Chained Local Repository Manager (CLRM)

Maven Resolver since 1.9.2 version introduced this new feature. Original intent was "IT LRM isolation", as part of

The Original User Story

Maven IT suite is run as part of Maven build. Let's call it the "outer build". The outer build may build plugins and some artifacts needed and used by ITs, but may also pre-populate the local repository with needed dependencies (as ITs are cut off from network). The ITs, let's call them "inner build", should run in isolated environment, so not using user settings, nor MRM and not being able to reach to remote repositories (aside of those that IT test itself set up).

Problem: the outer build is usually affected by user environment (settings.xml, use of MRM, and may use user own local repository unless alternate specified), but also we do not want user LRM to be altered by IT runs. One IT may alter LRM contents in such way that LRM becomes corrupted or incomplete for other ITs. The "inner build" on the other hand, may fail if use same LRM as outer build, as they are isolated, do not use settings.xml from the outer build, may not use MRM and most probably not using same remote repository IDs. All these may lead to mysterious "artifact not found" problems. Typical case: outer build may use MRM that defines mirrorOf with ID "my-mrm", while inner would use defaults, where only remote repository is "central": this leads that user LRM gets populated with artifacts from "my-mrm" remote repository, while inner build would know and hence, ask only about "central" remote repository. Enhanced LRM (default since Maven 3.0) refuses to serve up these artifacts, reports them as "unavailable".

Solution is CLRM: with CLRM user is able to specify isolated LRM for all ITs, or even per each IT, while still making artifacts from outer LRM available for the IT build. Inner build uses isolated LRM solely, but for resolution purposes still is able to resolve anything from outer LRM. Moreover, the outer LRM is guaranteed to be read-only, it cannot be altered by inner builds in any way.

What it is and what it is not?

CLRM simply allows several LRM (FS directories, if we want to simplify) to be "chained" and to be used as source of locally available artifacts (cached or installed). It does not split, redirects writes vs reads, in essence it is very simple thing. It also helps to overcome one less well known feature of Enhanced Local Repository Manager: scoping of cached content by its origin. Since Maven 3.0 Enhanced Local Repository Manager is being used as LRM, that was doing this work of "scoping" of cached artifacts.

CLRM consist of head LRM and list of tail LRMs (one or more). The head literally works completely same as one would expect from LRM: it receives all the cached artifacts and all the installed artifacts, in short, it is volatile/modified in case build resolves/caches or installs. The tail LRMs are used as read-only, they are not and cannot be modified in any way by CLRM and Maven using CLRM.

A bit of digression: when LRM caches an artifact from remote repository, it will track it's repositoryId as well. This means that in LRM the artifact lies on layout path, but there are some metadata listing artifact origin as well. When resolving an artifact, the API of LRM will be asked for artifact in context of remote repositories (as many as many project defines, by default the "central" remote repository), and if none of the remote repository IDs are found among tracked origin repository IDs, the LRM will still return the artifact file, but will mark it as "unavailable". See also related aether.artifactResolver.simpleLrmInterop on Configuration page.

This "scoping" goes step further in split local repository: here, if there is no overlap between tracked remote repository IDs and context remote repository IDs, the local file is not even returned. Hence, scoping becomes even more apparent (and physically enforced) when split feature of local repository is used. This implies, that overriding scoping in combination of split local repository is impossible!

CLRM and SyncContext

Given CLRM delegates to head, the head will be the local repository holding advisory locks as well (if file advisory locking used). Moreover, as tail local repositories are read-only, they are NOT locked separately. Best practice to be on safe side, if you ensure that tail local repositories are not modified during their use in CLRM, just like it happens in original user story.

How to enable it?

In case of Maven, version 3.9.0 and after introduced these user properties:

- maven.repo.local.tail - User property for list of "tail" local repository paths (separated by comma, default is null).
- maven.repo.local.tail.ignoreAvailability - User property setting whether Maven should ignore artifact availability in tail local repositories or not (boolean value, default is true, so it ignores availability).

For example, one can invoke Maven as this:

- mvn clean install -Dmaven.repo.local=/home/user/outer
- mvn clean verify -Dmaven.repo.local=/home/user/inner -Dmaven.repo.local.tail=/home/user/outer

First invocation will build and install artifacts of "outer" build into /home/user/outer directory, while second invocation will use it's own /home/user/inner local repository (to cache) but artifacts from /home/user/outer will be available to it as well. Moreover, while not present in example, the two invocations may use different settings.xml as well, but artifacts in outer will still be available to inner, as by default, availability (scoping) is ignored.