Technical Explorations

by Keith R. Bennett

Hello, Nailgun; Goodbye, JVM Startup Delays

Sep 15, 2012 • keithrbennett

One of the frustrations of working with JRuby is that every single time you run it, you start a whole new JVM. This takes seconds:

```
>time jruby -e 'puts(123)'
123
jruby -e 'puts(123)'  1.94s user 0.11s system 178% cpu 1.144 total
```

If you're using JRuby, and working with gem, rspec, irb, and other JRuby tools, this waiting time adds up and can be frustrating.

Enter Nailgun

Nailgun is a Java utility that starts up a JVM and behaves like a server, accepting client requests to run Java based software on it. The JRuby team did a great job of integrating it into JRuby, making it trivially simple to use.

To start the server, just run:

```
>jruby --ng-server  # 'jruby' can be replaced with 'ruby' if running in rvm
```

To connect to it, all you need to do is add "–ng" to the JRuby command or the JRUBY_OPTS environment variable's value. If you always wanted to use it, you could just include "–ng" in the export of JRUBY_OPTS in your startup script (.bashrc, .zshrc, etc.). However, this may not be a good idea. The Nailgun web site says "…it's not secure. Not even close". In addition, you probably want longer running tasks to have their own JVM's. This being the case, it can be better to default to *not* use Nailgun, instead specifying the use of it when needed.

In my work, I found that I always wanted to use Nailgun with utilities (rspec, irb, etc.), but that I couldn't use it with my gem's bin executable running in 1.9 mode. (I believe this is fixed in a 1.7 version of JRuby.) So here's what I did…

I created the scripts below in my ~/bin directory. (I use a ~/bin directory for home grown scripts and such that I don't want to bother installing in a root-owned directory.)

---

ngs, to run the Nailgun server:

```
>JRUBY_OPTS="" ruby --ng-server
```

I set JRUBY_OPTS to the empty string because my default setting is "–1.9", and at this version, the Nailgun server will not start when JRUBY_OPTS is nonempty or when certain options are specified on its command line. (See issues 6246, 5611, and 6251).

---

ng, to run any JRuby command with Nailgun (used by ngem, etc.):

```
>JRUBY_OPTS="$JRUBY_OPTS --ng"  $*
```

---

nruby, to run the JRuby interpreter itself:

```
>JRUBY_OPTS="$JRUBY_OPTS --ng" ruby $*
```

---

ngem:

```
>ng gem $*
```

---

nrspec:

```
>ng rspec $*
```

---

nirb:

```
>ng irb $*
```

This can also be done with rails, of course, but I haven't tested it thoroughly, so I suggest keeping an eye on things to make sure it works ok.

nrails:

```
>ng rails $*
```

To quickly create these all, change to the directory that will contain them and run this script:

```
echo 'JRUBY_OPTS="" ruby --ng-server'          > ngs;      chmod +x ngs
echo 'JRUBY_OPTS="$JRUBY_OPTS --ng" $*'       > ng;       chmod +x ng
echo 'JRUBY_OPTS="$JRUBY_OPTS --ng" jruby $*' > nruby;    chmod +x nruby
echo 'ng rspec $*'                            > nrspec;   chmod +x nrspec
echo 'ng irb $*'                              > nirb;     chmod +x nirb
echo 'ng gem $*'                              > ngem;     chmod +x ngem
echo 'ng rails $*'                            > nrails;   chmod +x nrails

# Rehash reloads binaries from your path to be available for autocompletion
# on the command line in this shell.
rehash
```

It would probably be simpler to create aliases instead of shell scripts, but I like being able to easily modify these kinds of scripts, sometimes with multiple lines, so this works well for me.

If you'll be executing the code below, make sure you've started the Nailgun server, either using the *ngs* script we created, or the command shown at the top of the article.

The scripts will not *always* work. When the arguments include quoted strings that include spaces, things may get messed up. Here's an example:

```
>>nruby -e "puts 123"  # (produces blank string)

>nruby -e "puts(123)"
123
```

Running the same trivial Ruby script as before, but with Nailgun this time, we get:

```
>>time nruby -e 'puts(123)'
123
nruby -e 'puts(123)'  0.00s user 0.01s system 3% cpu 0.197 total
```

I recently needed to do some cycles of delete gem; build gem; install gem. This was easy to put together on a single line so I could just scroll up my history to repeat the process. Here are the timings, first without, and then with, Nailgun. I've removed irrelevant output for brevity's sake.

Without Nailgun:

```
>>time (echo Y | gem uninstall life_game_viewer; gem build *gemspec; gem install life_game_viewer)
31.74s user 1.29s system 235% cpu 14.021 total
```

With Nailgun:

```
>>time (echo Y | ngem uninstall life_game_viewer; ngem build *gemspec; ngem install life_game_viewer)
0.01s user 0.02s system 0% cpu 3.073 total
```

Pretty amazing, eh?

Published with [GitHub Pages]