Introduction to Network Programming (java.net)

Mazen Malek Shiaa
malek@item.ntnu.no

Outline

• Applets
• All what you need about URLs
• Socket programming
• Implementation of client and server programs
• Datagrams

Sources:
1. Java.sun.com
(http://java.sun.com/docs/books/tutorial/networking/index.html)
1. core JAVA 2 by Horstmann & Cornell

Custom networking (browsing)

“The Java platform is highly regarded in part because of its suitability for writing programs that use and interact with the resources on the Internet and the World Wide Web. In fact, Java-compatible browsers use this ability of the Java platform to the extreme to transport and run applets over the Internet”

Keywords: applets, URLs, sockets, datagrams, servlets (not introduced)

Networking in Java
Browsing (www)

• Applets are the main tool
• Applets are Java-compatible programs that can be embedded in a Web page
• using a Java-enabled browser you can run applets
• Applets are referenced by a special tag in an HTML file—the <APPLET> tag
• Applets can be located anywhere, locally or remotely but invisibly to the user
• The location of the applet is encoded within the <APPLET> tag. The browser decodes this information, locates the applet, and runs it
How to implement Applets!

- write your application: classes should be derived from `applet`, use `init`, `start`, `stop`, `destroy` methods, … and make `.class` available at a URL
- write your HTML files that contain references to your applets:
  ```html
  <BODY>
  ...
  <APPLET CODE="Example.class WIDTH = 200 HEIGHT = 200">
  </APPLET>
  ...
  </BODY>
  ```
- The creation, initialization, running, stopping, and destruction of an applet are all managed by the browser that loads the applet

What happens when visiting a page that contains an applet!

1. The browser finds the class file for the applet's `Applet` subclass. The location of the class file (which contains Java bytecodes) is specified with the `CODE` and `CODEBASE` attributes of the `<APPLET>` tag.
2. The browser brings the bytecodes over the network to the user's computer.
3. The browser creates an instance of the Applet subclass. When we refer to an applet, we're generally referring to this instance.
4. The browser calls the applet's `init` method. This method performs any one-time initialization that is required.
5. The browser calls the applet's `start` method. This method often starts a thread to perform the applet's duties.

Applets capabilities

- Applets can usually make network connections to the host they came from
- Applets running within a Web browser can easily cause HTML documents to be displayed
- Applets can invoke public methods of other applets on the same page
- Applets that are loaded from the local file system (from a directory in the user's CLASSPATH) have none of the restrictions that applets loaded over the network do
- Although most applets stop running once you leave their page, they don't have to

Applets restrictions

- An applet cannot load libraries or define native methods
- It cannot ordinarily read or write files on the host that's executing it
- It cannot make network connections except to the host that it came from
- It cannot start any program on the host that's executing it
- It cannot read certain system properties
- Windows that an applet brings up look different than windows that an application brings up
Networking Basics

Computers running communicate to each other using either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP).

Typically, you don't need to concern yourself with the TCP and UDP layers. Instead, you can use the classes in the `java.net` package.

TCP/UDP

**Definition:** TCP (Transmission Control Protocol) is a connection-based protocol that provides a reliable flow of data between two computers. TCP guarantees that data sent from one end of the connection actually gets to the other end and in the same order it was sent. This is analogous to making a telephone call.

**Definition:** UDP (User Datagram Protocol) is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP. Sending datagrams is much like sending a letter through the mail service so the order of delivery is not important.

Example: Connection Oriented Transport and Network Service

A transport entity has the responsibility for the execution of a transport protocol type within an end system.
Understanding ports

A computer has a single connection to the network—32-bit IP address. All data arrives through that connection. However, data intended for different applications will be forwarded through the use of ports—16-bit number.

In connection-based communication such as TCP, a server application binds a socket to a specific port number. A client can then rendezvous with the server at the server's port.

In datagram-based communication such as UDP, the datagram packet contains the port number of its destination and UDP routes the packet to the appropriate application.

