POLITECNICO
MILANO 1863

PowerEnJoy Project

EPJ

Apache Open For Business (OFBiz) 16.11.01
Code Inspection Document

Alfredo Maria Fomitchenko (mat. 874656)

Version: 1.0

Release date: 5 February 2017

1. Classes that were assigned to the group

The class I have been assigned to is

```
../
apache-ofbiz-16.11.01/
  framework/
    service/
      src/
        main/
          java/
            org/
              apache/
                ofbiz/
                  service/
                    engine/
                      SOAPClientEngine.java
```

2. Functional role of assigned set of classes

`SOAPClientEngine` implements a SOAP service engine.

A Simple Object Access Protocol is an XML-based messaging protocol that provides the envelope for sending Web Services messages over the Internet via HTTP, an alternative to Representation State Transfer (REST) and JavaScript Object Notation (JSON) even though the last two are often used thanks to their flexibility and lack of rigid standardization. It is also particularly useful for performing Remote Procedure Call request-response dialogues, a common distributed computing framework in which a client causes the execution of a procedure onto another computer, as if the procedure were coded locally.

`SOAPClientEngine` extends other engine classes that ultimately implement the `GenericEngine` interface. This interface abstract methods are divided into synchronous and asynchronous service types. Since a SOAP puts in place a messaging protocol, the class has no interest in adding asynchronous functionalities to a generic asynchronous engine (in fact it extends the `GenericAsyncEngine` class) and instead it overrides `runSync` calling its own peculiar `serviceInvoker` method.

The class heavily resides onto Axis2 resources, the core Apache project related to Web Services / SOAP / Web Services Description Language engines.

3. List of issues found by applying the checklist

In the following, code is quoted within … symbols, which represent omissis code lines that precedes or follows.

3.1 Naming Conventions

> 5. Method names should be verbs, with the first letter of each addition word capitalized. Examples: getBackground(); computeTemperature().

The main computational class method name is not a verb, a noun instead:

```
…
    // Invoke the remote SOAP service
    private Map<String, Object> serviceInvoker(ModelService modelService, Map<String, Object>
context) throws GenericServiceException {
…
```

7. Constants are declared using all uppercase with words separated by an underscore. Examples: MIN WIDTH; MAX HEIGHT.

The string ***module*** is defined as **final**, but its name is not all uppercased:

```
…
    public static final String module = SOAPClientEngine.class.getName();
…
```

However, after some other classes inspection I noticed this practice of putting the class name in a `public static final String module` is widely spread, so I assumed the practice is commonly accepted within the project.

3.2 Braces

11. All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces. Example: Avoid this: …

One-line if statements without braces are common within the class, but half of them consitutes log checks before writing:

```
…
        if (Debug.infoOn()) Debug.logInfo("[SOAPClientEngine.invoke] : Parameter length - " +
inModelParamList.size(), module);
…

…
            if (Debug.infoOn()) Debug.logInfo("[SOAPClientEngine.invoke} : Parameter: " +
p.name + " (" + p.mode + ") - " + i, module);
…
```

However, the other if statements represent very important logical parts, and readability should have been key:

```
…
    public Map<String, Object> runSync(String localName, ModelService modelService,
Map<String, Object> context) throws GenericServiceException {
        Map<String, Object> result = serviceInvoker(modelService, context);

        if (result == null)
            throw new GenericServiceException("Service did not return expected result");
        return result;
    }
…
```

```
…
    private Map<String, Object> serviceInvoker(ModelService modelService, Map<String, Object>
context) throws GenericServiceException {
        Delegator delegator = dispatcher.getDelegator();
        if (modelService.location == null || modelService.invoke == null)
            throw new GenericServiceException("Cannot locate service to invoke");
…
```

3.3 File Organization

12. Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).

As shown in the Comments paragraph, code authors have been very greedy for comments. Then the only possible way to separate sections throughout the code are blank lines, which however are not always used consistently.

