Ruby master - Bug #13146

Float::NANs in Hashes are confusing (more than usual).

01/22/2017 03:46 PM - yxhuvud (Linus Sellberg)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td>ruby 2.3.1p112</td>
</tr>
<tr>
<td>Backport:</td>
<td>2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN</td>
</tr>
</tbody>
</table>

Description

```ruby
test = {Float::NAN => 1, -Float::NAN => 2}
=> (NaN=>1, NaN=>2)

test.values_at(Float::NAN, -Float::NAN)
=> [1, nil]
```

I don't know what the correct behaviour ought to be, but it seems inconsistent to create a hash with two elements but not be able to extract both values.

History

#1 - 01/22/2017 08:56 PM - shevegen (Robert A. Heiler)

This is actually interesting altogether.

First that Float::NAN becomes NaN in the display; this is the first surprise to me.

But second, that we have seemingly the same key twice (???), at the least in the display.

```ruby
{NaN=>1, NaN=>2}
```

Should not the second key be a negative one? I guess NaN does not specify this as any more accurate but it treats them as two different entries in the hash regardless, or? That is a bit confusing to me.

I have no idea if this is a correct behaviour or not but to me this was a surprise to see.

Some docu can be found here:

http://ruby-doc.org/core-2.0.0/Float.html

"NAN: An expression representing a value which is "not a number".

To me I guess the surprise is that NaN in itself should be unique? But in hashes you can store different NaNs? Then again I probably do not understand anything anyway. :)

#2 - 01/22/2017 09:31 PM - yxhuvud (Linus Sellberg)

Robert A. Heiler wrote:

To me I guess the surprise is that NaN in itself should be unique?

NaN is defined by IEEE to fail equality tests with all floats (including itself), so it could be argued it is more unique than any other value. It could certainly be argued that as each NaN fail equality tests, indexing by NaN should never find anything in a hash.

#3 - 01/26/2017 01:29 PM - sos4nt (Stefan Schüßler)

Somehow, using -Float::NAN (for example in a hash or array) results in new objects being created:

```ruby
-Float::NAN.object_id
#=> -70362448918680

[-Float::NAN, -Float::NAN].map(&:object_id)
#=> [70362448604580, 70362448604520, 70362448604500]
```

As a result, these objects are no longer equal. It seems like -Float::NAN is duplicated upon assignment:
Float::NAN on the other hand behaves as expected (always the same object):

```
Float::NAN.object_id
#=> 70362448918680
[Float::NAN, Float::NAN, Float::NAN].map(&:object_id)
#=> [70362448918680, 70362448918680, 70362448918680]
```

The same happens with -Float::INFINITY.

Are you suggesting to make infinities and a NaN multiton?

I don't know how these are implemented, but I'd expect -Float::NAN to work like Float::NAN in that regard. If I'm not mistaken, there is only one Float::NAN instance. Having multiple instances of -Float::NAN on the other hand is very confusing. Especially because they are not even equal:

```
[-Float::NAN, -Float::NAN, -Float::NAN].uniq
#=> [NaN, NaN, NaN]
```

Are you suggesting to make infinities and a NaN multiton?

One alternative is to copy the approach given here: [https://research.swtch.com/randhash](https://research.swtch.com/randhash)

TLDR:

1. NaN is given a random hash (to avoid hash key collisions).
2. Hash insertion work, but never overwrite - always adds a new key.
3. Lookup never finds anything
4. Iteration over the values works.

I reported similar issue on BigDecimal.

It doesn't work right now as you expect, since the hash assumes same elements are always equal, regardless their hash values.

You have to understand what NaN is, before playing with it.