ソケットライブラリの改善
socket library improvement
(too long version, 87 pages)

田中 哲

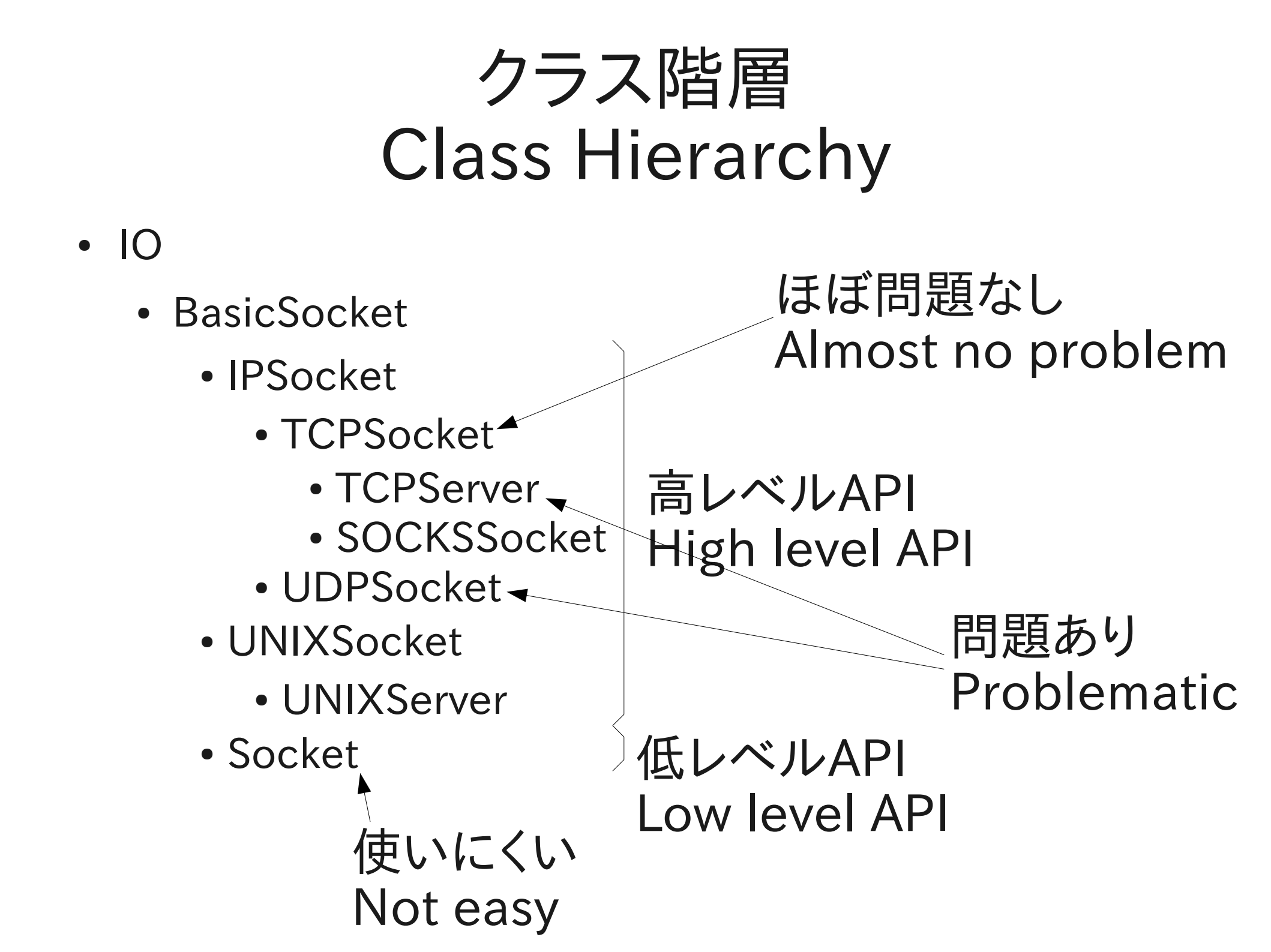
産業技術総合研究所
2009-07-19

ソケットライブラリには不満がある
There are problems in socket library

- 扱いにくい引数・返り値: 冗長な表記やバイナリ
  Difficult arguments/return values: redundant notation, binary, etc.
- TCPServer, UDPSocket はプロトコル依存
  TCPServer and UDPSocket is protocol dependent
- 足りない機能: sendmsg/recvmsg など
  Lacked features: sendmsg/recvmsg, etc.
- TCPSocket はほぼ問題ない
  Almost no problem with TCPSocket

クラス階層
Class Hierarchy

- IO
  - BasicSocket
    - IPSocket
      - TCPSocket
        - TCPServer
        - SOCKSSocket
      - UDPSocket
    - UNIXSocket
      - UNIXServer
    - Socket



