« Sometimes, things just work... | Main

... and sometimes they don't.

I've spent most of the last year involved in an intensive project that really drained me - hence the lack of blogging. I want to blog in a positive fashion this year, so I'll start by getting a lot of gripes off my chest. :) Call it things that suck.

Here's a short list, before I define what I mean by "suck":

- Maven2
- EJB3 Persistence
- Hibernate, caches, and the way they can kill your database.
- Maven2 (it just sucks a lot!)
- No Java 6 on the Mac
- No "next/previous word" keyboard navigation in the Mac terminal
- Mac Firefox, drop-down boxes, and tabbing

and I'm sure that there's more that will come to me. Today, I'm doing Maven - the rest will come later.

What do I mean by "suck"?

Mostly, I mean "it doesn't do what I want it to do". Most of these things actually have a lot of positive aspects to them, but there are features about them that just piss me off.

Maven2

We shifted to using Maven2 at work about this time last year. What a can of worms that was. I'm going to stress here that it was certainly a net-positive move, but it was a very bumpy ride. (I'm just going to call it Maven from here on)

The basic idea of Maven is great, and there are two sweet spots that Maven hits very well - building JARs, and building single webapps. If you're doing one of these, particularly if you're doing an open-source one, then Maven will have a lot of positive things for you. But if you're building more complex things, then Maven will almost certainly throw a few roadblocks in the way.

First, it's buggy, and the bugs don't get fixed fast. Great example - the Surefire plugin (the JUnit testrunner) 2.2 was released in April 2006. It was the version used by

SEARCH

Search this blog:

Search

ABOUT

This page contains a single entry from the blog posted on **January 21, 2008 11:16 PM**.

The previous post in this blog was Sometimes, things just work....

Many more can be found on the main index page or by looking through the archives.

Subscribe to this blog's feed
[What is this?]

Powered by
Movable Type 3.35

default in Maven 2.0.4 (also released in April 2006), and in Maven 2.0.5 (released January 2007). It had a bug - a pretty nasty one. When you ran several tests at once, the XML log files for a given test contained all of the tests run previously as well. Our Cruise builds were reporting our builds had nearly half a million tests, instead of the couple of thousand they should have. You can imagine the size of the log files, too. And, of course, if one of the early tests failed, then you had a couple of thousand test failures reported.

The bug was reported 22 May 2006. It was reported again on 17 July, 05 August, 10 August, 11 October and 24 March 2007. 3 of those were marked as Major, while 2 were marked as "Cannot Reproduce". A precise fix for the problem was mentioned on 9 August, with a 6-line patch provided (by the same person) on 28 August. This fix - for a Major bug identified 5 times that essentially broke the plugin for use in a CI environment - wasn't applied to the codebase until 27 November 2006. They didn't get a 2.3 release out until 23 February - too late for the Maven 2.0.5 release in January. (You could select it explicitly if you wanted to, but you had to know to do so) So it didn't get into the mainstream until 1 April 2007 (and who releases software on April Fool's Day?) - over 10 months after the original buggy release, on a project that brags about how it can be used to quickly push bug fixes out.

On average, our builds have just stopped dead due to a bug from a Maven plugin every 2 months or so. The fun ones are for the bugs in plugins that "auto-update" (i.e. don't have pinned versions). Case in point - over Christmas, the Maven Invoker plugin 1.1 was released. Somehow, this broke the Antrunner plugin so that if it was invoked in a multi-module reactor build, the working directory was changed. This broke one of our modules - but only if you invoked the module as part of a multi-module build. If you built the module by itself, it worked fine. Makes it easy to debug when the problem goes away when you try to reproduce it!

Combining multi-module builds causes fun interactions. The best one I've got is that the classpath for a plugin can only be set once. So, as a *purely hypothetical* example, if you had, oh, one module that wanted to run SQL against, say, an Oracle database, using the SQL plugin, and another module that wanted to run SQL against, say, a MySQL database, you have to give *both* modules *both* sets of JDBC drivers. (Why do it for both? Because Maven isn't deterministic between versions/JDKs for modules that don't have any dependencies - i.e., you can build them in any order).

