001/\*  
002 \* Licensed to the Apache Software Foundation (ASF) under one or more  
003 \* contributor license agreements. See the NOTICE file distributed with  
004 \* this work for additional information regarding copyright ownership.  
005 \* The ASF licenses this file to You under the Apache License, Version 2.0  
006 \* (the "License"); you may not use this file except in compliance with  
007 \* the License. You may obtain a copy of the License at  
008 \*  
009 \* http://www.apache.org/licenses/LICENSE-2.0  
010 \*  
011 \* Unless required by applicable law or agreed to in writing, software  
012 \* distributed under the License is distributed on an "AS IS" BASIS,  
013 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
014 \* See the License for the specific language governing permissions and  
015 \* limitations under the License.  
016 \*/  
017  
018  
019package org.apache.commons.beanutils;  
020  
021import java.lang.reflect.InvocationTargetException;  
022import java.util.Map;  
023  
024  
025  
026/\*\*  
027 \* <p>Utility methods for populating JavaBeans properties via reflection.</p>  
028 \*  
029 \* <p>The implementations are provided by {@link BeanUtilsBean}.  
030 \* These static utility methods use the default instance.  
031 \* More sophisticated behaviour can be provided by using a <code>BeanUtilsBean</code> instance.</p>  
032 \*  
033 \* @version $Id$  
034 \* @see BeanUtilsBean  
035 \*/  
036  
037public class BeanUtils {  
038  
039  
040 // ------------------------------------------------------ Private Variables  
041  
042  
043 /\*\*  
044 \* The debugging detail level for this component.  
045 \*  
046 \* Note that this static variable will have unexpected side-effects if  
047 \* this class is deployed in a shared classloader within a container.  
048 \* However as it is actually completely ignored by this class due to its  
049 \* deprecated status, it doesn't do any actual harm.  
050 \*  
051 \* @deprecated BeanUtils now uses commons-logging for all log messages.  
052 \* Use your favorite logging tool to configure logging for  
053 \* this class.  
054 \*/  
055 @Deprecated  
056 private static int debug = 0;  
057  
058 /\*\*  
059 \* The <code>debug</code> static property is no longer used  
060 \* @return debug property  
061 \* @deprecated BeanUtils now uses commons-logging for all log messages.  
062 \* Use your favorite logging tool to configure logging for  
063 \* this class.  
064 \*/  
065 @Deprecated  
066 public static int getDebug() {  
067 return (debug);  
068 }  
069  
070 /\*\*  
071 \* The <code>debug</code> static property is no longer used  
072 \* @param newDebug debug property  
073 \* @deprecated BeanUtils now uses commons-logging for all log messages.  
074 \* Use your favorite logging tool to configure logging for  
075 \* this class.  
076 \*/  
077 @Deprecated  
078 public static void setDebug(final int newDebug) {  
079 debug = newDebug;  
080 }  
081  
082 // --------------------------------------------------------- Class Methods  
083  
084  
085 /\*\*  
086 \* <p>Clone a bean based on the available property getters and setters,  
087 \* even if the bean class itself does not implement Cloneable.</p>  
088 \*  
089 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
090 \*  
091 \* @param bean Bean to be cloned  
092 \* @return the cloned bean  
093 \*  
094 \* @throws IllegalAccessException if the caller does not have  
095 \* access to the property accessor method  
096 \* @throws InstantiationException if a new instance of the bean's  
097 \* class cannot be instantiated  
098 \* @throws InvocationTargetException if the property accessor method  
099 \* throws an exception  
100 \* @throws NoSuchMethodException if an accessor method for this  
101 \* property cannot be found  
102 \* @see BeanUtilsBean#cloneBean  
103 \*/  
104 public static Object cloneBean(final Object bean)  
105 throws IllegalAccessException, InstantiationException,  
106 InvocationTargetException, NoSuchMethodException {  
107  
108 return BeanUtilsBean.getInstance().cloneBean(bean);  
109  
110 }  
111  
112  
113 /\*\*  
114 \* <p>Copy property values from the origin bean to the destination bean  
115 \* for all cases where the property names are the same.</p>  
116 \*  
117 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
118 \*  
119 \* @param dest Destination bean whose properties are modified  
120 \* @param orig Origin bean whose properties are retrieved  
121 \*  
122 \* @throws IllegalAccessException if the caller does not have  
123 \* access to the property accessor method  
124 \* @throws IllegalArgumentException if the <code>dest</code> or  
125 \* <code>orig</code> argument is null or if the <code>dest</code>  
126 \* property type is different from the source type and the relevant  
127 \* converter has not been registered.  
