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  
021  
022import java.io.Serializable;  
023import java.lang.reflect.Constructor;  
024import java.lang.reflect.InvocationTargetException;  
025import java.util.HashMap;  
026  
027  
028/\*\*  
029 \* <p>Minimal implementation of the <code>DynaClass</code> interface. Can be  
030 \* used as a convenience base class for more sophisticated implementations.</p> \*  
031 \* <p><strong>IMPLEMENTATION NOTE</strong> - The <code>DynaBean</code>  
032 \* implementation class supplied to our constructor MUST have a one-argument  
033 \* constructor of its own that accepts a <code>DynaClass</code>. This is  
034 \* used to associate the DynaBean instance with this DynaClass.</p>  
035 \*  
036 \* @version $Id$  
037 \*/  
038  
039public class BasicDynaClass implements DynaClass, Serializable {  
040  
041  
042 // ----------------------------------------------------------- Constructors  
043  
044  
045 /\*\*  
046 \* Construct a new BasicDynaClass with default parameters.  
047 \*/  
048 public BasicDynaClass() {  
049  
050 this(null, null, null);  
051  
052 }  
053  
054  
055 /\*\*  
056 \* Construct a new BasicDynaClass with the specified parameters.  
057 \*  
058 \* @param name Name of this DynaBean class  
059 \* @param dynaBeanClass The implementation class for new instances  
060 \*/  
061 public BasicDynaClass(final String name, final Class<?> dynaBeanClass) {  
062  
063 this(name, dynaBeanClass, null);  
064  
065 }  
066  
067  
068 /\*\*  
069 \* Construct a new BasicDynaClass with the specified parameters.  
070 \*  
071 \* @param name Name of this DynaBean class  
072 \* @param dynaBeanClass The implementation class for new intances  
073 \* @param properties Property descriptors for the supported properties  
074 \*/  
075 public BasicDynaClass(final String name, Class<?> dynaBeanClass,  
076 final DynaProperty[] properties) {  
077  
078 super();  
079 if (name != null) {  
080 this.name = name;  
081 }  
082 if (dynaBeanClass == null) {  
083 dynaBeanClass = BasicDynaBean.class;  
084 }  
085 setDynaBeanClass(dynaBeanClass);  
086 if (properties != null) {  
087 setProperties(properties);  
088 }  
089  
090 }  
091  
092  
093 // ----------------------------------------------------- Instance Variables  
094  
095  
096 /\*\*  
097 \* The constructor of the <code>dynaBeanClass</code> that we will use  
098 \* for creating new instances.  
099 \*/  
100 protected transient Constructor<?> constructor = null;  
101  
102  
103 /\*\*  
104 \* The method signature of the constructor we will use to create  
105 \* new DynaBean instances.  
106 \*/  
107 protected static Class<?>[] constructorTypes = { DynaClass.class };  
108  
109  
110 /\*\*  
111 \* The argument values to be passed to the constructore we will use  
112 \* to create new DynaBean instances.  
113 \*/  
114 protected Object[] constructorValues = { this };  
115  
116  
117 /\*\*  
118 \* The <code>DynaBean</code> implementation class we will use for  
119 \* creating new instances.  
120 \*/  
121 protected Class<?> dynaBeanClass = BasicDynaBean.class;  
122  
123  
124 /\*\*  
125 \* The "name" of this DynaBean class.  
126 \*/  
127 protected String name = this.getClass().getName();  
128  
129  
130 /\*\*  
131 \* The set of dynamic properties that are part of this DynaClass.  
132 \*/  
133 protected DynaProperty[] properties = new DynaProperty[0];  
134  
135  
136 /\*\*  
137 \* The set of dynamic properties that are part of this DynaClass,  
138 \* keyed by the property name. Individual descriptor instances will  
139 \* be the same instances as those in the <code>properties</code> list.  
140 \*/  
141 protected HashMap<String, DynaProperty> propertiesMap = new HashMap<String, DynaProperty>();  
142  
143  
144 // ------------------------------------------------------ DynaClass Methods  
145  
146  
147 /\*\*  
148 \* Return the name of this DynaClass (analogous to the  
149 \* <code>getName()</code> method of <code>java.lang.Class</code), which  
150 \* allows the same <code>DynaClass</code> implementation class to support  
151 \* different dynamic classes, with different sets of properties.  
152 \*  
153 \* @return the name of the DynaClass  
154 \*/  
155 public String getName() {  
156  
157 return (this.name);  
158  
159 }  
160  
161  
162 /\*\*  
163 \* Return a property descriptor for the specified property, if it exists;  
164 \* otherwise, return <code>null</code>.  
