Ruby master - Feature #17151

Support multiple builtin ruby code for implimatation in Ruby & C

09/04/2020 04:45 PM - S_H_ (Shun Hiraoka)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Summrary

Support these multiple builtin code in implimatation in Ruby & C.

```
# excerpt of trueclass.rb
class TrueClass
  def to_s
    Primitive.attr! 'inline'
    Primitive.cexpr! 'rb_cTrueClass_to_s'
  end
end

# excerpt of kernel.rb
module Kernel
  def class
    Primitive.attr! 'inline'
    Primitive.cexpr! 'rb_obj_class(self)'
  end
end

// excerpt of object.c
#include "kernel.rbinc"
#include "trueclass.rbinc"
```

Why

I want to running these code, for more faster to TrueClass and other classes.

ref: Feature #17127 Some TrueClass methods are faster if implemented in Ruby
ref: Feature #17054 Some NilClass methods are faster if implemented in Ruby

Background

Currently, these multiple builtin code can not build(for redefination error).

Because, tool/mk_builtin_loader.rb defined same function.

example

```
//  Kernel#class method's function defination in kernel.rbinc
static void
mjit_compile_invokebuiltin_for__bi0(FILE *f, long index, unsigned stack_size, bool inlinable_p)
{
    fprintf(f, "    VALUE self = GET_SELF();\n");
    fprintf(f, "    typedef VALUE (*func)(rb_execution_context_t *, VALUE);\n");
    if (inlinable_p) {
        fprintf(f, "%s", "    {\n");
        fprintf(f, "%s", "#line 20 \"../ruby/kernel.rb\"\n");
        fprintf(f, "%s", "    return rb_obj_class(self);\n");
        fprintf(f, "%s", "#line 44 \"../ruby/kernel.rbinc\"\n");
        fprintf(f, "%s", "    }\n");
        fprintf(f, "%s", "    \n");
```

```
        return;
    }
    fprintf(f, "    func f = (func)%"PRIdPTR"; /* == builtin_inline_class_20 */\n", (intptr_t)
builtin_inline_class_20);
    fprintf(f, "    val = f(ec, self);\n");
}

// TrueClass#to_s method's function defination in trueclass.rbinc
static void
mjit_compile_invokebuiltin_for__bi0(FILE *f, long index, unsigned stack_size, bool inlinable_p)
{
    fprintf(f, "    VALUE self = GET_SELF();\n");
    fprintf(f, "    typedef VALUE (*func)(rb_execution_context_t *, VALUE);\n");
    if (inlinable_p) {
        fprintf(f, "%s", "    {\n");
        fprintf(f, "%s", "#line 17 \"../ruby/trueclass.rb\"\n");
        fprintf(f, "%s", "    return rb_cTrueClass_to_s;\n");
        fprintf(f, "%s", "#line 30 \"../ruby/trueclass.rbinc\"\n");
        fprintf(f, "%s", "    }\n");
        fprintf(f, "%s", "    \n");
        return;
    }
    fprintf(f, "    func f = (func)%"PRIdPTR"; /* == builtin_inline_class_17 */\n", (intptr_t)
builtin_inline_class_17);
    fprintf(f, "    val = f(ec, self);\n");
}
```

So, this code cause redefination error.

```
In file included from ../ruby/object.c:4617:
../ruby/trueclass.rbinc:27:1: エラー : ‘mjit_compile_invokebuiltin_for__bi0’ が再定義されました
   27 | mjit_compile_invokebuiltin_for__bi0(FILE *
f, long index, unsigned stack_size, bool inlinable_p)
      | ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
In file included from ../ruby/object.c:4616:
../ruby/kernel.rbinc:41:1: 備考 : 前の ‘mjit_compile_invokebuiltin_for__bi0’ の宣言はここです
   41 | mjit_compile_invokebuiltin_for__bi0(FILE *
f, long index, unsigned stack_size, bool inlinable_p)
      | ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
In file included from ../ruby/object.c:4617:
../ruby/trueclass.rbinc:45:1: エラー : ‘mjit_compile_invokebuiltin_for__bi1’ が再定義されました
   45 | mjit_compile_invokebuiltin_for__bi1(FILE *
f, long index, unsigned stack_size, bool inlinable_p)
      | ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
In file included from ../ruby/object.c:4616:
../ruby/kernel.rbinc:75:1: 備考 : 前の ‘mjit_compile_invokebuiltin_for__bi1’ の宣言はここです
   75 | mjit_compile_invokebuiltin_for__bi1(FILE *
f, long index, unsigned stack_size, bool inlinable_p)
      | ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
cc1: 警告 : 認識できないコマンドラインオプション ‘-Wno-self-assign’ です
cc1: 警告 : 認識できないコマンドラインオプション ‘-Wno-parentheses-equality’ です
cc1: 警告 : 認識できないコマンドラインオプション ‘-Wno-constant-logical-operand’ です
```

Propasal

Add the class(or module) name to the function definition.

exmaple:

```
//  Kernel#class method's function defination in kernel.rbinc
static void
mjit_compile_invokebuiltin_for_kernel__bi0(FILE *f, long index, unsigned stack_size, bool
inlinable_p)
{
    fprintf(f, "    VALUE self = GET_SELF();\n");
    fprintf(f, "    typedef VALUE (*func)(rb_execution_context_t *, VALUE);\n");
```

```
    if (inlinable_p) {
        fprintf(f, "%s", "    {\n");
        fprintf(f, "%s", "#line 20 \"../ruby/kernel.rb\"\n");
        fprintf(f, "%s", "    return rb_obj_class(self);\n");
        fprintf(f, "%s", "#line 44 \"../ruby/kernel.rbinc\"\n");
        fprintf(f, "%s", "    }\n");
        fprintf(f, "%s", "    \n");
        return;
    }
    fprintf(f, "    func f = (func)%"PRIdPTR"; /* == builtin_inline_class_20 */\n", (intptr_t)
builtin_inline_class_20);
    fprintf(f, "    val = f(ec, self);\n");
}

// TrueClass#to_s method's function defination in trueclass.rbinc
static void
mjit_compile_invokebuiltin_for_trueclass__bi0(FILE *f, long index, unsigned stack_size, bool
inlinable_p)
{
    fprintf(f, "    VALUE self = GET_SELF();\n");
    fprintf(f, "    typedef VALUE (*func)(rb_execution_context_t *, VALUE);\n");
    if (inlinable_p) {
        fprintf(f, "%s", "    {\n");
        fprintf(f, "%s", "#line 17 \"../ruby/trueclass.rb\"\n");
        fprintf(f, "%s", "    return rb_cTrueClass_to_s;\n");
        fprintf(f, "%s", "#line 30 \"../ruby/trueclass.rbinc\"\n");
        fprintf(f, "%s", "    }\n");
        fprintf(f, "%s", "    \n");
        return;
    }
    fprintf(f, "    func f = (func)%"PRIdPTR"; /* == builtin_inline_class_17 */\n", (intptr_t)
builtin_inline_class_17);
    fprintf(f, "    val = f(ec, self);\n");
}
```

Pull Request

https://github.com/ruby/ruby/pull/3517

History

#1 - 09/04/2020 04:59 PM - S_H_ (Shun Hiraoka)

- Description updated

#2 - 09/05/2020 01:21 AM - nobu (Nobuyoshi Nakada)

Why do they need to be separate files?

#3 - 09/05/2020 01:41 AM - S_H_ (Shun Hiraoka)

Beacause, if file can be separeted when number of cases implemented by builtin code in future, readability for code will be good.(I think so)

Currently, amount of code does not increase to much, but I thought it would be good to consider dividing it may increase in future.