Ruby - Feature #3447

argument delegation

06/16/2010 06:57 PM - nobu (Nobuyoshi Nakada)

Status:	Closed
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	

Description

なかだです。

http://www.rubyist.net/~matz/20100615.html#p01 を実装してみました。
foo(...)でブロックまでコミ、foo(..)はブロック抜きにしてあります。

```
diff --git i/compile.c w/compile.c
index 4621cd9..d769c56 100644
--- i/compile.c
+++ w/compile.c
@@ -2729,7 +2729,6 @@ defined_expr(rb_iseq_t *iseq, LINK_ANCHOR *ret,
  return 1;

       case NODE_SUPER:
-      case NODE_ZSUPER:
  ADD_INSN(ret, nd_line(node), putnil);
  ADD_INSN3(ret, nd_line(node), defined, INT2FIX(DEFINED_ZSUPER), 0,
     needstr);
@@ -2919,6 +2918,67 @@ setup_args(rb_iseq_t *iseq, LINK_ANCHOR *args, NODE *argn, unsigned long *f
lag)
      POP_ELEMENT(args);
      break;
    }
+   case NODE_DELEGATE: {
+     int i;
+     rb_iseq_t *liseq = iseq->local_iseq;
+
+     if (argn->nd_state) *flag |= VM_CALL_SUPER_BIT;
+     argc = INT2FIX(liseq->argc);
+
+     /* normal arguments */
+     for (i = 0; i < liseq->argc; i++) {
+  int idx = liseq->local_size - i;
+  ADD_INSN1(args, nd_line(argn), getlocal, INT2FIX(idx));
+     }
+
+     if (!liseq->arg_simple) {
+  if (liseq->arg_opts) {
+      /* optional arguments */
+      int j;
+      for (j = 0; j < liseq->arg_opts - 1; j++) {
+   int idx = liseq->local_size - (i + j);
+   ADD_INSN1(args, nd_line(argn), getlocal, INT2FIX(idx));
+      }
+      i += j;
+      argc = INT2FIX(i);
+  }
+
+  if (liseq->arg_rest != -1) {
+      /* rest argument */
+      int idx = liseq->local_size - liseq->arg_rest;
+      ADD_INSN1(args, nd_line(argn), getlocal, INT2FIX(idx));
+      argc = INT2FIX(liseq->arg_rest + 1);
+      *flag |= VM_CALL_ARGS_SPLAT_BIT;
+  }
```

```
+
+  if (liseq->arg_post_len) {
+      /* post arguments */
+      int post_len = liseq->arg_post_len;
+      int post_start = liseq->arg_post_start;
+
+      if (liseq->arg_rest != -1) {
+   int j;
+   for (j=0; j<post_len; j++) {
+       int idx = liseq->local_size - (post_start + j);
+       ADD_INSN1(args, nd_line(argn), getlocal, INT2FIX(idx));
+   }
+   ADD_INSN1(args, nd_line(argn), newarray, INT2FIX(j));
+   ADD_INSN (args, nd_line(argn), concatarray);
+   /* argc is setteled at above */
+      }
+      else {
+   int j;
+   for (j=0; j<post_len; j++) {
+       int idx = liseq->local_size - (post_start + j);
+       ADD_INSN1(args, nd_line(argn), getlocal, INT2FIX(idx));
+   }
+   argc = INT2FIX(post_len + post_start);
+      }
+  }
+     }
+
+            break;
+          }
    default: {
      rb_bug("setup_arg: unknown node: %s\n", ruby_node_name(nd_type(argn)));
    }
@@ -4115,8 +4175,7 @@ iseq_compile_each(rb_iseq_t *iseq, LINK_ANCHOR *ret, NODE * node, int poped)
  }
  break;
       }
-      case NODE_SUPER:
-      case NODE_ZSUPER:{
+      case NODE_SUPER: {
  DECL_ANCHOR(args);
  VALUE argc;
  unsigned long flag = 0;
@@ -4124,72 +4183,12 @@ iseq_compile_each(rb_iseq_t *iseq, LINK_ANCHOR *ret, NODE * node, int pope
d)

  INIT_ANCHOR(args);
  iseq->compile_data->current_block = Qfalse;
- if (nd_type(node) == NODE_SUPER) {
-     argc = setup_args(iseq, args, node->nd_args, &flag);
- }
- else {
-     /* NODE_ZSUPER */
-     int i;
-     rb_iseq_t *liseq = iseq->local_iseq;
-
-     argc = INT2FIX(liseq->argc);
-
-     /* normal arguments */
-     for (i = 0; i < liseq->argc; i++) {
-  int idx = liseq->local_size - i;
-  ADD_INSN1(args, nd_line(node), getlocal, INT2FIX(idx));
-     }
-
-     if (!liseq->arg_simple) {
-  if (liseq->arg_opts) {
-      /* optional arguments */
-      int j;
```

```
-      for (j = 0; j < liseq->arg_opts - 1; j++) {
-   int idx = liseq->local_size - (i + j);
-   ADD_INSN1(args, nd_line(node), getlocal, INT2FIX(idx));
-      }
-      i += j;
-      argc = INT2FIX(i);
-  }
-
-  if (liseq->arg_rest != -1) {
-      /* rest argument */
-      int idx = liseq->local_size - liseq->arg_rest;
-      ADD_INSN1(args, nd_line(node), getlocal, INT2FIX(idx));
-      argc = INT2FIX(liseq->arg_rest + 1);
-      flag |= VM_CALL_ARGS_SPLAT_BIT;
-  }
-
-  if (liseq->arg_post_len) {
-      /* post arguments */
-      int post_len = liseq->arg_post_len;
-      int post_start = liseq->arg_post_start;
-
-      if (liseq->arg_rest != -1) {
-   int j;
-   for (j=0; j<post_len; j++) {
-       int idx = liseq->local_size - (post_start + j);
-       ADD_INSN1(args, nd_line(node), getlocal, INT2FIX(idx));
-   }
-   ADD_INSN1(args, nd_line(node), newarray, INT2FIX(j));
-   ADD_INSN (args, nd_line(node), concatarray);
-   /* argc is setteled at above */
-      }
-      else {
-   int j;
-   for (j=0; j<post_len; j++) {
-       int idx = liseq->local_size - (post_start + j);
-       ADD_INSN1(args, nd_line(node), getlocal, INT2FIX(idx));
-   }
-   argc = INT2FIX(post_len + post_start);
-      }
-  }
-     }
- }
+ argc = setup_args(iseq, args, node->nd_args, &flag);

  /* dummy receiver */
  ADD_INSN1(ret, nd_line(node), putobject,
-    nd_type(node) == NODE_ZSUPER ? Qfalse : Qtrue);
+    (flag & VM_CALL_SUPER_BIT) ? Qfalse : Qtrue);
+ flag &= ~VM_CALL_SUPER_BIT;
  ADD_SEQ(ret, args);
  ADD_INSN3(ret, nd_line(node), invokesuper,
     argc, parent_block, LONG2FIX(flag));
diff --git i/gc.c w/gc.c
index 58e4550..0d5fbad 100644
--- i/gc.c
+++ w/gc.c
@@ -1671,7 +1671,7 @@ gc_mark_children(rb_objspace_t *objspace, VALUE ptr, int lev)
      goto again;

    case NODE_ZARRAY: /* - */
-   case NODE_ZSUPER:
+   case NODE_DELEGATE:
    case NODE_VCALL:
    case NODE_GVAR:
    case NODE_LVAR:
diff --git i/insns.def w/insns.def
index f75007d..6c1efdc 100644
```

```
--- i/insns.def
+++ w/insns.def
@@ -993,10 +993,10 @@ send
 {
     const rb_method_entry_t *me;
     VALUE recv, klass;
-    rb_block_t *blockptr = 0;
+    rb_block_t *blockptr = (op_flag & VM_CALL_SUPER_BIT) ? GET_BLOCK_PTR() : 0;
     int num = caller_setup_args(th, GET_CFP(), op_flag, (int)op_argc,
     (rb_iseq_t *)blockiseq, &blockptr);
-    rb_num_t flag = op_flag;
+    rb_num_t flag = op_flag & ~VM_CALL_SUPER_BIT;
     ID id = op_id;

     /* get receiver */
diff --git i/node.c w/node.c
index 65bc541..f2900d3 100644
--- i/node.c
+++ w/node.c
@@ -408,10 +408,17 @@ dump_node(VALUE buf, VALUE indent, int comment, NODE *node)
  F_NODE(nd_args, "arguments");
  break;

-      case NODE_ZSUPER:
- ANN("super invocation with no argument");
- ANN("format: super");
- ANN("example: super");
+      case NODE_DELEGATE:
+        if (node->nd_state) {
+     ANN("argument delegation with block");
+     ANN("format: ...");
+     ANN("example: foo(...)");
+        }
+ else {
+     ANN("argument delegation without block");
+     ANN("format: ..");
+     ANN("example: foo(..)");
+ }
  break;

       case NODE_ARRAY:
diff --git i/node.h w/node.h
index f8cf7de..74320c0 100644
--- i/node.h
+++ w/node.h
@@ -96,8 +96,8 @@ enum node_type {
 #define NODE_VCALL       NODE_VCALL
     NODE_SUPER,
 #define NODE_SUPER       NODE_SUPER
-    NODE_ZSUPER,
-#define NODE_ZSUPER      NODE_ZSUPER
+    NODE_DELEGATE,
+#define NODE_DELEGATE    NODE_DELEGATE
     NODE_ARRAY,
 #define NODE_ARRAY       NODE_ARRAY
     NODE_ZARRAY,
@@ -414,7 +414,7 @@ typedef struct RNode {
 #define NEW_FCALL(m,a) NEW_NODE(NODE_FCALL,0,m,a)
 #define NEW_VCALL(m) NEW_NODE(NODE_VCALL,0,m,0)
 #define NEW_SUPER(a) NEW_NODE(NODE_SUPER,0,0,a)
-#define NEW_ZSUPER() NEW_NODE(NODE_ZSUPER,0,0,0)
+#define NEW_DELEGATE(b) NEW_NODE(NODE_DELEGATE,0,0,b)
 #define NEW_ARGS(m,o) NEW_NODE(NODE_ARGS,o,m,0)
 #define NEW_ARGS_AUX(r,b) NEW_NODE(NODE_ARGS_AUX,r,b,0)
 #define NEW_OPT_ARG(i,v) NEW_NODE(NODE_OPT_ARG,i,v,0)
diff --git i/parse.y w/parse.y
index 9fac5bd..735d1bf 100644
```

```
--- i/parse.y
+++ w/parse.y
@@ -2399,6 +2399,20 @@ opt_paren_args : none

 opt_call_args : none
   | call_args
+  | tDOT2
+      {
+      /*%%%*/
+   $$ = NEW_DELEGATE(0);
+      /*%
+      %*/
+      }
+  | tDOT3
+      {
+      /*%%%*/
+   $$ = NEW_DELEGATE(1);
+      /*%
+      %*/
+      }
   ;

 call_args : command
@@ -3647,7 +3661,7 @@ method_call : operation paren_args
   | keyword_super
       {
       /*%%%*/
-   $$ = NEW_ZSUPER();
+   $$ = NEW_SUPER(NEW_DELEGATE(1));
       /*%
    $$ = dispatch0(zsuper);
       %*/
diff --git i/test/ruby/test_method.rb w/test/ruby/test_method.rb
index 7be70b0..a04a285 100644
--- i/test/ruby/test_method.rb
+++ w/test/ruby/test_method.rb
@@ -345,4 +345,38 @@ class TestMethod < Test::Unit::TestCase
     obj.extend(m)
     assert_equal([:m1, :a], obj.public_methods(false), bug)
   end
+
+  def test_argument_delegate
+    class << (o = Object.new)
+      def foo(*args)
+        yield(*args)
+      end
+      def foo1(*)
+        foo(...)
+      end
+      def foo2(*)
+        foo1(...)
+      end
+      def bar(*args, &block)
+        [args, block]
+      end
+      def bar1(*)
+        bar(..)
+      end
+      def bar2(*)
+        bar1(..)
+      end
+    end
+    called = nil
+    assert_equal([42], o.foo1(42) {|*x| called = x})
+    assert_equal([42], called)
+    called = nil
+    assert_equal([42], o.foo2(42) {|*x| called = x})
```

```
+     assert_equal([42], called)
+     called = :not_called
+     assert_equal([[42], nil], o.bar1(42) {|*x| called = true})
+     assert_equal(:not_called, called)
+     assert_equal([[42], nil], o.bar2(42) {|*x| called = true})
+     assert_equal(:not_called, called)
+   end
 end
```

