Ruby master - Feature #12319

`Module#const_get` does not accept symbol with nested name

04/26/2016 01:25 AM - sawa (Tsuyoshi Sawada)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Module#const_get accepts non-nested string, nested string, and non-nested symbol:

```
class A; class B; class C end end end

A.const_get("B")     # => A::B
A.const_get("B::C") # => A::B::C
A.const_get(:B)      # => A::B
```

but does not accept nested symbol:

```
A.const_get(:"B::C") # => NameError: wrong constant name B::C
```

Related issues:

Related to Ruby master - Feature #5690: Module#qualified_const_get	**Closed**

History

#1 - 04/26/2016 04:50 AM - nobu (Nobuyoshi Nakada)

- Related to Feature #5690: Module#qualified_const_get added

#2 - 04/26/2016 04:56 AM - nobu (Nobuyoshi Nakada)

It's the intended behavior.
Symbol is a single name, not a class/module path.

#3 - 04/26/2016 05:57 AM - sawa (Tsuyoshi Sawada)

Nobuyoshi Nakada wrote:

> It's the intended behavior.
> Symbol is a single name, not a class/module path.

I don't see any mentioning to symbol in #5690. If this is intended, I would like to ask it as a feature request. It is confusing/surprising that something you can do with string can be done with symbol, but cannot be completely done.

#4 - 04/26/2016 05:57 AM - sawa (Tsuyoshi Sawada)

- Tracker changed from Bug to Feature

#5 - 04/26/2016 07:01 AM - duerst (Martin Dürst)

Tsuyoshi Sawada wrote:

> It is confusing/surprising that something you can do with string can be done with symbol, but cannot be completely done.

It's absolutely not confusing to me that A.const_get("B::C") works, but A.const_get(:"B::C") doesn't. If at all, I'd expect it to be A.const_get(:B, :C) or A.const_get([:B, :C]). Strings can have lots of structure; Symbols essentially don't have structure.

#6 - 04/26/2016 07:21 AM - sawa (Tsuyoshi Sawada)

Martin Dürst wrote:

> Strings can have lots of structure; Symbols essentially don't have structure.

If you look at the literal :"B::C", then it might seem somewhat contrived, but in practical use, the argument passed to const_get can be (and in most

case is) a variable. And since it is more common to express method or constant names as symbols rather than strings, it is natural that a symbol (perhaps derived somewhere in the code from a string via to_sym) sneaks into the relevant variable argument of const_get. If it did not accept symbol at all, then it is easy to be reminded to just apply to_s to it, but if symbol works sometimes, then to_s can be easily forgotten. Then, later during the program run, when the variable turns out to hold a symbol representing a nested name, it suddenly raises a problem.

#7 - 04/26/2016 09:52 AM - Eregon (Benoit Daloze)

I think as well that Symbols should not be used to generate dynamic names like that "SomModName::SomeOtherName".
Symbol are for identifiers, this is two or more identifiers with "::" in between.

#8 - 04/26/2016 06:19 PM - shevegen (Robert A. Heiler)

I am also for keeping symbols simple! I love symbols, I abuse them a lot in my ruby code to yield special instructions/behaviour such as:

```
disable :colours # where disable() is a method
```

But it would scare me if everyone would start to misuse symbols or expand their usage too much. You also have to keep in mind that newcomers are often confused about Strings versus Symbols. And I think in the old ruby, it was actually not meant that Symbols would become so exposed (not that I mind it, I myself love code like the above)