Ruby master - Feature #11256

anonymous block forwarding

06/12/2015 08:18 PM - bughit (bug hit)

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

Description

since capturing a block into a proc is slow: foo(&block)
and creating chains of blocks is kind of ugly and ultimately also inefficient: foo{yield}
why not allow block forwarding without capturing: foo(&) foo(1, 2, &)

Related issues:

Related to Ruby master - Feature #3447: argument delegation	**Assigned**
Related to Ruby master - Feature #14045: Lazy Proc allocation for block param...	**Closed**

History

#1 - 06/13/2015 06:44 PM - bughit (bug hit)

```
# takes a block
def bar
  # forwards it to foo without instantiating a proc
  foo(1, &)
end
```

#2 - 06/22/2015 07:44 AM - matz (Yukihiro Matsumoto)

- Related to Feature #3447: argument delegation added

#3 - 11/27/2017 11:50 AM - mame (Yusuke Endoh)

- Related to Feature #14045: Lazy Proc allocation for block parameters added

#4 - 11/27/2017 11:55 AM - mame (Yusuke Endoh)

- Assignee set to matz (Yukihiro Matsumoto)

- Status changed from Open to Assigned

Lazy Proc allocation for block parameters (#14045) is implemented, so capturing a block into a proc is not slow.

One of the motivation now disappeared, but there is another motivation of this feature: the simplicity of the notation. Matz, is this syntax still hopeful to be accepted?

#5 - 11/29/2017 09:45 AM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Closed

As mame (Yusuke Endoh) stated, we don't need it after we had lazy Proc allocation.

Matz.

#6 - 12/04/2017 07:49 AM - ko1 (Koichi Sasada)

- Status changed from Closed to Assigned

Matz:

> but there is another motivation of this feature: the simplicity of the notation

With proposal syntax, we don't need to use a variable name for block parameter. What do you think about it?

#7 - 12/12/2017 08:34 AM - matz (Yukihiro Matsumoto)

Accepted.

Matz.

#8 - 12/12/2017 09:45 AM - nobu (Nobuyoshi Nakada)

Just to be clear, does it require both of the definition and the use, instead of only the latter?

I mean this is allowed:

```
def foo(&)
  bar(&)
end
```

but these are not:

```
def foo()
  bar(&)
end

def foo(&block)
  bar(&)
end
```

#9 - 12/12/2017 10:04 AM - nobu (Nobuyoshi Nakada)

Current patch.

```
diff --git a/parse.y b/parse.y
index 1026d5c896..2a98016002 100644
--- a/parse.y
+++ b/parse.y
@@ -347,6 +347,8 @@ static int parser_yyerror(struct parser_params*, const char*);

 #define lambda_beginning_p() (lpar_beg && lpar_beg == paren_nest)

+#define ANON_BLOCK_ID '&'
+
 static enum yytokentype yylex(YYSTYPE*, YYLTYPE*, struct parser_params*);

 #ifndef RIPPER
@@ -2560,6 +2562,18 @@ block_arg    : tAMPER arg_value
            $$ = $2;
            %*/
            }
+       | tAMPER
+           {
+           /*%%%*/
+           if (!local_id(ANON_BLOCK_ID)) {
+               compile_error(PARSER_ARG "no anonymous block parameter");
+           }
+           $$ = NEW_BLOCK_PASS(new_lvar(ANON_BLOCK_ID, &@1));
+           $$->nd_loc = @$;
+           /*%
+           $$ = Qnil;
+           %*/
+           }
        ;

 opt_block_arg  : ',' block_arg
@@ -4913,6 +4927,15 @@ f_block_arg  : blkarg_mark tIDENTIFIER
            $$ = dispatch1(blockarg, $2);
            %*/
            }
+       | blkarg_mark
+           {
+           /*%%%*/
+           $$ = ANON_BLOCK_ID;
+           arg_var($$);
+           /*%
+           $$ = dispatch1(blockarg, Qnil);
+           %*/
+           }
        ;

 opt_f_block_arg    : ',' f_block_arg
```

#10 - 12/12/2017 04:05 PM - bughit (bug hit)

nobu (Nobuyoshi Nakada) wrote:

> Just to be clear, does it require both of the definition and the use, instead of only the latter?
>
> I mean this is allowed:
>
> ```
> def foo(&)
>   bar(&)
> end
> ```
>
> but these are not:
>
> ```
> def foo()
>   bar(&)
> end
>
> def foo(&block)
>   bar(&)
> end
> ```

def foo(&) is more self documenting so should be legal syntax, but should not be required because in general a method taking a block does not have to be marked.

