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  
018package org.apache.commons.beanutils.locale;  
019  
020  
021import java.beans.IndexedPropertyDescriptor;  
022import java.beans.PropertyDescriptor;  
023import java.lang.reflect.InvocationTargetException;  
024import java.util.Locale;  
025  
026import org.apache.commons.beanutils.BeanUtilsBean;  
027import org.apache.commons.beanutils.ContextClassLoaderLocal;  
028import org.apache.commons.beanutils.ConvertUtils;  
029import org.apache.commons.beanutils.ConvertUtilsBean;  
030import org.apache.commons.beanutils.DynaBean;  
031import org.apache.commons.beanutils.DynaClass;  
032import org.apache.commons.beanutils.DynaProperty;  
033import org.apache.commons.beanutils.MappedPropertyDescriptor;  
034import org.apache.commons.beanutils.PropertyUtilsBean;  
035import org.apache.commons.beanutils.expression.Resolver;  
036import org.apache.commons.logging.Log;  
037import org.apache.commons.logging.LogFactory;  
038  
039  
040/\*\*  
041 \* <p>Utility methods for populating JavaBeans properties  
042 \* via reflection in a locale-dependent manner.</p>  
043 \*  
044 \* @since 1.7  
045 \* @version $Id$  
046 \*/  
047  
048public class LocaleBeanUtilsBean extends BeanUtilsBean {  
049  
050 /\*\*  
051 \* Contains <code>LocaleBeanUtilsBean</code> instances indexed by context classloader.  
052 \*/  
053 private static final ContextClassLoaderLocal<LocaleBeanUtilsBean>  
054 LOCALE\_BEANS\_BY\_CLASSLOADER = new ContextClassLoaderLocal<LocaleBeanUtilsBean>() {  
055 // Creates the default instance used when the context classloader is unavailable  
056 @Override  
057 protected LocaleBeanUtilsBean initialValue() {  
058 return new LocaleBeanUtilsBean();  
059 }  
060 };  
061  
062 /\*\*  
063 \* Gets singleton instance  
064 \*  
065 \* @return the singleton instance  
066 \*/  
067 public static LocaleBeanUtilsBean getLocaleBeanUtilsInstance() {  
068 return LOCALE\_BEANS\_BY\_CLASSLOADER.get();  
069 }  
070  
071 /\*\*  
072 \* Sets the instance which provides the functionality for {@link LocaleBeanUtils}.  
073 \* This is a pseudo-singleton - an single instance is provided per (thread) context classloader.  
074 \* This mechanism provides isolation for web apps deployed in the same container.  
075 \*  
076 \* @param newInstance a new singleton instance  
077 \*/  
078 public static void setInstance(final LocaleBeanUtilsBean newInstance) {  
079 LOCALE\_BEANS\_BY\_CLASSLOADER.set(newInstance);  
080 }  
081  
082 /\*\* All logging goes through this logger \*/  
083 private final Log log = LogFactory.getLog(LocaleBeanUtilsBean.class);  
084  
085 // ----------------------------------------------------- Instance Variables  
086  
087 /\*\* Convertor used by this class \*/  
088 private final LocaleConvertUtilsBean localeConvertUtils;  
089  
090 // --------------------------------------------------------- Constructors  
091  
092 /\*\* Construct instance with standard conversion bean \*/  
093 public LocaleBeanUtilsBean() {  
094 this.localeConvertUtils = new LocaleConvertUtilsBean();  
095 }  
096  
097 /\*\*  
098 \* Construct instance that uses given locale conversion  
099 \*  
100 \* @param localeConvertUtils use this <code>localeConvertUtils</code> to perform  
101 \* conversions  
102 \* @param convertUtilsBean use this for standard conversions  
103 \* @param propertyUtilsBean use this for property conversions  
104 \*/  
105 public LocaleBeanUtilsBean(  
106 final LocaleConvertUtilsBean localeConvertUtils,  
107 final ConvertUtilsBean convertUtilsBean,  
108 final PropertyUtilsBean propertyUtilsBean) {  
109 super(convertUtilsBean, propertyUtilsBean);  
110 this.localeConvertUtils = localeConvertUtils;  
111 }  
112  
113 /\*\*  
114 \* Construct instance that uses given locale conversion  
115 \*  
116 \* @param localeConvertUtils use this <code>localeConvertUtils</code> to perform  
117 \* conversions  
118 \*/  
119 public LocaleBeanUtilsBean(final LocaleConvertUtilsBean localeConvertUtils) {  
120 this.localeConvertUtils = localeConvertUtils;  
121 }  
122  
123 // --------------------------------------------------------- Public Methods  
124  
125 /\*\*  
126 \* Gets the bean instance used for conversions  
127 \*  
128 \* @return the locale converter bean instance  
129 \*/  
130 public LocaleConvertUtilsBean getLocaleConvertUtils() {  
131 return localeConvertUtils;  
132 }  
133  
134 /\*\*  
135 \* Gets the default Locale  
136 \* @return the default locale  
137 \*/  
138 public Locale getDefaultLocale() {  
139  
140 return getLocaleConvertUtils().getDefaultLocale();  
141 }  
142  
143  
144 /\*\*  
145 \* Sets the default Locale.  
