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 \*/  
017package org.apache.commons.beanutils;  
018  
019import java.lang.reflect.Array;  
020import java.util.ArrayList;  
021import java.util.Collection;  
022import java.util.Map;  
023  
024/\*\*  
025 \* <h2><i>Lazy</i> DynaBean List.</h2>  
026 \*  
027 \* <p>There are two main purposes for this class:</p>  
028 \* <ul>  
029 \* <li>To provide <i>Lazy List</i> behaviour - automatically  
030 \* <i>growing</i> and <i>populating</i> the <code>List</code>  
031 \* with either <code>DynaBean</code>, <code>java.util.Map</code>  
032 \* or POJO Beans.</li>  
033 \* <li>To provide a straight forward way of putting a Collection  
034 \* or Array into the lazy list <i>and</i> a straight forward  
035 \* way to get it out again at the end.</li>  
036 \* </ul>  
037 \*  
038 \* <p>All elements added to the List are stored as <code>DynaBean</code>'s:</p>  
039 \* <ul>  
040 \* <li><code>java.util.Map</code> elements are "wrapped" in a <code>LazyDynaMap</code>.</i>  
041 \* <li>POJO Bean elements are "wrapped" in a <code>WrapDynaBean.</code></i>  
042 \* <li><code>DynaBean</code>'s are stored un-changed.</i>  
043 \* </ul>  
044 \*  
045 \* <h4><code>toArray()</code></h4>  
046 \* <p>The <code>toArray()</code> method returns an array of the  
047 \* elements of the appropriate type. If the <code>LazyDynaList</code>  
048 \* is populated with <code>java.util.Map</code> objects a  
049 \* <code>Map[]</code> array is returned.  
050 \* If the list is populated with POJO Beans an appropriate  
051 \* array of the POJO Beans is returned. Otherwise a <code>DynaBean[]</code>  
052 \* array is returned.  
053 \* </p>  
054 \*  
055 \* <h4><code>toDynaBeanArray()</code></h4>  
056 \* <p>The <code>toDynaBeanArray()</code> method returns a  
057 \* <code>DynaBean[]</code> array of the elements in the List.  
058 \* </p>  
059 \*  
060 \* <p><strong>N.B.</strong>All the elements in the List must be the  
061 \* same type. If the <code>DynaClass</code> or <code>Class</code>  
062 \* of the <code>LazyDynaList</code>'s elements is  
063 \* not specified, then it will be automatically set to the type  
064 \* of the first element populated.  
065 \* </p>  
066 \*  
067 \* <h3>Example 1</h3>  
068 \* <p>If you have an array of <code>java.util.Map[]</code> - you can put that into  
069 \* a <code>LazyDynaList</code>.</p>  
070 \*  
071 \* <pre><code>  
072 \* TreeMap[] myArray = .... // your Map[]  
073 \* List lazyList = new LazyDynaList(myArray);  
074 \* </code></pre>  
075 \*  
076 \* <p>New elements of the appropriate Map type are  
077 \* automatically populated:</p>  
078 \*  
079 \* <pre><code>  
080 \* // get(index) automatically grows the list  
081 \* DynaBean newElement = (DynaBean)lazyList.get(lazyList.size());  
082 \* newElement.put("someProperty", "someValue");  
083 \* </code></pre>  
084 \*  
085 \* <p>Once you've finished you can get back an Array of the  
086 \* elements of the appropriate type:</p>  
087 \*  
088 \* <pre><code>  
089 \* // Retrieve the array from the list  
090 \* TreeMap[] myArray = (TreeMap[])lazyList.toArray());  
091 \* </code></pre>  
092 \*  
093 \*  
094 \* <h3>Example 2</h3>  
095 \* <p>Alternatively you can create an <i>empty</i> List and  
096 \* specify the Class for List's elements. The LazyDynaList  
097 \* uses the Class to automatically populate elements:</p>  
098 \*  
099 \* <pre><code>  
100 \* // e.g. For Maps  
101 \* List lazyList = new LazyDynaList(TreeMap.class);  
102 \*  
103 \* // e.g. For POJO Beans  
104 \* List lazyList = new LazyDynaList(MyPojo.class);  
