5 Web サービスを IPv6 対応にするには

小山哲志
合同会社ほげ技研／日本 UNIX ユーザ会

インターネット上のさまざまな Web サービスは今や生活に密着しており，なくてはならないものと呼べるものも多い．無論それらのサービスは，現在はたいてい IPv4 で動いている．ここでは IPv4 で動いている Web サービスを IPv6 対応にするための具体的な手法について解説する．

なお本稿で例として使用される IP アドレスは，以下の RFC で規定されている文書サンプル用のアドレスを用いることとする．

RFC5737 192.0.2.0/24
RFC3849 2001:DB8::/32

なお近年の Web サービスは単独のサーバのみで運用されていることはほとんどなく，役割ごとのサーバを複数用意してその組合せでサービス全体をまかなっている場合が多い．その点では IPv6 対応といっても以下の 2 つのやり方が考えられる．

- サーバをすべて IPv6 対応にする
- 直接ユーザとやりとりするサーバのみ IPv6 対応にする

サーバ群を構成するホストをすべて IPv6 対応にすることは，一見単純に見えるが，サーバ群の内部ネットワークに関しても IPv4，IPv6 の両方を同時に管理することになるので，必要以上に構成を複雑化することになる．

サーバ群の内部ネットワークは IPv4，IPv6 のどちらかで統一し，直接ユーザに対してサービスする部分のみを IPv4/IPv6 のデュアルスタックにするのが，効率的なネットワーク設計というものだろう．そしてサーバ群の内部ネットワークにどちらを使うかは，よほど大規模で複雑な構成でないかぎりはこれまで通りの IPv4 で問題ないだろう．

IPv6 対応の実際

Web サービスを IPv6 に対応させるのは，基本的にはさほど難しいことではない．Apache Httpd や nginx などの Web サーバソフトウェアはかなり以前より IPv6 対応が完了しており，設定ファイルに IPv6 用の記述を追加するだけで IPv4/IPv6 のデュアルスタックで動作するようになっている．ただし現在の Web サービスはサーバ上で各種のプログラムを動作させ，HTML ページを動的に生成する等が当たり前となっているので，そのプログラムが IP アドレスを扱う場合はその部分も IPv6 対応になっている必要があるのは言うまでもない．

世界で最も使われている Web サーバである Apache Httpd を例に，IPv6 の設定の実際をみていこう．Apache Httpd はバージョン 2.0 以降では IPv6 をサポートしている．IP アドレスとポート番号を指定する個所では，従来の IPv4 アドレスと同様に IPv6 アドレスを記述できる．ただし IPv6 アドレスの表記は [ ] で囲む必要がある．

サーバに以下の IP アドレスが付いているとする．

IPv4 : 192.0.2.80
IPv6 : 2001:db8::ac1d:cafe

このときに上記それぞれの IP アドレスの port 80 にて Web サーバを立ち上げる場合の設定は以下のようになる．

```
Listen 192.0.2.80:80
Listen [2001:db8::ac1d:cafe]:80
```

名前ベースのバーチャルホストを使用する場合は，IPv4，IPv6 それぞれに NameVirtualHost を指定する．

```
NameVritualHost 192.0.2.80:80
NameVritualHost [2001:db8::ac1d:cafe]:80

<VirtualHost 192.0.2.80:80
[2001:db8::ac1d:cafe]:80>
  ServerName www.example.com
  DocumentRoot /var/www/html
  CustomLog logs/www.example.com/access_
    log common
  ErrorLog logs/www.example.com/error_
    log
</VirtualHost>
```

上記のように VirtualHost ディレクティブには，IPv4, IPv6 の両方を書くことができるが，むしろそれぞれのログを分けたり多少コンテンツを変えたいと思うのが普通だろう．その場合はそれぞれに対してバーチャルホストを設定すればよい．

```
<VirtualHost 192.0.2.80:80>
  ServerName www.example.com
  DocumentRoot /var/www/v4html
  CustomLog logs/www.example.com/
    v4.access_log common
  ErrorLog logs/www.example.com/
    v4.error_log
</VirtualHost>

<VirtualHost [2001:db8::ac1d:cafe]:80>
  ServerName www.example.com
  DocumentRoot /var/www/v6html
  CustomLog logs/www.example.com/
    v6.access_log common
  ErrorLog logs/www.example.com/
    v6.error_log
</VirtualHost>
```

また，ACL（Access Control List）で IPv6 アドレスをブロックで指定する場合は [ ] で囲まずに記述する．

```
Allow from 2001:db8:dec0::/64
```