146 \*  
147 \* @param locale the default locale  
148 \*/  
149 public void setDefaultLocale(final Locale locale) {  
150  
151 getLocaleConvertUtils().setDefaultLocale(locale);  
152 }  
153  
154 /\*\*  
155 \* Is the pattern to be applied localized  
156 \* (Indicate whether the pattern is localized or not)  
157 \*  
158 \* @return <code>true</code> if pattern is localized,  
159 \* otherwise <code>false</code>  
160 \*/  
161 public boolean getApplyLocalized() {  
162  
163 return getLocaleConvertUtils().getApplyLocalized();  
164 }  
165  
166 /\*\*  
167 \* Sets whether the pattern is applied localized  
168 \* (Indicate whether the pattern is localized or not)  
169 \*  
170 \* @param newApplyLocalized <code>true</code> if pattern is localized,  
171 \* otherwise <code>false</code>  
172 \*/  
173 public void setApplyLocalized(final boolean newApplyLocalized) {  
174  
175 getLocaleConvertUtils().setApplyLocalized(newApplyLocalized);  
176 }  
177  
178  
179 // --------------------------------------------------------- Public Methods  
180  
181 /\*\*  
182 \* Return the value of the specified locale-sensitive indexed property  
183 \* of the specified bean, as a String. The zero-relative index of the  
184 \* required value must be included (in square brackets) as a suffix to  
185 \* the property name, or <code>IllegalArgumentException</code> will be  
186 \* thrown.  
187 \*  
188 \* @param bean Bean whose property is to be extracted  
189 \* @param name <code>propertyname[index]</code> of the property value  
190 \* to be extracted  
191 \* @param pattern The conversion pattern  
192 \* @return The indexed property's value, converted to a String  
193 \*  
194 \* @throws IllegalAccessException if the caller does not have  
195 \* access to the property accessor method  
196 \* @throws InvocationTargetException if the property accessor method  
197 \* throws an exception  
198 \* @throws NoSuchMethodException if an accessor method for this  
199 \* propety cannot be found  
200 \*/  
201 public String getIndexedProperty(  
202 final Object bean,  
203 final String name,  
204 final String pattern)  
205 throws  
206 IllegalAccessException,  
207 InvocationTargetException,  
208 NoSuchMethodException {  
209  
210 final Object value = getPropertyUtils().getIndexedProperty(bean, name);  
211 return getLocaleConvertUtils().convert(value, pattern);  
212 }  
213  
214 /\*\*  
215 \* Return the value of the specified locale-sensitive indexed property  
216 \* of the specified bean, as a String using the default conversion pattern of  
217 \* the corresponding {@link LocaleConverter}. The zero-relative index  
218 \* of the required value must be included (in square brackets) as a suffix  
219 \* to the property name, or <code>IllegalArgumentException</code> will be thrown.  
220 \*  
221 \* @param bean Bean whose property is to be extracted  
222 \* @param name <code>propertyname[index]</code> of the property value  
223 \* to be extracted  
224 \* @return The indexed property's value, converted to a String  
225 \*  
226 \* @throws IllegalAccessException if the caller does not have  
227 \* access to the property accessor method  
228 \* @throws InvocationTargetException if the property accessor method  
229 \* throws an exception  
230 \* @throws NoSuchMethodException if an accessor method for this  
231 \* propety cannot be found  
232 \*/  
233 @Override  
234 public String getIndexedProperty(  
235 final Object bean,  
236 final String name)  
237 throws  
238 IllegalAccessException,  
239 InvocationTargetException,  
240 NoSuchMethodException {  
241  
242 return getIndexedProperty(bean, name, null);  
243 }  
244  
245 /\*\*  
246 \* Return the value of the specified locale-sensetive indexed property  
247 \* of the specified bean, as a String using the specified conversion pattern.  
