

なんで新しい debug.gem が必要なの？

Koichi Sasada
<ko1@cookpad.com>



cookpad

About this talk

- Introduce “debug.gem” <https://github.com/ruby/debug>
 - Newly created debugger for Ruby 2.6 and later
 - Will be bundled with Ruby 3.1 (Dec/2021)
- Demonstrate “debug.gem”
 - Basic usage instructions
 - Advanced features
- The presentation slides with the talk script is available at here: <https://www.atdot.net/~ko1/activities/>

About Koichi Sasada

- Ruby interpreter developer employed by Cookpad Inc. (2017-) with @mame
 - YARV (Ruby 1.9-)
 - Generational/Incremental GC (Ruby 2.1-)
 - Ractor (Ruby 3.0-)
 - ...
- Ruby Association Director (2012-)



What is a debugger?

- A tool to help debugging
 - To investigate the cause of problems
 - To know the program live state
 - To understand the program
- Basic features
 - CONTROL execution
 - STOP at breakpoints
 - STEP forward to the next line
 - ...
 - QUERY program status



Ruby's existing debuggers

- `lib/debug.rb`
 - `ruby -r debug script.rb`
 - Standard library, but maybe nobody uses it
- `byebug`
 - `byebug script.rb`
- `debase / ruby-debug-ide`
 - Used by IDE (rubymine, vscode, ...)

Why create yet another debugger?

- Performance
 - Existing debuggers slow with breakpoints
 - Recent TracePoint API support line-specific
- Native support for remote execution and IDE
- Native support for Ractors
- (and I like to make this kind of tools)

Introduction of “debug.gem”

<https://github.com/ruby/debug>

All information are explained in <https://github.com/ruby/debug>

debug.rb

This library provides debugging functionality to Ruby.

This debug.rb is replacement of traditional lib/debug.rb standard library which is implemented by `set_trace_func`.

New debug.rb has several advantages:

- Fast: No performance penalty on non-stepping mode and non-breakpoints.
- **Remote debugging**: Support remote debugging natively.
 - UNIX domain socket
 - TCP/IP
 - Integration with rich debugger frontend
 - VSCode/DAP ([VSCode rdbg Ruby Debugger - Visual Studio Marketplace](#))
 - Chrome DevTools
- Extensible: application can introduce debugging support with several ways:
 - By `rdbg` command
 - By loading libraries with `-r` command line option
 - By calling Ruby's method explicitly
- Misc
 - Support threads (almost done) and ractors (TODO).
 - Support suspending and entering to the console debugging with `Ctrl-C` at most of timing.
 - Show parameters on backtrace command.
 - Support recording & reply debugging.

Installation

debug.gem

- Created from scratch (2021 Feb~)
- Supports Ruby 2.6 and later
 - Utilize recent introduced APIs
- Ruby 3.1 (2021/Dec) will be shipped with debug.gem
 - Replacement with old lib/debug.rb
- Like other libraries (lib/debug.rb, byebug, gdb, lldb, ...) debug.gem provides REPL to execute debug commands

Performance

```
def fib n
  if n < 0
    raise # breakpoint
  elsif n < 2
    n
  else
    fib(n-1) + fib(n-2)
  end
end

require 'benchmark'
Benchmark.bm{ |x|
  x.report{ fib(35) }
}
```

	Without breakpoint	With breakpoint
ruby	0.93	N/A
rdbg (debug.gem)	0.92 (x0.98)	0.92 (x0.98)
byebug	1.23 (x1.32)	75.15 (x80.80)
RubyMine	0.97 (x1.04)	22.66 (x24.36)
old lib/debug.rb	221.88 (x238.58)	285.99 (x307.51)

Execution time in sec (ratio with ruby result (smaller is better))

ruby 3.0.1p64
rdbg 1.0.0.rc2
byebug 11.1.3
RubyMine 2021.2.1 w/ debase 0.2.5.beta2

