# Ruby - Bug #13002

## Hash calculations no longer using universal hashing

12/04/2016 12:16 AM - duerst (Martin Dürst)

| | | | |
|---|---|---|---|
| **Status:** | Closed | | |
| **Priority:** | Normal | | |
| **Assignee:** | naruse (Yui NARUSE) | | |
| **Target version:** | | | |
| **ruby -v:** | ruby 2.4.0dev (2016-12-02 trunk 56965) [x86_64-cygwin] | **Backport:** | 2.1: DONTNEED, 2.2: DONTNEED, 2.3: DONTNEED |

**Description**

When preparing for my lecture on hash tables last week, I found that Ruby trunk doesn't do universal hashing anymore. See http://events.ccc.de/congress/2011/Fahrplan/attachments/2007_28C3_Effective_DoS_on_web_application_platforms.pdf for background.

I contacted security@ruby-lang.org, but was told by Shugo that because trunk is not a published version, we can talk about it publicly.

Shugo also said that the change was introduced in r56650.

Following is some output from two different versions of Ruby that show the problem:

On Ruby 2.2.3, different hash value for the same number every time Ruby is restarted:

```
C:\Users\duerst>ruby -v
ruby 2.2.3p173 (2015-08-18 revision 51636) [i386-mingw32]

C:\Users\duerst>ruby -e 'puts 12345678.hash'
611647260

C:\Users\duerst>ruby -e 'puts 12345678.hash'
-844752827

C:\Users\duerst>ruby -e 'puts 12345678.hash'
387106497
```

On Ruby trunk, always the same value:

```
duerst@Arnisee /cygdrive/c/Data/ruby
$ ruby -v
ruby 2.4.0dev (2016-12-02 trunk 56965) [x86_64-cygwin]

duerst@Arnisee /cygdrive/c/Data/ruby
$ ruby -e 'puts 12345678.hash'
1846311797112760547

duerst@Arnisee /cygdrive/c/Data/ruby
$ ruby -e 'puts 12345678.hash'
1846311797112760547

duerst@Arnisee /cygdrive/c/Data/ruby
$ ruby -e 'puts 12345678.hash'
1846311797112760547
```

**Related issues:**

| | |
|---|---|
| Related to Ruby - Feature #12142: Hash tables with open addressing | **Closed** |

### Associated revisions

**Revision 5714a26b90b40846733fb2a5764d4c61285f5ea1 - 12/06/2016 04:43 AM - nobu (Nobuyoshi Nakada)**

switching hash removal

- st.h (struct st_hash_type): Remove strong_hash.

(struct st_table): Remove inside_rebuild_p and curr_hash.
- st.c (do_hash): Use type->hash instead of curr_hash.
(make_tab_empty): Remove setting up curr_hash.
(st_init_table_with_size): Remove setting up inside_rebuild_p.
(rebuild_table): Remove clearing inside_rebuild_p.
(reset_entry_hashes, HIT_THRESHOULD_FOR_STRONG_HASH): Remove code recognizing a denial attack and switching to strong hash.
- hash.c (rb_dbl_long_hash, rb_objid_hash, rb_ident_hash): Use rb_hash_start to randomize the hash.
(str_seed): Remove.
(any_hash): Remove strong_p and use always rb_str_hash for strings.
(any_hash_weak, rb_any_hash_weak): Remove.
(st_hash_type objhash): Remove rb_any_hash_weak.

based on the patch by Vladimir N Makarov vmakarov@redhat.com at [ruby-core:78490]. [Bug #13002]

- test/ruby/test_hash.rb (test_wrapper): objects other than special constants should be able to be wrapped.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@56992 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

**Revision 5714a26b - 12/06/2016 04:43 AM - nobu (Nobuyoshi Nakada)**

switching hash removal

- st.h (struct st_hash_type): Remove strong_hash.
(struct st_table): Remove inside_rebuild_p and curr_hash.
- st.c (do_hash): Use type->hash instead of curr_hash.
(make_tab_empty): Remove setting up curr_hash.
(st_init_table_with_size): Remove setting up inside_rebuild_p.
(rebuild_table): Remove clearing inside_rebuild_p.
(reset_entry_hashes, HIT_THRESHOULD_FOR_STRONG_HASH): Remove code recognizing a denial attack and switching to strong hash.
- hash.c (rb_dbl_long_hash, rb_objid_hash, rb_ident_hash): Use rb_hash_start to randomize the hash.
(str_seed): Remove.
(any_hash): Remove strong_p and use always rb_str_hash for strings.
(any_hash_weak, rb_any_hash_weak): Remove.
(st_hash_type objhash): Remove rb_any_hash_weak.