165 \*  
166 \* @param name Name of the dynamic property for which a descriptor  
167 \* is requested  
168 \* @return The descriptor for the specified property  
169 \*  
170 \* @throws IllegalArgumentException if no property name is specified  
171 \*/  
172 public DynaProperty getDynaProperty(final String name) {  
173  
174 if (name == null) {  
175 throw new IllegalArgumentException  
176 ("No property name specified");  
177 }  
178 return propertiesMap.get(name);  
179  
180 }  
181  
182  
183 /\*\*  
184 \* <p>Return an array of <code>ProperyDescriptors</code> for the properties  
185 \* currently defined in this DynaClass. If no properties are defined, a  
186 \* zero-length array will be returned.</p>  
187 \*  
188 \* <p><strong>FIXME</strong> - Should we really be implementing  
189 \* <code>getBeanInfo()</code> instead, which returns property descriptors  
190 \* and a bunch of other stuff?</p>  
191 \*  
192 \* @return the set of properties for this DynaClass  
193 \*/  
194 public DynaProperty[] getDynaProperties() {  
195  
196 return (properties);  
197  
198 }  
199  
200  
201 /\*\*  
202 \* Instantiate and return a new DynaBean instance, associated  
203 \* with this DynaClass.  
204 \*  
205 \* @return A new <code>DynaBean</code> instance  
206 \* @throws IllegalAccessException if the Class or the appropriate  
207 \* constructor is not accessible  
208 \* @throws InstantiationException if this Class represents an abstract  
209 \* class, an array class, a primitive type, or void; or if instantiation  
210 \* fails for some other reason  
211 \*/  
212 public DynaBean newInstance()  
213 throws IllegalAccessException, InstantiationException {  
214  
215 try {  
216 // Refind the constructor after a deserialization (if needed)  
217 if (constructor == null) {  
218 setDynaBeanClass(this.dynaBeanClass);  
219 }  
220 // Invoke the constructor to create a new bean instance  
221 return ((DynaBean) constructor.newInstance(constructorValues));  
222 } catch (final InvocationTargetException e) {  
223 throw new InstantiationException  
224 (e.getTargetException().getMessage());  
225 }  
226  
227 }  
228  
229  
230 // --------------------------------------------------------- Public Methods  
231  
232  
233 /\*\*  
234 \* Return the Class object we will use to create new instances in the  
235 \* <code>newInstance()</code> method. This Class <strong>MUST</strong>  
236 \* implement the <code>DynaBean</code> interface.  
237 \*  
238 \* @return The class of the {@link DynaBean}  
239 \*/  
240 public Class<?> getDynaBeanClass() {  
241  
242 return (this.dynaBeanClass);  
243  
244 }  
245  
246  
247 // ------------------------------------------------------ Protected Methods  
248  
249  
250 /\*\*  
251 \* Set the Class object we will use to create new instances in the  
252 \* <code>newInstance()</code> method. This Class <strong>MUST</strong>  
253 \* implement the <code>DynaBean</code> interface.  
254 \*  
255 \* @param dynaBeanClass The new Class object  
256 \*  
257 \* @throws IllegalArgumentException if the specified Class does not  
258 \* implement the <code>DynaBean</code> interface  
259 \*/  
260 protected void setDynaBeanClass(final Class<?> dynaBeanClass) {  
261  
262 // Validate the argument type specified  
263 if (dynaBeanClass.isInterface()) {  
264 throw new IllegalArgumentException  
265 ("Class " + dynaBeanClass.getName() +  
266 " is an interface, not a class");  
267 }  
268 if (!DynaBean.class.isAssignableFrom(dynaBeanClass)) {  
269 throw new IllegalArgumentException  
270 ("Class " + dynaBeanClass.getName() +  
271 " does not implement DynaBean");  
272 }  
273  
274 // Identify the Constructor we will use in newInstance()  
275 try {  
276 this.constructor = dynaBeanClass.getConstructor(constructorTypes);  
277 } catch (final NoSuchMethodException e) {  
278 throw new IllegalArgumentException  
279 ("Class " + dynaBeanClass.getName() +  
280 " does not have an appropriate constructor");  
281 }  
282 this.dynaBeanClass = dynaBeanClass;  
283  
284 }  
285  
286  
287 /\*\*  
288 \* Set the list of dynamic properties supported by this DynaClass.  
289 \*  
290 \* @param properties List of dynamic properties to be supported  
291 \*/  
292 protected void setProperties(final DynaProperty[] properties) {  
293  
294 this.properties = properties;  
295 propertiesMap.clear();  
296 for (DynaProperty propertie : properties) {  
297 propertiesMap.put(propertie.getName(), propertie);  
298 }  
299  
300 }  
301  
302  
303}