105 \*  
106 \* // e.g. For DynaBeans  
107 \* List lazyList = new LazyDynaList(MyDynaBean.class);  
108 \* </code></pre>  
109 \*  
110 \* <h3>Example 3</h3>  
111 \* <p>Alternatively you can create an <i>empty</i> List and specify the  
112 \* DynaClass for List's elements. The LazyDynaList uses  
113 \* the DynaClass to automatically populate elements:</p>  
114 \*  
115 \* <pre><code>  
116 \* // e.g. For Maps  
117 \* DynaClass dynaClass = new LazyDynaMap(new HashMap());  
118 \* List lazyList = new LazyDynaList(dynaClass);  
119 \*  
120 \* // e.g. For POJO Beans  
121 \* DynaClass dynaClass = (new WrapDynaBean(myPojo)).getDynaClass();  
122 \* List lazyList = new LazyDynaList(dynaClass);  
123 \*  
124 \* // e.g. For DynaBeans  
125 \* DynaClass dynaClass = new BasicDynaClass(properties);  
126 \* List lazyList = new LazyDynaList(dynaClass);  
127 \* </code></pre>  
128 \*  
129 \* <p><strong>N.B.</strong> You may wonder why control the type  
130 \* using a <code>DynaClass</code> rather than the <code>Class</code>  
131 \* as in the previous example - the reason is that some <code>DynaBean</code>  
132 \* implementations don't have a <i>default</i> empty constructor and  
133 \* therefore need to be instantiated using the <code>DynaClass.newInstance()</code>  
134 \* method.</p>  
135 \*  
136 \* <h3>Example 4</h3>  
137 \* <p>A slight variation - set the element type using either  
138 \* the <code>setElementType(Class)</code> method or the  
139 \* <code>setElementDynaClass(DynaClass)</code> method - then populate  
140 \* with the normal <code>java.util.List</code> methods(i.e.  
141 \* <code>add()</code>, <code>addAll()</code> or <code>set()</code>).</p>  
142 \*  
143 \* <pre><code>  
144 \* // Create a new LazyDynaList (100 element capacity)  
145 \* LazyDynaList lazyList = new LazyDynaList(100);  
146 \*  
147 \* // Either Set the element type...  
148 \* lazyList.setElementType(TreeMap.class);  
149 \*  
150 \* // ...or the element DynaClass...  
151 \* lazyList.setElementDynaClass(new MyCustomDynaClass());  
152 \*  
153 \* // Populate from a collection  
154 \* lazyList.addAll(myCollection);  
155 \*  
156 \* </code></pre>  
157 \*  
158 \* @version $Id$  
159 \* @since 1.8.0  
160 \*/  
161public class LazyDynaList extends ArrayList<Object> {  
162  
163 /\*\*  
164 \* The DynaClass of the List's elements.  
165 \*/  
166 private DynaClass elementDynaClass;  
167  
168 /\*\*  
169 \* The WrapDynaClass if the List's contains  
170 \* POJO Bean elements.  
171 \*  
172 \* N.B. WrapDynaClass isn't serlializable, which  
173 \* is why its stored separately in a  
174 \* transient instance variable.  
175 \*/  
176 private transient WrapDynaClass wrapDynaClass;  
177  
178 /\*\*  
179 \* The type of the List's elements.  
180 \*/  
181 private Class<?> elementType;  
182  
183 /\*\*  
184 \* The DynaBean type of the List's elements.  
185 \*/  
186 private Class<?> elementDynaBeanType;  
187  
188  
189 // ------------------- Constructors ------------------------------  
190  
191 /\*\*  
192 \* Default Constructor.  
193 \*/  
194 public LazyDynaList() {  
195 super();  
196 }  
197  
198 /\*\*  
199 \* Construct a LazyDynaList with the  
200 \* specified capacity.  
201 \*  
202 \* @param capacity The initial capacity of the list.  
203 \*/  
204 public LazyDynaList(final int capacity) {  
205 super(capacity);  
206  
207 }  
208  
209 /\*\*  
210 \* Construct a LazyDynaList with a  
211 \* specified DynaClass for its elements.  
212 \*  
213 \* @param elementDynaClass The DynaClass of the List's elements.  
