Internet Week 2011

IPv4 アドレス枯渇時代のアプリケーション開発

Lightweight Language と IPv6

2011/12/1

社団法人日本ネットワークインフォメーションセンター

関根 佳直

JPNIC 社団法人 日本ネットワークインフォメーションセンター

自己紹介

2009年～ JPNIC。
普段はサーバ / ネットワークの構築・運用をしています。

よく使う言語

C, Perl, PHP

好きな言語

Prolog, Lisp 等

興味のあること

プログラミング言語全般, コンパイラ, コンピュータアーキテクチャ等

本セッションの概要

Lightweight Language (後述) の現時点での IPv6 対応状況を, 実例を交えながら概観します。

本セッションで扱う Lightweight Language

- Perl http://www.perl.org/
- Python http://python.org/
- PHP http://php.net/
- Ruby http://www.ruby-lang.org/

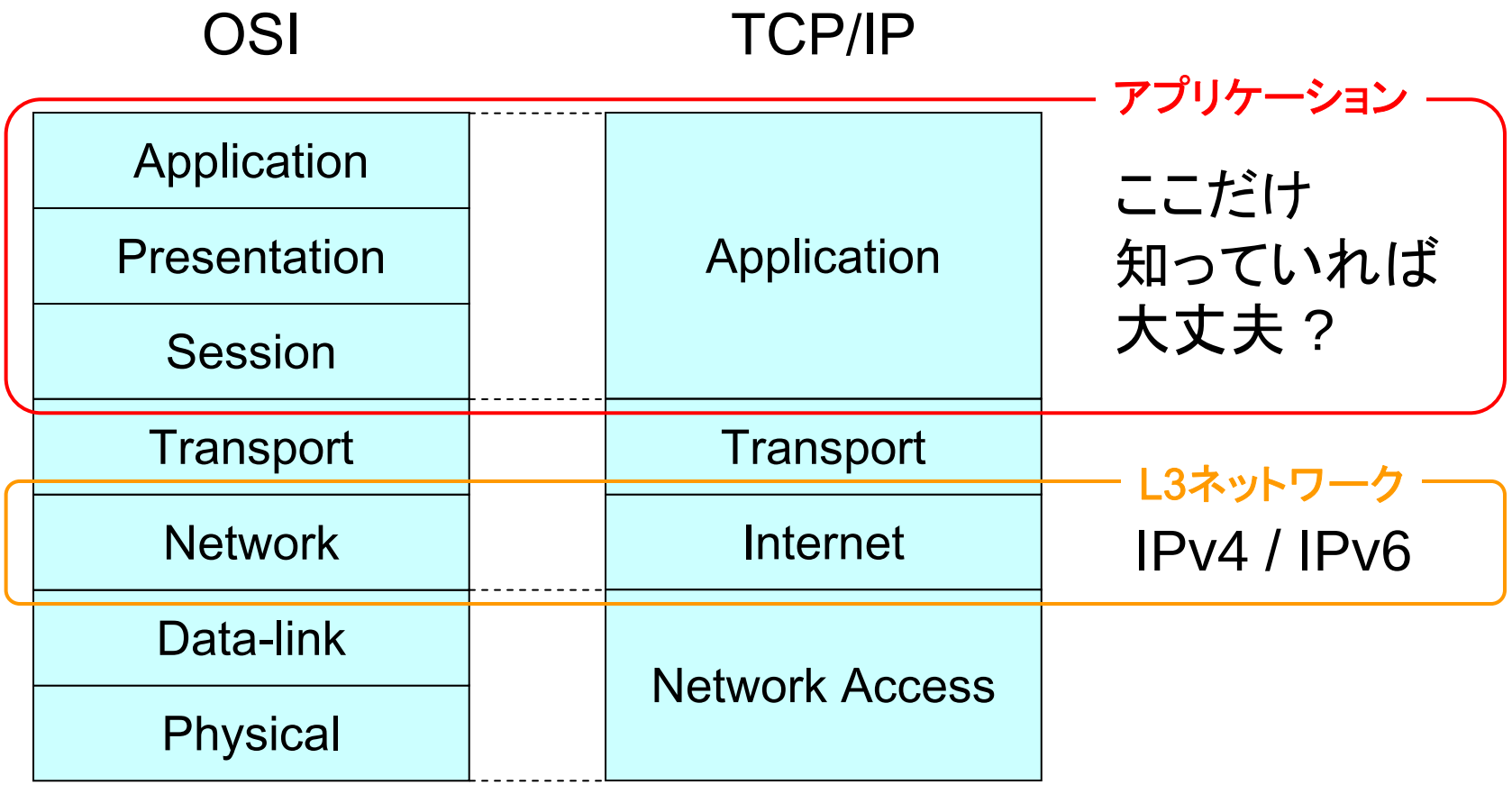
※ 時間の都合上, 内容に偏りがあります

プログラミング言語の IPv6 対応とは

- ソケット
  - アドレスファミリーが AF_INET6 のソケットを扱えるか
  - プロトコル非依存なインタフェースがあるか (getaddrinfo() 等)
- DNS
  - ホスト名から IPv6 アドレス (AAAA レコード) が引けるか (正引き)
  - IPv6 アドレスからホスト名が (簡単に) 引けるか (逆引き)
- 各種プロトコル
  - IPv6 で Web アクセス (HTTP) やメール送信 (SMTP) が行えるか
- IPアドレスの処理
  - IPv6 アドレスの表記の変換等の処理ができるか

etc.

レイヤの話



Lightweight Language とは

一言で言うと, “豊富な機能を簡単に使える言語” (?)

一般的に以下のような特徴がよく見られる

- コンパイル不要
- 動的型付け
- 高度な抽象化 (コード量が少なくなる)
- 便利なデータ型 (動的配列やハッシュ等)
- (標準) ライブラリが豊富 (やりたいことがすぐにできる)

→ プログラミングの負担が “軽い”

≒ Web アプリケーションに使用されるスクリプト言語

※ 英語圏では上記の意味では通じない可能性が高い

雑談 – 各言語でのハッシュ

Perl (ハッシュ)

```
my %var = ("v4" => "192.0.2.1", "v6" => "2001:db8::1");
print $var{"v6"};
```

Python (辞書, dictionary)

```
var = {"v4": "192.0.2.1", "v6": "2001:db8::1"}
