American Association for Science and Technology
AASCIT Communications
Volume 1, Issue 3
October 20, 2014 online

Adapting Data for Web Applications That Use IPv6 Internet Protocol

Dănuţ-Octavian Simion

Keywords: Java Custom Methods, Ipv6 Internet Protocol, Java Objects Integration, Open-Source Java Components, UML Diagrams, Web Applications

The paper presents the study cases for using different types of data in the web applications that use IPv6 Internet Protocol. These types of applications may use various sets of data that come from different sources or results from other business applications. A different way of using these data in web applications that use IPv6 Internet Protocol is to apply Java custom methods from a package written by the developer for this case. The data will be transferred from the server side to the numerous clients according to the request made through the web application. The custom Java package that perform transfer of data between server and clients is build according to IPv6 Internet Protocol and makes possible the coded and distributed to a large number of users interconnected in a large network. Using Java programming language to develop this package makes possible the integration of data into any type of web applications and even to include custom methods from specialized drivers that connect user's application to the data stored on the server side.

Introduction

An important role in Web applications has the oriented-object language Java, who can perform tasks as access data from databases managed by SGBD-s as Oracle, Microsoft SQL Server, MySQL, Microsoft Access, Visual Fox, etc. and for semi-structured data the usual language is XML. Java programming language extends the possibilities of the Web servers and is the basic element in Web tools like Servlet, JSP, JSF, Taglibs, Struts, etc. It also is the programming language used for building beans such as CMP Beans (beans which have the state synchronized with elements from databases on which the mapping is made), EJB Beans (beans that are Enterprise JavaBeans used in transactions, user commands, etc.), ADF JavaBeans (beans used in models like ADF – Application Development Framework, for transactions, user interfaces, session beans.

The programming language Java binds the Web interfaces with databases through specific connections drives specific for each SGBD, manages the users sessions, manages the users connections, transmits and treats the request-response events. In the project will be presented also others programming languages used in building Web applications such as: PHP, C#, Perl, etc. the programming languages that make possible the accessing data from databases managed by SGBD-s or semi-structured data which can be found in XML files. The data can be transmitted through network very easily because is packed for IPv6 Internet Protocol packages and ready to use after it is handled by a custom Java package built to work with different types of data and with this new Internet Protocol [3], [4]. There are modern ways of building Web applications through the usage the latest technologies in Web domain, by using the programming language Java inside of this applications and specify different ways of accessing different types of data through Java connections.

The Java Programming Language for IPv6 Internet Protocol

Internet Protocol version 6 (IPv6) is the latest revision of the Internet Protocol (IP), the communications protocol that provides an identification and location system for computers on networks and routes traffic across the Internet. IPv6 was developed by the Internet Engineering Task Force (IETF) to deal with the long-anticipated problem of IPv4 address exhaustion. Internet Protocol IPv6 intends to replace IPv4, which still carries the vast majority of Internet traffic. IPv6 uses a 128-bit address, allowing 2^{128}, or approximately 3.4×10^{38} addresses, as IPv4 uses 32-bit addresses. IPv4 allows only approximately 4.3 billion addresses. The two protocols aren't designed to be interoperable, complicating the transition to IPv6.

IPv6 addresses are represented as eight groups of four hexadecimal digits separated by colons, for example 1875:0gh6:46a9:0968:703g:6a4e:0620:5631, but methods of abbreviation of this full notation exist [4].

IPv6 is an Internet Layer protocol for packet-switched internetworking and provides end-to-end datagram transmission across multiple IP networks, closely adhering to the design principles developed in the previous version of the protocol, Internet Protocol Version 4 (IPv4). IPv6 was first formally described in Internet standard document RFC 2460, published in December 1998. In addition to offering more addresses, IPv6 also implements features not present in IPv4. It simplifies aspects of address assignment (stateless address auto configuration), network renumbering and router announcements when changing network connectivity providers. It simplifies processing of packets by routers by placing the need for packet fragmentation into the end points. The IPv6 subnet size is standardized by fixing the size of the host identifier portion of an address to 64 bits to facilitate an automatic mechanism for forming the host identifier from link layer addressing information (MAC address). Network security was a design requirement of the IPv6 architecture, and included the original specification of IPsec [1], [4].