based on the patch by Vladimir N Makarov vmakarov@redhat.com at [ruby-core:78490]. [Bug #13002]

- test/ruby/test_hash.rb (test_wrapper): objects other than special constants should be able to be wrapped.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@56992 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

## History

**#1 - 12/04/2016 12:16 AM - duerst (Martin Dürst)**

*- Related to Feature #12142: Hash tables with open addressing added*

**#2 - 12/04/2016 12:24 AM - duerst (Martin Dürst)**

*- Backport changed from 2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN to 2.1: DONTNEED, 2.2: DONTNEED, 2.3: DONTNEED*

**#3 - 12/04/2016 01:30 AM - vmakarov (Vladimir Makarov)**

The new hash table implementation uses a different approach to deal with the denial attacks.

Instead of using siphash (a secure hash function) all the time, the new hash tables use faster hash functions and when they recognize an ongoing denial attack, they are **rebuilt and switches to secure hash function** (the same siphash). One table can use a secure hash when another one can still use faster hash functions.

The new hash table implementation **has a code to recognize a denial attack** which is recognized when a number of different collided elements having the same hash reaches a threshold (as I remember it is about 10 now). Yura Sokolov proposed even faster method to recognize the attack (as he wrote a "statistical* approach) but I am not sure it will speed up the hash tables as the code where it is used inside is memory bound code.

The advantage of such approach is faster hash tables and possibility of using different secure hash functions (even crypto-level one as sha2/3) which are much slower than siphash without loosing the speed in practice.

I don't know other hash tables with such approach to the denial attack. So it is an unique feature for MRI right now.

If you need a secure hash for objects (like integer). I guess it can be fixed (the hash tables still can use the current approach and different hash). But I believe people should not use this hash for secure applications because underlying siphash is not a crypto-level one (sha3 or blake would be a better choice).

#### #4 - 12/04/2016 02:45 AM - nobu (Nobuyoshi Nakada)

Vladimir Makarov wrote:

> Instead of using siphash (a secure hash function) all the time, the new hash tables use faster hash functions and when they recognize an ongoing denial attack, they are **rebuilt and switches to secure hash function** (the same siphash). One table can use a secure hash when another one can still use faster hash functions.

