Changeset 2

Timestamp: 2013-10-06 21:50:14 (7 days ago)
Author: coonsdw

Message: Implemented "Add timestamps to log" by replacing calls to write to the log with a method call that prepends the system date.

File: 1 edited

org/gjt/sp/util/Log.java (1 diff)

Unmodified Added Removed

org/gjt/sp/util/Log.java Tabular | Unified

r1	r2	
1	1	/*
2	2	* Log.java - A class for logging events
3	3	* :tabSize=8:indentSize=8:noTabs=false:
4	4	* :folding=explicit:collapseFolds=1:
5	5	*
6	6	* Copyright (C) 1999, 2003 Slava Pestov
7	7	*
8	8	* This program is free software; you can redistribute it and/or
9	9	* modify it under the terms of the GNU General Public License
10	10	* as published by the Free Software Foundation; either version 2
11	11	* of the License, or any later version.
12	12	*
13	13	* This program is distributed in the hope that it will be useful,
14	14	* but WITHOUT ANY WARRANTY; without even the implied warranty of
15	15	* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16	16	* GNU General Public License for more details.
17	17	*
18	18	* You should have received a copy of the GNU General Public License
19	19	* along with this program; if not, write to the Free Software
20	20	* Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
21	21	*/
22	22	
23	23	package org.gjt.sp.util;
24	24	
25	25	import java.io.*;
26	26	import java.util.*;
27	27	import javax.swing.*;
28	28	import javax.swing.event.*;
29	29	
30	30	/**
31	31	* This class provides methods for logging events. In terms of functionality,
32	32	* it is somewhere in between <code>System.out.println()</code> and
33	33	* full-blown logging packages such as log4j.<p>
34	34	*
35	35	* All events are logged to an in-memory buffer and optionally a stream,
36	36	* and those with a high urgency (warnings and errors) are also printed
37	37	* to standard output.<p>
38	38	*
39	39	* Logging of exception tracebacks is supported.<p>
40	40	*
41	41	* This class can also optionally redirect standard output and error to the log.
42	42	*
43	43	* @author Slava Pestov
44	44	* @version $Id: Log.java 8347 2007-01-11 21:13:27Z kpouer $
45	45	*/
46	46	public class Log

```
47  47  {
48  48      //{{{ Constants
49  49      /**
50  50       * The maximum number of log messages that will be kept in memory.
51  51       * @since jEdit 2.6pre5
52  52       */
53  53      public static final int MAXLINES = 500;
54  54
55  55      /**
56  56       * Debugging message urgency. Should be used for messages only
57  57       * useful when debugging a problem.
58  58       * @since jEdit 2.2pre2
59  59       */
60  60      public static final int DEBUG = 1;
61  61
62  62      /**
63  63       * Message urgency. Should be used for messages which give more
64  64       * detail than notices.
65  65       * @since jEdit 2.2pre2
66  66       */
67  67      public static final int MESSAGE = 3;
68  68
69  69      /**
70  70       * Notice urgency. Should be used for messages that directly
71  71       * affect the user.
72  72       * @since jEdit 2.2pre2
73  73       */
74  74      public static final int NOTICE = 5;
75  75
76  76      /**
77  77       * Warning urgency. Should be used for messages that warrant
78  78       * attention.
79  79       * @since jEdit 2.2pre2
80  80       */
81  81      public static final int WARNING = 7;
82  82
83  83      /**
84  84       * Error urgency. Should be used for messages that signal a
85  85       * failure.
86  86       * @since jEdit 2.2pre2
87  87       */
88  88      public static final int ERROR = 9;
89  89      //}}}
90  90
91  91      //{{{ init() method
92  92      /**
93  93       * Initializes the log.
94  94       * @param stdio If true, standard output and error will be
95  95       * sent to the log
96  96       * @param level Messages with this log level or higher will
97  97       * be printed to the system console
98  98       * @since jEdit 3.2pre4
99  99       */
100 100     public static void init(boolean stdio, int level)
101 101     {
102 102         if(stdio)
103 103         {
104 104             if(System.out == realOut && System.err == realErr)
105 105             {
106 106                 System.setOut(createPrintStream(NOTICE,null));
107 107                 System.setErr(createPrintStream(ERROR,null));
108 108             }
109 109         }
110 110
111 111         Log.level = level;
112 112
113 113         // Log some stuff
114 114         log(MESSAGE,Log.class,"When reporting bugs, please"
```

```
115 115             + " include the following information:");
116 116         String[] props = {
117 117             "java.version", "java.vm.version", "java.runtime.version",
118 118             "java.vendor", "java.compiler", "os.name", "os.version",
119 119             "os.arch", "user.home", "java.home",
120 120             "java.class.path",
121 121             };
122 122         for(int i = 0; i < props.length; i++)
123 123         {
124 124             log(MESSAGE,Log.class,
125 125                 props[i] + '=' + System.getProperty(props[i]));
126 126         }
127 127     } //}}}
128 128
129 129     //{{{ setLogWriter() method
130 130     /**
131 131      * Writes all currently logged messages to this stream if there was no
132 132      * stream set previously, and sets the stream to write future log
133 133      * messages to.
134 134      * @param stream The writer
135 135      * @since jEdit 3.2pre4
136 136      */
137 137     public static void setLogWriter(Writer stream)
138 138     {
139 139         if(Log.stream == null && stream != null)
140 140         {
141 141             try
142 142             {
