Code inspection

Figure 1: Politecnico di Milano

Version 1.0

Relasese date : 05/01/2016

- Claudio Cardinale (mat. 849760)
- Gilles Dejaegere (mat. 853950)
- Massimo Dragano (mat. 775392)

Contents

2. Gilles Dejaegere
3. Massimo Dragano

1. Introduction

1.1. Purpose

The purpose of this document is to show all the problems found during the inspection of a small amount of code of a specific version of Glassfish. The process of inspecting source code has two main purposes. The first and most obvious one is to enhance the quality of the code and eventually to identify the remaining bugs. The second purpose is to improve the coding skills of the team. The inspectors improve themselves by analysing the code made by others and eventually discovering coding methods that they did not know. The original authors of the code receive a list of possible mistakes that they could have done, which of course help them improving themselves.

Each group of the project has different methods assigned of a specific version of Glassfish. We have to analyse these methods by checking that they are in agreement with every point of a given checklist. We also have to find other problems, then we have to report the problems found in this document.

1.2. Scope

Glassfish is the official implementation of JEE. It is an open source project that uses svn as version system, in fact we used it to retrieve a specific version of Glassfish 4.1.1 (of 16 Oct 2015 05:11:30 UTC).
This version is the version required by the assignment since we have been assigned some of its methods to check.
Glassfish is a maven project, in fact we imported the pom file into intellij IDEA and we used it, sonar and regex to verify some checks of the checklist.

1.3. Definitions, acronyms, abbreviations

- JEE: Java enterprise edition
- SVN: apache subversion, it is a version controller system, the successor of CVS
- CVS: Concurrent versions system, a former version controller system
- Apache: open source company famous for its web server called apache
- Apache tomacat catalina: It is an open source web server developed by apache foundation (not oracle) for and only for servlets.
- Context: Contextual Information: it is a design pattern that consists in storing the main information inside one object and this object is used to pass everything
- Servlet: A java program that runs on a dedicated web server that is able to elaborate http requests and reply to them.

- MIME type: Multipurpose Internet Mail Extensions type. These are two-parts identifiers used to identify formats of content transmitted on the web.
- K&R style: Indentation style named after Kernighan and Ritchie, who used this style in their book "The C Programming Language".
- Regex: regular expression, it is a finite automata used to define a search pattern
- Sonar: open-source product used to improve code quality via defined metrics.

1.4. Reference documents

- Assignment document: Code inspection.pdf
- Glassfish javadoc of this version: http://glassfish.pompel.me/
- Methods assigned to each group: http://assignment.pompel.me/

1.5. Document structure

- **Introduction:** this section introduces the inspection document. It contains a justification of its utility and indications on which parts are covered in this document.
- **Classes:** this section describes the classes and the methods that have been inspected.
- **Functional role:** this section describes the functional role of the class which the methods assigned belong to.
- **Issues list found by applying the checklist:** this section describes the issues found applying the checklist given.
- **Other problems:** this section describes other problems found that are not strictly related to the checklist.

2. Classes

All methods assigned to us belong to the same class.

2.1. StandardContext

Namespace: org.apache.catalina.core
Extends: ContainerBase
Implements: Context, ServletContext

Methods

- Line 5457 : contextListenerStop( )
- Line 5509 : eventListenerStop( )
- Line 5537 : mergeParameters( )
- Line 5564 : resourcesStart( )
- Line 5597 : alternateResourcesStart( )
- Line 5635 : resourcesStop( )
- Line 5662 : alternateResourcesStop( )
- Line 5708 : loadOnStartup(Container children [ ])

3. Functional role

3.1. Javadoc

This class is the standard implementation of the *Context* interface.
According to the javadoc it is:

> A **Context** is a Container that represents a servlet context, and therefore an individual web application, in the Catalina servlet engine. It is therefore useful in almost every deployment of Catalina (even if a Connector attached to a web server (such as Apache) uses the web server's facilities to identify the appropriate Wrapper to handle this request). It also provides a convenient mechanism to use Interceptors that see every request processed by this particular web application.
> The parent Container attached to a Context is generally a Host, but may be some other implementation, or may be omitted if it is not necessary.
> The child containers attached to a Context are generally implementations of Wrapper (representing individual servlet definitions).

