POLITECNICO
MILANO 1863

Politecnico di Milano
A.A. 2016–2017
Software Engineering 2: "PowerEnJoy"
Code Inspection Document

Pietro Ferretti, Nicole Gervasoni, Danilo Labanca

February 5, 2017

Contents

1 Assigned Class

apache-ofbiz-16.11.01/framework/service/
src/main/java/org/apache/ofbiz/service/job/JobManager.java

2 Functional Role

```
1
2  /**
3  * Job manager. The job manager queues and manages jobs. Client
       code can queue a job to be run immediately
4  * by calling the runJob({@link #runJob(Job)}) method, or schedule
       a job to be run later by calling the
5  * {@link #schedule(String, String, String, Map, long, int, int,
       int, long, int)} method.
6  * Scheduled jobs are persisted in the JobSandbox entity.
7  * <p>A scheduled job's start time is an approximation - the
       actual start time will depend
8  * on the job manager/job poller configuration (poll interval) and
       the load on the server.
9  * Scheduled jobs might be rescheduled if the server is busy.
       Therefore, applications
10 * requiring a precise job start time should use a different
       mechanism to schedule the job.</p>
11 */
```

3 List of Issues

3.1 Naming Conventions

1. Meaningful Names

sono tutti buoni

2. One-character variables Ok. There are no one-character variables.

3. Class names Ok. Every class name is in mixed case and properly capitalized.

4. Interface names Ok. No interfaces are declared. (se ce ne sono) every interface used by the code is in mixed case and properly capitalized.

5. Method names Ok. Every method name is a verb. Every method name is camelCase and properly capitalized.

6. Class variables Ok. Every class variable is in mixed case and properly capitalized.

7. Constants NO. module and istanceId are immutable, so they can be considered constant. They should be capitalized. registeredManagers is fine because it's mutable

3.2 Indentation

8. Number of spaces Ok. The code is consistently indented with 4 spaces.

9. No tabs for indentation Ok. No tabs are used to indent the code.

3.3 Braces

10. Consistent bracing style Ok. The code is consistently braced following the *"Kernighan and Ritchie"* style.

11. One-line statements bracing NO "if" riga 326, 351, 354

3.4 File Organization

12. Blank lines as separation Ok. Blank lines are present between each method, around imports and variable declarations. Most of the methods also begin with a Javadoc.

13. Where practical, line length under 80 characters NOPE righe 73, 74, 89, 126, 147, 150, 154, 156, 161, 182, 186-190, 195, 198, 201, A great number of lines exceed 80 characters

14. Line length always under 120 characters NEPPURE righe 74, 186, 198, 217, 221, 222, 261-264, 273, 311, 315, 317, 387, 409, 429, 453, 498, 543, 560, 561 le dichiarazioni dei metodi sono lunghissime e wrappate poco

3.5 Wrapping Lines

15. Line breaks after commas and operators NO riga 152, la virgola dovrebbe stare sopra

16. Higher-level breaks are used Ok. Non ci sono line break con operatori

17. Statements are aligned to previous ones Ok. Sì, per tutti

3.6 Comments

18. Comments use The method

```
1     public synchronized void reloadCrashedJobs()
```

on line 305 is not commented and so it isn't easy to understand.
Per il resto tutto appoito

19. Commented out code There aren't lines of code commented in the source code.

3.7 Java Source Files

20. Single public class or interface Ok. Job manager is the only public class declared in the file. There are no other classes.

21. The public class is the first class in the file Ok. Job manager is the only public class declared in the file. There are no other classes.

22. External program interfaces are consistent with the Javadoc Ok abbiamo vari metodi pubblici: getter: - getDelegator - getDispatcher - getInstance - getPoolState poi altre robe - isAvailable - reloadCrashedJobs - runJob - schedule di tutti i tipi
la Javadoc parla di runJob e schedule

23. The Javadoc is complete NO.

- No javadoc for 'module'! line 71
- No javadoc for 'instanceId'! line 71
- No javadoc for reloadCrashedJob!! line 304
- Missing @return tag on getInstance, line 88
- Missing @return tag on getDelegator, line 119
- Missing @return tag on getDispatcher, line 124
- Missing @param tag for 'limit' on poll, line 174
- Missing @return tag on poll, line 174
- Missing @param tag for 'job' on runJob, line 363
- Missing @throws tag for 'JobManagerException' on runJob, line 363
- Missing @throws tag for 'JobManagerException' on schedule, line 386, 408, 428, 453, 469, 498, 543

assertIsRunning, getRunPools sono private quindi non hanno necessariamente bisogno di javadoc

3.8 Package and Import Statements

24. Package statements are first, import statements second Ok. One package statements. All import statements immediately follow.