問題1: 扱いにくい引数・帰り値
Problem1: Difficult Args/Return

- Socket.new(Socket::AF_INET,
  Socket::SOCK_STREAM, 0)
- UDPSocket.new(Socket::AF_INET6)
- sock.getsockopt(Socket::SOL_SOCKET,
  Socket::SO_LINGER)
  #=> "¥x00¥x00¥x00¥x00¥x00¥x00¥x00¥x00"
- Socket.sockaddr_in(port, host)
  #=>"¥x02¥x00¥x00P¥xDD¥xBA¥xB8D¥x00¥x00..."
- Socket.getaddrinfo("www.ruby-lang.org", "http")
  #=> [["AF_INET", 80, "carbon.ruby-lang.org",
  "221.186.184.68", 2, 1, 6]]

問題2: プロトコル依存
Problem2: Protocol Dependent

- IPv6 のスタイルは、プロトコル非依存
  IPv6 style is protocol independent
- TCPServer.open(9999) は IPv4 を受け付けないことがある
  TCPServer.open(9999) may refuse IPv4
- UDPSocket.new() では IPv4 しか動かない
  UDPSocket.new() works only IPv4
- UDPSocket.new(Socket::AF_INET6) は IPv6 しか動かない
  UDPSocket.new(Socket::AF_INET6) works only IPv6

問題3: 足りない機能
Problem3: Lacked Features

- ホストの IP アドレスを得る
  Obtain the IP addresses of the host
- sendmsg/recvmsg
- getpeereid

どうやって解決するか?
How they are solved?

- 長い定数 → シンボルを受け付ける
  Long constant name → Accept symbols
- わかりにくい値 → inspect を持つ新しいクラス
  Difficult values → New class with inspect
- プロトコル依存 → 新しいAPI
  Protocol dependent → New API
- 足りない機能 → 新しい API
  Lacked Features → New API

扱いにくい引数・返り値: 冗長な表記やバイナリ

長い定数とデフォルトの 0
Long constants and zero as default

- Ruby 1.9.1:
  Socket.new(Socket::AF_INET,
  Socket::SOCK_STREAM, 0)
- C: socket(AF_INET, SOCK_STREAM, 0)
- C より長いのはおかしい
  It is wrong because longer than C
- Ruby 1.9.2: Socket.new(:INET, :STREAM)
- 第3引数はの0はデフォルトの意味なので省略可能
  The 3rd argument 0 is optional because it means default

Design Decision (1)
Constant Prefix

- "Socket::" が冗長なのは明らか
  "Socket::" is cleary redundant
- "AF_" や "SOCK_" も冗長
  "AF_" and "SOCK_" is redundant too
- "AF_" は socket() の第1引数としては冗長
  "AF_" is redandant for 1st arg. of socket()
- "SOCK_" は socket() の第2に引数としては冗長
  "SOCK_" is redundant for 2nd arg. of socket()
- AF v.s. PF の争いに決別する
  No more AF v.s. PF war

いろいろな長い定数
Various long constants

- socket family: AF_xxx
- socket type: SOCK_xxx
- socket protocol: IPPROTO_xxx, etc.
- socket option
  - protocol level: SOL_xxx, IPPROTO_xxx, etc.
  - optoin name: SO_xxx, IP_xxx, etc.
- shutdown: SHUT_xxx
- Socket.new と同様にシンボルを使える
  Symbols are accepted as Socket.new

わかりにくい値
Hard to Understand Values

- Ruby 1.9.1:
  Socket.getaddrinfo("www.ruby-lang.org", "http")
  #=> [["AF_INET", 80, "carbon.ruby-lang.org",
  "221.186.184.68", 2, 1, 6]]
- "2, 1, 6" は意味不明
  "2, 1, 6" is illegible
- Ruby 1.9.2:
  Addrinfo.getaddrinfo("www.ruby-lang.org", "http")
  #=> [#<Addrinfo: 221.186.184.68:80 TCP
  (www.ruby-lang.org:http)>]
- Addrinfo クラスの導入と inspect による表示
  New Addrinfo class and inspect method

Addrinfo

- family + socktype + protocol + sockaddr
- サービスを示すプロトコル非依存なアドレス
  Protocol independent address for services
- C の getaddrinfo() が返す struct addrinfo に対応
  Similar to struct addrinfo returned by getaddrinfo() in C
- 中身を考慮したわかりやすい inspect
  Tailored inspect method
  - ["AF_INET", 80, "carbon.ruby-lang.org", "221.186.184.68", 2, 1, 6] →
  - #<Addrinfo: 221.186.184.68:80 TCP (www.ruby-lang.org:http)>

Addrinfo Example

- TCP/IP
  - a = Addrinfo.tcp("www.ruby-lang.org", "http")
    #=> #<Addrinfo: 221.186.184.68:80 TCP (www.ruby-lang.org:http)>
  - a.ip_address #=> "221.186.184.68"
  - a.ip_port #=> 80
  - a.to_s #=> "¥x02¥x00¥x00P¥xDD¥xBA¥xB8D..."
- UNIX domain stream socket
  - a = Addrinfo.unix("/tmp/s")
    #=> #<Addrinfo: /tmp/s SOCK_STREAM>
  - a.unix_path #=> "/tmp/s"

