Ruby trunk - Feature #15022

Oneshot coverage

08/24/2018 01:28 PM - mame (Yusuke Endoh)

Status:	Closed	
Priority:	Normal	
Assignee:	mame (Yusuke Endoh)	
Target version:	2.6	

Description

I'd like to introduce a new feature to the coverage library, namely, "oneshot coverage".

Synopsis

The following is a sample target program "test-cov.rb":

```
1: def foo
2:   :foo
3: end
4:
5: def bar
6:   :bar
7: end
```

The following program measures line coverage with oneshot mode:

```
require "coverage"

# Start the measurement of line coverage with oneshot mode
Coverage.start(oneshot_lines: true)

# Load the target file
load "test-cov.rb"

# Get all executed lines so far:
p Coverage.peek_result #=> {"/.../test-cov.rb"=>{:oneshot_lines=>[1, 5]}}
  # This means that Lines 1 (def foo) and 5 (def bar) were executed

# Clear the counters
Coverage.clear

# Run one of the target functions:
foo()

# Get newly executed lines:
p Coverage.peek_result #=> {"/.../test-cov.rb"=>{:oneshot_lines=>[2]}}
  # This means Line 2 (the body of foo) was newly executed

# Clear the counters
Coverage.clear

# Run the other target function:
bar()

# Get newly executed lines:
p Coverage.peek_result #=> {"/.../test-cov.rb"=>{:oneshot_lines=>[6]}}
  # This means Line 6 (the body of foo) was newly executed

# Clear the counters
Coverage.clear

# Again, run the target functions:
foo()
```

```
bar()

# Get newly executed lines:
p Coverage.peek_result #=> {"/.../test-cov.rb"=>{:oneshot_lines=>[]}}
  # This means that no new lines were executed
```

What this is

Traditional coverage tells us "how many times each line was executed". However, it is often enough just to know "whether each line was executed at least once, or not". In this case, the counting just bring unneeded overhead.

Oneshot coverage records only the first execution of each line, and returns line numbers of newly executed lines. It contains less information than traditional coverage, but still useful in many use cases. The hook for each line is executed just once, so after it was fired, the program can run with zero-overhead.

Why this is needed

I expect two use cases:

coverage measurement in production

In Ruby, it is difficult to determine if some code is a dead code or not. To check this, some people insert a logging code to the possibly-dead code, run it in production for a while, and check if the log is not emitted.

Oneshot coverage can be used to make this process automatic, comprehensive, and non-invasive.

CPU-intensive programs

It is known that traditional coverage measurement brings about 20x overhead at worst. It does not matter when testing IO-intensive programs (like Rails application), however, it sometimes matters for CPU-intensive programs.

Oneshot coverage brings the same (or a bit worse) overhead when each line is first executed, but brings no overhead for second and later executions.

Proposal

Oneshot coverage consists of three parts of APIs.

API 1: Coverage.start(oneshot_lines: true)

This enables the measurement of line coverage with oneshot mode.

In this mode, Coverage.peek_result and result returns the following format:

```
{ "/path/to/file.rb" => { :oneshot_lines => [an array of executed line numbers] } }
```

API 2: Coverage.clear

This clears all the internal counters of coverage, but keeps the measuring target.

```
p Coverage.peek_result #=> {"/.../test-cov.rb"=>{:oneshot_lines=>[1,5]}}
foo()
p Coverage.peek_result #=> {"/.../test-cov.rb"=>{:oneshot_lines=>[1,5,2]}}
Coverage.clear
p Coverage.peek_result #=> {"/.../test-cov.rb"=>{:oneshot_lines=>[]}}
bar()
p Coverage.peek_result #=> {"/.../test-cov.rb"=>{:oneshot_lines=>[6]}}
```

You can also use this not only for oneshot coverage, but also for traditional one:

```
Coverage.start(lines: true)
load "test-cov.rb"
p Coverage.peek_result #=> {"/.../test-cov.rb"=>{:lines=>[1,0,nil,nil,1,0,nil,nil]}}
Coverage.clear
p Coverage.peek_result #=> {"/.../test-cov.rb"=>{:lines=>[0,0,nil,nil,0,0,nil,nil]}}
```

```
foo()
p Coverage.peek_result #=> {"/.../test-cov.rb"=>{:lines=>[0,1,nil,nil,0,0,nil,nil]}}
```

I'm not fully comfortable with this API because it returns incomplete and strange result. However, there have been some requests about restarting coverage (#4796, #9572, #12480), and I think this solves a part of the problem. Actually, it is useful to take coverage per test:

```
Coverage.start(lines: true)
load "target.rb"
each_test do |test|
  Coverage.clear
  test.run
  Coverage.peek_result #=> coverage of each test
end
```

The same result can be get by subtraction between two peek_results, but it is faster and easier.

API 3: Coverage.line_stub(filename)

This is just a simple helper function that returns the "stub" of line coverage from a given source code:

```
Coverage.line_stub("test-cov.rb") #=> [0, 0, nil, nil, 0, 0, nil]
```

This is needed because oneshot coverage tells "which line was executed" but does not tell nothing about other lines.
We need to distinguish that other lines are just "not executed yet" or "not a measuing target (because it is non-significant lines like empty line)".

I don't like the name "line_stub". Counterproposal is welcome.

