

Java Sockets

CSE 794R/ECE 694R

1

Introduction

- Socket: a bi-directional channel treated as a stream of bytes
 - sender writes to the stream
 - receiver reads from the stream
- Byte encoding requires agreement between sender & receiver
 - character set (ASCII? ISO-LATIN? UTF-16?)
 - big vs little-endian?
 - (java & linux: big endian, windows: little-endian)

CSE 794R/ECE 694R

2

Connection-Oriented Sockets

- reliable (packets get there or receive failure notification)
- in order delivery

CSE 794R/ECE 694R

3

Client Side

- Socket constructor opens (or "connects")
 - `new Socket(String hostname, int port)`
 - exceptions:
 - no host
 - not listening on that port
 - `securityManager.checkConnect()`?
- Use Socket `getInputStream()` method to return stream for reading
- Can read bytes directly, or use a `BufferedReader`

CSE 794R/ECE 694R

4

Server Side

- Create "server side socket", `ServerSocket`
 - new `ServerSocket` (int port, int backlog)
 - associated with a port number (but don't confuse with `Socket`!)
 - backlog: no. of clients that can be waiting to be accepted (rest receive `connectionRefused` exception)
 - `securityManager.checkListen()`?

CSE 794R/ECE 694R

5

Server Side cont.

- Call `accept()` method to accept client connections
 - accepting blocks
 - accepting returns a `Socket`
 - `_returned_ socket` has in & out streams (for sending/receiving)
 - `securityManager.checkAccept()`?

CSE 794R/ECE 694R

6

Multithreaded Server

- A main thread that listens for connection requests
- When accept completes, it spawns a new thread to handle client request

CSE 794R/ECE 694R

7

Serializing Objects: Standard Classes

- Class must implement `java.io.Serializable` (interface)
 - empty interface (marker)
- Sender: use `ObjectOutputStream` and `writeObject()`
- Receiver: use `ObjectInputStream` `readObject()`
 - result of `readObject` is an `Object`
 - cast to narrow

CSE 794R/ECE 694R

8

Serializing Objects: Standard Classes cont.

- If object refers to other objects?
 - members recursively serialized
 - so serializing root ==> entire tree is serialized
 - cycles not a problem
- collections (e.g., `java.util.Map`) implement `Serializable`

CSE 794R/ECE 694R

9

Serializing Objects: Nonstandard Classes

- Modify `Product` class: `Serializable` (and override `toString()`)
- In receiving a `Product` instance over the socket, the `ProductClient` needs the bytecode of this class!
 - should be able to load class
 - what if client doesn't have access to byte code?
 - remote class loader! (from last lecture)
 - jar files can be used too

CSE 794R/ECE 694R

10

Connectionless Sockets

- Lose (compared to connection-oriented):
 - reliability
 - in-order delivery
- Limited payload (64kB)
- Cheaper (no connect / close required)
- Supports multicast too

Classes

- Sockets: DatagramSockets
- Messages: DatagramPackets
 - individually stamped & mailed envelopes

Sender Side

- Create a DatagramSocket
 - bind it to a LOCAL port
- Create a DatagramPacket
 - fill in the payload (message)
 - set the destination address (host + port)
- Call `dsock.send (dpacket)`
 - best effort delivery (doesn't block)

CSE 794R/ECE 694R

13

Receiver Side

- Create a DatagramSocket
 - no need to accept connections from clients!
- Create a DatagramPacket
 - blank (but allocated) payload buffer
- Call `dsock.receive (dpacket)`
 - blocks, returns with cargo of `dpacket` filled in
 - received `dpacket` also has sender info (addr & port)

CSE 794R/ECE 694R

14

Multicast Sockets

- Extend DatagramSocket
- Bound as before: ip address + port
- Use "group" address
224.0.0.0—239.255.255.255 + port

Multicast Sockets cont.

- Multicast socket can be bound to any port
 - datagram address matters
- Default behavior: loopback!
- Two-way communication is fine too

Nonblocking Sockets

- Receives we have seen so far (both TCP & UDP) have been blocking
- `java.nio.channels` package provides non-blocking sockets
 - `ServerSocketChannel`
 - `SocketChannel`
 - `DatagramChannel`
- Create server-side and client-side sockets the same way
- Each can be configured: blocking or non-blocking

CSE 794R/ECE 694R

17

Server Side

- Create `ServerSocketChannel` (with call to static `ServerSocketChannel.open()`)
- Bind the associate `socketChannel` to local port
- Configure the `ssc` (blocking/nonblocking)
- Obtain a `SocketChannel` with `accept`

CSE 794R/ECE 694R

18

Client Side

- Create SocketChannel (with call to static SocketChannel.open())
- Configure the sc (blocking/nonblocking)
- connect() to remote sc

Selectors

- Channels usually are used together with selectors (java.nio.channels.Selector)
- Return "key"s indicating the channel is ready for an op (e.g., accept, connect, read, write)
(java.nio.channels.SelectionKey)

Security

- **SSL (Securer Socket Layer)**
 - javax.net.ssl
 - SSLSocket
 - SSLServerSocket
- **Provides**
 - peer authentication
 - message encryption
 - message integrity