PowerEnJoy

Code Inspection

A.Y 2016/2017

Francesco Tinarelli (matr:806146)

Marco Wenzel (matr:878021)

Versione 1.0

Index

1. Introduction

1.1 Purpose

Code inspection is the systematic examination (often known as peer review) of computer source code. It is intended to find mistakes overlooked during the initial development phase, with the aim of improving both the overall quality of software and the developers' skills.

1.2 Classes Assigned

- **Name**: EntityDataReader
  - **Location:** /apache-ofbiz-16.11.01/framework/entity/src /main/java/org/apache/ofbiz/entity/config/model
  - **Package:** org.apache.ofbiz.entity.config.model
  - **Modifier:** public-final
  - **Method:** EntityDataReader(String), EntityDataReader(Element), getName, getResourceList

- **Name**: SetOperation
  - **Location**: /apache-ofbiz-16.11.01/framework/minilang/ src/main/java/org/apache/ofbiz/minilang/method/envops
  - **Package:** org.apache.ofbiz.minilang.method.envops
  - **Modifier:** public final
  - **Method:** autoCorrect, SetOperation, exec, toString, <inner class> SetOperationFactory

2. EntityDataReader inspection

2.1 Functional role

For understanding the functional role of this class, first we need to understand what are nodes and elements.

The node is the smallest datatype in the DOM, is an interface implemented by almost all object.

The element is a type of node, is an interface that extends the node interface, this type of node corresponding to structure in XML pages.

The EntityDataReader class have two constructors:

The first one permits through a string variable to insert a value on the variable: name; and provide an empty list of resource.

The second one permits through an Element to extract all children elements node that they have the same type of node (element node) and with the node name equals to “resource”. After the extraction insert for each element the respective resource object into the resource list.

For control the unexpected results this class has an exception if the EntityDataReader have already a name value or if the content of the parameter: element is empty.

So, this class should be use for capture the content and the data structure of a specific XML file.

2.2 Analysis method

For analyze this class we start from analyze the modifier of its attributes and methods than the double constructor and the external method calls.

To understand the intrinsic meaning, we use the OfBiz documentation but this work was hard because in this class miss the Javadoc.

2.3 Code

```
19  package org.apache.ofbiz.entity.config.model;
20
21  import java.util.ArrayList;
22  import java.util.Collections;
23  import java.util.List;
24
25  import org.apache.ofbiz.base.lang.ThreadSafe;
26  import org.apache.ofbiz.base.util.UtilXml;
27  import org.apache.ofbiz.entity.GenericEntityConfException;
28  import org.w3c.dom.Element;
29
30  /**
31   * An object that models the <code>&lt;entity-data-reader&gt;</code> element.
32   *
33   * @see <code>entity-config.xsd</code>
34   */
35  @ThreadSafe
36  public final class EntityDataReader {
37
38      private final String name; // type = xs:string
39      private final List<Resource> resourceList; // <resource>
40
41      public EntityDataReader(String name) throws GenericEntityConfException {
42          if (name == null || name.isEmpty()) {
43              throw new GenericEntityConfException("EntityDataReader name cannot be empty");
44          }
45          this.name = name;
46          this.resourceList = Collections.emptyList();
47      }
48
49      EntityDataReader(Element element) throws GenericEntityConfException {
50          String lineNumberText = EntityConfig.createConfigFileLineNumberText(element);
51          String name = element.getAttribute("name").intern();
52          if (name.isEmpty()) {
53              throw new GenericEntityConfException("<entity-data-reader> element name attribute is empty" + lineNumberText);
54          }
55          this.name = name;
56          List<? extends Element> resourceElementList = UtilXml.childElementList(element, "resource");
57          if (resourceElementList.isEmpty()) {
58              this.resourceList = Collections.emptyList();
59          } else {
60              List<Resource> resourceList = new ArrayList<Resource>(resourceElementList.size());
61              for (Element resourceElement : resourceElementList) {
```

```
61              for (Element resourceElement : resourceElementList) {
62                  resourceList.add(new Resource(resourceElement));
63              }
64              this.resourceList = Collections.unmodifiableList(resourceList);
65          }
66      }
67
68      /** Returns the value of the <code>name</code> attribute. */
69      public String getName() {
70          return this.name;
71      }
72
73      /** Returns the <code>&lt;resource&gt;</code> child elements. */
74      public List<Resource> getResourceList() {
75          return this.resourceList;
76      }
77  }
78
```

2.4 Issues

The numbers of the issues refer to the corresponding numbers in the ispection checklist.