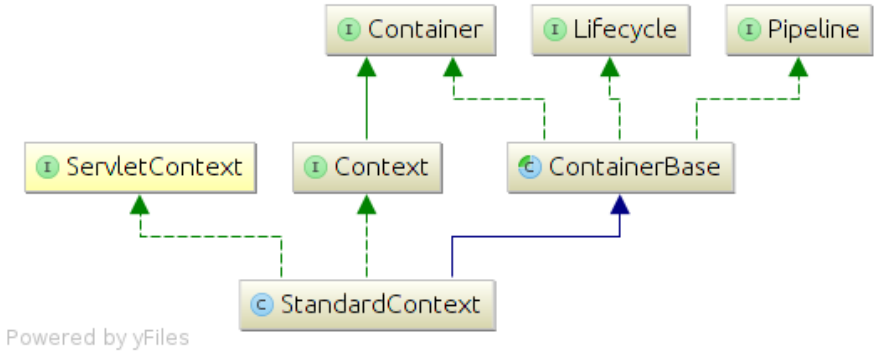

Figure 2: inheritance diagram

It extends *ContainerBase* that, according to javadoc, is:

> A **Container** is an object that can execute requests received from a client, and return responses based on those requests. A Container may optionally support a pipeline of valves that process the requests in an order configured at runtime, by implementing the **Pipeline** interface as well.

And implements *ServletContext* that is a standard *javax* interface that defines the basic methods to build a context for Servlets such as *addServlet*, *createListener* and so on.
According to the javadoc it is:

> Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.
> There is one context per "web application" per Java Virtual Machine. (A "web application" is a collection of servlets and content installed under a specific subset of the server's URL namespace such as `/catalog` and possibly installed via a `.war` file.)
> In the case of a web application marked as "distributed" in its deployment descriptor, there will be one context instance for each virtual machine. In this situation, the context cannot be used as a location to share global information (because the information will not be truly global). Use an external resource like a database instead.

3.2. Usages

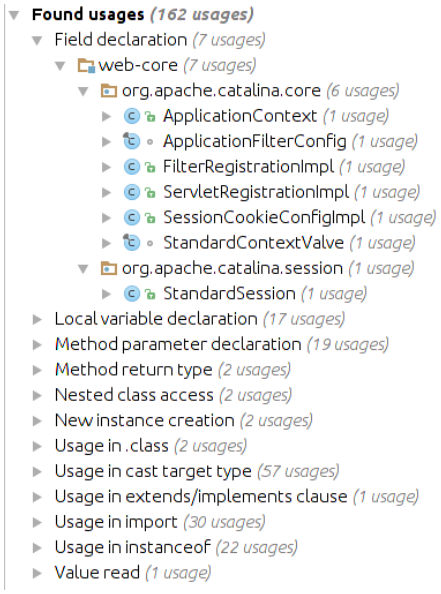

Figure 3: Usages

It is used in a lot of classes. It is particularly used as private property in catalina core classes.
For example we see that it is used by *ApplicationContext* that uses it to add

everything, such as new servlets (*addServlet* on line *672* of *ApplicationContext*). **N.B.** there is only one instance of *StandardContext* inside *ApplicationContext*: 'one context per "web application"'

3.3. Role

Usages and javadoc suggest us that this class is very important because it can be seen as the standard "manager" of apache tomacat catalina (that is a servlets server). In fact this class belongs to an host implementation (that uses it to manage all features inserted at high level) and it contains the servlets.
In fact in the *context pattern* (Contextual Information) we have a main class context containing the main information, in this case it contains the servlets, and it refers or allows to modify the requests or responses via interceptors (in fact it extends *ContainerBase*).
The *context pattern* is very useful where there is a great amount of data. For example it is used in android applications to interact with the user. With this pattern you can manage a lot of features. From the usage side, features can be inserted dynamically via a single object, and from the creation side you can choose where to use these features simply by choosing the appropriate context (the appropriate web application in this case). All data must pass via the context.