--
--- 僕の前にBugはない。
--- 僕の後ろにBugはできる。
中田 伸悦

Related issues:

Related to Ruby - Feature #11256: anonymous block forwarding	**Closed**

History

#1 - 06/16/2010 07:11 PM - matz (Yukihiro Matsumoto)

まつもと ゆきひろです

In message "Re: [ruby-dev:41623] [Feature:trunk] argument delegation"
on Wed, 16 Jun 2010 18:57:40 +0900, Nobuyoshi Nakada nobu@ruby-lang.org writes:

> http://www.rubyist.net/~matz/20100615.html#p01 を実装してみました。

手が早い。

> foo(...)でブロックまでコミ、foo(..)はブロック抜きにしてあります。

foo(..)は不要だと思います。

#2 - 06/17/2010 03:37 AM - mame (Yusuke Endoh)

遠藤です。

2010年6月16日18:57 Nobuyoshi Nakada nobu@ruby-lang.org:

> http://www.rubyist.net/~matz/20100615.html#p01 を実装してみました。

argument delegation は、

- ブロックを引き継がせたいだけ (引数は変えたい) ときに不便
- 引数の変数へうっかり代入して、間違った引数が渡る事件が多発しそう
- define_method でサポートできていない (将来的にされる予定はある?)

ので、あんまり好きじゃないです。

ブロックを引き継がせるのが面倒くさいという動機はとてもよくわかるので、

`foo(a, b, c, &)`

と書いたらこのコンテキストのブロックを渡す、というのはどうでしょうか。

Yusuke Endoh mame@tsg.ne.jp

#3 - 06/17/2010 06:31 AM - matz (Yukihiro Matsumoto)

まつもと ゆきひろです

In message "Re: [ruby-dev:41625] Re: [Feature:trunk] argument delegation"
on Thu, 17 Jun 2010 03:36:56 +0900, Yusuke ENDOH mame@tsg.ne.jp writes:

> argument delegation は、
>
> 1. ブロックを引き継がせたいだけ (引数は変えたい) ときに不便
> 2. 引数の変数へうっかり代入して、間違った引数が渡る事件が多発しそう
> 3. define_method でサポートできていない (将来的にされる予定はある?)

