Code Inspection document - v1.0

Gianpaolo Branca Luca Butera Andrea Cini

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to analyze two assigned classes which are part of a specific release of the Apache OFBiz Project, an open source software dedicated to the automation of enterprise processes. The analysis method that will be used is called Code Inspection and consists in a series of systematic verifications of the quality of the code based on a checklist. The aim of this analysis is two sided, first it helps the improvement of the code, providing to the developers a set of probable imperfections or errors in the code to be verified; but on the other side allows the inspector to deeply analyze the code becoming more and more skilled in finding possible mistakes even in the case he's the coder.

1.2 Scope

Apache OFBiz is implemented using different standards, mainly Java, JEE and XML. The version assigned to us for inspection has been downloaded from the provided mirror and since the project was developed in Eclipse we used it to further lookup some points on the checklist. We also analyzed the project with Sonarqube.

2 Classes

2.1 PersistedServiceJob

Namespace: org.apache.ofbiz.service.job
Extends: GenericServiceJob
Implements: N/A

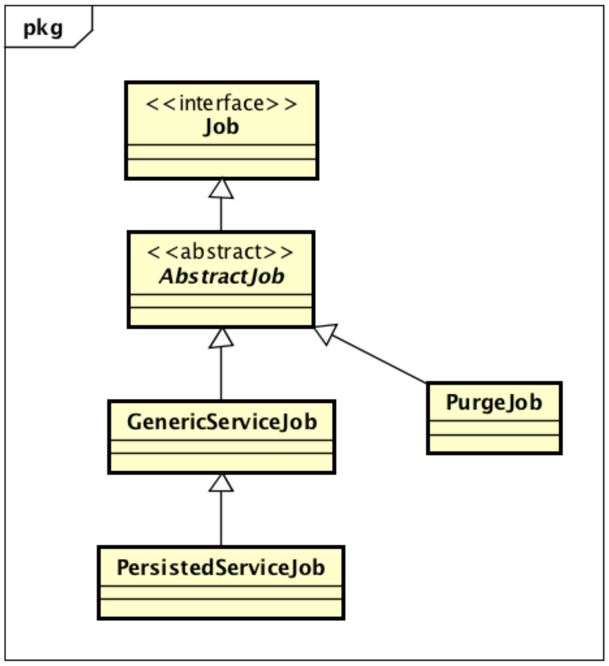
2.2 JobPoller

Namespace: org.apache.ofbiz.service.job.JobPoller
Extends: N/A
Implements: org.apache.ofbiz.service.config.ServiceConfigListener

3 Functional role

Note: The ofbiz project is very poorly documented and many aspect are not self explained.
For the usages sections, importing the project with eclipse we managed to scan the entire project for the usages of these classes.

3.1 PersistedServiceJob



Role

According to the Javadoc a **PersistedServiceJob** is A Job that is backed by the entity engine, and his data are stored in the JobSandbox entity.
The JobSandbox entity is an instance of GenericValue called JobValue, with a bad naming. Note that the class **JobSandbox** does not exist in the entire project.

PersistedServiceJob extends **GenericServiceJob**, an async-service job and the main realization of a Job. Itself extend **AbstractServiceJob**, and this last implements the interface **Job**.

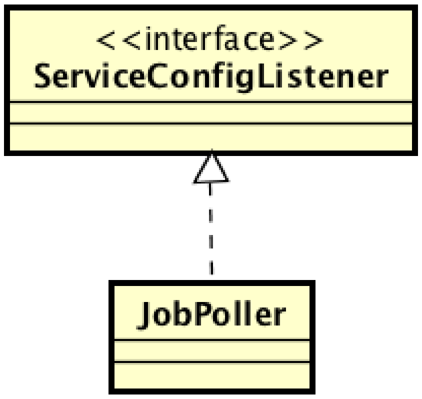
According to the JavaDoc:
A job starts out in the created state. When the job is queued for execution, it transitions to the queued state. While the job is executing it is in the running state. When the job execution ends, it transitions to the finished or failed state - depending on the outcome of the task that was performed.

A PersistedServiceJob works as a GenericServiceJob, it can be queued, dequeued, executed and finish, but every overridden method also store the time, the status and the result of the job in the **JobSandBox**. It can also fail, but differently to its superclass it can retry a certain number of time, decided by the JobSandBox.