Intel(R) Core(TM) i7-10810U CPU, Windows 10, WSL2

Use debug.gem

1. Use “rdbg” command

- `rdbg target.rb`
- `rdbg -c -- bin/rails`
- `rdbg -c -- bundle exec rake`

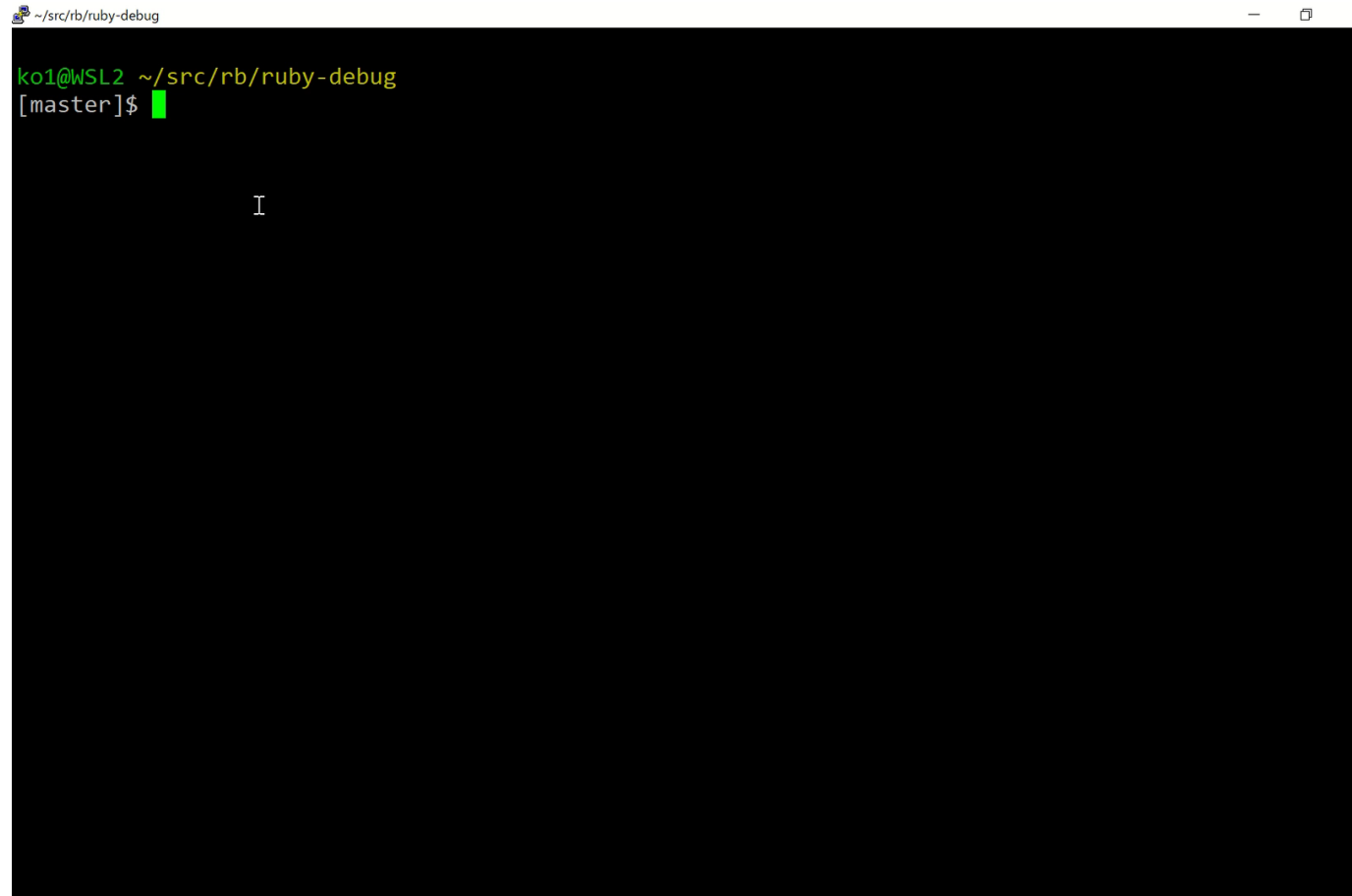
2. Load “debug.gem” in your application

- `require “debug”` (or “debug/...”, see doc)
- `gem ‘debug’` in Gemfile (and `Bundler.require`)

3. Use with IDE

- (VSCode) `.vscode/launch.json` (ruby-rdbg extension will make) and push “Start debugging” button

Demo: Basic usage



```
~/src/rb/ruby-debug  
ko1@WSL2 ~/src/rb/ruby-debug  
[master]$ █  
  
I
```

RUN AND DEBUG Debug current file app.rb record.rb basic.rb

VARIABLES

WATCH

CALL STACK

BREAKPOINTS

- rescue any exception
- rescue RuntimeError

mnt > c > ko1 > src > rb > ruby-debug > rk2011 > basic.rb

```

1 def foo a, b
2   x = a + b
3   return x ** 10
4 end
5
6 a = ['string', [:nested_array, {key: :value}]]
7 i = 10
8 j = 20
9
10 r = foo(i, j)
11
12 p r
13

```

DEBUG CONSOLE Filter (e.g. text, lexclude)

```

> ["string", [:nested_array, {key=>:value...
  > 0: "string"
  > 1: [:nested_array, {key=>:value}]
    > 0: :nested_array
    > 1: {key=>:value}
      > #class: Hash
      > :key: :value
        > #class: Symbol

```

Basic features

- Control the program execution
 - Set breakpoints
 - Step execution (step-in/over/out)
- Query the program status
 - See the source code at breakpoint
 - See the backtrace
 - Select the frame in backtrace
 - Access to variables of the specific frames
 - Evaluate an expression on the specific frame

Set a breakpoint

- Use “break” command at the beginning
 - `break 10 # break at 10 line on current file`
 - `break foo.rb:10 # break at the location`
 - `break MyClass#my_method # break at the method`
 - `catch FooException # break at FooException is raised`
 - `break ... if foo == bar # break if foo == bar`
- Write “`binding.break`” line in your program
 - You can insert it like “`binding.irb`”
 - “`binding.b`” for short and “`debugger`” like JavaScript
- Use IDEs/editors breakpoint support

Set a breakpoint (cont.)