> > ので、あんまり好きじゃないです。
>
> 1. については、ブロックを引き継がせたい時には、明示的にブロックを引き渡せばよい(ひとつのことをしたいときに、ひとつのことをすればよい。冗長性はない)ので、手当ての必要があるかどうか疑問です。今回の動機は「ひとつのことをするのに、ふたつのことを書かないといけない」という冗長性ですから。
>
> 2. これはsuperでもそうですよね。昔、superは引数の変数に代入してもそのまま渡る仕様だったのですが、実装上の問題であきらめました。
>
> 3. 将来的には、define_methodでもサポートされるべきだと思います。できてないのは実装上の制約だと思ってください。
>
> ブロックを引き継がせるのが面倒くさいという動機はとてもよくわかるので、
>
> foo(a, b, c, &)
>
> と書いたらこのコンテキストのブロックを渡す、というのはどうでしょうか。

繰り返しになりますが、これでは「ひとつのことをするのに、ふたつのことを書かないといけない」という冗長性が解決できてません。実質追加は2文字だから負担は軽くなりますが。

が、もともとの動機とは別として、「&」1文字でそのコンテキストのブロックを渡すのは、面白いアイディアだと思います。この場合は、仮引数リストのブロック引数がなくても、ブロックを渡すことになるんですかね。

まつもと ゆきひろ /:|)

#4 - 06/17/2010 10:45 AM - ko1 (Koichi Sasada)

□ささだです．

(2010/06/17 6:31), Yukihiro Matsumoto wrote::

> > argument delegation は、
> >
> > 1. ブロックを引き継がせたいだけ (引数は変えたい) ときに不便
> > 2. 引数の変数へうっかり代入して、間違った引数が渡る事件が多発しそう
> > 3. define_method でサポートできていない (将来的にされる予定はある?)
> >
> > ので、あんまり好きじゃないです。
>
> 1. については、ブロックを引き継がせたい時には、明示的にブロックを引き渡せばよい(ひとつのことをしたいときに、ひとつのことをすればよい。冗長性はない)ので、手当ての必要があるかどうか疑問です。今回の動機は「ひとつのことをするのに、ふたつのことを書かないといけない」という冗長性ですから。
>
> 2. これはsuperでもそうですよね。昔、superは引数の変数に代入してもそのまま渡る仕様だったのですが、実装上の問題であきらめました。
>
> 3. 将来的には、define_methodでもサポートされるべきだと思います。できてないのは実装上の制約だと思ってください。

□「実装上の制約」はそのとおりで，言語デザインとしてあるべきではないと思います．

（実装が簡単かどうかは別の話ですけど）

□それは置いといて，これ，そんなに使うんでしょうか．あまり使わない機能に，新しい記号を入れるのは抵抗があります．とくに，foo(...) とか，擬似コードでよく使うので，抵抗があります．文脈でわかるっちゃわかるんですが．

利用例:

http://rurema.clear-code.com/query:%28...%29/

□遠藤さんの

> ブロックを引き継がせるのが面倒くさいという動機はとてもよくわかるので、
>
> `foo(a, b, c, &)`
>
> と書いたらこのコンテキストのブロックを渡す、というのはどうでしょうか。

の話は，ああ，あったら使うかも，と思いました．

// SASADA Koichi at atdot dot net

#5 - 06/17/2010 11:09 AM - matz (Yukihiro Matsumoto)

まつもと ゆきひろです

In message "Re: [ruby-dev:41629] Re: [Feature:trunk] argument delegation"
on Thu, 17 Jun 2010 10:45:11 +0900, SASADA Koichi ko1@atdot.net writes:
https://bugs.ruby-lang.org/issues/3447#

> □それは置いといて，これ，そんなに使うんでしょうか．あまり使わない機能
> に，新しい記号を入れるのは抵抗があります．とくに，foo(...) とか，擬似
> コードでよく使うので，抵抗があります．文脈でわかるっちゃわかるんですが．
>
> 利用例:
> http://rurema.clear-code.com/query:%28...%29/

そんなに頻繁に使うかと言うと、そこまでではないと思います。が、
全然ないわけではありませんし、繰り返しになりますが、「やりた
いことに対する冗長性」から考えると手当てをした方がよいと思い
ます。

しかし、「...」にそこまでこだわっているわけではありませんので、
他の文法でも構いません。ただし、RやLuaでは同じ目的に「...」が
使われていますから、そんなに悪い案でもないだろうと思ってます。

#6 - 06/17/2010 12:05 PM - akr (Akira Tanaka)

2010年6月17日6:31 Yukihiro Matsumoto matz@ruby-lang.org:

> > 1. ブロックを引き継がせたいだけ (引数は変えたい) ときに不便
>
> 1. については、ブロックを引き継がせたい時には、明示的にブロッ
> クを引き渡せばよい(ひとつのことをしたいときに、ひとつのこ
> とをすればよい。冗長性はない)ので、手当ての必要があるか
> どうか疑問です。今回の動機は「ひとつのことをするのに、ふ
> たつのことを書かないといけない」という冗長性ですから。

今回の話にはなぜか触れられていないんですが、
この機能の背景のひとつにはキーワード引数があるように思えます。

キーワード引数を加えたときに、キーワード引数が
普通の引数でもブロック引数でもないものとすると、
すべての引数を引き渡すのに 3つそれぞれについて引き渡さないといけません。

それを避けるためにキーワード引数を普通の引数の一部にするということにしたら、
反論が出まくって仕様がまとまらなかった、というのが私の認識です。
ruby-talk:162431

それならすべての引数を引き渡すための機能をつける、というアイデアを
ruby-talk:162561 や ruby-dev:30892 に書いたことがありますが、
その時点では反応を得られませんでした。

そういう流れで今回のを見ると、あり得る方向だと思うわけですが、
2つ懸念があります。

今は普通の引数とブロック引数のふたつしかないので、
ひとつを変えるなら、残りはひとつしかありません。
したがって、残りすべてを引き渡すのはひとつ書けば済みます。

しかし、キーワード引数が入るとみっつになるので、
ひとつを変えるなら、残りはふたつになります。
残りすべてを引き渡すのにふたつ書く必要が出てきます。
おそらくこれはうれしくないでしょう。

つまり、argument delegation はキーワード引数を入れるときに困る問題を解決する
かもしれないと思ったら、実際のところはあまり解決していない、というわけです。

1. define_method でサポートできていない (将来的にされる予定はある？)

1. 将来的には、define_methodでもサポートされるべきだと思いま
   す。できてないのは実装上の制約だと思ってください。

もうひとつは define_method や lambda との関連です。

... という構文には名前が入っていないので、

```
def f(a)
  lambda {|b|
    g(...)
  }
end
```

としたときに ... はどの引数を指すのか不明です。

仕様としてどちらかに固定することはできるでしょうが、
プログラマが選べるような構文にはなっていません。

そのうち、選べるようにする構文が要求されるようになるんじゃないでしょうか。

[田中 哲][たなか あきら][Tanaka Akira]

#7 - 06/17/2010 01:59 PM - shugo (Shugo Maeda)

前田です。

argument delegationについてはまだ態度を決めかねているのですが、
super(...)が導入されたら、(警告を出すなどの移行ステップを経て)
括弧なしsuperはsuper()と同じ意味にしてしまってはどうでしょうか。

2010年6月17日6:31 Yukihiro Matsumoto matz@ruby-lang.org:

1. これはsuperでもそうですよね。昔、superは引数の変数に代入
   してもそのまま渡る仕様だったのですが、実装上の問題であき
   らめました。

superではブロック引数への代入が反映されない動作になっていますが、
argument delegationではどうなりますか？

```
def foo(&b)
  b = lambda { puts "lambda in foo" }
  bar(...)
end
```

というケースの話ですが。

Shugo Maeda

#8 - 06/17/2010 02:53 PM - matz (Yukihiro Matsumoto)

まつもと ゆきひろです

In message "Re: [ruby-dev:41634] Re: [Feature:trunk] argument delegation"
on Thu, 17 Jun 2010 13:58:53 +0900, Shugo Maeda shugo@ruby-lang.org writes:

> argument delegationについてはまだ態度を決めかねているのですが、
> super(...)が導入されたら、(警告を出すなどの移行ステップを経て)
> 括弧なしsuperはsuper()と同じ意味にしてしまってはどうでしょうか。

うーん、superがsuper()の意味であってうれしいのは、

- ルールが少なくなる
- 無引数superの呼び出しが十分に多い

のいずれかだと思うのですが、前者は結局はRubyの設計では優先度
が低い「一貫性」を高めるということなので、この際「うれしい」
とは評価しません。

後者ですが、これが実際的にうれしいのは

- superを含むメソッドが引数を1個以上受け取り
- superにはひとつも渡さない

というケースだけで、superの使い道のうち多くを含む(と思われる)

