - POP サーバアクセス権による確認
  - XTND XMIT (POP) を利用したメールの発信
  - identd
- ヘッダから internal address を隠す
- 無用な転送の防止
  - メール爆撃、スパムの踏み台

7. DNSの仕組みと管理

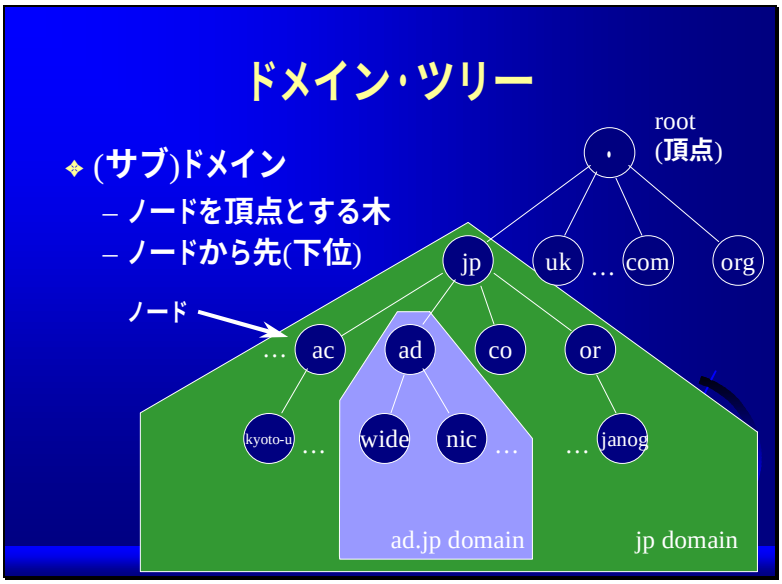
DNS (Domain Name System)：DNSは広域分散データベースでホスト名とIPアドレスに関するの対応関係を広域で保持します。

- 広域分散データベース
- ホスト名とIPアドレスの対応表
- 自律分散管理

ドメイン・ツリー：

メールアドレスでも出てきますが、wide.ad.jpとかnic.ad.jpとかjanog.or.jpとかありますが、そのようなメールアドレスなどで使うものはピリオドで区切っていますが、このピリオドのところを単位として分解するとツリー構造にマップすることができます。下図のような頂点をrootとしたツリー構造をドメイン・ツリーといいます。rootからツリー構造を見たときはインターネットのドメイン全部になりますが、その1つ下のjpから見たときにそのツリーをjpドメインといい、さらにその下のadを頂点としたツリーを見たときにad.jpドメインといいます。このようにそれぞれのツリー構造のノードを見たときにそこから下の部分に関してそれぞれのドメインが存在します。

ネームサーバの管理は、基本的にこのようなドメイン構造、あるノードに対してそれから下の部分を見たときに存在するものに対して管理を行われていると考えるのが素直ですが、実際は少しズレがあります。



分散管理と検索:

必要に応じてノード間の上下リンクで分割します。つまり、ドメイン・ツリーの図のように全てのノード間が全部切れているわけではなく、ある程度管理のし易さ、あるいは管理の対象の組織の単位と実際のドメインの構造、分割の単位によってノードの上下関係を切るところと切らないところが出てきます。

切ったところは下に対してDelegationが発生します。例えば、rootはinterNIC的に全世界でルートサーバが13とかありますが、それらが管理していて、次にjpドメインはJPNICが管理しているとか、adもそうですが、さらにwideとなるとwideプロジェクトが管理しているという形で、ドメインをドットで区切ったところを伸ばして行くとそれぞれに対して、管理するところが変わります。管理するところがネームサーバを持っています。

検索を考えたときに、ドメイン・ツリーの図で、kyoto-uドメイン辺りにいる人がwide.ad.jpの情報を知りたいと思った場合は、そこから隣りへ行く方法はありませんから、検索をするときは基本的に全てのノードはrootだけ知っているというところから始まります。そこで、wide.ad.jpの情報を知りたい場合は先ずrootに聞きに行って、次にjpがどこにあるのかを聞きjpが見つかると、次にadがどこにあるのかを聞きに行くという形で、rootから順番にたどっていく形で検索します。

リンクは上から下の方向へはつながっていますが、逆に下から上の方向にはつながっていませんので、逆方向への検索はできない構造になっています。

- 必要に応じてノード間の上下リンクで分割
  - ノードの下流へのリンク
    - Delegation(権限委譲)
  - TOP domain, 2nd(3rd)-level domain
    - NIC が管理
- 単方向リンク(上から下へ)
  - 上位へはrootまで戻ってから辿る
  - 全サーバはrootを知っている

ゾーンとドメイン:

ドメインとゾーンは微妙に捉え方が違います。ネームサーバでデータを管理するときはドメインではなくゾーンを管理します。ゾーンは下図で緑で囲んだ単位です。rootゾーンはrootの直下にあるnet、jp、com、orgなどのトップレベルのネームサーバがどこにあるのかを管理しています。管理しているというのは、例えばjpという情報の問い合わせが来たら、そのネームサーバのありかをポインタで指定して返事を返します。そのときjpはここにあるよ、と上のrootが指定することによって、このjpというのはそのjpに関する権限を持ちます。上からの矢印は、上のゾーンから指されていることにネームサーバの管理に関しては大きな意味を持ち、その矢印がdelegationを示しています。jpに関して見ると、jpはadに対してそのネームサーバを指すポインタを持っており、それによってadはjpから権限を委譲されています。さらにwideというネームサーバに対してadからdelegationの矢印があり、それによってwideのネームサーバは権威付けされています。

このように、ドメイン・ツリーがあるノードを頂点とした枠であるのに対し、ゾーンはあるノード

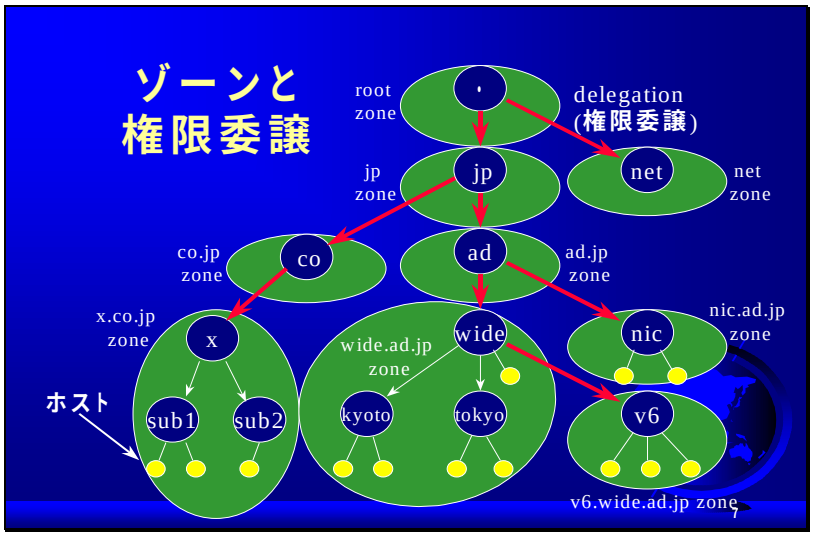
だけが含まれるものであったり、あるノードから下の全部が含まれるもので、管理する対象で分けられます。ネームサーバでデータベースを管理するとき、緑の単位であるゾーンでデータを管理します。1つの組織の中に全部すっぽり入っている場合もあれば、一部の部分だけがさらに別のネームサーバにdelegationされている場合もあります。(図の黄色の丸はホストを表す)

