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.converters;  
020  
021  
022import java.util.List;  
023import org.apache.commons.beanutils.ConversionException;  
024  
025  
026/\*\*  
027 \* <p>Standard {@link org.apache.commons.beanutils.Converter} implementation that converts an incoming  
028 \* String into a primitive array of boolean. On a conversion failure, returns  
029 \* a specified default value or throws a {@link ConversionException} depending  
030 \* on how this instance is constructed.</p>  
031 \*  
032 \* <p>By default, the values to be converted are expected to be those  
033 \* recognized by a default instance of BooleanConverter. A customised  
034 \* BooleanConverter can be provided in order to recognise alternative values  
035 \* as true/false. </p>  
036 \*  
037 \* @version $Id$  
038 \* @since 1.4  
039 \* @deprecated Replaced by the new {@link ArrayConverter} implementation  
040 \*/  
041  
042@Deprecated  
043public final class BooleanArrayConverter extends AbstractArrayConverter {  
044  
045  
046 // ----------------------------------------------------------- Constructors  
047  
048  
049 /\*\*  
050 \* Create a {@link org.apache.commons.beanutils.Converter} that will throw  
051 \* a {@link ConversionException} if a conversion error occurs.  
052 \*  
053 \* <p>Conversion of strings to boolean values will be done via a default  
054 \* instance of class BooleanConverter.</p>  
055 \*/  
056 public BooleanArrayConverter() {  
057  
058 super();  
059 this.booleanConverter = DEFAULT\_CONVERTER;  
060  
061 }  
062  
063  
064 /\*\*  
065 \* Create a {@link org.apache.commons.beanutils.Converter} that will return  
066 \* the specified default value if a conversion error occurs.  
067 \*  
068 \* <p>Conversion of strings to boolean values will be done via a default  
069 \* instance of class BooleanConverter.</p>  
070 \*  
071 \* @param defaultValue The default value to be returned  
072 \*/  
073 public BooleanArrayConverter(final Object defaultValue) {  
074  
075 super(defaultValue);  
076 this.booleanConverter = DEFAULT\_CONVERTER;  
077  
078 }  
079  
080  
081 /\*\*  
082 \* Create a {@link org.apache.commons.beanutils.Converter} that will return  
083 \* the specified default value if a conversion error occurs.  
084 \*  
085 \* <p>Conversion of strings to boolean values will be done via the  
086 \* specified converter.</p>  
087 \*  
088 \* @param converter is the converter object that will be used to  
089 \* convert each input string-value into a boolean.  
090 \*  
091 \* @param defaultValue is the default value to be returned by method  
092 \* convert if conversion fails; null is a valid default value. See the  
093 \* documentation for method "convert" for more information.  
094 \* The value BooleanArrayConverter.NO\_DEFAULT may be passed here to  
095 \* specify that an exception should be thrown on conversion failure.  
096 \*  
097 \*/  
098 public BooleanArrayConverter(final BooleanConverter converter, final Object defaultValue) {  
099  
100 super(defaultValue);  
101 this.booleanConverter = converter;  
102  
103 }  
104  
105 // ------------------------------------------------------- Static Variables  
106  
107 /\*\*  
108 \* Type which this class converts its input to. This value can be  
109 \* used as a parameter to the ConvertUtils.register method.  
110 \* @since 1.8.0  
111 \*/  
112 public static final Class MODEL = new boolean[0].getClass();  
113  
114 /\*\*  
115 \* The converter that all instances of this class will use to  
116 \* do individual string->boolean conversions, unless overridden  
117 \* in the constructor.  
118 \*/  
119 private static final BooleanConverter DEFAULT\_CONVERTER  
120 = new BooleanConverter();  
121  
122 // ---------------------------------------------------- Instance Variables  
123  
124 /\*\*  
125 \* This object is used to perform the conversion of individual strings  
126 \* into Boolean/boolean values.  
127 \*/  
128 protected final BooleanConverter booleanConverter;  
129  
130 // --------------------------------------------------------- Public Methods  
131  
132  
133 /\*\*  
134 \* Convert the specified input object into an output object of type  
135 \* array-of-boolean.  
