Additional File Handling Techniques

Contents

1. XML files
2. Java properties files
3. New I/O classes

Demo project:
DemoFileHandlingAdditionalTechniques

2

1. XML Data

- Overview
- Reading documents with StAX
- Creating an `XMLStreamReader`
- Iterating through a document
- Writing documents with StAX
- Creating an `XMLStreamWriter`
- Writing content

3

Overview

- An XML parser is a software component/object that loads XML documents into memory
  - You can process the XML document once it's in memory
- There are two standard types of parser
  - Simple API for XML (SAX)
  - Document Object Model (DOM)
- Java 1.6 also provides the Streaming API for XML (StAX)
  - Pull-processing API
  - Based on the iterator design pattern
  - Read-write access

4

Reading Documents with StAX

- To stream-read XML, use `XMLStreamReader`
  - Represents a cursor that you move through the document from beginning to end
- At any given time, the cursor points to one thing:
  - Start of document, start tag, text node, end tag, etc.
- To get information about the content at the cursor position, call methods such as:
  - `getName()`, `getLocalName`, `getNamespaceURI()`
  - `getText()`, `getElementText()`
  - `getEventType()`, `getLocation()`
  - `getAttributeName()`, `getAttributeValue()`

5

Creating an XMLStreamReader

- To create an XMLStreamReader:

```
import javax.xml.stream.*;
...
XMLInputFactory factory = XMLInputFactory.newInstance();
XMLStreamReader parser  = factory.createXMLStreamReader(
                                    new FileReader(xmlFilename));
```

- Or… you can create an XMLEventReader
  - Uses an event-based approach, similar to SAX
  - (We won't pursue this example here)

```
import javax.xml.stream.*;
...
XMLInputFactory factory = XMLInputFactory.newInstance();
XMLEventReader reader   = factory.createXMLEventReader(
                                    new FileReader(xmlFilename));
```

6

Iterating through a Document

- Use `XMLStreamReader` to iterate the document
  - Use `XMLStreamConstants` to indicate content type

```
for (int event = parser.next();
     event != XMLStreamConstants.END_DOCUMENT;
     event = parser.next())
{
    switch (event)
    {
        case XMLStreamConstants.START_ELEMENT:  …
        case XMLStreamConstants.END_ELEMENT:    …
        case XMLStreamConstants.ATTRIBUTE:      …
        case XMLStreamConstants.CDATA:          …
        case XMLStreamConstants.COMMENT:        …
        case XMLStreamConstants.CHARACTERS:     …
        case XMLStreamConstants.SPACE:          …
        …
    }
}
parser.close();
```

See StaxReaderDemo.java for complete example

Writing Documents with StAX

- To stream-write XML, use `XMLStreamWriter`
  - Enables you to write the content from start to end
- To write content to the document, call methods such as:
  - `writeStartDocument()`, `writeEndDocument()`
  - `writeStartElement()`, `writeEndElement()`
  - `writeAttribute()`, `writeNamespace()`
  - `writeCharacters()`
  - `etc…`

8

Creating an XMLStreamWriter

- To create an XMLStreamWriter:

```
import javax.xml.stream.*;
...
XMLOutputFactory factory = XMLOutputFactory.newInstance();
XMLStreamWriter  writer  = factory.createXMLStreamWriter(
                                    new FileWriter(xmlFilename));
```

Writing Content

- Use `XMLStreamWriter` to write content

```
writer.writeComment("Document created electronically.");
writer.writeStartDocument();

writer.writeStartElement("Employee");
writer.writeNamespace(null, "urn:olsen-software");
writer.writeAttribute("EmpId", "007");

writer.writeStartElement("FirstName");
writer.writeCharacters("James");
writer.writeEndElement();

writer.writeStartElement("LastName");
writer.writeCharacters("Bond");
writer.writeEndElement();

writer.writeEndElement();
writer.writeEndDocument();

writer.flush();
writer.close();
```

```
<!-- Document created electronically. -->
<?xml version="1.0" ?>
<Employee xmlns="urn:olsen-software" EmpId="007">
  <FirstName>James</FirstName>
  <LastName>Bond</LastName>
</Employee>
```

10

2. Java Properties Files

- Overview of Java properties files
- Accessing properties files
- Example

Overview of Java Properties Files

- Java properties files are simple text files, with entries in the following format:
  - propertyName =value
- Here's a sample example, to illustrate various syntax rules for properties file
  - See `wiley.properties`

```
name=Wiley Coyote
location=Arizona
contactNumber=555 111 2222
email=wiley.coyote@acme.com
bio=Wiley is a dedicated and determined coyote. He is sure one day \
    he\'ll catch the roadrunner, then the chickens will come home \
    to roost!
iq=35
```

