NAME

perldtrace - Perl's support for DTrace

SYNOPSIS

```
 # dtrace -Zn 'perl::sub-entry, perl::sub-return { trace(copyinstr(arg0))
}'
 dtrace: description 'perl::sub-entry, perl::sub-return ' matched 10 probes

 # perl -E 'sub outer { inner(@_) } sub inner { say shift } outer("hello")'
 hello

 (dtrace output)
 CPU     ID                    FUNCTION:NAME
   0  75915       Perl_pp_entersub:sub-entry   BEGIN
   0  75915       Perl_pp_entersub:sub-entry   import
   0  75922      Perl_pp_leavesub:sub-return   import
   0  75922      Perl_pp_leavesub:sub-return   BEGIN
   0  75915       Perl_pp_entersub:sub-entry   outer
   0  75915       Perl_pp_entersub:sub-entry   inner
   0  75922      Perl_pp_leavesub:sub-return   inner
   0  75922      Perl_pp_leavesub:sub-return   outer
```

DESCRIPTION

DTrace is a framework for comprehensive system- and application-level tracing. Perl is a DTrace *provider*, meaning it exposes several *probes* for instrumentation. You can use these in conjunction with kernel-level probes, as well as probes from other providers such as MySQL, in order to diagnose software defects, or even just your application's bottlenecks.

Perl must be compiled with the `-Dusedtrace` option in order to make use of the provided probes. While DTrace aims to have no overhead when its instrumentation is not active, Perl's support itself cannot uphold that guarantee, so it is built without DTrace probes under most systems. One notable exception is that Mac OS X ships a */usr/bin/perl* with DTrace support enabled.

HISTORY

5.10.1

Perl's initial DTrace support was added, providing `sub-entry` and `sub-return` probes.

5.14.0

The `sub-entry` and `sub-return` probes gain a fourth argument: the package name of the function.

5.16.0

The `phase-change` probe was added.

5.18.0

The `op-entry`, `loading-file`, and `loaded-file` probes were added.

PROBES

sub-entry(SUBNAME, FILE, LINE, PACKAGE)

Traces the entry of any subroutine. Note that all of the variables refer to the subroutine that is being invoked; there is currently no way to get ahold of any information about the subroutine's *caller* from a DTrace action.

```
 :*perl*::sub-entry {
     printf("%s::%s entered at %s line %d\n",
           copyinstr(arg3), copyinstr(arg0), copyinstr(arg1), arg2);
 }
```

sub-return(SUBNAME, FILE, LINE, PACKAGE)

Traces the exit of any subroutine. Note that all of the variables refer to the subroutine that is returning; there is currently no way to get ahold of any information about the subroutine's *caller* from a DTrace action.

```
:*perl*::sub-return {
    printf("%s::%s returned at %s line %d\n",
          copyinstr(arg3), copyinstr(arg0), copyinstr(arg1), arg2);
}
```

phase-change(NEWPHASE, OLDPHASE)

Traces changes to Perl's interpreter state. You can internalize this as tracing changes to Perl's `${^GLOBAL_PHASE}` variable, especially since the values for `NEWPHASE` and `OLDPHASE` are the strings that `${^GLOBAL_PHASE}` reports.

```
:*perl*::phase-change {
    printf("Phase changed from %s to %s\n",
        copyinstr(arg1), copyinstr(arg0));
}
```

op-entry(OPNAME)

Traces the execution of each opcode in the Perl runloop. This probe is fired before the opcode is executed. When the Perl debugger is enabled, the DTrace probe is fired *after* the debugger hooks (but still before the opcode itself is executed).

```
:*perl*::op-entry {
    printf("About to execute opcode %s\n", copyinstr(arg0));
}
```

loading-file(FILENAME)

Fires when Perl is about to load an individual file, whether from `use`, `require`, or `do`. This probe fires before the file is read from disk. The filename argument is converted to local filesystem paths instead of providing `Module::Name`-style names.

```
:*perl*:loading-file {
    printf("About to load %s\n", copyinstr(arg0));
}
```

loaded-file(FILENAME)