print(var["v6"])
```

PHP (配列) // インデックスが数字でも文字列でも同じ配列型

```
$var = array("v4" => "192.0.2.1", "v6" => "2001:db8::1");
print $var["v6"];
```

Ruby (ハッシュ)

```
var = {"v4" => "192.0.2.1", "v6" => "2001:db8::1"}
print var["v6"]
```

Perl

Perl

Perl のバージョン

最新版: 5.14.2 (2011/9/26)

検証に使用したバージョン: 5.8.8, 5.12.4, 5.14.2

“Programming Perl 3rd Edition” (O’reilly, 通称ラクダ本)で扱っているのは5.6

Perl のライブラリ

標準で豊富な機能を備えたモジュールが付属 (コアモジュール)

拡張ライブラリとして, 膨大な数のモジュールの集合である CPAN (Comprehensive Perl Archive Network) がある

http://www.cpan.org/

Perl で書かれた有名なアプリケーション

SpamAssassin, MRTG, Movable Type, etc.

Linux でのバージョン

ディストリビューションのパッケージでインストールされるバージョン
(2011/11/20現在)

ディストリビューション		Perlのバージョン
RHEL	5.7	5.8.8
	6.1	5.10.1
Debian	5.0	5.10.0
	6.0	5.10.1
Ubuntu	10.04 LTS	5.10.1
	10.10	5.10.1
	11.10	5.12.4
Fedora	15	5.12.3
	16	5.14.1

Perl のソケット

- 基本的なソケット関数はビルトイン
  - socket(), bind(), listen(), connect(), …
- Socket
  - コアモジュール, Perl 5.14 から本格的に IPv6 に対応
- IO::Socket::INET
  - コアモジュール, IPv4 のみ
  - 多くの CPAN モジュールが依存
- IO::Socket::INET6
  - IPv4 / IPv6 両対応
- IO::Socket::IP
  - IPv4 / IPv6 両対応
  - IO::Socket::INET の置き換え

Socket

低レベルのソケットインタフェース

Perl 5.14 で IPv6 フルサポート

- AF_INET6, IN6ADDR_ANY, IN6ADDR_LOOPBACK
- inet_pton(), inet_ntop()
- pack_sockaddr_in6(), unpack_sockaddr_in6()
- getaddrinfo(), getnameinfo()