ネームサーバ的には、ホストは単にAレコードが付いているだけの意味しか持ちません。

ここでは、ドメインとゾーンの捉え方が違うことを理解して下さい。また、上から指されていることが重要であること、ネームサーバの定義といった場合にNSレコードに書かれていることではなく、上のゾーンにNSレコードが書かれていることに意味があります。

- 必ずしもノード単位で分割管理の必要なし
- ゾーン
  - 共同管理される隣接ノードの集合
  - 必ずしもドメインとは一致しない
    - 1ゾーンで複数ドメインを管理
- データの管理単位
  - 部所単位/地域単位に分割
  - 1つのネームサーバに対応
    - 末端ではドメインと一致

ゾーンと権限委譲:



サーバの種類:

サービスの種類としては、ゾーンを管理していてデータをインターネット上の他のホストに提供する機能を持っているもの、ゾーンを持たずに単に検索専用のために置かれているキャッシュサーバの機能のものの分け方、あるいはゾーンの管理に関してはプライマリとセカンダリ、すなわち、そこで編集して直接管理しているものとバックアップ的にプライマリからもらって管理しているものの分け方として2種類があります。

delegationに関しては、上から矢印が付いていたauthorizedされているものと矢印の付いていないものというわけ方があります。

サービス対象として、組織外向けと組織内向けの分け方があります。例えば、データ提供用で、セカンダリで、unauthorizedであるとか、それぞれの項目に対して1種類の性質しか持ちません。

- サービスの種類
  - データ提供用 (検索もする) / 検索専用
- データ(ゾーン)の管理
  - そこで編集 / 他からコピー
- 権限
  - Authorized / Unauthorized
- サービス対象
  - 組織外向け / 組織内向け

提供するデータ(ゾーン)の管理:

プライマリとセカンダリがあり、マスターとスレーブという分け方をするとNISに近くなるのでわかり易いですが、プライマリの方はデータベースそのものを直接握っていてそこで実際に管理者が編集作業をするサーバです。セカンダリは、プライマリサーバからデータを拾ってきてコピーを持ちます。場合によっては、セカンダリのセカンダリもできます。また、フェイルセーフの

ためにコピー元のサーバを複数指定することもできます。

複数用意するのはバックアップのためで、プライマリがこけているときにセカンダリがサービスをします。但し、外から見ただけでは、どちらがプライマリかセカンダリかは判断付きませんので、そういうふうな区別、つまりプライマリがこけたらセカンダリにサービスの先が移るという仕掛けにはなっておらず、どちらも同じように使われます(実際、使われるかどうかは、次のauthorizedかどうかという話になります)。プライマリ・セカンダリの区別無しに、検索要求は平等に来ます。

あるサーバがプライマリ、あるサーバがセカンダリというふうに、サーバ毎に特定できるものではなく、例えばwideに対してはプライマリだが、kyoto-uにたいしてはセカンダリなど、ゾーン毎にそのネームサーバがプライマリかセカンダリかが別れますので、サーバ個体に対する区別ではなくゾーンに対してそのネームサーバがどういう役割を果たしているかという分類になります。

- プライマリ(マスタ)・サーバ
  - データベース・ファイルの編集を行なう
- セカンダリ(スレーブ)・サーバ
  - プライマリ・サーバからデータをコピー
    - 別のセカンダリからでも可
      - コピーチェイン
    - コピー元サーバを複数指定可能
  - プライマリのサービス・バックアップ
  - 同時に到達不能にならない場所に配置
- 検索要求は平等に来る
  - プライマリ・セカンダリの区別はない
- ゾーンに対する区別
  - 一つのサーバで複数のゾーンを管理
    - ゾーンAに対してはプライマリ
    - ゾーンBに対してはセカンダリ
  - サーバ個体に対する区別ではない

データの提供に関する権限:

データの提供に関する権限でAuthorizedとUnauthorizedという分け方があります。これはゾーンの図の赤い矢印と非常に関係深いですが、Authorizedは上から権限を委譲されている、つまり上から順番に検索してたどってくるとこのサーバにたどり着くというのがAuthorized serverです。

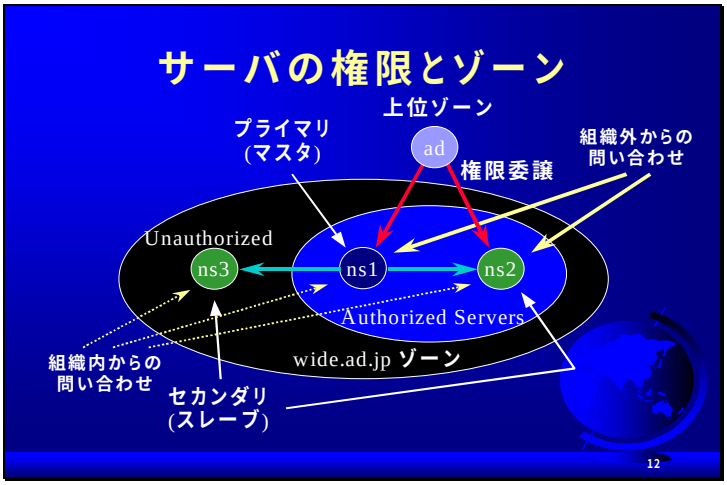
Unauthorized Serverというのは、セカンダリみたいにデータは持っているがドメイン・ツリーのrootからたどって行ってもそのサーバにはたどり着かないサーバです。これは、手元で恒常的にキャッシュをしておきたいとか、データを隣クライアントに提供するために用意されています。管理上の都合で必要になったりもします。

プライマリとセカンダリの区別と同じように、あるゾーンに対してはAuthorizedだが、あるゾーンに対してはUnauthorized、つまりあるゾーンに関して上から検索をして行くとそのサーバにたどり着きますが、別のゾーンに関して検索をして行くとそのサーバにたどり着かないという関

係があります。

・ Authorized Server
– データをインターネットに提供
– 上位ゾーンからのリンク(権限委譲)がある
・ Unauthorized Server
– 手元の恒常的キャッシュ
– データを近隣クライアントに提供
– 上位ゾーンからのリンク(権限委譲)がない
・ ゾーンに対する区別

