Network Programming

[CACS355]

BCA 6th Sem

Er. Sital Prasad Mandal

(Email : info.sitalmandal@gmail.com)

Bhadrapur, Jhapa, Nepal

https://networkprogam-mmc.blogspot.com/

https://networkprogam-mmc.blogspot.com/

Unit-7

Sockets for Servers

1. Using ServerSockets
    I. Serving Binary Data
    II. Multithreaded Servers
    III. Writing to Servers with Sockets
    IV. Closing Server Sockets
2. Logging
    I. What to Log
    II. How to Log
3. Constructing Server Sockets
    I. Constructing Without Binding
4. Getting Information About a Server Socket
    i. Socket Options
    ii. SO_TIMEOUT
    iii. SO_REUSEADDR
    iv. SO_RCVBUF
    v. Class of Service
5. HTTP Servers
    i. A Single-File Server
    ii. A Redirector
    iii. A Full-Fledged HTTP Server

Unit-7

Sockets for Servers

Server Sockets

1. A server socket binds to a particular port on the local machine.
2. Once it has successfully bound to a port, it listens for incoming connection attempts.
3. When a server detects a connection attempt, it accepts the connection. This creates a socket between the client and the server over which the client and the server communicate.

https://networkprogam-mmc.blogspot.com/

Unit-7

Sockets for Servers

The java.net.ServerSocket Class

1. The java.net.ServerSocket class represents a server socket.
2. A ServerSocket object is constructed on a particular local port. Then it calls accept() to listen for incoming connections.
3. accept() blocks until a connection is detected. Then accept() returns a java.net.Socket object that performs the actual communication with the client.

Unit-7

Sockets for Servers

Constructors

There are three constructors that let you specify the port to bind to, the queue length for incoming connections, and the IP address to bind to:

i. public ServerSocket(int port) throws IOException
ii. public ServerSocket(int port, int backlog) throws IOException
iii. public ServerSocket(int port, int backlog, InetAddress networkInterface) throws IOException

Unit-7

Reading Data with a ServerSocket

```
public class WriteServerSocketBetter {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(2345);
            Socket s = ss.accept();
            Writer out = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
            out.write("Hello There!\r\n");
            out.write("Goodbye now.\r\n");
            out.flush();
            s.close();
        }
        catch (IOException e) {
            System.err.println(e);
        }
    }
}
```

```
public class ReadServerSocketBetter{
    public static void main(String[] args) throws IOException {
        Socket s=new Socket("localhost", 2345);
        InputStream in = s.getInputStream();
        InputStreamReader isr = new InputStreamReader(in, "ASCII");
        BufferedReader br = new BufferedReader(isr);
        br.lines().forEach(System.out::println);
    }
}
```

Unit-7

Sockets for Servers

- Daytime Protocol (RFC 867): TCP Port 13 (Examples 8-1 and 8-2 Client)
- Iterative server: there's one big loop, and in each pass through the loop a single connection is completely processed

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

public class DaytimeServer {

  public final static int PORT = 13;

  public static void main(String[] args) {
   try (ServerSocket server = new ServerSocket(PORT)) {
     while (true) {
       try (Socket connection = server.accept()) {
         Writer out = new OutputStreamWriter(connection.getOutputStream());
         Date now = new Date();
         out.write(now.toString() +"\r\n");
         out.flush();
         connection.close();
       } catch (IOException ex) {}
     }
   } catch (IOException ex) {
     System.err.println(ex);
   }
  }
}
```

```
$ telnet localhost 13
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Sat Mar 30 16:15:10 EDT 2013
Connection closed by foreign host
```

1. creates a server socket that listens on port 13.
   - Note that ports 0-1023 are privileged by root
2. Calls accept(), stops here and waits. When a client does connect, it returns a Socket object
3. Serves the request: writes / returns the daytime
4. Closes the connection and loops back to accept next requests

- Java 7's try-with-resources used to autoclose the socket
- The client closes and the server throws Interrupted InterruptedIOException
  - The server gets ready to process the next incoming connection

Ctrl-C could terminate the program

Unit-7

Reading Data with a ServerSocket