Addrinfo でソケット生成
Socket creation using Addrinfo

- Addrinfo#connect
  接続したソケットを生成する
  create a connected socket
- Addrinfo#bind
  bind したソケットを生成する
  create a bound socket
- Addrinfo#listen
  listen したソケットを生成する
  create a listening socket

Addrinfo で HTTP
HTTP using Addrinfo

```
Addrinfo.tcp("www.ruby-lang.org",
                    80).connect {|s|
  s.write "GET / HTTP/1.0¥r¥n"
  s.write "Host: www.ruby-lang.org¥r¥n¥r¥n"
  s.close_write
  p s.read
}
```

Design Decision (2)
Addrinfo

- 逆引きしない
  No reverse name lookup
- かわりに正引きで使った名前をコメントとして表示
  Show original name as comment
- Addrinfo を返す getaddrinfo は新規メソッド:
  New method for getaddrinfo for Addrinfo:
  Addrinfo.getaddrinfo
- Socket.getaddrinfo は以前通りで互換
  Socket.getaddrinfo is not changed so compatible

バイナリな値
Binary Values

- struct sockaddr
  Socket.sockaddr_in("http", "www.ruby-lang.org")
  #=> "¥x02¥x00¥x00P¥xDD¥xBA¥xB8D¥x00¥..."
  sock.getsockname
  #=> "¥x02¥x00¥xE2¥xB4¥x00¥x00¥x00¥x00..."
- socket option
  sock.getsockopt(Socket::SOL_SOCKET,
  Socket::SO_LINGER)
  #=> "¥x00¥x00¥x00¥x00¥x00¥x00¥x00¥x00"

sockaddr をバイナリで扱うメソッド
Methods which uses sockaddr as binary

- BasicSocket
  - send
  - getsockname, getpeername
- Socket
  - connect, connect_nonblock
  - accept, accept_nonblock, sysaccept
  - recvfrom, recvfrom_nonblock
  - Socket.{pack,unpack}_sockaddr_{in,un}

その他のバイナリを扱うメソッド
Other methods which use binary

- BasicSocket
  - getsockopt, setsockopt (socket option)
- Socket
  - Socket.gethostbyname (IP address)

バイナリの問題
Binary Problem

- String#inspect は構造を考慮しない
  String#inspect doesn't consider the structure
- 環境によって構造が異なるかもしれない
  structure may differ for each environment
  - struct sockaddr
    sa_len はあるか?
    Does it contain sa_len?
  - SNDTIMEO, RCVTIMEO socket option
    struct timeval
    time_t : 32bit / 64bit
  - little endian / big endian

pack/unpack は難しい
pack/unpack is difficult

- 環境の違いを考慮しなければならない
  It must consider various environments
- ヘッダファイルの構造体定義に従うのが正しい
  The right way is follow the struct definition in header files
- C ならヘッダを include すればよい
  We can include header in C
- Ruby の場合、プログラマが調べる必要がある
  Programmar should examine header in Ruby
- C より Ruby が面倒なのはおかしい
  It is wrong if Ruby is harder than C

バイナリの隠蔽
Hide binary

- バイナリをラップするクラスを定義する
  Define a class which wraps binary
- inspect がわかりやすくなる
  Easy to understand inspect
- pack/unpack が不要になる
  No more pack/unpack
  - pack の代わりにコンストラクタで生成
    constructor instead of pack
  - unpack の代わりにアクセサを使う
    accessor instead of unpack

sockaddr を Addrinfo でラップ
Wrap sockaddr as Addrinfo

struct sockaddr → Addrinfo

```
a = Addrinfo.tcp("www.ruby-lang.org", "http")
#=> #<Addrinfo: 221.186.184.68:80 TCP
(www.ruby-lang.org:http)>
a.ip_address #=> "221.186.184.68"
a.ip_port #=> 80
a.to_s #=> "¥x02¥x00¥x00P¥xDD¥xBA¥xB8D..."
```

Design Decision (3)
sockaddr

- sockaddr の表現に Addrinfo を使う
  Use Addrinfo to represent sockaddr
- Addrinfo は接続するのに便利
  Addrinfo is useful to connect
- Addrinfo = family + socktype + protocol + sockaddr
- family, socktype, protocol を得る方法
  How to obtain family, socktype and protocol
  - コンストラクタの名前
    The name of constructor (e.g. Addrinfo.tcp)
  - address family, SO_TYPE, 0 from socket
  - AF_UNSPEC, 0, 0 if no information

socket option のラップ
Wrap socket option

socket option → Socket::Option

- 1.9.1

```
s = Socket.new(Socket::AF_INET,
                        Socket::SOCK_STREAM, 0)
s.getsockopt(Socket::SOL_SOCKET,
                      Socket::SO_TYPE)
#=> "¥x01¥x00¥x00¥x00"
```