Working with URLs

Uniform Resource Locator (URL) is a reference (address) to a resource on the Internet. Java programs can use a class called URL in the java.net package to represent a URL address.

URL has two main components:
1. Protocol identifier
2. Resource name

The format of the resource name depends entirely on the protocol used; but mainly the following:
- Host Name
- Filename
- Port Number
- Reference

Using URL in Java applications

- In your Java program, use a String containing text to create a URL object:
  
  ```java
  URL item = new URL("http://www.item.ntnu.no/");
  ```

- URL has other constructors:
  
  ```java
  URL item = new URL("http", "www.item.ntnu.no", "/fag/ttm4100/project.html");
  URL item = new URL("http", "www.item.ntnu.no", 80, "/fag/ttm4100/project.html");
  ```

- check java.net Class URL

- use the following to capture errors due to availability or unknown protocols:
  
  ```java
  try {
    URL myURL = new URL(...)
  } catch (MalformedURLException e) {
    ...
    // exception handler code here
    ...
  }
  ```

more URL!!!

The URL class provides several methods that let you query URL objects:

- `getProtocol`:
  
  Returns the protocol identifier component of the URL.

- `getHost`:
  
  Returns the host name component of the URL.

- `getPort`:
  
  Returns the port number component of the URL. The getPort method returns an integer that is the port number. If the port is not set, getPort returns -1.

- `getFile`:
  
  Returns the filename component of the URL.

- `getRef`:
  
  Returns the reference component of the URL.
import java.net.*;
import java.io.*;
public class ParseURL {
    public static void main(String[] args) throws Exception {
        URL aURL = new URL("http://java.sun.com:80/docs/books/" +
                "tutorial/index.html#DOWNLOADING");
        System.out.println("protocol = " + aURL.getProtocol());
        System.out.println("host = " + aURL.getHost());
        System.out.println("filename = " + aURL.getFile());
        System.out.println("port = " + aURL.getPort());
        System.out.println("ref = " + aURL.getRef());
    }
}

protocol = http
dest = java.sun.com
filename = /docs/books/tutorial/index.html
port = 80
ref = DOWNLOADING

• Reading Directly from a URL
After you’ve successfully created a URL, you can call the URL’s openStream() method to get a stream from which you can read the contents of the URL. This returns a java.io.InputStream object, for example:

    URL item = new URL("http://www.item.ntnu.no/");
    BufferedReader in = new BufferedReader(new InputStreamReader(item.openStream()));
    String inputLine;
    inputLine = in.readLine();
    in.close();

• Connecting to a URL and then reading/writing
When you connect to a URL, you are initializing a communication link between your Java program and the URL over the network… and then you might read/write to the link, for example

    URL item = new URL("http://www.item.ntnu.no/");
    URLConnection ic = item.openConnection();
    BufferedReader in = new BufferedReader(new InputStreamReader(ic.getInputStream()));
    String inputLine;
    inputLine = in.readLine();
    in.close();

Writing to a URLConnection
HTML pages may contain forms for sending data to the server. A cgi-bin script or servlet program on the server receives the data, processes it, and then sends back a response, usually in the form of a new HTML page. These scripts use POST METHOD (or more restrictive GET METHOD) to read from the clients.

A Java program may communicate with the server side script via:
1. Create a URL
2. Open a connection to the URL (set output capability on the URLConnection)
3. Get an output stream from the connection
4. Write to the output stream
5. Close the output stream

Example:
    URL url = new URL("http://java.sun.com/cgi-bin/backwards");
    URLConnection connection = url.openConnection();
    connection.setDoOutput(true);
    PrintWriter out = new PrintWriter(connection.getOutputStream());
    out.println("string=" + "something");
    out.close();
    ...........

What are Sockets?

A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes, Socket and ServerSocket.