```
public class DaytimeServer {
  public final static int PORT = 13; // 13
  public static void main(String[] args) {
   try (ServerSocket server = new ServerSocket(PORT)) {
     while (true) {
       try (Socket connection = server.accept()) {
         Writer out = new OutputStreamWriter(connection.getOutputStream());
         Date now = new Date();
         out.write(now.toString() + "\n");
         out.flush();
         connection.close();
       } catch (IOException ex) {}
     }
     } catch (IOException e) {
       System.out.println(e);
     }
     }
}
```

Unit-7

A Multithreaded Daytime Server

```
public class MultithreadedDaytimeServer {
  public final static int PORT = 13;
  public static void main(String[] args) {
   try (ServerSocket server = new ServerSocket(PORT)) {
     while (true) {
       try {
         Socket connection = server.accept();
         Thread task = new DaytimeThread(connection);
         task.start();
       } catch (IOException ex) {}
     }
    } catch (IOException ex) {
      System.err.println("Couldn't start server");
    }
  }
```

```
  private static class DaytimeThread extends Thread {
    private Socket connection;
    DaytimeThread(Socket connection) {
      this.connection = connection;
    }
    @Override
    public void run() {
      try {
        Writer out = new OutputStreamWriter(connection.getOutputStream());
        Date now = new Date();
        out.write(now.toString() +"\r\n");
        out.flush();
      } catch (IOException ex) {
        System.err.println(ex);
      } finally {
        try {
          connection.close();
        } catch (IOException e) {
          // ignore;
        }
      }
    }
  }
}
```

Does not use try-with-resources, or the main thread would close the socket as soon as it gets to the end of the while loop before the thread is spawned

A thread per connection design

Problem: Numerous roughly simultaneous incoming connections can cause it to spawn an indefinite number of threads

https://networkprogam-mmc.blogspot.com/

Unit-7

A Multithreaded Daytime Server

```
public class MultithreadedDaytimeServer {
    public final static int PORT = 13;
    public static void main(String[] args) {
        try (ServerSocket server = new ServerSocket(PORT)) {
            while (true) {
                try {
                    Socket connection = server.accept();
                    Thread task = new DaytimeThread(connection);
                    task.start();
                } catch (IOException ex) {}
            }
        } catch (IOException ex) {
            System.err.println("Couldn't start server");
        }
    }

    private static class DaytimeThread extends Thread {
        private Socket connection;
        DaytimeThread(Socket connection) {
            this.connection = connection;
        }

        @Override
        public void run() {
            try {
                Writer out = new OutputStreamWriter(connection.getOutputStream());
                Date now = new Date();
                out.write(now.toString() +"\r\n");
                out.write("From Server" +"\r\n");
                out.flush();
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }
    }
}
```

```
public class ReadServerSocketBetter {
    public static void main(String[] args) throws IOException {
        Socket s=new Socket("localhost",13);
        InputStream in = s.getInputStream();
        InputStreamReader isr = new InputStreamReader(in, "ASCII");
        BufferedReader br = new BufferedReader(isr);
        br.lines().forEach(System.out::println);
    }
}
```

https://networkprogam-mmc.blogspot.com/

Unit-7

A Multithreaded Daytime Server

```
public class MultithreadedDaytimeServer {
    public final static int PORT = 13;
    public static void main(String[] args) {
        try (ServerSocket server = new ServerSocket(PORT)) {
            while (true) {
                try {
                    Socket connection = server.accept();
                    Thread task = new DaytimeThread(connection);
                    task.start();
                } catch (IOException ex) {}
            }
        } catch (IOException ex) {
            System.err.println("Couldn't start server");
        }
    }

    private static class DaytimeThread extends Thread {
        private Socket connection;
        DaytimeThread(Socket connection) {
            this.connection = connection;
        }

        @Override
        public void run() {
            try {
                Writer out = new OutputStreamWriter(connection.getOutputStream());
                Date now = new Date();
                out.write(now.toString() +"\r\n");
                out.write("From Server" +"\r\n");
                out.flush();
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }
    }
}
```

```
public class ReadServerSocketBetter {
    public static void main(String[] args) throws IOException {
        Socket s=new Socket("localhost",13);
        InputStream in = s.getInputStream();
        InputStreamReader isr = new InputStreamReader(in, "ASCII");
        BufferedReader br = new BufferedReader(isr);
        br.lines().forEach(System.out::println);
    }
}
```

Unit-7

Closing Server Sockets

- Frees a port on the local host, allowing another server to bind to the port
  - Closing a Socket object just frees the spawned end-to-end TCP connection
- It also *breaks all currently open sockets* that the ServerSocket has accepted
- ServerSocket is closed automatically when a program dies
  - However, it's good to close it as is no longer needed: three ways to close

1. Typical constrouctor