プライマリのネームサーバがwide.ad.jpというゾーンに対して用意されており、そのゾーンを管理するサーバ(ns1,ns2,ns3)を3つ用意し、そのときns1がプライマリサーバ、ns2及びns3がセカンダリサーバとします。つまり、ns1で人がデータベースを編集し、そのコピーを自動的にns2やns3に送り、データの管理が行われます(下図参照)。



そういった管理の関係に対して、権限の関係があり、上位のゾーン、つまりadゾーンがあったときにこのadゾーンの中でwideというゾーンを管理しているネームサーバはこれとこれだと書いてあるとするとそれが権限委譲となり、上からドメイン・ツリーを検索して来るとその検索の結果ns1やns2は外から検索して行くとたどり着くことができますが、ns3は外から見て来てもadのところから矢印が付いていませんから外から検索してきてもたどり着きません。権限を持っているネームサーバはns1とns2だけです。青の楕円がAuthorized Serverで、それに含まれないものがUnauthorized Serverになり、これはキャッシュをしていて組織内から問い合わせをしたときにどれを見に行っても組織内の情報をすぐ返してくれます。

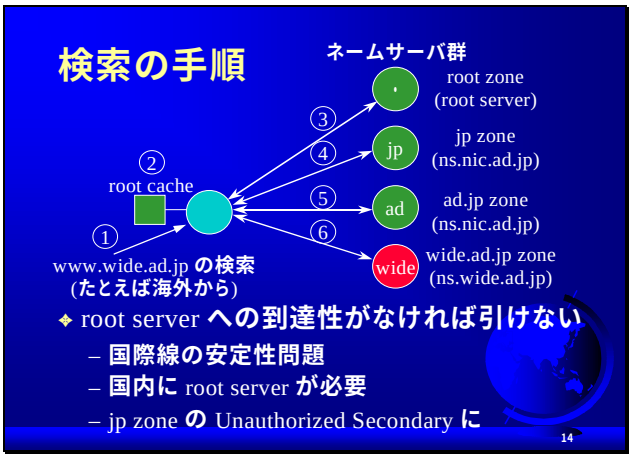
このように、ゾーンの管理関係と上からの権限関係は別の概念で合わさっていることをネームサーバを管理する場合は押さえておいて下さい。

検索専用:

ゾーンを全く持たない検索だけのキャッシュサーバがあります。該当するレコードが存在しなかった場合にそれを記憶することができるようになっています。存在しないアドレスを何回も引きに行った場合に、最初は時間が掛かるが、2回目以降はすぐに無いことがわかります。(ネガティブ・キャッシュ)

- キャッシュサーバ
  - 一度検索したデータをしばらく記憶
    - Unauthoritative Answer として応答
  - プライマリでもセカンダリでもない
    - どのゾーンに対しても
- 参考:
  - 該当レコードが存在しなかったことを保持(全サーバ)

検索の手順:



あるネームサーバに対してクライアントから問い合わせ(www.wide.ad.jpの検索)が来た場合、このネームサーバはドメイン・ツリーをたどりながら目的に達します。

ネームサーバの設定時にroot cacheという情報ファイルにroor serverがどこにあるかをあらかじめヒントとして与えられていますので、全てのネームサーバはrootを知っており、先ずこのroot serverに聞きに行き、jpをどれが持っているかを教えてもらい、次そこでad.jpがどこにあるかを聞いてそれを教えてもらって、次にそこでwide. ad.jpがどこにあるかを聞いてそれを教えてもらって、最後にwide. ad.jpに対してwwwが何かを聞くと、アドレスやCNAMEを教えてくれます。このように常に上から順番にたどって行きます。

1回引いた情報はキャッシュされますので、2回目からは、www.nic.ad.jpを引きに行くと、ad.jpは1回引いてわかっていますので、次の検索はいきなりnic.ad.jpに聞きに行くところから始められます。最初はrootから検索を始めますが、キャッシュによって1回引いたところは再利用されます。

root serverの問題はいつもあり、何か検索を始めようとするとrootから始めないといけないという問題がありますので、例えば日本において昔は国際線が不安定でWIDEの線が切れると海外が見えなくなり、root serverが見えないので国内も一緒に引きずられて見えなくなりまし

た。最近は、トラフィックの問題もあり、ようやく国内にroot serverとしてm(root serverはアルファベット1文字でaからnくらいまで順番に付いている)ができました。

DNS Servers:

Berkeley Internet Name Domain(BIND)が広く使われていますが、bind 4.9.6やbind 8.1.1が最新版です。ネームサーバに関してもセキュリティ的な問題が最近報告されていますので、そういった点からもできるだけ最新版を使いましょう。resolverに関しても同様に最新版を使いましょう。

Windows NTに関してもネームサーバが付いていますので、それを使うこともできますが、概念的なところは同じですので、あとは信頼性の問題です。Windows NTに最初から付いているものは、あまり信頼性がないといわれており、BINDで移植されているものがありますので、それを使った方が良いでしょう。

サーバの設定ファイル:

bind8になって、named.bootという最初の設定ファイルがnamed.confに変わりましたが、書き方がdrasticに変わっており躊躇することもありますがbind8の中にnamed-bootconf.plというフォーマット変換ツールが入っていますので、それを通すとbind4の形式がbind8の形式のものに変換できます。

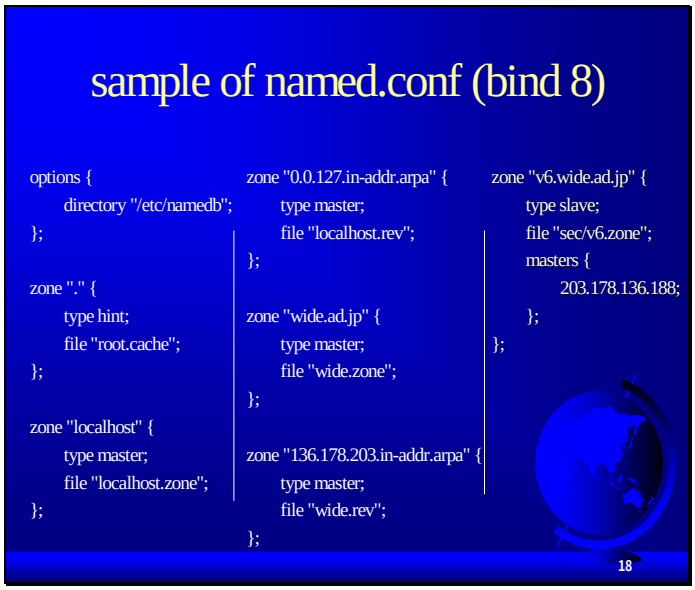
重要なポイントは、bindでファイルを設定するときはセミコロン「;」 がコメントの開始として利用されますので、それを覚えておいて下さい。

sample of named.boot (bind 4):

bind4のnamed.bootを見ると、directryの場所やroot.cacheファイルにルートサーバ情報の入っているファイル、自分がプライマリファイルとして管理している情報やファイルの場所、セカンダリとして他のホストから拾って来るゾーンやそれはどのサーバから拾って来るか、などが書い