Then there's the quality of the POMs. Commons Logging is a great example - here's a tool designed to isolate you from the exact logging library you use, and it brings in *3* logging libraries into your "compile" scope, one of which is Log4J which commons-logging "greedily" looks for. It also brings in the servlet API for some reason - again, at "compile" scope. Now, sure, you *could* use the commons-logging-api module instead. Except *nobody* else does. So if you're not using commons-logging explicitly (and unless you're building an open-source JAR to be used by someone else, why would you?) you have to deal with these stupid dependencies **that should never have been forced**

on you.

Now, it's not hard to fix this. Create a `<dependencyManagement>` section (preferably in the parent POM for your multi-module project) and define the exclusions for commons-logging. Except that, sometime between Maven 2.0.4 and Maven 2.0.6, this stopped working for transitive dependencies (esp. if the dependency was third-or-more hand). So you use, say, commons-beanutils, and it brings in Log4j. WTF??? Furthermore, the exact place that this occurs in the transitive hierarchy seems to shift between Maven versions. The short version of this is that I have had to alter POMs with 3 of the 4 Maven releases since I started using Maven just to cope with build breakages *caused by upgrading Maven*.

I mentioned earlier that Maven has a sweet spot with building webapps. The sour spot is building EAR files - especially if you've got multiple webapps. Why? Because Maven will try to shove all the libraries into the WEB-INF of the webapps - great for building standalone webapps, bad for EARs where you want to put the libraries in the EAR instead (to share with any EJBs, or just to reduce duplication between webapps). Now, the WAR plugin for maven does realised this, so they give a simple switch to turn off the "bundle in the WEB-INF" behaviour. So far so good.

Except that many web-frameworks (notably Struts and WebWork (aka Struts2)) use static variables for configuration (the "static-as-singleton" antipattern). So if you put two Webwork-based webapps in the same EAR, you need to bundle the Webwork libraries in the WEB-INF/lib dir of each webapp. Except the *only* way to do this with Maven is to go through and explicitly make all the other dependencies "provided" scope and THEN go into the EAR's pom and make them all "compile" scope so that they get bundled in the EAR. Aaarggh!

You also absolutely have to set up your own internal Maven repository, and pin down all the versions of both dependencies and the plugins. Without this, you don't get repeatable builds - ie, you can't go back, say, six months in your source control and do a build (even if you use the same version of Maven you did then). The plugins would have changed, and they change in incompatible ways. Some of those ways are just designed to make your life a pain - case in point with the Surefire plugin is that they changed what the default values of some options were. However, there are no good and easy ways to do an internal Maven repository (though there are some getting better).

Multi-module builds and versioning are a pain. You can define a parent POM for your POM - you can even specify a relative path to the parent POM. However, you have to specify the version number of the parent POM - it can't just look it up from the relative path. So if you have, say, a dozen modules in your project, you've repeated the version of the parent POM 12 times. WTF??

Some petty gripes to wind up on:

- the Eclipse plugin (for Maven, not the Maven plugin for Eclipse) doesn't remember that it couldn't fetch source. So when you generate the Eclipse projects for your dozen-module projects, it goes out to the web a dozen times for each dependency it can't find the source for (i.e. most of them)
- *What* is the problem with attributes on XML elements? Ivy specifies dependencies in one line. Maven needs 5. When you're reading through a POM, the excessive repetition makes it harder to find problems. Seriously, why can't we just say `<dependency groupId="foo" artifactId="bar" version="1.2.3" scope="compile">`? BTW, this repetition tosses away any size benefits of Maven vs Ant - when I migrated from Ant to Maven, the *total byte size* of the build files went up by about 10% (the line count went up by about a third). Today, I'm averaging about 25 lines of Java code for every line in our POMs.
- Not so petty - why can't we have extra scopes? The hard-coded scopes aren't flexible enough (see the webapp problem)
- I've almost certainly spent more time and effort in the last year wrestling with POM files than I spent in the previous 7 years wrestling with Ant.
- It's not like using Maven gets you away from using Ant anyway. I mean, why can't they at least provide an "execute" plugin?
- The documentation is patchy - some parts are great, most are crappy with holes in it. The "Better Builds With Maven" book , for example, has a section on defining your own plugins using a version of the plugin tools that was not released when the book was published (and I think it's still not released). You also can't see documentation for older versions of plugins, and it can be hard to work out which version you're using (great when, for example, the behaviour of default values for options change!)