Benchmark

As a micro benchmark, I timed the period which it took to run a 10M-line function with three modes: no coverage measurement, oneshot_lines, and lines.

mode	1st call	2nd call
no coverage	0.035 sec	0.035 sec
oneshot_lines	0.618 sec	0.034 sec
lines	0.405 sec	0.425 sec

The first call under oneshot_lines is slow, but the second call is as fast as no coverage measurement. On the other hand, lines mode is always slow.

As a relatively bigger CPU-intensive benchmark, I run optcarrot with the three modes.

mode	time
no coverage	5.5 sec
oneshot_lines	5.5 sec
lines	22 sec

Oneshot lines mode is as fast as no coverage.

Limitation

Currently, oneshot coverage supports only line coverage. It is theoretically possible to implement it for branch coverage. But, it was difficult for me to design the API, and I think line coverage is enough in many cases.

Due to implementation limitation, traditional line coverage and oneshot one cannot be enabled simultaneously: Coverage.start(lines: true, oneshot_lines: true) raises an exception.

Associated revisions

Revision 47ea999b - 10/20/2018 05:33 AM - mame (Yusuke Endoh)

ext/coverage/: add the oneshot mode

This patch introduces "oneshot_lines" mode for Coverage.start, which
checks "whether each line was executed at least once or not", instead of
"how many times each line was executed". A hook for each line is fired
at most once, and after it is fired, the hook flag was removed; it runs
with zero overhead.

See [Feature #15022] in detail.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@65195 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 65195 - 10/20/2018 05:33 AM - mame (Yusuke Endoh)

ext/coverage/: add the oneshot mode

This patch introduces "oneshot_lines" mode for Coverage.start, which
checks "whether each line was executed at least once or not", instead of
"how many times each line was executed". A hook for each line is fired
at most once, and after it is fired, the hook flag was removed; it runs
with zero overhead.

See [Feature #15022] in detail.

Revision 65195 - 10/20/2018 05:33 AM - mame (Yusuke Endoh)

ext/coverage/: add the oneshot mode

This patch introduces "oneshot_lines" mode for Coverage.start, which
checks "whether each line was executed at least once or not", instead of
"how many times each line was executed". A hook for each line is fired
at most once, and after it is fired, the hook flag was removed; it runs
with zero overhead.

See [Feature #15022] in detail.

History

#1 - 08/27/2018 04:22 AM - ioquatix (Samuel Williams)

What about using trace points?

#2 - 08/27/2018 04:23 AM - ioquatix (Samuel Williams)

Did you take a look at https://github.com/ioquatix/covered - I'd be interested in your feedback. Can we discuss further, and also how to improve performance? There is some more discussion here: https://bugs.ruby-lang.org/issues/14888

#3 - 09/13/2018 08:33 PM - shevegen (Robert A. Heiler)

I think the consensus at the last ruby developer meeting in September 2018 was to go for
it (https://docs.google.com/document/d/1RgKID1guTYC6AbhCQxs_bRiViyefIbFefA--KwLEx10/edit
e. g. "all: let's go" as summary note).

#4 - 10/20/2018 05:33 AM - mame (Yusuke Endoh)

- Status changed from Assigned to Closed

Applied in changeset trunk|r65195.

---

ext/coverage/: add the oneshot mode

This patch introduces "oneshot_lines" mode for Coverage.start, which
checks "whether each line was executed at least once or not", instead of
"how many times each line was executed". A hook for each line is fired
at most once, and after it is fired, the hook flag was removed; it runs
with zero overhead.

See [Feature #15022] in detail.

#5 - 10/20/2018 05:59 AM - mame (Yusuke Endoh)

I've committed with modification of API.

At the developers' meeting, some attendees pointed out that Coverage.clear had a design flaw; Coverage.peek_result and Coverage.result causes race condition when the program runs in multi-threads, i.e., it might miss some coverage data that is counted up between the two method calls.

Instead of a newly added Coverage.clear, the final API extended Coverage.result to receive two keyword arguments: Coverage.result(stop: true, clear: true). If clear is true, it resets the coverage counter to zero. If stop is true, it disables coverage measurement. Coverage.peek_result is equal to result(stop: false, clear: false), and Coverage.clear is equal to result(stop: false, clear: true).

#6 - 10/20/2018 06:07 AM - mame (Yusuke Endoh)

ioquatix (Samuel Williams) wrote:

> Did you take a look at https://github.com/ioquatix/covered - I'd be interested in your feedback. Can we discuss further, and also how to improve performance? There is some more discussion here: https://bugs.ruby-lang.org/issues/14888

I think that the approach is the same as deep-cover. It looks smarter and more configurable. It would be a better choice to measure coverage when testing, though I'm not fully sure if the instrumentation causes no practical problem. This patch, however, aims to provide a approach applicable to production; I believe that a less-intrusive, and less-overhead approach is preferable for this use case.

#7 - 12/26/2018 05:15 PM - danmayer (Dan Mayer)

This is great thanks mame (Yusuke Endoh), I think this will be a significant performance win when I add support for this into Coverband https://github.com/danmayer/coverband most folks only care if the line is hit on production as you said opposed to knowing total count. Also, being able to clear will help me remove some of the code related to calculating the diff between peek_result runs.

This looks like a big improvement for production code coverage. Looking forward to getting to use this.

Files

oneshot-coverage.patch	15 KB	08/24/2018	mame (Yusuke Endoh)