#11 - 12/14/2017 05:52 AM - mame (Yusuke Endoh)

- Target version set to 2.6

According to ko1, Matz said that the details of the spec is not mature yet, so this ticket is postponed to 2.6.

A patch that allow def foo; bar(&); end:

```
diff --git a/compile.c b/compile.c
index 1b7158979a..79fde2f1a9 100644
--- a/compile.c
+++ b/compile.c
@@ -4341,8 +4341,13 @@ setup_args(rb_iseq_t *iseq, LINK_ANCHOR *const args, const NODE *argn,
     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;
+   if (argn->nd_body != NODE_SPECIAL_ANONYMOUS_BLOCK) {
+       COMPILE(arg_block, "block", argn->nd_body);
+       *flag |= VM_CALL_ARGS_BLOCKARG;
+   }
+   else {
+       *flag |= VM_CALL_ARGS_BLOCKARG | VM_CALL_ARGS_BLOCKARG_THROUGH;
+   }
    argn = argn->nd_head;
     }

diff --git a/insns.def b/insns.def
index 1c20573254..cf5702fea0 100644
--- a/insns.def
+++ b/insns.def
@@ -841,7 +841,7 @@ DEFINE_INSN
 send
 (CALL_INFO ci, CALL_CACHE cc, ISEQ blockiseq)
 (...)
-(VALUE val) // inc += - (int)(ci->orig_argc + ((ci->flag & VM_CALL_ARGS_BLOCKARG) ? 1 : 0));
+(VALUE val) // inc += - (int)(ci->orig_argc + (((ci->flag & (VM_CALL_ARGS_BLOCKARG | VM_CALL_ARGS_BLOCKARG_TH
ROUGH)) == VM_CALL_ARGS_BLOCKARG) ? 1 : 0));
 {
     struct rb_calling_info calling;

@@ -924,7 +924,7 @@ DEFINE_INSN
 invokesuper
 (CALL_INFO ci, CALL_CACHE cc, ISEQ blockiseq)
 (...)
-(VALUE val) // inc += - (int)(ci->orig_argc + ((ci->flag & VM_CALL_ARGS_BLOCKARG) ? 1 : 0));
+(VALUE val) // inc += - (int)(ci->orig_argc + (((ci->flag & (VM_CALL_ARGS_BLOCKARG | VM_CALL_ARGS_BLOCKARG_TH
ROUGH)) == VM_CALL_ARGS_BLOCKARG) ? 1 : 0));
```

```
 {
     struct rb_calling_info calling;
     calling.argc = ci->orig_argc;
diff --git a/iseq.c b/iseq.c
index 186f8622e7..b7b398c47e 100644
--- a/iseq.c
+++ b/iseq.c
@@ -1433,6 +1433,7 @@ rb_insn_operand_intern(const rb_iseq_t *iseq,
         if (ci->flag & VM_CALL_ARGS_SPLAT) rb_ary_push(flags, rb_str_new2("ARGS_SPLAT"));
         if (ci->flag & VM_CALL_ARGS_BLOCKARG) rb_ary_push(flags, rb_str_new2("ARGS_BLOCKARG"));
         if (ci->flag & VM_CALL_ARGS_BLOCKARG_BLOCKPARAM) rb_ary_push(flags, rb_str_new2("ARGS_BLOCKARG_BLOCKPA
RAM"));
+        if (ci->flag & VM_CALL_ARGS_BLOCKARG_THROUGH) rb_ary_push(flags, rb_str_new2("ARGS_BLOCKARG_THROUGH"))
;
         if (ci->flag & VM_CALL_FCALL) rb_ary_push(flags, rb_str_new2("FCALL"));
         if (ci->flag & VM_CALL_VCALL) rb_ary_push(flags, rb_str_new2("VCALL"));
         if (ci->flag & VM_CALL_ARGS_SIMPLE) rb_ary_push(flags, rb_str_new2("ARGS_SIMPLE"));
diff --git a/node.c b/node.c
index 5fa5e1fa50..f28653a3fc 100644
--- a/node.c
+++ b/node.c
@@ -773,7 +773,12 @@ 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 (node->nd_body != NODE_SPECIAL_ANONYMOUS_BLOCK) {
+        F_NODE(nd_body, "block argument");
+    }
+    else {
+        F_MSG(nd_body, "block argument", "NODE_SPECIAL_ANONYMOUS_BLOCK");
+    }
     return;