In particular, all the issues found involve one-line if statements, which are not spaced enough from lines before or after:

```
…
        Map<String, Object> result = serviceInvoker(modelService, context);

        if (result == null)
            throw new GenericServiceException("Service did not return expected result");
        return result;
…
```

```
…
        Delegator delegator = dispatcher.getDelegator();
        if (modelService.location == null || modelService.invoke == null)
            throw new GenericServiceException("Cannot locate service to invoke");
…
```

```
…
           if (!p.internal) {
                parameterMap.put(p.name, context.get(p.name));
           }
           i++;
…
```

Also, an empty Map declaration could have been spaced from the following for loop:

```
…
           int i = 0;

           Map<String, Object> parameterMap = new HashMap<String, Object>();
           for (ModelParam p: inModelParamList) {
…
```

13. Where practical, line length does not exceed 80 characters.

Overall lines exceeding 80 characters are 16. Seven of them, illustrated in the following list for completeness sake, exceed the limit by no more than 5–10 characters making the issue overall negligible, whereas the other 9 lines are discussed in the next paragraph:

```
…
            throw new GenericServiceException("Service did not return expected result");
…
            throw new GenericServiceException("Cannot locate service to invoke");
…
        String axis2XmlFileLocation = System.getProperty("ofbiz.home") + axis2XmlFile;
…
            EndpointReference endPoint = new
EndpointReference(this.getLocation(modelService));
…
            serviceName = new QName(modelService.nameSpace, modelService.invoke);
…
            // exclude params that ModelServiceReader insert into (internal params)
…
            OMXMLParserWrapper builder = OMXMLBuilderFactory.createStAXOMBuilder(reader);
…
```

14. When line length must exceed 80 characters, it does NOT exceed 120 characters.

Lines exceeding 120 characters are 9 in total. While some may not cause major issues being javadoc parts, namely

```
…
     * @see org.apache.ofbiz.service.engine.GenericEngine#runSyncIgnore(java.lang.String,
org.apache.ofbiz.service.ModelService, java.util.Map)
…
     * @see org.apache.ofbiz.service.engine.GenericEngine#runSync(java.lang.String,
org.apache.ofbiz.service.ModelService, java.util.Map)
…
```

the other should have definitely been broken.

The following two overriden methods below could have been broken, giving also the opportunity to highlight the similarities between them:

```
…
    public void runSyncIgnore(String localName, ModelService modelService, Map<String, Object>
context) throws GenericServiceException {
…
    public Map<String, Object> runSync(String localName, ModelService modelService,
Map<String, Object> context) throws GenericServiceException {
…
```

Breaking the first log string into two different semantic parts, even though inserting one redundant + operator, would have produced two code lines of even appearance:

```
…
        if (Debug.infoOn()) Debug.logInfo("[SOAPClientEngine.invoke] : Parameter length - " +
inModelParamList.size(), module);
…
```

In the following code line, many different + operators give many opportunities to break where appropriate

```
…
            if (Debug.infoOn()) Debug.logInfo("[SOAPClientEngine.invoke} : Parameter: " +
p.name + " (" + p.mode + ") - " + i, module);
…
```

The final three code lines could have been broken before method calls, especially the first one where the method name itself is 40 characters long:

```
…
            ConfigurationContext configContext =
ConfigurationContextFactory.createConfigurationContextFromFileSystem(axis2RepoLocation,
axis2XmlFileLocation);
…
            XMLStreamReader reader = XMLInputFactory.newInstance().createXMLStreamReader(new
StringReader(xmlParameters));
…
            results = UtilGenerics.cast(SoapSerializer.deserialize(respOMElement.toString(),
delegator));
…
```

3.4 Comments

18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

While the first methods are commented out with a `@see` annotation redirecting to the corresponding comments section of the implemented interface, the class main computational method, `serviceInvoker`, has altogether two lines of comments, providing no additional information to understand the code other than the code itself.

3.5 Inizialization and Declarations

33. Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces '{' and '}'). The exception is a variable can be declared in a for loop.

Declarations are very sparse throughout `serviceInvoker` code. For example, simple initial declarations without any sort of computation are made just before the code blocks where the object is actually used:

```
…
        Map<String, Object> parameterMap = new HashMap<String, Object>();
        for (ModelParam p: inModelParamList) {
…
```

```
…
        OMElement parameterSer = null;

        try {
…


…
        Map<String, Object> results = null;
        try {
…
```

This is of course understandable from a logical point of view, but it makes the actual computation implementation be less smoothly readable and sink inside many not well-organized code lines.