214 \*/  
215 public LazyDynaList(final DynaClass elementDynaClass) {  
216 super();  
217 setElementDynaClass(elementDynaClass);  
218 }  
219  
220 /\*\*  
221 \* Construct a LazyDynaList with a  
222 \* specified type for its elements.  
223 \*  
224 \* @param elementType The Type of the List's elements.  
225 \*/  
226 public LazyDynaList(final Class<?> elementType) {  
227 super();  
228 setElementType(elementType);  
229 }  
230  
231 /\*\*  
232 \* Construct a LazyDynaList populated with the  
233 \* elements of a Collection.  
234 \*  
235 \* @param collection The Collection to populate the List from.  
236 \*/  
237 public LazyDynaList(final Collection<?> collection) {  
238 super(collection.size());  
239 addAll(collection);  
240 }  
241  
242 /\*\*  
243 \* Construct a LazyDynaList populated with the  
244 \* elements of an Array.  
245 \*  
246 \* @param array The Array to populate the List from.  
247 \*/  
248 public LazyDynaList(final Object[] array) {  
249 super(array.length);  
250 for (Object element : array) {  
251 add(element);  
252 }  
253 }  
254  
255  
256 // ------------------- java.util.List Methods --------------------  
257  
258 /\*\*  
259 \* <p>Insert an element at the specified index position.</p>  
260 \*  
261 \* <p>If the index position is greater than the current  
262 \* size of the List, then the List is automatically  
263 \* <i>grown</i> to the appropriate size.</p>  
264 \*  
265 \* @param index The index position to insert the new element.  
266 \* @param element The new element to add.  
267 \*/  
268 @Override  
269 public void add(final int index, final Object element) {  
270  
271 final DynaBean dynaBean = transform(element);  
272  
273 growList(index);  
274  
275 super.add(index, dynaBean);  
276  
277 }  
278  
279 /\*\*  
280 \* <p>Add an element to the List.</p>  
281 \*  
282 \* @param element The new element to add.  
283 \* @return true.  
284 \*/  
285 @Override  
286 public boolean add(final Object element) {  
287  
288 final DynaBean dynaBean = transform(element);  
289  
290 return super.add(dynaBean);  
291  
292 }  
293  
294 /\*\*  
295 \* <p>Add all the elements from a Collection to the list.  
296 \*  
297 \* @param collection The Collection of new elements.  
298 \* @return true if elements were added.  
299 \*/  
300 @Override  
301 public boolean addAll(final Collection<?> collection) {  
302  
303 if (collection == null || collection.size() == 0) {  
304 return false;  
305 }  
306  
307 ensureCapacity(size() + collection.size());  
308  
309 for (final Object e : collection) {  
310 add(e);  
311 }  
312  
313 return true;  
314  
315 }  
316  
317 /\*\*  
318 \* <p>Insert all the elements from a Collection into the  
319 \* list at a specified position.  
320 \*  
321 \* <p>If the index position is greater than the current  
322 \* size of the List, then the List is automatically  
323 \* <i>grown</i> to the appropriate size.</p>  
324 \*  
325 \* @param collection The Collection of new elements.  
326 \* @param index The index position to insert the new elements at.  
327 \* @return true if elements were added.  
328 \*/  
329 @Override  
330 public boolean addAll(final int index, final Collection<?> collection) {  
331  
332 if (collection == null || collection.size() == 0) {  
333 return false;  
334 }  
335  
336 ensureCapacity((index > size() ? index : size()) + collection.size());  
337  
338 // Call "transform" with first element, before  
339 // List is "grown" to ensure the correct DynaClass  
340 // is set.  
341 if (size() == 0) {  
342 transform(collection.iterator().next());  
343 }  
344  
345 growList(index);  
346  
347 int currentIndex = index;  
348 for (final Object e : collection) {  
349 add(currentIndex++, e);  
350 }  
351  
352 return true;  
353  
354 }  
355  
356 /\*\*  
357 \* <p>Return the element at the specified position.</p>  
358 \*  
359 \* <p>If the position requested is greater than the current  
360 \* size of the List, then the List is automatically  
361 \* <i>grown</i> (and populated) to the appropriate size.</p>  
362 \*  
363 \* @param index The index position to insert the new elements at.  
364 \* @return The element at the specified position.  