- 1.9.2

```
s = Socket.new(:INET, :STREAM)
o = s.getsockopt(:SOCKET, :TYPE)
#=> #<Socket::Option: INET SOCKET TYPE SOCK_STREAM>
o.to_s #=> "¥x01¥x00¥x00¥x00"
```

Socket::Option

- Socket::Option = family + level + name + value
- 強力な inspect
  powerful inspect
- 生成メソッド
  - Socket::Option.int
  - Socket::Option.bool
  - Socket::Option.linger
  - Socket::Option.new
- アクセサ
  - Socket::Option#int
  - Socket::Option#bool
  - Socket::Optoin#linger

Socket::Option#inspect

family
- AF_INET
- AF_INET6
- AF_UNIX
- ...

protocol level
- SOL_SOCKET
- IPPROTO_IP
- IPPROTO_TCP
- ...

option name
- SO_TYPE
- SO_REUSEADDR
- IP_OPTIONS
- TCP_NODELAY
- ...

#<Socket::Option: INET SOCKET TYPE SOCK_STREAM>

If AF_INET, SOL_SOCKET, SO_TYPE, little endian
"¥x01¥x00¥x00¥x00" = SOCK_STREAM

Design Decision (4)
Socket::Option contents

- わかりやすく inspect 可能?
  Easy to understand inspect possible?
  - value No
  - level + name + value No
  - family + level + name + value Yes
- inspect可能 = 意味がオブジェクト内で決まる
  inspectable = semantics determinable in an object
- 何をオブジェクトにすべきか?
  What should be an object?

Design Decision (5) Socket::Option subclass

- Socket::Option::Linger クラスは作らない
  Don't define Socket::Option::Linger class
- Ruby が知らないオプションがあるかもしれない
  OS may have options which Ruby don't know
- getsockopt の挙動が Ruby のバージョンで変わると旧いので新しいの互換にするのが難しい
  If getsockopt is changed between Ruby versions, it is difficult to emulate new version in old version
- Socket::Option にメソッドがつくだけなら簡単
  The emulation is easy for method addition

Socket.gethostbyname

- Socket.gethostbyname("www.ruby-lang.org")
  #=> ["carbon.ruby-lang.org", [], 2, "¥xDD¥xBA¥xB8D"]
- "¥xDD¥xBA¥xB8D" = 221.186.184.68
- とくに変更しない
  No changes
- Addrinfo.getaddrinfo を使ってください
  Please use Addrinfo.getaddrinfo

引数と返り値
Arguments and Return Values

- バイナリを受け取るメソッドの変更
  Changes to methods which accept binary
  - バイナリに加えて新しいクラスも受け付ける
    Accept new class addition to binary
  - 互換性の問題は起きない
    No compatibility problem
- バイナリを返すメソッドの変更
  Changes to methods which return binary
  - 新しいクラスを返すと非互換なので個々に考える
    Consider each method because incompatible if just return new class

バイナリを受け取るメソッド
Methods which accepts binary

- BasicSocket
  - send
  - setsockopt (socket option)
- Socket
  - connect, connect_nonblock
  - Socket.unpack_sockaddr_{in,un}

Socket#connect(server_sockaddr) →
Socket#connect(server_sockaddr_or_addrinfo)
互換性が保たれている
No compatibility problem

バイナリを返すメソッド
Methods which returns binary

- BasicSocket
  - getsockname
  - getpeername
  - getsockopt (socket option)
- Socket
  - accept, accept_nonblock, sysaccept
  - recvfrom, recvfrom_nonblock
  - Socket.pack_sockaddr_{in,un}
  - Socket.gethostbyname (IP address)

新規メソッドによる非互換性回避
Avoid incompatibility by new methods

- 新しいクラスを返す新しいメソッドをつくる
  Add new methods to return new class
- BasicSocket
  - getsockname → local_address
  - getpeername → remote_address

Design Decision (6) getsockname

- getsockname という名前はローカルなアドレスか、接続先のアドレスか曖昧
  The name, getsockname, is ambiguous between local address and remote (peer) address
- せっかくなので、わかりやすい名前を導入
  Good chance to introduce good name
  getsockname → local_address

返り値を変える非互換な変更
Return New Class, even if Incompatible

- 返り値を変える
  Return value is just changed
- 以前のバイナリ値は to_s で得られる
  to_s can be use to obtain old binary value
- BasicSocket
  - getsockopt (socket option)
- Socket
  - accept, accept_nonblock, sysaccept
  - recvfrom, recvfrom_nonblock

Design Decision (7)
recvfrom

- より良い名前を思いつかない
  No better name found
- いちばん良い名前をいちばん良い挙動にする
  Best name should be best behavior
- あと、Socket はあまり使われていない
  Socket is not used widely