てあります。

sample of named.conf (bind 8):



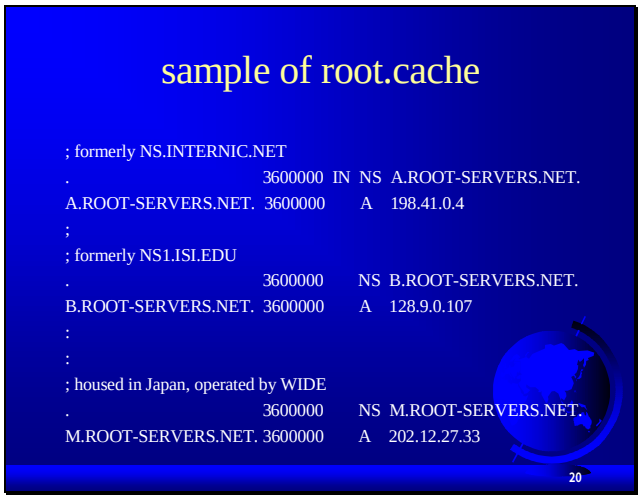
bind8のnamed.confはこのように書かれ、gatedに近いものです。

root cache:

全世界のインターネット上のroot serverに関する情報が列挙されているものです。最新版はこのURL(ftp://ftp.rs.internic.net/domain/named.root)から拾って来ることができますが、13番目のroot serverとして日本で1997年8月に稼動を開始しました。

インターネット的なroot serverはこのようにして拾って来ますが、firewallの内側で外の世界から隔絶した形で別のＤＮＳの空間を用意する場合は、組織の内側向けのroot serverを用意してそれが書かれたroot cacheを用意しそこからdelegationされたドメイン構造を作る必要があります。

sample of root.cache：



root.casheの内容はこのような感じで書いてあり、Mのところはwideプロジェクトがやっているというコメントと共にアドレス情報が書かれています。

数値の単位は秒です。cache はヒントですから、本当の情報はroot serverに1回問い合わせて正式なものへとアップデイトされます。

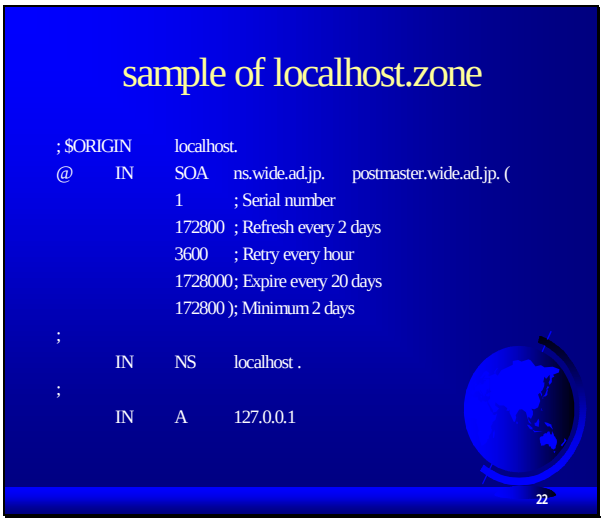
forwarders：

firewallを考えた場合に、forwardersを使って組織内からも組織外のネームサーバが引けるようにすることができます。例えば、socks対応のfirewallを使ったときにこのような方式を利用することがあります。そのような場合にslaveと共にforwardersを指定します。forwardersは自分が知らない情報はforwardersに指定したサーバに聞きに行くことを指定しますが、もしforwardersが指定したサーバが知らなかった場合は自分で引きに行ってしまいますが、自分がインターネットにreachableではないので問い合わせが失敗しますので、必ずforwardersを通して聞くようにするためにslaveを書く必要があります。

forwardersに指定するのは、組織の外からも内からもアクセス可能なサーバを指定します。forwardersの使い方として、昔、回線が細かったときに1回聞いたデータを別のサーバが再度聞きに行くのは無駄ですので、webのプロキシと同様にキャッシュを有効利用し、ネームサーバの場合と同じような構成を取ることによってネームサーバのトラフィックを押さえることができます。

- 組織内から外部のアドレスの問い合わせ
  - 外部のネームサーバに問い合わせを転送
    - socks 対応 firewall などの場合
  - slave とともに指定
    forwarders 12.34.56.79 (内外両側からアクセス可能なサーバ)
    slave (options forward-only - 4.9.3 or later)
- キャッシュの有効利用
  - データを特定のサーバに集約
  - トラフィックを抑える(回線が細いときなど)

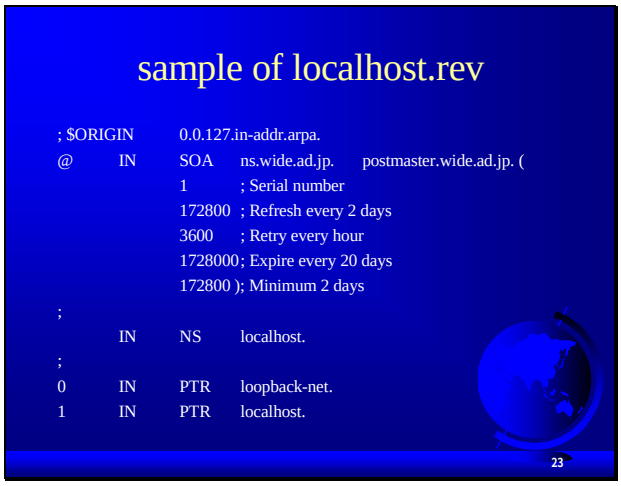
sample of localhost.zone：



ゾーンファイルの書き方です。localhostというゾーンがあり、localhostすなわち127.0.0.1ですが、ネームサーバに登録しておく必要があり、そのためのゾーンwideの書き方です。

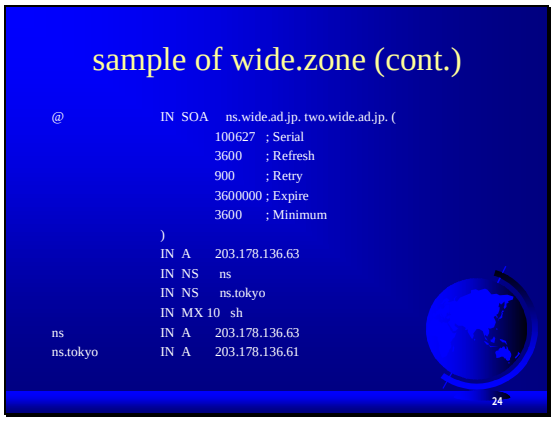
必要最小限の定義ですが、SOAというレコードの次にNSレコードがあり、次がAレコードという形です。(詳細後述)

