CS 580 Client-Server Programming  
Fall Semester, 2000  
Doc 7 Simple Java Client  

Contents  

Building a Connection ........................................................................................................... 2  
java.net.Socket.......................................................................................................................... 5  
   java.net.Socket Methods...................................................................................................... 6  
Domain Name Service (DNS) ................................................................................................. 7  
Closing Sockets........................................................................................................................ 11  
Socket Options......................................................................................................................... 13  
   TCP_NODELAY..................................................................................................................... 13  
   SO_LINGER .......................................................................................................................... 14  
   SO_TIMEOUT...................................................................................................................... 15  
   SO_RCVBUF & SO_SNDBUF................................................................................................. 16  
   SO_KEEPALIVE.................................................................................................................... 16  

References  

Dr. Vinge's Java Networking lecture, Spring 2000, http://www-rohan.sdsu.edu/faculty/vinge/courses/spring00/cs580/notes/jav anet/javanet.html  

Java Network Programming 2nd Edition, Harold, 2000, O'Reilly, Chapter 10  

Java.net.Socket, JDK 1.3,  
http://java.sun.com/j2se/1.3/docs/api/java/net/Socket.html
Building a Connection

Transport layer association is uniquely defined by the following:

- The protocol (TCP or UDP)
- The address of the local machine (the one building the connection)
- The port number used by the local machine
- The address of the remote machine (the one we are connecting to)
- The port number used on the remote machine

Identify the parts:

```
$ telnet sdsu.edu 13
Trying 130.191.229.14...
Connected to sdsu.edu.
Escape character is '^]'.
Thu Feb  3 22:30:35 2000
Connection closed by foreign host.
$```

IP Connecting Parts

Information a client normally needs:

<table>
<thead>
<tr>
<th>Information</th>
<th>Data from example</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>TCP (implied by the telnet program)</td>
</tr>
<tr>
<td>name of the server machine</td>
<td>sdsu.edu</td>
</tr>
<tr>
<td>port number on the server machine</td>
<td>13</td>
</tr>
</tbody>
</table>

The missing parts:

<table>
<thead>
<tr>
<th>Information</th>
<th>Where does it come from?</th>
</tr>
</thead>
<tbody>
<tr>
<td>address of the local machine</td>
<td>Automatically retrieved from the OS</td>
</tr>
<tr>
<td>port number on the local machine</td>
<td>OS supplies random open port</td>
</tr>
</tbody>
</table>
Simple Java Client

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

class GetTime {
    public static void main(String[] args) {
        String hostName;
        Socket client = null;

        if ( args.length != 1 ) {
            System.out.println("Usage: java GetTime hostName");
            System.exit(1);
        }

        hostName = args[0];
        try {
            client = new Socket(hostName, 13);
        } catch (UnknownHostException hostError) {
            hostError.printStackTrace();
            System.exit(1);
        } catch (IOException genericError) {
            System.err.println(genericError.getMessage());
            System.exit(1);
        }

        try {
            BufferedReader in =
                new BufferedReader(
                    new InputStreamReader(client.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                System.out.println("Received: " + inputLine);
            }
        } catch (IOException IOError) {
            System.err.println(IOError.getMessage());
            System.exit(1);
        }
    }
}
```
java.net.Socket

Several constructors:

Socket(String hostname, int portNumber)
Socket(InetAddress hostAddress, int portNumber)

The constructor builds the connection

Exceptions thrown by the constructors:

<table>
<thead>
<tr>
<th>Exception</th>
<th>Reason it is raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnknownHostException</td>
<td>The specified host is not valid or cannot be looked up</td>
</tr>
<tr>
<td>SocketException</td>
<td>Something went wrong attempting to build a connection. Most common cause is &quot;Connection refused&quot; - no service on the specified port</td>
</tr>
<tr>
<td>IOException</td>
<td>Catchall</td>
</tr>
</tbody>
</table>
java.net.Socket Methods

    close( )

Terminates an existing connection.

    getInputStream( )
    getOutputStream( )

These two provide access to IO through the connection.