       case NODE_DEFN:
diff --git a/node.h b/node.h
index 6ff68e1800..88d8493770 100644
--- a/node.h
+++ b/node.h
@@ -458,6 +458,7 @@ typedef struct RNode {

 #define NODE_SPECIAL_REQUIRED_KEYWORD ((NODE *)-1)
 #define NODE_SPECIAL_NO_NAME_REST     ((NODE *)-1)
+#define NODE_SPECIAL_ANONYMOUS_BLOCK  ((NODE *)-1)

 RUBY_SYMBOL_EXPORT_BEGIN

diff --git a/parse.y b/parse.y
index 2eb1a0e1f8..a772d06f6d 100644
--- a/parse.y
+++ b/parse.y
@@ -2565,6 +2565,15 @@ block_arg    : tAMPER arg_value
             $$ = $2;
             %*/
             }
+        | tAMPER
+            {
+            /*%%%*/
+            $$ = NEW_BLOCK_PASS(NODE_SPECIAL_ANONYMOUS_BLOCK);
+            $$->nd_loc = @$;
+            /*%
+            $$ = dispatch0(anonymous_block_arg);
+            %*/
+            }
        ;

 opt_block_arg  : ',' block_arg
@@ -4924,6 +4933,13 @@ f_block_arg  : blkarg_mark tIDENTIFIER
             $$ = dispatch1(blockarg, $2);
             %*/
             }
+        | blkarg_mark
+            {
+            /*%%%*/
+            /*%
+            $$ = dispatch1(blockarg, Qnil);
+            %*/
+            }
        ;

 opt_f_block_arg    : ',' f_block_arg
diff --git a/vm_args.c b/vm_args.c
index 997b0b2f48..8a2eb055cb 100644
--- a/vm_args.c
+++ b/vm_args.c
@@ -839,6 +839,10 @@ vm_caller_setup_arg_block(const rb_execution_context_t *ec, rb_control_frame_t *
        !VM_ENV_FLAGS(VM_CF_LEP(reg_cfp), VM_FRAME_FLAG_MODIFIED_BLOCK_PARAM)) {
        calling->block_handler = VM_CF_BLOCK_HANDLER(reg_cfp);
    }
+   else if (ci->flag & VM_CALL_ARGS_BLOCKARG_THROUGH) {
+       ++reg_cfp->sp;
+       calling->block_handler = GET_BLOCK_HANDLER();
+   }
    else if (NIL_P(block_code)) {
        calling->block_handler = VM_BLOCK_HANDLER_NONE;
    }
diff --git a/vm_core.h b/vm_core.h
index 374fcff1b2..e5f37ce48f 100644
--- a/vm_core.h
+++ b/vm_core.h
@@ -938,6 +938,7 @@ enum vm_call_flag_bits {
     VM_CALL_ARGS_SPLAT_bit,      /* m(*args) */
     VM_CALL_ARGS_BLOCKARG_bit,   /* m(&block) */
     VM_CALL_ARGS_BLOCKARG_BLOCKPARAM_bit,   /* m(&block) and block is a passed block parameter */
+    VM_CALL_ARGS_BLOCKARG_THROUGH_bit,      /* m(&) */
     VM_CALL_FCALL_bit,           /* m(...) */
     VM_CALL_VCALL_bit,           /* m */
     VM_CALL_ARGS_SIMPLE_bit,     /* (ci->flag & (SPLAT|BLOCKARG)) && blockiseq == NULL && ci->kw_arg == NULL *
/
@@ -953,6 +954,7 @@ enum vm_call_flag_bits {
 #define VM_CALL_ARGS_SPLAT       (0x01 << VM_CALL_ARGS_SPLAT_bit)
 #define VM_CALL_ARGS_BLOCKARG    (0x01 << VM_CALL_ARGS_BLOCKARG_bit)
 #define VM_CALL_ARGS_BLOCKARG_BLOCKPARAM (0x01 << VM_CALL_ARGS_BLOCKARG_BLOCKPARAM_bit)
+#define VM_CALL_ARGS_BLOCKARG_THROUGH    (0x01 << VM_CALL_ARGS_BLOCKARG_THROUGH_bit)
 #define VM_CALL_FCALL            (0x01 << VM_CALL_FCALL_bit)
 #define VM_CALL_VCALL            (0x01 << VM_CALL_VCALL_bit)
 #define VM_CALL_ARGS_SIMPLE      (0x01 << VM_CALL_ARGS_SIMPLE_bit)
```