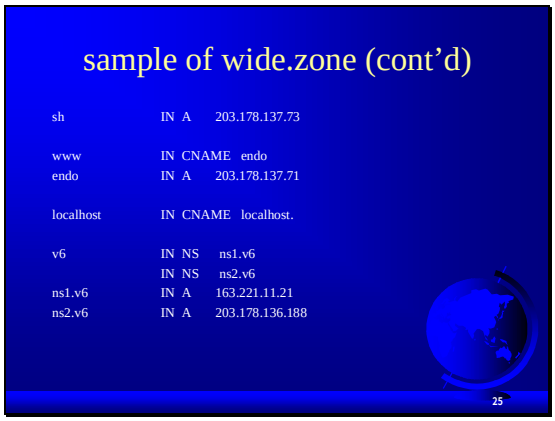
sample of localhost.rev：



逆引きの話です。ネームサーバで先ほどはlocalhostという名前から127.0.0.1というアドレスを引いてくる場合の定義でしたが、逆に1.0.0.127.in-addr.arpa.という定義です。127.0.0.1というIPアドレスからlocalhostという名前を引いてくるための定義です。(詳細後述)

sample of wide.zone:





wide.zoneでは、MXなどが増えています。逆引きはあまり変わりません。

レコード定義の基本型:

内容は前半で話したことと同じですが、ttlは省略可能で、そのレコードのキャッシュ期間、つまりそのレコードをどこかのネームサーバが引いたときにそのネームサーバがそれを覚えている時間ですが、ネームサーバを設定する側で定義することができます。普通、1時間とか2時間を書いておきます。

INはInternet Domainであることを表し、r-id(resource-ID)はSOA、NS、A、MXなどがあります。その後ろにvalueが続きます。このvalueがいくつ続くかはレコードの種類によって変わります。

key [ttl] IN r-id value1 value2 ...
<左辺> <右辺>

- ttl - 省略可
  - 当該レコードのキャッシュ期間
- IN (class-ID) - Internet Domain
- r-id (resource-ID)
  - レコードの種類 (SOA, NS, A, MX, ...)
- value
  - そのレコードの値 (r-id によって形式が違う)

レコード定義の基礎知識:

同一keyに対する定義は後続の定義のkeyに関して省略することができます。$ORIGINが先ほどのサンプルで出てきましたが、デフォルトのドメイン名の指定です。FQDN表記のホスト名の末尾にはピリオドを必ず補いましょう。

- 同一keyに対する定義
  - 後続の定義のkeyは省略可
- $ORIGIN <domain>
  - デフォルトのドメイン名の指定
  - 初期デフォルトはnamed.{boot,conf}のzone
- $INCLUDE <filename> [<domain>]
  - ファイルの挿入
- FQDN表記のホスト名の末尾には ． を

SOA(Start Of Authority) RR:

ネームサーバにゾーンを定義する場合には、SOA(Start Of Authority)というリソースレコードを定義する必要があります。これがゾーン間のコピーやゾーンの中で使われるタイムアウト情報を管理するためのレコードになります。

最初のkeyが@になっていますが、@はそのゾーン自体を表し、その後ろにネームサーバの名前を書いて管理者のメールアドレスを付けます。但し、管理者のメールアドレスは@をピリオドにしたもので、ネームサーバのアドレスがns.wide.ad.jp.となっていて、管理者のメールアドレスがありますがtwoというユーザ名で@wide.ad.jpと@がピリオドになっています。両方ともピリオドが最後に付きます。これがFQADNで、ここで完全に終わっていることを言うためにピリオドが最後に付きます。

次に数字が5つ続きますが、最初がSerialでマスターサーバとスレーブサーバ、プライマリサーバとセカンダリサーバの間でデータのコピーをするときにプライマリサーバでここを1つ増やすとセカンダリサーバがバージョンが上がったことに気が付いてそのゾーンのデータをコピーしてくれますから、内容が変わったことをセカンダリサーバに知らせるためにこれが使われます。

Refreshはセカンダリ・ネームサーバがSerialでバージョンが上がったどうかをチェックしにくる間隔を指定します。これを大きくしておくとrefreshが遅くなります。セカンダリサーバの意志で決まるのではなく、プライマリサーバで設定した数字によってセカンダリが見に来る設定が行われます。

Retryはrefreshに関連します。セカンダリ・ネームサーバがserialをチェックしに来る間隔はrefreshに指定されていますが、refreshの間隔で問い合わせに来たときにプライマリ・ネームサーバがunreachだったときはそれ以後はretryの間隔で聞きに行きます。

それでもプライマリ・ネームサーバが回復しなかった場合はそのデータがだんだん信頼できなくなって来ますので、最終的にExpirreという項目の時間を過ぎてしまったときはセカンダリに関してはプライマリから拾ったデータが有効性を保証できなくなりますので、そのゾーンに対しては問い合わせに対する応答を拒否することになります。そういう状態になっているときにnslookupで問い合わせを行うとサーバfailedという結果になりますので、このような状況の場合はどこかネームサーバの設定が間違っていることがわかります。

Minimum TTLはネームサーバに指定するデータベース、リソースレコードを書く行に対してTTLを省略可能な形で指定できましたが、そのデフォールトを指定します。全部のネームサーバに対して、そのデータを拾った瞬間から指定した秒数を覚えつづけます。

SOAパラメータ:

- Serial
  – Sec-NSのデータ更新判定用
- Refresh (秒)
  – Sec-NSのSerialチェック間隔
- Retry (秒)
  – Refresh経過後のチェック間隔
- Expire (秒)
  – サービス停止までのチェック不能期間
  – この状態で nslookup をすると
    *** ns.provider.ad.jp can't find x.co.jp.: Server failed
- Minimum TTL (time to live) (秒)
  – ゾーン内に定義される全レコードのデフォルト・キャッシュ期間
    (全NSに対して効果を持つ)

Serialについて:

Serialに関して1つ注意点があります。Serialは32ビットで表現されています。Serialの中にピリオドを入れて1.01などのバージョン管理をしているところがありますが、ピリオドは単純にゼロ3つと純粋に置き換えるだけの実装になっていますので、ピリオドを使ったからといって意味がありません。逆に、1.99から1.100のようにしたときに桁数が増えるときにおかしくなりますので、混乱を防ぐためにピリオドは使わない方が良いでしょう。

どうするかといいますと、2000年問題をクリアした西暦表記を付けて最後に日付の後ろに3桁くらい用意しておくとよく、1日1回更新しても西暦で4294年まで使えます。

上限はなく、32ビットの数字が増えて行くと1に戻ってまた増えて行くことを繰り返します。serialを増やしすぎたので戻したい場合、セカンダリサーバが沢山あるときにserialを戻すと人