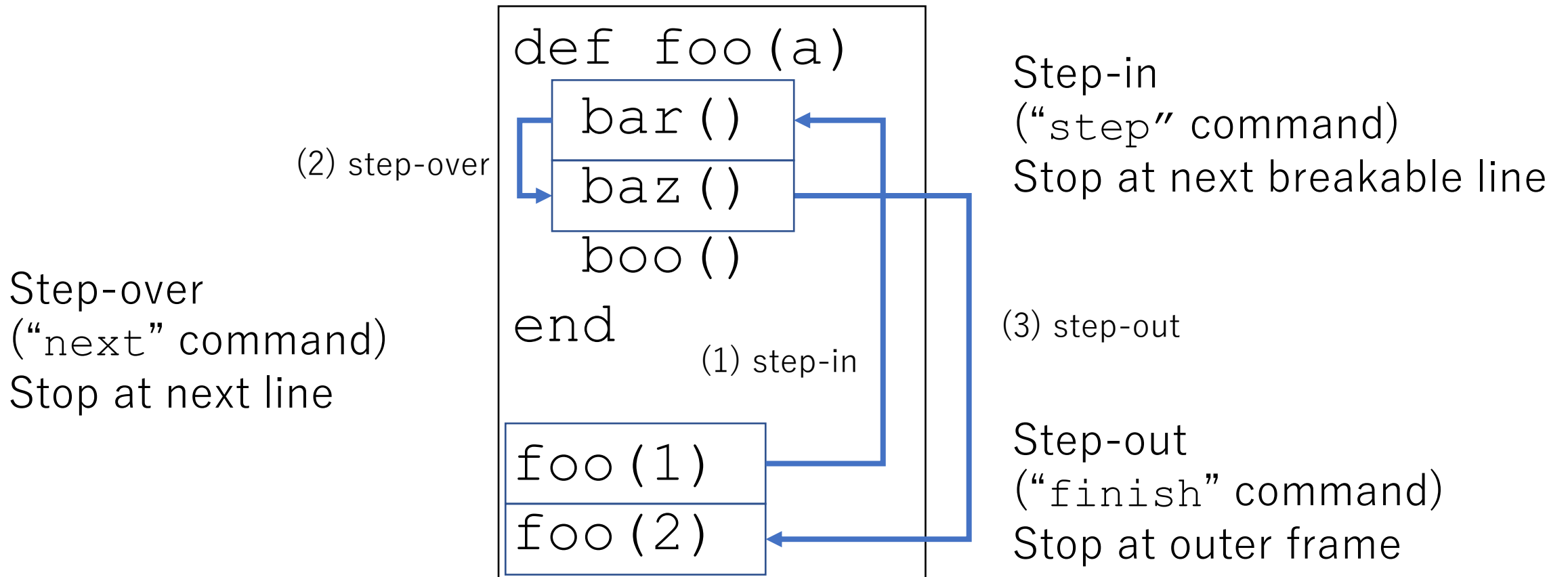
- Use “break” command at the beginning (and IDE)
 - Do not need to modify the source code
 - Cooperation with IDE/Editor (e.g. set it with F9 on VSCode)
- Write “`binding.break`” method in your program
 - Straight forward for some Ruby users

Control debugger from the program by
`binding.break do: expr`

```
# enable "trace line" feature while bar()  
def foo  
  binding.break do: 'trace line'  
  bar()  
  binding.break do: 'trace off line'  
end
```

Step execution

Step-in, Step-over, Step-out



Access to the local variables in the specific frame

- See the backtrace with “backtrace”
- Select the frame
 - “frame <num>”
 - “up” / “down” to select upper/lower
- Access to the frame local variables
 - “outline” command and “info” command for overview
 - “p <expr>” and “pp <expr>”

```
(rdbg) p a ** 10 # command  
=> 25937424601
```

```
[1, 6] in target.rb  
1|  
2| def foo(a) = bar(a+1)  
=> 3| def bar(a) = a * 2  
4| foo 10  
5|  
6| END  
=>#0 Object#bar(a=11) at target.rb:3 #=> 22  
#1 Object#foo(a=10) at target.rb:2  
# and 1 frames (use `bt' command for all frames)
```

```
(rdbg) outline # command  
Object.methods: inspect to_s  
locals: a
```

```
(rdbg) info # c  
%self = main  
%return = 22  
a = 11  
(rdbg) backtrace
```