IPv6 does not specify interoperability features with IPv4, but essentially creates a parallel, independent network. Exchanging traffic between the two networks requires translator gateways or other transition technologies. Multicasting, the transmission of a packet to multiple destinations in a single send operation, is part of the base specification in IPv6. In IPv4 this is an optional although commonly implemented feature. IPv6 multicast addressing shares common features and protocols with IPv4 multicast, but also provides changes and improvements by eliminating the need for certain protocols.

Using Custom Java Package that is Designed for Data in IPv6 Architecture

Java programming language permits building business applications, in special Web applications, regardless of system operations and can access different types of data that can be structured in databases or semi-structured in XML files. Java is a programming language object oriented that offers many possibilities in the domain of accessing data. It is very adaptive, is open-source, can be used with any type of operating system, it can be expand through the creation of new classes, new packages, new libraries by a large community of programmers, it extends the capacities and possibilities of Web servers and also is the most popular and robust programming language for building business applications [2].

The most efficient way to customize applications is to build the own specific Java package that is able to use data in IPv6 architecture and use them in a persistent way through specific methods that are encapsulated in this package. My own solution is to build a custom Java class that transfer different types of data from the server side in IPv6 packages to the client side. The code is designed to work with IPV6 protocol and so the data is prepared for these types of IP packages and ready to for transition over a network.

The source code for the custom package is:

```
  package pckg1;

  public class Server1 {

  private static final Object AF_INET6
= null;
  private static final String PF_INET6
= null;
  private static final String
SOCK_STREAM = null;
  private static final String
IPPROTO_TCP = null;
  private static final String F_SETFL =
null;
  private static final String
NONBLCK = null;
  private static final String
SOL_SOCKET = null;
  private static final String
REUSEADDR = null;
  int BUFLEN=32698;
  int MAXCLIENTS=4096;
  short DEBUG_LEVEL = 0;

  void usage(String argv, String string)

  {
        System.out.println("Error: " +
string);
        System.out.println("Usage: "
+ argv);
  }

  void debug(int i, String argv, String s)
  {
   if (i <= DEBUG_LEVEL)
    {
     System.out.println("s: " + argv + s);
    }
  }
```