143 143                 if(wrap)
144 144                 {
145 145                     for(int i = logLineCount; i < log.length; i++)
146 146                     {
147     -                       stream.write(log[i]);
148     -                       stream.write(lineSep);
    147 +                       writeLineToStream(log[i]);
149 148                     }
150 149                 }
151 150                 for(int i = 0; i < logLineCount; i++)
152 151                 {
153     -                   stream.write(log[i]);
154     -                   stream.write(lineSep);
    152 +                   writeLineToStream(log[i]);
155 153                 }
156 154
157 155                 stream.flush();
158 156             }
159 157             catch(Exception e)
160 158             {
161 159                 // do nothing, who cares
162 160             }
163 161         }
164 162
165 163         Log.stream = stream;
166 164     } //}}}
167 165
168 166     //{{{ flushStream() method
169 167     /**
170 168      * Flushes the log stream.
171 169      * @since jEdit 2.6pre5
172 170      */
173 171     public static void flushStream()
174 172     {
175 173         if(stream != null)
176 174         {
177 175             try
```

```
178 176             {
179 177                 stream.flush();
180 178             }
181 179             catch(IOException io)
182 180             {
183 181                 io.printStackTrace(realErr);
184 182             }
185 183         }
186 184     } //}}}
187 185
188 186     //{{{ closeStream() method
189 187     /**
190 188      * Closes the log stream. Should be done before your program exits.
191 189      * @since jEdit 2.6pre5
192 190      */
193 191     public static void closeStream()
194 192     {
195 193         if(stream != null)
196 194         {
197 195             try
198 196             {
199 197                 stream.close();
200 198                 stream = null;
201 199             }
202 200             catch(IOException io)
203 201             {
204 202                 io.printStackTrace(realErr);
205 203             }
206 204         }
207 205     } //}}}
208 206
209 207     //{{{ getLogListModel() method
210 208     /**
211 209      * Returns the list model for viewing the log contents.
212 210      * @since jEdit 4.2pre1
213 211      */
214 212     public static ListModel getLogListModel()
215 213     {
216 214         return listModel;
217 215     } //}}}
218 216
219 217     //{{{ log() method
220 218     /**
221 219      * Logs an exception with a message.
222 220      *
223 221      * If an exception is the cause of a call to {@link #log}, then
224 222      * the exception should be explicitly provided so that it can
225 223      * be presented to the (debugging) user in a useful manner
226 224      * (not just the exception message, but also the exception stack trace)
227 225      *
228 226      * @since jEdit 4.3pre5
229 227      */
230 228     public static void log(int urgency, Object source, Object message,
231 229         Throwable exception)
232 230     {
233 231         // We can do nicer here, but this is a start...
234 232         log(urgency,source,message);
235 233         log(urgency,source,exception);
236 234     } //}}}
237 235
238 236     //{{{ log() method
239 237     /**
240 238      * Logs a message. This method is thread-safe.<p>
241 239      *
242 240      * The following code sends a typical debugging message to the activity
243 241      * log:
244 242      * <pre>Log.log(Log.DEBUG,this,"counter = " + counter);</pre>
```

```
245 243      * The corresponding activity log entry might read as follows:
246 244      * <pre>[debug] JavaParser: counter = 15</pre>
247 245      *
248 246      * @param urgency The urgency; can be one of
249 247      * <code>Log.DEBUG</code>, <code>Log.MESSAGE</code>,
250 248      * <code>Log.NOTICE</code>, <code>Log.WARNING</code>, or
251 249      * <code>Log.ERROR</code>.
252 250      * @param source The source of the message, either an object or a
253 251      * class instance. When writing log messages from macros, set
254 252      * this parameter to <code>BeanShell.class</code> to make macro
255 253      * errors easier to spot in the activity log.
256 254      * @param message The message. This can either be a string or
257 255      * an exception
258 256      *
259 257      * @since jEdit 2.2pre2
260 258      */
261 259     public static void log(int urgency, Object source, Object message)
262 260     {
263 261         String _source;
264 262         if(source == null)
265 263         {
266 264             _source = Thread.currentThread().getName();
267 265             if(_source == null)
268 266             {
269 267                 _source = Thread.currentThread().getClass().getName();
270 268             }
271 269         }
272 270         else if(source instanceof Class)
273 271             _source = ((Class)source).getName();
274 272         else
275 273             _source = source.getClass().getName();
276 274         int index = _source.lastIndexOf('.');
277 275         if(index != -1)
278 276             _source = _source.substring(index+1);
279 277
280 278         if(message instanceof Throwable)
281 279         {
282 280             _logException(urgency,source,(Throwable)message);
283 281         }
284 282         else
285 283         {
286 284             String _message = String.valueOf(message);
287 285             // If multiple threads log stuff, we don't want
288 286             // the output to get mixed up
289 287             synchronized(LOCK)
290 288             {
