Ruby.inspect

Koichi Sasada

<ko1@heroku.com>

"Ruby.inspect" by Koichi Sasada, RDRC2014
Summary.inspect

• Introduction of new Ruby
  • Stable 2.1
  • Next version of 2.2

• How to inspect your application behavior
  • With tools & services
  • Make a tools by inspection primitives
  • Inspection from outside
“Today’s Message”.inspect

Become a Low-level engineer
(sometimes)

"Ruby.inspect" by Koichi Sasada, RDRC2014
Ko1.inspect
#=> <Ko1: @name="“Koichi Sasada””>

• Koichi Sasada a.k.a. ko1
• From Japan
• 笹田 (family name) 耕一 (given name) in Kanji character
  • “Ichi” (Kanji character “一”) means “1” or first
  • This naming rule represents I’m the first son of my parents
• Ko”ichi” → ko1

"Ruby.inspect" by Koichi Sasada, RDRC2014
Ko1.inspect
 #=> <Ko1: @job="Programmer">

• CRuby/MRI committer
  • Virtual machine (YARV) from Ruby 1.9
  • YARV development since 2004/1/1
  • Recently, improving GC performance

• Matz team at Heroku, Inc.
  • Full-time CRuby developer
  • Working in Japan

• Director of Ruby Association

"Ruby.inspect" by Koichi Sasada, RDRC2014
The Ruby Association was founded to further development of the programming language Ruby. The goals of the Ruby Association are to improve relationship between Ruby-related projects, communities and businesses, and to address issues connected with using Ruby in an enterprise environment.

Quoted from [http://www.ruby.or.jp/en/](http://www.ruby.or.jp/en/)
• Foundation to encourage Ruby dev. and communities

• Activities
  • Ruby programmer certification program
  • Grant project. We have selected 3 proposals in 2013
  • Ruby Prize
    • To recognize the efforts of “New members” to the Ruby community
  • Maintenance of Ruby (Cruby) interpreter
    • Now, it is for Ruby 2.0.0
  • Events, especially RubyWorld Conference
• **Donation** for Ruby developments and communities

"Ruby.inspect" by Koichi Sasada, RDRC2014
• Heroku, Inc.  http://www.heroku.com

You should know about Heroku!!

"Ruby.inspect" by Koichi Sasada, RDRC2014
• Heroku, Inc.  [http://www.heroku.com](http://www.heroku.com)

• Heroku supports OSSs / Ruby development
  • Many talents for Ruby, and also other languages
  • Heroku employs 3 **Ruby interpreter core developers**
    • Matz
    • Nobu
    • Ko1 (me)
  • We name our group “Matz team”
“Matz team”.inspect

Matz @ Shimane
Title collector

Nobu @ Tochigi
Patch monster

ko1 @ Tokyo
EDD developer

"Ruby.inspect" by Koichi Sasada, RDRC2014
Matz.inspect

#=> Title collector

- He has so many (job) title
  - Chairman - Ruby Association
  - Fellow - NaCl
  - Chief architect, Ruby - Heroku
  - Research institute fellow – Rakuten
  - Chairman – NPO mruby Forum
  - Senior researcher – Kadokawa Ascii Research Lab
  - Visiting professor – Shimane University
  - Honorable citizen (living) – Matsue city
  - Honorable member – Nihon Ruby no Kai
  - …

- This margin is too narrow to contain
Nobu.inspect

 #=> Patch monster

• Great patch creator
Nobu is Great Patch Monster

Commit ratio in last 5 years

- nobu: 29%
- akr: 12%
- svn: 9%
- naruse: 8%
- usa: 4%
- ko1: 4%
- drbrain: 3%
- naruse: 3%
- tenderlove: 2%
- zzak: 2%
- kkou: 2%
- kkou: 2%
- kosaki: 2%
- usa: 4%
- nagachika: 1%
- mehr: 1%
- yuji: 1%
- shou: 1%
- shugo: 1%
- nari: 0%
- shouhei: 1%
- nagai: 0%
- eregon: 0%
- ngoto: 0%
- wanabe: 0%
- azav: 0%
- keiju: 0%
- suke: 0%
- keiji: 0%
- duerst: 0%
- takano32: 0%
- luislavena: 0%
- jeg2: 0%
- hsbt: 0%
-arton: 0%
-seki: 0%
- kanemoto: 0%
-tmm1: 0%
-eban: 0%
-muraken: 0%
-headius: 0%
-evan: 0%
-a_matsuda: 0%
-iwamatsu: 0%
-technorama: 0%
-davidflanagan: 0%
-gotoken: 0%
-okkez: 0%
"Ruby.inspect" by Koichi Sasada, RDRC2014
Ko1.inspect