128 \* @throws InvocationTargetException if the property accessor method  
129 \* throws an exception  
130 \* @see BeanUtilsBean#copyProperties  
131 \*/  
132 public static void copyProperties(final Object dest, final Object orig)  
133 throws IllegalAccessException, InvocationTargetException {  
134  
135 BeanUtilsBean.getInstance().copyProperties(dest, orig);  
136 }  
137  
138  
139 /\*\*  
140 \* <p>Copy the specified property value to the specified destination bean,  
141 \* performing any type conversion that is required.</p>  
142 \*  
143 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
144 \*  
145 \* @param bean Bean on which setting is to be performed  
146 \* @param name Property name (can be nested/indexed/mapped/combo)  
147 \* @param value Value to be set  
148 \*  
149 \* @throws IllegalAccessException if the caller does not have  
150 \* access to the property accessor method  
151 \* @throws InvocationTargetException if the property accessor method  
152 \* throws an exception  
153 \* @see BeanUtilsBean#copyProperty  
154 \*/  
155 public static void copyProperty(final Object bean, final String name, final Object value)  
156 throws IllegalAccessException, InvocationTargetException {  
157  
158 BeanUtilsBean.getInstance().copyProperty(bean, name, value);  
159 }  
160  
161  
162 /\*\*  
163 \* <p>Return the entire set of properties for which the specified bean  
164 \* provides a read method.</p>  
165 \*  
166 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
167 \*  
168 \* @param bean Bean whose properties are to be extracted  
169 \* @return Map of property descriptors  
170 \*  
171 \* @throws IllegalAccessException if the caller does not have  
172 \* access to the property accessor method  
173 \* @throws InvocationTargetException if the property accessor method  
174 \* throws an exception  
175 \* @throws NoSuchMethodException if an accessor method for this  
176 \* property cannot be found  
177 \* @see BeanUtilsBean#describe  
178 \*/  
179 public static Map<String, String> describe(final Object bean)  
180 throws IllegalAccessException, InvocationTargetException,  
181 NoSuchMethodException {  
182  
183 return BeanUtilsBean.getInstance().describe(bean);  
184 }  
185  
186  
187 /\*\*  
188 \* <p>Return the value of the specified array property of the specified  
189 \* bean, as a String array.</p>  
190 \*  
191 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
192 \*  
193 \* @param bean Bean whose property is to be extracted  
194 \* @param name Name of the property to be extracted  
195 \* @return The array property value  
196 \*  
197 \* @throws IllegalAccessException if the caller does not have  
198 \* access to the property accessor method  
199 \* @throws InvocationTargetException if the property accessor method  
200 \* throws an exception  
201 \* @throws NoSuchMethodException if an accessor method for this  
202 \* property cannot be found  
203 \* @see BeanUtilsBean#getArrayProperty  
204 \*/  
205 public static String[] getArrayProperty(final Object bean, final String name)  
206 throws IllegalAccessException, InvocationTargetException,  
207 NoSuchMethodException {  
208  
209 return BeanUtilsBean.getInstance().getArrayProperty(bean, name);  
210 }  
211  
212  
213 /\*\*  
214 \* <p>Return the value of the specified indexed property of the specified  
215 \* bean, as a String.</p>  
216 \*  
217 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
218 \*  
219 \* @param bean Bean whose property is to be extracted  
220 \* @param name <code>propertyname[index]</code> of the property value  
221 \* to be extracted  
222 \* @return The indexed property's value, converted to a String  
223 \*  
224 \* @throws IllegalAccessException if the caller does not have  
225 \* access to the property accessor method  
226 \* @throws InvocationTargetException if the property accessor method  
227 \* throws an exception  
228 \* @throws NoSuchMethodException if an accessor method for this  
229 \* property cannot be found  
230 \* @see BeanUtilsBean#getIndexedProperty(Object, String)  
231 \*/  
232 public static String getIndexedProperty(final Object bean, final String name)  
233 throws IllegalAccessException, InvocationTargetException,  
234 NoSuchMethodException {  
235  
236 return BeanUtilsBean.getInstance().getIndexedProperty(bean, name);  
237  
238 }  
239  
240  
241 /\*\*  
242 \* Return the value of the specified indexed property of the specified  
243 \* bean, as a String. The index is specified as a method parameter and  
244 \* must \*not\* be included in the property name expression  
245 \*  
246 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
247 \*  
248 \* @param bean Bean whose property is to be extracted  
249 \* @param name Simple property name of the property value to be extracted  
250 \* @param index Index of the property value to be extracted  