The main method:

```
  int main(int argc, String[] argv, char
envp)
  {
    int i, j, rval;
    int sockfd6;
    int nclients, maxfd;
    int clients[]=new
int[MAXCLIENTS];
    struct host_ent;
    struct sockaddr_in6, destipv6;
    socklen_t socklen;
    int so_optval;
    struct servent, srvp;
    int e_save;
    int success;
    char addrlist;
    FDIPV6SET read_fds, write_fds,
except_fds;
    char buf[]=new char[BUFLEN];
    int mlen;
    char s[]=new char[BUFLEN];
    char u[]=new char[BUFLEN];
    time_t t;
    char ts;

    /* Check to see if debug level
specified */
    if (argc == 3)
     {
  if (argv[1].compareTo("3") !=0)
usage(argv[0], "Invalid option.");

     rval=(int) strtol(argv[2], "", 0);
     int errno;
         if (errno != 0)
      {
          System.out.println("Invalid
debug level " + argv[0] + argv[3]);
      }
      else
      {
      DEBUG_LEVEL=(short) rval;
      }
    }
    /* Otherwise, no arguments
expected. */
    else if (argc != 1)
     {
     usage(argv[0], "Incorrect
arguments.");
     }
    /* Create an empty IPv6 socket
interface specification */
    (boolean) memset(destipv6, 0,
sizeof(destipv6));
    /* Set up for IPv6 */
    destipv6.sock6f = AF_INET6;

    /* Choose tcp service */
    if ((srvp = getservbyname("test",
"tcp")) == 0) {
            System.out.println("cannot
find port number for test service." +
argv[0]);
    } else {
    int s_port;
    destipv6.portipv6 = (int) srvp-
>s_port;
    }
    /* Bind to any and all local
addresses */
    Object in6addr_any;
    destipv6.addripv6 = in6addr_any;

    /* Create the sockets */
    if ((sockfd6 = socket(PF_INET6,
SOCK_STREAM, IPPROTO_TCP))
== -1)
    {
     System.out.printf(s, BUFLEN,
"failed to create socket for v6  listener",
argv[0], mlen);
      perror(s);
    }
    System.out.printf(s, BUFLEN,
"Socket created: ", null, sockfd6);
    debug(5, argv[0], s);
   /* Set to non-blocking */
    if (fcntl(sockfd6, F_SETFL,
NONBLCK) < 0)
    {
     System.out.printf(s, BUFLEN,
"could not set v6 nonblocking",argv[0],
mlen);
      perror(s);
    }
      System.out.printf(s, BUFLEN,
"Socket set to v6 non-blocking: ", null,
sockfd6);
    debug(5, argv[0], s);

    /* Mark as re-usable (accept more
than one connection to same socket) */
    so_optval = 1;
    if (setsockopt(sockfd6,
SOL_SOCKET, REUSEADDR, (char)
so_optval,
            sizeof(REUSEADDR)) < 0)
    {
     System.out.printf(s, BUFLEN,
"setsockopt on failed", argv[0],
sockfd6);
     perror(s);
    }

    /* Actually bind the socket to the
port and addresses */

    if (bind(sockfd6, destipv6,
sizeof(destipv6)) == -1)
    {
     System.out.printf(s, BUFLEN,
"bind v6 failed", argv[0], mlen);
     perror(s);
    }

    /* Tell the kernel to listen for new
connections, queue up to 10
connections */
    listen(sockfd6, 10);

    /* Track the highest active file
descriptor number for select */
    Object stdin;
   maxfd = (fileno(stdin) > sockfd6 ?
fileno(stdin) : sockfd6);

    nclients = 0;
    while(1)
    {
     FD_ZERO(read_fds);
     FD_ZERO(write_fds);
     FD_ZERO(except_fds);
     FDIPV6SET(sockfd6, read_fds);
     FDIPV6SET(sockfd6, except_fds);
     for (i = 0; i < nclients; i++)
     {
      System.out.printf(s, BUFLEN,
"FDIPV6SET %d [%d] for read and
      exceptions", i, clients[i]);
      debug(5, argv[0], s);
      FDIPV6SET(clients[i],
&read_fds);
      FDIPV6SET(clients[i],
&except_fds);
     }
     System.out.printf(s, BUFLEN,
"Entering select with maxfd:", maxfd,
     mlen);
     debug(5, argv[0], s);
     /* Wait for someone to do
something */
     select(maxfd + 1, read_fds,
write_fds, except_fds, null);
     /* Process an exception on the
socket itself */
     if (FDIPV6SET(sockfd6,
&except_fds))
     {
      perror("Exception on socket.");
      fprintf(stderr, "Exiting");
      exit(7);