- superを含むメソッドが引数を1個以上受け取り
- superにそのまま渡す

場合、これまでsuperだけですんだものをsuper(...)と書かねばなら
ず、ちっともうれしくありません。

- superを含むメソッドが引数を1個以上受け取り
- superには別の引数を渡す

場合、argment delegationは役に立たないのは事実ですが、それは
superをsuper()にすることとは無関係でしょう。

> superではブロック引数への代入が反映されない動作になっていますが、
> argument delegationではどうなりますか?
>
> ```
> def foo(&b)
>   b = lambda { puts "lambda in foo" }
>   bar(...)
> end
> ```
>
> というケースの話ですが。

「superと同じ動作」というのが基本だと思います。本当は、通常
引数も代入が反映されないようにしたいのですが、YARVの実装だと
難しいと聞いたような気がします。

```
                                まつもと ゆきひろ /:|)
```

#9 - 06/17/2010 03:03 PM - ko1 (Koichi Sasada)

(2010/06/17 14:53), Yukihiro Matsumoto wrote::

> 「superと同じ動作」というのが基本だと思います。本当は、通常
> 引数も代入が反映されないようにしたいのですが、YARVの実装だと
> 難しいと聞いたような気がします。

□実装じゃなくてパフォーマンスを理由にお願いしました．引数領域のコピー
を，スタックフレーム積むごとに行うのは嫌です（嫌なのは俺だけかも）．

zsuper があったときだけ，というのも **eval** があるから結局ダメ．

□意味的にも，個人的には 1.9 の仕様が好きなのですが.

// SASADA Koichi at atdot dot net

#10 - 06/17/2010 03:15 PM - matz (Yukihiro Matsumoto)

まつもと ゆきひろです

In message "Re: [ruby-dev:41631] Re: [Feature:trunk] argument delegation"

on Thu, 17 Jun 2010 12:05:22 +0900, Tanaka Akira akr@fsij.org writes:

> そういう流れで今回のを見ると、あり得る方向だと思うわけですが、
> 2つ懸念があります。
>
> 今は普通の引数とブロック引数のふたつしかないので、
> ひとつを変えるなら、残りはひとつしかありません。
> したがって、残りすべてを引き渡すのはひとつ書けば済みます。
>
> しかし、キーワード引数が入るとみっつになるので、
> ひとつを変えるなら、残りはふたつになります。
> 残りすべてを引き渡すのにふたつ書く必要が出てきます。
> おそらくこれはうれしくないでしょう。

「ひとつを変える」という意味がいまいち読み取れなかったのです
が、これはたとえば、「通常の引数の前にひとつ別の引数を加える」
とか、「通常引数は同じだが、渡すブロックだけ変える」とかそう
いう意味なんでしょうか。

だとすると、おっしゃることは理解できます。ただし、いまのとこ
ろキーワード引数は通常の引数の一部として扱おうと思っているの
で、あまり問題にはならないかもしれません。

> もうひとつは define_method や lambda との関連です。
>
> ... という構文には名前が入っていないので、
>
> ```
> def f(a)
>   lambda {|b|
>     g(...)
>   }
> end
> ```
>
> としたときに ... はどの引数を指すのか不明です。

確かに。ただし、「一番内側の引数」とか「lambdaは無視、メソッ
ドの引数」でもどちらでもあまり問題ないような気もします。とい
うのも、この記法は、できないことをできるようにするものではな
くて、めんどくさいものをより楽に表現しようというものですから、
仕様の複雑化を避けてある程度で「あきらめる」というのは十分あ
りえる判断だと思います。

> 仕様としてどちらかに固定することはできるでしょうが、
> プログラマが選べるような構文にはなっていません。
>
> そのうち、選べるようにする構文が要求されるようになるんじゃないでしょうか。

そのような仕様のひとつの案としては、「...」そのものを式として
しまって、その値をruby-talk:162561 で示されたようなオブジェ
クトにする方法がありますね。その場合には「**」と「**obj」の方
が良いかもしれませんが。

つまり、

```
def f(a)
  c = **
  lambda {|b|
    g(**c)     # aを渡す
  }
end

def f(a)
  lambda {|b|
    g(**)     # bを渡す
  }
end
```

ということですね。ただ、over specな印象があります。

```
                                まつもと ゆきひろ /:|)
```

#11 - 06/17/2010 03:17 PM - matz (Yukihiro Matsumoto)

まつもと ゆきひろです

In message "Re: [ruby-dev:41636] Re: [Feature:trunk] argument delegation"
on Thu, 17 Jun 2010 15:03:33 +0900, SASADA Koichi ko1@atdot.net writes:

> □実装じゃなくてパフォーマンスを理由にお願いしました．引数領域のコピー
> を，スタックフレーム積むごとに行うのは嫌です（嫌なのは俺だけかも）．

そういえばそうだったような。

> □意味的にも，個人的には 1.9 の仕様が好きなのですが．

特に今のではダメと思ってるわけではないのですが(引数を書き換
えるやつが悪い)、ただし、ささだくんがどうして「1.9 の仕様が
好き」なのか、その理由には関心があります。

まつもと ゆきひろ /:|)

#12 - 06/17/2010 03:42 PM - ko1 (Koichi Sasada)

□ささだです．

(2010/06/17 15:17), Yukihiro Matsumoto wrote::

> > □意味的にも，個人的には 1.9 の仕様が好きなのですが．
>
> 特に今のではダメと思ってるわけではないのですが(引数を書き換
> えるやつが悪い)、ただし、ささだくんがどうして「1.9 の仕様が
> 好き」なのか、その理由には関心があります。

□書き換える自由があるので，そっちのほうがいいかなぁ，と．

// SASADA Koichi at atdot dot net

#13 - 06/17/2010 05:05 PM - shugo (Shugo Maeda)

前田です。

2010年6月17日14:53 Yukihiro Matsumoto matz@ruby-lang.org:

> > argument delegationについてはまだ態度を決めかねているのですが、
> > super(...)が導入されたら、(警告を出すなどの移行ステップを経て)
> > 括弧なしsuperはsuper()と同じ意味にしてしまってはどうでしょうか。
>
> うーん、superがsuper()の意味であってうれしいのは、
>
> - ルールが少なくなる
> - 無引数superの呼び出しが十分に多い
>
> のいずれかだと思うのですが、前者は結局はRubyの設計では優先度
> が低い「一貫性」を高めるということなので、この際「うれしい」
> とは評価しません。

強いていえば前者ですが、ルールが少なくなることそのものがうれしい
のではなくて、通常のメソッド呼出しだと()を付けなくてもよいのに、
superのときだけ引数がないのに()を付けないといけないのが(見た目
的に)気持ち悪い、という理由です。

> 後者ですが、これが実際的にうれしいのは
>
> - superを含むメソッドが引数を１個以上受け取り
> - superにはひとつも渡さない
>
> というケースだけで、

私の経験では、initializeをオーバーライドするときにsuper()と書かないと
いけなくて気持ち悪いと感じることが結構ありました。
ちなみに標準添付ライブラリにおけるsuper/super()の利用箇所を調べたところ

super: 391箇所
super(): 292箇所

という結果でした。意外といい勝負じゃないでしょうか。

> superの使い道のうち多くを含む(と思われる)
>
> - superを含むメソッドが引数を1個以上受け取り
> - superにそのまま渡す
>
> 場合、これまでsuperだけですんだものをsuper(...)と書かねばなら
> ず、ちっともうれしくありません。

5文字の節約が重要になるほどよく使う機能でもないと思うので、
短い記述で済むことよりも、他の機能との類推でsuperの挙動を
推測できることの方が利点が大きいように思います。