That means the hash function must be stronger when strong_p, doesn't it?
Integer#hash calls rb_any_hash, not rb_any_hash_weak, so its result should be strong but it isn't now.

```
diff --git c/hash.c w/hash.c
index b4c74ed..a5f660e 100644
--- c/hash.c
+++ w/hash.c
@@ -147,24 +147,19 @@ long rb_objid_hash(st_index_t index);

 long
-rb_dbl_long_hash(double d)
+rb_dbl_long_hash(double d, int strong_p)
 {
+    union {double d; st_index_t i;} u;
+    st_index_t hnum;
     /* normalize -0.0 to 0.0 */
     if (d == 0.0) d = 0.0;
-#if SIZEOF_INT == SIZEOF_VOIDP
-    return rb_memhash(&d, sizeof(d));
-#else
-    {
- union {double d; uint64_t i;} u;
-
- u.d = d;
- return rb_objid_hash(u.i);
-    }
-#endif
+    u.d = d;
+    hnum = strong_p ? rb_hash_start(u.i) : u.i;
+    return rb_objid_hash(hnum);
 }

 #if SIZEOF_INT == SIZEOF_VOIDP
-static const st_index_t str_seed = 0xfa835867;
+# define str_seed rb_hash_start(0xfa835867)
 #else
-static const st_index_t str_seed = 0xc42b5e2e6480b23bULL;
+# define str_seed rb_hash_start(0xc42b5e2e6480b23bULL)
 #endif

@@ -184,4 +180,5 @@ any_hash_general(VALUE a, int strong_p, st_index_t (*other_func)(VALUE))
       goto flt;
   }
+ if (strong_p) a = rb_hash_start(a);
   hnum = rb_objid_hash((st_index_t)a);
     }
@@ -200,5 +197,5 @@ any_hash_general(VALUE a, int strong_p, st_index_t (*other_func)(VALUE))
     else if (BUILTIN_TYPE(a) == T_FLOAT) {
       flt:
- hnum = rb_dbl_long_hash(rb_float_value(a));
+ hnum = rb_dbl_long_hash(rb_float_value(a), strong_p);
     }
     else {
diff --git c/internal.h w/internal.h
index 3a4fbd5..8f04165 100644
--- c/internal.h
+++ w/internal.h
@@ -1086,5 +1086,5 @@ VALUE rb_hash_default_value(VALUE hash, VALUE key);
 VALUE rb_hash_set_default_proc(VALUE hash, VALUE proc);
 long rb_objid_hash(st_index_t index);
```

```
-long rb_dbl_long_hash(double d);
+long rb_dbl_long_hash(double d, int strong_p);
 st_table *rb_init_identtable(void);
 st_table *rb_init_identtable_with_size(st_index_t size);
diff --git c/numeric.c w/numeric.c
index d2c9cf7..c4af7f1 100644
--- c/numeric.c
+++ w/numeric.c
@@ -1454,5 +1454,5 @@ VALUE
 rb_dbl_hash(double d)
 {
-    return LONG2FIX(rb_dbl_long_hash (d));
+    return LONG2FIX(rb_dbl_long_hash(d, TRUE));
 }

diff --git c/test/ruby/test_rand.rb w/test/ruby/test_rand.rb
index 46d10f8..4d2805b 100644
--- c/test/ruby/test_rand.rb
+++ w/test/ruby/test_rand.rb
@@ -570,3 +570,35 @@
     assert_operator(size.fdiv(n), :>, 15)
   end
+
+  def assert_hash_random(obj, dump = obj.inspect)
+    a = [obj.hash.to_s]
+    3.times {
+      assert_in_out_err(["-e", "print #{dump}.hash"], "") do |r, e|
+        a += r
+        assert_equal([], e)
+      end
+    }
+    assert_not_equal([obj.hash.to_s], a.uniq)
+    assert_operator(a.uniq.size, :>, 2, proc {a.inspect})
+  end
+
+  def test_str_hash
+    assert_hash_random('abc')
+  end
+
+  def test_int_hash
+    assert_hash_random(0)
+    assert_hash_random(+1)
+    assert_hash_random(-1)
+    assert_hash_random(+(1<<100))
+    assert_hash_random(-(1<<100))
+  end
+
+  def test_float_hash
+    assert_hash_random(0.0)
+    assert_hash_random(+1.0)
+    assert_hash_random(-1.0)
+    assert_hash_random(1.72723e-77)
+    assert_hash_random(Float::INFINITY, "Float::INFINITY")
+  end
 end
diff --git c/test/ruby/test_string.rb w/test/ruby/test_string.rb
index 7ee7fa7..2bd35b1 100644
--- c/test/ruby/test_string.rb
+++ w/test/ruby/test_string.rb
@@ -980,16 +980,4 @@
   end

-  def test_hash_random
-    str = 'abc'
-    a = [str.hash.to_s]
-    3.times {
-      assert_in_out_err(["-e", "print #{str.dump}.hash"], "") do |r, e|
-        a += r
-        assert_equal([], e)
-      end
-    }
-    assert_not_equal([str.hash.to_s], a.uniq)
-  end
-
   def test_hex
```

```
    assert_equal(255,  S("0xff").hex)
```

#### #5 - 12/04/2016 04:12 AM - nobu (Nobuyoshi Nakada)

test_wrapper_of_special_const failed, that is, it's impossible to emulate switching weak and strong versions under the hood.
I think this behavior is quirky.

#### #6 - 12/04/2016 05:37 AM - vmakarov (Vladimir Makarov)

Nobuyoshi Nakada wrote:

> That means the hash function must be stronger when strong_p, doesn't it?
> Integer#hash calls rb_any_hash, not rb_any_hash_weak, so its result should be strong but it isn't now.

Sorry, if I understand Ruby documentation wrongly.  But it (http://ruby-doc.org/core-2.3.3/Object.html#method-i-hash) says "The hash value for an object may not be identical across invocations or implementations of Ruby".  I understand it as the same value is not guaranteed.  But the values can be the same.  Also I did not find in the documentation that the hash is strong one.

