Experiment - Explicit JPMS support

The existing "guesswork" code should be replaced with proper support, this "experiment" (w/ some PoC code) will aim to achieve that.

To achieve the goal, several aspects of Maven (core and plugins) will need changes. The goal of this experiment is to produce a "demo", but not to produce code that may be used as-is, the goal of this experiment is NOT to produce production ready code. Most probably will have more hacks than anticipated (at least this is what I envision).

- Dependency types
- Packaging
- Progress
- Ideas/TODOS
- Related documents:

### Dependency types

New types with existing "jar" present just for reference.

<table>
<thead>
<tr>
<th>name</th>
<th>extension</th>
<th>classifier</th>
<th>[flags]</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>jar</td>
<td>jar</td>
<td>-</td>
<td>[CP]</td>
<td>same as today, added to classpath</td>
</tr>
<tr>
<td>module</td>
<td>jar</td>
<td>-</td>
<td>[MP]</td>
<td>a JPMS module, added to modulepath</td>
</tr>
<tr>
<td>java-agent</td>
<td>jar</td>
<td>-</td>
<td>[AG]</td>
<td></td>
</tr>
<tr>
<td>annotation-processor</td>
<td>jar</td>
<td>-</td>
<td>[AP]</td>
<td>not on classpath/modulepath, m-compiler-p handles it</td>
</tr>
<tr>
<td>doclet</td>
<td>jar</td>
<td>-</td>
<td>[DOC]</td>
<td>not on classpath/modulepath, m-javadoc-p handles it</td>
</tr>
<tr>
<td>fatjar</td>
<td>jar</td>
<td>-</td>
<td>[ID,CP]</td>
<td>new type to express &quot;self contained&quot; [jar added to classpath]</td>
</tr>
<tr>
<td>fatmodule</td>
<td>jar</td>
<td>-</td>
<td>[ID,MP]</td>
<td>new type to express &quot;self contained&quot; module added to modulepath</td>
</tr>
</tbody>
</table>

Flags:
- CP - added to classpath
- MP - added to modulepath
- AG - agent (do whatever needed with it) -> ?
- AP - annotation processor (do whatever needed with it) -> m-compiler-p handles it
- DOC - doclet (do whatever neede with it) -> m-javadoc-p handles it
- ID - includes dependencies

Note: from Resolver perspective only the ID flag makes sense (so should be defined in resolver, presence of this flag effectivel stops collection), while all the other flags are "Maven customizations", have no meaning for Resolver.

### Packaging

packaging=module

Pretty much same as today "jar" packaging, but

In "validate" phase:
- enforce Java9+
- enforce only "module" type of dependencies present in some (which? "compile"+"runtime"+"provided") scopes
- having "test" scoped "jar" type of dependency is okay?
- forbid sytem scope totally?

In "package" phase:
- enforce presence of module descriptor?
- validate module descriptor?
- etc

The output of this packaging would be artifactId-version.jar (yes, same as for packaging "jar"), that still can be consumed downstream either as type=jar or type=module, as use case requires.

### Progress

Maven master changes:
- extend Type to be able to represent flags
- introduce new "module" packaging TBD currently very same as "jar" packaging (!!!) later to refine
- introduce new Types (as per gist)
• In Maven, so far there was type-registry in resolver, then artifact handler manager in Maven3, and new Maven4 API that was bolted on top of Maven3. This is now changes: the new Maven4 API "drives" and is single source of truth, and Maven3 handler manager and resolver goes back to it to consult.

• A hack for legacy (maven3) plugins is needed and made example work: https://github.com/cstamas/maven/commit/89b0651bba2e84ba7b9bd7d5c9f7a2068a1a077

The modified Maven is here: https://github.com/cstamas/maven/tree/module-experiment

The experimental project (uses slf4j-api and slf4j-simple sources, thanks!) consist of 3 parts:

• A maven4 plugin (uses NEW Maven4 API!) just to show what plugin sees (it merely dumps out flags), example output: https://gist.github.com/cstamas/2831be5e56e330e14f29934867a16dd3

• A modular project api, impl and fat (api=slf4j-api, impl=slf4j-simple sources, fat=api+impl), where fat produces a "squashed module" out of api and impl, so 3 artifacts are being built w/ new "module" packaging

• A modular consumer project that showcases consumption of these

The experimental build project (buildable with modified maven ONLY) is here: https://github.com/cstamas/maven4-jpms-experiment/tree/main

Note: the experimental build project contains a "hack" for IDEA IDE: when you load it up, enable "idea" profile, and IDE will work for you. Still, on CLI you will get what you expect to get.

Ideas/TODOS

• I’d rename JPMS related types to "jpms-module" instead, as we may want to get in "play' OSGi as well?

Related documents:

• Gist listing needed artifact types and packaging https://gist.github.com/cstamas/4e9bcbef25ce912a90ad1e127b0c5db8