Fires when Perl has successfully loaded an individual file, whether from `use`, `require`, or `do`. This probe fires after the file is read from disk and its contents evaluated. The filename argument is converted to local filesystem paths instead of providing `Module::Name`-style names.

```
:*perl*:loaded-file {
    printf("Successfully loaded %s\n", copyinstr(arg0));
}
```

EXAMPLES

Most frequently called functions

```
 # dtrace -qZn 'sub-entry {
@[strjoin(strjoin(copyinstr(arg3),"::"),copyinstr(arg0))] = count() }
```

```
END {trunc(@, 10)}'

Class::MOP::Attribute::slots                                   400
Try::Tiny::catch                                               411
Try::Tiny::try                                                 411
Class::MOP::Instance::inline_slot_access                       451
Class::MOP::Class::Immutable::Trait:::around                   472
Class::MOP::Mixin::AttributeCore::has_initializer              496
Class::MOP::Method::Wrapped::__ANON__                          544
Class::MOP::Package::_package_stash                            737
Class::MOP::Class::initialize                                 1128
Class::MOP::get_metaclass_by_name                             1204
```

Trace function calls

```
# dtrace -qFZn 'sub-entry, sub-return { trace(copyinstr(arg0)) }'

0  -> Perl_pp_entersub                        BEGIN
0  <- Perl_pp_leavesub                        BEGIN
0  -> Perl_pp_entersub                        BEGIN
0    -> Perl_pp_entersub                      import
0    <- Perl_pp_leavesub                      import
0  <- Perl_pp_leavesub                        BEGIN
0  -> Perl_pp_entersub                        BEGIN
0    -> Perl_pp_entersub                      dress
0    <- Perl_pp_leavesub                      dress
0    -> Perl_pp_entersub                      dirty
0    <- Perl_pp_leavesub                      dirty
0    -> Perl_pp_entersub                      whiten
0    <- Perl_pp_leavesub                      whiten
0  <- Perl_dounwind                           BEGIN
```

Function calls during interpreter cleanup

```
# dtrace -Zn 'phase-change /copyinstr(arg0) == "END"/ { self->ending
= 1 } sub-entry /self->ending/ { trace(copyinstr(arg0)) }'

CPU     ID                    FUNCTION:NAME
  1  77214       Perl_pp_entersub:sub-entry   END
  1  77214       Perl_pp_entersub:sub-entry   END
  1  77214       Perl_pp_entersub:sub-entry   cleanup
  1  77214       Perl_pp_entersub:sub-entry   _force_writable
  1  77214       Perl_pp_entersub:sub-entry   _force_writable
```

System calls at compile time

```
 # dtrace -qZn 'phase-change /copyinstr(arg0) == "START"/ {
self->interesting = 1 } phase-change /copyinstr(arg0) == "RUN"/ {
self->interesting = 0 } syscall::: /self->interesting/ { @[probefunc]
 = count() } END { trunc(@, 3) }'

 lseek                                                         310
 read                                                          374
 stat64                                                       1056
```

Perl functions that execute the most opcodes

```
 # dtrace -qZn 'sub-entry { self->fqn = strjoin(copyinstr(arg3),
strjoin("::", copyinstr(arg0))) } op-entry /self->fqn != ""/ {
@[self->fqn] = count() } END { trunc(@, 3) }'

 warnings::unimport                                              4589
 Exporter::Heavy::_rebuild_cache                                 5039
 Exporter::import                                               14578
```

REFERENCES

DTrace Dynamic Tracing Guide

http://dtrace.org/guide/preface.html

DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD

http://www.amazon.com/DTrace-Dynamic-Tracing-Solaris-FreeBSD/dp/0132091518/

SEE ALSO

Devel::DTrace::Provider

This CPAN module lets you create application-level DTrace probes written in Perl.

AUTHORS

Shawn M Moore `sartak@gmail.com`