12

Accessing Properties Files

- The `java.util.Properties` class provides access to Java properties files
- `java.util.Properties` inherits from `Hashtable`
  - i.e. it's a key/value map collection
  - The keys and values are always `Strings`
- You can use `java.util.Properties` to:
  - Load properties from a properties file
  - Get a property value
  - Change a property value
  - Add a new property value
  - Save properties back to the properties file

13

Example

```
import java.util.Properties;
…

public class UsingPropertiesFiles {

  private static final String PROP_FILE = "Wiley.properties";

  public static void main(String[] args) {

  Properties properties = new Properties();

    try {
      properties.load(new FileInputStream(PROP_FILE));

      String name = (String)properties.get("email");
      String iq = (String)properties.get("iq");
      System.out.printf("%s has an iq of %s", name, iq);

      properties.setProperty("iq", "25");
      properties.put("pethate", "Road Runner");

      properties.store(new FileOutputStream(PROP_FILE), "Modified at " + new Date());
    }
    catch (IOException e) {
      System.err.println(e.getMessage());
    }
  }
}
```

14

3. New I/O Classes

- Overview
- Channels and buffers
- Buffer operations
- Buffer state information
- Direct buffers
- Mapped buffers
- File locking
- Online examples

15

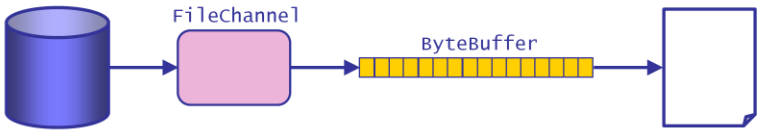
Overview

- Java has always had I/O functionality
  - `java.io` package
- In Java 1.4, Sun introduced the New I/O api (NIO)
  - `java.nio` package
- Low-level I/O operations
  - Channels and buffers
  - Memory-mapped I/O
  - Non-blocking I/O
  - Much improved performance
- File locking
  - Shared and exclusive locks

16

Channels and Buffers (1 of 2)

- Channels are bidirectional sources or sinks of data
  - E.g. `FileChannel` reads/writes a file
  - Data manipulated in blocks
- Buffers are the unit of data moved through channels
  - Basic type of buffer is `ByteBuffer`, also `IntBuffer` etc.
  - Array of data
  - Operations and attributes to simplify data movement



- The Java I/O libraries have been re-implemented to use channels and buffers under the surface
  - And you can use them directly in your code too...

17

Channels and Buffers (2 of 2)

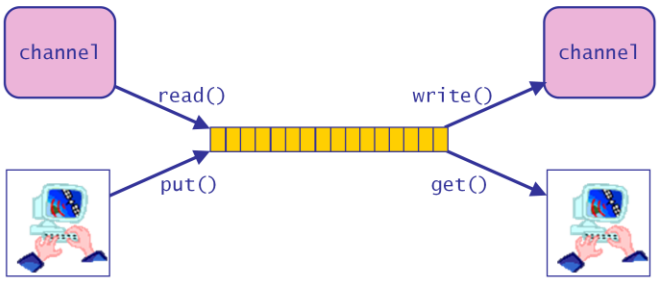
- Example - file copy:

```
public static void demoChannelsAndBuffers(String infilename, String outfilename) {

    try (FileInputStream fis = new FileInputStream(infilename);
         FileOutputStream fos = new FileOutputStream(outfilename)) {

        FileChannel inchannel= fis.getChannel();
        FileChannel outchannel= fos.getChannel();

        // You create a buffer either by calling allocate() or wrap().
        ByteBuffer buf = ByteBuffer.allocate(512);

        while (inchannel.read(buf) >= 0) {

            // Prepare buffer for writing.
            buf.flip();

            outchannel.write(buf);

            // Prepare buffer for reading on next iteration.
            buf.clear();
        }
        System.out.println("Finished file copy.");

    } catch (IOException ex) {
        System.out.println("IOException: " + ex.getMessage());
    }
}
```

NewIO.java 18

Buffer Operations

- To place data into a buffer
  - Invoke channel `read()` method to read data from channel
  - Or invoke buffer `put()` method to put in data manually
- To take data from a buffer
  - Invoke channel `write()` method to write data into channel
  - Or invoke buffer `get()` method to get out data manually