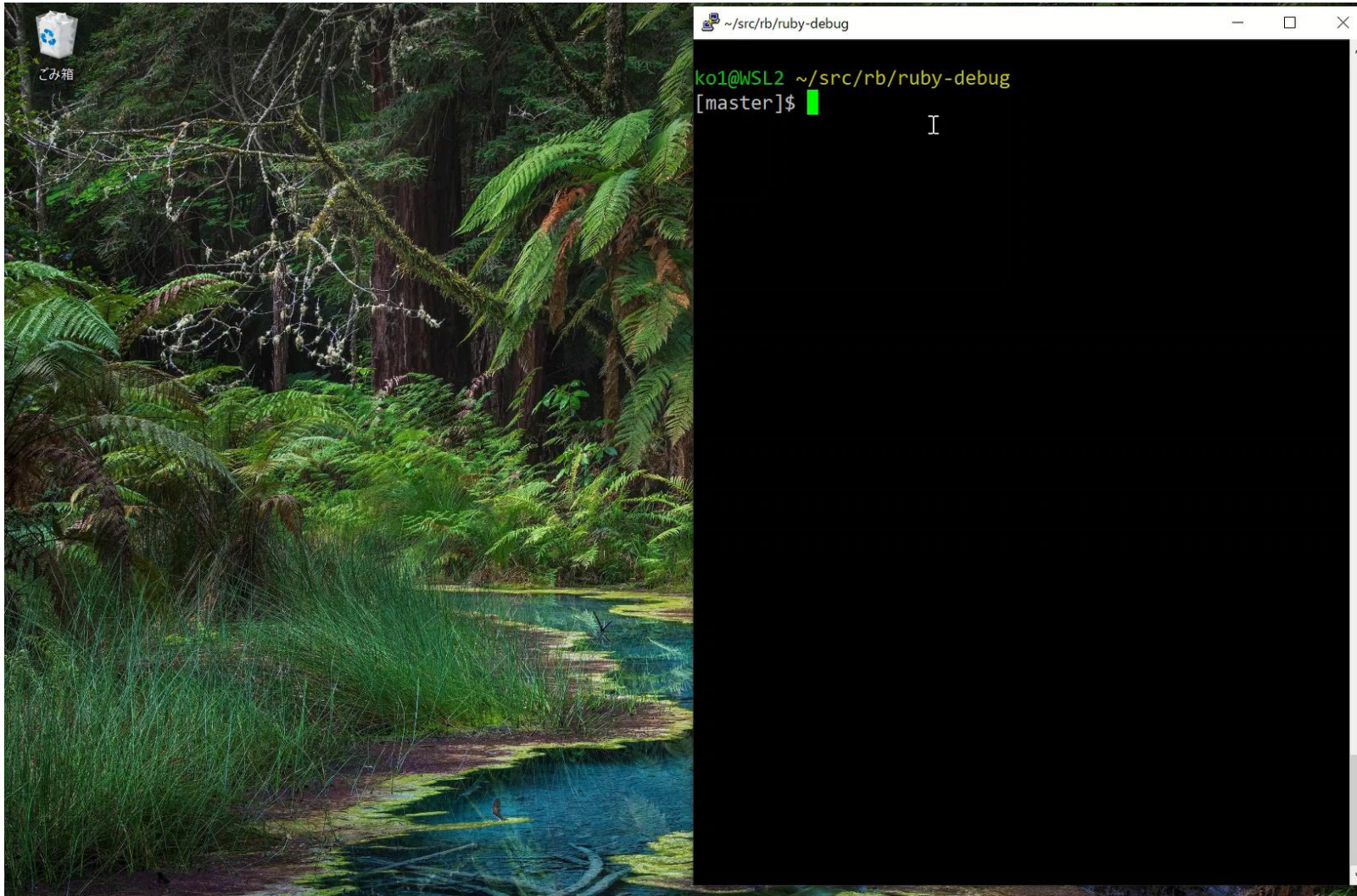
Advanced features

- [demo] Pause with “Ctrl-C” or when attaching the debugger
- [demo] VSCode/Chrome browser seamless integration
- [demo] Remote debugging
- [demo] Postmortem debugging
- [demo] Record and replay debugging
- [demo] Multi-process debugging
- [demo?] Event tracing