1) createConfigFileLineNumberText(element) this method doesn't have a meaningful name because returns a string that indicates the number of the starting line a more meaningful name can be startingLineNumber(element).

13) Line 53 (110c) Line 56 (92c) Line 60(83c).

18) There are no comments about code are doing.

19) No comment for explain the reasoning behind the code.

23) There are no Javadoc about this class.

27) There is double constructor but they are control well and the variable: name is defined two times but the two variables are not the same and is controlled well(this.name=name).

36) lineNumberText variable is not used if the variable: name is empty.

56) That FOR type is correct but we don't know the initial value.

3. SetOperation inspection

3.1 Functional role

This class takes an element and first correct the deprecated value with the autoCorrect() method, then this class "opens" the attribute searching script or operation and puts them into the variables. This searching of the operation is done by using Scriptlet, flexibleMapAccessor and flexibleStringExpander.

First the class searches the Script form the "from-attribute", second uses the flexibleMapAccessor on the same attribute, third searches from the "value" attribute and finally uses the default attribute.

The method exec executes the operation that are in the variables. This method return, true if script execution should continue, or false if script execution should stop. For the control of the execution this class use the exception.

An object SetOperation can be created by the object factory that is an override of the method that is in the MethodContext superclass.

3.2 Analysis method

For this class we do a top-down analysis. We start from understand how is the role of this class in high terms, second we analyze the role of every attributes of the class, entering in detail. Than we analyze the methods and the return of this method. At last we control all the variables and if that are useless, the exception lauched, and the right use of spaces and braces control.

