Working with Ruby

Occasional blog posts about Ruby updates, tools, editor tweaks, and random snippets. You might also be interested in my newer project that docuements lesser-known features in Ruby: Idiosyncratic Ruby.

How to properly check for your Ruby interpreter, version and OS

September 2010, ruby · gem · hints · tutorial · zucker

Zucker 4 adds accessors to some environment information:

- **OS**: returns the current operating system
- **RubyEngine**: returns the current Ruby implementation
- **RubyVersion**: returns the current Ruby version

And here is how it works.

OS

The basic way to get the operating system from a Ruby script is the`RUBY_PLATFORM` constant. But it's not recommended, because some Ruby implementations report the virtual machine on which they run (e.g.`java`). A simple solution is the `RbConfig::CONFIG` hash, which is build when Ruby is build.

```
require 'rbconfig'; RbConfig::CONFIG['host_os']
```

Let's abstract this information to build a helpful OS constant:

os

```
require 'rbconfig'

module OS
  class << self
    def is?(what)
      what === RbConfig::CONFIG['host_os']
    end
    alias is is?

    def to_s
      RbConfig::CONFIG['host_os']
    end
  end

  module_function

  def linux?
    OS.is? /linux|cygwin/
  end

  def mac?
    OS.is? /mac|darwin/
```

```ruby
  end

  def bsd?
    OS.is? /bsd/
  end

  def windows?
    OS.is? /mswin|win|mingw/
  end

  def solaris?
    OS.is? /solaris|sunos/
  end

  def posix?
    linux? or mac? or bsd? or solaris? or Process.respond_to?(:fork)
  end

  #def symbian?
    #TODO who knows what symbian returns?
  #end

  # ...
end
```

Because of the `module_function` method, you can either call the methods on the module or include the module to call them without prefix.

RubyEngine

Most Ruby implementations set the `RUBY_ENGINE` constant to identify themselves, but not all – for example, the official Ruby 1.8 does not have one. This snippet takes care of some exceptions:

ruby_engine

```ruby
module RubyEngine
  class << self
    # try to guess it
    @interpreter = case
    when RUBY_PLATFORM == 'parrot'
      'cardinal'
    when Object.constants.include?( :RUBY_ENGINE ) ||
         Object.constants.include?( 'RUBY_ENGINE'  )
      if RUBY_ENGINE == 'ruby'
        if RUBY_DESCRIPTION =~ /Enterprise/
          'ree'
        else
          'mri'
        end
      else
        RUBY_ENGINE.to_s # jruby, rbx, ironruby, macruby, etc.
      end
    else # probably 1.8
      'mri'
    end

    def is?(what)
      what === @interpreter
    end
    alias is is?

    def to_s
      @interpreter
    end
  end

module_function

  def mri?
    RubyEngine.is? 'mri'
  end
  alias official_ruby? mri?
  alias ruby? mri?

  def jruby?
    RubyEngine.is? 'jruby'
  end
  alias java? jruby?

  def rubinius?
    RubyEngine.is? 'rbx'
  end
  alias rbx? rubinius?

  def ree?
    RubyEngine.is? 'ree'
  end
  alias enterprise? ree?

  def ironruby?
    RubyEngine.is? 'ironruby'
  end
  alias iron_ruby? ironruby?

  def cardinal?
    RubyEngine.is? 'cardinal'
  end
  alias parrot? cardinal?
  alias perl? cardinal?
end
```

RubyVersion

The used Ruby version can be accessed with `RUBY_VERSION`. To simplify version checking, this snippet adds some methods for querying and the possibility to check for 1.8 / 1.9 using a Float:

ruby_version

```ruby
### usage examples
# RubyVersion
### check for the main version with a Float
# RubyVersion.is? 1.8
### use strings for exacter checking
# RubyVersion.is.above '1.8.7'
# RubyVersion.is.at_least '1.8.7' # or below, at_most, not
### you can use the common comparison operators
# RubyVersion >= '1.8.7'
# RubyVersion.is.between? '1.8.6', '1.8.7'
### relase date checks
# RubyVersion.is.older_than Date.today
# RubyVersion.is.newer_than '2009-08-19'
### accessors
# RubyVersion.major # e.g. => 1
# RubyVersion.minor # e.g. => 8
# RubyVersion.tiny  # e.g. => 7
# RubyVersion.patchlevel # e.g. => 249
# RubyVersion.description # e.g. => "ruby 1.8.7 (2010-01-10 patchlevel 249) [i486-linux]"

require 'date'
require 'time'

module RubyVersion
  class << self
    def to_s
      RUBY_VERSION
    end

    # comparable
    def <=>(other)
      value = case other
        when Integer
          RUBY_VERSION.to_i
        when Float
          RUBY_VERSION.to_f
        when String
          RUBY_VERSION
        when Date,Time
          other.class.parse(RUBY_RELEASE_DATE)
        else
          other = other.to_s
          RUBY_VERSION
        end
      value <=> other
    end
    include Comparable

    # chaining for dsl-like language
    def is?(other = nil)
      if other
        RubyVersion == other
      else
        RubyVersion
      end
    end
    alias is is?

    # aliases
    alias below     <
    alias below?    <
    alias at_most   <=
    alias at_most?  <=
    alias above     >
    alias above?    >
    alias at_least  >=
    alias at_least? >=
    alias exactly   ==
    alias exactly?  ==
    def not(other)
      self != other
    end
    alias not?     not
    alias between between?

    # compare dates
    def newer_than(other)
      if other.is_a? Date or other.is_a? Time
        RubyVersion > other
      else
        RUBY_RELEASE_DATE > other.to_s
      end
    end
    alias newer_than? newer_than

    def older_than(other)
      if other.is_a? Date or other.is_a? Time
        RubyVersion < other
      else
        RUBY_RELEASE_DATE < other.to_s
      end
    end
    alias older_than? older_than

    def released_today
      RubyVersion.date == Date.today
    end
    alias released_today? released_today

    # accessors

    def major
      RUBY_VERSION.to_i
    end
    alias main major

    def minor
      RUBY_VERSION.split('.')[1].to_i
    end
    alias mini minor

    def tiny
      RUBY_VERSION.split('.')[2].to_i
    end

    alias teeny tiny

    def patchlevel
      RUBY_PATCHLEVEL
    end

    def platform
      RUBY_PLATFORM
    end

    def release_date
      Date.parse RUBY_RELEASE_DATE
    end
    alias date release_date

    def description
      RUBY_DESCRIPTION
    end
  end
end
```

Bugfixes are welcome ;) **Update:** new RubyVersion implementation (thanks to Hanmac for the hint)

random | September 02, 2010

Nice colors on the code syntax. what's the theme called?

J-_-L | September 03, 2010

Hi random, It's hand crafted (inspired by railscasts), see the css for the source ;)

trans | September 03, 2010

Looks like a bug in RubyEngine, it can return a symbol but #is? compares a string.

Also, here's an idea... extend the actual constants with your methods. e.g. Get rid of the `class << self` and then `RUBY_VERSION.extend(RubyVersion)`. Or just do `class << RUBY_VERSION`. Then we can can do `RUBY_VERSION.major`, etc.

J-_-L | September 03, 2010

Hi trans,
thank you for your interest and thanks for spotting the bug :).

About the idea: It's very interesting. I've tried it, but noticed that I had to recreate RUBY_VERSION with <code>Object.send :remove_const, :RUBY_VERSION</code>, because it's frozen. That might not be a problem, but I think, I stick to the extra constant. One expects, that it offers extra methods, because of the slightly different name. However, from RUBY_VERSION, most people expect it to be a normal string.

sampablokuper | December 23, 2011

Hi Jan,
I see you've made os.rb available under CC-BY. I'd be really grateful if you'd make it available under a GPL-compatible license too!
Thanks,
Sam

J-_-L | January 05, 2012

Hi sampablokuper, you can use it under the terms of the gpl version 3. :)

dbirtwell | May 30, 2012

Seems like there might be a bug under Mac OS X. The following

puts "Is Mac: #{OS::mac?}"

returns true. Probably because "Darwin" contains "win"

dbirtwell | May 30, 2012

Sorry, that should be

puts "Is Windows: #{OS::windows?}"

returns true under Mac OS X

Joseph | August 09, 2012

@dbirtwell try this fix:

def windows?

OS.is? /mswin|^win|mingw/

end