365 \*/  
366 @Override  
367 public Object get(final int index) {  
368  
369 growList(index + 1);  
370  
371 return super.get(index);  
372  
373 }  
374  
375 /\*\*  
376 \* <p>Set the element at the specified position.</p>  
377 \*  
378 \* <p>If the position requested is greater than the current  
379 \* size of the List, then the List is automatically  
380 \* <i>grown</i> (and populated) to the appropriate size.</p>  
381 \*  
382 \* @param index The index position to insert the new element at.  
383 \* @param element The new element.  
384 \* @return The new element.  
385 \*/  
386 @Override  
387 public Object set(final int index, final Object element) {  
388  
389 final DynaBean dynaBean = transform(element);  
390  
391 growList(index + 1);  
392  
393 return super.set(index, dynaBean);  
394  
395 }  
396  
397 /\*\*  
398 \* <p>Converts the List to an Array.</p>  
399 \*  
400 \* <p>The type of Array created depends on the contents  
401 \* of the List:</p>  
402 \* <ul>  
403 \* <li>If the List contains only LazyDynaMap type elements  
404 \* then a java.util.Map[] array will be created.</li>  
405 \* <li>If the List contains only elements which are  
406 \* "wrapped" DynaBeans then an Object[] of the most  
407 \* suitable type will be created.</li>  
408 \* <li>...otherwise a DynaBean[] will be created.</li>  
409 \*  
410 \* @return An Array of the elements in this List.  
411 \*/  
412 @Override  
413 public Object[] toArray() {  
414  
415 if (size() == 0 && elementType == null) {  
416 return new LazyDynaBean[0];  
417 }  
418  
419 final Object[] array = (Object[])Array.newInstance(elementType, size());  
420 for (int i = 0; i < size(); i++) {  
421 if (Map.class.isAssignableFrom(elementType)) {  
422 array[i] = ((LazyDynaMap)get(i)).getMap();  
423 } else if (DynaBean.class.isAssignableFrom(elementType)) {  
424 array[i] = get(i);  
425 } else {  
426 array[i] = ((WrapDynaBean)get(i)).getInstance();  
427 }  
428 }  
429 return array;  
430  
431 }  
432  
433 /\*\*  
434 \* <p>Converts the List to an Array of the specified type.</p>  
435 \*  
436 \* @param <T> The type of the array elements  
437 \* @param model The model for the type of array to return  
438 \* @return An Array of the elements in this List.  
439 \*/  
440 @Override  
441 public <T> T[] toArray(final T[] model) {  
442  
443 final Class<?> arrayType = model.getClass().getComponentType();  
444 if ((DynaBean.class.isAssignableFrom(arrayType))  
445 || (size() == 0 && elementType == null)) {  
446 return super.toArray(model);  
447 }  
448  
449 if ((arrayType.isAssignableFrom(elementType))) {  
450 T[] array;  
451 if (model.length >= size()) {  
452 array = model;  
453 } else {  
454 @SuppressWarnings("unchecked")  
455 final  
456 // This is safe because we know the element type  
457 T[] tempArray = (T[]) Array.newInstance(arrayType, size());  
458 array = tempArray;  
459 }  
460  
461 for (int i = 0; i < size(); i++) {  
462 Object elem;  
463 if (Map.class.isAssignableFrom(elementType)) {  
464 elem = ((LazyDynaMap) get(i)).getMap();  
465 } else if (DynaBean.class.isAssignableFrom(elementType)) {  
466 elem = get(i);  
467 } else {  
468 elem = ((WrapDynaBean) get(i)).getInstance();  
469 }  
470 Array.set(array, i, elem);  
471 }  
472 return array;  
473 }  
474  
475 throw new IllegalArgumentException("Invalid array type: "  
476 + arrayType.getName() + " - not compatible with '"  
477 + elementType.getName());  
478  
479 }  
480  
481  
482 // ------------------- Public Methods ----------------------------  
483  
484 /\*\*  
485 \* <p>Converts the List to an DynaBean Array.</p>  
486 \*  
487 \* @return A DynaBean[] of the elements in this List.  