3.3 Code

```
/*******************************************************************************
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 *******************************************************************************/
package org.apache.ofbiz.minilang.method.envops;

import java.util.HashMap;
import java.util.LinkedList;
import java.util.Locale;

import org.apache.ofbiz.base.util.Debug;
import org.apache.ofbiz.base.util.ObjectType;
import org.apache.ofbiz.base.util.Scriptlet;
import org.apache.ofbiz.base.util.StringUtil;
import org.apache.ofbiz.base.util.collections.FlexibleMapAccessor;
import org.apache.ofbiz.base.util.string.FlexibleStringExpander;
import org.apache.ofbiz.minilang.MiniLangException;
import org.apache.ofbiz.minilang.MiniLangUtil;
import org.apache.ofbiz.minilang.MiniLangValidate;
import org.apache.ofbiz.minilang.SimpleMethod;
import org.apache.ofbiz.minilang.method.MethodContext;
import org.apache.ofbiz.minilang.method.MethodOperation;
import org.w3c.dom.Element;

/**
 * Implements the &lt;set&gt; element.
 *
 * @see <a href="https://cwiki.apache.org/confluence/display/OFBADMIN/Mini-language+Reference#Mini-languageReference-{{%3Cset%3E}}">Mini-language Reference</a>
 */
```

```
 */
public final class SetOperation extends MethodOperation {

    public static final String module = SetOperation.class.getName();

    // This method is needed only during the v1 to v2 transition
    private static boolean autoCorrect(Element element) {
        boolean elementModified = false;
        // Correct deprecated default-value attribute
        String defaultAttr = element.getAttribute("default-value");
        if (defaultAttr.length() > 0) {
            element.setAttribute("default", defaultAttr);
            element.removeAttribute("default-value");
            elementModified = true;
        }
        // Correct deprecated from-field attribute
        String fromAttr = element.getAttribute("from-field");
        if (fromAttr.length() > 0) {
            element.setAttribute("from", fromAttr);
            element.removeAttribute("from-field");
            elementModified = true;
        }
        // Correct value attribute expression that belongs in from attribute
        String valueAttr = element.getAttribute("value").trim();
        if (valueAttr.startsWith("${") && valueAttr.endsWith("}")) {
            valueAttr = valueAttr.substring(2, valueAttr.length() - 1);
            if (!valueAttr.contains("${")) {
                element.setAttribute("from", valueAttr);
                element.removeAttribute("value");
                elementModified = true;
            }
        }
        return elementModified;
    }

    private final FlexibleStringExpander defaultFse;
    private final FlexibleStringExpander formatFse;
    private final FlexibleMapAccessor<Object> fieldFma;
    private final FlexibleMapAccessor<Object> fromFma;
    private final Scriptlet scriptlet;
    private final boolean setIfEmpty;
    private final boolean setIfNull;
    private final Class<?> targetClass;
    private final String type;
```

```
    private final String type;
    private final FlexibleStringExpander valueFse;

    public SetOperation(Element element, SimpleMethod simpleMethod) throws MiniLangException {
        super(element, simpleMethod);
        if (MiniLangValidate.validationOn()) {
            MiniLangValidate.deprecatedAttribute(simpleMethod, element, "from-field", "replace with \"from\"");
            MiniLangValidate.deprecatedAttribute(simpleMethod, element, "default-value", "replace with \"default\"");
            MiniLangValidate.attributeNames(simpleMethod, element, "field", "from-field", "from", "value", "default-value", "default", "format", "type", "set-if-null", "set-if-empty");
            MiniLangValidate.requiredAttributes(simpleMethod, element, "field");
            MiniLangValidate.requireAnyAttribute(simpleMethod, element, "from-field", "from", "value");
            MiniLangValidate.constantPlusExpressionAttributes(simpleMethod, element, "value");
            MiniLangValidate.constantAttributes(simpleMethod, element, "type", "set-if-null", "set-if-empty");
            MiniLangValidate.expressionAttributes(simpleMethod, element, "field");
            MiniLangValidate.noChildElements(simpleMethod, element);
        }
        boolean elementModified = autoCorrect(element);
        if (elementModified && MiniLangUtil.autoCorrectOn()) {
            MiniLangUtil.flagDocumentAsCorrected(element);
        }
        this.fieldFma = FlexibleMapAccessor.getInstance(element.getAttribute("field"));
        String fromAttribute = element.getAttribute("from");
        if (MiniLangUtil.containsScript(fromAttribute)) {
            this.scriptlet = new Scriptlet(StringUtil.convertOperatorSubstitutions(fromAttribute));
            this.fromFma = FlexibleMapAccessor.getInstance(null);
        } else {
            this.scriptlet = null;
            this.fromFma = FlexibleMapAccessor.getInstance(fromAttribute);
        }
        this.valueFse = FlexibleStringExpander.getInstance(element.getAttribute("value"));
        this.defaultFse = FlexibleStringExpander.getInstance(element.getAttribute("default"));
        this.formatFse = FlexibleStringExpander.getInstance(element.getAttribute("format"));
        this.type = element.getAttribute("type");
        Class<?> targetClass = null;
        if (!this.type.isEmpty() && !"NewList".equals(this.type) && !"NewMap".equals(this.type)) {
            try {
                targetClass = ObjectType.loadClass(this.type);
            } catch (ClassNotFoundException e) {
                MiniLangValidate.handleError("Invalid type " + this.type, simpleMethod, element);
            }
        }
```

```
        this.targetClass = targetClass;
        this.setIfNull = "true".equals(element.getAttribute("set-if-null")); // default to false, anything but true is false
        this.setIfEmpty = !"false".equals(element.getAttribute("set-if-empty")); // default to true, anything but false is true
        if (!fromAttribute.isEmpty() && !this.valueFse.isEmpty()) {
            throw new IllegalArgumentException("Cannot include both a from attribute and a value attribute in a <set> element.");
        }
    }

    @Override
    public boolean exec(MethodContext methodContext) throws MiniLangException {
        boolean isConstant = false;
        Object newValue = null;
        if (this.scriptlet != null) {
            try {
                newValue = this.scriptlet.executeScript(methodContext.getEnvMap());
            } catch (Exception exc) {
                Debug.logWarning(exc, "Error evaluating scriptlet [" + this.scriptlet + "]: " + exc, module);
            }
        } else if (!this.fromFma.isEmpty()) {
            newValue = this.fromFma.get(methodContext.getEnvMap());
            if (Debug.verboseOn())
                Debug.logVerbose("In screen getting value for field from [" + this.fromFma.toString() + "]: " + newValue, module);
        } else if (!this.valueFse.isEmpty()) {
            newValue = this.valueFse.expand(methodContext.getEnvMap());
            isConstant = true;
        }
        // If newValue is still empty, use the default value
        if (ObjectType.isEmpty(newValue) && !this.defaultFse.isEmpty()) {
            newValue = this.defaultFse.expand(methodContext.getEnvMap());
            isConstant = true;
        }
        if (!setIfNull && newValue == null && !"NewMap".equals(this.type) && !"NewList".equals(this.type)) {
            if (Debug.verboseOn())
                Debug.logVerbose("Field value not found (null) with name [" + fromFma + "] and value [" + valueFse + "], and there was not default value, not setting field", module);
            return true;
        }
        if (!setIfEmpty && ObjectType.isEmpty(newValue)) {
            if (Debug.verboseOn())
                Debug.logVerbose("Field value not found (empty) with name [" + fromFma + "] and value [" + valueFse + "], and there was not default value, not setting field", module);
            return true;
        }
```

```
        if (this.type.length() > 0) {
            if ("NewMap".equals(this.type)) {
                newValue = new HashMap<String, Object>();
            } else if ("NewList".equals(this.type)) {
                newValue = new LinkedList<Object>();
            } else {
                try {
                    String format = null;
                    if (!this.formatFse.isEmpty()) {
                        format = this.formatFse.expandString(methodContext.getEnvMap());
                    }
                    Class<?> targetClass = this.targetClass;
                    if (targetClass == null) {
                        targetClass = MiniLangUtil.getObjectClassForConversion(newValue);
                    }
                    if (isConstant) {
                        // We use en locale here so constant (literal) values are converted properly.
                        newValue = MiniLangUtil.convertType(newValue, targetClass, Locale.ENGLISH, methodContext.getTimeZone(), format);
                    } else {
                        newValue = MiniLangUtil.convertType(newValue, targetClass, methodContext.getLocale(), methodContext.getTimeZone(), format);
                    }
                } catch (Exception e) {
                    String errMsg = "Could not convert field value for the field: [" + this.fieldFma.toString() + "] to the [" + this.type + "] type for the value [" +
                    newValue + "]: " + e.getMessage();
                    Debug.logWarning(e, errMsg, module);
                    this.simpleMethod.addErrorMessage(methodContext, errMsg);
                    return false;
                }
            }
        }
        if (Debug.verboseOn())
            Debug.logVerbose("Setting field [" + this.fieldFma.toString() + "] to value: " + newValue, module);
        this.fieldFma.put(methodContext.getEnvMap(), newValue);
        return true;
    }