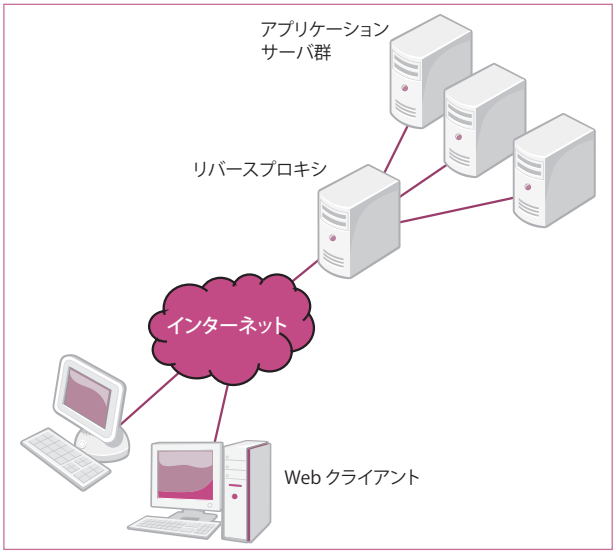

図 -1　Web サービスのサーバ構成例

リバースプロキシを IPv6 対応にして内部に転送する

複数台のホストで Web サービスを構築する際には，機能的な役割ごとに Web サーバを複数用意して分散処理を行うことが一般的である（図 -1）．その場合ユーザからの HTTP リクエストを引き受けて，その内容により適切な役割の内部サーバにリクエストを転送する，いわばユーザから見たフロントエンドとしての Web サーバが用いられる．これをリバースプロキシと呼ぶ．

サーバ構成がこのような形態になっていれば，ユーザのリクエストを直接処理するリバースプロキシのみを IPv6 対応すれば，IPv6 アドレスから接続してきたユーザにもサービスを提供することができる．

リバースプロキシを実装するにはいくつもの方法がある．Apache Httpd に mod_proxy や mod_rewrite といったモジュールを組み合わせてもよいし，近年では Varnish というキャッシュサーバがよく用いられている．

たとえば Apache Httpd に mod_proxy を組み合わせて，グローバル IPv6 アドレス 2001:db8::ac1d:cafe から 10.10.0.1 の内部サーバにすべてのリクエストを転送する設定は，以下のようになる．

```
Listen [2001:db8::ac1d:cafe]:80

<VirtualHost [2001:db8::ac1d:cafe]:80>
  ServerName www.example.com
  ProxyPass / 10.10.0.1
  ProxyPassReverse / 10.10.0.1
</VirtualHost>
```

プログラミングでの IPv6 アドレスの扱い

サーバアプリ内で IPv6 アドレスを扱う場合にも注意しなければいけない点がいくつか存在する．

❖ IP アドレスを表す文字列の最大長

IPv4 アドレスの場合は

```
XXX.XXX.XXX.XXX
```

という形式なので最大で 15 文字となるが，これが IPv6 アドレスの場合は通常の形式（0 省略なし）の場合でも 39 文字になる．

```
XXXX:XXXX:XXXX:XXXX:XXXX:XXXX:XXXX:XXXX
```

netinet/in.h には

```
#define INET6_ADDRSTRLEN 46
```

と定義されており，さまざまな表記を用いてもこの文字数を超えることはないことが保証されている．

データベースなどにおいて IP アドレスをレコードに保存している場合，可変長の文字列として扱っている場合は問題ないが，固定長の文字列または最大長付きの文字列としてカラムを定義してある場合は，上記の制限にぶつかっていないか確認する必要がある．

余談だがオープンソース RDBMS である PostgreSQL には INET 型，CIDR 型という IP アドレスを保存する専用のデータ型があり，これを用いると IPv4/IPv6 両方のデータを問題なく保存することができる．

❖ IP アドレスのバリデーション

HTML フォームでユーザに IP アドレスの入力をさせている場合，それが正しい IP アドレスかどうかを確認する必要があるかもしれない．IPv4 の場合は単純な正規表現でもある程度正当性を確認できるが，省略記法が一般的な IPv6 アドレスの場合は正規表現 1 つで確認するのは多少難しい．もちろん正規表現で頑張る人たちもいるので，検索すれば複雑な正規表現を見つけることもできるだろう．通常は各プログラミング言語に用意されている IPv6 アドレスの検証を行うライブラリを使うのがよいだろう．

たとえば PHP では，PEAR の Net_IPv6 パッケージを用いて以下のように書くことができる．

```
<?php
require_once 'Net/IPv6.php';

$ipv6addr = '2001:db8::ac1d:cafe';
$valid = Net_IPv6::checkIPv6($ipv6addr);
?>
```

