Distributed Systems
Lecture 7

Client-side Sockets

Joseph Phillips

Last modified 2019 October 22

Reading

- Chapter 8 “*Sockets for Clients*” of Elliotte Rusty Harold “*Java Network Programming: 4th Ed.*”

Topics

- Client-side sockets in Java
- Client-side sockets in C

Motivation

The Java URLConnection class and the C Curl library handle many useful things for us including handling:

- https certificate authentication
- cookies
- caching (for Java)

But what if we are not communicating with a web server?

Answer: "***Gotta go old school and communicate by sockets!***"

Sockets!

- Duplex!
  - Can both read and write to same object
- Use ports
  - An integer from 1..65535
  - Acts like a “mailbox”

Java: the Socket class

- Constructors:
  - Socket (String host, int port) throws UnknownHostException, IOException
  - Socket (InetAddress host, int port) throws IOException
- Accessors:
  - public OutputStream getOutputStream()
  - public InputStream getInputStream()
  - public InetAddress getInetAddress() // Remote Internet address
  - public int getPort() // Remote port
  - public InetAddress getLocalInetAddress() // Local Internet address
  - public int getLocalPort() // Local port

HostnamePort.java

```
/*-------------------------------------------------------------------------*
 *---      ---*
 *---      HostnamePort.java  ---*
 *---      ---*
 *---      This file defines a class that interprets hostnames and   ---*
 *---  ports given in a command line argument as:       ---*
 *---      host:port  ---*
 *---      host ---*
 *---  or that asks the user to provide one or both from System.in.
---*
 *---      ---*
 *---  ----   ----   ----   ----   ----   ----   ----   ----   ---*
 *---      ---*
 *---  Version 1a   2019 May 12   Joseph Phillips---*
 *---      ---*
 *-------------------------------------------------------------------------*/

import java.io.*;

public class HostnamePort
{
  public static final int   BAD_PORT = -1;
  public static final int   LO_LEGAL_PORT   = 1;
  public static final int   HI_LEGAL_PORT    = 65535;
  public static final String  DEFAULT_HOSTNAME= "localhost";
  private          String hostname_;
  private          int   port_;
```

```
  public String   getHostname   ()
  { return(hostname_); }

  public intgetPort ()
  { return(port_); }

  public boolean isLegalPortNum (int   port
    )
  { return( (port >= LO_LEGAL_PORT) && (port <=
HI_LEGAL_PORT) ); }



  public HostnamePort (String  arg0,
 int   defaultPort
)
throws NumberFormatException,IOException
  {
    {
      hostname_= "";
      port_  = BAD_PORT;

      if  (arg0 != null)
      {
String[] components = arg0.trim().split(":");

hostname_  = components[0];
```

HostnamePort.java, cont'd

```
if  (components.length > 1)
{
  port_  = Integer.parseInt(components[1]);
}
      }

      String text;
      InputStream   in = System.in;
      BufferedReader reader  = new BufferedReader
(new InputStreamReader(in,"UTF-8"));

      if  (hostname_.equals(""))
      {
System.out.print("Hostname[" + DEFAULT_HOSTNAME + "]? ");
hostname_  = reader.readLine();

if  (hostname_.equals(""))
{
  hostname_= DEFAULT_HOSTNAME;
}
      }

      while  ( !isLegalPortNum(port_) )
      {
try
{
  System.out.print
("Port ("    +
 LO_LEGAL_PORT +
 "-"    +
 HI_LEGAL_PORT   +
 ")"
);

  if  ( isLegalPortNum(defaultPort) )
  {
    System.out.print("[" + defaultPort + "]");
  }

  System.out.print("? ");

  text= reader.readLine();
  port_  = ( text.equals("") && isLegalPortNum(defaultPort) )
  ? defaultPort
  : Integer.parseInt(text);
}
catch  (NumberFormatException ex)
{
  port_  = BAD_PORT;
}
      }

    }
  }

}
```