手が掛かって非常に面倒くさいです。2147483647(7fffffff)以内を2回足すと元に戻せますので、このテクニックを使ってserialを下げることができます。

- 32ビット
- . による混乱に注意(使わない)
  – 1.01 = 100001 ("." は "000" と同値)
- 1997122501 など日付を使うと明瞭
  – 一日100回更新で4294年まで
- 上限なし(ループ):RFC1912(I)
  – 1に戻すことが可能
  – 2147483647(7fffffff)以内を2回足す

NS (Name Server) RR：

そのゾーンに対するネームサーバがどれかを指定するためのレコードです。Authorized Serverは上位ゾーンでそのネームサーバが書かれているものです。ネームサーバとはNSレコードがあるものです。

ゾーンに対してネームサーバの記述がありますので、wide.ad.jpのサンプルで、wide.ad.jpのゾーンに対してネームサーバがnsとns.tokyoとshが書いてあり、後ろがピリオドで終わっていませんのでこれに関しては自動的にwide.ad.jpが補われます(MXフィールドのshを除く)。このように、そのゾーン自体にネームサーバの情報を書くことも必要ですが、さらに上位のゾーン、つまりad.jpを管理しているネームサーバにおいてwideというゾーンがこのネームサーバに書かれていると、これによってはじめて上位から正式に権限を受けたことになります。

権限関係を調べるときは、そのゾーン自体のネームサーバ、NSレコードをチェックすることに加えて、さらに上位ゾーンのサーバに対して同じように検索して、両方に対してネームサーバの定義が行われているかをチェックする必要があります。ネームサーバの名前を書いたときにそれに対応するIPアドレスを書いておく必要があります。

別のゾーンに対するIPアドレスの定義のことをglue recordに書いてあります。ad.jpに対してwideはそのゾーンの中に含まれる情報になりますがns.wideまで指定したものはそのゾーンではなく、wideゾーンの中に本来管理されるアドレスのはずなんですが、上から順番にたどって行くためにはこの名前を知ったときにその名前がその時点で解決されていないといけませんのでglue recordと呼ばれるものが必要になります。

- Pri-NS および Sec-NS を記述
  – 上位ゾーンでの記述が重要
    - Authorized Server
  – 上位ゾーンに記述がない
    - Unauthorized Server
- 該当する NS に対する A RR も記述
  – glue record (逆引き zone には不要)

```
$ORIGIN ad.jp.
wide      IN  NS  ns.wide.ad.jp.      ;ad.jp.zone からの delegation
ns.wide   IN  A   203.178.136.63
```

lame (不完全な) NS:

ネームサーバを設定したときに、ネームサーバがメッセージを出して警告してくる内容がいくつかありますが、その1つとしてlameネームサーバという警告があります。これは上からdeligateされている、上のゾーンにNSが書いてあるサーバがあって、それに自分が聞きたいアドレスを問い合わせに行ったとき、それが実はそのゾーンに対するコピーを持っていなかった場合があります。つまり、上のゾーンに対するネームサーバの設定が間違っているか、上のゾーンにおける下のネームサーバの設定が間違っているか、ネームサーバ自体が正しくセカンダリに設定されていない状態がこれに当たりますが、その場合にlameサーバという警告が出ます。

これとメールの関係があり、こういう状況でAuthorized NSが全てアクセス不能になっていた場合に問題があるネームサーバがたまたま見えていたとき、これを上からたどって行くとAuthorizedサーバなので聞きに行くとそれに対するゾーン情報を持っていないので知らないと答えて来るので、Authorizedサーバから知らないと言われたので存在するはずのアドレスが無かったことになりメールが落ちてしまいます。従って、このような設定に関してはAuthorized関係をきちっと設定する必要があります。

MX (Mail eXchanger) RR:

$ORIGINがあった場合は、以下のピリオドで終わっていない指定に対して、このアドレスを自動的に補う機能を持っています。例では@はそのままwide.ad.jpと置き換えられますので、wide.ad.jpに対するMXレコードはプレファレンスが10でメールサーバはsh. wide.ad.jpと補われます。

- MX RR
  – メールアドレスから配信先ホスト名へのマップ
  $ORIGINwide.ad.jp.
  @ IN MX 10 sh
- MX は A より優先(メールの配信)
- A を優先させたいとき
  – 1st-MX で転送

CNAME(Canonical NAME) RR:

CNAMEというのがある、例えばarchieに対して、別のAレコードやMXレコードを定義してはいけないという決まりがあります。さらにネームサーバの右辺やMXレコードの右辺にCNAMEで定義されるkeyを指定することもできますが、最近のネームサーバはそういった定義に対して警告を発するようになっていますので、たとえそれでうまく動くとしても、そのような使い方はRFC的に禁止されていますので、使わないようにしましょう。

- ホストの別名定義
  $ORIGINwide.ad.jp.
  archie IN CNAME sun3.tokyo
  – CNAME チェインはできるだけ避ける

– 同一 key に別の種類のレコードを定義しない
– 同一 key に複数の CNAME は定義しない
・ NS, MX の右辺に CNAMEで定義される名前を使わない

PTR (domain name PoinTeR) RR：

ポインタのレコードは、IPアドレスからホストへのマッピング(逆引き)をします。例はsh.wide.ad.jpのホスト名を逆引きするための設定です。

逆引きのためには、IPアドレスをドメイン構造に対して対応付けるために、IPアドレスを203.178.137.73のように上の桁から順に書いて行き一番右側がホスト毎に変化するわけでサブネットという単位でIPアドレスを管理するものをネームサーバのドメイン構造と対応付けた場合には一番変化する部分がドメイン名の表記においては一番左側になるということになりますので、IPアドレスの表記は左右が逆になります。というわけで、IPアドレスは203.178.137.73というホストに対する定義においては73.137.178.203.in-addr.arpa.というレコードを定義することになります。

ポインタレコードで問題になるのは、逆引きできないホストからのサービスは受け付けないサービスをしているホストがあったり、あるいはドメイン名が偽ってないかなどを確認したりする目的で使われます。

うそつき問題は最近あまり話題でなくなってしまいました。逆引きはIPアドレスからホスト名を対応付けるデータベースですが、IPアドレスは自分が管理している、IPアドレスに関するゾーンは自分が管理している場合に、ここには何を書いてもいいわけで、自分のホストを逆引きしたときに出てくる名前は本当の名前でなくてもよく、うその名前でもよいので、単方向で設定されているデータベースだと、このようにうそをつくことができます。最近は、うそつき問題が気になるようなホストであれば、1回引いたこの名前でもう1回引き直しを行って、それが最初のIPアドレスと等しいかどうかを確認して、それで始めて信用することになっていますので、うそをつきにくくなっています。

・ IP アドレスからホスト名へのマッピング
– 逆引き

```
$ORIGIN      137.178.203.in-addr.arpa.
73           IN PTR sh.wide.ad.jp.
```