#=> EDD developer

Commit number of ko1 (last 3 years)

EDD: Event Driven Development
“Mission of Matz team”. inspect

• **Improve quality of next version of CRuby**
  • Matz decides a spec finally
  • Nobu fixed huge number of bugs
  • Ko1 improves the performance
“Ruby 2.1”.inspect
#=> Current stable

http://www.flickr.com/photos/loginesta/5266114104
“Ruby 2.1”.inspect
 #=> a bit old Ruby

- **Ruby 2.1.0** was released at **2013/12/25**
  - New features
  - Performance improvements
- **Ruby 2.1.1** was released at **2014/02/24**
  - Includes many bug fixes found after 2.1.0 release
  - Introduce a new GC tuning parameter to change generational GC behavior (introduce it later)
- **Ruby 2.1.2** was released at **2014/05/09**
  - Solves critical bugs (OpenSSL and so on)
Ruby 2.1 the biggest change

Version policy

• Change the versioning policy
  • Drop “patch level” in the version
  • Teeny represents patch level
    • Release new teeny versions about every 3 month
    • Teeny upgrades keep compatibility
  • Minor upgrades can break backward compatibility
    • We make an effort to keep compatibility
      (recently. Remember Ruby 1.9 😊)
Ruby 2.1 New syntax

• New syntaxes
  • Required keyword parameter
  • Rational number literal
  • Complex number literal
  • `def` returns symbol of method name

http://www.flickr.com/photos/rooreynolds/4133549889

"Ruby.inspect" by Koichi Sasada, RDRC2014
Ruby 2.1 Syntax

Required keyword parameter

• Keyword argument (from Ruby 2.0.0)
  • def foo(a: 1, b: 2); end
  • `a’ and `b’ are optional parameters
  • OK: foo(); foo(a: 1); foo(a: 1, b: 2); foo(b: 2)

• Required keyword argument from 2.1
  • def foo(a: 1, b: )
  • `a’ is optional, but `b’ is required parameter
  • OK: foo(a: 1, b: 2); foo(b: 2)
  • NG: foo(); foo(a: 1)
Ruby 2.1 Syntax
Rational number literals

• To represent \( \frac{1}{2} \), in Ruby “Rational(1, 2)” → Too long!!
• Introduce “r” suffix
  \( \frac{1}{2} \rightarrow 1/2r \)
• “[digits]r” represents “Rational([digits], 1)”
• \( \frac{1}{2} \rightarrow 1/2r \)
  • 1/2r \rightarrow 1/Rational(2, 1)
  • 1/Rational(2, 1) \rightarrow \text{Rational}(1/2)
Ruby 2.1 Syntax
Complex number literals

• We already have “Integer#i” method to make imaginary number like “1+2.i”
• We already introduced “r” suffix for Rational → No reason to prohibit “i” suffix!!
• [digits]i represents “Complex(0, [digits])”
• 1+2i #=> 1+Complex(0, 2)
• 1+Complex(0, 2) #=> Complex(1, 2)

• You can mix “r” and “i” suffix
Ruby 2.1 Syntax
Return value of `def` syntax

• Return value of method definition
  • Method definition syntax returns symbol of defined method name
  • `def foo; ...; end` #=> :foo

• Method modifier methods
  • Example:
    • private def foo; ...; end
    • public static void def main(args); ...; end
Ruby 2.1 Runtime new features

• String#scrub
• Process.clock_gettime
• Binding#local_variable_get/set
• Bignum now uses GMP (if available)
• Extending ObjectSpace

"Ruby.inspect" by Koichi Sasada, RDRC2014
Performance improvements

• Optimize “string literal”.freeze
• Sophisticated inline method cache
• Introducing Generational GC: RGenGC

"Ruby.inspect" by Koichi Sasada, RDRC2014
RGenGC: Generational GC for Ruby

• RGenGC: Restricted Generational GC
  • Generational GC (minor/major GC uses M&S)
  • **Dramatically speedup for GC-bottleneck applications**
  • New generational GC algorithm allows mixing “Write-barrier protected objects” and “WB unprotected objects”
    → No (mostly) **compatibility issue** with C-exts

• Inserting WBs gradually
  • We can concentrate WB insertion efforts for major objects and major methods
  • Now, most of objects (such as Array, Hash, String, etc.) are WB protected
    • Array, Hash, Object, String objects are very popular in Ruby
    • Array objects using **RARRAY_PTR() change to WB unprotected** objects (called as Shady objects), so existing codes still works.
RGenGC
Performance evaluation (RDoc)

About x15 speedup!