なにも変えない
No Changes at All

- pack は pack
  pack is pack
- Socket.gethostbyname は気にしない
  Don't mind Socket.gethostbyname
- Socket
  - Socket.pack_sockaddr_{in,un}
  - Socket.gethostbyname (IP address)

TCPServer, UDPSocket はプロトコル依存

プロトコル非依存
Protocol Independent

- IPv4 と IPv6 をいっしょに扱う
  Handle IPv4 and IPv6 together
- 概念的には、IPv4, IPv6 以外も扱う
  In general, handle something other than IPv4, IPv6 too
- クライアントは IPv4 と IPv6 の両方を試す
  Clients try both IPv4 and IPv6
- サーバは IPv4 と IPv6 の両方で待つ
  Servers waits connections from IPv4 and IPv6

プロトコル非依存 TCP サーバ
Protocol independent TCP server

- Socket.tcp_server_loop(7777) {|s|
  # s is an accepted socket
  }
- TCPServer.open(7777) はポータブルでない
  TCPServer.open(7777) is not portable
  - 4.4BSD では IPv4 を扱えない
    doesn't work with IPv4 on 4.4BSD
  - Windows XP でも
    On Windows XP too

IPv6 の TCP サーバ
TCP server with IPv6

- IPv6 のソケットは IPv4 を扱えるか?
  Can IPv6 socket handle IPv4?
  - Yes, by default
    - GNU/Linux
    - Mac OS X
    - Solaris
  - No, by default
    - NetBSD
    - FreeBSD
  - No, always
    - OpenBSD
    - Windows XP

セキュリティ上の理由
Because security reason

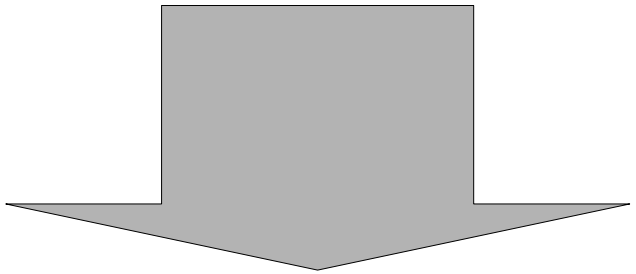
ポータブルな TCP サーバ
Portable TCP server

- IPv4 と IPv6 のふたつのサーバソケットを使う
  Use two server sockets for IPv4 and IPv6
- IPv6 のソケットは IPv4 を扱わないように設定
  Force the IPv6 socket don't handle IPv4
- select() を使って両方で待つ
  Wait the both sockets using select()

TCPServer の問題と解決
TCPServer problem and solution

- TCPServer.open(7777) {|serv|
  loop {
    s = serv.accept
    ...
  }
}

ひとつのサーバソケット
Single server socket



- Socket.tcp_server_loop(7777) {|s|
  ...
}

サーバソケットは隠蔽
Server sockets hidden

Design Decision (8)
TCPServer

- TCPServer オブジェクトが複数の fd を保有するという案もあった
  Another idea is TCPServer object contains multiple file descriptors
- IO.select で扱えないなどの問題がある
  Problems such as IO.select cannot handle TCPServer with multiple fds
- 双方向パイプの経験からも良くない
  It is bad idea from duplex pipe experience

Design Decision (9) tcp_server_loop

- tcp_server_loop は平行処理しない
  tcp_server_loop has no concurrency
- コネクション毎にスレッドを作りたければブロックの中でスレッドを作る
  If thread per connection is desired, start a thread in the block
- スレッドプールを使いたければブロックの中で使う
  If thread pool is desired, use it in the block
- 並行処理はアプリケーション依存なのでしない
  No concurrency because application dependent

UDP ソケットの問題
Problems of UDP socket

- プロトコル非依存な UDP は難しい
  Protocol independent UDP is difficult
- マルチホーム環境で、届いたアドレスから返したい
  Reply packets from received address on multihomed environment

プロトコル非依存な UDP
Protocol independent UDP

- プロトコル非依存な UDP は難しい
  Protocol independent UDP is difficult
  - サーバは IPv4, IPv6 両方で待てば可能
    Server is possible if wait IPv4 and IPv6
  - クライアントはうまくいかない
    Client doesn't work

マルチホーム環境の UDP
UDP on Multihome Environment

- マルチホーム環境で、届いたアドレスから返したい
  Reply packets from received address on multihomed environment
  - IPv4 では IP アドレス毎にソケットを作る
    socket for each IP address
  - IPv6 では recvmsg/sendmsg で IPV6_PKTINFO
    IPV6_PKTINFO using recvmsg/sendmsg for IPv6

プロトコル非依存 UDP サーバ
Protocol independent UDP server

- Socket.udp_server_loop(port) {|msg, msg_src|
  msg_src.reply(msg)
  }
- msg_src は送信元アドレスと宛先アドレスを保持
  msg_src contains source and destination address
- reply メソッドは宛先アドレスから引数を送信する
  reply method sends the arguments from the destination address