Usages

It is used only once in the JobManager class, in the *poll(int)* method, at line 225. According to the documentation:
this method scans the JobSandbox entity and returns a list of jobs that are due to run. Returns an empty list if there are no jobs due to run. This method is called by the **JobPoller** polling thread.
A PersistedServiceJob is created whenever there is a job to run, added to the poll and returned to the JobPoller. The JobPoller class is explained in the next section.

3.2 JobPoller



Role

The JobPoller is a singleton class created to handle the execution of the Jobs contained into different JobManagers creating a queue that balances the Jobs execution ordering to ensure that the execution time is well spread among the JobManagers.

As one can see, the JobPoller relies on a ThreadPoolExecutor which is properly configured by taking information about the service configuration parameters from the ServiceConfigUtil class, this is achieved using the *createThreadPoolExecutor* method. The JobPoller, itself, contains an instance of an extension of the Thread class, which is the JobManagerPoller; this class is the main thread that manages the queueing of the Jobs, relying on the ThreadPoolExecutor. The JobPoller also offers the access to informations about the Jobs he handles with the *getPoolState* method, and also about the waiting time of the poll, with the *pollWaitTime* method. Along with these, the JobPoller contains a method to register a JobManager to the JobPoller, which of course is *registerJobManager*, this method clarifies the fact that the JobPoller operates as a “subscription service” and handles automatically only the execution of the Jobs related to subscripted JobManagers. Furthermore one method allows the direct insertion of a Job into the ThreadPoolExecutor queue, the said method is *queueNow*. In the end there’s a method to enable the JobPoller and one to stop it.

Taken into account all these informations, it should result clear that the role of this class is the one stated at the beginning of this paragraph. The services offered by the JobPoller class, and the tasks that he carries out, clearly identify it as a manager created with the purpose of storing and organizing the Jobs and their execution along with relevant informations that can be retrieved upon necessity.

Usages

The JobPoller class is used only by the JobManager class, to be specific in four different points in the code of the class, at lines 98, 109, 136, 367, following this order we’ll explain the purpose of each usage case.

The first usage is in the *getInstance* method which is basically used to retrieve a JobManager associated to a specific **Delegator** or a new one if there’s no existing. In the latter case a boolean value defines if the *registerJobManager* method of the JobPoller must be called to perform the subscription of the newly created JobManager.

The second usage case is pretty straightforward, happens inside the *shutDown* method, which is called upon the closing of OFBiz. This method calls the *getInstance* method of the JobPoller and sequentially its *stop* method which correctly shuts the JobPoller before proceeding to close the rest of the system.

The third usage case is also a pretty simple one, in fact it just involves the *get-PoolState* method which, after getting the JobPoller instance, calls its homonym method to retrieve the pool state Map to use as return value.

The fourth and last interaction instead is fairly interesting since it underlies one of the principal methods of the JobManager. The *runJob(Job)* method is, in fact, used to directly insert a Job into execution queue, and relies on the *queueNow(Job)* method of the JobPoller, which functionality has been already stated in the related Role paragraph.

4 Issues list found by applying the checklist

4.1 PersistedServiceJob

Naming convention

- Constant variable *module* should be all uppercase
- Method *longValue* should start with a verb
- Method *verboseOn* used at line 116 should start with a verb (hint: *isVerbose*)
- Method *nowTimestamp* used at line 137 should start with a verb (hint: *getNowTimestamp*)
- Method *infoOn* used at line 224 should start with a verb and has an ambiguous name
- Variable *next* at line 179 should have a more meaningful name
- Variable *next* at line 251 should have a more meaningful name
- Variable *jobValue* at line 68 should be named *jobSandBox* or similar, for consistency with comments, log outputs and javadoc.

Indention

- Everything is ok

Braces

- Single statement *if* without braces at line 187
- Single statement *if* without braces at line 191
- Single statement *if* without braces at line 212

File organization

- Line 83 can be rewritten with an if/else statement to not exceed 80 columns
- Line 114 exceed 120 columns
- Line 142 exceed 120 columns
- Line 153 exceed 120 columns
- Line 159 exceed 120 columns

- Line 187 exceed 120 columns
- Line 249 exceed 120 columns
- Line 259 exceed 120 columns
- Line 288 exceed 120 columns
- Line 323 exceed 120 columns

Wrapping lines

- Everything ok

Comments

- Everything ok

Java source file

- Everything ok

Package and import statements

- Everything ok

Class and interface declaration

- Method *deQueue* should be grouped with *queue*
- Method *init* should be grouped with the other protected overridden methods

