JAC444 - Introduction to Java for C++ Programmers

Lesson 8: Network Programming in Java

Agenda

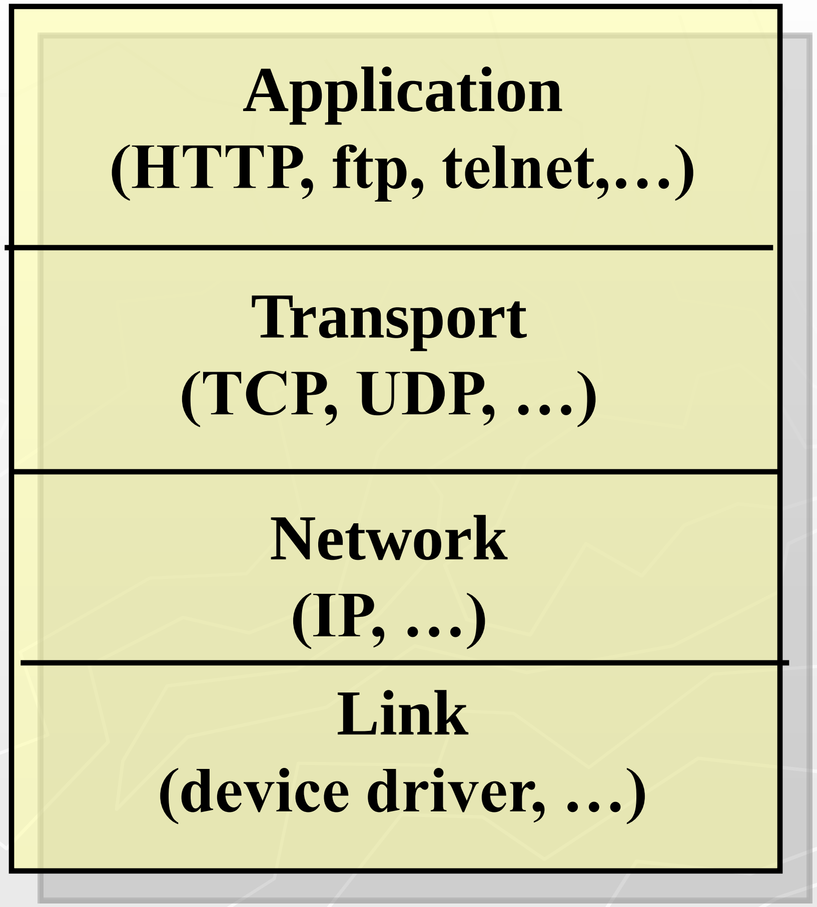
- ►Networking Basics
- ►Socket-Based Client/Server Programming
- ►URL Processing

Networking Basics

- The term network programming refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.
- Computers on the network use either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP).
- In Java, the java.net package provides support for the both protocols.

The Internet Protocol Layers

- Application layers 7, 6 :
  - email, HTTP, FTP, Telnet, file 'sharing', streaming media, VoIP
  - remote access: VPN
- Session layer 5: connections.
  - e.g. sockets
- Transport layer 4: TCP, UDP
- Network layer 3: IP, the Internet
- Link layer 2, 1: NICs, device drivers, magic.



Transport Control Protocol (TCP)

- TCP is a connection-based protocol that provides a realiable flow of data between two computers.
- TCP provides point-to-point channel for applications that require reliable communications:
  - Hypertext Transfer Protocol (HTTP)
  - Telnet
  - File Transfer Protocol (FTP)
- 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.

User Datagram Protocol (UDP)

- UDP is a protocol that sends independent packets of data, called datagrams, from one computer to another.

- UDP is not connection-based like TCP. UDP does not guarantees about arrival of data.

Java Classes for Networking

- Network programming is at the application layer.
  - So typically, you don't need to concern yourself with the TCP and UDP layers.
- The classes in the java.net package provide system-independent network communication for using both TCP and UDP.
  - The classes for communication by using TCP:
    - URL
    - URLConnection
    - Socket
    - ServerSocket
    - InetAddress

Business Applications

1. run on single computers(e.g. desktops, legacy systems)

2. run on a network of computers
   - client/server applications: applications in which several computer systems collaborate to get some work done (e.g. socket-based applications)
   - distributed applications: a higher level of abstraction (e.g. RMI applications)
   - Web applications (e.g. Java servlets and JSP)

Programming Client/Server Applications

- client: a machine/program that requests services
- server: a host machine/program that handles service requests from clients
- Example: SimpleClient.java, SimpleServer.java
  - client: makes a request for calculating the area of a circle
  - server: provides the service of calculating the area
- Java packages: java.net, java.io