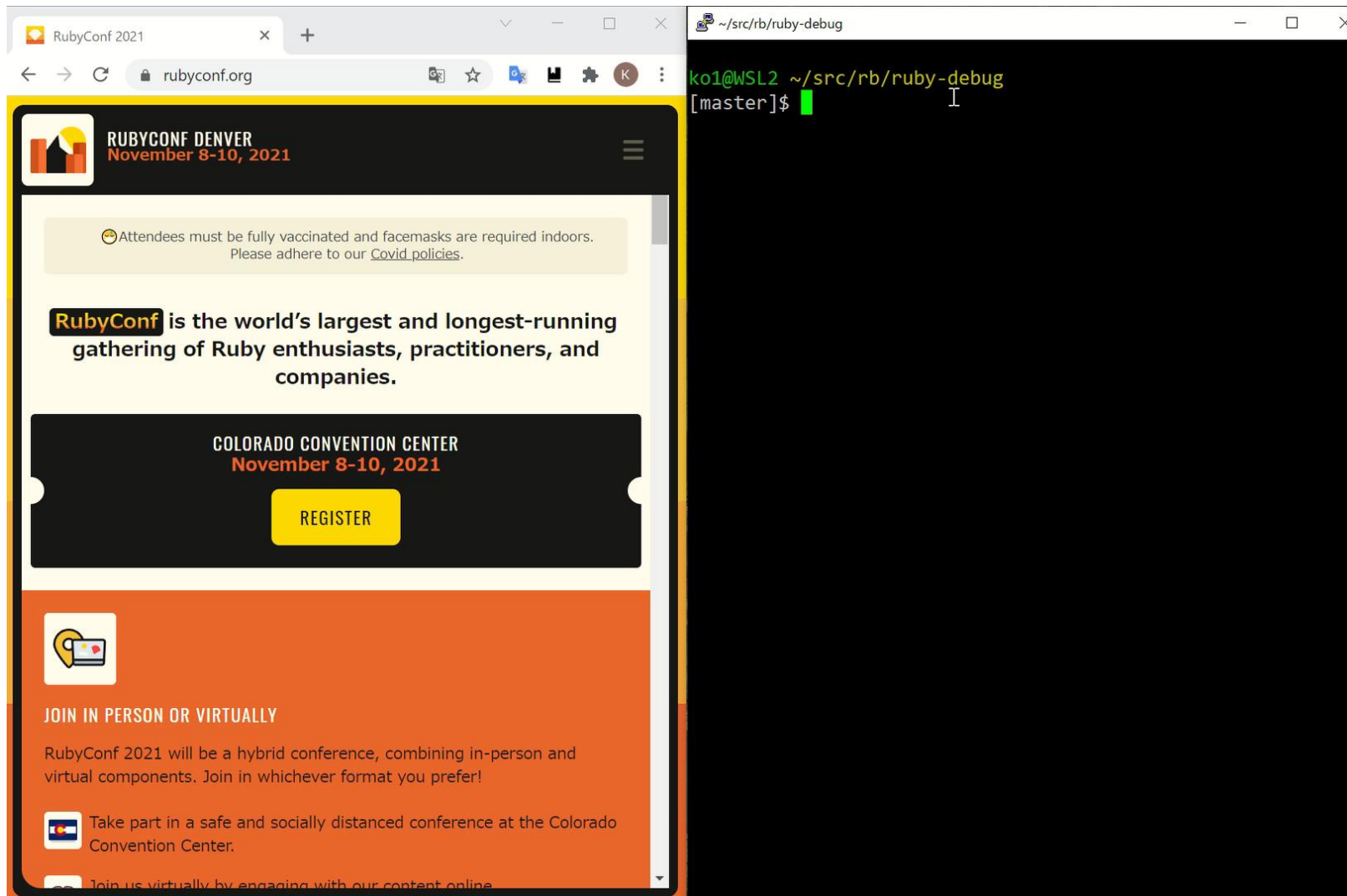
Demo: Pause with “Ctrl-C” or when attaching the debugger

- You can see the current execution status with the debugger

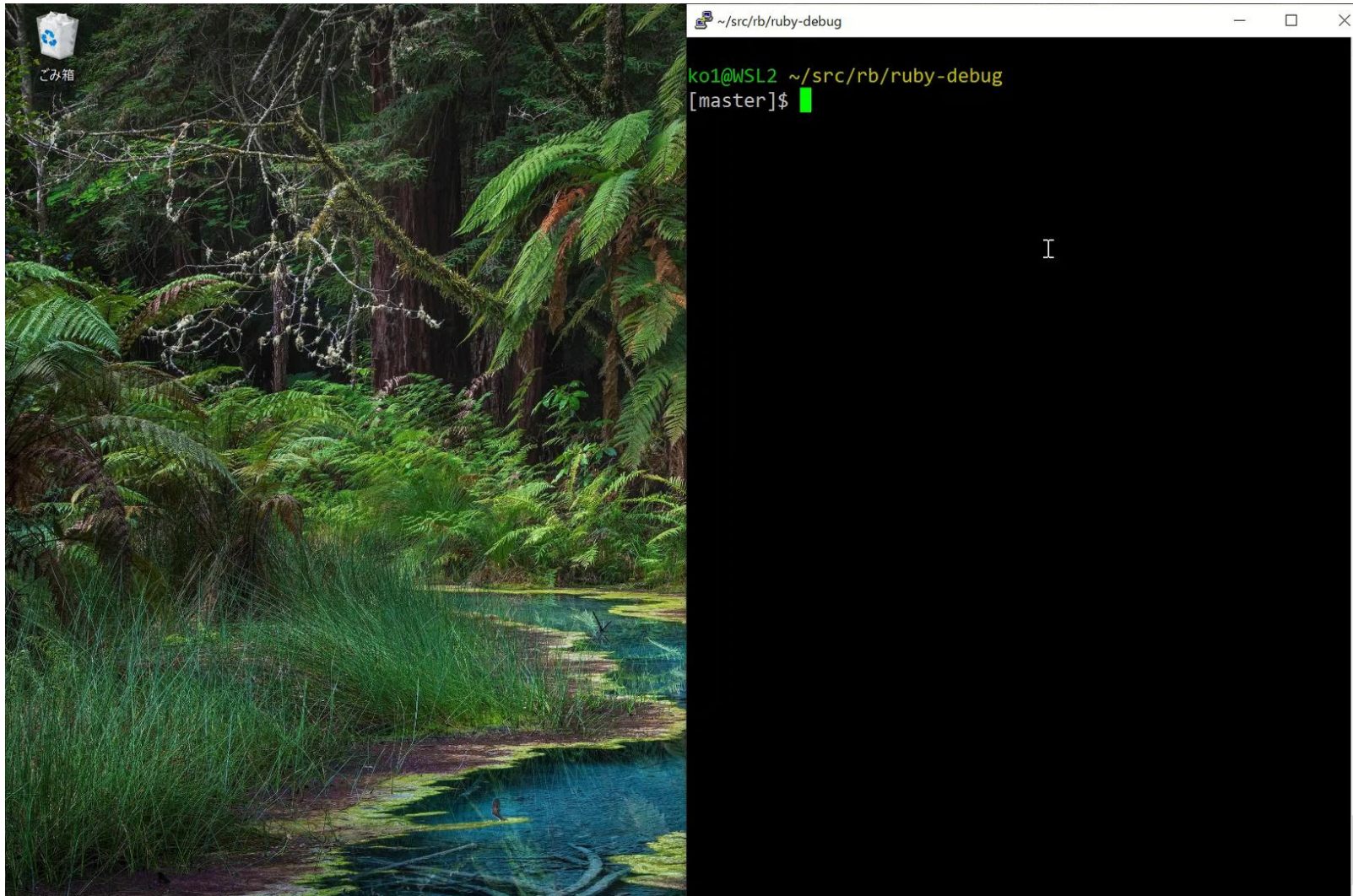
Demo: Seamless integration with VSCode/Chrome browser



