CS 580 Client-Server Programming
Fall Semester, 2005
Doc 12 Configuration Files & Logging
Contents

References

SDSU Java Library, http://www.eli.sdsu.edu/java-SDSU/docs/

Java Logging Overview, http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html

Java Logging API http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/package-summary.html

Patterns for Logging Diagnostic Messages by Neil Harrison in Pattern Languages of Program Design 3 Eds Martin, Riehle, Buschman, 1998, pp 277-289

SDSU Java Library, http://www.eli.sdsu.edu/java-SDSU/docs/

Application Parameters & Configuration Files

Applications normal have configuration files to store

- User preferences
- Cached values
- Window settings
- Port numbers
- Database connection information
- Log file information
- Recent documents/web pages
- Cookies
- Values that need changing without recompiling

Some programs use environment variables

```
cvs co assignment2
```

Command line program/servers/utilities have flags

```
ls -la
ps -aux
```

Servers

Servers normally use configuration files & command line flags

Environment variables are uses much in servers (why?)

- Flag values override configuration file values
- Configuration file values override defaults in the code

Java

Some systems have libraries to handle config files & command line arguments

JDK does not seem to have such classes

There should be a number of Java libraries that provide such support

sdsu Java library is one such library

sdsu.util.ProgramProperties

Parses

Configuration files
Command line arguments

Command Line argument

-flag=value
-flag value
-flag
--xyz
-- (ignore rest of the command line)

File Formats

properties format

```
#A comment to the end of the line
key1=value1
key2=value2     with spaces
key3 with spaces=value3 #part of the value
```

sdsu.util.LabeledData format

```
#A comment to the end of the line,
key1 = value1;
key2='value2    with spaces';
'key3 with spaces'=value3;            # a comment
```

Simple Example

```
import sdsu.util.ProgramProperties;

public class ConfigurationExample {

   public static void main(String args[]) {
      try
         {
         ProgramProperties flags =
            new ProgramProperties( args, "configurationFile");
         String nameValue =
            flags.getString( "name" , "No name given");
         int size = flags.getInt( "size", 0);
         boolean switchOn = flags.containsKey( "s");
         System.out.println( " nameValue: " + nameValue);
         System.out.println( " size: " + size);
         System.out.println( " switchOn: " + switchOn);

         }
      catch (java.io.IOException readParseProblem)
         {
         System.err.println( "Program aborted on error " +
         readParseProblem);
         }
   }
}
```

File "configurationFile.labeledData"

```
name=Roger;
size=12;
```

Sample Runs

java ConfigurationExample

Output

nameValue: Roger
size: 12
switchOn: false

java ConfigurationExample -s -name Pete

Output

nameValue: Pete
size: 12
switchOn: true

java ConfigurationExample -conf=otherFile

Output

nameValue: Sam
size: 8
switchOn: true

Why logging?

- Performance tuning
- Upgrade justification
- Problem tracking
- Access counting

How many times was the assignment 1 server accessed?

What should be logged?

Normally a log entry contains several pieces of information:

- Date and time
- Service that caused the entry
- Client address that caused the entry
- Host on which the server runs
- Event

Log File Examples

Apache
Access

211.90.88.43 - - [21/Oct/2002:08:33:29 -0700] "GET /scripts/..%25%35%63../winnt/system32/cmd.exe?/c+dir HTTP/1.0" 404 303
211.90.88.43 - - [21/Oct/2002:08:33:30 -0700] "GET /scripts/..%252f../winnt/system32/cmd.exe?/c+dir HTTP/1.0" 404 303

Error

[Mon Oct 21 08:33:29 2002] [error] [client 211.90.88.43] File does not exist: /opt/etc/apache-1.3.26/htdocs/scripts/..%5c../winnt/system32/cmd.exe
[Mon Oct 21 08:33:30 2002] [error] [client 211.90.88.43] File does not exist: /opt/etc/apache-1.3.26/htdocs/scripts/..%2f../winnt/system32/cmd.exe

Note that the log files contain the client IP address not the name of the machine

Dnews – A News Server

Log files

- dnews.in
- dnews.log
- dnews.out
- dnews_post.log
- expire.log
- status.log
- users.log

Configuration files

- access.conf
- control.conf
- expire.conf
- fts.conf
- hp.conf
- moderators.conf
- newsfeeds.conf

Dnews Log file examples

users.log

```
Mon Oct 21 04:50:23 Bytes Read,  Current Connections,
Area/IPrange
Mon Oct 21 04:50:23      197k          1   194.170.42.68
Mon Oct 21 04:50:23      197k          1   byu
Mon Oct 21 04:50:23   0mb              2   Grand Totals
```

dnews.log

```
21 09:41:27  :info: db_piles_trim called
21 09:42:08  :info: db_pile_status
21 09:42:08  :info: db_pile_status_done
21 09:42:08  :info: dnews_status 1
21 09:42:08  :info: dnews_status 2
21 09:42:08  :info: dnews_status 3
21 09:42:08  :info: Timezone Offset 25200 dst 3600
21 09:42:16  :info: Writing list of read groups to a file for this
slave
21 09:42:16  :info: Wrote 17 records
21 09:42:28  :info: db_piles_trim called
```

Log file formats

Log files get big

Assignment1 log file was 12.8MB

Web server logs get 10 - 100 Mbs before rotating the file

Make the log file machine parsable!