```
ServerSocket server = null;
try {
  server = new ServerSocket(port);
  // ... work with the server socket
} finally {
  if (server != null) {
    try {
      server.close();
    } catch (IOException ex) {
      // ignore
    }
  }
}
```

2. Uses noargs constructor and calls bind() later to prevent exception

```
ServerSocket server = new ServerSocket();
try {
  SocketAddress address = new InetSocketAddress(port);
  server.bind(address);
  // ... work with the server socket
} finally {
  try {
    server.close();
  } catch (IOException ex) {
    // ignore
  }
}
```

3. Java 7+: AutoCloseable try-with-resources

```
try (ServerSocket server = new ServerSocket(port)) {
  // ... work with the server socket
}
```

- isBound(): whether the ServerSocket has been bound to a port

```
public static boolean isOpen(ServerSocket ss) {
  return ss.isBound() && !ss.isClosed();
}
```

Unit-7

Adding Threading to a Server

i. It's better to make your server multi-threaded.

ii. There should be a loop which continually accepts new connections.

iii. Rather than handling the connection directly the socket should be passed to a Thread object that handles the connection.

Multi-threading is a good thing but it's still not a perfect solution.

Thread Pool

i. Create a pool of threads when the server launches, store incoming connections in a queue, and have the threads in the pool progressively remove connections from the queue and process them.

ii. The main change you need to make to implement this is to call accept() in the run() method rather than in the main() method.

Unit-7

Logging

- Two primary things to store in the logs
  - Audit log: requests
  - Error log: server errors
    - The general rule of thumb: every line in the error log should be looked at and resolved
    - Do not keep debug logs in production; put it in another separate file
- **java.util.logging** package since Java 1.4
  - **Logger.getLogger()**: create one per class with a (dot-separated) log name

```
private final static Logger auditLogger = Logger.getLogger("requests");
```

    - Normally based on the packet name or class name
    - Loggers are thread safe
    - Multiple Logger objects can output to the same log, but usually exactly one log
  - log(): log messages with specified levels

```
catch (RuntimeException ex) {
  logger.log(Level.SEVERE, "unexpected error " + ex.getMessage(), ex);
}
```

    - Seven levels defined as named constants in **java.util.logging.Level**

      Level.SEVERE > .WARNING > .INFO > .CONFIG > .FINE > .FINER > Level.FINEST

      –Examples: Level.INFO for audit logs, and Level.WARNING or Level.SEVERE for error logs or Logger.info(), Logger.warning()/Logger.severe() instead
  - By default, the logs are just output to the console
    - Set log to file when launching the JVM

      –Djava.util.logging.config.file=_filename_

https://networkprogam-mmc.blogspot.com/

Unit-7

Logging

Logging Levels

Log Level	Priority Order
FATAL	High
ERROR	
INFO	
WARN	
DEBUG	Low

```java
import java.util.logging.*;
public class Logging {
    private static Logger logger = Logger.getLogger("requests");
    public static void main(String[] args) {
        try {
            logger.info("Hello World");
            System.out.println("Hi");
        } catch (RuntimeException ex) {
            logger.log(Level.SEVERE, "erro" + ex.getMessage(),ex);
        }
    }
}
```

https://networkprogam-mmc.blogspot.com/

Unit-7

Logging

```
import java.util.logging.*;
public class Logging {
    private static Logger logger = Logger.getLogger("requests");

    public static void main(String[] args) {
           try {
            logger.info("Line No: " + new Exception().getStackTrace()[0].getLineNumber());
            System.out.println("Hi");
            logger.info("Line No throw: " + new Throwable().getStackTrace()[0].getLineNumber());

        } catch (RuntimeException ex) {
            logger.log(Level.SEVERE, "erro" + ex.getMessage(),ex);
        }
    }
}
```

Unit-7

3. Constructing ServerSockets

- 4 constructors, throw BindException