Identifying Hosts

- IP stands for Internet Protocol and is the protocol that moves packets of data between source and destinations.
- IP Address is a unique four bytes (32bit) number that identifies a computer in a network.

  e.g. 142.204.1.2
- Ports are identified by a 16-bit number, which TCP, UDP use to deliver the data to the right application.

  e.g. Port 80: HTTP service

  Port 21: telnet service

Sockets

- Socket
  - an endpoint of a two-way communication link between two programs running on the network
  - a software abstraction of a TCP/IP network connection: Java streams of data
- A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

Socket-Based Client/Server Programming

- A server program
  - create a Java **server socket**
  - listen for a **connection** from a client
  - connect streams to a Java **socket**
  - read from and write to a socket

- Example: SimpleServer.java

Socket-Based Client/Server Programming

- A client program
  - establish a connection to a server program
  - connect streams to a Java socket
  - read from and write to a socket

- Example: SimpleClient.java

The java.net.Socket Class

- the constructors
  - creation of client sockets
- InetAddress getInetAddress( )
  - the InetAddress associated with the Socket object is returned
- int getPort( )
  - the remote port of the socket connection is returned
- int getLocalPort( )
  - the local port of the socket conncetion is returned
- InputStream getInputStream( )
  - the InputStream associated with the socket connection
- OutputStream getOutputStream( )
  - the OutputStream associated with the socket connection

The java.net.SocketServer Class

- constructors
  - creation of server sockets with port numbers
- Socket accept( )
  - a blocking call that waits for a client to initiate communication

The java.net.InetAddress Class

- represents an Internet Protocol (IP) address.
- some factory methods: static methods that return instances of the class
  - static InetAddress getLocalHost( )
    throws UnknownHostException

    - the returned InetAddress object represents the local host

  - static InetAddress getByName(String hostName)
    throws UnKnownHostException

    - the returned InetAddress object represents the host name that is passed into the method

The java.net.InetAddress Class

- some instance methods
  - String getHostAddress( )

    - the returned string represents the host address associated with the InetAddress obejct

  - String getHostName( )

    - the returned string represents the host name associated with the InetAddress object

Programming a Multi-threaded Server

- threads are used to handle multiple client requests
- Example: SimpleServerMT.java

A Client/Server Application: Transfer of Objects

- a client/server application that exchanges objects

- Example: Car.java, CarsClient.java, CarsServer.java

Java Serialization

- Serialization is the process of translating data structures or object state into a format that can be
  - stored (for example, in a file or memory buffer)
  - or transmitted across a network connection link,
  - and reconstructed/deserialized later in the same or another computer environment.
- In Java, if a class implements the java.io.Serializable interface, the its object is *serializable.*
- Classes ObjectInputStream and ObjectOutputStream are high-level streams that contain the methods for serializing and deserializing an object.

Transient Variable

- Transient keyword provides you some control over serialization process and gives you flexibility to exclude some of object properties from serialization process.
- e.g.

```
public class Stock {
    private transient Logger logger
        = Logger.getLogger(Stock.class); // not serialized
    private String symbol;     //will be serialized
    private BigInteger price; //serialized
    private long quantity;     //serialized
}
```

URL Processing

- URL (Uniform Resource Locator):
  - a reference (i.e. an address) to a resource on the Internet.
- The syntax of a URL

http://www.senecacollege.ca/demoweb/url-primer.html#part1

protocol name (e.g. http)
host name (e.g. www.senecacollege.ca)
port number (optional)
  (e.g. 80 for the predefined HTTP port)
web resource/file path
  (e.g. demoweb/url-primer.html)
reference (e.g. #part1)

The java.net.URL Class

- Creating URL objects
  - e.g. URL myURL = new URL("http://example.com/");
  - throws java.io.MalformedURLException
- Connecting to a URL
  - e.g. myURL.openConnection( )
  - throws IOException
  - return a URLConncection object
- Reading directly from a URL
  - InputStream openStream( ) throws IOExcepton;
    - open a connection to the URL and return an input stream for reading its contents

The java.net.URLConnection Class

- A general-purpose class used for accessing a Web resource
- Reading from a URLConnection - InputStream getInputStream( )
- Writing to a URLConnection
  - void setDoOutput(boolean flag)
    - the URLConnection must be set to true for writing purpose
  - OutputStream getOutputStream( )

Example

- Downloading file form a server
  - URLDemo.java

Resourceful Links

- Java Tutorial on Networking

Thank You!