"Ruby.inspect" by Koichi Sasada, RDRC2014

* Disabled lazy sweep to measure correctly.
RGenGC Performance evaluation (RDoc)

* 12% improvements compare with w/ and w/o RGenGC
* Disabled lazy sweep to measure correctly.
"Ruby 2.2".inspect
 #=> Next version
Schedule of Ruby 2.2

• Not published officially
• Schedule draft is available by Naruse-san
Ruby 2.2 schedule

2013/12
Ruby 2.1.0

2014/12/25
Ruby 2.2.0

Events are important for
EDD (Event Driven Development) Developers

"Ruby.inspect" by Koichi Sasada, RDRC2014
2.2 big features (planned)

• New syntax: not available now
• New method: no notable methods available now
• Libraries:
  • Minitest and test/unit will be removed (provided by bundled gem)
2.2 internal changes

• Internal
  • C APIs
    • Hide internal structures for Hash, Struct and so on
    • Remove obsolete APIs
  • GC
    • Symbol GC (merged recently)
    • 2age promotion strategy for RGenGC
    • Incremental GC to reduce major GC pause time
  • VM
    • More sophisticated method cache

"Ruby.inspect" by Koichi Sasada, RDRC2014
Symbol GC

• Symbols remain forever → Security issue
  • “n.times{|i| i.to_s.to_sym}”
    creates “n” symbols and they are never collected
• Symbol GC: Collect dynamically created symbols
Break

http://www.flickr.com/photos/donkeyhotey/8422065722

"Ruby.inspect" by Koichi Sasada, RDRC2014
Ruby.inspect

https://www.flickr.com/photos/theloushe/4640871734/
"Ruby.inspect" by Koichi Sasada, RDRC2014
Inspecting Ruby

• You may want to know “what happen?” on your application
• Ruby has many “inspecting” features to see applications behavior
  • Some features are supported only by MRI/CRuby

"Ruby.inspect" by Koichi Sasada, RDRC2014
Why “inspect” is needed?

• Code reading
• Debugging
• Performance tuning
• Understanding Ruby’s implementation
• ...

"Ruby.inspect" by Koichi Sasada, RDRC2014
How to inspect your app?

• Use “Tools and services” for Ruby
• Make tools with “Standard inspect features”
• Inspect Ruby process itself from outside

"Ruby.inspect" by Koichi Sasada, RDRC2014
Inspection features on computer layers

Your app

Libraries/Tools

Ruby interpreter

Tools & Services

Primitives

Operating System

Hardware

"Ruby.inspect" by Koichi Sasada, RDRC2014
Tools & Services

https://www.flickr.com/photos/bunnyrel/9015937323

"Ruby.inspect" by Koichi Sasada, RDRC2014
Tools & Services

- Your app
- Libraries/Tools
- Ruby interpreter

Operating System

Hardware

"Ruby.inspect" by Koichi Sasada, RDRC2014
Tools & Services

• Benchmarking
  • benchmark
  • benchmark/ips

• Profiling
  • [Time] ruby-prof (deterministic profiler)
  • [Time] perftools.rb, stackprof, rblineprof (sampling profilers)
  • [Memory] GCTracer, AllocationTracer, ...
  • [Total] NewRelic

• Debugging
  • ruby-debug
  • byebug (2.0~)
  • tracer (standard library)

"Ruby.inspect" by Koichi Sasada, RDRC2014
New Relic

• “Dive into Ruby VM Stats with New Relic”
  http://blog.newrelic.com/2014/04/23/ruby-vm-stats/

• “Ruby VM measurements”
  https://docs.newrelic.com/docs/ruby/ruby-vm-stats

IMPORTANT
You can use New Relic very easily on Heroku as an Add-on

"Ruby.inspect" by Koichi Sasada, RDRC2014
Tools & Services

You can find manuals for tools! Enjoy!

