

DTrace for Ruby Applications

DTrace is a dynamic instrumentation for tracing technology, built in Solaris kernel. It might be used to analyze the whole system, or specific applications. To analyze Ruby applications, you need to use an instrumented Ruby implementation. The default Solaris Ruby 1.8 package is DTrace instrumented. We'll take a quick trip to have a (partial) glance on what can be done with DTrace for Ruby.

From Netbeans, you might define a new Ruby environment (this time “CRuby”, as opposed to Jruby), and run your application under a DTrace instrumented Ruby interpreter. It is easy to switch between Ruby environments in Netbeans. You can either run a CRuby project from within Netbeans or from a shell prompt. From Netbeans. You can add a CRuby environment and run a project in this environment, by right clicking a Rails project, choose properties, choose 'Manage' to add a new Ruby environment (specify /usr/ruby/1.8/bin/ruby), and then set this CRuby as the project Ruby platform. Once you have set the Solaris “CRuby” environment, you might run all tracing scripts of the DTrace Toolkit, under /opt/DTT/Ruby.

Steps

1. In Netbeans, right click the 'Hello' project in the project pane, select 'Properties'.
 1. Click the "Manage..." button on the top right corner of the dialog.
 2. A 'Ruby Platform Manager' dialog opens, click 'Add Platform' button (at bottom right).
 3. In the file navigation dialog, navigate to and select '/usr/ruby/1.8/bin/ruby'
 4. click OK
2. In Netbeans, right click the 'Hello' project in the project pane, select 'Properties'.
 1. From the 'Ruby Platform' list options, select 'Ruby 1.8.6-p287'
 2. Server should be 'WEBrick'
 3. Set Server Port Number to 3000
 4. click OK
3. Open a Solaris terminal (either select the terminal top toolbar icon or by right clicking on the desktop and selecting 'Open Terminal')
4. From the shell prompt, run:

```
# su
# cd /opt/DTT/Ruby
```
5. These are the available Ruby DTrace Scripts. You can freely try, and we'll give 2 examples below.

rb_calldist.d - measure Ruby elapsed times for types of operation.

Written for the Ruby DTrace provider.

USAGE: rb_calldist.d # hit Ctrl-C to end

rb_calls.d - count Ruby calls using DTrace.

Written for the Ruby DTrace provider.

USAGE: rb_calls.d # hit Ctrl-C to end

rb_calltime.d - measure Ruby elapsed times for types of operation.

Written for the Ruby DTrace provider.

USAGE: rb_calltime.d # hit Ctrl-C to end

rb_cpudist.d - measure Ruby on-CPU times for types of operation.

Written for the Ruby DTrace provider.
 USAGE: rb_cpudist.d # hit Ctrl-C to end

rb_cputime.d - measure Ruby on-CPU times for types of operation.
 Written for the Ruby DTrace provider.
 USAGE: rb_cputime.d # hit Ctrl-C to end

rb_flow.d - snoop Ruby execution showing method flow using DTrace.
 Written for the Ruby DTrace provider.
 USAGE: rb_flow.d # hit Ctrl-C to end

rb_flowinfo.d - snoop Ruby function (method) flow with info using DTrace.
 Written for the Ruby DTrace provider.
 USAGE: rb_flowinfo.d # hit Ctrl-C to end

rb_flowtime.d - snoop Ruby function (method) flow using DTrace.
 Written for the Ruby DTrace provider.
 USAGE: rb_flowtime.d # hit Ctrl-C to end

rb_funcalls.d - count Ruby function (method) calls using DTrace.
 Written for the Ruby DTrace provider.
 USAGE: rb_funcalls.d # hit Ctrl-C to end

rb_lines.d - trace Ruby line execution by process using DTrace.
 Written for the Ruby DTrace provider.
 USAGE: rb_lines.d # hit Ctrl-C to end