※ getaddrinfo(), getnameinfo(), inet_ntop(), inet_pton() や一部の定数はデフォルトでエクスポートされない

getaddrinfo()

getaddrinfo(*$host*, *$service* [, *$hints*])

- $host: ホスト名
- $service: サービス名またはポート番号
- $hints: 結果を限定するための情報を含んだハッシュのリファレンス
  - ハッシュのキーには次のようなものがある
  - family: アドレスファミリー (AF_INET, AF_INET6)
  - protocol: プロトコル (IPPROTO_TCP, IPPROTO_UDP)
- 返り値: ($err, @results) というリスト
- $err: エラー情報
- @results: ホストの情報を含んだハッシュのリファレンスのリスト
  - ハッシュは次のようなキーを含む
  - family: アドレスファミリー (socket() に渡す)
  - addr: パックされたバイト列 (connect() や bind() に渡す)
  - canonname: ホストのカノニカル名 ($hintsのflagsにAI_CANONNAMEを指定した場合のみ)

IO::Socket::INET

コアモジュール (Perl 5.6.0～)

AF_INET ドメインソケットのオブジェクトインタフェースを提供

→ IPv4 しか扱えない

IO::Socket::INET を使用している多くの CPAN モジュールも IPv6 非対応 (対応方法は後述)

```
# TCPのクライアントの例
use IO::Socket::INET;
:
my $sock = IO::Socket::INET->new(
    PeerAddr => $host,   # 接続先のホスト
    PeerPort => $port,   # ポート番号
    Proto => 'tcp'       # トランスポートにTCPを使う
) or die "cannot create socket: $!¥n";
```

IO::Socket::IP (CPAN)

IPv4 / IPv6 に対応したソケットインタフェース

コンストラクタやメソッドは IO::Socket::INET と互換性がある
(“a drop-in replacement”, 一部例外あり)

```
use IO::Socket::IP;                    ← ここと
:
my $sock = IO::Socket::IP->new(        ← ここを書き換えるだけで
    PeerAddr => $host,                     IO::Socket::INETを使用していた
    PeerPort => $port,                     プログラムがIPv6対応になる(はず)
    Proto => 'tcp'
) or die "cannot create socket: $!¥n";
```

今後, IO::Socket::INET でやっていたことをやりたい場合は,
このモジュールを使いましょう。

Net::DNS (CPAN)

DNS リゾルバ

- IPv6 関連の RR (AAAA, IPv6 アドレスの PTR) は問題なく引ける
- AAAA を引いた結果の文字列表現は, "::" による省略はされない (Net::DNS::RR の print() 等)
- IP アドレスはそのままの形式で逆引きできる (in-addr.arpa., ip6.arpa. 形式にする必要がない)
- IPv6 アドレスを逆引きするときは, "::" で省略したアドレスを渡しても OK

Net::DNS (続き)

```
use Net::DNS;
:
my $r = Net::DNS::Resolver->new;
my $result = $r->search($name, $type);
foreach my $i ($result->answer) {
    print $i->rdatastr, "¥n";
}

# $name = 'internetweek.jp', $type = 'aaaa' の場合
2001:dc2:1000:2006:0:0:c0:ffee

# $name = '2001:dc2:1000:2006::c0:ffee', $type = 'ptr' の場合
internetweek.jp.
```

HTTP::Tiny

シンプルな HTTP クライアント

コアモジュール (Perl 5.13.9～)

IO::Socket::INET を使用しているため, IPv6 非対応

```
# URLの内容を取得して表示する例
use HTTP::Tiny;
:
my $http = HTTP::Tiny->new;
my $response = $http->get($url);
print $response->{content}, "¥n";
```

HTTP::Lite (CPAN)

簡易 HTTP クライアント

IPv6 非対応 (内部で socket() に PF_INET を渡しているため)

```
# URLの内容を取得して表示する例
use HTTP::Lite;
:
my $http = HTTP::Lite->new;
my $result = $http->request($url);
print $http->body();
```