How should clients and servers log?

Basic choices:

- Appending to a log file
- System logging facility

Simple appending problems

This simplistic approach can cause problems with the following:

- Concurrency

  Two server threads concurrently writing to the same logfile

- Performance

  Every log entry requires lots of overhead (opening and closing the logfile)

Some solutions:

- Rely on the OS to perform atomic appends to the log file. (One unbuffered write per log entry)

- Use a dedicated logging thread (Keep the log file open)

- Synchronize the writing to the log file.

JDK 1.4 Logging

Starting with JDK 1.4 Java API has a logging system

Supports
- Multiple log levels
- Multiple output formats
- Output to different IO devices

- Filters for additional filtering of message to accept
- ResourceBundles for localization of log messages
- Initialization of loggers by configuration file
- Hierarchical loggers in one program

Log Levels

- ALL
- SEVERE (highest value)
- WARNING
- INFO (usual default)
- CONFIG
- FINE
- FINER
- FINEST (lowest value)
- OFF

Output formats

- XML (default for files output)
- Normal Text (default for screen output)

Output devices

- Stream
- System.err
- File or rotating set of files
- Socket for network logging
- Memory

SimpleExample

```
import java.util.logging.*;

public class SimpleLoggingExample
   {
   private static Logger logger = Logger.getLogger("edu.sdsu.cs580");

   public static void main (String args[])
      {
      new SimpleLoggingExample().someLogMessages();
      }

   public void someLogMessages()
      {
      logger.severe( "A severe log message");
      Logger.getLogger("edu.sdsu.cs580").fine( "A fine message");
      logger.warning( "Be careful" );
      }
   }
```

Output To System.err

```
Feb 16, 2004 10:51:37 PM Logging someLogMessages
SEVERE: A severe log message
Feb 16, 2004 10:51:37 PM Logging someLogMessages
WARNING: Be careful
```

Default Settings

Use a ConsoleHandler
Level set to INFO
System administrator can change default settings

Five Categories of Logging Messages
Convenience Methods

- severe(String message);
- warning(String message);
- info(String message);
- config(String message);
- fine(String message);
- finer(String message);
- finest(String message);

Convenience Methods for Tracing Methods

entering(String sourceClass, String sourceMethod);
entering(String sourceClass, String sourceMethod, Object parameter);
entering(String sourceClass, String sourceMethod, Object[] parameters);
exiting(String sourceClass, String sourceMethod);
exiting(String sourceClass, String sourceMethod, Object result);

Log Methods

log(Level logLevel, String message);
log(Level logLevel, String message, Object parameter);
log(Level logLevel, String message, Object[] parameters);
log(Level logLevel, String message, Throwable exception);

Currently parameters argument is ignored

Precise Log Methods

Add class and method to log messages

logp(Level logLevel, String class, String method,String message);
etc.

Logs with Resource Bundles

Add resource bundle to logp messages

logrb(Level logLevel, String class, String method, String bundlename,
String message);
etc.

Example of Different Message Types

```
import java.io.*;
import java.util.Vector;
import java.util.logging.*;

public class MessageTypes
   {
   private static Logger logger = Logger.getLogger("edu.sdsu.cs580");

   static
      {
      try
         {
         Handler textLog = new FileHandler("textLog.txt");
         textLog.setFormatter( new SimpleFormatter());
         textLog.setLevel(Level.ALL);
         Handler xmlLog = new FileHandler("xmlLog.txt");
         xmlLog.setFormatter( new XMLFormatter());
         xmlLog.setLevel(Level.ALL);

         logger.addHandler(textLog);
         logger.addHandler(xmlLog);
         logger.setLevel(Level.ALL);
         }
      catch (IOException fileError)
         {
         System.err.println( "Could not open log files");
         }
      }
```

