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.beans.BeanInfo;  
020import java.beans.IndexedPropertyDescriptor;  
021import java.beans.IntrospectionException;  
022import java.beans.Introspector;  
023import java.beans.PropertyDescriptor;  
024import java.lang.reflect.Method;  
025import java.util.List;  
026  
027import org.apache.commons.logging.Log;  
028import org.apache.commons.logging.LogFactory;  
029  
030/\*\*  
031 \* <p>  
032 \* The default {@link BeanIntrospector} implementation.  
033 \* </p>  
034 \* <p>  
035 \* This class implements a default bean introspection algorithm based on the JDK  
036 \* classes in the <code>java.beans</code> package. It discovers properties  
037 \* conforming to the Java Beans specification.  
038 \* </p>  
039 \* <p>  
040 \* This class is a singleton. The single instance can be obtained using the  
041 \* {@code INSTANCE} field. It does not define any state and thus can be  
042 \* shared by arbitrary clients. {@link PropertyUtils} per default uses this  
043 \* instance as its only {@code BeanIntrospector} object.  
044 \* </p>  
045 \*  
046 \* @version $Id$  
047 \* @since 1.9  
048 \*/  
049public class DefaultBeanIntrospector implements BeanIntrospector {  
050 /\*\* The singleton instance of this class. \*/  
051 public static final BeanIntrospector INSTANCE = new DefaultBeanIntrospector();  
052  
053 /\*\* Constant for argument types of a method that expects no arguments. \*/  
054 private static final Class<?>[] EMPTY\_CLASS\_PARAMETERS = new Class[0];  
055  
056 /\*\* Constant for arguments types of a method that expects a list argument. \*/  
057 private static final Class<?>[] LIST\_CLASS\_PARAMETER = new Class[] { java.util.List.class };  
058  
059 /\*\* Log instance \*/  
060 private final Log log = LogFactory.getLog(getClass());  
061  
062 /\*\*  
063 \* Private constructor so that no instances can be created.  
064 \*/  
065 private DefaultBeanIntrospector() {  
066 }  
067  
068 /\*\*  
069 \* Performs introspection of a specific Java class. This implementation uses  
070 \* the {@code java.beans.Introspector.getBeanInfo()} method to obtain  
071 \* all property descriptors for the current class and adds them to the  
072 \* passed in introspection context.  
073 \*  
074 \* @param icontext the introspection context  
075 \*/  
076 public void introspect(final IntrospectionContext icontext) {  
077 BeanInfo beanInfo = null;  
078 try {  
079 beanInfo = Introspector.getBeanInfo(icontext.getTargetClass());  
080 } catch (final IntrospectionException e) {  
081 // no descriptors are added to the context  
082 log.error(  
083 "Error when inspecting class " + icontext.getTargetClass(),  
084 e);  
085 return;  
086 }  
087  
088 PropertyDescriptor[] descriptors = beanInfo.getPropertyDescriptors();  
089 if (descriptors == null) {  
090 descriptors = new PropertyDescriptor[0];  
091 }  
092  
093 handleIndexedPropertyDescriptors(icontext.getTargetClass(),  
094 descriptors);  
095 icontext.addPropertyDescriptors(descriptors);  
096 }  
097  
098 /\*\*  
099 \* This method fixes an issue where IndexedPropertyDescriptor behaves  
100 \* differently in different versions of the JDK for 'indexed' properties  
101 \* which use java.util.List (rather than an array). It implements a  
102 \* workaround for Bug 28358. If you have a Bean with the following  
103 \* getters/setters for an indexed property:  
104 \*  
105 \* <pre>  
106 \* public List getFoo()  
107 \* public Object getFoo(int index)  
108 \* public void setFoo(List foo)  
109 \* public void setFoo(int index, Object foo)  
110 \* </pre>  
111 \*  
112 \* then the IndexedPropertyDescriptor's getReadMethod() and getWriteMethod()  
113 \* behave as follows:  
114 \* <ul>  
115 \* <li>JDK 1.3.1\_04: returns valid Method objects from these methods.</li>  
116 \* <li>JDK 1.4.2\_05: returns null from these methods.</li>  
117 \* </ul>  
118 \*  
119 \* @param beanClass the current class to be inspected  
120 \* @param descriptors the array with property descriptors  
121 \*/  
122 private void handleIndexedPropertyDescriptors(final Class<?> beanClass,  
123 final PropertyDescriptor[] descriptors) {  
124 for (final PropertyDescriptor pd : descriptors) {  
125 if (pd instanceof IndexedPropertyDescriptor) {  
126 final IndexedPropertyDescriptor descriptor = (IndexedPropertyDescriptor) pd;  
127 final String propName = descriptor.getName().substring(0, 1)  
128 .toUpperCase()  
129 + descriptor.getName().substring(1);  
130  
131 if (descriptor.getReadMethod() == null) {  
132 final String methodName = descriptor.getIndexedReadMethod() != null ? descriptor  
133 .getIndexedReadMethod().getName() : "get"  
134 + propName;  
135 final Method readMethod = MethodUtils  
136 .getMatchingAccessibleMethod(beanClass, methodName,  
137 EMPTY\_CLASS\_PARAMETERS);  
138 if (readMethod != null) {  
139 try {  
140 descriptor.setReadMethod(readMethod);  
141 } catch (final Exception e) {  
142 log.error(  
143 "Error setting indexed property read method",  
144 e);  
145 }  
146 }  
147 }  
148 if (descriptor.getWriteMethod() == null) {  
149 final String methodName = descriptor.getIndexedWriteMethod() != null ? descriptor  
150 .getIndexedWriteMethod().getName() : "set"  
151 + propName;  
152 Method writeMethod = MethodUtils  
153 .getMatchingAccessibleMethod(beanClass, methodName,  
154 LIST\_CLASS\_PARAMETER);  
155 if (writeMethod == null) {  
156 for (final Method m : beanClass.getMethods()) {  
157 if (m.getName().equals(methodName)) {  
158 final Class<?>[] parameterTypes = m.getParameterTypes();  
159 if (parameterTypes.length == 1  
160 && List.class  
161 .isAssignableFrom(parameterTypes[0])) {  
162 writeMethod = m;  
163 break;  
164 }  
165 }  
166 }  
167 }  
168 if (writeMethod != null) {  
169 try {  
170 descriptor.setWriteMethod(writeMethod);  
171 } catch (final Exception e) {  
172 log.error(  
173 "Error setting indexed property write method",  
174 e);  
175 }  
176 }  
177 }  
178 }  
179 }  
180 }  
181}