Initialization and declaration

- *cancelTime* not declared at the beginning of block at line 104
- *startTime* not declared at the beginning of block at line 105
- *maxRecurrenceCount* not declared at the beginning of block at line 148
- *currentRecurrenceCount* not declared at the beginning of block at line 149
- *expr* not declared at the beginning of block at line 150
- *recurrence* not declared at the beginning of block at line 151
- *next* not declared at the beginning of block at line 179
- *newJob* not declared at the beginning of block at line 197
- *jobResult* not declared at the beginning of block at line 222
- *next* not declared at the beginning of block at line 251
- *count* not declared at the beginning of block at line 321

Method calls

- Everything ok

Arrays

- Everything ok, no arrays

Object comparison

- Everything ok

Output format

- Everything ok

Computation, Comparisons and Assignments

- Everything ok

Exceptions

- Everything ok

Flow of Control

- Everything ok, no switches and no loops

Files

- Everything ok, no files

4.2 JobPoller

Naming convention

- Constant variable *module* should be all uppercase
- Method *pollEnabled* declared at line 154 should start with a verb
- Method *onServiceConfigChange* declared at line 145 should start with a verb

- Method *remainingCapacity* used at line 217 should start with a verb (hint: *getRemainingCapacity*)
- Method *values* used at line 220 should start with a verb (hint: *getValues*)
- Method *infoOn* used at line 224 should start with a verb and has an ambiguous name
- Method *iterator* used at line 228 should start with a verb
- Variable *created* at line 51 should have a more meaningful name

Indention

- Line 67 starts with a mismatching number of spaces
- Line 71 starts with a mismatching number of spaces

Braces

- Single statement *if* without braces at line 224

File organization

- Line 52 exceed 120 columns
- Line 67 exceed 120 columns
- Line 69 exceed 120 columns
- Line 71 exceed 120 columns
- Line 80 exceed 120 columns

Wrapping lines

- Everything ok

Comments

- Everything ok

Java source file

- Everything ok

Package and import statements

- Everything ok

Class and interface declaration

- Variable *jobManagerPollerThread* declared ad line 98 should be declared after the static variables
- Constructor *JobPoller* at line 100 should be declared after the variables.

Initialization and declaration

- *serviceName* not declared at the beginning of block at line 131
- *queueCandidates* not declared at the beginning of block at line 231
- *addingJobs* not declared at the beginning of block at line 232

Method calls

- Everything ok

Arrays

- Everything ok

Object comparison

- At line 213 "!=" is used instead of **!LEFT_PART.equals(RIGHT_PART)**.

Output format

- Everything ok

Computation, Comparisons and Assignments

- At line 191 the catch clause catches a general Exception instead of an InvalidJobException.

Exceptions

- Catch block at line 172 should log the exception.
- Catch block at line 192 contains no message for a generic Exception.
- Catch block at line 254 should log the exception.

Flow of Control

- Everything ok, no switches

Files

- Everything ok, no files

5 Other problems

GenericValue class should have a more meaningful name, since it is used often in the inspected code for non trivial operation. According to the Javadoc it "Handles persistence for any defined entity". Its name should reflect the offered functionalities.

We managed to analyze the project with Sonarqube, founding the following extra issues:

5.1 PersistedServiceJob

- "jobID" literal duplicated 4 times, should be replaced by a constant.
- "currentRecurrenceCount" literal duplicated 3 times, should be replaced by a constant.
- "currentRetryCount" literal duplicated 3 times, should be replaced by a constant.
- "runByInstanceID" literal duplicated 4 times, should be replaced by a constant.
- "statusId" literal duplicated 7 times, should be replaced by a constant.
- "startDateTime" literal duplicated 4 times, should be replaced by a constant.
- "parentJobId" literal duplicated 4 times, should be replaced by a constant.
- "startTime" variable at line 105 should be renamed since hides the field declared at line 69.
- Method init should be refactored since it have too high cognitive and cyclomatic complexity.
- "Long" constructor at line 206 should be removed.
- "Long" constructor at line 208 should be removed.
- Useless Assignment at line 222
- *if* statement at line 309 always evaluated to false.

5.2 JobPoller

- *pollWaitTime* method should be moved into JobManagerPoller class
- Useless Assignment at line 125
- Method *newThread* at line 200 should have the @Override notation
- Method *run* at line 210 should have the @Override notation, and it should be refactored since it is too complex (it contains several if/for/while/switch/try statements nested).
- Nested *try* block at line 245 should be extracted into a separate method

6 Effort spent

Every member of the group spent about 15 hours working on this assignment