Ruby master - Feature #15881

Optimize deconstruct in pattern matching

05/27/2019 04:19 PM - marcandre (Marc-Andre Lafortune)

Status:	Open	
Priority:	Normal	
Assignee:	ktsj (Kazuki Tsujimoto)	
Target version:		

Description

```
class A
  def deconstruct
    puts 'deconstruct called'
    [1]
  end
end

case A.new
in [2]
  2
in [1]
  1
else
end
```

Currently this outputs:

```
deconstruct called
deconstruct called
=> 1
```

Shouldn't deconstruct called print only once, whenever the first deconstruction needed occurs?

Related issues:

Related to Ruby master - Feature #14912: Introduce pattern matching syntax	**Assigned**

History

#1 - 06/12/2019 12:56 PM - mame (Yusuke Endoh)

I talked with ktsj, the author of pattern matching. He had actually considered caching the result of deconstruct, but we found it difficult because of some reasons.

- If destructive operation is applied to the object being matched (this is possible in a guard expression), the behavior of pattern matching would get messed up.
- ktsj investigated Scala's pattern match, and it calls unapply method each time without caching.
- We believe Array#deconstruct and Hash#deconstruct_keys would be most often called. They just return the receiver itself, so no object is generated. So, caching is useless in the typical case.
- If the overhead of a method call itself matters, we can optimize it by adding a special instruction like opt_deconstruct.
- If you need to cache the result of your own deconstruct definition, it is not so difficult to manually memoize the result.

#2 - 06/12/2019 01:23 PM - marcandre (Marc-Andre Lafortune)

mame (Yusuke Endoh) wrote:

> I talked with ktsj, the author of pattern matching. He had actually considered caching the result of deconstruct, but we found it difficult because of some reasons.
>
> - If destructive operation is applied to the object being matched (this is possible in a guard expression), the behavior of pattern matching would get messed up.

Is there a valid use case for this though? Seems more like an anti-pattern that is better not being supported at all.

- ktsj investigated Scala's pattern match, and it calls unapply method each time without caching.

Interesting. Do we know if there a is good reason for that? Scala is in general faster than Ruby, so it might not matter as much there...

- We believe Array#deconstruct and Hash#deconstruct_keys would be most often called. They just return the receiver itself, so no object is generated. So, caching is useless in the typical case.

Well, unless I'm mistaken, it would not be completely useless as it would avoid send(:respond_to?, :deconstruct_keys) and send(:deconstruct_keys), but the main issue really is for user defined classes.

- If you need to cache the result of your own deconstruct definition, it is not so difficult to manually memoize the result.

I disagree. You can't simply use @cache ||= ..., you need to invalidate @cache if any dependency of the ... changes. That may be quite tricky. Let's remember:

```
There are only two hard things in Computer Science: cache invalidation and naming things.

-- Phil Karlton
```

I remain convinced that it would be better to cache this result.

#3 - 06/12/2019 10:18 PM - mame (Yusuke Endoh)

marcandre (Marc-Andre Lafortune) wrote:

> mame (Yusuke Endoh) wrote:
>
> > I talked with ktsj, the author of pattern matching. He had actually considered caching the result of deconstruct, but we found it difficult because of some reasons.
> >
> > - If destructive operation is applied to the object being matched (this is possible in a guard expression), the behavior of pattern matching would get messed up.
>
> Is there a valid use case for this though? Seems more like an anti-pattern that is better not being supported at all.

ktsj and I prefer simpleness and robustness to performance when we consider the design of Ruby language. In Ruby, optimization is not primary; it is good as long as it does not change the naive semantics. For example, if we disallow the redefinition of bulit-in methods including Integer#+, we can make the interpreter faster and can make the implementation much simpler. But we still allow and respect the redefinition.

In this case, more intelligent optimization (including nice cache invalidation) that does not affect the semantics is needed (if the performance is really needed).

> - If you need to cache the result of your own deconstruct definition, it is not so difficult to manually memoize the result.
>
> I disagree. You can't simply use @cache ||= ..., you need to invalidate @cache if any dependency of the ... changes. That may be quite tricky. Let's remember:
>
> ```
> There are only two hard things in Computer Science: cache invalidation and naming things.
>
> -- Phil Karlton
> ```

Agreed. The same goes to Ruby interpreter itself :-)

#4 - 06/14/2019 11:41 PM - ktsj (Kazuki Tsujimoto)

- Related to Feature #14912: Introduce pattern matching syntax added

#5 - 06/22/2019 07:32 AM - Eregon (Benoit Daloze)

- Description updated

#6 - 07/29/2019 08:12 AM - ko1 (Koichi Sasada)

- Assignee set to ktsj (Kazuki Tsujimoto)

#7 - 12/25/2019 04:28 AM - naruse (Yui NARUSE)

- Target version deleted (2.7)