SocketClient.java

```
/*-------------------------------------------------------------------------*
 *---  ---*
 *---  SocketClient.java ---*
 *---  ---*
 *---      This file defines a class with main() that connects to a---*
 *---  host, sends it a single line of text obtained from System.in, ---
*
 *---  and outputs the response returned by the server.    ---*
 *---  ---*
 *---  ----    ----    ----    ----    ----    ----    ----    ----    ---*
 *---  ---*
 *---  Version 1a    2019 May 12    Joseph Phillips ---*
 *---  ---*
 *-------------------------------------------------------------------------*/

import java.net.*;
import java.io.*;

public class SocketClient
{
  public static final int    TIMEOUT     = 15000;
  public static final int    DEFAULT_PORT = 20001;

  public static void main (String[] args)
  {
    Socket socket  = null;
    HostnamePort  hostnamePort  = null;

    try
    {
      hostnamePort= new HostnamePort
( ( (args.length<1) ? null : args[0] ),
  DEFAULT_PORT
);
      socket        = new Socket
              (hostnamePort.getHostname(),
 hostnamePort.getPort()
);
      InputStream   in = System.in;
      BufferedReader reader  = new BufferedReader
(new InputStreamReader(in,"UTF-8"));
      OutputStream out;
      Writer     writer;
      String     text;

      socket.setSoTimeout(TIMEOUT);

      System.out.print("Text? ");
      text = reader.readLine();
      out = socket.getOutputStream();
      writer  = new BufferedWriter(new OutputStreamWriter(out,"UTF-8"));

      in    = socket.getInputStream();
      reader    = new BufferedReader(new InputStreamReader(in,"UTF-8"));

      writer.write(text + "\r\n");
      writer.flush();
      text = reader.readLine();
      System.out.println("Server response: " + text);
    }
```

SocketClient.java, cont’d

```
    catch  (NumberFormatException ex)
    {
      System.err.println("Bad port number.");
    }
    catch  (IOException ex)
    {
      System.err.println(ex);
    }
    finally
    {
      // dispose
      if  (socket != null)
      {
try
{
  socket.close();
}
catch  (IOException ex)
{
  //  ignore
}
      }
    }
  }
}
```

Note the usage of getInputStream() and getOutputStream()

```
out   = socket.getOutputStream();
writer   = new BufferedWriter(new
OutputStreamWriter(out,"UTF-8"));

in = socket.getInputStream();
reader  = new
BufferedReader(new
InputStreamReader(in,"UTF-8"));

writer.write(text + "\r\n");
writer.flush();
String text    = reader.readLine();
System.out.println("Server
response: " + text);
```

- BufferedReader/Writer
  - Buffer for efficiency
  - Tell them the charset
  - flush() writer
  - readLine() reader

More class Socket goodies

- public void setTcpNoDelay (boolean on) throws SocketException
- public boolean getTcpNoDelay () throws SocketException
  - Ensures packets sent as soon as possible
- public void setSoLinger (boolean on, int seconds) throws SocketException
- public int getSoLinger () throws SocketException
  - If seconds > 0 and data yet be sent exists on close(), then close() blocks for specified seconds while try to send data

More class Socket goodies

- public void setSoTimeout (int milliseconds) throws SocketException
- public int getSoTimeout () throws SocketException
  - How long to wait for read()
  - milliseconds == 0 means “Wait forever”
  - After time expires throws InterruptedIOException
    - Prepare to catch it!
    - Socket still open, next read() might succeed

More class Socket goodies

- public void setReceiveBufferSize (int size) throws SocketException, IllegalArgumentException
- public int getReceiveBufferSize () throws SocketException
- public void setSendBufferSize (int size) throws SocketException, IllegalArgumentException
- public int getSendBufferSize () throws SocketException
  - Suggests how big to make buffers
  - Must be > 0.
  - Author states that nowadays 128 kbytes is common

More class Socket goodies

- public void setKeepAlive (boolean on) throws SocketException
- public boolean getKeepAlive () throws SocketException
  - Send packet every few minutes to keep socket alive?
  - false by default
- public void setOOBInline (boolean on) throws SocketException
- public boolean getOOBInline () throws SocketException
  - Receive urgent data (inline with ordinary data)?
  - false by default (ignores urgent data)
- public void setReuseAddress (boolean on) throws SocketException
- public boolean getReuseAddress () throws SocketException
  - Allow another socket to bind to same port immediately after close()?
  - false by default

Oops, we need a server, don’t we?