Design Decision (10)
UDP server

- Socket.udp_server_loop のブロック引数
  The block arguments of udp_serve_loop
  - {|msg, src_addrinfo, dst_addrinfo| ... }
    IPV6_PKTINFO の情報が一部欠ける
    Can't be complete information of IPV6_PKTINFO
  - {|msg, msg_src| ... }
    msg_src は任意の情報を持てる
    msg_src can contain any information

プロトコル非依存UDPクライアント
Protocol Independent UDP client

- connect できても実際にサーバがあるか不明
  server availability is unknown even if connect succeed
- サーバが存在するかどうか判断するのもアプリケーション依存
  It depends on application to test server availability
- プロトコルを隠蔽できない
  Cannot hide protocol dependency

プロトコル非依存 TCP クライアント
Protocol Independent TCP client

- TCPSocket はほぼ問題ない
  Almost no problem with TCPSocket
- Socket でプロトコル非依存にやるのは面倒
  It is hard to use Socket for TCP with protocol independent style

Socket だとコードが長すぎる
Too Long Code with Socket

TCPSocket

```
require 'socket'

TCPSocket.open(
    ARGV[0],
    ARGV[1]) {|s|
  ...
}
```

Socket

```
require 'socket'

s = nil
info = Socket.getaddrinfo(
  ARGV[0], ARGV[1], nil,
  Socket::SOCK_STREAM)
info.each {|ai|
  begin
    t = Socket.new(ai[4], ai[5], ai[6])
    t.connect(
      Socket.sockaddr_in(ai[1], ai[3]))
    s = t
    break
  ensure
    t.close if !s
  end
}
raise if !s
begin
  ...
ensure
  s.close
end
```

IPv4 のみにするとかなり簡単
Easier if IPv4 only

TCPSocket

```
require 'socket'

TCPSocket.open(
    ARGV[0],
    ARGV[1]) {|s|
  ...
}
```

Socket

```
require 'socket'

Socket.open(
    Socket::AF_INET,
    Socket::SOCK_STREAM,
    0) {|s|
  s.connect(
    Socket.sockaddr_in(
      ARGV[1], ARGV[0]))
  ...
}
```

C のコード
C code

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netdb.h>

int main(int argc, char *argv[])
{
  int s;
  int ret;
  struct addrinfo hints, *res, *a;

  memset(&hints, 0, sizeof(hints));
  hints.ai_socktype = SOCK_STREAM;

  ret = getaddrinfo(argv[1], argv[2], &hints, &res);
  if (ret != 0) exit(1);

  for (a = res; a; a = a->ai_next) {
    s = socket(a->ai_family, a->ai_socktype, a->ai_protocol);
    if (s == -1) { close(s); continue; }

    ret = connect(s, a->ai_addr, a->ai_addrlen);
    if (ret == -1) { close(s); continue; }

    break;
  }
  if (!a) exit(1);

  ...