But if I understand it wrongly or/and the same behaviour is really desirable because some applications assume this, than your patch has sense. Probably the MRI documentation should be changed too as you are adding the tests checking the randomness (it means you are expecting this behaviour).

```
diff --git c/hash.c w/hash.c
index b4c74ed..a5f660e 100644
--- c/hash.c
+++ w/hash.c
@@ -147,24 +147,19 @@ long rb_objid_hash(st_index_t index);

 long
-rb_dbl_long_hash(double d)
+rb_dbl_long_hash(double d, int strong_p)
 {
+    union {double d; st_index_t i;} u;
+    st_index_t hnum;
     /* normalize -0.0 to 0.0 */
     if (d == 0.0) d = 0.0;
-#if SIZEOF_INT == SIZEOF_VOIDP
-    return rb_memhash(&d, sizeof(d));
-#else
-    {
- union {double d; uint64_t i;} u;
-
- u.d = d;
- return rb_objid_hash(u.i);
-    }
-#endif
+    u.d = d;
+    hnum = strong_p ? rb_hash_start(u.i) : u.i;
+    return rb_objid_hash(hnum);
 }
```

I don't like that the same code is now used for 32-bit and 64-bit targets. For ILP32 targets, st_index is 32-bit and basically half of double is thrown away.  This decreases hash function quality considerably.  Depending on target endianess, the result can be pretty bad in most widely used cases. So I would still use rb_memhash for ILP32  and rb_hash_start(rb_memhash(...)) for strong_p and ILP32.

#### #7 - 12/04/2016 06:28 AM - vmakarov (Vladimir Makarov)

Nobuyoshi Nakada wrote:

> test_wrapper_of_special_const failed, that is, it's impossible to emulate switching weak and strong versions under the hood.
> I think this behavior is quirky.

You are right, the behavior with strong/weak hashes is wrong for Ruby.  This test is not guaranteed to work.  After reading the documentation thoroughly, I got a conclusion that hash value used for the hash tables should be *always* the same.  So the hash tables can not use two different hash functions at all.

Although strong/weak hash approach gave about 5-6% improvement out of 45% on Ruby hash table benchmarks, I think we should not use it for Ruby.  I will provide a patch to get rid of it in 2 days.

Sorry for inconvenience.

#### #8 - 12/04/2016 11:04 PM - vmakarov (Vladimir Makarov)

*- File switching_hash_removal.patch added*

Vladimir Makarov wrote:

> Although strong/weak hash approach gave about 5-6% improvement out of 45% on Ruby hash table benchmarks, I think we should not use it for Ruby. I will provide a patch to get rid of it in 2 days.

The following patch removes hash function switching. The patch also randomizes simple type hashes making them strong. The patch was successfully compiled by GCC and LLVM and tested by make check. The actual average performance decrease of the hash tables on Ruby hash table benchmarks is about 3.5% which was measured on 4.2GHz i7-4790K by

```
ruby ../ruby/benchmark/driver.rb -p hash -r 3 -e before::before/miniruby -e current::./miniruby 2>/dev/null|aw
k 'NF==2 && /hash/ {s+=$2;n++;print} END{print s/n}'
```

That still makes the new hash tables by about 42% faster than the old implementation.

Nobuyoshi, I was not sure about adding your tests checking hash randomness. I think it is upto you to add them. The tests should pass after applying the patch.

I also found that for symbols the hash is always the same in previous MRI version (I used 2.3.1). So I keep this behaviour.

Martin, thank you for reporting this issue.

#### #9 - 12/05/2016 10:31 AM - duerst (Martin Dürst)

Hello Victor, others,

On 2016/12/04 15:28, vmakarov@redhat.com wrote:

> Issue #13002 has been updated by Vladimir Makarov.
>
> Nobuyoshi Nakada wrote:
>
> > test_wrapper_of_special_const failed, that is, it's impossible to emulate switching weak and strong versions under the hood.
> > I think this behavior is quirky.
>
> You are right, the behavior with strong/weak hashes is wrong for Ruby. This test is not guaranteed to work. After reading the documentation thoroughly, I got a conclusion that hash value used for the hash tables should be *always* the same. So the hash tables can not use two different hash functions at all.

Over the weekend, I started to think about this issue some more and to write an email. First, I should apologize for not having remembered the strong/weak hash proposal, because I had read it.

But I came up with a very simple example where I had doubts. After Nobu's mail, I saw that this is very similar to the example in Bug #9381.