rb_malloc.d - Ruby operations and libc malloc statistics.
 Written for the Ruby DTrace provider.
 USAGE: rb_malloc.d { -p PID | -c cmd } # hit Ctrl-C to end

rb_objcpu.d - measure Ruby object creation on-CPU time using DTrace.
 Written for the Ruby DTrace provider.
 USAGE: rb_objcpu.d # hit Ctrl-C to end

rb_objnew.d - count Ruby object creation using DTrace.
 Written for the Ruby DTrace provider.
 USAGE: rb_objnew.d # hit Ctrl-C to end

rb_stat.d - Ruby operation stats using DTrace.
 Written for the Ruby DTrace provider.
 USAGE: rb_stat.d [interval [count]]

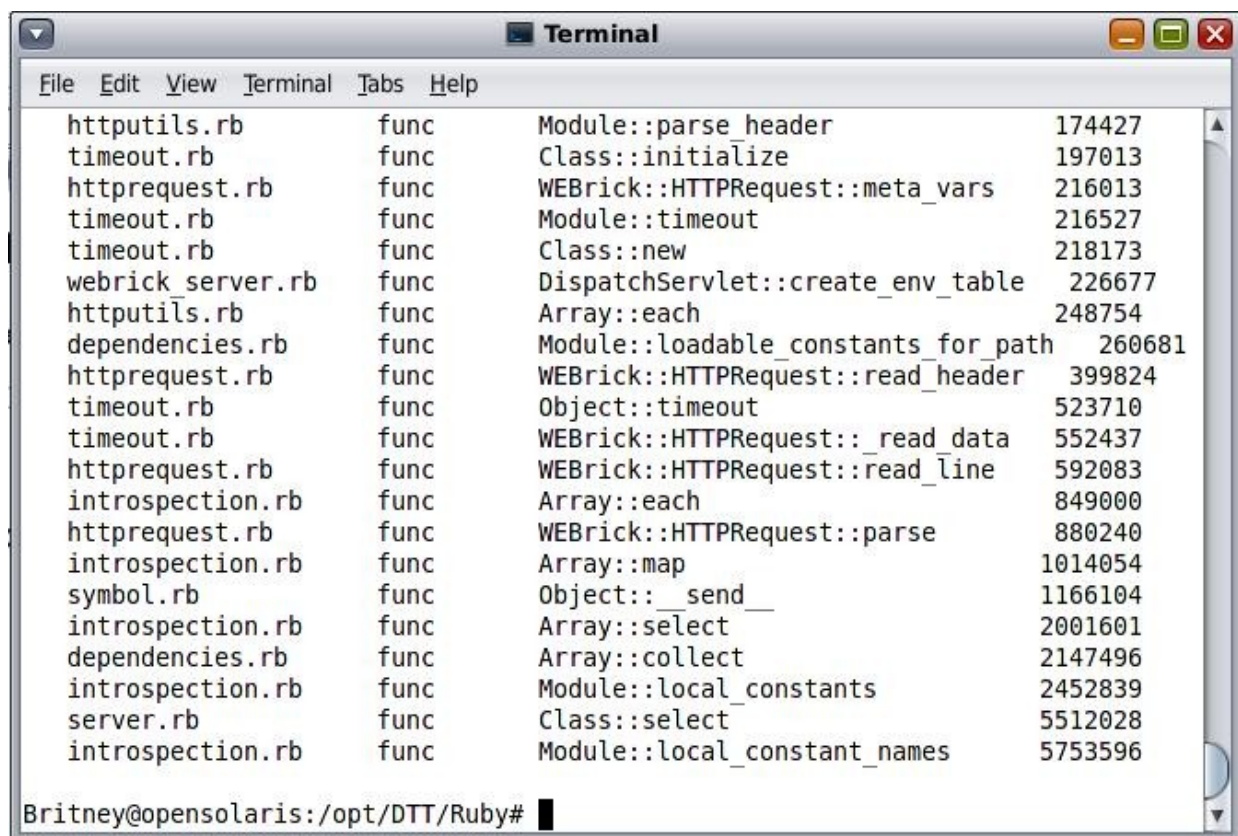
rb_syscalls.d - count Ruby calls and syscalls using DTrace.
 Written for the Ruby DTrace provider.
 USAGE: rb_syscalls.d { -p PID | -c cmd } # hit Ctrl-C to end