ただ、互換性の問題もある(既存のスクリプトは機械的に置換できる
んじゃないかと思いますが)ので、それほど強くは主張しません。

>> superではブロック引数への代入が反映されない動作になっていますが、
>> argument delegationではどうなりますか?
>>
>> ```
>> def foo(&b)
>>   b = lambda { puts "lambda in foo" }
>>   bar(...)
>> end
>> ```
>>
>> というケースの話ですが。
>
> 「superと同じ動作」というのが基本だと思います。本当は、通常
> 引数も代入が反映されないようにしたいのですが、YARVの実装だと
> 難しいと聞いたような気がします。

性能上の理由ということで了解しました。
個人的には、引数に代入するのは悪いスタイルだと思ってるので、どっちでも
いいんですけど。

Shugo Maeda

#14 - 06/17/2010 07:06 PM - matz (Yukihiro Matsumoto)

まつもと ゆきひろです

In message "Re: [ruby-dev:41641] Re: [Feature:trunk] argument delegation"
on Thu, 17 Jun 2010 17:05:10 +0900, Shugo Maeda shugo@ruby-lang.org writes:

> 私の経験では、initializeをオーバーライドするときにsuper()と書かないと
> いけなくて気持ち悪いと感じることが結構ありました。
> ちなみに標準添付ライブラリにおけるsuper/super()の利用箇所を調べたところ
>
> super: 391箇所
> super(): 292箇所
>
> という結果でした。意外といい勝負じゃないでしょうか。

うーん、私が lib/**/*.rb で数えたら super()、つまり明示的に無
引数で呼び出したsuperは66箇所しか見つかりませんでした。うち2
箇所はメソッドに引数がないのでsuperでも同じことです。逆に暗黙
に引数を引き渡すsuperは243箇所です。少々の数えもれがあっても
おかしくないですが、あまりにも違うので、もしかして数え方が違
う?

> 5文字の節約が重要になるほどよく使う機能でもないと思うので、
> 短い記述で済むことよりも、他の機能との類推でsuperの挙動を
> 推測できることの方が利点が大きいように思います。

私、けっこう(引数転送の) super 使うんだけどな。

```
                                        まつもと ゆきひろ　/:|)
```

#15 - 06/17/2010 10:29 PM - mame (Yusuke Endoh)

遠藤です。

2010年6月17日6:31 Yukihiro Matsumoto matz@ruby-lang.org:

> 1. ブロックを引き継がせたいだけ (引数は変えたい) ときに不便

> 1. (略) 今回の動機は「ひとつのことをするのに、ふ
> たつのことを書かないといけない」という冗長性ですから。

なるほど。「通常の引数とブロック引数をまとめて delegate したい」
と感じたことがあまりなかったので、動機を勘違いしていました。

lib/ を粗っぽく探してみたら、24 箇所ほど見つかりました (メールの
最後に貼りつけてあります) 。それなりに需要はあるのかな。

> が、もともとの動機とは別として、「&」1文字でそのコンテキスト
> のブロックを渡すのは、面白いアイディアだと思います。この場合
> は、仮引数リストのブロック引数がなくても、ブロックを渡すこと
> になるんですかね。

通常の引数は関係なく、ブロック引数を引き継がせたい事例なら、上と
同じような検索で 183 箇所 (上の 24 箇所を含む) も見つかったので、
& の省略記法は是非欲しくなってきました。

```
$ ruby check.rb
lib/drb/drb.rb
1079:    def method_missing(msg_id, *a, &b)
1083:   return obj.__send__(msg_id, *a, &b)

lib/test/unit/assertions.rb
22:      def assert_raise(*args, &b)
23:        assert_raises(*args, &b)

lib/rake.rb
743:      def create_rule(*args, &block)
744:        Rake.application.create_rule(*args, &block)

lib/rake.rb
844:def task(*args, &block)
845:  Rake::Task.define_task(*args, &block)

lib/rake.rb
862:def file(*args, &block)
863:  Rake::FileTask.define_task(*args, &block)

lib/rake.rb
868:def file_create(args, &block)
869:  Rake::FileCreationTask.define_task(args, &block)

lib/rake.rb
892:def multitask(args, &block)
893:  Rake::MultiTask.define_task(args, &block)

lib/rake.rb
918:def rule(*args, &block)
919:  Rake::Task.create_rule(*args, &block)

lib/scanf.rb
607:  def scanf(str,&b)
608:    return block_scanf(str,&b) if b

lib/scanf.rb
687:  def scanf(fstr,&b)
689:      block_scanf(fstr,&b)
```

```
lib/scanf.rb
716:  def scanf(fs,&b)
717:    STDIN.scanf(fs,&b)

lib/rubygems/user_interaction.rb
62:  def use_ui(new_ui, &block)
63:    Gem::DefaultUserInteraction.use_ui(new_ui, &block)

lib/optparse.rb
832:  def accept(*args, &blk) top.accept(*args, &blk) end
836:  def self.accept(*args, &blk) top.accept(*args, &blk) end

lib/optparse.rb
1202:  def on(*opts, &block)
1203:    define(*opts, &block)

lib/optparse.rb
1216:  def on_head(*opts, &block)
1217:    define_head(*opts, &block)

lib/optparse.rb
1230:  def on_tail(*opts, &block)
1231:    define_tail(*opts, &block)

lib/optparse.rb
1425:  def visit(id, *args, &block)
1427:      el.send(id, *args, &block)

lib/csv.rb
2308:def CSV(*args, &block)
2309:  CSV.instance(*args, &block)

lib/erb.rb
334:      def percent_line(line, &block)
336:      return @scan_line.call(line, &block)

lib/erb.rb
334:      def percent_line(line, &block)
341:      @scan_line.call(line, &block)

lib/matrix.rb
1457:  def map2(v, &block) # :yield: e1, e2
1459:    els = collect2(v, &block)

lib/set.rb
613:  def initialize(*args, &block)     # :nodoc:
615:    initialize(*args, &block)

lib/delegate.rb
141:  def method_missing(m, *args, &block)
144:      target.respond_to?(m) ? target.__send__(m, *args, &block) :
super(m, *args, &block)

lib/open-uri.rb
27:  def open(name, *rest, &block) # :doc:
35:      open_uri_original_open(name, *rest, &block)

count: 24

$ cat check.rb
count = 0
Dir.glob("lib/**/*.rb").each do |path|
  next unless File.file?(path)
  open(path) do |file|
    while line = file.gets
      if line =~ /^(\s*)def\s+[\d\w]+(\(.*,.*&.*\))/ # □□□□□□□□□□□
      #if line =~ /^(\s*)def\s+[\d\w]+\(.*(&.*\))/   # □□□□□□□□□□□□□□□□□
        sig = line
        lineno = file.lineno
        re1 = /^\s{#{ $1.size }}end/
        re2 = Regexp.new(Regexp.quote($2))
        while line = file.gets
          break if re1 =~ line
          if re2 =~ line
```

```
            puts path
            puts "#{ lineno }:#{ sig }"
            puts "#{ file.lineno }:#{ line }"
            puts
            count += 1
          end
        end
      end
    end
  end
end
puts "count: #{ count }"
```

Yusuke Endoh mame@tsg.ne.jp

#16 - 06/18/2010 05:43 AM - matz (Yukihiro Matsumoto)

まつもと ゆきひろです

In message "Re: [ruby-dev:41644] Re: [Feature:trunk] argument delegation"
on Thu, 17 Jun 2010 22:29:19 +0900, Yusuke ENDOH mame@tsg.ne.jp writes:

> 通常の引数は関係なく、ブロック引数を引き継がせたい事例なら、上と
> 同じような検索で 183 箇所 (上の 24 箇所を含む) も見つかったので、
> & の省略記法は是非欲しくなってきました。

これは良いアイディアだと思います。もしかすると元のアイディア
よりも。

#17 - 06/18/2010 08:34 AM - shugo (Shugo Maeda)

前田です。

2010年6月17日19:06 Yukihiro Matsumoto matz@ruby-lang.org:

>> 私の経験では、initializeをオーバーライドするときにsuper()と書かないと
>> いけなくて気持ち悪いと感じることが結構ありました。
>> ちなみに標準添付ライブラリにおけるsuper/super()の利用箇所を調べたところ
>>
>> super: 391箇所
>> super(): 292箇所
>>
>> という結果でした。意外といい勝負じゃないでしょうか。
>
> うーん、私が lib/**/*.rb で数えたら super()、つまり明示的に無
> 引数で呼び出したsuperは66箇所しか見つかりませんでした。うち2
> 箇所はメソッドに引数がないのでsuperでも同じことです。逆に暗黙
> に引数を引き渡すsuperは243箇所です。少々の数えもれがあっても
> おかしくないですが、あまりにも違うので、もしかして数え方が違
> う?