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>//For socket()
#include <netinet/in.h>//For sockaddr_in and htons()
#include <netdb.h> //For getaddrinfo()
#include <errno.h> //For errno var
#include <sys/stat.h> //For open(), read(),write()
#include <fcntl.h> // and close()
#include <signal.h>
#include <wait.h>

const int   BUFFER_LEN      = 256;
const int   DEFAULT_PORT_NUM   = 20001;

void  handleClient (int    fd)
{
  char    buffer[BUFFER_LEN];
  int   numRead = read(fd,buffer,BUFFER_LEN);
  int   i;

  printf("Received: %s",buffer);

  for  (i = 0;  i < numRead;  i++)
    buffer[i] = toupper(buffer[i]);

  printf("Sending:  %s",buffer);
  write(fd,buffer,numRead);
  close(fd);
}
```

```
void  sigChildHandler   (int   sig)
{
  int   status;
  pid_t   childId;

  while ( (childId = waitpid(-1,&status,WNOHANG)) > 0)
  {
    printf("Child %d finished.\n",childId);
  }
}


void  installSigChildHandler  ()
{
  // Set up struct to specify the new action.
  struct sigaction act;
  memset(&act,'\0',sizeof(struct sigaction));
  sigemptyset(&act.sa_mask);
  act.sa_flags = SA_NOCLDSTOP | SA_RESTART;;
  act.sa_handler = sigChildHandler;
  // Handle with simpleHandler()
  sigaction(SIGCHLD,&act,NULL);
}
```

uppercaseServer.c, cont’d

```
int main()
{
  int   port;
  char    buffer[BUFFER_LEN];

  printf("Please enter port number to monopolize [%d]: ",DEFAULT_PORT_NUM);
  fgets(buffer,BUFFER_LEN,stdin);

  if  ( (buffer[0] == '\0')  ||  (buffer[0] == '\n') )
    port = DEFAULT_PORT_NUM;
  else
    port = strtol(buffer,NULL,10);

  // Create a socket
  int socketDescriptor = socket(AF_INET, // AF_INET domain
        SOCK_STREAM, // Reliable TCP
        0);

  // We'll fill in this datastruct
  struct sockaddr_in socketInfo;
  // Fill socketInfo with 0's
  memset(&socketInfo,'\0',sizeof(socketInfo));

  // Use std TCP/IP
  socketInfo.sin_family = AF_INET;

  // Tell port in network endian with htons()
  socketInfo.sin_port = htons(port);

  // Allow machine to connect to this service
  socketInfo.sin_addr.s_addr = INADDR_ANY;

  // Try to bind socket with port and other specifications
  int status = bind(socketDescriptor, // from socket()
    (struct sockaddr*)&socketInfo,
    sizeof(socketInfo)
   );

  if  (status < 0)
  {
    fprintf(stderr,"Could not bind to port %d\n",port);
    exit(EXIT_FAILURE);
  }

  listen(socketDescriptor,5);
  installSigChildHandler();

  while (1)
  {
    // Accept connection to client
    int clientDescriptor = accept(socketDescriptor,NULL,NULL);

    if  (fork() == 0)
    {
      close(socketDescriptor);
      handleClient(clientDescriptor);
      exit(EXIT_SUCCESS);
    }

    close(clientDescriptor);
  }

  return(EXIT_SUCCESS);
}
```

Client-side sockets C

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>//For socket()
#include <netinet/in.h>//For sockaddr_in and htons()
#include <netdb.h> //For getaddrinfo()
#include <errno.h> //For errno var
#include <sys/stat.h> //For open(), read(),write()
#include <fcntl.h> // and close()

const int   BUFFER_LEN     = 256;
const int   DEFAULT_PORT_NUM   = 20001;
#define DEFAULT_HOSTNAME  "localhost"


int openSocketToServer
(const char* hostname,
 int   port
)
{
  // Create a socket
  int socketDescriptor = socket(AF_INET, // AF_INET domain
SOCK_STREAM, // Reliable TCP
0);

  struct addrinfo* hostPtr;
  int status = getaddrinfo(hostname,NULL,NULL,&hostPtr);

  if (status != 0)
  {
    fprintf(stderr,gai_strerror(status));
    exit(EXIT_FAILURE);
  }

  // Connect to server
  struct sockaddr_in server;
  // Clear server datastruct
  memset(&server, 0, sizeof(server));

  // Use TCP/IP
  server.sin_family = AF_INET;

  // Tell port # in proper network byte order
  server.sin_port = htons(port);

  // Copy connectivity info from info on server ("hostPtr")
  server.sin_addr.s_addr =
  ((struct sockaddr_in*)hostPtr->ai_addr)->sin_addr.s_addr;

  status = connect(socketDescriptor,(struct
sockaddr*)&server,sizeof(server));
```