248 \* The index is specified as a method parameter and  
249 \* must \*not\* be included in the property name expression  
250 \*  
251 \* @param bean Bean whose property is to be extracted  
252 \* @param name Simple property name of the property value to be extracted  
253 \* @param index Index of the property value to be extracted  
254 \* @param pattern The conversion pattern  
255 \* @return The indexed property's value, converted to a String  
256 \*  
257 \* @throws IllegalAccessException if the caller does not have  
258 \* access to the property accessor method  
259 \* @throws InvocationTargetException if the property accessor method  
260 \* throws an exception  
261 \* @throws NoSuchMethodException if an accessor method for this  
262 \* propety cannot be found  
263 \*/  
264 public String getIndexedProperty(final Object bean,  
265 final String name, final int index, final String pattern)  
266 throws IllegalAccessException, InvocationTargetException,  
267 NoSuchMethodException {  
268  
269 final Object value = getPropertyUtils().getIndexedProperty(bean, name, index);  
270 return getLocaleConvertUtils().convert(value, pattern);  
271 }  
272  
273 /\*\*  
274 \* Return the value of the specified locale-sensetive indexed property  
275 \* of the specified bean, as a String using the default conversion pattern of  
276 \* the corresponding {@link LocaleConverter}.  
277 \* The index is specified as a method parameter and  
278 \* must \*not\* be included in the property name expression  
279 \*  
280 \* @param bean Bean whose property is to be extracted  
281 \* @param name Simple property name of the property value to be extracted  
282 \* @param index Index of the property value to be extracted  
283 \* @return The indexed property's value, converted to a String  
284 \*  
285 \* @throws IllegalAccessException if the caller does not have  
286 \* access to the property accessor method  
287 \* @throws InvocationTargetException if the property accessor method  
288 \* throws an exception  
289 \* @throws NoSuchMethodException if an accessor method for this  
290 \* propety cannot be found  
291 \*/  
292 @Override  
293 public String getIndexedProperty(final Object bean,  
294 final String name, final int index)  
295 throws IllegalAccessException, InvocationTargetException,  
296 NoSuchMethodException {  
297 return getIndexedProperty(bean, name, index, null);  
298 }  
299  
300 /\*\*  
301 \* Return the value of the specified simple locale-sensitive property  
302 \* of the specified bean, converted to a String using the specified  
303 \* conversion pattern.  
304 \*  
305 \* @param bean Bean whose property is to be extracted  
306 \* @param name Name of the property to be extracted  
307 \* @param pattern The conversion pattern  
308 \* @return The 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 \*/  
317 public String getSimpleProperty(final Object bean, final String name, final String pattern)  
318 throws IllegalAccessException, InvocationTargetException,  
319 NoSuchMethodException {  
320  
321 final Object value = getPropertyUtils().getSimpleProperty(bean, name);  
322 return getLocaleConvertUtils().convert(value, pattern);  
323 }  
324  
325 /\*\*  
326 \* Return the value of the specified simple locale-sensitive property  
327 \* of the specified bean, converted to a String using the default  
328 \* conversion pattern of the corresponding {@link LocaleConverter}.  
329 \*  
330 \* @param bean Bean whose property is to be extracted  
331 \* @param name Name of the property to be extracted  
332 \* @return The property's value, converted to a String  
333 \*  
334 \* @throws IllegalAccessException if the caller does not have  
335 \* access to the property accessor method  
336 \* @throws InvocationTargetException if the property accessor method  
337 \* throws an exception  
338 \* @throws NoSuchMethodException if an accessor method for this  
339 \* property cannot be found  
340 \*/  
341 @Override  
342 public String getSimpleProperty(final Object bean, final String name)  
343 throws IllegalAccessException, InvocationTargetException,  
344 NoSuchMethodException {  
345  
346 return getSimpleProperty(bean, name, null);  
347 }  
348  
349 /\*\*  
350 \* Return the value of the specified mapped locale-sensitive property  
351 \* of the specified bean, as a String using the specified conversion pattern.  
352 \* The key is specified as a method parameter and must \*not\* be included in  
353 \* the property name expression.  