19

Buffer State Information

- 3 buffer attributes describe buffer state…
- position
  - The index where the next read/write begins
  - Invoke `position()` to get or set
- limit
  - Index of first element that should not be read/written
  - Invoke `limit()` to get or set
- capacity
  - Size of underlying array
  - Read-only attribute, set when the buffer is allocated
  - Invoke `capacity()` to get value

20

Direct Buffers

- Direct buffers use system-level data structures and operations
  - Rather than Java arrays
- Major performance improvements
  - No need to copy to/from Java array memory
- To allocate a direct buffer:
  - Invoke `allocateDirect()` rather than `allocate()`
  - No other changes

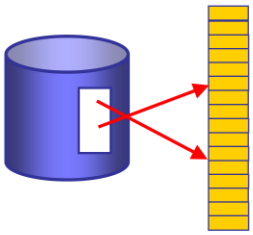
```
ByteBuffer buf = ByteBuffer.allocateDirect(512);
```

- Note
  - You can't create a direct buffer by invoking `wrap()`
  - Why not… ?

21

Mapped Buffers (1 of 2)

- `MappedByteBuffer`
  - Supports mapping file directly into address space
  - Uses a direct buffer under the covers
- Utilizes Windows/Unix sophisticated virtual memory management capabilities
  - Allow files (or portions of files) to be mapped directly to a process' address space
  - Thereafter, accessing a memory location will automatically cause data in the file to be read/written
  - Can give large performance gains



22

Mapped Buffers (2 of 2)

- To create a mapped buffer:
  - Invoke `map()` on a `FileChannel`
  - Specify the portion of file to map, plus level of access required
  - Same operations as `ByteBuffer`
- Example - visit each byte in a file:

```
public static void demoMappedByteBuffer(String infilename) {

    try (FileInputStream fis = new FileInputStream(infilename)) {

        // Create a channel to read from a file.
        FileChannel inchannel = fis.getChannel();

        // Create a MappedByteBuffer to map the entire file into memory.
        long len = new File(infilename).length();
        MappedByteBuffer buf = inchannel.map(FileChannel.MapMode.READ_ONLY, 0, len);

        // Visit every byte in the file
        for (int i = 0; i < len; i++) {
            byte b = buf.get(i);
            System.out.printf("%c", (char)b);
        }
    } catch (IOException ex) { … }
}
```

NewIO.java

23

File Locking (1 of 2)

- Supported on `FileChannel` objects
  - Lock all (or a portion) of a file
- Lock represented by `FileLock` object
  - Dependent on underlying system support
- Shared and exclusive locks available
  - There may be more than one shared lock on a region of a file, but only one exclusive lock
- Treat locks as advisory
  - Locks are enforced through cooperation between competing processes
  - Each process should check the lock before attempting to access a shared portion of a file

24

File Locking (2 of 2)

- To acquire a lock:
  - Invoke `lock()` on a `FileChannel`
  - Thread blocks until lock is available
  - Beware deadlock!

```
FileOutputStream fos = new FileOutputStream(outfilename);
FileChannel outchannel= fos.getChannel();

// Lock file region starting at offset 20 bytes, length 30 bytes, not shared.
FileLock lock = outchannel.lock(20, 30, false);
…
// Release lock.
lock.release();
```

- To try to acquire a lock (without blocking):
  - Invoke `trylock()` on a `FileChannel`
  - Returns `null` if lock cannot be obtained

```
…
FileLock lock = outchannel.trylock(20, 30, false);
if (lock != null) {
    …
    lock.release();
}
```

25

Online Examples (1 of 2)

- Oracle provides excellent examples
  - http://docs.oracle.com/javase/7/docs/technotes/guides/io/example/
- Simple examples:
  - Sum.java - Using NIO mapped byte buffers (fast memory usage)
  - Grep.java - Regular expressions, charsets, and mapped buffers
  - Ping.java - Non-blocking socket connections, multithreading

Online Examples (2 of 2)

- The samples illustrate some more advanced NIO techniques:
  - TimeQuery.java
    - Blocking client, using NIO socket channels (connecting and reading)
    - Uses buffers, charsets, and regular expressions
  - TimeServer.java
    - Blocking server, using NIO socket channels (accepting and writing)
    - Uses buffers, charsets, and regular expressions
  - NBTimeServer.java
    - Non-blocking Internet time server

27

Any Questions?