3.9 Class and Interface Declarations

25. The class declarations should follow a specific order - javadoc ok - class declaration ok - altri commenti / - static variables ok - public ok - private ok - normal variables - constructors - methods

no, abbiamo variabili statiche, poi un po' di metodi statici, poi variabili normali, poi costruttori (getInstance è un costruttore), setter e getter poi un metodo statico (ma private!!)

26. Methods are grouped by functionality Ok
assertIsRunning getInstance shutDown
getDelegator getDispatcher getPoolState
isAvailable getRunPools pool reloadCrashedJobs runJob schedule

27. The code is free of duplicates, long methods, big classes, breaking encapsulation, and coupling and cohesion are adequate small class duplicates? no short methods no breaking encapsulation

low/loose coupling -> ci sono un sacco di delegator e dispatcher high cohesion -> tutti i metodi servono a runnare/queuare jobs

3.10 Initialization and Declarations

28. Visibility All variables and class members are of the correct type and have the proper visibility. In line 305 the method

```
1 public synchronized void reloadCrashedJobs()
```

could be stated as protected.

29. Proper scope. OK. All variables are declared in the proper scope

30. New objects. OK. Each time a new object is desired the proper constructor is called

31. All object references are initialized before use. OK

32. Variables initialization. OK. All variables are initialized where they are declared, unless dependent upon a computation.

33. Declarations. OK. Each declaration appear at the beginning of blocks.

3.11 Method Calls

34. Correct orders parameters Sembra tutto bene

35. The called method is the right method Sembra di si

36. The returned value from the method is used properly Me pare de si

3.12 Arrays

No off-by-one errors in array indexing Ok. The only indexing is with foreach, no off-by-one errors.

No out-of-bounds indexes Ok. No number indexing.

Constructors are called when a new array item is desired Ok. quali nuovi array? non ce ne sono

3.13 Object Comparison

Objects are compared with equals Ok. There are no object comparisons.

3.14 Output Format

Displayed output is free of spelling and grammatical errors riga 156: Debug.logWarning(e, "Exception thrown while check lock on JobManager : " + instanceId, module); dovrebbe essere "while checking"

riga 182: Debug.logWarning("Unable to locate DispatchContext object; not running job!", module); dovrebbe essere "job:", come negli altri log di debug

Error messages are comprehensive and useful si

Output is formatted correctly in terms of line breaks and spacing No line breaks in outputs Some debug outputs don't have a trailing space

3.15 Computation, Comparisons and Assignments

44."Brutish programming". the avoids OK. The implementation avoids brute force solutions; the code is simple and concise.

45. Operator precedence and parenthesizing. OK. Computation/evaluation of operator precedence and parentheses is in the proper order.

46. The liberal use of parenthesis is used to avoid operator precedence problems. OK.

47. All denominators of a division are prevented from being zero. OK. There are no division.

48. Integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding. OK. Integer arithmetic is used only to increment variable.

49. Comparison and Boolean operators are correct. OK.

50. Throw-catch expressions. OK. The error condition is always legitimate

51. The code is free of any implicit type conversions. OK.

3.16 Exceptions

52. Relevant exceptions are caught. OK.

53. The appropriate action is taken for each catch block. OK. There are two general

```
1 catch (Throwable t)
```

in order to guarantee a working jobPoller even when a database connection is not available.

3.17 Flow of Control

All switch cases are addressed with a break Ok, no switch statements.

All switch statements have a default branch Ok, no switch statements.

All loops are correctly formed, with appropriate initialization, increments and termination expressions Ok. All for loops are foreach, everything is fine. The while loop at line 219: GenericValue jobValue = jobsIterator.next(); while (jobValue != null) jobValue = jobIterator.next(); tutto ok, l'iteratore va avanti finch non finiscono i valori, poi esce dal while while a riga 275 uguale
a posto

3.18 Files

The JobManager class does not have to handle file.

4 Other Problems

nah

5 Effort Spent

- Pietro Ferretti: hours of work
- Nicole Gervasoni: hours of work
- Danilo Labanca: hours of work

6 Revisions

6.1 Changelog

- CID v1.0, published on February 5, 2017