354 \*  
355 \* @param bean Bean whose property is to be extracted  
356 \* @param name Simple property name of the property value to be extracted  
357 \* @param key Lookup key of the property value to be extracted  
358 \* @param pattern The conversion pattern  
359 \* @return The mapped property's value, converted to a String  
360 \*  
361 \* @throws IllegalAccessException if the caller does not have  
362 \* access to the property accessor method  
363 \* @throws InvocationTargetException if the property accessor method  
364 \* throws an exception  
365 \* @throws NoSuchMethodException if an accessor method for this  
366 \* property cannot be found  
367 \*/  
368 public String getMappedProperty(  
369 final Object bean,  
370 final String name,  
371 final String key,  
372 final String pattern)  
373 throws  
374 IllegalAccessException,  
375 InvocationTargetException,  
376 NoSuchMethodException {  
377  
378 final Object value = getPropertyUtils().getMappedProperty(bean, name, key);  
379 return getLocaleConvertUtils().convert(value, pattern);  
380 }  
381  
382 /\*\*  
383 \* Return the value of the specified mapped locale-sensitive property  
384 \* of the specified bean, as a String  
385 \* The key is specified as a method parameter and must \*not\* be included  
386 \* in the property name expression  
387 \*  
388 \* @param bean Bean whose property is to be extracted  
389 \* @param name Simple property name of the property value to be extracted  
390 \* @param key Lookup key of the property value to be extracted  
391 \* @return The mapped property's value, converted to a String  
392 \*  
393 \* @throws IllegalAccessException if the caller does not have  
394 \* access to the property accessor method  
395 \* @throws InvocationTargetException if the property accessor method  
396 \* throws an exception  
397 \* @throws NoSuchMethodException if an accessor method for this  
398 \* property cannot be found  
399 \*/  
400 @Override  
401 public String getMappedProperty(final Object bean,  
402 final String name, final String key)  
403 throws IllegalAccessException, InvocationTargetException,  
404 NoSuchMethodException {  
405  
406 return getMappedProperty(bean, name, key, null);  
407 }  
408  
409  
410 /\*\*  
411 \* Return the value of the specified locale-sensitive mapped property  
412 \* of the specified bean, as a String using the specified pattern.  
413 \* The String-valued key of the required value  
414 \* must be included (in parentheses) as a suffix to  
415 \* the property name, or <code>IllegalArgumentException</code> will be  
416 \* thrown.  
417 \*  
418 \* @param bean Bean whose property is to be extracted  
419 \* @param name <code>propertyname(index)</code> of the property value  
420 \* to be extracted  
421 \* @param pattern The conversion pattern  
422 \* @return The mapped property's value, converted to a String  
423 \*  
424 \* @throws IllegalAccessException if the caller does not have  
425 \* access to the property accessor method  
426 \* @throws InvocationTargetException if the property accessor method  
427 \* throws an exception  
428 \* @throws NoSuchMethodException if an accessor method for this  
429 \* property cannot be found  
430 \*/  
431 public String getMappedPropertyLocale(  
432 final Object bean,  
433 final String name,  
434 final String pattern)  
435 throws  
436 IllegalAccessException,  
437 InvocationTargetException,  
438 NoSuchMethodException {  
439  
440 final Object value = getPropertyUtils().getMappedProperty(bean, name);  
441 return getLocaleConvertUtils().convert(value, pattern);  
442 }  
443  
444  
445 /\*\*  
446 \* Return the value of the specified locale-sensitive mapped property  
447 \* of the specified bean, as a String using the default  
448 \* conversion pattern of the corresponding {@link LocaleConverter}.  
449 \* The String-valued key of the required value  
450 \* must be included (in parentheses) as a suffix to  
451 \* the property name, or <code>IllegalArgumentException</code> will be  
452 \* thrown.  
453 \*  
454 \* @param bean Bean whose property is to be extracted  
455 \* @param name <code>propertyname(index)</code> of the property value  
456 \* to be extracted  
457 \* @return The mapped property's value, converted to a String  
458 \*  
459 \* @throws IllegalAccessException if the caller does not have  
460 \* access to the property accessor method  
461 \* @throws InvocationTargetException if the property accessor method  
462 \* throws an exception  
463 \* @throws NoSuchMethodException if an accessor method for this  
464 \* property cannot be found  
465 \*/  
466 @Override  
467 public String getMappedProperty(final Object bean, final String name)  
468 throws  
469 IllegalAccessException,  
470 InvocationTargetException,  
471 NoSuchMethodException {  
472  
473 return getMappedPropertyLocale(bean, name, null);  
474 }  
475  
476 /\*\*  
477 \* Return the value of the (possibly nested) locale-sensitive property  
478 \* of the specified name, for the specified bean,  
479 \* as a String using the specified pattern.  
480 \*  
481 \* @param bean Bean whose property is to be extracted  
482 \* @param name Possibly nested name of the property to be extracted  
483 \* @param pattern The conversion pattern  
484 \* @return The nested property's value, converted to a String  
485 \*  
486 \* @throws IllegalAccessException if the caller does not have  
487 \* access to the property accessor method  
488 \* @throws IllegalArgumentException if a nested reference to a  
489 \* property returns null  
490 \* @throws InvocationTargetException if the property accessor method  
491 \* throws an exception  
492 \* @throws NoSuchMethodException if an accessor method for this  
493 \* property cannot be found  
494 \*/  
495 public String getNestedProperty(  
496 final Object bean,  
497 final String name,  
498 final String pattern)  
499 throws  
500 IllegalAccessException,  
501 InvocationTargetException,  
502 NoSuchMethodException {  
503  
504 final Object value = getPropertyUtils().getNestedProperty(bean, name);  
505 return getLocaleConvertUtils().convert(value, pattern);  
506 }  
507  
508 /\*\*  
509 \* Return the value of the (possibly nested) locale-sensitive property  
510 \* of the specified name, for the specified bean, as a String using the default  
511 \* conversion pattern of the corresponding {@link LocaleConverter}.  