Example Continued

```
public static void main (String args[])
      {
      new MessageTypes().someLogMessages();
      }

   public void someLogMessages()
      {
      logger.entering("MessageTypes", "someLogMessages");
        Vector data = new Vector();
      data.add( "Cat");
      logger.log(Level.SEVERE, "Show Vector", data);
      logger.severe( "A severe log message");
      logger.logp(Level.SEVERE, "MessageTypes", "someLogMessages", "Logp
example");
      try
         {
         int zeroDivide = 1/ (1 - 1);
         }
      catch (Exception zeroDivide)
         {
         logger.log(Level.SEVERE, "Exception example", zeroDivide);
         }
   logger.exiting("MessageTypes", "someLogMessages");
      }
   }
```

SimpleFormatter Output

```
Feb 16, 2004 11:01:53 PM MessageTypes someLogMessages
FINER: ENTRY
Feb 16, 2004 11:01:53 PM MessageTypes someLogMessages
SEVERE: Show Vector
Feb 16, 2004 11:01:53 PM MessageTypes someLogMessages
SEVERE: A severe log message
Feb 16, 2004 11:01:54 PM MessageTypes someLogMessages
SEVERE: Logp example
Feb 16, 2004 11:01:54 PM MessageTypes someLogMessages
SEVERE: Exception example
java.lang.ArithmeticException: / by zero
        at MessageTypes.someLogMessages(MessageTypes.java:45)
        at MessageTypes.main(MessageTypes.java:32)
Feb 16, 2004 11:01:54 PM MessageTypes someLogMessages
FINER: RETURN
```

XMLFormatter Sample Output

```
<?xml version="1.0" encoding="US-ASCII" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2004-02-16T23:01:53</date>
  <millis>1077001313695</millis>
  <sequence>0</sequence>
  <logger>edu.sdsu.cs580</logger>
  <level>FINER</level>
  <class>MessageTypes</class>
  <method>someLogMessages</method>
  <thread>10</thread>
  <message>ENTRY</message>
</record>
```

FileHandlers

Can be set to rotate files

Can be located in temp directory

Can be set to
- Append existing files
- Overwrite existing files (default)

To change append setting either

- Use constructor

  FileHandler(String pattern, boolean append)

- Or use configuration file

Loggers

Can have

- Multiple handlers
- Multiple handlers of same type

Loggers and handlers have differ log levels

Logger
- Drops all messages below it log level
- Passes remaining messages to all handlers
- Handler can further drop more messages

Logger Names

Logger names are arbitrary

```
Logger.getLogger("edu.sdsu.cs580")
Logger.getLogger("foo")
Logger.getLogger("")
```

Sun recommends using hierarchical names with format

"domain.package"
"domain.package.class"

Loggers inherit settings from “parent” logger

Logger "edu.sdsu.cs580" would inherit settings of "edu.sdsu"

Debug Example

```
import java.io.*;
import java.util.logging.*;
public class Logging
   {
   private static Logger logger = Logger.getLogger("edu.sdsu.cs580");
   private static Logger debug = Logger.getLogger("debug");
   // screen loging is default, as in INFO level

   static
      {
      try
         {
         // Parent sends log to the screen
         logger.setUseParentHandlers( false);

         Handler textLog = new FileHandler("textLog.txt");
         textLog.setLevel(Level.ALL);

         logger.addHandler(textLog);
         logger.setLevel(Level.ALL);

         debug.setUseParentHandlers( false);
         Handler screenLog = new ConsoleHandler();
         debug.addHandler( screenLog);
         debug.setLevel( Level.INFO );
         }
      catch (IOException fileError)
         {
         System.err.println( "Could not open log files");
         }
      }
```

Example continued

```
public static void main (String args[])
    {
    new Logging().someLogMessages();
    }

 public void someLogMessages()
    {
    logger.info( "A severe log message");
    debug.info( "this is a debug statement");
    }
 }
```

Logger Scope

Logger settings can be defined in

- Program
- Configuration File

Logger settings defined in a program exist only in that program

Logger settings defined in a configuration file can be used by multiple programs

Sample Configuration File

```
# Use two loggers
handlers= java.util.logging.FileHandler, java.util.logging.ConsoleHandler

# Default global logging level.
.level= WARNING

# File logger default settings
# Default file output is in user's home directory (%h/).
# %g – use generation numbers to distinguish rotated logs
# limit = max size of each log file
# count = number of output files to cycle through
java.util.logging.FileHandler.pattern = %h/cs580Server%g.log
java.util.logging.FileHandler.limit = 50000
java.util.logging.FileHandler.count = 3
java.util.logging.FileHandler.formatter = java.util.logging.XMLFormatter

# Limit the message that are printed on the console to INFO and above.
java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter


# Set levels of specific loggers
edu.sdsu.level = SEVERE
edu.sdsu.cs580.level = INFO
```

Using the Configuration File

Assume that configuration file is in

- Local directory
- In a file called cs580Log.properties

The following command will use the configuration file

```
java -Djava.util.logging.config.file=cs580Log.properties  yourClassGoesHere
```