291 289                 StringTokenizer st = new StringTokenizer(
292 290                     _message,"\r\n");
293 291                 int lineCount = 0;
294 292                 boolean oldWrap = wrap;
295 293                 while(st.hasMoreTokens())
296 294                 {
297 295                     lineCount++;
298 296                     _log(urgency,_source,st.nextToken()
299 297                         .replace('\t',' '));
300 298                 }
301 299                 listModel.update(lineCount,oldWrap);
302 300             }
303 301         }
304 302     } //}}}
305 303
306 304     //{{{ Private members
307 305
308 306     //{{{ Instance variables
309 307     private static final Object LOCK = new Object();
310 308     private static final String[] log;
311 309     private static int logLineCount;
```

```
312 310     private static boolean wrap;
313 311     private static int level = WARNING;

314 312     private static Writer stream;
315 313     private static final String lineSep;

316 314     private static final PrintStream realOut;
317 315     private static final PrintStream realErr;
318 316     private static final LogListModel listModel;
319 317     //}}}
320 318
321 319     //{{{ Class initializer
322 320     static
323 321     {
324 322         level = WARNING;
325 323
326 324         realOut = System.out;
327 325         realErr = System.err;
328 326
329 327         log = new String[MAXLINES];
330 328         lineSep = System.getProperty("line.separator");
331 329         listModel = new LogListModel();
332 330     } //}}}
333 331
334 332     //{{{ createPrintStream() method
335 333     private static PrintStream createPrintStream(final int urgency,
336 334         final Object source)
337 335     {
338 336         return new LogPrintStream(urgency, source);
339 337     } //}}}
340 338
341 339     //{{{ _logException() method
342 340     private static void _logException(final int urgency,
343 341         final Object source,
344 342         final Throwable message)
345 343     {
346 344         PrintStream out = createPrintStream(urgency,source);
347 345
348 346         synchronized(LOCK)
349 347         {
350 348             message.printStackTrace(out);
351 349         }
352 350     } //}}}
353 351
354 352     //{{{ _log() method
355 353     private static void _log(int urgency, String source, String message)
356 354     {
357 355         String fullMessage = '[' + urgencyToString(urgency) + "] " + source
358 356             + ": " + message;
359 357
360 358         try
361 359         {
362 360             log[logLineCount] = fullMessage;
363 361             if(++logLineCount >= log.length)
364 362             {
365 363                 wrap = true;
366 364                 logLineCount = 0;
367 365             }
368 366
369 367             if(stream != null)
370 368             {
371             stream.write(fullMessage);
372             stream.write(lineSep);
    369             writeLineToStream(fullMessage);
373 370             }
374 371         }
375 372         catch(Exception e)
```

```
                catch(Exception e)
                {
                    e.printStackTrace(realErr);
                }

                if(urgency >= level)
                {
                    if(urgency == ERROR)
                        realErr.println(fullMessage);
                    else
                        realOut.println(fullMessage);
                }
            } //}}}

            //{{{ urgencyToString() method
            private static String urgencyToString(int urgency)
            {
                switch(urgency)
                {
                case DEBUG:
                    return "debug";
                case MESSAGE:
                    return "message";
                case NOTICE:
                    return "notice";
                case WARNING:
                    return "warning";
                case ERROR:
                    return "error";
                }

                throw new IllegalArgumentException("Invalid urgency: " + urgency);
            } //}}}

            //}}}

            //{{{ LogListModel class
            static class LogListModel implements ListModel
            {
                final List<ListDataListener> listeners = new ArrayList<ListDataListener>();

                private void fireIntervalAdded(int index1, int index2)
                {
                    for(int i = 0; i < listeners.size(); i++)
                    {
                        ListDataListener listener = listeners.get(i);
                        listener.intervalAdded(new ListDataEvent(this,
                            ListDataEvent.INTERVAL_ADDED,
                            index1,index2));
                    }
                }

                private void fireIntervalRemoved(int index1, int index2)
                {
                    for(int i = 0; i < listeners.size(); i++)
                    {
                        ListDataListener listener = listeners.get(i);
                        listener.intervalRemoved(new ListDataEvent(this,
                            ListDataEvent.INTERVAL_REMOVED,
                            index1,index2));
                    }
                }

                public void addListDataListener(ListDataListener listener)
                {
                    listeners.add(listener);
                }
```

```
443 440         public void removeListDataListener(ListDataListener listener)
444 441         {
445 442             listeners.remove(listener);
446 443         }
447 444
448 445         public Object getElementAt(int index)