Demo: Seamless integration with VSCode/Chrome browser



Demo: Start VSCode for debugger frontend



Demo: Remote debugging

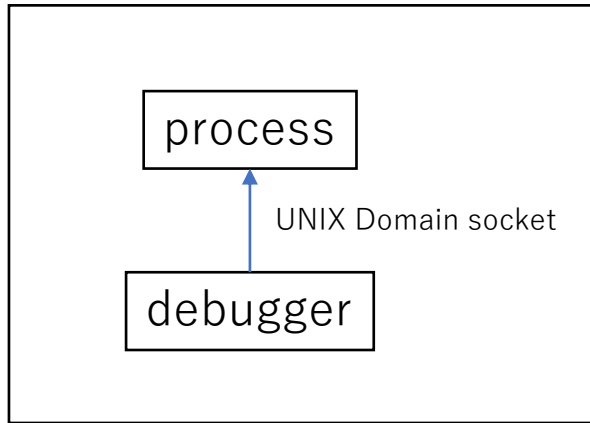
Connect over network

- Easy to open remote debug port and attach
 - `rdbg --open script.rb` (or `rdbg -O`)
 - Run program with opening debug port
 - `require 'debug/open'` # in script
 - `rdbg -attach` (or `rdbg -A`)
 - Access to debug port
- Debug no TTY attached processes
 - Daemon processes
 - Redirecting by shell's pipe
- Query the process status like `sigdump` but more details

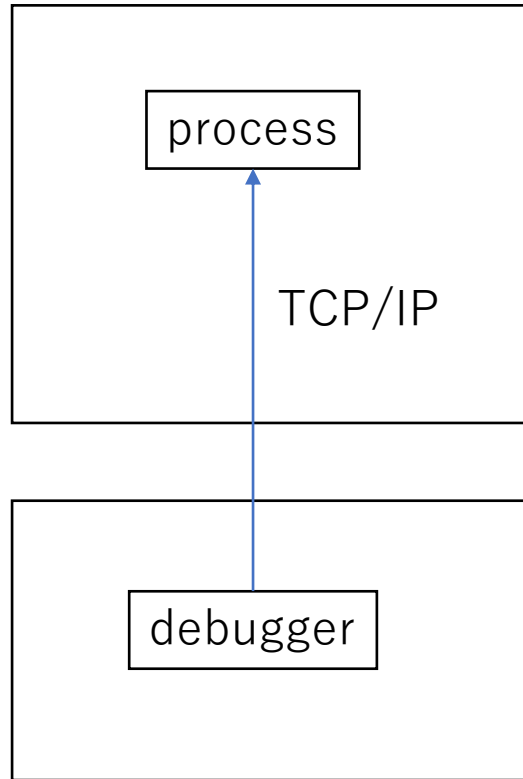
```
ko1@WSL2 ~/src/rb/ruby-debug/rk2011  
[master]$
```

```
ko1@WSL2 ~/src/rb/ruby-debug/rk2011  
[master]$
```