     }

     /* A read event on the socket is a
new connection */
     if (FD_ISSET(sockfd6, read_fds))
     {
      socklen = sizeof(destipv6);
      /* Accept the new connection */
      rval = accept(sockfd6, (struct
sockaddr *) destipv6, socklen);
      if (rval == -1)
      {
       inetipv6ntp(destipv6.sock6f,
destipv6.addripv6.addressv6, buf,
       BUFLEN);
       System.out.printf(s, BUFLEN,
"ipv6 Accept failed for %s %d\0",
           buf, destipv6.portipv6);
       perror(s);
      }
      else
      {
       /* Too many clients? */
       if (nclients == MAXCLIENTS)
       {
        (void) send(rval, "Too many
clients, please try later.\n",
                   strlen("Too many
clients, try later.\n"),
MESSAGE_DNTW);
        close(rval);
       }
       else
       {
        /* Add client to list of clients
*/
        clients[nclients++] = rval;
        if (rval > maxfd) maxfd = rval;
        (void)
inetipv6ntp(destipv6.sock6f,
destipv6.addripv6.addressv6,

                                    buf,
BUFLEN);
        System.out.printf(s, BUFLEN,
"Accepted V6 connection from %s %d
        as %d\n", buf,
destipv6.portipv6, rval);
        debug(1, argv[0], s);
        System.out.printf(s, BUFLEN,
"You are client %d [%d]. Client
       connected.\n\0", nclients, rval);
       send(rval, s, strnlen(s,
BUFLEN), MESSAGE_DNTW);
      }
     }
    }

    /* Check for events from each
client */
    for (i = 0; i < nclients; i++)
    {
     System.out.printf(s, BUFLEN,
"Checking client %d [%d] for read
     indicator.\n",i, clients[i]);
     debug(5, argv[0], s);
     /* Client read events */
     if (FD_ISSET(clients[i],
read_fds))
     {
      System.out.printf(s, BUFLEN,
"Client %d [%d] marked for read.\n",
      i, clients[i]);
      debug(1, argv[0], s);
      /* Read from client */
      if ((rval=recv(clients[i], buf,
BUFLEN-1, MESSAGE_DNTW)) < 1)
      {
       System.out.printf(s, BUFLEN,
"Short recv %d octets from %d
       [%d]\0", rval, i, clients[i]);
       perror(s);
       /* Treat a 0 byte receive as an
exception */
       FDIPV6SET(clients[i],
except_fds);
      }
      buf[rval]=0;
      System.out.printf(s, BUFLEN,
"Received: %d (%d) bytes containing
      %s", rval, strnlen(buf,
BUFLEN), buf);
      debug(5, argv[0], s);
      t=time(NULL);
      ts=ctime(t);
      ts[24]=0;
      System.out.printf(s, BUFLEN,
"Client %d [%d] at %s: %s\0", i,
      clients[i], ts, buf);
      System.out.printf(u, BUFLEN,
"Message Length: %d, %s", strnlen(s,
      BUFLEN), s);
      debug(5, argv[0], u);
      /* Send the message to every
other client */
      for(j=0; j < nclients; j++)
      {
       /* Skip the sender */
       if (j == i) continue;
       /* Send the message */
       send(clients[j], s, strnlen(s,
BUFLEN), MESSAGE_DNTW);
      }
     }
     /* Client eception events */
     if (FD_ISSET(clients[i],
except_fds))
     {
      /* Close the client connection */
      close(clients[i]);
      /* Flag the client as no longer
connected */
      clients[i]=-1;
     }
    }
    /* Remove disconnected clients
from list and recompute maxfd */
    maxfd = fileno(stdin);
    if (sockfd6 > maxfd) maxfd =
sockfd6;
    /* Iterate through and condense
list of clients */
    for(i=0; i < nclients; i++)
    {
     if (clients[i] == -1)
     {
      System.out.printf(s, BUFLEN,
"Client: %d Removed.\n", i);
      debug(1, argv[0], s);
      for(j=i; j < nclients-1; j++)
      {
       clients[j]=clients[j+1];
      }
      nclients--;
     }
     if (clients[i] > maxfd) maxfd =
clients[i];
     System.out.printf(s, BUFLEN,
"End of loop %d / %d (%d)\n", i,
     nclients, maxfd);
     debug(5, argv[0], s);
    }
    System.out.printf(s, BUFLEN,
"Finished removal loop
(maxfd: %d).\n",
    maxfd);
    debug(3, argv[0], s);
   }
   exit(0);
  }
```