~/src/ruby/**/*.rbでやってました。
で、多いのはTkみたいでした…。

>> 5文字の節約が重要になるほどよく使う機能でもないと思うので、
>> 短い記述で済むことよりも、他の機能との類推でsuperの挙動を
>> 推測できることの方が利点が大きいように思います。
>
> 私、けっこう (引数転送の) super 使うんだけどな。

まあ、自分も結構使ってるかもしれませんが、うれしかったことは
すぐに忘れて、不満に思ったことだけ覚えているということかも
しれません。

Shugo Maeda

#18 - 08/25/2010 12:30 AM - mame (Yusuke Endoh)

遠藤です。

2010年6月17日6:31 Yukihiro Matsumoto matz@ruby-lang.org:

> が、もともとの動機とは別として、「&」1文字でそのコンテキスト
> のブロックを渡すのは、面白いアイディアだと思います。この場合
> は、仮引数リストのブロック引数がなくても、ブロックを渡すこと
> になるんですかね。

パッチを書いてみました。思ったより小さく書けました。

```
def foo
  yield
end

def bar
  foo(&)
end

bar { p 1 }  #=> 1
```

カッコを省略した場合、

```
foo &
bar
```

というのが foo & bar なのか foo(&); bar なのか曖昧という問題が
ありました。が、互換性重視で foo & bar と解釈するようになってい
ます。というか書いてみたらそうなった。conflict は出ません。

```
diff --git a/compile.c b/compile.c
index 2fd804c..3f8c331 100644
--- a/compile.c
+++ b/compile.c
@@ -2863,8 +2863,13 @@ setup_args(rb_iseq_t *iseq, LINK_ANCHOR *args,
NODE *argn, unsigned long *flag)
     INIT_ANCHOR(arg_block);
     INIT_ANCHOR(args_splat);
     if (argn && nd_type(argn) == NODE_BLOCK_PASS) {
- COMPILE(arg_block, "block", argn->nd_body);
- *flag |= VM_CALL_ARGS_BLOCKARG_BIT;
+ if ((VALUE)argn->nd_body != (VALUE)-1) {
+     COMPILE(arg_block, "block", argn->nd_body);
+     *flag |= VM_CALL_ARGS_BLOCKARG_BIT;
+ }
+ else {
+     *flag |= VM_CALL_ARGS_BLOCKTRGH_BIT;
+ }
   argn = argn->nd_head;
     }

diff --git a/node.c b/node.c
index 65bc541..85574b4 100644
--- a/node.c
+++ b/node.c
@@ -621,7 +621,12 @@ dump_node(VALUE buf, VALUE indent, int comment, NODE *node)
   ANN("example: foo(x, &blk)");
   F_NODE(nd_head, "other arguments");
   LAST_NODE;
- F_NODE(nd_body, "block argument");
+ if ((VALUE)node->nd_body != (VALUE)-1) {
+     F_NODE(nd_body, "block argument");
+ }
+ else {
+     F_MSG(nd_body, "block argument", "-1 (anonymous)");
+ }
   break;

       case NODE_DEFN:
diff --git a/parse.y b/parse.y
index e085088..b0b946b 100644
--- a/parse.y
+++ b/parse.y
@@ -2459,6 +2459,14 @@ block_arg : tAMPER arg_value
```

```
    $$ = $2;
       %*/
       }
+  | tAMPER
+      {
+      /*%%%*/
+    $$ = NEW_BLOCK_PASS(-1);
+      /*%
+    $$ = dispatch0(anonymous_block_arg);
+      %*/
+      }
   ;

 opt_block_arg : ',' block_arg
diff --git a/vm_core.h b/vm_core.h
index 7676b2f..f0fa3a3 100644
--- a/vm_core.h
+++ b/vm_core.h
@@ -544,6 +544,7 @@ typedef struct {
 #define VM_CALL_TAILRECURSION_BIT  (0x01 << 6)
 #define VM_CALL_SUPER_BIT          (0x01 << 7)
 #define VM_CALL_OPT_SEND_BIT       (0x01 << 8)
+#define VM_CALL_ARGS_BLOCKTRGH_BIT (0x01 << 9)

 #define VM_SPECIAL_OBJECT_VMCORE       0x01
 #define VM_SPECIAL_OBJECT_CBASE        0x02
diff --git a/vm_insnhelper.c b/vm_insnhelper.c
index 985a2fb..2a127d7 100644
--- a/vm_insnhelper.c
+++ b/vm_insnhelper.c
@@ -261,6 +261,10 @@ caller_setup_args(const rb_thread_t *th,
rb_control_frame_t *cfp, VALUE flag,
   *block = blockptr;
       }
   }
+ else if (flag & VM_CALL_ARGS_BLOCKTRGH_BIT) {
+     rb_control_frame_t *reg_cfp = cfp;
+     *block = GET_BLOCK_PTR();
+ }
   else if (blockiseq) {
       blockptr = RUBY_VM_GET_BLOCK_PTR_IN_CFP(cfp);
       blockptr->iseq = blockiseq;
```

Yusuke Endoh mame@tsg.ne.jp

#19 - 08/25/2010 03:23 AM - ko1 (Koichi Sasada)

□□□□□□□□

□□□□□□□□□□□□□□□□□□□

(2010/08/25 0:30), Yusuke ENDOH wrote:

```
    --- a/vm_core.h
    +++ b/vm_core.h
    @@ -544,6 +544,7 @@ typedef struct {
     #define VM_CALL_TAILRECURSION_BIT  (0x01 << 6)
     #define VM_CALL_SUPER_BIT          (0x01 << 7)
     #define VM_CALL_OPT_SEND_BIT       (0x01 << 8)
    +#define VM_CALL_ARGS_BLOCKTRGH_BIT (0x01 << 9)

     #define VM_SPECIAL_OBJECT_VMCORE       0x01
     #define VM_SPECIAL_OBJECT_CBASE        0x02
    diff --git a/vm_insnhelper.c b/vm_insnhelper.c
    index 985a2fb..2a127d7 100644
    --- a/vm_insnhelper.c
    +++ b/vm_insnhelper.c
    @@ -261,6 +261,10 @@ caller_setup_args(const rb_thread_t *th,
    rb_control_frame_t *cfp, VALUE flag,
       *block = blockptr;
           }
       }
    + else if (flag & VM_CALL_ARGS_BLOCKTRGH_BIT) {
    +     rb_control_frame_t *reg_cfp = cfp;
```

```
+     *block = GET_BLOCK_PTR();
+ }
  else if (blockiseq) {
      blockptr = RUBY_VM_GET_BLOCK_PTR_IN_CFP(cfp);
      blockptr->iseq = blockiseq;
```

□できれば，VM_CALL_ARGS_BLOCKTRGH_BIT を増やすんじゃなく
て，VM_CALL_ARGS_BLOCKARG_BIT だけど，blockiseq□かなんかの値を特殊にし
ておくとか，そういう実装にしてもらえるといいんじゃないかと思います．

