Streamless Socket Access

VisualWorks

Example – From Date Server

Testing

Java JDK 1.4 NIO & Sockets

Date Server Example

How do we Test this?

References

Java On-line API http://java.sun.com/j2se/1.4.1/docs/api/


VisualWorks 7 Source code

NIO On-line Examples http://java.sun.com/j2se/1.4/docs/guide/nio/example/index.html

Martin Kobetic, private communications

Copyright ©, All rights reserved. 2000 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.
Streamless Socket Access

Both VisualWorks and Java (JDK 1.4) support

Reading/writing on sockets without streams

Provides access to more socket functionality
VisualWorks

Supports two versions:

• Read/Write
• Send/Receive

Read/Write

Works for TPC only

Slightly simpler than send/receive

Exceptions raised differ on Window & UNIX

Send/Receive

Works with UDP & TCP
Waiting for Data

SocketAccessor supports:

• readWait

  Blocks until there is data to read on socket

• readWaitWithTimeoutMs: an integer

  Blocks until there is data to read or socket or time out occurs

  Return true if a time out occurred

• writeWait
• writeWaitWithTimeoutMs: an Integer

  Same as read versions but used on writing data

Send/Receive always requires the use of these waits

Read/Write sometimes works without them

So always use them
Read/Write
Example – From Date Server

processRequest: aSocketAccessor
  | timedOut readBuffer charactersRead date |

  [timedOut := aSocketAccessor readWaitWithTimeoutMs: 10000.
   timedOut ifTrue: [^nil].
   readBuffer := String new: 50.
   charactersRead := aSocketAccessor readInto: readBuffer
     untilFalse: [:count | count < 5].
   (readBuffer startsWith: 'date')
     ifTrue:
       [aSocketAccessor writeWait.
        date := Date today printString.
        aSocket_accessor writeFrom: date]]

  ensure: [aSocketAccessor close]
Basic Operation

Before read/write call wait operation ready

In both read & write only part of the data may be processed!

Read/write methods return the number of bytes processed

You code has to make sure all data is processed
Read/Write operations

readInto: aBuffer
readInto: aBuffer startingAt: index for: count
readInto: aBuffer untilFalse: aBlock

writeAll: aBuffer
writeFrom: aBuffer
writeFrom: aBuffer startingAt: index for: count
writeFrom: aBuffer startingAt: index forSure: count
writeFrom: aBuffer startingAt: index for: count untilFalse: aBlock

See Internet Client Developer’s Guide Chapter 2 for details
Testing

How to test processRequest: aSocketAccessor?

Build a mock SocketAccessor

Implement the read/write/wait methods

Read methods read from stream on a String

Write methods write to a write stream on a String

Provide methods to access data written to the mock object
Waiting for Data & Streams

One can do a readWait with streams

```smalltalk
processWithStreamsRequest: aSocketAccessor
    | clientRequest aReadStream aWriteStream |

    [aReadStream := aSocketAccessor readStream lineEndTransparent.
      aWriteStream := aSocketAccessor writeStream lineEndTransparent.

      (aSocketAccessor readWaitWithTimeoutMs: 10000) ifTrue: [^nil].
      clientRequest := aReadStream through: Character cr.
      (clientRequest startsWith: 'date')
      ifTrue:
        [aWriteStream
          nextPutAll: Date today printString;
          commit]
      ensure: [aSocketAccessor close]

    One wants to use

    aReadStream basicAtEnd not or:
      [ (aSocketAccessor readWaitWithTimeoutMs: timeout) not ]

    before reading
```
Java JDK 1.4 NIO & Sockets

JDK 1.4 has streamless access to sockets

Important new classes

• Channels
• Buffers
• Encoders
• Decoders

New packages

• java.nio
• java.nio.channels
• java.nio.charset
Channels

Two-way connection to an IO device

Has

- Blocking IO
- Multiplexed non-blocking IO with selectors

Supports

- Sockets
- Files
- Pipes
Buffers

Channels read/write into buffers

Buffer class for each primitive data type

  Byte, int, float, char, double, long, short

Encoders & Decoders

Maps Unicode strings to/from bytes
import java.io.*;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
import java.nio.charset.*;
import java.util.*;

public class NIOTimeServer
{
    ServerSocketChannel acceptor;

    private static Charset usAscii = Charset.forName("US-ASCII");
    private static CharsetDecoder asciiDecoder = usAscii.newDecoder();
    private static CharsetEncoder asciiEncoder = usAscii.newEncoder();

    public static void main(String[] args) throws IOException {
        int port = Integer.parseInt( args[0]);
        NIOTimeServer server = new NIOTimeServer( port );
        server.run();
    }

    public NIOTimeServer(int port ) throws IOException {
        InetSocketAddress serverAddress =
            new InetSocketAddress(InetAddress.getLocalHost(), port);
        acceptor = ServerSocketChannel.open();
        acceptor.socket().bind( serverAddress );
    }
}
Date Server Example Continued

public void run()
{
    while (true)
    {
        try
        {
            SocketChannel client = acceptor.accept();
            processRequest( client );
        }
        catch (IOException acceptError)
        {
            // for a later lecture
        }
    }
}

void processRequest( SocketChannel client) throws IOException
{
    try
    {
        String request = readLine( client );
        String response = processRequest( request);
        CharBuffer charsOut = CharBuffer.wrap( response + "\n\n");
        ByteBuffer bytesOut = asciiEncoder.encode(charsOut);
        client.write(bytesOut);
    }
    finally
    {
        client.close();
    }
}
Date Server Example Continued

```java
String readLine( SocketChannel client) throws IOException {
    ByteBuffer inputBytes = ByteBuffer.allocate(1024);
    String input = "";
    CharBuffer inputChars;
    while (input.lastIndexOf("\n") < 0 ) {
        inputBytes.clear();
        client.read( inputBytes );
        inputBytes.flip();
        inputChars = asciiDecoder.decode(inputBytes);
        input = input + inputChars.toString();
    }
    return input;
}

String processRequest( String request ) {
    if (request.startsWith("date"))
        return new Date().toString();
    else
        return "";
}
```
Comments

Note the variation of detail in processRequest(SocketChannel client)

Would it be better to have:

```java
try {
    String request = readLine(client);
    String response = processRequest(request);
    write(client, response);
}
```

Does it make sense to have so many methods in such a small example?
How do we Test this?

• Keep the IO separate from the handling of the request
  Allows us to test the logic of handling without IO

• Create a Mock SocketChannel