– PTR レコード検索によるサービス制限
・ 引けないホストからのアクセス拒否
・ ドメイン名の確認
・ うそつき問題
– アドレス→ホスト名の単方向だと騙れる
– 引き直しチェック

nslookup で逆引きの確認:

これを使って逆引きのチェックをしたい場合、ホストのIPアドレスが1.2.3.4のときはこれを逆に並べて.in-addr.arpa.を後ろに付けて引きます。queryのタイプはptrにします。

昔は、.in-addr.arpa.を付けて書いていましたが、最近は、逆にひっくり返して.in-addr.arpa.を付けるのは面倒なので、4.8.以降のnslookupの場合は、そのIPアドレスを書くと自動的に逆引きに変換して引いてくれます。

- ホストのIPアドレスが 1.2.3.4 のとき
  % nslookup
  > set q=ptr
  > 4.3.2.1.in-addr.arpa.
- 新しい(4.8.3以降)nslookup
  % nslookup 1.2.3.4

ネットワーク名の定義:

ホスト名ではなくネットワークに対する名前を付けることもネームサーバで管理することができます。例えば、京都大学は130.54.0.0というクラスBのネットワークですが、京都大学のネットワークの名前がkuinsであることを以下のようにして定義できます。最下行はマルチキャストの定義の例です。

- RFC1101(?): DNS Encoding of Network Names and Other Types
- netstat -i, -r などで参照される

```
0.0.54.130.in-addr.arpa.        IN PTR kuins.kyoto-u.ac.jp.
                          IN A  255.255.0.0
kuins.kyoto-u.ac.jp.            IN PTR 0.0.54.130.in-addr.arpa.

0.0.0.224.in-addr.arpa.         IN PTR BASE-ADDRESS.MCAST.NET.
```

その他のレコード:

いままで紹介していない以下のレコードがあります。新しいレコードが次々と実験されています。

- HINFO, TXT, WKS
  – HINFO は必ず2つ以上のパラメータを書く!
- NULL, MB, MG, MR, MINFO (experimental) (RFCがstandardのものでexperimentalなもの)
  – RFC1035(S)
- AFSDB, ISDN, RP, RT, X25 (RFC自体がexperimentalなもの)
  – RFC1183(E)
- PX (RFC自体がexperimentalなもの)
  – RFC1664(E)

localhost/127.in-addr.arpa zone:

localhostのゾーンのところの例で話しが出ましたが、localhostあるいはそれに対応するIPアドレス127.0.0.1からの逆引きは、全てのネームサーバに設定するようにしましょう。一応rootネームサーバに対してもその設定はされていますので、もし手元のネームサーバにlocalhostに関する設定がされていなかったら全ての問い合わせはroot serverから始まりますので、そこに行ってlocalhostを検索してroot serverから返事が返って来ます。従って、定義しなくても引けなくはないのですが、できるだけ無駄を無くすために全てのネームサーバでlocalhostあるいは逆引きに関する設定をしましょう。

localhostにIN Aと書いて127.0.0.1を記述することがありますが、それよりも、例のようにCNAMEでlocalhost.という名前にマップしておく方が良いでしょう。というのは、逆引きをしたときに127.0.0.1を引くとlocalhostが出てきますが、それをさらに引き直しに行くとlocalhostがもし自分のゾーンになっていると、my.domain.jp.とかにあってwhoとかとかしたときにlocalhostからのアクセスが自分のドメインが付いた形で記録されてしまったりしますので、それが煩わしくなります。localhostで切りたいときは正引きのときは例のように書くといいです。

- すべてのネームサーバに設定すべき
  - root server まで問い合わせるのは無駄

```
$ORIGIN my.domain.jp.
localhost IN CNAME localhost.
```

- 引き直しの際の不整合の防止
  - 127.0.0.1 → localhost.my.domain.jp にならないように

CIDRと逆引き管理:

最近、クラスレスの割り当てが始まっていますが、ネームサーバの逆引きの区切りは8ビット単位で行われているわけですので、例のように/25とか/26とかで割り当てられたときにどうするかという問題です。やり方は複数ありますが、CNAMEで散らす方式がインターネット・ドラフトdraft-ietf-dnsind-classless-inaddr-03.txtで定義されています。

- class less なアドレスの割り当て
  - 192.0.2.0/25 - 組織Aに
  - 192.0.2.128/26 - 組織Bに
- 逆引きゾーンの管理単位問題
  - オクテット(8ビット)単位の権限委譲との不整合
- 解決策
  - CNAMEで散らす
    - draft-ietf-dnsind-classless-inaddr-03.txt
  - NSで散らす

Classless IN-ADDR.ARPA delegation：

上位ゾーンからの権限委譲の方式としては、そのゾーンそのものに対してそれぞれ書いて、それをそれぞれ持っているゾーンで、さらに、割り当てた組織でそれぞれ管理できるように、割り当てられた範囲だけを管理できるゾーンを1つ用意して、そこへのCNAMEのマップを用意します。

割り当てられた方は、それに対して正しいホスト名を定義します。これによって、8ビット単位で区切られていないところに関しても正しく逆引きを管理することができる方式です。

・ 上位ゾーンからの権限委譲

```
$ORIGIN 2.0.192.in-addr.arpa.
;   <<0-127>> /25
0/25    NS    ns.A.domain.jp.
1               IN    CNAME    1.0/25.2.0.192.in-addr.arpa.
2               IN    CNAME    2.0/25.2.0.192.in-addr.arpa.
 :
126             IN    CNAME    126.0/25.2.0.192.in-addr.arpa.
```

・ 当該ゾーンでの定義

```
$ORIGIN 0/25.2.0.192.in-addr.arpa.
@               IN    SOA ...
                IN    NS    ns.A.domain.jp.
1               IN    PTR    host1.A.domain.jp.
2               IN    PTR    host2.A.domain.jp.
 :
126             IN    PTR    host126.A.domain.jp.
```

DNSとIPv6：

IPv6の場合は4ビット毎に区切って管理します。4ビットの境界に対して対応しない場合には、先ほどのやり方を使います。

・ IPv6 アドレスは 128 ビット長
・ CIDR で発生する問題をどう解決するか
  – 逆引きの区切りは 4 ビット単位
    ・ 16進表記
  – IPv4 と同じ方式を利用する

```
sh.v6.wide.ad.jp.               IN    AAAA    3ffe:501:1000::1
$ORIGIN    0.0.0.1.1.0.5.0.e.f.f.3.ip6.int.
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0          IN PTR sh.v6.wide.ad.jp.
```

・ 古いnamedによるセカンダリ
  – ゾーンの転送がうまくいかない
  – 少なくとも bind 4.9.3以降にする
・ 古い sendmail(?)