• “Debugging Ruby Performance” by Aman Gupta will help you to survey

https://speakerdeck.com/tmm1/debugging-ruby-performance
Ruby’s Inspection primitives
How to make inspection tools?

https://www.flickr.com/photos/fiddleoak/6691220069/

"Ruby.inspect" by Koichi Sasada, RDRC2014
Inspection features on computer layers

- Your app
- Libraries/Tools
- Ruby interpreter
- Primitives

Operating System

Hardware

"Ruby.inspect" by Koichi Sasada, RDRC2014
Ruby’s Inspection primitives

• Show object
• Reflections
• Statistics
• Tracing
Show objects
Kernel#p and pp library

• Debug print
  • Kernel#p(obj): print result of “obj.inspect”
  • pp: print pretty printed result
  • Both print onto STDOUT
  • You can modify Object#inspect for better representation

• Everyone love to use 😊
  • Traditional “printf” debug

"Ruby.inspect" by Koichi Sasada, RDRC2014
Show objects
Kernel#p and pp library

• Tips
  • Use p() method with keyword argument
    foo=[1, 2]; bar={a: 1, b: ['bar']}
    p foo: foo, bar: bar
    #=> {foo=>[1, 2],
         bar=>{:a=>1, :b=>['bar']}}

  • PP.pp(obj, STDERR) prints onto STDERR, not STDOUT

"Ruby.inspect" by Koichi Sasada, RDRC2014
Show objects

ObjectSpace::dump(obj)

• Dump the contents of a ruby object as JSON
  • Not for serializing, but for seeking internal “implementation specific” information

• ObjectSpace::dump_all() dumps all objects and relations
  • It will help us to find out memory leak (unexpected relation to prevent GC collection)

• Introduced from Ruby 2.1

"Ruby.inspect" by Koichi Sasada, RDRC2014
Reflections

https://www.flickr.com/photos/cbpphotos/11934804573
"Ruby.inspect" by Koichi Sasada, RDRC2014
Reflections

• Stack trace
  • caller, caller_locations
  • Thread#backtrace, Thread#backtrace_locations

• Access variables
  • Object#instance_variable_get(name)
  • Binding#local_variable_get(name)
  • Kernel#global_variable_get(name)
  • Module#class_variable_get(name)
  • Module#const_get

• Definitions
  • #source_location, #arity, #parameters for Method and Proc objects

• Last weapon
  • Kernel#eval, Object#instance_eval, ...

"Ruby.inspect" by Koichi Sasada, RDRC2014
Getting stack trace
caller, caller_locations

• `caller()` returns Backtrace strings array.
  • like `"t.rb:1:in `<main>''"]

• `caller_locations()` returns OO style backtrace information
  • `caller_locations(0).each{|loc|
    p "#{loc.path}:#{loc.lineno}"}
  • No need to parse "backtrace" string!
Getting more rich trace
debug_inspector gem

• Binding information for each frame
  • General version of caller_binding
  • https://github.com/banister/debug_inspector

"Ruby.inspect" by Koichi Sasada, RDRC2014
Accessing variables

• Object#instance_variable_get(name)
• Binding#local_variable_get(name)
• Kernel#global_variable_get(name)
• Module#class_variable_get(name)
• Module#const_get
Getting definitions

- Method#source_location, Proc#source_location
- Method#arity, Proc#arity
- Method#parameters, Proc#parameters
Evil eval

- eval series
  - Kernel#eval, Binding#eval
  - Object#instance_eval
  - Module#module_eval

- Can do everything
  - Accessing any variable (getting and setting)
  - Evaluate any expression
  - Strong, but dangerous

"Ruby.inspect" by Koichi Sasada, RDRC2014
Statistics

https://www.flickr.com/photos/cimmyt/5428317596
"Ruby.inspect" by Koichi Sasada, RDRC2014
Statistics features

• GC.stat for GC (memory management)
• ObjectSpace::count_objects
Statistics information

GC.stat returns “current information of GC”

• Counts
  • :count=>2,                     # GC count
  • :minor_gc_count=>2,            # minor GC count
  • :major_gc_count=>0,            # major GC count

• Current slot information
  • :heap_live_slot=>6836, #=> # of live objects
  • :heap_free_slot=>519, #=> # of freed objects
  • :heap_final_slot=>0, #=> # of waiting finalizer objects
  • total_slots = heap_live_slot + heap_free_slot + heap_final_slot