#12 - 12/14/2017 10:09 AM - Eregon (Benoit Daloze)

Just my opinion, but I think the shortcut syntax is going to cause more confusion than it would help.
& as argument without a & as a parameter looks very weird to me (it's like it in other languages, I would strongly advise against such magic).
I argue adding this makes Ruby less readable as a language.

This only saves one character in the definition, and it's not hard to name a block: &b or &block,
that's what the Ruby code out there uses and is clear *because of it*.
Moreover, it's inconsistent with arguments which must be named to be passed (except via zsuper):
foo(*) does not work as a call, and def meth(*) means ignore arguments, which is meaningless for blocks (just omitting &b is enough to ignore a block).

This also seems to conflict with one of the nicer proposals for short block notations like
enum.map(&.to_s.upcase.ljust(3)), a much more general and useful feature than this shortcut,
which only applies for a few methods forwarding the block and saves 2 character per forwarding method.

Also, the original poster started with "since capturing a block into a proc is slow: foo(&block)".
There is no such reason anymore, and I see no clear motivation of why such a special edge-case syntax is worth adding.

From the developer meeting notes, [Feature #11256] anonymous block forwarding:
Matz: it’s a good property to avoid naming variables. Also I personaly like it. However prompting such name-less programming might let people write cryptic codes.
Matz: hmm… Made up my mind. Accepted.

I agree it is cryptic and moreover argue it has little use, is inconsistent with other arguments and likely to cause syntax restrictions for other features.

bughit (bug hit) Could you explain your motivation for this shortcut now that the performance is no longer a concern?
I agree foo{yield} was not nice, but is it worth to have foo(&) foo(1, 2, &) over foo(&b) foo(1, 2, &b) ?

matz (Yukihiro Matsumoto) Could you detail your opinion and reconsider?

#13 - 12/14/2017 04:58 PM - bughit (bug hit)

Eregon (Benoit Daloze) wrote:

> bughit (bug hit) Could you explain your motivation for this shortcut now that the performance is no longer a concern?
> I agree foo{yield} was not nice, but is it worth to have foo(&) foo(1, 2, &) over foo(&b) foo(1, 2, &b) ?

ko1 convinced matz to accept this and he provided his reasons, simplicity of notation, not requiring an otherwise pointless variable. Which makes sense to me.

Also keep in mind that lazy proc allocation is an implementation detail. Conceptually when you declare a block params it still looks like you're doing unnecessary work of instantiating a proc object which you have no intention of using.

Every method has an invisible, nameless, optional block, and a naked & seems like an intuitive, logical way to forward it.

#14 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)