– AAAA RRを拾うと v4 宛てのメールが落ちる
・ additional information で送られてくる
・ まだ運用系に定義しない方が安全か

メールアドレスの補完:

ワイルドカードMXのところで少し説明しましたが、ドメインをresolv.confに書くときに、例のようにsub.x.co.jpと書いた場合、サーチをかけるときと同じ意味になります。サーチの階層は米国の場合を考えて作られているので、米国の場合は組織名.eduとか組織名.orgとか組織名.comなど最初のところの2階層で組織を特定できます。ところが、日本の場合はトップレベルが国の識別子ですので、このようなサーチは意味がありませんので、そのようなことが起こらないように考慮する必要があります。つまり、サーチで明示的に自分の組織のところまでしか書かないようにする必要があります。最近のバージョンはRFC1535(I)で、そのような暗黙の遡りのサーチをドメインに関しては行わないようになってきていますので、実際にこのように複数の階層でサーチをかけたいときはサーチをちゃんと使う必要があります。

・ MX RR と A RR を用いる
– ワイルドカードMX問題
・ /etc/resolv.conf に定義

domain sub.x.co.jp

– search sub.x.co.jp x.co.jp co.jp と同値
・ 遡って3階層分調べる (MAXDFLSRCH)
・ 最短は2レベル (LOCALDOMAINPARTS)
– JP domain の実状にあわない
– RFC1535(I)で暗黙の遡りを禁止

サーチをsearch sub1.x.co.jp sub2.x.co.jp x.co.jpのように書いたときにどのように検索されるかというと、下のようになります。補完をしないものを最初に検索することによって、無駄な検索を省きます。

・ LOCALDOMAIN 環境変数によるユーザ設定
– 最大6ドメイン (MAXDNSRCH)
・ 検索の順序

nic.ad.jp
nic.ad.jp.sub.x.co.jp
nic.ad.jp.x.co.jp
nic.ad.jp.co.jp

– RFC1535(I)より前は nic.ad.jp を最後に検索

Wildcard MX is harmful:

どのような問題があるか言っておきます。exact RRが存在しない場合にマッチするというのがこのWildcardのレコードの性質です。その場合の1つ目の問題として、WildcardMXが存在した場合に本来受け取ることができないアドレスに対して、メールサーバにメールが飛んでしまい、メールサーバに届いた時点でエラーになります。存在しないアドレスが送信する時点でわかれば余計なトラフィックが流れないのですが、余計な配信が行われてしまいます。

存在しないアドレスに補完されるので、sendmailにおいてメールアドレスを補完する機能を使った場合にWildcardMXがあるとそれが付いた形でメールアドレスが補完されてしまい、本当は正しく届くはずだったアドレスが余計な自分のドメイン名が二重に付いた形にアドレスが書き換えられ、存在しないアドレスに補完されてしまいメールが落ちることもあります。どうしてもWildcardMXが必要な場合は、sendmail.cfのResolverOptionsの定義項目にHasWildcardMXを書くと余計な補完、WildcardMXを見た補完をしなくなります。

- exact RRが存在しない場合にマッチ
- 存在しないアドレスにもメールが飛ぶ
  - 送信時に存在しないアドレスであることが不明
- 存在しないアドレスに補完される
  - ResolverOptionsにHasWildcardMXを定義
    - sendmail.cf
- 配信先に対応するMX RRが引けない
  - 配信先ホスト名の最後に必ず . を補う

→ どうしても必要な場合にのみ利用する

古いglueレコードが消えない:

最近のメールサーバはこれらの問題は回避されるようになっていますので、古いメールサーバを使っていない限り問題ないでしょう。

- 4.8.3以前?
- server A: primary of x.co.jp
- server B: primary of sub.x.co.jp
  - お互いにセカンダリになっている
- x.co.jp の NS (server C)のアドレスを変更
- server C の古い glue レコードが消えない
  - server A で消しても
  - server B のからの zone transfer で甦る

  → セカンダリコピーからも消す

サーバが報告するエラー:

先ほど説明したlameサーバがありましたが、CNAMEの先に何も書いていないとか、MXやNSの右辺がCNAMEであるとか、NSが指している先にSOAが見つからないとか、それらを見つけてレポートするようになっていますので、これらのメッセージも参考になります。

- bad referral

– NS があるのに SOA がない
- NS points to a CNAME
- MX points to a CNAME
- dangling CNAME pointer
  – CNAME の先が何も指していない
- Lame server on 'x.co.jp'
  – Authorizedのはずなのに、Unauthoritative answerが返ってきた
- Response from unexpected source
  – 違うインターフェースアドレスからの応答?
  – アタック?
- zone "xxx" (class 1) SOA serial# (nn) is < ours (mm)
  – SOA serial が減った!

RFC1912(I): Common DNS Operational and Configuration Errors

デバッグ用ツール:

デバッグツールも以下のいろいろなものがありますので、それらも参考になります。

RFC1713: Tools for DNS debugging
- Host (bind 8 に添付)
  – ftp://ftp.nikhef.nl/pub/network/host_YYMMDD.tar.Z
- Dnswalk (bind 8 に添付)
  – ftp://ftp.pop.psu.edu/pub/src/dnswalk
- Lamers
  – ftp://terminator.cc.umich.edu/dns/lame-delegations/
- Doc (Domain Obscenity Control)
  – ftp://ftp.uu.net/networking/ip/dns/doc.2.0.tar.Z
- DDT (Domain Debug Tools)
  – ftp://ns.dns.pt/pub/dns/ddt-2.0.1.tar.gz
- Checker
  – ftp://catarina.usc.edu/pub/checker
- Dig (bind 8 に添付)

設定変更の作業手順:

組織のネームサーバのIPアドレスを変更するとかメールサーバのIPアドレスを変更するときには、作業手順を考えて行わないとメールが落ちたりするので気を付けて下さい。

- ドメイン名の変更
  – メールアドレスの二重運用
- ドメインのIPアドレスの変更
- ネームサーバの変更(別のホストに)
- ネームサーバのIPアドレスの変更

– RR に新旧を定義
・メールサーバの変更(別のホストに)
– sendmail.cf で新しい方に転送
・メールサーバのIPアドレスの変更

DNSの今後:

DNSはインターネットの基本機能の一つですが、現在もなお改良や機能拡張が続けられています。たとえば現在では次に示すようなトピックがあります。

・ Dynamic Update
– レコード単位のデータ更新
・ Incremental Zone Transfer (IXFR)
– トラフィックの削減と更新速度の向上
・ Security Extention
– SIG RR, NXT RR

ポイントをかき集めて説明しましたが、これからメールサーバとかDNSサーバを管理して行こうとする人にとっては難しい話しだったかも知れません。私の経験上、しっかり押さえておいた方がよい、重要なポイントをまとめてお話しましたので、この辺をしっかりおさえて管理していかれると一人前の管理者としてやって行けるのではないかと思います。

以上