4. Issues list found by applying the checklist

4.1. NamingConventions

- from class `RestrictedServletContextListener`
  - method `contextInitialized` should start with a verb (hint: `onContextInitialized`)
  - method `contextDestroyed` should start with a verb (hint: `onContextDestroyed`)
- method `backgroundProcess` should start with a verb (hint: `runBackgroundProcess`)
- field `count` is not meaningful (hint: `backgroundProcessCounter`)
- method `contextListenerStart` should start with a verb (hint: `notifyContextStarted`)
- method `contextListenerStop` should start with a verb (hint: `stopContextListening`)
- method `create` is not clear and it looks like a simple alias of the `init` method
- method `create` is not used (hint: delete it)
- method `engineBase` should start with a verb (hint: `getEngineBase`)
- method `eventListenerStop` should start with a verb (hint: `stopEventListening`)
- method `eventListenerStop` always return true (hint: change to `void`)
- method `filterStart` should start with a verb (hint: `startFilters`)
- method `filterStop` should start with a verb (hint: `stopFilters`)
- method `managerStart` should start with a verb (hint: `startManager`)
- method `managerStop` should start with a verb (hint: `stopManager`)
- method `resourcesStart` should start with a verb (hint: `allocateResources`)
- method `resourcesStop` should start with a verb (hint: `freeResources`)
- method `restrictedSetPipeline` should start with a verb (hint: `setPipeline`)
- method `restrictedSetPipeline` should be made accessible only to certain packages (hint: declare it as `protected` and give a `friendly` accessor from the child class)
- method `sessionCreatedEvent` should start with a verb (hint: `onSessionCreatedEvent`)
- method `sessionDestroyedEvent` should start with a verb (hint: `onSessionDestroyedEvent`)
- method `sessionRejectedEvent` should start with a verb (hint: `onSessionRejectedEvent`)

- method `sessionExpiredEvent` should start with a verb (hint: `onSessionExpiredEvent`)
- method `sessionPersistedStartEvent` should start with a verb (hint: `onSessionPersistedStartEvent`)
- method `sessionPersistedEndEvent` should start with a verb (hint: `onSessionPersistedEndEvent`)
- method `sessionActivatedStartEvent` should start with a verb (hint: `onSessionActivatedStartEvent`)
- method `sessionActivatedEndEvent` should start with a verb (hint: `onSessionActivatedEndEvent`)
- method `sessionPassivatedStartEvent` should start with a verb (hint: `onSessionPassivatedStartEvent`)
- method `sessionPassivatedEndEvent` should start with a verb (hint: `onSessionPassivatedEndEvent`)
- method `sessionListenerStop` should start with a verb (hint: `stopSessionListening`)

4.2. Indention

- line `5479` start with a mismatching number of spaces
- line `5482` start with a mismatching number of spaces
- line `5486` start with a mismatching number of spaces
- line `5488` start with a mismatching number of spaces
- line `5625` start with a mismatching number of spaces

4.3. Braces

- single statement `if` without braces at line `5546`

N.B.: K&R style is used.

4.4. File Organization

- line `5487` can be easily rewritten not to exceed 80 columns.
- line `5574` can be easily rewritten not to exceed 80 columns.
- line `5576` can be easily rewritten not to exceed 80 columns.
- line `5582` can be easily rewritten not to exceed 80 columns.
- line `5613` can be easily rewritten not to exceed 80 columns.
- line `5618` can be easily rewritten not to exceed 80 columns.
- line `5621` can be easily rewritten not to exceed 80 columns.
- line `5624` can be easily rewritten not to exceed 80 columns.

- line 5680 can be easily rewritten not to exceed 80 columns.
- line 5734 can be easily rewritten not to exceed 80 columns.
- line 5735 can be easily rewritten not to exceed 80 columns.

N.B.: there are no lines that exceed 120 columns.

4.5. Wrapping Lines

Everything ok

4.6. Comments

- commented code without any visible reason from line 5704 to 5706

4.7. Java Source Files

Everything ok

4.8. Package and Import Statements

Everything ok

4.9. Class and Interface Declarations

Everything ok

4.10. Initialization and Declarations

- can be private at line 5564
- can be private at line 5597
- can be private at line 5635
- can be private at line 5662
- can be private at line 5708
- `event` not declared at beginning of the block at line 5465
- `len` not declared at beginning of the block at line 5467
- `msg` not declared at beginning of the block at line 5487
- `iter` not declared at beginning of the block at line 5514
- `sc` not declared at beginning of the block at line 5553
- `env` not declared at beginning of the block at line 5603

- `alternateWebappResources` not declared at beginning of the block at line `5680`
- `key` not declared at beginning of the block at line `5719`
- `list` not declared at beginning of the block at line `5720`

4.11. Method Calls

- return value of method `contextListenerStop` is never used (hint: change to `void`)
- return value of method `eventListenerStop` is never used (hint: change to `void`)
- return value of method `resourcesStart` is never used (hint: change to `void`)
- return value of method `resourcesStop` is never used (hint: change to `void`)
- return value of method `alternateResourcesStop` is never used (hint: change to `void`)