❖ IPv6 アドレスの比較

すでに書いたように，IPv6 アドレスは複数の省略形が許されるので表記が一意に定まらない．そのため単純な文字列比較だけでは，本来同じ IPv6 アドレスとしてマッチするはずがそうはならないケースも考えられる．

具体的な例を挙げてみよう．2001:db8:0:0:d1ce:0:0:1 という IPv6 アドレスはまったく省略なしに書くと以下のようになる．

```
2001:0db8:0000:000:d1ce:0000:0000:0001
```

これに対して以下の省略ルールを適用する．

- 連続した 0 は 1 回のみ「::」という省略記法が使用できる
- フィールド内の先頭の 0 は省略できる

また 16 進表記の a-f は大文字も許されるので，この IPv6 アドレスに対する表記方法は

```
2001:db8::d1ce:0:0:0001
2001:db8::d1ce:0:0:1
2001:db8:0:0:d1ce::1
2001:DB8:0:0:D1CE::1
2001:0DB8:0:0:D1CE::1
```

など無数に考えられる．

この解決案として考えられるのは，比較する IPv6 アドレスをあらかじめ何らかの表記に正規化しておくことだ．RFC5952 ではこのような多様な表記の IPv6 アドレスに対して一定の正規化を行い，

人間に分かりやすい省略記法を維持しながら表記が一意になるように規定をしている．上記のアドレスに RFC5952 の正規化を適用すると

```
2001:db8::d1ce:0:0:1
```

となる [1]．

ただし RFC5952 は 2010 年 8 月発行された比較的新しい RFC なので，各プログラミング言語の IPv6 ライブラリでこれに対応しているものはあまり多くない．

JavaScript で実装された Validator/ 正規化コードとしてとして，以下のサイトで公開されているものがある．

```
IPv6 Address Validation
http://oldsite2.dartware.com/ipv6regex
```

ここで使用されている JavaScript のコード

```
http://download.dartware.com/
  thirdparty/ipv6validator.js
```

は超絶な正規表現を用いて IPv6 アドレスのバリデーションを行い，RFC5957 に従った正規化を行っている．

プログラミング言語から直接 IPv6 接続をする

これまではプログラミング言語を Web サーバの裏側で使用する場合について解説してきた．このケースでは IPv6 の接続そのものは Web サーバソフトウェアが管理するので，プログラム言語が直接扱う必要はなかった．しかしさまざまな Web サービス API を叩いて外部から情報を得ることは今や普通であり，その場合はプログラム言語から直接ネットワーク接続を行うことになる．

といっても，Web サービス API 側が IPv6 のみの接続を許すということはここしばらくは考えられず，従来通り IPv4 でアクセスすれば問題ないだろう．必要性としてはそれほど高くはないが，プログラム言語から直接 IPv6 接続を扱う場合のことを考えてみる．

IPv6 接続をプログラミング言語がサポートするには，C 言語の API レベルで gethostbyaddr/gethostbyname という従来の関数ではなく，getaddrinfo/getnameinfo というプロトコル非依存の新しい関数を使用している必要がある．

結論から言えば，さまざまなスクリプト言語を含め最近のプログラミング言語ではほぼ間違いなく新しい API を用いており，問題なく IPv6 接続を扱うことができる．たとえば PHP で以下のようなプログラムを作成し，サーバに接続を行うとする [2]．

```
<?php
$fp = fsockopen('www.example.com', 80,
$errno, $errstr, 30);
if (!$fp) {
    echo "$errstr ($errno) <br />\n";
} else {
    $out = "GET / HTTP/1.1\r\n";
    $out .= "Host: www.example.com\r\n";
    $out .= "Connection: Close\r\n\r\n";
    fwrite($fp, $out);
    while (!feof($fp)) {
        echo fgets($fp, 128);
    }
    fclose($fp);
}
?>
```

このプログラムを実行した場合，IPv6 が有効な環境ではまず IPv6 での接続が試みられ，それが失敗すると IPv4 にフォールバックして再び接続が試みられる．

このプログラムを見ても分かるように IPv6 に対応する特別な処理はなんら書かれていないが，プログラム言語そのものが IPv6 に対応することで，従来のプログラムがそのままで IPv6 対応になっている．

以上のように，Web サービスを IPv6 に対応させるための技術は整ってきており，少しの追加と注意をすれば対応が可能になったと言えるだろう．

参考文献

1) http://www.nic.ad.jp/ja/newsletter/No46/0800.html
2) fsockopen PHP マニュアルより．http://php.net/manual/ja/function.fsockopen.php

(2011 年 7 月 7 日受付)

小山哲志 koyama@hoge.org
会社員とフリーランスの二足のわらじを履く Web サービス開発者．
twitter: @koyhoge