```
public ServerSocket(int port) throws BindException, IOException
public ServerSocket(int port, int queueLength)
    throws BindException, IOException
public ServerSocket(int port, int queueLength, InetAddress bindAddress)
    throws IOException
public ServerSocket() throws IOException
```

  - port: the port to listen to
    - 0: the system will select an available port (anonymous port)
  - queuelength: hold incoming connection request
  - bindAddress: specify the local network interface to bind to
    - By default, the server socket listens on all the interfaces and IP addresses of the host
- Constructing without binding
  - bind() later; port 0 for anonymous port

```
public void bind(SocketAddress endpoint) throws IOException
public void bind(SocketAddress endpoint, int queueLength) throws IOException
```

I. Constructing Without Binding

```
ServerSocket ss = new ServerSocket();
// set socket options...
SocketAddress  http = new InetSocketAddress(80);
ss.bind(http);
```

Unit-7

Example: LocalPortScannerServer

- Look for local ports (for ports 1024 and above)
  - Attempt to open a server on that port

```
import java.io.*;
import java.net.*;
public class LocalPortScannerServer {
  public static void main(String[] args) {
    for (int port = 1; port <= 65535; port++) {
      try {
        // the next line will fail and drop into the catch block if
        // there is already a server running on the port
        ServerSocket server = new ServerSocket(port);
      } catch (IOException ex) {
        System.out.println("There is a server on port " + port + ".");
      }
    }
  }
}
```

There is a server on port 23.
There is a server on port 135.
There is a server on port 139.
.
.
There is a server on port 51921.
There is a server on port 57053.
There is a server on port 63342.

Unit-7

4. Getting Information About a Server Socket

- Two getter methods
  - getInetAddress(): return the address being used for an accepted connection
    - Return null if not yet bound

```
ServerSocket httpd = new ServerSocket(80);
InetAddress ia = httpd.getInetAddress();
```

  - getLocalPort(): find out what port is listening on (for anonymous port)

```
public int getLocalPort()
```

- toString(): for debugging

```
ServerSocket[addr=0.0.0.0,port=0,localport=5776]
```

- Example 9. RandomPort
  - Bind to an anonymous port

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

public class RandomPort {

  public static void main(String[] args) {
    try {
      ServerSocket server = new ServerSocket(0);
      System.out.println("This server runs on port "
          + server.getLocalPort());
    } catch (IOException ex) {
      System.err.println(ex);
    }
  }
}
```

```
$ java RandomPort
This server runs on port 1154
D:\JAVA\JNP4\examples\9>java RandomPort
This server runs on port 1155
D:\JAVA\JNP4\examples\9>java RandomPort
This server runs on port 1156
```

Unit-7

5. Socket Options

Three options supported for server sockets

- SO_TIMEOUT: timeout in ms for accept()

```
public void setSoTimeout(int timeout) throws SocketException
public int  getSoTimeout() throws IOException
```

  - Set before calling accept(). Can't change while accept() is waiting
  - SocketTimeoutException thrown

```
try (ServerSocket server = new ServerSocket(port)) {
  server.setSoTimeout(30000); // block for no more than 30 seconds:
  try {
    Socket s = server.accept();
    // handle the connection
    // ...
  } catch (SocketTimeoutException ex) {
    System.err.println("No connection within 30 seconds");
  }
} catch (IOException ex) {
  System.err.println("Unexpected IOException: " + e);
}
```

```
public void printSoTimeout(ServerSocket server) {
  int timeout = server.getSoTimeOut();
  if (timeout > 0) {
    System.out.println(server + " will time out after "
        + timeout + "milliseconds.");
  } else if (timeout == 0) {
    System.out.println(server + " will never time out.");
  } else {
    System.out.println("Impossible condition occurred in " + server);
    System.out.println("Timeout cannot be less than zero." );
  }
}
```

- SO_REUSEADDR: allowed to bind a used port even there might still be data

```
public boolean getReuseAddress() throws SocketException
public void setReuseAddress(boolean on) throws SocketException
```

  - Default is platform dependent; true on Linux and Mac OS X by default
- SO_RCVBUF: set the default receive buffer size for the accepted socket

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

- Class of Service: setPerformancePreferences(), hint for TCP stack

```
public void setPerformancePreferences(int connectionTime, int latency,
    int bandwidth)
```

```
ss.setPerformancePreferences(2, 1, 3);
```

  - Many implementations including Android ignore it