488 \*/  
489 public DynaBean[] toDynaBeanArray() {  
490  
491 if (size() == 0 && elementDynaBeanType == null) {  
492 return new LazyDynaBean[0];  
493 }  
494  
495 final DynaBean[] array = (DynaBean[])Array.newInstance(elementDynaBeanType, size());  
496 for (int i = 0; i < size(); i++) {  
497 array[i] = (DynaBean)get(i);  
498 }  
499 return array;  
500  
501 }  
502  
503 /\*\*  
504 \* <p>Set the element Type and DynaClass.</p>  
505 \*  
506 \* @param elementType The type of the elements.  
507 \* @throws IllegalArgumentException if the List already  
508 \* contains elements or the DynaClass is null.  
509 \*/  
510 public void setElementType(final Class<?> elementType) {  
511  
512 if (elementType == null) {  
513 throw new IllegalArgumentException("Element Type is missing");  
514 }  
515  
516 final boolean changeType = (this.elementType != null && !this.elementType.equals(elementType));  
517 if (changeType && size() > 0) {  
518 throw new IllegalStateException("Element Type cannot be reset");  
519 }  
520  
521 this.elementType = elementType;  
522  
523 // Create a new object of the specified type  
524 Object object = null;  
525 try {  
526 object = elementType.newInstance();  
527 } catch (final Exception e) {  
528 throw new IllegalArgumentException("Error creating type: "  
529 + elementType.getName() + " - " + e);  
530 }  
531  
532 // Create a DynaBean  
533 DynaBean dynaBean = null;  
534 if (Map.class.isAssignableFrom(elementType)) {  
535 dynaBean = createDynaBeanForMapProperty(object);  
536 this.elementDynaClass = dynaBean.getDynaClass();  
537 } else if (DynaBean.class.isAssignableFrom(elementType)) {  
538 dynaBean = (DynaBean)object;  
539 this.elementDynaClass = dynaBean.getDynaClass();  
540 } else {  
541 dynaBean = new WrapDynaBean(object);  
542 this.wrapDynaClass = (WrapDynaClass)dynaBean.getDynaClass();  
543 }  
544  
545 this.elementDynaBeanType = dynaBean.getClass();  
546  
547 // Re-calculate the type  
548 if (WrapDynaBean.class.isAssignableFrom(elementDynaBeanType )) {  
549 this.elementType = ((WrapDynaBean)dynaBean).getInstance().getClass();  
550 } else if (LazyDynaMap.class.isAssignableFrom(elementDynaBeanType )) {  
551 this.elementType = ((LazyDynaMap)dynaBean).getMap().getClass();  
552 }  
553  
554 }  
555  
556 /\*\*  
557 \* <p>Set the element Type and DynaClass.</p>  
558 \*  
559 \* @param elementDynaClass The DynaClass of the elements.  
560 \* @throws IllegalArgumentException if the List already  
561 \* contains elements or the DynaClass is null.  
562 \*/  
563 public void setElementDynaClass(final DynaClass elementDynaClass) {  
564  
565 if (elementDynaClass == null) {  
566 throw new IllegalArgumentException("Element DynaClass is missing");  
567 }  
568  
569 if (size() > 0) {  
570 throw new IllegalStateException("Element DynaClass cannot be reset");  
571 }  
572  
573 // Try to create a new instance of the DynaBean  
574 try {  
575 final DynaBean dynaBean = elementDynaClass.newInstance();  
576 this.elementDynaBeanType = dynaBean.getClass();  
577 if (WrapDynaBean.class.isAssignableFrom(elementDynaBeanType)) {  
578 this.elementType = ((WrapDynaBean)dynaBean).getInstance().getClass();  
579 this.wrapDynaClass = (WrapDynaClass)elementDynaClass;  
580 } else if (LazyDynaMap.class.isAssignableFrom(elementDynaBeanType)) {  
581 this.elementType = ((LazyDynaMap)dynaBean).getMap().getClass();  
582 this.elementDynaClass = elementDynaClass;  
583 } else {  
584 this.elementType = dynaBean.getClass();  
585 this.elementDynaClass = elementDynaClass;  
586 }  
587 } catch (final Exception e) {  
588 throw new IllegalArgumentException(  
589 "Error creating DynaBean from " +  
590 elementDynaClass.getClass().getName() + " - " + e);  
591 }  
592  
593 }  
594  
595  
596 // ------------------- Private Methods ---------------------------  
597  
598 /\*\*  
599 \* <p>Automatically <i>grown</i> the List  
600 \* to the appropriate size, populating with  
601 \* DynaBeans.</p>  
602 \*  
603 \* @param requiredSize the required size of the List.  