• Statistics
  • :total_allocated_object=>7674,  # total allocated objects
  • :total_freed_object=>838,# total freed objects
  • Current living objects = total_allocated_object - total_freed_object
100_000.times{|i| ""}; # Generate an empty string
h = GC.stat
puts "#{i}¥t#{h[:total_allocated_object]}¥t#{h[:total_freed_object]}"
GC.stat example: Leakey behavior

```ruby
ary = []
100_000.times{|i| ary << ''} # generate an empty string and store (leak)
h = GC.stat
puts "#{i}¥t#{h[:total_allocated_object]}¥t#{h[:total_freed_object]}"
```
Statistics information
ObjectSpace::count_objects

• ObjectSpace::count_objects returns counts for each type

Example:

p ObjectSpace::count_objects
 #=>
 {:TOTAL=>30235, :FREE=>1226, :T_OBJECT=>60, :T_CLASS=>513, :T_MODULE=>24
 , :T_FLOAT=>7, :T_STRING=>9527, :T_REGEXP=>68, :T_ARRAY=>1718, :T_HASH=>8
 9, :T_STRUCT=>1, :T_BIGNUM=>5, :T_FILE=>21, :T_DATA=>1013, :T_MATCH=>26, :T
_COMPLEX=>1, :T_NODE=>15904, :T_ICLASS=>32}

• Sister methods
  • ObjectSpace::count_objects_size in ‘ objspace ’ lib
Tracing

• TracePoint
• DTrace
• Object allocation tracing
• Trace object relations

"Ruby.inspect" by Koichi Sasada, RDRC2014
TracePoint

• Track Ruby’s execution
  • Insert tracing points by block
  • Introduced from Ruby 2.0
  • Lightweight OO-style version of “set_trace_func” method

```ruby
# old style
set_trace_func(lambda{|ev,file,line,id,klass,binding|  
  puts "#{ev} #{file}:#{line}"
})

# new style with TracePoint
trace = TracePoint.trace{|tp|  
  puts "#{tp.event}, #{tp.path}:#{tp.line}"
}
```

"Ruby.inspect" by Koichi Sasada, RDRC2014
TracePoint
Advantages

• Advantage of TracePoint compare with set_trace_func
  • OO style
  • Easy enable and disable
  • **Lightweight**
    • Creating binding object each time is too costly
  • Event filtering

"Ruby.inspect" by Koichi Sasada, RDRC2014
TracePoint
Traceable events

• Same as set_trace_func
  • line
  • call/return, c_call/c_return
  • class/end
  • raise

• New events (only for TracePoint)
  • thread_begin/thread_end
  • b_call/b_end (block start, block end)
TracePoint
Filtering

- TracePoint.new(events) only hook “events”
  - “set_trace_func” track all events
  - Example:
    TracePoint.new(:call, :return){...}

- Aliases
  - a_call -> call, c_call, b_call
  - a_return -> return, c_return, b_return

"Ruby.inspect" by Koichi Sasada, RDRC2014
TracePoint
Event information

• Same as set_trace_func
  • event
  • path, lineno
  • defined_class, method_id
  • binding

• New event info
  • return_value (only for return, c_return, b_return)
  • raised_exception (only for raise)
TracePoint
Internal events

• Added events
  • RUBY_INTERNAL_EVENT_NEWOBJ
    • When object is created
  • RUBY_INTERNAL_EVENT_FREEOBJ
    • When object is freed
  • RUBY_INTERNAL_EVENT_GC_START
    • When GC is started
  • RUBY_INTERNAL_EVENT_GC_END_MARK
    • When marking of GC is finished
  • RUBY_INTERNAL_EVENT_GC_END_SWEEP
    • When sweeping of GC is finished

"Ruby.inspect" by Koichi Sasada, RDRC2014
TracePoint
Internal events

• Timeline

"Ruby.inspect" by Koichi Sasada, RDRC2014
DTrace

• Solaris, MacOSX FreeBSD and Linux has DTrace tracing features
• Ruby interpreter support some events
• See https://bugs.ruby-lang.org/projects/ruby/wiki/DTraceProbes

