AspectJ 1.8.0 Release Review - 2Q2014

Planned Review Date: [Date]

Communication Channel: aspectj-users@eclipse.org, aspectj-dev@eclipse.org

Andy Clement

Introduction

- AspectJ is a seamless extension to Java that adds the ability to capture cross-cutting concerns
- It adds a few new keywords and constructs (e.g. pointcut, aspect) to the Java language and provides a compiler that understands these extensions
  - The compiler is a modified form of the JDT core compiler
- It also includes a weaver that can be used to apply cross cutting concerns to code that has previously been compiled to bytecode
  - The weaver can be used as an offline post-compile step or as a load-time weaver.

Features

- AspectJ major/minor version numbers have traditionally tracked Java version numbers
  - AspectJ 1.8.0 is the first Java 1.8 version of AspectJ
  - AspectJ takes and modifies the JDT compiler. For 1.8.0 AspectJ has been rebased on the 'Java 8 patch' released alongside Eclipse 4.3.2.
  - Basic 1.8.0 readme:

    http:www.eclipse.org/aspectj/doc/released/README-180.
    - Simply showing ability to use 1.8 constructs in AspectJ code.

Features

- Weaving into Java8 code
  - Required updating to using the asm toolkit v5 as it understands Java8 bytecode (actual version: v5.0.1)
  - Required updating the bcel derivative used in AspectJ to understand Java8 bytecode (e.g. TypeAnnotation attributes)

- Fewer features in AspectJ 8 because resource was spent helping Eclipse itself support Java8
  - Type annotations, lambda serialization

Features

- Weaver upgrade for Java 1.8
  - On the back end the AspectJ weaver has been upgraded to understand the new bytecode changes in Java 1.8
    - It already understood bootstrap methods/invokedynamic since AspectJ 1.7
    - New changes to support included type annotation attributes in the classfiles
    - Only tolerating these features for now, not exploiting them

Non-Code Aspects

- The readmes for each release continue to provide the most up to date documentation, some of the new features discussed in these do need folding into the main documentation.
- All the existing documentation (getting started, reference material, etc) remains valid and relevant to AspectJ 1.8.0
- Moved to git from cvs for 1.7.0 release
  - Ditched some unwanted code/modules in the move

APIs

- Primary API exposed for integration into AJDT
  - recent releases have increased the granularity in the API to enable finer grained interactions between AJ/AJDT → improving incremental compilation

Architectural Issues

- On the front end AspectJ continues to be based on a modified JDT core compiler, there is no real need for additional extensibility in this area
  - However, continuing to maintain a large 'patch' on JDT core does slow down the ability to keep up with Eclipse versions
  - There were concerns as to whether the patching could be done in the same way on ECJ for Java 8 because Java 8 is such a big change, but it appears to be OK
  - Experimenting with different patching approaches to reduce the amount of patch work (using diffs rather than file-by-file compare)

Tool Usability

- For the Eclipse UI, defer to the AJDT project
- As a pure compiler/weaver the project is currently actively (and successfully) used through:
  - Command line batch invocation
  - Loadtime weaving (-javaagent)
  - Maven AspectJ plugin
  - Gradle (no central plugin but a number of users building their own custom plugins pulling in AspectJ)
- The maven plugin does fall behind with supporting new options as it isn't the AspectJ team maintaining it – we may try to get more involved with it

End-of-Life

- AspectJ continues to maintain a high degree of backwards compatibility. Programs compiled with versions back to AspectJ 1.2 will work just fine with the latest AspectJ release
- Nothing is being end-of-lifed/removed in 1.8.0

Bugzilla

- Bugs/Enh opened since 1.7.0: 90
- Bugs/Enh resolved since 1.7.0: 73
- Total bugs/enh open against AJ: 412bugs 205enh
  - No P1 Bugs open
- Bugzilla could still do with a pass to close a number of the minor/niche problems that we just won't get to in the foreseeable future

Standards

- J2SE
  - AspectJ now utilizes generics in its source code
    - Requires Java 1.5 (this is a divergence from JDT core which only requires Java 1.4)
  - Code generated by AspectJ can run on Java 1.1 and later
  - AspectJ 1.8.0 can now cope with compiling Java 1.8 source code or weaving into previously compiled Java 1.8 class files

UI Usability

- Defer to AJDT project for Eclipse UI usability

Schedule

- AspectJ 1.8 builds were made available very early (July 2013) due to requirements from other projects (Spring Framework).
- The most recent AspectJ available included the Kepler SR2 Java8 patch and was released on the **same day** as Java8
- Basic upgrade to Java 1.8 was relatively easy as AspectJ could build upon the work done in JDT core
  - In recent user testing, some issues occurring that will need to be fixed before 1.8.0 release, related to the impact of type annotations on type bindings in Eclipse JDT
- AspectJ 1.8.1 likely at the same time as Luna
  - Folding in Eclipse JDT Java8 fixes made in that timeframe

Communities

- Mailing list continues to be the most active place for AspectJ discussions – 99% of posts getting a response within 24hours
- Bug triage time a little worse than the 'within 48hours' it used to be
- Inclusion of AJDT in SpringSource Tool Suite drives some traffic on the STS forums related to AspectJ
- Blog on AspectJ and other eclipsey stuff: http://andrewclement.blogspot.ca/
  - Could do with a recent article!

IP Log

- Nothing unusual to report for 1.8.0
  - Moved to asm version 5.0.1 (from orbit)
- Iplog hosted here:
  - http://www.eclipse.org/projects/ip_log.php?projectid=tools.asp

IP Issues

- The EMO explicitly asks during the Release Review if any Member would like to assert that this release infringes their IP rights.
- If so, the EMO and the project will follow the Eclipse IP Policy in discussions with that Member.

Project Plan

- http://www.eclipse.org/projects/project-plan.php?projectid=tools.aspectj
- Work items on the horizon
  - persistent build state to avoid full builds being required on eclipse startup
  - For the 'Spring insight' project
    - more memory optimization work
    - more loadtime weaving performance work
- Future plans may include
  - adding new language constructs to support weaving of the invokedynamic instruction
  - pointcuts that match and bind on type annotations