604 \*/  
605 private void growList(final int requiredSize) {  
606  
607 if (requiredSize < size()) {  
608 return;  
609 }  
610  
611 ensureCapacity(requiredSize + 1);  
612  
613 for (int i = size(); i < requiredSize; i++) {  
614 final DynaBean dynaBean = transform(null);  
615 super.add(dynaBean);  
616 }  
617  
618 }  
619  
620 /\*\*  
621 \* <p>Transform the element into a DynaBean:</p>  
622 \*  
623 \* <ul>  
624 \* <li>Map elements are turned into LazyDynaMap's.</li>  
625 \* <li>POJO Beans are "wrapped" in a WrapDynaBean.</li>  
626 \* <li>DynaBeans are unchanged.</li>  
627 \* </li>  
628 \*  
629 \* @param element The element to transformed.  
630 \* @param The DynaBean to store in the List.  
631 \*/  
632 private DynaBean transform(final Object element) {  
633  
634 DynaBean dynaBean = null;  
635 Class<?> newDynaBeanType = null;  
636 Class<?> newElementType = null;  
637  
638 // Create a new element  
639 if (element == null) {  
640  
641 // Default Types to LazyDynaBean  
642 // if not specified  
643 if (elementType == null) {  
644 setElementDynaClass(new LazyDynaClass());  
645 }  
646  
647 // Get DynaClass (restore WrapDynaClass lost in serialization)  
648 if (getDynaClass() == null) {  
649 setElementType(elementType);  
650 }  
651  
652 // Create a new DynaBean  
653 try {  
654 dynaBean = getDynaClass().newInstance();  
655 newDynaBeanType = dynaBean.getClass();  
656 } catch (final Exception e) {  
657 throw new IllegalArgumentException("Error creating DynaBean: "  
658 + getDynaClass().getClass().getName()  
659 + " - " + e);  
660 }  
661  
662 } else {  
663  
664 // Transform Object to a DynaBean  
665 newElementType = element.getClass();  
666 if (Map.class.isAssignableFrom(element.getClass())) {  
667 dynaBean = createDynaBeanForMapProperty(element);  
668 } else if (DynaBean.class.isAssignableFrom(element.getClass())) {  
669 dynaBean = (DynaBean)element;  
670 } else {  
671 dynaBean = new WrapDynaBean(element);  
672 }  
673  
674 newDynaBeanType = dynaBean.getClass();  
675  
676 }  
677  
678 // Re-calculate the element type  
679 newElementType = dynaBean.getClass();  
680 if (WrapDynaBean.class.isAssignableFrom(newDynaBeanType)) {  
681 newElementType = ((WrapDynaBean)dynaBean).getInstance().getClass();  
682 } else if (LazyDynaMap.class.isAssignableFrom(newDynaBeanType)) {  
683 newElementType = ((LazyDynaMap)dynaBean).getMap().getClass();  
684 }  
685  
686 // Check the new element type, matches all the  
687 // other elements in the List  
688 if (elementType != null && !newElementType.equals(elementType)) {  
689 throw new IllegalArgumentException("Element Type " + newElementType  
690 + " doesn't match other elements " + elementType);  
691 }  
692  
693 return dynaBean;  
694  
695 }  
696  
697 /\*\*  
698 \* Creates a new {@code LazyDynaMap} object for the given property value.  
699 \*  
700 \* @param value the property value  
701 \* @return the newly created {@code LazyDynaMap}  
702 \*/  
703 private LazyDynaMap createDynaBeanForMapProperty(final Object value) {  
704 @SuppressWarnings("unchecked")  
705 final  
706 // map properties are always stored as Map<String, Object>  
707 Map<String, Object> valueMap = (Map<String, Object>) value;  
708 return new LazyDynaMap(valueMap);  
709 }  
710  
711 /\*\*  
712 \* Return the DynaClass.  
713 \*/  
714 private DynaClass getDynaClass() {  
715 return (elementDynaClass == null ? wrapDynaClass : elementDynaClass);  
716 }  
717}