3.6 Output Format

41. Check that displayed output is free of spelling and grammatical errors.

Clearly not a strictly spelling or grammatical related error, but

```
…
            if (Debug.infoOn()) Debug.logInfo("[SOAPClientEngine.invoke} : Parameter: " +
p.name + " (" + p.mode + ") - " + i, module);
…
```

has the class invocation surrounded by different brace styles, while in

```
…
        if (Debug.infoOn()) Debug.logInfo("[SOAPClientEngine.invoke] : Parameter length - " +
inModelParamList.size(), module);
…
```

square braces are used.

42. Check that error messages are comprehensive and provide guidance as to how to correct the problem.

This engine peculiarity is that it overrides the method `runSync` calling its own main computational method `serviceInvoker`, whose returned Map `result` makes a `GenericServiceException("Service did not return expected result")` be thrown when null:

```
…
    @Override
    public Map<String, Object> runSync(String localName, ModelService modelService,
Map<String, Object> context) throws GenericServiceException {
        Map<String, Object> result = serviceInvoker(modelService, context);

        if (result == null)
            throw new GenericServiceException("Service did not return expected result");
        return result;
    }
…
```

Problem is that inside `serviceInvoker` the Map meant to be returned is computed within a try block packed with different types of computation, and a simple `Exception e` is caught:

```
…
        Map<String, Object> results = null;
        try {
            OMFactory factory = OMAbstractFactory.getOMFactory();
            OMElement payload = factory.createOMElement(serviceName);
            payload.addChild(parameterSer.getFirstElement());
            OMElement respOMElement = client.sendReceive(payload);
            client.cleanupTransport();
            results = UtilGenerics.cast(SoapSerializer.deserialize(respOMElement.toString(),
delegator));
        } catch (Exception e) {
            Debug.logError(e, module);
        }
        return results;
…
```

Tracking down the possible necessity to catch an exception yields different causes, namely

— calling `parameterSer` method when it is null due to an exception caught in the following earlier code lines

```
…
        OMElement parameterSer = null;

        try {
…
        } catch (Exception e) {
            Debug.logError(e, module);
        }
…
```

— calling `client` method when it is null due to an exception caught in the following earlier code lines

```
…
        ServiceClient client = null;
…
        try {
            ConfigurationContext configContext =
ConfigurationContextFactory.createConfigurationContextFromFileSystem(axis2RepoLocation,
axis2XmlFileLocation);
            client = new ServiceClient(configContext, null);
…
            client.setOptions(options);
        }
…
```

Of course several `Debug.logError(e, module)` calls give the possibility to track what fails inside the log, but maybe the cause itself could have been provided more promptly and easily readable as an output string.

3.7 Exceptions

53. Check that the appropriate action are taken for each catch block.

As previously highlighted, a `GenericServiceException("Service did not return expected result")` is thrown when `serviceInvoker` returns a null Map due to whatever reason triggered this inside `serviceInvoker` (see Output Format paragraph).

This seems to be a very loose exceptions handling and not so convenient in terms of error traceability, but of course with this single class view and without deep knowledge about the Apache project as a whole, it appears very difficult to determine whether this process yields to coherent or inconsistent results.

4. Any other problem you have highlighted

I don’t really see at the very beginning of the `serviceInvoker` method the use of computing `delegator` right away before checking whether a `GenericServiceException` is immediately thrown:

```
…
    private Map<String, Object> serviceInvoker(ModelService modelService, Map<String, Object>
context) throws GenericServiceException {
        Delegator delegator = dispatcher.getDelegator();
        if (modelService.location == null || modelService.invoke == null)
            throw new GenericServiceException("Cannot locate service to invoke");
…
```

I would say that this choice could cause a slowdown in some particular situation, but of course at this level of analysis the issue cannot be highlighted along with its check priority.

5. Effort

-	6 January 2016:	**2 h**
-	8 January 2016:	**1,5 h**
-	12 January 2016:	**0,3 h**
-	19 January 2016:	**2,5 h**
-	23 January 2016:	**1,7 h**
-	25 January 2016:	**1,2 h**
-	26 January 2016:	**2,1 h**
-	28 January 2016:	**2 h**
-	29 January 2016:	**1 h**
		14,3 h

6. Changelog

- v1.0