251 \* @return The indexed property's value, converted to a String  
252 \*  
253 \* @throws IllegalAccessException if the caller does not have  
254 \* access to the property accessor method  
255 \* @throws InvocationTargetException if the property accessor method  
256 \* throws an exception  
257 \* @throws NoSuchMethodException if an accessor method for this  
258 \* property cannot be found  
259 \* @see BeanUtilsBean#getIndexedProperty(Object, String, int)  
260 \*/  
261 public static String getIndexedProperty(final Object bean,  
262 final String name, final int index)  
263 throws IllegalAccessException, InvocationTargetException,  
264 NoSuchMethodException {  
265  
266 return BeanUtilsBean.getInstance().getIndexedProperty(bean, name, index);  
267  
268 }  
269  
270  
271 /\*\*  
272 \* </p>Return the value of the specified indexed property of the specified  
273 \* bean, as a String.</p>  
274 \*  
275 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
276 \*  
277 \* @param bean Bean whose property is to be extracted  
278 \* @param name <code>propertyname(index)</code> of the property value  
279 \* to be extracted  
280 \* @return The mapped property's value, converted to a String  
281 \*  
282 \* @throws IllegalAccessException if the caller does not have  
283 \* access to the property accessor method  
284 \* @throws InvocationTargetException if the property accessor method  
285 \* throws an exception  
286 \* @throws NoSuchMethodException if an accessor method for this  
287 \* property cannot be found  
288 \* @see BeanUtilsBean#getMappedProperty(Object, String)  
289 \*/  
290 public static String getMappedProperty(final Object bean, final String name)  
291 throws IllegalAccessException, InvocationTargetException,  
292 NoSuchMethodException {  
293  
294 return BeanUtilsBean.getInstance().getMappedProperty(bean, name);  
295  
296 }  
297  
298  
299 /\*\*  
300 \* </p>Return the value of the specified mapped property of the specified  
301 \* bean, as a String.</p>  
302 \*  
303 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
304 \*  
305 \* @param bean Bean whose property is to be extracted  
306 \* @param name Simple property name of the property value to be extracted  
307 \* @param key Lookup key of the property value to be extracted  
308 \* @return The mapped property's value, converted to a String  
309 \*  
310 \* @throws IllegalAccessException if the caller does not have  
311 \* access to the property accessor method  
312 \* @throws InvocationTargetException if the property accessor method  
313 \* throws an exception  
314 \* @throws NoSuchMethodException if an accessor method for this  
315 \* property cannot be found  
316 \* @see BeanUtilsBean#getMappedProperty(Object, String, String)  
317 \*/  
318 public static String getMappedProperty(final Object bean,  
319 final String name, final String key)  
320 throws IllegalAccessException, InvocationTargetException,  
321 NoSuchMethodException {  
322  
323 return BeanUtilsBean.getInstance().getMappedProperty(bean, name, key);  
324  
325 }  
326  
327  
328 /\*\*  
329 \* <p>Return the value of the (possibly nested) property of the specified  
330 \* name, for the specified bean, as a String.</p>  
331 \*  
332 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
333 \*  
334 \* @param bean Bean whose property is to be extracted  
335 \* @param name Possibly nested name of the property to be extracted  
336 \* @return The nested property's value, converted to a String  
337 \*  
338 \* @throws IllegalAccessException if the caller does not have  
339 \* access to the property accessor method  
340 \* @throws IllegalArgumentException if a nested reference to a  
341 \* property returns null  
342 \* @throws InvocationTargetException if the property accessor method  
343 \* throws an exception  
344 \* @throws NoSuchMethodException if an accessor method for this  
345 \* property cannot be found  
346 \* @see BeanUtilsBean#getNestedProperty  
347 \*/  
348 public static String getNestedProperty(final Object bean, final String name)  
349 throws IllegalAccessException, InvocationTargetException,  
350 NoSuchMethodException {  
351  
352 return BeanUtilsBean.getInstance().getNestedProperty(bean, name);  
353  
354 }  
355  
356  
357 /\*\*  
358 \* <p>Return the value of the specified property of the specified bean,  
359 \* no matter which property reference format is used, as a String.</p>  
360 \*  
361 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
362 \*  
363 \* @param bean Bean whose property is to be extracted  
364 \* @param name Possibly indexed and/or nested name of the property  
365 \* to be extracted  
366 \* @return The property's value, converted to a String  
367 \*  
368 \* @throws IllegalAccessException if the caller does not have  
369 \* access to the property accessor method  
370 \* @throws InvocationTargetException if the property accessor method  