136 \*  
137 \* <p>If the input value is null, then the default value specified in the  
138 \* constructor is returned. If no such value was provided, then a  
139 \* ConversionException is thrown instead.</p>  
140 \*  
141 \* <p>If the input value is of type String[] then the returned array shall  
142 \* be of the same size as this array, with a true or false value in each  
143 \* array element depending on the result of applying method  
144 \* BooleanConverter.convert to each string.</p>  
145 \*  
146 \* <p>For all other types of value, the object's toString method is  
147 \* expected to return a string containing a comma-separated list of  
148 \* values, eg "true, false, true". See the documentation for  
149 \* {@link AbstractArrayConverter#parseElements} for more information on  
150 \* the exact formats supported.</p>  
151 \*  
152 \* <p>If the result of value.toString() cannot be split into separate  
153 \* words, then the default value is also returned (or an exception thrown).  
154 \* </p>  
155 \*  
156 \* <p>If any of the elements in the value array (or the elements resulting  
157 \* from splitting up value.toString) are not recognized by the  
158 \* BooleanConverter associated with this object, then what happens depends  
159 \* on whether that BooleanConverter has a default value or not: if it does,  
160 \* then that unrecognized element is converted into the BooleanConverter's  
161 \* default value. If the BooleanConverter does <i>not</i> have a default  
162 \* value, then the default value for this object is returned as the  
163 \* <i>complete</i> conversion result (not just for the element), or an  
164 \* exception is thrown if this object has no default value defined.</p>  
165 \*  
166 \* @param type is the type to which this value should be converted. In the  
167 \* case of this BooleanArrayConverter class, this value is ignored.  
168 \*  
169 \* @param value is the input value to be converted.  
170 \*  
171 \* @return an object of type boolean[], or the default value if there was  
172 \* any sort of error during conversion and the constructor  
173 \* was provided with a default value.  
174 \*  
175 \* @throws ConversionException if conversion cannot be performed  
176 \* successfully and the constructor was not provided with a default  
177 \* value to return on conversion failure.  
178 \*  
179 \* @throws NullPointerException if value is an array, and any of the  
180 \* array elements are null.  
181 \*/  
182 @Override  
183 public Object convert(final Class type, final Object value) {  
184  
185 // Deal with a null value  
186 if (value == null) {  
187 if (useDefault) {  
188 return (defaultValue);  
189 } else {  
190 throw new ConversionException("No value specified");  
191 }  
192 }  
193  
194 // Deal with the no-conversion-needed case  
195 if (MODEL == value.getClass()) {  
196 return (value);  
197 }  
198  
199 // Deal with input value as a String array  
200 //  
201 // TODO: use if (value.getClass().isArray() instead...  
202 // this requires casting to Object[], then using values[i].toString()  
203 if (strings.getClass() == value.getClass()) {  
204 try {  
205 final String[] values = (String[]) value;  
206 final boolean[] results = new boolean[values.length];  
207 for (int i = 0; i < values.length; i++) {  
208 final String stringValue = values[i];  
209 final Object result = booleanConverter.convert(Boolean.class, stringValue);  
210 results[i] = ((Boolean) result).booleanValue();  
211 }  
212 return (results);  
213 } catch (final Exception e) {  
214 if (useDefault) {  
215 return (defaultValue);  
216 } else {  
217 throw new ConversionException(value.toString(), e);  
218 }  
219 }  
220 }  
221  
222 // We only get here if the input value is not of type String[].  
223 // In this case, we assume value.toString() returns a comma-separated  
224 // sequence of values; see method AbstractArrayConverter.parseElements  
225 // for more information.  
226 try {  
227 final List list = parseElements(value.toString());  
228 final boolean[] results = new boolean[list.size()];  
229 for (int i = 0; i < results.length; i++) {  
230 final String stringValue = (String) list.get(i);  
231 final Object result = booleanConverter.convert(Boolean.class, stringValue);  
232 results[i] = ((Boolean) result).booleanValue();  
233 }  
234 return (results);  
235 } catch (final Exception e) {  
236 if (useDefault) {  
237 return (defaultValue);  
238 } else {  
239 throw new ConversionException(value.toString(), e);  
240 }  
241 }  
242  
243 }  
244  
245  
246}