Client-side sockets C, cont’d

```
  if  (status < 0)
  {
    fprintf(stderr,"Could not connect %s:%d\n",hostname,port);
    return(EXIT_FAILURE);
  }

  freeaddrinfo(hostPtr);
  return(socketDescriptor);
}


int main ()
{
  char   buffer[BUFFER_LEN];
  char   hostname[BUFFER_LEN];
  int   port;

  printf("Machine name [%s]? ",DEFAULT_HOSTNAME);
  fgets(hostname,BUFFER_LEN,stdin);

  char*  cPtr  = strchr(hostname,'\n');

  if  (cPtr != NULL)
    *cPtr = '\0';

  if  (hostname[0] == '\0')
    strncpy(hostname,DEFAULT_HOSTNAME,BUFFER_LEN);

  printf("Port number [%d]? ",DEFAULT_PORT_NUM);
  fgets(buffer,BUFFER_LEN,stdin);

  if  ( (buffer[0] == '\0')  ||  (buffer[0] == '\n') )
    port = DEFAULT_PORT_NUM;
  else
    port = strtol(buffer,NULL,10);

  int socketDescriptor = openSocketToServer(hostname,port);

  if  (socketDescriptor < 0)
    exit(EXIT_FAILURE);

  printf("Please enter a string to send: ");
  fgets(buffer,BUFFER_LEN,stdin);
  write(socketDescriptor,buffer,BUFFER_LEN);
  read (socketDescriptor,buffer,BUFFER_LEN);

  printf("%s\n",buffer);
  close(socketDescriptor);
  return(EXIT_SUCCESS);
}
```

Let’s look at the steps: socket()

```
  // Create a socket
  int socketDescriptor =
socket(AF_INET,SOCK_STREAM,0);
```

- Asks OS for socket file descriptor
  - SOCK_STREAM for TCP
  - socket(AF_INET, SOCK_DGRAM, 0) for UDP

Let’s look at the steps: getaddrinfo()

```
struct addrinfo* hostPtr;
int status = getaddrinfo(hostname,NULL,NULL,&hostPtr);

if (status != 0)
{
  fprintf(stderr,gai_strerror(status));
  exit(EXIT_FAILURE);
}
```

- Sets ‘hostPtr’ to linked list of ‘hostname’ allows connection
- Sets to ‘NULL’ if cannot connect
- Remember to ‘freeaddrinfo(hostPtr)’ when finished

Let’s look at the steps: struct sockaddr_in

```
// Connect to server
struct sockaddr_in server;
memset(&server, 0, sizeof(server));     // Clear server datastruct

server.sin_family = AF_INET;   // Use TCP/IP

server.sin_port = htons(port);     // Tell port # in proper network byte order

server.sin_addr.s_addr =
((struct sockaddr_in*)hostPtr->ai_addr)->sin_addr.s_addr;    // Copy connectivity info from info on server ("hostPtr")
```

- Choose how to connect:
  - sin_family: Protocol (TCP vs UDP)
  - port: which port (network Endian!)
  - sin_addr.s_addr: IP address (here use first member of linked list)

Let’s look at the steps: connect(), freeaddrinfo()

```
status = connect(socketDescriptor,(struct sockaddr*)&server,sizeof(server));

if  (status < 0)
{
  fprintf(stderr,"Could not connect %s:%d\n",hostname,port);
  return(EXIT_FAILURE);
}

freeaddrinfo(hostPtr);
return(socketDescriptor);
```

- Try to connect to server
- freeaddrinfo() list when finished

Your turn

- Make a Java client that write()s two integers to a server that adds them
- Use:

```
DataOutputStream out;
try
{
  out = new DataOutputStream
           (sock.getOutputStream());
}
catch (IOException e)
{
   // Bail out
}
out.writeInt(5);

DataInputStream  readInt() // reads int
```

- Make a C client that write()s two integers to a server that adds them
- Use:
  - int htonl(int hostInt) convert ‘hostInt’ from local int to network int
  - int ntohl(int netInt) convert ‘netInt’ from network int to local int

References:

- Elliotte Rusty Harold “*Java Network Programming: 4th Ed.*”