"Ruby.inspect" by Koichi Sasada, RDRC2014
Object allocation tracing

- **ObjectSpace::trace_object_allocations**
  - Trace object allocation and record allocation-site
    - Record filename, line number, creator method’s id and class
    - Implemented by TracePoint with internal events NEWOBJ/FREEOBJ
  - **Usage:**
    ```ruby
    ObjectSpace.trace_object_allocations{ # record only in the block
      o = Object.new
      file = ObjectSpace.allocation_sourcefile(o) #=> __FILE__
      line = ObjectSpace.allocation_sourceline(o) #=> __LINE__ -2
    }
    ```
Trace objects relations

- ObjectSpace.reachable_objects_from(obj) returns directly reachable objects
  - Examples:
    1. When obj is [“a”, “b”, “c”], returns [Array, “a”, “b”, “c”]
    2. When obj is [“a”, “a”], returns [Array, “a”, “a”]
    3. When obj is [a = “a”, a], returns [Array, “a”]
Trace objects relations

• You can analyze memory leak. ... Maybe.
• Combination with ObjectSpace.memsiz eof() (introduced at 1.9) is also helpful to calculate how many memories consumed by obj.

Total 14 bytes (this is fake example)
Trace objects from root

- **ObjectSpace.reachable_objects_from_root** -> hash
  - Return all reachable objects from root.
  - You can get all objects graph in the heap.
  - **ObjectSpace::dump_all()** is implemented with this method.

"Ruby.inspect" by Koichi Sasada, RDRC2014
Make tools!!

• Example: combination of GC.stat and TracePoint
  • ObjectSpace::trace_object_allocation
  • gc_tracer: GC behavior
  • allocation_tracer: Allocation tracing

• You can make your own tools if you need!!
Inspect from outside

https://www.flickr.com/photos/justin_case/2842104135
Inspection features on computer layers

- Inspect Ruby process from outside
  - Ruby process
  - Your app
  - Libraries/Tools
  - Ruby interpreter

- From other process
- From OS

"Ruby.inspect" by Koichi Sasada, RDRC2014
Inspect from outside

• System level tracing
  • strace (system call tracer)
  • Dtrace, systemtap, ... (with Ruby’s dtrace support)

• System level profilers
  • Valgrind (massif for memory usage)
  • prof, proftools, ...

• System level debugger
  • gdb
Advanced inspection

https://www.flickr.com/photos/usnavy/5958545513
"Ruby.inspect" by Koichi Sasada, RDRC2014
Inspection features on computer layers

Hardware

Operating System

Ruby interpreter

Libraries/Tools

Your app

They are only software!!

"Ruby.inspect" by Koichi Sasada, RDRC2014
You can modify software

• Modify inspection tools
  • Most of tools are placed on github

• Modify Ruby interpreter
  • Make an C extension libraries with C-APIs
    • Some tools are written as C-extensions
  • Modify Ruby interpreter written in C

• Modify operating systems and system software layers

"Ruby.inspect" by Koichi Sasada, RDRC2014
Hacking Ruby

• “Ruby Under a Microscope”
  • By Pat Shaughnessy
  • http://patshaughnessy.net/ruby-under-a-microscope

• “Ruby Hacking Guide”
  • By Minero Aoki, written in Japanese
  • English translation: http://ruby-hacking-guide.github.io/

"Ruby.inspect" by Koichi Sasada, RDRC2014
Advanced computer layers

- Your app
- Libraries/Tools
- Ruby interpreter

Operating System

Hardware

"Ruby.inspect" by Koichi Sasada, RDRC2014
Advanced computer layers

Virtual Machine Monitor (VMM) system

They are also only software!!

"Ruby.inspect" by Koichi Sasada, RDRC2014
Important idea: Understanding Lower-layers

• Understanding computer layers and lower-layers helps your understanding of your application
  • Which information we can inspect
  • What happen on the computer

• Ruby hides computers details, but understanding details will help you
  • This is why “Computer science” study is important
  • Or try to ask lower-layer professionals 😊

• Balance is matter between higher-layers and lower-layers

"Ruby.inspect" by Koichi Sasada, RDRC2014
Today’s Message

Become a Low-level engineer (somtimes)

"Ruby.inspect" by Koichi Sasada, RDRC2014
Talk.inspect

Summary of this talk

- Introduction of Ruby 2.1, 2.2
- How to inspect your application behavior
  - With tools & services
  - Make a tools by inspection primitives
  - Inspection from outside
- Knowing “low-level” helps you
- Happy hacking

"Ruby.inspect" by Koichi Sasada, RDRC2014
“Ruby.inspect”
Thank you for your attention

Koichi Sasada
<ko1@heroku.com>