512 \*  
513 \* @param bean Bean whose property is to be extracted  
514 \* @param name Possibly nested name of the property to be extracted  
515 \* @return The nested property's value, converted to a String  
516 \*  
517 \* @throws IllegalAccessException if the caller does not have  
518 \* access to the property accessor method  
519 \* @throws IllegalArgumentException if a nested reference to a  
520 \* property returns null  
521 \* @throws InvocationTargetException if the property accessor method  
522 \* throws an exception  
523 \* @throws NoSuchMethodException if an accessor method for this  
524 \* property cannot be found  
525 \*/  
526 @Override  
527 public String getNestedProperty(final Object bean, final String name)  
528 throws  
529 IllegalAccessException,  
530 InvocationTargetException,  
531 NoSuchMethodException {  
532  
533 return getNestedProperty(bean, name, null);  
534 }  
535  
536 /\*\*  
537 \* Return the value of the specified locale-sensitive property  
538 \* of the specified bean, no matter which property reference  
539 \* format is used, as a String using the specified conversion pattern.  
540 \*  
541 \* @param bean Bean whose property is to be extracted  
542 \* @param name Possibly indexed and/or nested name of the property  
543 \* to be extracted  
544 \* @param pattern The conversion pattern  
545 \* @return The nested property's value, converted to a String  
546 \*  
547 \* @throws IllegalAccessException if the caller does not have  
548 \* access to the property accessor method  
549 \* @throws InvocationTargetException if the property accessor method  
550 \* throws an exception  
551 \* @throws NoSuchMethodException if an accessor method for this  
552 \* property cannot be found  
553 \*/  
554 public String getProperty(final Object bean, final String name, final String pattern)  
555 throws  
556 IllegalAccessException,  
557 InvocationTargetException,  
558 NoSuchMethodException {  
559  
560 return getNestedProperty(bean, name, pattern);  
561 }  
562  
563 /\*\*  
564 \* Return the value of the specified locale-sensitive property  
565 \* of the specified bean, no matter which property reference  
566 \* format is used, as a String using the default  
567 \* conversion pattern of the corresponding {@link LocaleConverter}.  
568 \*  
569 \* @param bean Bean whose property is to be extracted  
570 \* @param name Possibly indexed and/or nested name of the property  
571 \* to be extracted  
572 \* @return The property's value, converted to a String  
573 \*  
574 \* @throws IllegalAccessException if the caller does not have  
575 \* access to the property accessor method  
576 \* @throws InvocationTargetException if the property accessor method  
577 \* throws an exception  
578 \* @throws NoSuchMethodException if an accessor method for this  
579 \* property cannot be found  
580 \*/  
581 @Override  
582 public String getProperty(final Object bean, final String name)  
583 throws  
584 IllegalAccessException,  
585 InvocationTargetException,  
586 NoSuchMethodException {  
587  
588 return getNestedProperty(bean, name);  
589 }  
590  
591 /\*\*  
592 \* Set the specified locale-sensitive property value, performing type  
593 \* conversions as required to conform to the type of the destination property  
594 \* using the default conversion pattern of the corresponding {@link LocaleConverter}.  
595 \*  
596 \* @param bean Bean on which setting is to be performed  
597 \* @param name Property name (can be nested/indexed/mapped/combo)  
598 \* @param value Value to be set  
599 \*  
600 \* @throws IllegalAccessException if the caller does not have  
601 \* access to the property accessor method  
602 \* @throws InvocationTargetException if the property accessor method  
603 \* throws an exception  
604 \*/  
605 @Override  
606 public void setProperty(final Object bean, final String name, final Object value)  
607 throws  
608 IllegalAccessException,  
609 InvocationTargetException {  
610  
611 setProperty(bean, name, value, null);  
612 }  
613  
614 /\*\*  
615 \* Set the specified locale-sensitive property value, performing type  
616 \* conversions as required to conform to the type of the destination  
617 \* property using the specified conversion pattern.  