LWP::UserAgent

多機能な HTTP クライアント

IPv6 非対応

```
# URLの内容を取得して表示する例
use LWP::UserAgent;
:
my $ua = LWP::UserAgent->new;
my $response = $ua->get($url);
print $response->decoded_content, "¥n";
```

Net::SMTP

SMTP クライアント (Perl 5.7.3からコアモジュール)

IO::Socket::INET のサブクラスのため, IPv6 非対応

```
use Net::SMTP;
:
my $smtp = Net::SMTP->new($host);
$smtp->mail($from);
$smtp->to($to);
$smtp->data();
$smtp->datasend($mail_str);
$smtp->dataend();
$smtp->quit();
```

- デュアルスタックのホストに対しては IPv4 で接続
- コンストラクタで IPv6 のみのホストを指定するとエラーになる
- localhost を指定した場合は, その先の動作はローカルの MTA に依存する

Net::Ping

ホストへの到達性 (ICMP) やサービス (TCP/UDP) の状態の確認

最新版に付属の Net::Ping 2.38 でも IPv6 非対応

ping() に IPv6 アドレス, AAAA レコードしか持たないホストの名前を渡すとエラーになる

(内部で inet_aton() 等の IPv4 依存関数を使用しているため)

```
# ping (ICMP echo request) を送信する例
use Net::Ping;
:
my $p = Net::Ping->new(“icmp”);
my $result = $p->ping($host); # $hostへの到達性があれば1，なければ0
```

IO::Socket::INET 依存コードの IPv6 対応

IO::Socket::INET に依存しているコードが…

- 自分で修正可能なコードの場合

  → IO::Socket::IP を使うように修正する

- 自分で修正したくないコードの場合 (CPAN モジュール等)

  → Net::INET6Glue を使う

※ もちろん, IPv4 アドレスが直書きしてあるような部分については, 別途修正の必要あり

Net::INET6Glue (CPAN)

使い方

IO::Socket::INET に依存した CPAN モジュール等を使用している既存のコードの先頭で, "use Net::INET6Glue;" するだけ

```
use Net::INET6Glue;  ← これだけ
:
# 既存のコード
:
```

仕組み

IO::Socket::INET6 から IO::Socket::INET へシンボルテーブルをコピーすることで, IO::Socket::INET が IO::Socket::INET6 と同じ動作をするようにしている

(詳細は Net::INET6Glue::INET_is_INET6.pm を参照)

Net::INET6Glue (続き)

LWP や Net::SMTP 等を使用したコードで IPv6 での通信ができるようになったことを確認

```
# HTTP::Tiny (IO::Socket::INET依存) を使用したコードをIPv6に対応させる例
use Net::INET6Glue;                    ← これを追加するだけ
use HTTP::Tiny;
:
my $http = HTTP::Tiny->new;
my $response = $http->get($url);
print $response->{content}, "¥n";
```

既存のコード
(そのままでよい)

Net::IP (CPAN)

IPv4 / IPv6 アドレス処理のための様々な機能を提供

version()

IP のバージョンを返す (4 or 6)

ip()

IPv6 アドレスの場合, 最も冗長な表現を返す
(各コロンの間が4文字, リーディングゼロあり)

short()

できるだけ省略された表記を返す

reverse_ip()

逆引き用の表記を返す

Net::IP (続き)

```
use Net::IP;
my $address = new Net::IP("2001:DB8::0123:0045:0006:07")

print $address->version(), "¥n";
# 6

print $address->ip(), "¥n";
# 2001:0db8:0000:0000:0123:0045:0006:0007

print $address->short(), "¥n";
# 2001:db8::123:45:6:7

print $address->reverse_ip(), "¥n";
# 7.0.0.0.6.0.0.0.5.4.0.0.3.2.1.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa.
```

IPv6 アドレスのアルファベットは小文字に正規化される模様

Python

Python

Python のバージョン

- 2系と3系が存在
- 2系の最新版: 2.7.2 (2011/6/11)
- 3系の最新版: 3.2.2 (2011/9/4)
- 検証に使用したバージョン: 2.7.2, 3.2.2

Python のライブラリ

- 標準で豊富なライブラリが付属
- 外部ライブラリとしては PyPI (Python Package Index) がある
- http://pypi.python.org/pypi

Python で書かれた有名なアプリケーション

- Mailman, Trac, Plone, etc.