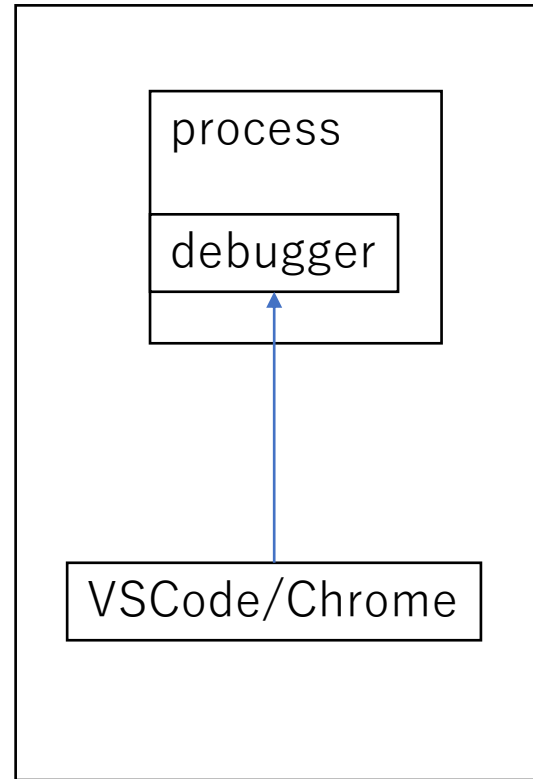
Same machine



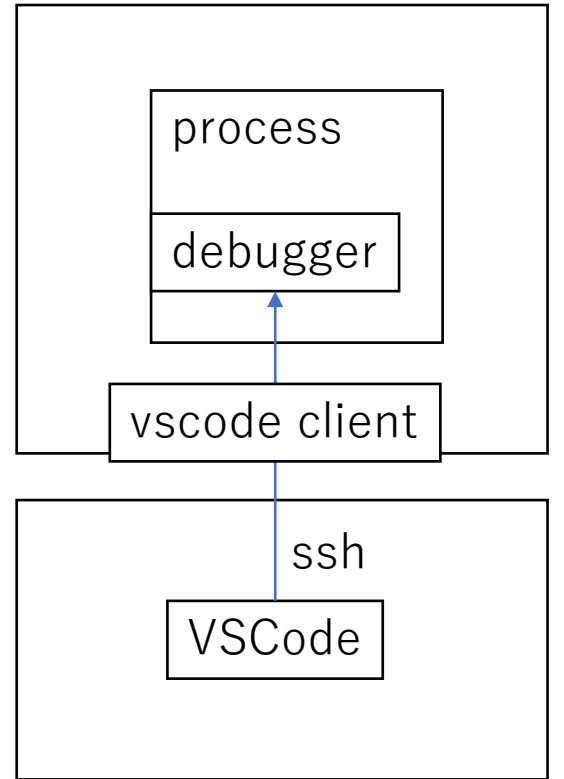
Cross machine



VSCoDe/Chrome



VSCoDe
cross machine

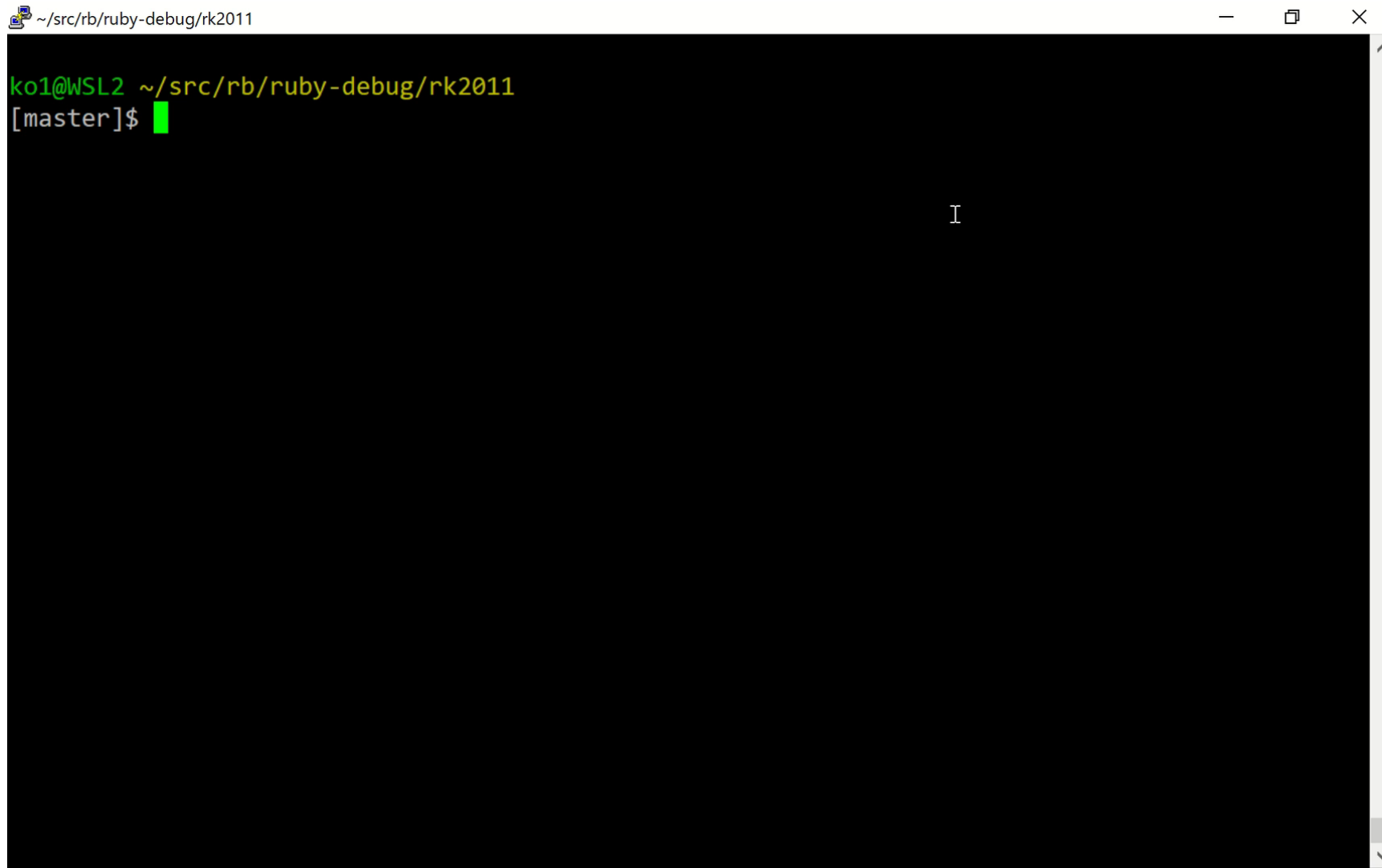


Demo: Postmortem debugging Debug dead Ruby process

A terminal window with a black background and white text. The window title is "~/src/rb/ruby-debug/rk2011". The prompt is "ko1@WSL2 ~/src/rb/ruby-debug/rk2011" and the current command is "[master]\$".

```
~/src/rb/ruby-debug/rk2011  
ko1@WSL2 ~/src/rb/ruby-debug/rk2011  
[master]$
```

Demo: Record and replay debugging Backward stepping execution



A terminal window with a black background and white text. The window title is `~/src/rb/ruby-debug/rk2011`. The prompt shows `ko1@WSL2 ~/src/rb/ruby-debug/rk2011` and `[master]$` followed by a green cursor. A single character `I` is visible in the center of the terminal.

The image shows a Visual Studio Code window with the following components:

- Menu Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- File Explorer:** Shows three files: app.rb, record.rb (selected), and basic.rb.
- Debugger View (Left):**
 - VARIABLES:** Empty.
 - WATCH:** Empty.
 - CALL STACK:** Empty.
 - BREAKPOINTS:** One breakpoint is set at line 3. Options: rescue any exception, rescue RuntimeError.
- Code Editor:** Contains the following Ruby code:

```
mnt > c > ko1 > src > rb > ruby-debug > rk2011 > record.rb
1 binding.b do: 'record on'
2
3 def foo a, b
4   x = a + b
5   return x ** 10
6 end
7
8 i = 10
9 j = 20
10
11 r = foo(i, j)
12
13 p r
14