618 \*  
619 \* @param bean Bean on which setting is to be performed  
620 \* @param name Property name (can be nested/indexed/mapped/combo)  
621 \* @param value Value to be set  
622 \* @param pattern The conversion pattern  
623 \*  
624 \* @throws IllegalAccessException if the caller does not have  
625 \* access to the property accessor method  
626 \* @throws InvocationTargetException if the property accessor method  
627 \* throws an exception  
628 \*/  
629 public void setProperty(  
630 final Object bean,  
631 String name,  
632 final Object value,  
633 final String pattern)  
634 throws  
635 IllegalAccessException,  
636 InvocationTargetException {  
637  
638 // Trace logging (if enabled)  
639 if (log.isTraceEnabled()) {  
640 final StringBuilder sb = new StringBuilder(" setProperty(");  
641 sb.append(bean);  
642 sb.append(", ");  
643 sb.append(name);  
644 sb.append(", ");  
645 if (value == null) {  
646 sb.append("<NULL>");  
647 }  
648 else if (value instanceof String) {  
649 sb.append((String) value);  
650 }  
651 else if (value instanceof String[]) {  
652 final String[] values = (String[]) value;  
653 sb.append('[');  
654 for (int i = 0; i < values.length; i++) {  
655 if (i > 0) {  
656 sb.append(',');  
657 }  
658 sb.append(values[i]);  
659 }  
660 sb.append(']');  
661 }  
662 else {  
663 sb.append(value.toString());  
664 }  
665 sb.append(')');  
666 log.trace(sb.toString());  
667 }  
668  
669 // Resolve any nested expression to get the actual target bean  
670 Object target = bean;  
671 final Resolver resolver = getPropertyUtils().getResolver();  
672 while (resolver.hasNested(name)) {  
673 try {  
674 target = getPropertyUtils().getProperty(target, resolver.next(name));  
675 name = resolver.remove(name);  
676 } catch (final NoSuchMethodException e) {  
677 return; // Skip this property setter  
678 }  
679 }  
680 if (log.isTraceEnabled()) {  
681 log.trace(" Target bean = " + target);  
682 log.trace(" Target name = " + name);  
683 }  
684  
685 // Declare local variables we will require  
686 final String propName = resolver.getProperty(name); // Simple name of target property  
687 final int index = resolver.getIndex(name); // Indexed subscript value (if any)  
688 final String key = resolver.getKey(name); // Mapped key value (if any)  
689  
690 final Class<?> type = definePropertyType(target, name, propName);  
691 if (type != null) {  
692 final Object newValue = convert(type, index, value, pattern);  
693 invokeSetter(target, propName, key, index, newValue);  
694 }  
695 }  
696  
697 /\*\*  
698 \* Calculate the property type.  
699 \*  
700 \* @param target The bean  
701 \* @param name The property name  
702 \* @param propName The Simple name of target property  
703 \* @return The property's type  
704 \*  
705 \* @throws IllegalAccessException if the caller does not have  
706 \* access to the property accessor method  
707 \* @throws InvocationTargetException if the property accessor method  
708 \* throws an exception  
709 \*/  
710 protected Class<?> definePropertyType(final Object target, final String name, final String propName)  
711 throws IllegalAccessException, InvocationTargetException {  
712  
713 Class<?> type = null; // Java type of target property  
714  
715 if (target instanceof DynaBean) {  
716 final DynaClass dynaClass = ((DynaBean) target).getDynaClass();  
717 final DynaProperty dynaProperty = dynaClass.getDynaProperty(propName);  
718 if (dynaProperty == null) {  
719 return null; // Skip this property setter  
720 }  
721 type = dynaProperty.getType();  
722 }  
723 else {  
724 PropertyDescriptor descriptor = null;  
725 try {  
726 descriptor =  
727 getPropertyUtils().getPropertyDescriptor(target, name);  
728 if (descriptor == null) {  
729 return null; // Skip this property setter  
730 }  
731 }  
732 catch (final NoSuchMethodException e) {  
733 return null; // Skip this property setter  
734 }  
735 if (descriptor instanceof MappedPropertyDescriptor) {  
736 type = ((MappedPropertyDescriptor) descriptor).  
737 getMappedPropertyType();  
738 }  
739 else if (descriptor instanceof IndexedPropertyDescriptor) {  
740 type = ((IndexedPropertyDescriptor) descriptor).  
741 getIndexedPropertyType();  
742 }  
743 else {  
744 type = descriptor.getPropertyType();  
745 }  
746 }  
747 return type;  
748 }  
749  
750 /\*\*  
751 \* Convert the specified value to the required type using the  
752 \* specified conversion pattern.  