```

```
204     @Override
205     public String toString() {
206         StringBuilder sb = new StringBuilder("<set ");
207         if (!this.fieldFma.isEmpty()) {
208             sb.append("field=\"").append(this.fieldFma).append("\" ");
209         }
210         if (!this.fromFma.isEmpty()) {
211             sb.append("from=\"").append(this.fromFma).append("\" ");
212         }
213         if (this.scriptlet != null) {
214             sb.append("from=\"").append(this.scriptlet).append("\" ");
215         }
216         if (!this.valueFse.isEmpty()) {
217             sb.append("value=\"").append(this.valueFse).append("\" ");
218         }
219         if (!this.defaultFse.isEmpty()) {
220             sb.append("default=\"").append(this.defaultFse).append("\" ");
221         }
222         if (this.type.length() > 0) {
223             sb.append("type=\"").append(this.type).append("\" ");
224         }
225         if (this.setIfNull) {
226             sb.append("set-if-null=\"true\" ");
227         }
228         if (!this.setIfEmpty) {
229             sb.append("set-if-empty=\"false\" ");
230         }
231         sb.append("/>");
232         return sb.toString();
233     }
234
```

```
234
235     /**
236      * A factory for the &lt;set&gt; element.
237      */
238     public static final class SetOperationFactory implements Factory<SetOperation> {
239         @Override
240         public SetOperation createMethodOperation(Element element, SimpleMethod simpleMethod) throws MiniLangException {
241             return new SetOperation(element, simpleMethod);
242         }
243
244         @Override
245         public String getName() {
246             return "set";
247         }
248     }
249 }
```

3.4 Issues

1) SetOperation it's a mining full name, don't explain what the class do.

11) Line 147-159-164-197 one statement without graph braces.

13) Line 94, line 128, line 129, line 143, line 148, line 160, line 165, line 187, line 190.

14) Line 94(172c) Line 160(166c) Line 165(168c) Line 187(123c) Line 190(182c).

18) The comment at the beginning of the class is useless, the internet page that the URL of this document see is moved to another page.

19) All the comment out of code are without the date.

23) There are no Javadoc about this class.

25) AutoCorrect() method came first than private final variables.

33) Line 107: variable stringAttribute.

40) Line 42, line 158, line 180.

4. Appendix

4.1 Reference

Code Inspection Assignment Task Description.pdf.

https://ofbiz.apache.org/documentation.html.

4.2 Hours of work

Marco Wenzel: 10 hours.

Francesco Tinarelli: 8 hours.