（命令オペランドのバリエーションが減った方が色々うれしい）

--
// SASADA Koichi at atdot dot net

#20 - 08/25/2010 09:07 PM - mame (Yusuke Endoh)

遠藤です。

2010年8月25日3:23 SASADA Koichi ko1@atdot.net:

> □できれば，VM_CALL_ARGS_BLOCKTRGH_BIT を増やすんじゃなく
> て，VM_CALL_ARGS_BLOCKARG_BIT だけど，blockiseq□かなんかの値を特殊にし
> ておくとか，そういう実装にしてもらえるといいんじゃないかと思います．
>
> （命令オペランドのバリエーションが減った方が色々うれしい）

コメントありがとうございます。でも、見かけ上バリエーションを増やさない
ところで、VM_CALL_ARGS_BLOCKTRGH_BIT に結構異なる 2 つの意味を突っ込む
ことになる (スタックに Proc を載せる・載せないの違いがあるので、細々条
件判定が必要になる) ので、大して嬉しくないと思います。メンテナンス性も
下がりますので、元のパッチの方がいいと思います。

一応 blockiseq に -1 が入ってたら BLOCKTRGH 扱いにするパッチも書いてみ
ましたが、かなり本質的でないコードでいっぱいになりました。
今 trunk が壊れてるようなので、test-all はできてません。

```
diff --git a/compile.c b/compile.c
index 2fd804c..1c04640 100644
--- a/compile.c
+++ b/compile.c
@@ -1411,11 +1411,18 @@ iseq_set_sequence(rb_iseq_t *iseq, LINK_ANCHOR *anchor)
         case TS_ISEQ: /* iseq */
    {
        VALUE v = operands[j];
-       rb_iseq_t *block = 0;
+       VALUE block = 0;
        if (v) {
-    GetISeqPtr(v, block);
+    if (v == Qundef) {
+        block = (VALUE)-1;
+    }
+    else {
+        rb_iseq_t *blk;
+        GetISeqPtr(v, blk);
+        block = (VALUE)blk;
+    }
        }
-       generated_iseq[pos + 1 + j] = (VALUE)block;
+       generated_iseq[pos + 1 + j] = block;
        break;
    }
         case TS_VALUE: /* VALUE */
@@ -2853,7 +2860,7 @@ add_ensure_iseq(LINK_ANCHOR *ret, rb_iseq_t *iseq, int is_return)
 }

 static VALUE
-setup_args(rb_iseq_t *iseq, LINK_ANCHOR *args, NODE *argn, unsigned long *flag)
+setup_args(rb_iseq_t *iseq, LINK_ANCHOR *args, NODE *argn, VALUE *blockiseq, unsigned long *flag)
 {
     VALUE argc = INT2FIX(0);
     int nsplat = 0;
```

```
@@ -2863,7 +2870,13 @@ setup_args(rb_iseq_t *iseq, LINK_ANCHOR *args, NODE *argn, unsigned long *flag)
     INIT_ANCHOR(arg_block);
     INIT_ANCHOR(args_splat);
     if (argn && nd_type(argn) == NODE_BLOCK_PASS) {
- COMPILE(arg_block, "block", argn->nd_body);
+ if ((VALUE)argn->nd_body != (VALUE)-1) {
+     COMPILE(arg_block, "block", argn->nd_body);
+ }
+ else {
+     arg_block = 0;
+     if (blockiseq) *blockiseq = Qundef;
+ }
  *flag |= VM_CALL_ARGS_BLOCKARG_BIT;
  argn = argn->nd_head;
     }
@@ -2932,7 +2945,7 @@ setup_args(rb_iseq_t *iseq, LINK_ANCHOR *args, NODE *argn, unsigned long *flag)
  ADD_SEQ(args, args_splat);
     }

-    if (*flag & VM_CALL_ARGS_BLOCKARG_BIT) {
+    if ((*flag & VM_CALL_ARGS_BLOCKARG_BIT) && arg_block) {
  ADD_SEQ(args, arg_block);
     }
     return argc;
@@ -3776,7 +3789,7 @@ iseq_compile_each(rb_iseq_t *iseq, LINK_ANCHOR *ret, NODE * node, int poped)
      boff = 1;
    default:
      INIT_ANCHOR(args);
-     argc = setup_args(iseq, args, node->nd_args->nd_head, &flag);
+     argc = setup_args(iseq, args, node->nd_args->nd_head, 0, &flag);
      ADD_SEQ(ret, args);
  }
  ADD_INSN1(ret, nd_line(node), dupn, FIXNUM_INC(argc, 1 + boff));
@@ -4083,7 +4096,7 @@ iseq_compile_each(rb_iseq_t *iseq, LINK_ANCHOR *ret, NODE * node, int poped)

  /* args */
  if (nd_type(node) != NODE_VCALL) {
-     argc = setup_args(iseq, args, node->nd_args, &flag);
+     argc = setup_args(iseq, args, node->nd_args, &parent_block, &flag);
  }
  else {
     argc = INT2FIX(0);
@@ -4121,7 +4134,7 @@ iseq_compile_each(rb_iseq_t *iseq, LINK_ANCHOR *ret, NODE * node, int poped)
  INIT_ANCHOR(args);
  iseq->compile_data->current_block = Qfalse;
  if (nd_type(node) == NODE_SUPER) {
-     argc = setup_args(iseq, args, node->nd_args, &flag);
+     argc = setup_args(iseq, args, node->nd_args, &parent_block, &flag);
  }
  else {
     /* NODE_ZSUPER */
@@ -4294,7 +4307,7 @@ iseq_compile_each(rb_iseq_t *iseq, LINK_ANCHOR *ret, NODE * node, int poped)
  }

  if (node->nd_head) {
-     argc = setup_args(iseq, args, node->nd_head, &flag);
+     argc = setup_args(iseq, args, node->nd_head, 0, &flag);
  }
  else {
     argc = INT2FIX(0);
@@ -4906,7 +4919,7 @@ iseq_compile_each(rb_iseq_t *iseq, LINK_ANCHOR *ret, NODE * node, int poped)

  INIT_ANCHOR(recv);
  INIT_ANCHOR(args);
- argc = setup_args(iseq, args, node->nd_args, &flag);
+ argc = setup_args(iseq, args, node->nd_args, 0, &flag);

  if (node->nd_recv == (NODE *) 1) {
     flag |= VM_CALL_FCALL_BIT;
diff --git a/insns.def b/insns.def
index fcd97ae..10b99be 100644
--- a/insns.def
+++ b/insns.def
@@ -989,7 +989,7 @@ DEFINE_INSN
 send
```

```
 (ID op_id, rb_num_t op_argc, ISEQ blockiseq, rb_num_t op_flag, IC ic)
 (...)
-(VALUE val) // inc += - (int)(op_argc + ((op_flag & VM_CALL_ARGS_BLOCKARG_BIT) ? 1 : 0));
+(VALUE val) // inc += - (int)(op_argc + (((op_flag & VM_CALL_ARGS_BLOCKARG_BIT) && (VALUE)blockiseq != (VALUE
)-1) ? 1 : 0));
 {
     const rb_method_entry_t *me;
     VALUE recv, klass;
@@ -1017,7 +1017,7 @@ DEFINE_INSN
 invokesuper
 (rb_num_t op_argc, ISEQ blockiseq, rb_num_t op_flag)
 (...)
-(VALUE val) // inc += - (int)(op_argc + ((op_flag & VM_CALL_ARGS_BLOCKARG_BIT) ? 1 : 0));
+(VALUE val) // inc += - (int)(op_argc + (((op_flag & VM_CALL_ARGS_BLOCKARG_BIT) && (VALUE)blockiseq != (VALUE
)-1) ? 1 : 0));
 {
     rb_block_t *blockptr = !(op_flag & VM_CALL_ARGS_BLOCKARG_BIT) ? GET_BLOCK_PTR() : 0;
     int num = caller_setup_args(th, GET_CFP(), op_flag,
diff --git a/node.c b/node.c
index 65bc541..85574b4 100644
--- a/node.c
+++ b/node.c
@@ -621,7 +621,12 @@ dump_node(VALUE buf, VALUE indent, int comment, NODE *node)
  ANN("example: foo(x, &blk)");
  F_NODE(nd_head, "other arguments");
  LAST_NODE;
- F_NODE(nd_body, "block argument");
+ if ((VALUE)node->nd_body != (VALUE)-1) {
+     F_NODE(nd_body, "block argument");
+ }
+ else {
+     F_MSG(nd_body, "block argument", "-1 (anonymous)");
+ }
  break;

       case NODE_DEFN:
diff --git a/parse.y b/parse.y
index e085088..b0b946b 100644
--- a/parse.y
+++ b/parse.y
@@ -2459,6 +2459,14 @@ block_arg : tAMPER arg_value
     $$ = $2;
        %*/
        }
+  | tAMPER
+       {
+       /*%%%*/
+    $$ = NEW_BLOCK_PASS(-1);
+       /*%
+    $$ = dispatch0(anonymous_block_arg);
+       %*/
+       }
    ;

 opt_block_arg : ',' block_arg
diff --git a/tool/instruction.rb b/tool/instruction.rb
index 4fd2127..4ce219f 100755
--- a/tool/instruction.rb
+++ b/tool/instruction.rb
@@ -66,13 +66,12 @@ class RubyVM
         ret = "int inc = 0;\n"

         @opes.each_with_index{|(t, v), i|
-          if t == 'rb_num_t' && ((re = /\b#{v}\b/n) =~ @sp_inc ||
-                                 @defopes.any?{|t, val| re =~ val})
+          if ((re = /\b#{v}\b/n) =~ @sp_inc || @defopes.any?{|t, val| re =~ val})
             ret << "        int #{v} = FIX2INT(opes[#{i}]);\n"
           end
         }
         @defopes.each_with_index{|((t, var), val), i|
-          if t == 'rb_num_t' && val != '*' && /\b#{var}\b/ =~ @sp_inc
+          if val != '*' && /\b#{var}\b/ =~ @sp_inc
             ret << "        #{t} #{var} = #{val};\n"
           end
         }
```

```
diff --git a/vm_insnhelper.c b/vm_insnhelper.c
index 985a2fb..f6418b7 100644
--- a/vm_insnhelper.c
+++ b/vm_insnhelper.c
@@ -240,25 +240,31 @@ caller_setup_args(const rb_thread_t *th, rb_control_frame_t *cfp, VALUE flag,
```

```
     if (block) {
  if (flag & VM_CALL_ARGS_BLOCKARG_BIT) {
-     rb_proc_t *po;
-     VALUE proc;
-
-     proc = *(--cfp->sp);
-
-     if (proc != Qnil) {
-  if (!rb_obj_is_proc(proc)) {
-      VALUE b = rb_check_convert_type(proc, T_DATA, "Proc", "to_proc");
-      if (NIL_P(b) || !rb_obj_is_proc(b)) {
-   rb_raise(rb_eTypeError,
-     "wrong argument type %s (expected Proc)",
-     rb_obj_classname(proc));
+     if ((VALUE)blockiseq != (VALUE)-1) {
+  rb_proc_t *po;
+  VALUE proc;
+
+  proc = *(--cfp->sp);
+
+  if (proc != Qnil) {
+      if (!rb_obj_is_proc(proc)) {
+   VALUE b = rb_check_convert_type(proc, T_DATA, "Proc", "to_proc");
+   if (NIL_P(b) || !rb_obj_is_proc(b)) {
+       rb_raise(rb_eTypeError,
+         "wrong argument type %s (expected Proc)",
+         rb_obj_classname(proc));
+   }
+   proc = b;
       }
-      proc = b;
+      GetProcPtr(proc, po);
+      blockptr = &po->block;
+      RUBY_VM_GET_BLOCK_PTR_IN_CFP(cfp)->proc = proc;
+      *block = blockptr;
   }
-  GetProcPtr(proc, po);
-  blockptr = &po->block;
-  RUBY_VM_GET_BLOCK_PTR_IN_CFP(cfp)->proc = proc;
-  *block = blockptr;
+     }
+     else {
+  rb_control_frame_t *reg_cfp = cfp;
+  *block = GET_BLOCK_PTR();
      }
  }
  else if (blockiseq) {
```