The source code for the client side:

```
  public class Client1 {

  private static final Object
PF_UNSPEC = null;
  private static final Object
SOCK_STREAM = null;
  private static final Object
IPPROTO_TCP = null;
  private static final String
MSG_DONTWAIT = null;
  int BUFLEN=32698;
  short DEBUG_LEVEL=0;

  int strnlen(char[] buf, int len)
  {
   int i;
   for(i=0; i<len && i< (buf.length+i);
i++);
   return i;
  }

  void usage(String argv, String string)
  {
   System.out.println("Error: "+
string);
   System.out.println("Usage:
<server>" + argv);
  }

  void debug(int i, String argv, char[] s)
  {
   if (i <= DEBUG_LEVEL)
   {
     System.out.println("%s: " +
argv + s);
   }
  }

  int main(int argc, String[] argv, char
envp)
  {
      int rval, sockfd6;
      struct addrinfo;
      struct res, r;
      struct host_ent;
      int e_save;
      int success;
      char addrlist;
      fd_set read_fds, write_fds,
except_fds;
      char buf[]=new
char[BUFLEN];
      char s[]=new char[BUFLEN];
      int mlen;
      boolean rr;
      int errno;

      if (argc == 4)
       {
        if
(argv[2].compareTo("3") !=0)
usage(argv[0], "Invalid
        option.");
        rval=(int) strtol(argv[3], "",
0);
        if (errno != 0)
        {

      System.out.println("Invalid
debug level " + argv[0] + argv[3]);
        }
        else
        {
         DEBUG_LEVEL=(short)
rval;
        }
       }
       else if (argc != 2)
       {
        usage(argv[0], "Incorrect
arguments.");
       }
       /* Get address info for
specified host and demo service */
        memset(addrinfo, 0,
sizeof(addrinfo));

addrinfo.ai_family=PF_UNSPEC;

addrinfo.ai_socktype=SOCK_STREA
M;

addrinfo.ai_protocol=IPPROTO_TCP;
      if (rval = getaddrinfo(argv[1],
"text", addrinfo, res) != 0) {
       System.out.println("Failed to
resolve address information." +
    argv[0]);
       }

      int ai_addr;
      struct sockaddr;
      for (r=res; r; r = (int)r-
>ai_next) {
              int ai_family;
              int ai_socktype;
              int ai_protocol;
              sockfd6 =socket(r-
>ai_family, r->ai_socktype, (int)r
         ->ai_protocol);
              int ai_addrlen;
                 if
(connect(sockfd6, r->ai_addr, r-
>ai_addrlen) < 0)
              {
               e_save = errno;
               close(sockfd6);
               errno = e_save;
System.out.println("Failed attempt to "
+ argv[0], get_ip_str((struct
sockaddr)r->ai_addr, buf, BUFLEN));
 System.out.println("Socket error");
              } else {
 System.out.println(s, BUFLEN,
"Succeeded to ",
argv[0],get_ip_str((struct sockaddr)r-
>ai_addr, buf, BUFLEN));
 debug(5, argv[0], s);
 success++;
 break;
  }
  }
  if (success == 0)
  {
 System.out.println("Failed to
connect to " + argv[0] + argv[1]);
 freeaddrinfo(res);
 }
 printf("%s: Successfully connected
to %s at %s on FD %d.\n",
argv[0],argv[1],get_ip_str((struct
sockaddr)r
 ->ai_addr, buf, BUFLEN), sockfd6);
  freeaddrinfo(res);
  while(rr)
  {
   FD_ZERO(read_fds);
   FD_ZERO(write_fds);
   FD_ZERO(except_fds);
   Object stdin;
   FD_SET(fileno(stdin), read_fds);

FD_SET(fileno(stdin),except_fds);
     FD_SET(sockfd6, read_fds);
     FD_SET(sockfd6, except_fds);
     select(fileno(stdin) > sockfd6 ?
fileno(stdin)+1 : sockfd6+1, read_fds,
write_fds, except_fds, "");
 if (FD_SET(fileno(stdin),
except_fds))
              {
 if ((Boolean) sizeof(stdin))
  {
  close(sockfd6);
 System.out.println("End of file
detected, exiting");
  }
 else
 {
 System.out.println("Exception on
STDIN");
 System.out.println("Exiting.");
  }
 }
 if (FD_SET(sockfd6, except_fds))
  {
 System.out.println("Exception on
socket.");
 System.out.println("Exiting.");
 }
 if (FD_SET(sockfd6, read_fds))
  {
  /* Read from socket and display to
user */
 mlen = recv(sockfd6, buf, BUFLEN-
1, MSG_DONTWAIT);
 buf[mlen]=0;
 if (mlen == 0)
 {
 System.out.println("Remote site was
not found.");
 }
  else
  {
  System.out.println("Received " +
mlen + "bytes: " + buf);
  }
 }
  if (FD_SET(fileno(stdin), read_fds))
  {
  fgets(buf, BUFLEN, stdin);
  String size_t;

System.out.println(BUFLEN + "Sent "
+ send(sockfd6, buf,
       (size_t) + "octets to
server." + strnlen(buf, BUFLEN), 0));
             debug(5,argv[0],
s);
    }
   }
 }
```