753 \*  
754 \* @param type The Java type of target property  
755 \* @param index The indexed subscript value (if any)  
756 \* @param value The value to be converted  
757 \* @param pattern The conversion pattern  
758 \* @return The converted value  
759 \*/  
760 protected Object convert(final Class<?> type, final int index, final Object value, final String pattern) {  
761  
762 if (log.isTraceEnabled()) {  
763 log.trace("Converting value '" + value + "' to type:" + type);  
764 }  
765  
766 Object newValue = null;  
767  
768 if (type.isArray() && (index < 0)) { // Scalar value into array  
769 if (value instanceof String) {  
770 final String[] values = new String[1];  
771 values[0] = (String) value;  
772 newValue = getLocaleConvertUtils().convert(values, type, pattern);  
773 }  
774 else if (value instanceof String[]) {  
775 newValue = getLocaleConvertUtils().convert((String[]) value, type, pattern);  
776 }  
777 else {  
778 newValue = value;  
779 }  
780 }  
781 else if (type.isArray()) { // Indexed value into array  
782 if (value instanceof String) {  
783 newValue = getLocaleConvertUtils().convert((String) value,  
784 type.getComponentType(), pattern);  
785 }  
786 else if (value instanceof String[]) {  
787 newValue = getLocaleConvertUtils().convert(((String[]) value)[0],  
788 type.getComponentType(), pattern);  
789 }  
790 else {  
791 newValue = value;  
792 }  
793 }  
794 else { // Value into scalar  
795 if (value instanceof String) {  
796 newValue = getLocaleConvertUtils().convert((String) value, type, pattern);  
797 }  
798 else if (value instanceof String[]) {  
799 newValue = getLocaleConvertUtils().convert(((String[]) value)[0],  
800 type, pattern);  
801 }  
802 else {  
803 newValue = value;  
804 }  
805 }  
806 return newValue;  
807 }  
808  
809 /\*\*  
810 \* Convert the specified value to the required type.  
811 \*  
812 \* @param type The Java type of target property  
813 \* @param index The indexed subscript value (if any)  
814 \* @param value The value to be converted  
815 \* @return The converted value  
816 \*/  
817 protected Object convert(final Class<?> type, final int index, final Object value) {  
818  
819 Object newValue = null;  
820  
821 if (type.isArray() && (index < 0)) { // Scalar value into array  
822 if (value instanceof String) {  
823 final String[] values = new String[1];  
824 values[0] = (String) value;  
825 newValue = ConvertUtils.convert(values, type);  
826 }  
827 else if (value instanceof String[]) {  
828 newValue = ConvertUtils.convert((String[]) value, type);  
829 }  
830 else {  
831 newValue = value;  
832 }  
833 }  
834 else if (type.isArray()) { // Indexed value into array  
835 if (value instanceof String) {  
836 newValue = ConvertUtils.convert((String) value,  
837 type.getComponentType());  
838 }  
839 else if (value instanceof String[]) {  
840 newValue = ConvertUtils.convert(((String[]) value)[0],  
841 type.getComponentType());  
842 }  
843 else {  
844 newValue = value;  
845 }  
846 }  
847 else { // Value into scalar  
848 if (value instanceof String) {  
849 newValue = ConvertUtils.convert((String) value, type);  
850 }  
851 else if (value instanceof String[]) {  
852 newValue = ConvertUtils.convert(((String[]) value)[0],  
853 type);  
854 }  
855 else {  
856 newValue = value;  
857 }  
858 }  
859 return newValue;  
860 }  
861  
862 /\*\*  
863 \* Invoke the setter method.  
864 \*  
865 \* @param target The bean  
866 \* @param propName The Simple name of target property  
867 \* @param key The Mapped key value (if any)  
868 \* @param index The indexed subscript value (if any)  
869 \* @param newValue The value to be set  
870 \*  
871 \* @throws IllegalAccessException if the caller does not have  
872 \* access to the property accessor method  
873 \* @throws InvocationTargetException if the property accessor method  
874 \* throws an exception  
875 \*/  
876 protected void invokeSetter(final Object target, final String propName, final String key, final int index, final Object newValue)  
877 throws IllegalAccessException, InvocationTargetException {  
878  
879 try {  
880 if (index >= 0) {  
881 getPropertyUtils().setIndexedProperty(target, propName,  
882 index, newValue);  
883 }  
884 else if (key != null) {  
885 getPropertyUtils().setMappedProperty(target, propName,  
886 key, newValue);  
887 }  
888 else {  
889 getPropertyUtils().setProperty(target, propName, newValue);  
890 }  
891 }  
892 catch (final NoSuchMethodException e) {  
893 throw new InvocationTargetException  
894 (e, "Cannot set " + propName);  
895 }  
896 }  
897  
898 /\*\*  
899 \* Resolve any nested expression to get the actual target property.  
900 \*  
901 \* @param bean The bean  
902 \* @param name The property name  
903 \* @return The property's descriptor  
904 \*  
905 \* @throws IllegalAccessException if the caller does not have  
906 \* access to the property accessor method  
907 \* @throws InvocationTargetException if the property accessor method  
908 \* throws an exception  
909 \* @deprecated Property name expressions are now processed by  
910 \* the configured {@link Resolver} implementation and this method  
911 \* is no longer used by BeanUtils.  
