Symbols

Erik Michaels-Ober (@sferik)
RUG::B, 7 May 2015
Symbols are Useless 🍤

1. Calling Symbol#to_s on every symbol in your program would not change its semantics.

2. Symbols are an archaic optimization. They were useful 20 years ago, but not today.

3. Symbols and strings have converged into one type.
Symbols are 🥶 Strings

"string".freeze.object_id
==
"string".freeze.object_id

# false on Ruby < 2.1
# true on Ruby >= 2.1
def measure
    init = Time.now
    yield if block_given?
    return Time.now - init
end
N = 1_000_000

measure { N.times { "string" } }

measure { N.times { "string".freeze } }

measure { N.times { :symbol } }
Results

string: 0.126956
freeze: 0.056473
symbol: 0.056416
1. Symbols and frozen strings perform identically.

2. Allocating a million strings takes about twice as long as allocating one string, putting it into a hash table, and looking it up a million times.

3. You can allocate a million short strings in about a tenth of a second on a typical 2015 laptop computer.
PowerBook 190

- 33 MHz CPU
- 8 MB RAM
- 500 MB HDD

$1850
August 1995
Ruby makes me happy.
Symbols make me sad.
def stringify_keys

  self.dup.stringify_keys!
end

def stringify_keys!

  self.keys.each do |k|

end
The Difference Between Ruby Symbols and Strings

January 11, 2009  Robert Sosinski  Ruby

Symbols are quite an interesting, and often ill-understood, facet of Ruby. Used extensively throughout Rails and many other Ruby libraries, Symbols are a common sight. However, their rational and purpose is something of a mystery to many Rubyists. This misunderstanding can probably be attributed to many methods throughout Ruby using Symbols and Strings interchangeably. Hopefully, this tutorial will show the value of Symbols and why they are a very useful attribute of Ruby.

Symbols are Strings, Sort Of

The truth of the matter is that Symbols are Strings, just with an important difference, Symbols are immutable. Mutable objects can be changed after assignment while immutable objects can only be overwritten. Ruby is quite unique in offering mutable Strings, which adds greatly to its expressiveness. However mutable Strings can have their share of issues in terms of creating unexpected results and reduced performance. It is for this reason Ruby also offers programmers the choice of Symbols.

So, how flexible are Symbols in terms of representing data? The answer is just as flexible as Strings. Lets take a look at some valid Strings and their Symbol equivalents.

"hello"  hello

123 123
13 Ways of Looking at a Ruby Symbol

Jan 20, 2007 • by Eric Kidd

New Ruby programmers often ask, "What, exactly, is a symbol? And how does it differ from a string?" No one answer works for everybody, so—with apologies to Wallace Stevens—here are 13 ways of looking at a Ruby symbol.

A Ruby symbol is:

1. ...the name of something, not just a blob of text
2. ...a label in a free-form enumeration
3. ...a constant, unique name
4. ...an "interned" string
5. ...an object with O(1) comparison
6. ...a Lisp identifier
7. ...a Ruby identifier
8. ...the keyword for a keyword argument
9. ...an excellent choice for a hash key
10. ...like a Mac OS::Type
11. ...a memory leak
12. ...a clever way to store only a single copy of a string
13. ...a C typedef named "ID"
Denial of Service and Unsafe Object Creation Vulnerability in JSON (CVE-2013-0269)

Posted by usa on 22 Feb 2013

There is a denial of service and unsafe object creation vulnerability in the json bundled with ruby. This vulnerability has been assigned the CVE identifier CVE-2013-0269. We strongly recommend to upgrade ruby.

Details

When parsing certain JSON documents, the JSON gem (includes bundled with ruby) can be coerced into creating Ruby symbols in a target system. Since Ruby symbols are not garbage collected, this can result in a denial of service attack.

The same technique can be used to create objects in a target system that act like internal objects. These "act alike" objects can be used to bypass certain security mechanisms and can be used as a spring board for SQL injection attacks.
Denial of Service Vulnerability in Action View when using render :text (CVE-2014-0082)

Aaron Patterson 2/18/14

Denial of Service Vulnerability in Action View when using render :text

There is a denial of service vulnerability in the text rendering component of Action View. This vulnerability has been assigned the CVE identifier CVE-2014-0082.

Versions Affected: 3.0.x, 3.1.x, 3.2.x
Not affected: 4.0.x
Fixed versions: 3.2.17

Impact

Strings sent in specially crafted headers will be converted to symbols. This can cause a denial of service since symbols are not removed by the garbage collector.

All users running an affected release should either upgrade or use one of the work
[PATCH]Symbol GC

Added by Narihiro Nakamura about 1 year ago. Updated about 1 year ago.

[ruby-core:61456]

Status: Closed
Priority: Normal
Assignee: Yukihiro Matsumoto

Description

I've written a patch to collect most symbols.

PATCH: https://github.com/authorNari/ruby/compare/4a91fb7a45f0e3c...symbol_gc.patch

Summary

- Most symbols in Ruby level are GC-able (generated by #to_sym, #intern, etc..)
- Exclude a symbol which is translated ID in C-level from GC-able symbols
- Keep Ruby's C extension compatibility
- Pass make test-all

Benchmark

A benchmark program is here.

```ruby
obj = Object.new
100_000.times do |i|
end
```
Proposal

:foo == "foo" # true