In the code above there are methods encapsulated in a Java package designed for IPv6 architecture used to transfer data of different types that are coming from web applications.

The advantage of using a custom package for manipulating data is that the programmer has the full control over the design and implementation of the applications, with little changes can access every type of data and also this package can be improved and used by a large number of software developers because is a Java class and so is open-source. In the source code of this package could be implemented various methods for managing data in databases (select, insert, update, delete, create, etc.) or from XML files [2], [5].

Using a custom package for manipulating data (own class) has many advantages like accessing different types of data, create the own objects, manage various structures of data including XML files and because this package is built with Java programming language makes it open-source, easy to customize by other software developers and can be improved by implementing different methods and classes.

In IPv6, the packet header and the process of packet forwarding are simplified. Although IPv6 packet headers are at least twice the size of IPv4 packet headers, packet processing by routers is generally more efficient, thereby extending the end-to-end principle of Internet design [1], [4].

Conclusions

The Java programming language allow to develop custom package (own package) that is design to work with data specific to IPv6 Internet Protocol. This network protocol is different from the old protocol IPv4 Internet Protocol, so the web applications have to be adapted to work with these types of data through a custom Java package that is platform free.

The manipulation of data is an important aspect because the programmer can use data from different types of sources and is a very important task to integrate them [2], [3]. The open-source code available in a custom package makes possible the implementation of different methods very easy and makes possible the customization afterwards by other groups of programmers and so the web application becomes open-source applications.

The custom package presented above can be used in different types of enterprise business applications, also may be improved by other programmers through the implementation of their own classes and methods and also may extend the current facilities of the existing methods for working with data [2], [5]. The transition to the IPv6 Internet Protocol impose rebuilding and rethinking of many web application in an open-source direction through object-oriented programming language that is free from existing platforms.■

Dănuţ-Octavian Simion
Business Administration, University of South East Europe Lumina, Bucharest, Romania
danut_so@yahoo.com

References

[1] David Gallardo, “Java design patterns 101”, ibm.com/developerWorks, 2011.

[2] Dănuţ-Octavian Simion, “Using Java in Business Applications”, WSEAS Conferences in the University Politehnica, Bucharest, Romania, pp. 218–223, April 20-22, 2010, Conference/Session: ECC_COMPUTING, ISBN: 978-960-474-178-6, ISSN: 1790-5117.

[3] Owen DeLong, “Porting IPv4 applications to IPv4/v6 dual stack”, Hurricane Electric, 2011.

[4] Owen DeLong, “Essential IPv6 for the Linux Systems Administrator”, Hurricane Electric, 2010.

[5] James W. Cooper, “Java Design Patterns At a Glance”, www.javacamp.org/designPattern/ 2010.

[6] URI: http://javaworld.com/javaworld/

[7] URI:http://www.packtpub.com/service-oriented-java-business-integration/

[8] URI: http://www.sun.com/software/javaenterprisesystem/

[9] URI: http://www.manageability.org/

[10] URI: http://www.javarules.org/