To finish, I'll just quickly pre-empt most of the comments I'm likely to get:

- "Maven is free - stop bitching". Yes, Maven is free, and yes you get what you pay for. But it's only free if your time is worth nothing. Most of these problems take serious time to get around.
- "Why not contribute back to the Maven community?" Well, for starters, I'm personally very busy - "intensive project that really drained me", remember? My family then takes up most of my time. Of course, work could contribute back - but it's hard to make the argument to say "we should spend time improving this buggy software we've chosen to use" (and Wotif is friendly with regards to contributing to open source). Furthermore, the Maven community has a real chip on its collective shoulder - I lurk on the Maven user and developer mailing lists, and there's a *lot* of attitude floating around on both. In order to make an improvement, you first have to identify a problem, and identifying problems is almost tantamount to attacking Maven on those lists - and many people take that personally. Even if you do come up with an

improvement, and do the work to put in a patch, it's likely to get either rejected because the plugin owner disagrees with the purpose of the patch, rejected because "it will be fixed in Maven 2.1", or just really slow going in (see the Surefire bug above). In other words, the Maven community is just hard to join.

- "If you don't like Maven, go back to Ant". Like I said, Maven is a net positive for us as a development team (even if it's probably a net negative for me personally). I just prefer my experiences to be smoother.
- "Why haven't you reported the bugs?" We did in some cases. There are plenty of bugs we didn't report, though - mostly because they tend to be hard to get into, and we end up scratching our heads about how to come up with a simple example for a bug report (we can't exactly attach our entire code base to a bug report). Also, it can be really hard to follow the build process in detail (i.e. in the code) to work what's going wrong so you can report a bug more detailed than "Our build doesn't work"
- "Don't like it? Come up with something better" Oh, please don't tempt me. :)

FWIW, I don't think the current state of Maven, and the Maven community in particular, is going to last. Either the tool and the community will mature (and get less defensive), or it's going to fall over in a flaming heap and be replaced by something better. If I had to bet, my money would be on the latter - probably around the end of the year, or early-to-mid next year. The Java community wants a tool like Maven, needs a tool like Maven - that's the only reason why a tool as shitty as Maven has developed such a large following. What it needs is a better Maven.

Posted by Robert on January 21, 2008 11:16 PM | Permalink

Comments (2)

Christophe Vanfleteren:

I feel your pain.

I was and still am responsible for the build system for my current project (which is pretty big: we have 5 webservices (each a seperate war), 2 jsf apps and 3 fully asynchronous headless apps, that are driven by jms). We have 15+ modules for the moment, and I'm sure there's more to come.

Setting up the right configuration for all this wasn't a walk in the park, but I do feel the same like you do:
the net result for the team is positive, but it took me quite some time to figure everything out, and I still feel it's not 100% finished.

The instability of some plugins certainly is a problem: its only now, with surefire 2.4 that we can properly use Testng integration. We depended on the surefire snapshots for quite some time, which could break our build randomly now and then.

On the other hand, you do gain quite some functionality. Deploying with cargo is very

nice, I do feel maven poms are more maintainable than equivalent ant scripts and it's very easy to add another module to your project.

About your "exec" remark, doesn't the maven exec plugin fill your need? See http://mojo.codehaus.org/exec-maven-plugin/introduction.html

Posted by Christophe Vanfleteren | January 22, 2008 8:42 AM

Posted on January 22, 2008 08:42

Michael Neale:

The dysfunctional maven community is particularly bad. Because its dysfunctional for no reason at all that I can see (other then "chip on shoulder" as identified above).

Great idea, not so great execution. I think its been long enough to sort itself out, I kind of want to see an acrimonious fork happen now.

Posted by Michael Neale | January 22, 2008 1:22 PM

Posted on January 22, 2008 13:22

Post a comment

If you have a TypeKey identity, you can sign in to use it here.

Name:

Email Address:

URL:

Remember personal info?

Comments:

Due to excessive Blog Spam, you need to answer a simple question:

What is the answer to Life, the Universe, and Everything? **(required)**:

Preview Post