```
- Terminal (Bottom):** Shows the rdbg connection process:

```
PROBLEMS OUTPUT TERMINAL ... 2: rdbg
DEBUGGER: Debugger can attach via UNIX domain socket (/home/ko1/.ruby-d
ebug-sock/ruby-debug-ko1-8325)
DEBUGGER: wait for debugger connection...
DEBUGGER: Connected.
5904900000000000
DEBUGGER: Disconnected.

ko1@WSL2 ~/src/rb/ruby-debug/rk2011
[master]$
```
- Status Bar (Bottom):** WSL: Ubuntu-20.04, master*, 0 errors, 0 warnings, Debug current file with rdbg (rk2011), Ln 3, Col 13, Spaces: 2, UTF-8, CRLF, Ruby.

Demo: Multi-process debugging

- You can debug multiple processes (fork family) with one debugger
 - Prompt shows which process
 - Only one process can be operated by the debug console at the same time

Acknowledgements

- Naoto Ono san (@ono-max) implements test-frameworks for the debugger and Chrome browser support. The part of works were done in GSoC project.
- Stan Lo san (@st0012) submits tremendous patches to improve the debugger usability such as coloring and so on based on his debugger trials. Also, he makes many tests for the debugger.
- Ruby committers helps me to design and implement the debugger

Conclusion

- “debug.gem” is newly created Ruby debugger from scratch
 - Faster.
 - Modern UI.
 - Many useful features.
- “`gem install debug`” now!
 - And give us your feedback.
 - I love to introduce the debugger on your meetup, please contact me.
- Ractor supports is not available, now working on.

Thank you for your listening!

なんで新しい
debug.gem が必要なの？

Koichi Sasada

<ko1@cookpad.com>



cookpad