912 \*/  
913 @Deprecated  
914 protected Descriptor calculate(final Object bean, String name)  
915 throws IllegalAccessException, InvocationTargetException {  
916  
917 // Resolve any nested expression to get the actual target bean  
918 Object target = bean;  
919 final Resolver resolver = getPropertyUtils().getResolver();  
920 while (resolver.hasNested(name)) {  
921 try {  
922 target = getPropertyUtils().getProperty(target, resolver.next(name));  
923 name = resolver.remove(name);  
924 } catch (final NoSuchMethodException e) {  
925 return null; // Skip this property setter  
926 }  
927 }  
928 if (log.isTraceEnabled()) {  
929 log.trace(" Target bean = " + target);  
930 log.trace(" Target name = " + name);  
931 }  
932  
933 // Declare local variables we will require  
934 final String propName = resolver.getProperty(name); // Simple name of target property  
935 final int index = resolver.getIndex(name); // Indexed subscript value (if any)  
936 final String key = resolver.getKey(name); // Mapped key value (if any)  
937  
938 return new Descriptor(target, name, propName, key, index);  
939 }  
940  
941 /\*\*  
942 \* @deprecated Property name expressions are now processed by  
943 \* the configured {@link Resolver} implementation and this class  
944 \* is no longer used by BeanUtils.  
945 \*/  
946 @Deprecated  
947 protected class Descriptor {  
948  
949 private int index = -1; // Indexed subscript value (if any)  
950 private String name;  
951 private String propName; // Simple name of target property  
952 private String key; // Mapped key value (if any)  
953 private Object target;  
954  
955 /\*\*  
956 \* Construct a descriptor instance for the target bean and property.  
957 \*  
958 \* @param target The target bean  
959 \* @param name The property name (includes indexed/mapped expr)  
960 \* @param propName The property name  
961 \* @param key The mapped property key (if any)  
962 \* @param index The indexed property index (if any)  
963 \*/  
964 public Descriptor(final Object target, final String name, final String propName, final String key, final int index) {  
965  
966 setTarget(target);  
967 setName(name);  
968 setPropName(propName);  
969 setKey(key);  
970 setIndex(index);  
971 }  
972  
973 /\*\*  
974 \* Return the target bean.  
975 \*  
976 \* @return The descriptors target bean  
977 \*/  
978 public Object getTarget() {  
979 return target;  
980 }  
981  
982 /\*\*  
983 \* Set the target bean.  
984 \*  
985 \* @param target The target bean  
986 \*/  
987 public void setTarget(final Object target) {  
988 this.target = target;  
989 }  
990  
991 /\*\*  
992 \* Return the mapped property key.  
993 \*  
994 \* @return the mapped property key (if any)  
995 \*/  
996 public String getKey() {  
997 return key;  
998 }  
999  
1000 /\*\*  
1001 \* Set the mapped property key.  
1002 \*  
1003 \* @param key The mapped property key (if any)  
1004 \*/  
1005 public void setKey(final String key) {  
1006 this.key = key;  
1007 }  
1008  
1009 /\*\*  
1010 \* Return indexed property index.  
1011 \*  
1012 \* @return indexed property index (if any)  
1013 \*/  
1014 public int getIndex() {  
1015 return index;  
1016 }  
1017  
1018 /\*\*  
1019 \* Set the indexed property index.  
1020 \*  
1021 \* @param index The indexed property index (if any)  
1022 \*/  
1023 public void setIndex(final int index) {  
1024 this.index = index;  
1025 }  
1026  
1027 /\*\*  
1028 \* Return property name (includes indexed/mapped expr).  
1029 \*  
1030 \* @return The property name (includes indexed/mapped expr)  
1031 \*/  
1032 public String getName() {  
1033 return name;  
1034 }  
1035  
1036 /\*\*  
1037 \* Set the property name (includes indexed/mapped expr).  
1038 \*  
1039 \* @param name The property name (includes indexed/mapped expr)  
1040 \*/  
1041 public void setName(final String name) {  
1042 this.name = name;  
1043 }  
1044  
1045 /\*\*  
1046 \* Return the property name.  
1047 \*  
1048 \* @return The property name  
1049 \*/  
1050 public String getPropName() {  
1051 return propName;  
1052 }  
1053  
1054 /\*\*  
1055 \* Set the property name.  
1056 \*  
1057 \* @param propName The property name  
1058 \*/  
1059 public void setPropName(final String propName) {  
1060 this.propName = propName;  
1061 }  
1062 }  
1063}  
1064  
1065