371 \* throws an exception  
372 \* @throws NoSuchMethodException if an accessor method for this  
373 \* property cannot be found  
374 \* @see BeanUtilsBean#getProperty  
375 \*/  
376 public static String getProperty(final Object bean, final String name)  
377 throws IllegalAccessException, InvocationTargetException,  
378 NoSuchMethodException {  
379  
380 return BeanUtilsBean.getInstance().getProperty(bean, name);  
381  
382 }  
383  
384  
385 /\*\*  
386 \* <p>Return the value of the specified simple property of the specified  
387 \* bean, converted to a String.</p>  
388 \*  
389 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
390 \*  
391 \* @param bean Bean whose property is to be extracted  
392 \* @param name Name of the property to be extracted  
393 \* @return The property's value, converted to a String  
394 \*  
395 \* @throws IllegalAccessException if the caller does not have  
396 \* access to the property accessor method  
397 \* @throws InvocationTargetException if the property accessor method  
398 \* throws an exception  
399 \* @throws NoSuchMethodException if an accessor method for this  
400 \* property cannot be found  
401 \* @see BeanUtilsBean#getSimpleProperty  
402 \*/  
403 public static String getSimpleProperty(final Object bean, final String name)  
404 throws IllegalAccessException, InvocationTargetException,  
405 NoSuchMethodException {  
406  
407 return BeanUtilsBean.getInstance().getSimpleProperty(bean, name);  
408  
409 }  
410  
411  
412 /\*\*  
413 \* <p>Populate the JavaBeans properties of the specified bean, based on  
414 \* the specified name/value pairs.</p>  
415 \*  
416 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
417 \*  
418 \* @param bean JavaBean whose properties are being populated  
419 \* @param properties Map keyed by property name, with the  
420 \* corresponding (String or String[]) value(s) to be set  
421 \*  
422 \* @throws IllegalAccessException if the caller does not have  
423 \* access to the property accessor method  
424 \* @throws InvocationTargetException if the property accessor method  
425 \* throws an exception  
426 \* @see BeanUtilsBean#populate  
427 \*/  
428 public static void populate(final Object bean, final Map<String, ? extends Object> properties)  
429 throws IllegalAccessException, InvocationTargetException {  
430  
431 BeanUtilsBean.getInstance().populate(bean, properties);  
432 }  
433  
434  
435 /\*\*  
436 \* <p>Set the specified property value, performing type conversions as  
437 \* required to conform to the type of the destination property.</p>  
438 \*  
439 \* <p>For more details see <code>BeanUtilsBean</code>.</p>  
440 \*  
441 \* @param bean Bean on which setting is to be performed  
442 \* @param name Property name (can be nested/indexed/mapped/combo)  
443 \* @param value Value to be set  
444 \*  
445 \* @throws IllegalAccessException if the caller does not have  
446 \* access to the property accessor method  
447 \* @throws InvocationTargetException if the property accessor method  
448 \* throws an exception  
449 \* @see BeanUtilsBean#setProperty  
450 \*/  
451 public static void setProperty(final Object bean, final String name, final Object value)  
452 throws IllegalAccessException, InvocationTargetException {  
453  
454 BeanUtilsBean.getInstance().setProperty(bean, name, value);  
455 }  
456  
457 /\*\*  
458 \* If we're running on JDK 1.4 or later, initialize the cause for the given throwable.  
459 \*  
460 \* @param throwable The throwable.  
461 \* @param cause The cause of the throwable.  
462 \* @return true if the cause was initialized, otherwise false.  
463 \* @since 1.8.0  
464 \*/  
465 public static boolean initCause(final Throwable throwable, final Throwable cause) {  
466 return BeanUtilsBean.getInstance().initCause(throwable, cause);  
467 }  
468  
469 /\*\*  
470 \* Create a cache.  
471 \* @param <K> the key type of the cache  
472 \* @param <V> the value type of the cache  
473 \* @return a new cache  
474 \* @since 1.8.0  
475 \*/  
476 public static <K, V> Map<K, V> createCache() {  
477 return new WeakFastHashMap<K, V>();  
478 }  
479  
480 /\*\*  
481 \* Return whether a Map is fast  
482 \* @param map The map  
483 \* @return Whether it is fast or not.  
484 \* @since 1.8.0  
485 \*/  
486 public static boolean getCacheFast(final Map<?, ?> map) {  
487 if (map instanceof WeakFastHashMap) {  
488 return ((WeakFastHashMap<?, ?>) map).getFast();  
489 } else {  
490 return false;  
491 }  
492 }  
493  
494 /\*\*  
495 \* Set whether fast on a Map  
496 \* @param map The map  
497 \* @param fast Whether it should be fast or not.  
498 \* @since 1.8.0  
499 \*/  
500 public static void setCacheFast(final Map<?, ?> map, final boolean fast) {  
501 if (map instanceof WeakFastHashMap) {  
502 ((WeakFastHashMap<?, ?>)map).setFast(fast);  
503 }  
504 }  
505}