Lattice: A Multi-module Java Build System written in Python
(Draft)

December 31, 2010

1 Overview

In Lattice build files are written not in XML, but in the Python language. The benefits are much better readability and powerful imperative build scripting supported by Python.

For multi-module projects. Lattice uses topological sorting to decide the correct order to build each module. It's also planned that Lattice will analyze the module dependency to determine how the module compilation can be parallelized.

Lattice's source code is extremely lean, currently it consists of about 500 lines of Python source code.

Lattice supports generating Eclipse projects from the build files.

2 Five Minute Introduction

A simple Lattice build file can be as short as two lines:

```
# build file module1/__init__.py
depends = [ 'module2']
libs = ['logging', 'commons']
```

A typical module is a directory that contains Java source files which are built into a single JAR file. A library is a directory that contains one or more third party binary JAR files required for compiling or running modules. Lattice's build files are named *__init__.py*. Here *module1/__init__.py* states module1 depends on another module (module2) and two libraries: logging and commons. On disk the files layout for such a two module project is:

```
-- module1 --+-- __init__.py
             |
             + src / main /java /  (Java sources)
```

```
-- module2 --+-- __init__.py
             |
             + src / main /java / ... (Java sources)

-- libs -----+--- logging / log4j.jar, slf4j-api.jar, slf4j-log4j.jar ...
             |
             +-- commons / commons-lang.jar, commons-io.jar, commons-codec.jar ...
```

Here *logging* is a user defined library (UDL) that consists of log4j and SLF4J's binding for log4j. *commons* is another UDL that contains a few Apache Commons libraries.

To build module1 and module2, type:

lat -m module1 -t jar_all

This will compile all the Java source code in module2 and then module1 and create *module1/target/module1.jar* and module2/target/module2.jar

3 Tasks

A typical Lattice command is in the form of :

```
lat -m <module> -t <task>
```

Lattice has several system built-in tasks: *build,clean,clean_local,jar,jar_all, junit, run_java_class, war* . It's also easy to have per-module user-defined tasks, for example, you can add a Python function to the build file:

```
def run_multiple_java_programs ():
    tasks.run_java_class(__name__, 'com.mycompany.JavaProg1', 'arg1',
mx='1024m')
    tasks.run_java_class(__name__, 'com.mycompany.JavaProg2', 'arg2',
mx='512m')
```

Here a custom task `run_multiple_java_programs` is defined to run two Java classes with specific arguments and JVM memory settings. The classpath settings needed for running the classes will be automatically calculated by Lattice. The task can be invoked as:

```
lat -m mymodule -t run_multiple_java_programs
```

Because a custom task is just a regular Python function, and Python is a very high-level language well suited for invoking operating system commands, shell scripts and other programs , a custom task can be created to do almost any types of work. It should be easy to invoke other Java build systems such as ant, maven or ivy in a custom task.

4 Installation

Source code: `https://github.com/hackingspirit/lattice` . To download a tar ball: https://github.com/hackingspirit/lattice/tarball/master . To check out a read-only copy:

```
git clone git://github.com/hackingspirit/lattice.git
```

To install:

```
sudo python setup.py install
```

This will install lattice into your Python system path. To run Lattice, copy the shell script "lat" to a directory that's in your executable search PATH, such as /usr/bin . Type "lat -h" to see a list of options.

5 A multi-module Java Web Project Example

A seven-module sample project where the topmost module builds into a .war file is included in the source distribution. For details please see README file under the sample_java_prj/ folder of the distribution.

6 Eclipse Support

To generate Eclipse projects:

```
lat --eclipse-gen
```

Each module or library is converted to one Eclipse project. In addition a projectList.txt file is created which can be used to import all the projects into Eclipse at once using the excellent Eclipse bulk import tool (http://code.nomad-labs.com/eclipse-bulk-import/) .

7 Some Build Details

The default task is *build*, when executed Lattice compiles all the dependent modules transitively first (unless the *–disable-transitive-dependencies* flag is on) and then compiles the module being built. A Java source file (.java) is only (re)compiled if it's newer than its corresponding .class file.

By default Java sources are found in `src/main/java`, resource files are under `src/main/resources`, web files (JSP, HTML etc) are under `src/main/web`, JAR's and WAR's are built into `./target/` and intermediate files (.class etc) are saved in `build/`. These settings are defined in the file *settings.py* and can be easily customized.

8 Future Work

Parallel builds, detect library collisions (different versions of same library/package etc), various plugin-tasks (GWT, ...)