URLs and URIs provide high-level mechanism for accessing resources on the Internet. Lower-level network communication is achieved by using socket programming. In client-server applications, the server provides some service that is used by the client. The communication that occurs between the client and the server must be reliable(TCP).
Establishing Socket connection

A client program needs first to establish a socket connection with the server

```java
echoSocket = new Socket("tapas1.item.ntnu.no", 7);
// machine name should be a fully qualified IP name, 7 is the port number
out = new PrintWriter(echoSocket.getOutputStream(), true);
// opens a PrintWriter on the port’s output stream
in = new BufferedReader(new InputStreamReader(echoSocket.getInputStream()));
// gets the socket’s input stream and opens a BufferedReader on it

more about streams refer to java.io (byte/character streams)
```

Reading from and Writing to a Socket

- **Reading from**
is achieved through reading from the input stream of a socket … `in.readLine()`
- **Writing to**
is achieved through writing on the output stream of a socket … `out.println()`

```java
while ((userInput = stdIn.readLine()) != null) { out.println(userInput); System.out.println("echo: " + in.readLine()); }
```

Housekeeping

```java
out.close();
in.close();
echoSocket.close();
```

Sockets in brief

1. import `java.net` package
2. encase your code in a `try/catch` block to catch any input/output errors
3. Open a socket and pass IP address and port number
4. Open an input stream and output stream to the socket
5. Read from and write to the stream according to the server's protocol
6. Close the streams
7. Close the socket
Implementing Client/Server

• The server program begins by creating a new ServerSocket object to listen on a specific port

```java
try {
    serverSocket = new ServerSocket(9999); // use a number that is not used by other service
} catch (IOException e) {
    System.out.println("port: 9999 doesn’t work");
    System.exit(-1);
}
```

• If the server successfully connects to its port, then the ServerSocket object is successfully created and the server continues to the next step, accepting a connection from a client

```java
Socket clientSocket = null;
try {
    clientSocket = serverSocket.accept();
} catch (IOException e) {
    System.out.println("Accept failed: 9999");
    System.exit(-1);
}
```

• The server can service multiple requests simultaneously through the use of threads – one thread per each client connection. The basic flow of logic in such a server is:

```java
while (true) {
    accept a connection;
    create a thread to deal with the client;
}
```

• Open input and output streams

```java
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
```

• define the protocol procedures, and define the actions for socket and stream close

• implement the client according to the same procedures

• run the server program first at the specified location (machine), and then run the client

Datagrams

• A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. The DatagramPacket and DatagramSocket classes in the java.net package implement system-independent datagram communication using UDP

• applications that communicate via datagrams send and receive completely independent packets of information. These clients and servers do not have and do not need a dedicated point-to-point channel

• open a datagram socket

```java
socket = new DatagramSocket(9999); // you can leave this empty as you use
......
// any available port
try {
    in = new BufferedReader(new FileReader("whatever.txt")); // a file stream
} catch (FileNotFoundException e) {
    System.err.println("Couldn't open.");
}
```

Implementing Datagram

• Receiving datagram

```java
byte[] buf = new byte[256];
DatagramPacket packet = new DatagramPacket(buf, buf.length);
socket.receive(packet);
```

• Receiving packet’s IP address and port number

```java
InetAddress address = packet.getAddress();
int port = packet.getPort();
```

• Sending datagram

```java
packet = new DatagramPacket(buf, buf.length, address, port);
socket.send(packet);
```
Remarks

• InetAddress is used to represent an IP address

• socket exceptions: why a socket connection was refused?
  java.net.BindException
  The local port is in use, or the requested bind address couldn't be assigned locally
  java.net.ConnectException
  This exception is raised when a connection is refused at the remote host
  java.net.NoRouteToHostException
  The connect attempt timed out, or the remote host is otherwise unreachable

• HttpURLConnection extends URLConnection, and provides additional functionality
  specific to HTTP: GET, POST, PUT, HEAD

• Use java.net.URLClassLoader to load classes and resources from a search path of
  URLs referring to both JAR files and directories