Linux でのバージョン

ディストリビューションのパッケージでインストールされるバージョン
(2011/11/20現在)

ディストリビューション		Pythonのバージョン
RHEL	5.7	2.4.3
	6.1	2.6.6
Debian	5.0	2.5.2
	6.0	2.6.6 / 3.1.3
Ubuntu	10.04 LTS	2.6.5 / 3.1.2
	10.10	2.6.6 / 3.1.3
	11.10	2.7.2 / 3.2.2
Fedora	15	2.7.1 / 3.2.1
	16	2.7.2 / 3.2.2

socket

低レベルのソケットインタフェース
IPv6 対応

- socket オブジェクト
  bind(), connect(), listen() 等のメソッド
- getaddrinfo()
  ソケットを作成するのに必要な情報をプロトコル非依存で取得
- inet_pton(), inet_ntop()
  IPv4 / IPv6 アドレスとバイナリ表現の変換
- has_ipv6
  プラットフォームで IPv6 がサポートされているかを表す真偽値 (2.3 以降)

etc.

socket.getaddrinfo()

getaddrinfo(*host*, *port*, *family=0*, *socktype=0*, *proto=0*, *flags=0*)

host: ホスト名, IP アドレス, None

port: サービス名, ポート番号, None

family や proto は結果を限定するのに使う

```
# internetweek.jpにHTTP接続するための情報を取得 (IPv4/IPv6問わず)
socket.getaddrinfo('internetweek.jp', 80, 0, 0, socket.SOL_TCP)
# [(10, 1, 6, '', ('2001:dc2:1000:2006::c0:ffee', 80, 0, 0)),
   (2, 1, 6, '', ('192.41.192.130', 80))]

# internetweek.jpにIPv6でHTTP接続するための情報を取得
socket.getaddrinfo('internetweek.jp', 80, socket.AF_INET6, 0,
   socket.SOL_TCP)
# [(10, 1, 6, '', ('2001:dc2:1000:2006::c0:ffee', 80, 0, 0))]
```

※ Python 2.2以降

urllib

HTTP クライアント (Python 2)

IPv6 対応

```
# URLの内容を取得して表示する例
import urllib
:
f = urllib.urlopen(url)
print(f.read())
```

デュアルスタックのホストに対しては IPv6 で接続した

urllib.request

HTTP クライアント (Python 3)

IPv6 対応

```
# URLの内容を取得して表示する例
import urllib.request
:
f = urllib.request.urlopen(url)
print(f.read().decode(enc))
```

デュアルスタックのホストに対しては IPv6 で接続した

smtplib

SMTP クライアント

IPv6 対応

```
# メールを送信する例
import smtplib
:
server = smtplib.SMTP(host)  # 接続先のホストを指定
server.sendmail(from_addr, to_addr, mail)
server.quit()
```

- デュアルスタックのホストに対しては IPv6 で接続した
- localhost を指定した場合は, その先の動作はローカルの MTA に依存する

PHP

PHP

PHP のバージョン

最新版: 5.3.8 (2011/8/23)

検証に使用したバージョン: 5.3.6

PHP のライブラリ

標準で豊富なライブラリが使用可能

拡張ライブラリとして PEAR (PHP Extension and Application Repository) がある

http://pear.php.net/

PHP で書かれた有名なアプリケーション

MediaWiki, WordPress, XOOPS, etc.

Linux でのバージョン

ディストリビューションのパッケージでインストールされるバージョン
(2011/11/20現在)

ディストリビューション		PHPのバージョン
RHEL	5.7	5.1.6
	6.1	5.3.3
Debian	5.0	5.2.6
	6.0	5.3.3
Ubuntu	10.04 LTS	5.3.2
	10.10	5.3.3
	11.10	5.3.6
Fedora	15	5.3.6
	16	5.3.8

標準ライブラリの DNS

PHP には getaddrinfo(), getnameinfo() はない

gethostbyname(*$hostname*)

ホスト名に対応する IP アドレスを返す

IPv4 アドレスしか返さない

```
echo gethostbyname('internetweek.jp'), "¥n";
// 192.41.192.130
(AAAAの2001:dc2:1000:2006::c0:ffeeは返さない)

echo gethostbyname('ipv6.google.com'), "¥n";
// ipv6.google.com
(失敗した場合は引数の文字列をそのまま返す)
```