4.12. Arrays

Everything ok

4.13. Object Comparison

Everything ok

4.14. Output Format

Everything ok

4.15. Computation, Comparisons and Assignments

- brutish programming from line `5467` to `5471` (replace with `Collections.reverse` on a list copy)
- brutish programming from line `5514` to `5516` (use foreach)
- brutish programming at line `5625` (use varargs intead of Object array)

4.16. Exceptions

Everything ok

4.17. Flow of Control

Everything ok (there are no switches)

4.18. Files

Everything ok, no files

5. Other problems

- loadOnStartup: parameter “children” declared in C-style array `5708`
- addEnvironment: never used
- addResource: never used
- addResourceLink: never used
- getStartupTime: never used
- getTldScanTime: never used
- setTldScanTime: never used
- redundant assignment at line `7498`
- redundant assignment at line `7545`
- redundant assignment at line `7559`
- getState: should return an enum or a class
- field `tldScanTime` is not used
- setCompilerClasspath: never used
- getOriginalDocBase: never used
- isReplaceWelcomeFiles: never used
- setUnloadDelay: never used
- setUnpackWAR: never used
- getCharsetMapperClass: never used
- setCharsetMapperClass: never used
- addResourceParams: never used
- addServletMapping(ServletMap): never used
- findMappingObject: never used
- findMessageDestination: never used
- findMessageDestinations: never used
- findMessageDestinationRef: never used
- findMessageDestinationRefs: never used
- removeMessageDestination: never used
- removeMessageDestinationRef: never used
- managerStart: never used
- managerStop: never used
- getDefaultConfigFile: never used
- getResourceNames: never used
- getResourceLinks: never used
- addEnvironment: never used
- addResource: never used
- addResourceLink: never used
- getStaticResources: never used
- startRecursive: never used
- getStartTimeMillis: never used
- isEventProvider: never used

- isStatisticsProvider: never used
- setCachingAllowed: never used
- setCaseSensitive: never used
- setCaseSensitiveMapping: never used
- setCacheTTL: never used
- setCacheMaxSize: never used
- getAntiJARLocking: never used
- setAntiJARLocking: never used
- MessageFormat is often called with an Object array intead of using vararg.
- some 'if' have a space before '(', but other not, the same thing for other construct like 'catch'
- } catch(Throwable t) { -> Throwable instead the right exception
  - 5619
  - 5648
  - 5483
  - 5580
  - 5675
  - 5685

6. Used tools

- intellij IDEA: JAVA EE IDE
- sonar: useful tools to analyse code from style point of view
- Github: for version controller
- Gedit and ReText: to write MarkDown with spell check

7. Conclusion

This StandarContext.java code has been developed by a team of professionals and qualified programmers. Still, in this document it is possible to see that this code is not perfect. Using a checklist is an efficient, systematic, and rigorous way to point out some remaining bugs and to make the code as coherent as possible. Unfortunately, code inspection is a very time consuming activity and it is not envisageable to apply it totally for large projects like Glassfish but fortunately it is possible for large projects to use tools such as Sonar to perform some of the checkings in an automated way. Moreover, even if using a checklist is a good way to improve the code, this is by no means a guarantee for making a perfect code. Indeed, some of the bugs or possible improvements shown in this document are not related to points of the checklist. For example we can see that some of the methods implemented in the java document are never used in the project. This is not a real bug but it is more a conceptual mistake. Time has been wasted making these methods and disk space is wasted on every machine hosting the project.

8. Hours of work

Claudio Cardinale

- 18/12/15: 1h
- 19/12/15: 5h
- 22/12/15: 30m
- 23/12/15: 5h
- 29/12/15: 4h
- 01/01/16: 2h
- 03/01/16: 30m
- 04/01/16: 30m
- 05/01/16: 1h

Gilles Dejaegere

- 24/12/15: 2h
- 25/12/15: 1h30
- 31/12/15: 2h
- 01/01/16: 1h30
- 02/01/16: 1h
- 04/01/16: 45min
- 05/01/16: 2h

Massimo Dragano

- 19/12/15: 3h
- 23/12/15: 2h
- 28/12/15: 6h
- 29/12/15: 7h
- 02/01/16: 2h
- 03/01/16: 1h
- 05/01/16: 3h