  close(s);
  return 0;
}
```

コードサイズが IPv4 onlyに誘導する
Code Size Leads IPv4 only

- 空白を除いた文字数
  Number of characters without spaces
  - TCPSocket 54
  - Socket (IPv4 only) 104
  - Socket (protocol independent) 238
  - C 507
- プログラマは小さいコードが好き
  Programmers like small code
- Socket が必要な場合、IPv4 only になりがち
  If Socket is required, programmers prefer IPv4 only

Socket.tcp(host, port)

- Socket が TCPSocket より面倒でなければいけない理由はない
  No reason Socket must be difficult than TCPSocket
- Socket でプロトコル非依存 TCP クライアントを実現するメソッドを用意する:
  New method for protocol independent TCP client with Socket:
  Socket.tcp(host, port)
- TCPSocket.open(host, port) とほぼ同じ
  Similar to TCPSocket.open(host, port)

Design Decision (11)
Socket.tcp

- プログラムの開発後に TCPSocket から Socket に移行するのは厄介
  Difficult to switch TCPSocket to Socket for existing programs
- Socket ですべて済ましたい
  It is good if Socket covers all
- 低レベルと使いやすさは両立できる
  It is possible that single class provides low level and easy to use
- クラスが多すぎて大クラス主義に反する
  Too many classes violates large class principle

足りない機能: sendmsg/recvmsg など

足りない機能
Lacked Features

- ホストの IP アドレスを得る
  Obtain the IP addresses of the host
- sendmsg/recvmsg
- getpeereid

ホストの IP アドレス
IP Addresses of the Host

- Socket.ip_address_list
  #=> [#<Addrinfo: 127.0.0.1>,
  #<Addrinfo: 221.186.184.67>,
  #<Addrinfo: ::1>,
  #<Addrinfo: fe80::211:43ff:fefd:66%eth0>]
- IPv4 の UDP で宛先アドレスを得るのに必要
  Required for destination address of IPv4 UDP
- IPv6 の接続性の判断にも使える
  Usable to determine IPv6 connectivity

IPv4 の UDP で宛先アドレス
Destination Address of IPv4 UDP

マルチホーム環境の UDP サーバ

UDP server on multi homed environment

- IP アドレス毎にソケットを作って bind する
  create and bind for each IP address
- パケットの宛先は到着したソケットから判明する
  The destination of a packet can be determined by socket
- 返事は到着したソケットから行う
  Reply using the socket which message arrives
- 返事の送信元は元のメッセージの宛先になる
  The source of the reply will be the destination of original message

IPv6 の接続性
IPv6 connectivity

IPv6 のグローバルな接続性がなければ、resolv.rb は IPv6 アドレスを返さない

If no global IPv6 connection, resolv.rb doesn't return IPv6 addresses

- グローバルでない IPv6 アドレス:
  Non-global IPv6 address:
  - loop back address ::1
  - link local address fe80::/10
- 上記以外の IPv6 アドレスがあるときのみ IPv6 アドレスを返す
  If any IPv6 address except above, IPv6 address is returned

IP アドレスを得る方法
How to obtain IP addresses

- 残念ながら標準化されていない
  No standard, sigh
- 実装方法
  - getifaddrs: BSD/OS, FreeBSD, NetBSD, OpenBSD, DragonFly BSD, MirOS BSD, GNU/Linux, MacOS X, AIX
  - SIOCGLIFCONF: Solaris
  - SIOCGIFCONF: 4.3BSD (No IPv6)
  - GetAdaptersAddresses: Windows

Design Decision (12) Socket.ip_address_list

- 名前解決やホスト名には依存しない
  Don't depend on name resolution and host name
- ネットワークインターフェースの設定はネットワークインターフェースに尋ねる
  Ask a network interface the network interface configuration

sendmsg/recvmsg

- 補助的なデータを受け渡せる send, recv
  send and recv with ancillary data
- 補助的なデータ:
  ancillary data:
  - rights: file descriptor passing
  - time stamp: When UDP packet arrived?
  - credential: Who is the sender of the packet?
  - destination address
  - IPv6 でいろいろつかわれる
    Extensively used with IPv6

Ruby で sendmsg/recvmsg
sendmsg/recvmsg on Ruby

- Socket::AncillaryData で補助データを表現
  New Socket::AncillayData class
- Addrinfo.udp("0.0.0.0", 9999).bind {|s|
  s.setsockopt(:SOCKET, :TIMESTAMP, 1)
  p s.recvmsg
  }

request packet arrival time

#=>
["a", #<Addrinfo: 127.0.0.1:50309 UDP>, 0,
 #<Socket::AncillaryData: INET SOCKET TIMESTAMP
  2009-07-15 00:50:11.793562>]

packet arrival time

file descriptor passing

- sendmsg/recvmsg により、fd を通信できる
  sendmsg/recvmsg can be use to send fd
- Ruby 1.8.0 から、UNIXSocket で可能
  Possible with UNIXSocket since Ruby 1.8.0
  UNIXSocket#{send_io,recv_io}
- Ruby 1.9.2 からは Socket でも可能
  Possible with Socket since Ruby 1.9.2
  BasicSocket#{sendmsg,recvmsg}
  4.3BSD の古い sendmsg/recvmsg は動かない
  Old 4.3BSD sendmsg/recvmsg doesn't work

file descriptor passing in 1.9.2

- Socket::AncillaryData.int: 補助データ作成 (Create ancillary data)
- Socket::AncillaryData#unix_rights: IO取得 (Obtain IO)
- s1, s2 = Socket.pair(:UNIX, :STREAM)
  s1.sendmsg "a", 0, nil,
    Socket::AncillaryData.int(
      :UNIX, :SOCKET, :RIGHTS, STDIN.fileno)
  msg, src, flags, ctl = s2.recvmsg(:scm_rights=>true)
  p ctl.unix_rights
  #=>
  [#<IO:fd 6>]

IPv6 で到着したアドレスから返事
Reply from dest. address in IPv6

- IPV6_PKTINFO を使う
- Addrinfo.udp("::", 9999).bind {|s|