> Although strong/weak hash approach gave about 5-6% improvement out of 45% on Ruby hash table benchmarks, I think we should not use it for Ruby. I will provide a patch to get rid of it in 2 days.

I think it may still be somewhat too early to completely give up on the strong/weak distinction. I really like using CPU cycles only when necessary, and that's exactly what this proposal is about.

I think we have to distinguish three different cases:

1. Calls to #hash inside a class (as e.g. in Bug #9381):
   This always has to use strong (in the sense of universal, not necessarily in the sense of cryptographically strong) hashing.

2. Calls to #hash for general objects/classes from inside st.c:
   These have to always use strong hashing, because the hash is defined as a method on the class, and can't be switched between weak and strong.

3. Calculations of hash for special objects such as String, Symbol, Integer,... from directly inside st.c: In this case, I think it would be possible to use the weak/strong distinction. I also think that the majority, probably even a big majority, of hash keys are Strings,

> Symbols, and Integers, so that would keep a big part of the savings.
> I haven't studied the code in detail, but I could imagine that
> special-casing for String, Symbol, and (the old Fixnum part of) Integer
> e.g. in do_hash could do the job. Something along the lines of the
> following pseudo-code:

```
/* The reserved hash value and its substitution.  */
#define RESERVED_HASH_VAL (~(st_hash_t) 0)
#define RESERVED_HASH_SUBSTITUTION_VAL ((st_hash_t) 0)

/* Return hash value of KEY for table TAB.  */
static inline st_hash_t
do_hash(st_data_t key, st_table *tab) {
    st_index_t h;
    switch (key.class) {
      case String:
      case Symbol:
      case Fixnum:
        # use weak or strong for basic types
        st_index_t h = (st_index_t)(tab->curr_hash)(key);
        break;
      default:
        # always use strong for all other types
        st_index_t h = (st_index_t)(strong_hash)(key);
    }
    #### aside: Don't see why we need conditional compilation for the
    ####        following 5 lines.
    #if SIZEOF_INT == SIZEOF_VOIDP
    st_hash_t hash = h;
    #else
    st_hash_t hash = h;
    #endif

    /* RESERVED_HASH_VAL is used for a deleted entry.  Map it into
    another value.  Such mapping should be extremely rare.  */
    return hash == RESERVED_HASH_VAL ? RESERVED_HASH_SUBSTITUTION_VAL : hash;
}
```

> I understand that we don't want to 'pollute' st.c with Ruby-specific
> stuff, so this is just pseudocode. It adds a switch, but we can maybe
> balance that by reducing the switch in any_hash_general in hash.c. Even
> if not, the switch may still be faster than a complicated hash
> calculation on a string or symbol, which will loop per character.

Hope this helps, Martin.

#### #10 - 12/05/2016 01:04 PM - nobu (Nobuyoshi Nakada)

In Bug #9381, it claims that st searches/inserts only on hash values and equalities.
Switching hash functions hiddenly from ruby space can't be compatible with Bug #9381.

The candidates would be:

1. revert hash switching,
2. revert Bug #9381,
3. add a way to switch #hash methods,
4. or something else.

#### #11 - 12/05/2016 04:44 PM - vmakarov (Vladimir Makarov)

Martin Dürst wrote:

> I think it may still be somewhat too early to completely give up on the
> strong/weak distinction. I really like using CPU cycles only when
> necessary, and that's exactly what this proposal is about.
>
> I think we have to distinguish three different cases:
>
> 1. Calls to #hash inside a class (as e.g. in Bug #9381):
>    This always has to use strong (in the sense of universal, not
>    necessarily in the sense of cryptographically strong) hashing.
>
> 2. Calls to #hash for general objects/classes from inside st.c:
>    These have to always use strong hashing, because the hash is defined as

a method on the class, and can't be switched between weak and strong.