NOTE network IO is always done in packets. This may produce strange results. Solution: use the Buffered IO classes.

    getLocalPort( )
Domain Name Service (DNS)

On UNIX:

- `bind` (Berkeley Internet Name Domain)
  - Standard UNIX DNS software
- `named` (name daemon)
  - Program that provides the DNS service

DNS

- Converts host and domain names into IP addresses
- Converts IP addresses into host and domain names

Why DNS?

- P addresses are hard to remember
- Allows for
  - Abstraction
  - Easy migration
  - Maintenance
DNS through java.net.InetAddress

Use following static methods to create InetAddress

InetAddress  getByName(String host)
InetAddress[]  getAllByName(String host)
InetAddress getLocalHost(  )

Host can be machine name or IP address

Useful methods

String getHostName(  )
byte[] getAddress(  )
InetAddress Example

```java
import java.net.InetAddress;
import java.net.InetAddress;
import java.io.*;

class SimpleDNS {

    public static void main(String[] args) {
        String hostName;

        if (args.length != 1) {
            System.out.println("Usage: java SimpleDNS hostName");
            System.exit(1);
        }

        hostName = args[0];
        try {
            InetAddress address = InetAddress.getByName(hostName);
            byte ip[] = address.getAddress();

            for (int octet=0; octet < ip.length; octet++) {
                System.out.print(((int)ip[octet] & 0xff) + " ");
            }
            System.out.println();
        }
        catch (IOException DNSerror) {
            System.err.println(DNSerror);
            System.exit(1);
        }
    }
}
```
Running the Example

$ java SimpleDNS www.sdsu.edu
130 191 13 5
$

Closing Sockets

A socket is closed when:

- Program ends
- One of its two streams is closed
- It is garbage collected
- You send the close message to it

Good practice to close socket when done with it

On a closed socket you can still get its

- InetAddress
- Port
- Local address
- Local port
- Input and output stream

An exception is raised if you use a stream on a close socket

In JDK 1.3 you can close 1/2 of the socket
PortScanner
From Java Network Programming chapter 10

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

public class PortScanner {
    public static void main(String[] args) throws IOException {
        String host = "localhost";
        Socket connection = null;

        try {
            for (int port = 1; port < 65536; port++) {
                try {
                    connection = new Socket(host, port);
                    System.out.println("Server on port "+ port);
                }
                catch (IOException error) {// no server on this port
                }
            }
        }
        catch (UnknownHostException error) {
            System.err.println(error);
        }

        finally {
            if (connection != null) connection.close();
        }
    }
}
Socket Options

You to change parameters controlling how native sockets send and receive data

TCP_NODELAY

public void setTcpNoDelay(boolean on) throws SocketException
public boolean getTcpNoDelay() throws SocketException

Disable/enable Nagle's algorithm

• Do not pool small packets
• Do not wait for acknowledgement of last packet before sending current packet

Exception is raised if underlying native sockets do not support this option
SO_LINGER

public void setSoLinger(boolean on, int lingerSeconds) throws SocketException
public boolean getSoLinger() throws SocketException

What happens to waiting packets when socket is closed?

Option 1
  Drop packets and close socket

Option 2
  Wait N seconds to send packets and receive acknowledgements

lingerSeconds is number of seconds to wait

0 value = no wait

Max value is 65,535 seconds (~18 hours)

Best to use system default
SO_TIMEOUT

public void setSoTimeout(int timeout) throws SocketException
public boolean getSoTimeout() throws SocketException

How long does read() block on a socket?

Default is to block until there is data.

SO_TIMEOUT
- specifies how long to wait in a read
- In milliseconds
- Default value is 0, meaning wait forever

An InterruptedException is thrown when time out occurs

Socket is still connected and usable
**SO_RCVBUF & SO_SNDBUF**

public void setReceiveBufferSize(int size) throws SocketException
public int getReceiveBufferSize() throws SocketException

public void setSendBufferSize(int size) throws SocketException
public int getSendBufferSize() throws SocketException

Suggests the input/output buffer size used in the TCP stack

**SO_KEEPALIVE**

New in JDK 1.3

Have client send data on an idle connection to insure server is still alive

Default is off