Yusuke Endoh mame@tsg.ne.jp

#21 - 08/26/2010 11:28 PM - yugui (Yuki Sonoda)

Yuguiです

2010/6/17 Yukihiro Matsumoto matz@ruby-lang.org:

> が、もともとの動機とは別として、「&」1文字でそのコンテキスト
> のブロックを渡すのは、面白いアイディアだと思います。この場合
> は、仮引数リストのブロック引数がなくても、ブロックを渡すこと
> になるんですかね。

深い呼び出し階層でブロックを受け取るために、上位のメソッドが延々とブロックを引き渡し続けるっていうケースは実際にプログラムしていてよく出会います。ですから、このとき下位呼び出しの実引数に&blockを足して、仮引数にも忘れずに&blockを足してというのは面倒で、このアイディアは便利そうです。

しかし、あまりにも短すぎる。&1文字は見落としそうです。

```
foo(a, b, &yield)
```

なんてのは駄目でしょうか。yield と block.callの類似性からの類推です。

Yuki Sonoda (Yugui)
yugui@yugui.jp
http://yugui.jp

#22 - 09/02/2010 03:06 AM - mame (Yusuke Endoh)

遠藤です。

2010年8月26日23:28 Yugui yugui@yugui.jp:

> しかし、あまりにも短すぎる。& 1文字は見落としそうです。

いやあ、考えすぎでしょう。
func(foo + bar, baz * qux, i) で i を見落とすとかないですよ。
& は i より目立つし。

> `foo(a, b, &yield)`
>
> なんてのは駄目でしょうか。yield と block.callの類似性からの類推です。

それは互換性に問題があります。

Yusuke Endoh mame@tsg.ne.jp

#23 - 03/18/2012 05:19 PM - shyouhei (Shyouhei Urabe)

- Description updated

- Status changed from Open to Feedback

- このスレッドの現状をどなたか教えていただけませんか。
- この記法のユースケースをどなたか教えていただけませんか。

#24 - 05/24/2012 11:54 PM - nobu (Nobuyoshi Nakada)

- Description updated

#25 - 11/20/2012 10:54 PM - mame (Yusuke Endoh)

- Status changed from Feedback to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

- Target version set to 2.6

shyouhei (Shyouhei Urabe) wrote:

> - このスレッドの現状をどなたか教えていただけませんか。

matz の最終的な go ahead 待ちだと思いますが、2.0.0 の feature deadline は過ぎたので next minor に。

> - この記法のユースケースをどなたか教えていただけませんか。

「この記法」は matz の元提案の話？ それとも & の話？
後者だとしたら、

```
def foo; ... yield ... end
foo { ... }
```

と書いてたんだけど、リファクタリングで foo の中身を bar にくくり出すことになった、というとき、

```
def bar; ... yield ... end
def foo(&blk); ... bar(&blk) ... end
foo { ... }
```

とすると思いますが、blk という一時変数を定義したくない and/or ブロックが Proc 化するので重くなるのが嫌、とかですかね。

Yusuke Endoh mame@tsg.ne.jp

#26 - 02/14/2014 10:10 AM - nobu (Nobuyoshi Nakada)

- Description updated

#27 - 06/22/2015 07:44 AM - matz (Yukihiro Matsumoto)

- Related to Feature #11256: anonymous block forwarding added

#28 - 12/25/2017 06:14 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)

#29 - 10/08/2021 10:04 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Assigned to Closed