449 446         {
450 447             if(wrap)
451 448             {
452 449                 if(index < MAXLINES - logLineCount)
453 450                     return log[index + logLineCount];
454 451                 else
455 452                     return log[index - MAXLINES + logLineCount];
456 453             }
457 454             else
458 455                 return log[index];
459 456         }
460 457
461 458         public int getSize()
462 459         {
463 460             if(wrap)
464 461                 return MAXLINES;
465 462             else
466 463                 return logLineCount;
467 464         }
468 465
469 466         void update(final int lineCount, final boolean oldWrap)
470 467         {
471 468             if(lineCount == 0 || listeners.isEmpty())
472 469                 return;
473 470
474 471             SwingUtilities.invokeLater(new Runnable()
475 472             {
476 473                 public void run()
477 474                 {
478 475                     if(wrap)
479 476                     {
480 477                         if(oldWrap)
481 478                             fireIntervalRemoved(0,lineCount - 1);
482 479                         else
483 480                         {
484 481                             fireIntervalRemoved(0,
485 482                                 logLineCount);
486 483                         }
487 484                         fireIntervalAdded(
488 485                             MAXLINES - lineCount + 1,
489 486                             MAXLINES);
490 487                     }
491 488                     else
492 489                     {
493 490                         fireIntervalAdded(
494 491                             logLineCount - lineCount + 1,
495 492                             logLineCount);
496 493                     }
497 494                 }
498 495             });
499 496         }
500 497     } //}}}
501 498
502 499     /**
503 500      * A print stream that uses the "Log" class to output the messages,
504 501      * and has special treatment for the printf() function. Using this
505 502      * stream has one caveat: printing messages that don't have a line
506 503      * break at the end will have one added automatically...
507 504      */
508 505     private static class LogPrintStream extends PrintStream {
```

```
509 506
510 507         private final ByteArrayOutputStream buffer;
511 508         private final OutputStream orig;
512 509
513 510         LogPrintStream(int urgency, Object source)
514 511         {
515 512             super(new LogOutputStream(urgency, source));
516 513             buffer = new ByteArrayOutputStream();
517 514             orig = out;
518 515         }
519 516
520 517         /**
521 518          * This is a hack to allow "printf" to not print weird
522 519          * stuff to the output. Since "printf" doesn't seem to
523 520          * print the whole message in one shot, our output
524 521          * stream above would break a line of log into several
525 522          * lines; so we buffer the result of the printf call and
526 523          * print the whole thing in one shot. A similar hack
527 524          * would be needed for the "other" printf method, but
528 525          * I'll settle for the common case only.
529 526          */
530 527         public PrintStream printf(String format, Object... args)
531 528         {
532 529             synchronized (orig)
533 530             {
534 531                 buffer.reset();
535 532                 out = buffer;
536 533                 super.printf(format, args);
537 534
538 535                 try
539 536                 {
540 537                     byte[] data = buffer.toByteArray();
541 538                     orig.write(data, 0, data.length);
542 539                     out = orig;
543 540                 }
544 541                 catch (IOException ioe)
545 542                 {
546 543                     // don't do anything?
547 544                 }
548 545                 finally
549 546                 {
550 547                     buffer.reset();
551 548                 }
552 549             }
553 550             return this;
554 551         }
555 552     }
556 553
557 554     private static class LogOutputStream extends OutputStream
558 555     {
559 556         private final int   urgency;
560 557         private final Object    source;
561 558
562 559         LogOutputStream(int urgency, Object source)
563 560         {
564 561             this.urgency    = urgency;
565 562             this.source     = source;
566 563         }
567 564
568 565         public synchronized void write(int b)
569 566         {
570 567             byte[] barray = { (byte)b };
571 568             write(barray,0,1);
572 569         }
573 570
```

```
574 571         public synchronized void write(byte[] b, int off, int len)
575 572         {
576 573             String str = new String(b,off,len);
577 574             log(urgency,source,str);
578 575         }
579 576     }
580 
    577 
    578     /**
    579      * This is a wrapper for the stream.write method.  It prepends the current date, and appends a line feed
    580      *
    581      * @param text  Text to write to the stream
    582      * @throws IOException
    583      */
    584     private static void writeLineToStream(String text) throws IOException
    585     {
    586         Date date = new Date();
    587         stream.write(date.toString() + ": ");
    588         stream.write(text);
    589         stream.write(lineSep);
    590     }
581 591 }
582 592 
```