```
  s.setsockopt(:IPV6, :RECVPKTINFO, 1)
  msg, src, flags, *ctls = s.recvmsg
  p [msg, src, flags, *ctls]
  s.sendmsg(msg, 0, src, *ctls)
}
#=>
["a",
 #<Addrinfo: [::1]:55150 UDP>, 0,
 #<Socket::AncillaryData: INET6 IPV6 PKTINFO ::1 lo>]
```

request pktinfo

receive pktinfo

specify pktinfo

宛先が ::1 のパケットが lo インターフェースに到着
The packet which dest. address is ::1 is arrived to lo interface

プロトコル非依存 UDP サーバ
Protocol Independent UDP Server

- Socket.udp_server_loop の実装:
  The implementation of udp_server_loop
- IPv4 は IP アドレス毎にソケットを作る
  Create a socket for each IP address for IPv4
- IPv6 は IPV6_PKTINFO を使う
  Use IPV6_PKTINFO for IPv6

Design Decision (13) Socket::AncillaryData

- fd を受けとるときは recvmsg にオプションが必要
  recvmsg needs :scm_rights option for receiving fd
  - sock.recvmsg(:scm_rights=>true)
  - オプションが与えられなければ、渡された fd は fd leak を避けるため即座に close される
    passed fd is immediately closed to avoid fd leak if the option is not specified
- 受け取った fd のクラスは IO か Socket になる
  IO or Socket is used as a class of received fd

getpeereid

- Unix domain stream ソケットで、接続した相手の実効 uid, gid を得る
  Obtain effective uid and gid for connected process using Unix domain stream socket
- ホスト内での認証に使う
  Used for authentication in a host
- パスワード不要
  No password
- DJB が提案
  Proposed by DJB

getpeereid の使い方
Usage of getpeereid

- Socket.unix_server_loop("/tmp/sock") {|s|
  p s.getpeereid
  }
  #=>
  [1000, 1000]

getpeereid の実装
getpeereid implementation

- getpeereid: OpenBSD, FreeBSD, NetBSD
- SO_PEERCRED: GNU/Linux
- getpeerucred: Solaris

Design Decision (14) getpeereid

- SO_PEERCRED や getpeerucred で得られる euid, egid 以外の情報は捨てる
  Drop extra information except euid and egid from SO_PEERCRED and getpeerucred
- 認証用なので、バグが出やすい環境依存の挙動は避ける
  Because it intend for authentication, avoid environment dependent behavior which tend to cause bugs

Socket は TCPSocket より使いやすくなったか？

TCPSocket の使いやすさ
Easiness of TCPSocket

- クラスメソッド
  - TCPSocket.open
    プロトコル非依存で短く記述できて素晴らしい
    Very good because protocol independent
    succinct description
- インスタンスメソッド
  - アドレス表現がバイナリでなくわかりやすい
    Address represented in easy format
  - 使えないメソッドが定義されない
    No unusable methods

インスタンスメソッドの使いやすさ Easiness of Instance methods

- アドレス表現がバイナリでなくわかりやすい
  Address represented as easy format
  - IPSocket: ["AF_INET", 46241, "localhost", "127.0.0.1"]
  - Socket: "¥x02¥x00¥x98r¥x7F¥x00¥x00¥x01¥x00¥x00¥x00¥x00¥x00¥x00¥x00¥x00"
  - addr, peeraddr, recvfrom, UNIXSocket#path
- 使えないメソッドが定義されない
  No unusable methods
  - TCPSocket にはサーバ用のメソッドがない
    TCPSocket has no methods for servers
  - listen, accept, accept_nonblock, sysaccept

TCPSocket と Socket
TCPSocket and Socket

- だいたい同じくらい使いやすくなった
  Similar easiness is realized
- TCPSocket.open
  - Socket.tcp
- アドレス表現
  Address representation
  - Addrinfo
- 使えないメソッド
  Unusable methods
  - 定義されますが無視してください
    Defined but please ignore

Design Decision (15)
Unusable Methods

- 常に定義しておく
  Always defined
- モジュールで定義して使用可能なソケットに extend するという案もあった
  Another idea is extending a socket with module which define methods not always usable
- トリッキー過ぎる
  Too tricky

まとめ

改善 (1)
improvement (1)

- 新しいメソッド:
  New methods:

Socket.tcp
Socket.tcp_server_loop
Socket.tcp_server_sockets
Socket.unix
Socket.unix_server_loop
Socket.unix_server_socket
Socket.accept_loop
Socket.ip_address_list

BasicSocket#local_address
BasicSocket#remote_address
BasicSocket#connect_address
BasicSocket#sendmsg
BasicSocket#sendmsg_nonblock
BasicSocket#recvmsg
BasicSocket#recvmsg_nonblock
BasicSocket#getpeereid
Socket#ipv6only!

- 新しいクラス:
  New classes:
  - Addrinfo
  - Socket::Option
  - Socket::AncillaryData

改善 (2)
improvement (2)

- 引数をより柔軟に受けつけるようにした:
  More flexible arguments:
  Socket.new(Socket::AF_INET, Socket::SOCK_STREAM, 0)
  →Socket.new(:INET, :STREAM)
- バイナリをオブジェクト化 (ちょっと非互換):
  Change binary to object (bit incompatible):
  udpsock.recvfrom(100)
  #=>
  ["a", "¥x02¥x00¥x98r¥x7F¥x00¥x00¥x01¥x00¥x00¥x00¥x00¥x00¥x00¥x00¥x00"]
  →
  ["a", #<Addrinfo: 127.0.0.1:39026 UDP>]

まとめ

- Socket は TCPSocket 並に使いやすくなった
- プロトコル非依存性が上がった
- OS のいろんな機能を使えるようになった