Equinox Weaving: Bytecode Weaving for OSGi

Martin Lippert (it-agile GmbH)

Bytecode manipulation

- Used for a wide variety of scenarios
  - aspect weaving
  - JPA weaving
  - profiler instrumentation
  - proxy generation
  - ...
- Different ways:
  - static (compiler)
  - load-time (classloader or agent)
  - dynamic (JVMTI, redefineClasses)

Bytecode manipulation in OSGi?

- Somewhat more complicated
  - static might not be possible (bundles, modularization, separate compilation)
  - bundles and modularization might affect scope of bytecode manipulation
  - bundles might contain the necessary information for the weaving

Equinox Weaving helps

- Equinox Weaving is a bytecode manipulation infrastructure for Equinox/OSGi
  - formerly known as Equinox Aspects
  - OSGi framework extension
  - allows separate bundles to contribute bytecode modifiers as OSGi services
- Runtime takes care of
  - calling the bytecode modifiers at runtime
  - caching of modified bytecode

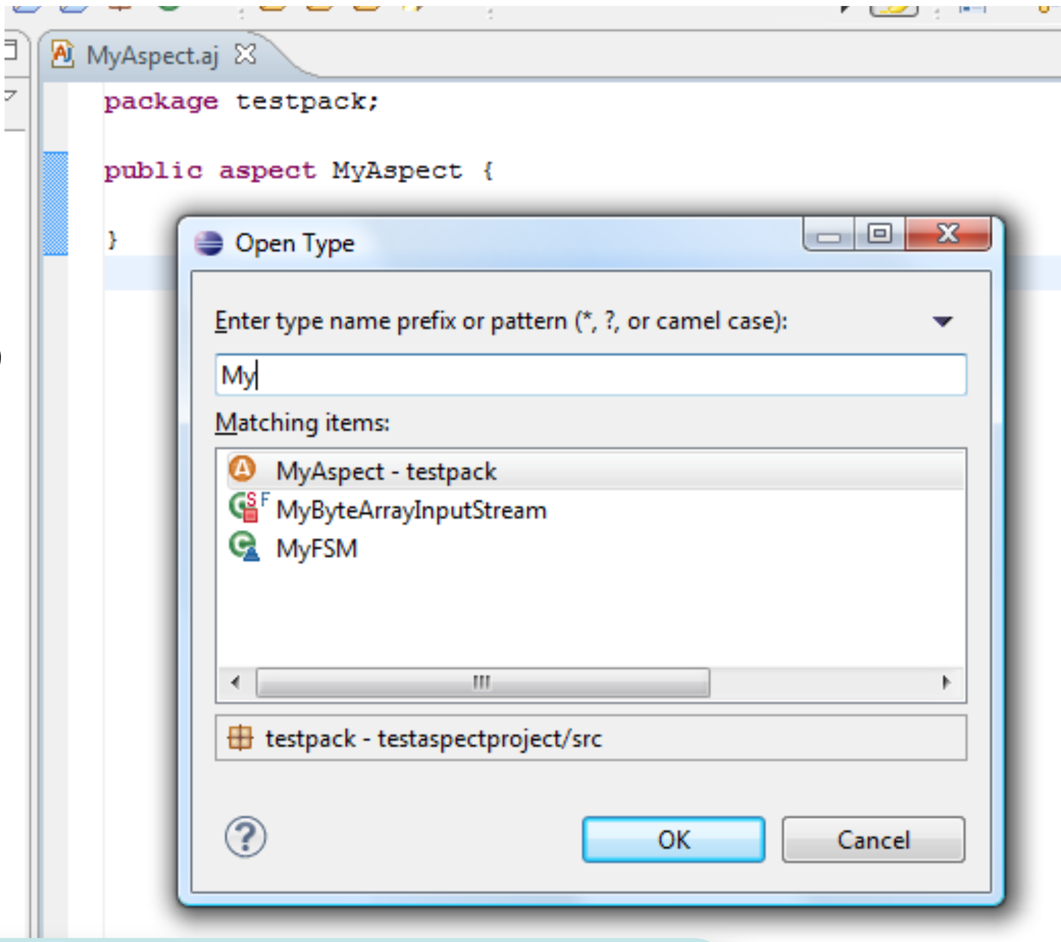
Aspect weaving for AspectJ

- A separate weaver bundle:
  - `org.eclipse.equinox.weaving.aspectj`
  - contributes aspect-weaving to the runtime
  - uses the Equinox weaving infrastructure

- Uses extender pattern
  - watches bundles that contain aspects
  - takes care of weaving those aspects into the „right“ target bundles

Adoption

- **AJDT** uses Equinox Weaving for AspectJ to weave into JDT
- **Scala IDE for Eclipse** uses the same JDT weaving



```
package testpack;

public aspect MyAspect {

}
```

Short Talk by Andrew Eisenberg at EclipseCon 2009: **Aspects Everywhere: Using Equinox Aspects to Provide Language Developers with Deep Eclipse Integration**

Weaving Dynamics

- “Be a good citizen of the OSGi community”
  - Weaving when resolved

- You can install and uninstall aspects at runtime
  - resolved or unresolved aspect bundles trigger dynamics
  - Refreshing other bundles automatically
  - But: other bundles need to be dynamic-aware

Manifest-Only Aspect Declaration

- You don’t need an aop.xml file anymore
- Declare your aspects within the manifest

```
Export-Package:
 mypackage;aspects="FooAspect,BarAspect"
```

Tells the runtime that **`mypackage`** is visible to other bundles and contains two aspects for weaving

Apply and contribute policies

- The **Aspect-Policy** is defined by the aspect bundle for the exported aspects
  - **Opt-In + Opt-Out**

- When you import an aspect, you can explicitly tell the system whether to **apply the aspects or not**
  - **True or False**

But remember...

- Aspect weaving for AspectJ is just one possibility...

Spring Dynamic Modules Bridge

- A weaver implementation that is a bridge between
  - Equinox Weaving
  - Springs Load-Time-Weaver implementation

- Provides a LoadTimeWeaver implementation
  - Spring‘s infrastructure for load-time bytecode weaving
  - allows typical Spring weavers to be registered
  - delegates weaving calls from the runtime to the registered Spring weavers (on a per-bundle base)

Caching

- The goal: **Zero overhead for cached scenario**
  - and independent of weaving implementation
- Many tweaks already done
  - Fast cache read/write/lookup IO
  - Just one load op (not the default and then the cached)
  - No cache lookup for non-woven bundles
  - Awareness of bundle versions + updates
- Latest addition:
  - Caching for generated classes (around closures)

Asynchronous Cache Writing

- Benefits:
  - Better performance for loading thread (doesn’t have to wait for cache IO)
  - Less concurrency for IO itself (simpler implementation)
  - More robustness for the system in case of IO problems
- Implementation:
  - Concurrent bounded queue (even if cache writing hangs, memory usage is bounded)

Past, Present, Future

- Evolved in Equinox incubator
- Now part of regular Equinox builds

- Future topics:
  - Ease of use
  - Better tooling
  - Management API
  - Multiple simultaneous weavers

Thank you for your attention!

- Questions and feedback welcome!

- Martin Lippert: lippert@acm.org