rb_syscolors.d - trace Ruby method flow plus syscalls, in color.
 Written for the Ruby DTrace provider.
 USAGE: rb_syscolors.d { -p PID | -c cmd } # hit Ctrl-C to end

rb_who.d - trace Ruby line execution by process using DTrace.
 Written for the Ruby DTrace provider.
 USAGE: rb_who.d # hit Ctrl-C to end

6. Examples:

1. From the shell prompt (while in /opt/DTT/Ruby, as root), run:
 # ./rb_calltime
2. Browse to or refresh the hello project page (like <http://localhost:3000/home/index>)
3. Wait 10 seconds and click Ctrl-C in the terminal window. You should get a list ended like here:



The image shows a terminal window titled "Terminal" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The window displays a list of 20 Ruby functions, each preceded by a file name and the word "func". The functions are sorted by their aggregated elapsed time in ascending order. The list includes functions like `Module::parse_header`, `Class::initialize`, `WEBrick::HTTPRequest::meta_vars`, and `Module::local_constant_names`. The terminal prompt at the bottom is `Britney@opensolaris:/opt/DTT/Ruby#`.

File	Type	Function	Time
httputils.rb	func	Module::parse_header	174427
timeout.rb	func	Class::initialize	197013
httprequest.rb	func	WEBrick::HTTPRequest::meta_vars	216013
timeout.rb	func	Module::timeout	216527
timeout.rb	func	Class::new	218173
webrick_server.rb	func	DispatchServlet::create_env_table	226677
httputils.rb	func	Array::each	248754
dependencies.rb	func	Module::loadable_constants_for_path	260681
httprequest.rb	func	WEBrick::HTTPRequest::read_header	399824
timeout.rb	func	Object::timeout	523710
timeout.rb	func	WEBrick::HTTPRequest::_read_data	552437
httprequest.rb	func	WEBrick::HTTPRequest::read_line	592083
introspection.rb	func	Array::each	849000
httprequest.rb	func	WEBrick::HTTPRequest::parse	880240
introspection.rb	func	Array::map	1014054
symbol.rb	func	Object::_send__	1166104
introspection.rb	func	Array::select	2001601
dependencies.rb	func	Array::collect	2147496
introspection.rb	func	Module::local_constants	2452839
server.rb	func	Class::select	5512028
introspection.rb	func	Module::local_constant_names	5753596

Called Ruby functions are listed, sorted in increasing order of their aggregated elapsed time

4. From the shell prompt (while in `/opt/DTT/Ruby`, as root), run:

`# ./rb_syscolors.d -p `pgrep -n ruby``

5. You should get a list ended like here:

```
Terminal
File Edit View Terminal Tabs Help
ncontext
0 1201 274 ":- syscall <- setco
ncontext
0 1201 1027 timeout.rb:55 method -> Class
::inherited
0 1201 1008 inheritable_attributes.rb:125 line
--
0 1201 921 inheritable_attributes.rb:125 line
--
0 1201 18993 ":- syscall <- nanosleep
0 1201 663 ":- syscall -> nanosleep
0 1201 4805 ":- syscall -> pol
lsys
0 1201 599 ":- syscall <- pol
lsys
0 1201 696 inheritable_attributes.rb:125 method
-> Object::respond_to?
0 1201 1264 inheritable_attributes.rb:125 method
<- Object::respond_to?
^C0 1201 1229 inheritable_attributes.rb:127 line
--
Britney@opensolaris:/opt/DTT/Ruby#
```

Methods flow is printed, as well as system calls which are called, with a special color coding.

Enjoy!