結果のリストを配列で返す gethostbynamel() も IPv4 のみ

標準ライブラリの DNS (続き)

gethostbyaddr(*$ip_address*)

IP アドレスに対応するホスト名を返す

名前からは IPv4 依存臭がするが, IPv6 アドレスを渡しても大丈夫だった ("::" による省略も OK)

```
echo gethostbyaddr('192.41.192.130'), "¥n";
// internetweek.jp

echo gethostbyaddr('2001:dc2:1000:2006::c0:ffee'), "¥n";
// internetweek.jp
```

Net_DNS (PEAR)

DNS リゾルバ

- AAAA は引ける
- 逆引きの場合, IPv4 アドレスはそのままで OK だが, IPv6 アドレスは in6.arpa. 形式でないと引けない

```
require 'Net/DNS.php';
:
$r = new Net_DNS_Resolver();
$result = $r->query($name, $type);
if ($result) {
    foreach ($result->answer as $rr) {
        echo $rr->rdatastr(), "¥n";
    }
}
```

file_get_contents()

ファイル / URL の内容を文字列として返す

IPv6 OK

```
// URLの内容を取得して表示する例
$result = file_get_contents($url);
echo $result;
```

デュアルスタックのホストに対しては IPv6 で接続した

cURL

libcurl を使用した各種プロトコルのデータ転送

IPv6 OK

```
// URLの内容を文字列として取得する例
$ch = curl_init($url);
// 取得した内容をブラウザに渡さない
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$result = curl_exec($ch);
:
curl_close($ch);
```

デュアルスタックのホストに対しては IPv6 で接続した

(動作は libcurl 依存と思われる)

mail()

システムの MTA を使用してメールを送信する
(ビルトイン関数)

```
// メール送信の例
:
mail($to, $subject, $message);
```

したがって, 動作はローカルの MTA に依存

Net_SMTP (PEAR)

SMTP クライアント

IPv6 OK

(ただし, システムのポリシーテーブルを無視して IPv4 を優先?)

```
// メール送信の例
require 'Net/SMTP.php';
:
$smtp = new Net_SMTP($host);  // 接続先のホストを指定
$smtp->connect();
$smtp->mailFrom($from);
$smtp->rcptTo($to);
$smtp->data($body);
$smtp->disconnect();
```

- デュアルスタックのホストに対しては IPv4 で接続する模様
- IPv6 アドレスを指定する場合は [ ] で括れば OK

Mail (PEAR)

メール送信

送信に使うバックエンドを以下の3つから選べる

- ビルトイン関数のmail()
- システムの sendmail
- ダイレクト SMTP 接続 (Net_SMTP を使用)

```
// メール送信の例
require 'Mail.php';
:
$params = array('host' => $host);
$m = Mail::factory('smtp', $params);
$m->send($to, $header, $body);
```

したがって, 動作はバックエンドに依存する

Net_IPv6 (PEAR)

IPv6 アドレスの処理

checkIPv6(*$address*)

引数が IPv6 アドレスの表記として正しければ true を返す

```
// '::'が2箇所ある誤った表記
checkIPv6('2001:db8::1::1');
// false
```

compress(*$address*, *$force = false*)

できるだけ省略した表記を返す

```
compress('2001:db8:0:0:0:0:0:1');
// 2001:db8::1

compress('2001:db8:0:0:0123:0045:0006:07');
// 2001:db8::123:45:6:7
```

Net_IPv6 (続き)

uncompress(*$address*, *$leading_zero* = *false*)

省略されたアドレスを冗長な表記にして返す
第2引数を true にすると, リーディングゼロを付ける
(各コロンの間が4文字の固定長になる)

```
// リーディングゼロなし
uncompress('2001:dc2:1000:2006::c0:ffee', false);
// 2001:dc2:1000:2006:0:0:c0:ffee

// リーディングゼロあり
uncompress('2001:dc2:1000:2006::c0:ffee', true);
// 2001:0dc2:1000:2006:0000:0000:00c0:ffee
```

Ruby

Ruby

Ruby のバージョン

- 1.8系と1.9系がある
- 1.8系の最新版: 1.8.7-p352 (2011/7/2)
- 1.9系の最新版: 1.9.3-p0 (2011/10/31)
- 検証に使用したバージョン: 1.8.7-p352, 1.9.2-p290