3. Calculations of hash for special objects such as String, Symbol, Integer,... from directly inside st.c: In this case, I think it would be possible to use the weak/strong distinction. I also think that the majority, probably even a big majority, of hash keys are Strings, Symbols, and Integers, so that would keep a big part of the savings. I haven't studied the code in detail, but I could imagine that special-casing for String, Symbol, and (the old Fixnum part of) Integer e.g. in do_hash could do the job. Something along the lines of the following pseudo-code:

That what I thought until I saw the test you mentioned

```
  def test_wrapper_of_special_const
    bug9381 = '[ruby-core:59638] [Bug #9381]'

    wrapper = Class.new do
      def initialize(obj)
        @obj = obj
      end

      def hash
        @obj.hash
      end

      def eql?(other)
        @obj.eql?(other)
      end
    end

    bad = [
      5, true, false, nil,
      0.0, 1.72723e-77,
      :foo, "dsym_#{self.object_id.to_s(16)}_#{Time.now.to_i.to_s(16)}".to_sym,
    ].select do |x|
      hash = {x => bug9381}
      hash[wrapper.new(x)] != bug9381
    end
    assert_empty(bad, bug9381)
  end
```

You can not use different hash functions for integer (strong for @obj.hash and weak inside the table), otherwise the test will fail. Integer.hash should be the same as hash in the tables (all hash tables). I even think that if the Integer.hash is saved (e.g. in initialize) and then used later in the wrapper object, people still expect the test success. In this case, even simultaneously switching all hashes (for tables and Integer.hash) will not work.

The strong/weak approach could work for other languages though. I wanted to use it in https://github.com/dino-lang/dino and I don't see problems with it. But actually I decided to use mum-hash (https://github.com/vnmakarov/mum-hash) which is fast as the fastest non-crypto hash functions and I believe strong enough to prevent a denial attack even when the seed is known. Using a random seed makes it even stronger. But I can not propose such solution to Ruby community as nobody did a crypto-analysis for mum-hash as for siphash.

```
    #### aside: Don't see why we need conditional compilation for the
    ####        following 5 lines.
    #if SIZEOF_INT == SIZEOF_VOIDP
    st_hash_t hash = h;
    #else
    st_hash_t hash = h;
    #endif
```

I suspect it is a leftover from some experiments with the code. I think about some small improvements for hash tables after 2.4 release. So if nobody removes it, I'll remove it with the changes I am planning.

Thank you, Martin.

#### #12 - 12/06/2016 04:28 AM - duerst (Martin Dürst)

*- Assignee set to naruse (Yui NARUSE)*

*- ruby -v set to ruby 2.4.0dev (2016-12-02 trunk 56965) [x86_64-cygwin]*

Hello Victor,

Thanks for clearing up my confusing re. Bug #9381. It's too bad that that means we can't switch hashing methods.

Nobu - One thing I don't understand is why there wasn't any test failure on CI because there is a test that checks for bug #9318.

Yui - I have assigned this bug to you because it's related to the release of 2.4. Please feel free to reassign it to somebody else (not me :-).

#### #13 - 12/06/2016 04:43 AM - nobu (Nobuyoshi Nakada)

*- Status changed from Open to Closed*

Applied in changeset r56992.

---

switching hash removal

- st.h (struct st_hash_type): Remove strong_hash.
  (struct st_table): Remove inside_rebuild_p and curr_hash.
- st.c (do_hash): Use type->hash instead of curr_hash.
  (make_tab_empty): Remove setting up curr_hash.
  (st_init_table_with_size): Remove setting up inside_rebuild_p.
  (rebuild_table): Remove clearing inside_rebuild_p.
  (reset_entry_hashes, HIT_THRESHOULD_FOR_STRONG_HASH): Remove code
  recognizing a denial attack and switching to strong hash.
- hash.c (rb_dbl_long_hash, rb_objid_hash, rb_ident_hash): Use
  rb_hash_start to randomize the hash.
  (str_seed): Remove.
  (any_hash): Remove strong_p and use always rb_str_hash for
  strings.
  (any_hash_weak, rb_any_hash_weak): Remove.
  (st_hash_type objhash): Remove rb_any_hash_weak.

based on the patch by Vladimir N Makarov vmakarov@redhat.com at
[ruby-core:78490]. [Bug #13002]

- test/ruby/test_hash.rb (test_wrapper): objects other than special
  constants should be able to be wrapped.

#### #14 - 12/06/2016 07:20 AM - nobu (Nobuyoshi Nakada)

Martin Dürst wrote:

> Nobu - One thing I don't understand is why there wasn't any test failure on CI because there is a test that checks for bug #9318.

strong_p argument was used only for Strings, but the test covered only special constants, as its name.
I added tests.

### Files

| | | | |
|---|---|---|---|
| switching_hash_removal.patch | 9.21 KB | 12/04/2016 | vmakarov (Vladimir Makarov) |