Ruby のライブラリ

- 標準で豊富な機能を備えたライブラリが付属
- 拡張ライブラリの配布サイトとしては RubyForge 等がある
- http://rubyforge.org/

Ruby で書かれた有名なアプリケーション

- Ruby on Rails

Linux でのバージョン

ディストリビューションのパッケージでインストールされるバージョン
(2011/11/20現在)

ディストリビューション		Rubyのバージョン
RHEL	5.7	1.8.5
	6.1	1.8.7
Debian	5.0	1.8.7 / 1.9.0
	6.0	1.8.7 / 1.9.2
Ubuntu	10.04 LTS	1.8.7 / 1.9.0 / 1.9.1
	10.10	1.8.7 / 1.9.2
	11.10	1.8.7 / 1.9.2
Fedora	15	1.8.7
	16	1.8.7

resolv

Resolv.getaddress(*hostname*)

ホスト名から IP アドレスを検索し, 最初の結果を返す

A / AAAA 両方あるホストを指定 → IPv4 アドレスしか返さない

```
p Resolv.getaddress("internetweek.jp")
# "192.41.192.130"
```

(AAAAを引いた結果である"2001:dc2:1000:2006::c0:ffee"は返さない)

AAAA しかないホストを指定 → IPv6 アドレスを返す

```
p Resolv.getaddress("ipv6.google.com")
# "2404:6800:8002::63"
```

resolv (続き)

Resolv.getaddresses(*hostname*)

ホスト名から IP アドレスを検索し, 結果のリストで返す

```
p Resolv.getaddresses("mixi.jp")
# ["110.44.179.199", "110.44.179.200", "110.44.179.196",
   "110.44.179.197", "110.44.179.198"]
```

A / AAAA 両方あるホストを指定すると, 結果には IPv4 / IPv6 両方のアドレスが含まれる

```
p Resolv.getaddresses("internetweek.jp")
# ["192.41.192.130", "2001:DC2:1000:2006::C0:FFEE"]
```

(IPv6アドレス中のアルファベットは大文字になる)

resolv (続き)

Resolv.getname(*address*)

IP アドレスからホスト名を検索する

```
p Resolv.getname("8.8.8.8")
# "google-public-dns-a.google.com"
```

IPv6 アドレスの逆引きも OK

```
p Resolv.getname("2001:dc2:1000:2006::c0:ffee")
# "internetweek.jp"
```

("::" で省略したアドレスを渡しても大丈夫)

net/http

HTTP クライアント

1.8.7, 1.9.2 とも IPv6 OK

```
# GETして表示
require 'net/http'
:
Net::HTTP.get_print host, path

# URIを使った方法
require 'net/http'
require 'uri'
:
Net::HTTP.get_print URI.parse(url)
```

デュアルスタックのホストに対しては IPv6 で接続した

net/smtp

SMTP クライアント

1.8.7, 1.9.2 とも IPv6 OK

```
# メール送信の例
require 'net/smtp'
:
Net::SMTP.start(host, 25) {|smtp|  # 接続先のホストを指定
    smtp.send_message mail_str, from, to
}
```

デュアルスタックのホストに対しては IPv6 で送信した

ipaddr

IPv4 / IPv6アドレスを扱うためのクラス IPAddr を提供

IPAddr.to_s

IPv6 アドレスの場合, 省略された表記を返す

```
p IPAddr.new('2001:db8::0123:0045:0006:07').to_s
# "2001:db8::123:45:6:7"
```

IPAddr.to_string

IPv6 アドレスの場合, 最も冗長な表記を返す

```
p IPAddr.new('2001:db8::0123:0045:0006:07').to_string
# "2001:0db8:0000:0000:0123:0045:0006:0007"
```

ipaddr (続き)

IPAddr.ipv6?

IPv6 アドレスなら真を返す

```
p IPAddr.new('192.0.2.1').ipv6?
# false
```

IPAddr.reverse

逆引き用の文字列を返す

```
p IPAddr.new('2001:db8::0123:0045:0006:07').reverse
# "7.0.0.0.6.0.0.0.5.4.0.0.3.2.1.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa"
```

Q & A

ありがとうございました