Ruby master - Feature #5898

raise and Exception#initialize

01/16/2012 10:03 AM - trans (Thomas Sawyer)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:	2.0.0	

Description

Calling #raise with a message parameter passes the argument on the Exception class' initialize method. But it does not support any additional arguments if the initialize method has been defined otherwise. Nor is the last optional argument, caller, passed to the initializer. All of which makes for a rather confusing mishmash of an interface between #raise and Exception#initialize.

Ideally I would think whatever arguments are passed to #raise would likewise be passed on to Exception#initialize, e.g.

```
class MyError < Exception
def initialize(a,b)
super("#{a} and #{b}")
end
end

raise MyError, 'foo', 'bar'
```

Alas, because caller can be passed to #raise this causes an error, as it thinks bar ought to be the caller array. So unless others see a way around it that I do not, this idealized scenario simply is not possible.

So I propose a second best approach. Notice that if caller is passed to #raise it is not being passed on to the #initialize method unlike the message argument. Instead #set_backtrace is being used to set the caller. I propose that the message argument be handled in the same way, and a new method #set_message(msg) be added to the Exception class to handle it.

This would then allow the initializer of subclasses to be freed up to be defined in other ways, should a specialized exception be able to make good use of a variant interface Which, btw, is the exact circumstance I presently find myself in for one of my projects. Consequently I had no choice by to define the #initialize method to take an initial blank argument that will almost always be set to +nil+.

History

#1 - 01/16/2012 11:23 AM - nobu (Nobuyoshi Nakada)

Hi,

(12/01/16 10:03), Thomas Sawyer wrote:

> ```
> class MyError < Exception
> def initialize(a,b)
> super("#{a} and #{b}")
> end
> end
>
> raise MyError, 'foo', 'bar'
> ```

Try:

```
class MyError < RuntimeError
def initialize(args)
a, b = *args
super("#{a} and #{b}")
end
end

raise MyError, ['foo', 'bar']
```

> So I propose a second best approach. Notice that if caller is
> passed to #raise it is not being passed on to the #initialize
> method unlike the message argument. Instead #set_backtrace is

> being used to set the caller. I propose that the message argument
> be handled in the same way, and a new method #set_message(msg) be
> added to the Exception class to handle it.

How do you tell if the second argument is a part of messages, or a caller?

--
Nobu Nakada

#2 - 01/16/2012 11:23 AM - nobu (Nobuyoshi Nakada)

Hi,

(12/01/16 11:03), Nobuyoshi Nakada wrote:

> How do you tell if the second argument is a part of messages, or a
> caller?

One of simplest ways I can think of would be:

raise MyError.new('foo', 'bar')

Wow, this works with your definition, doesn't it? ;)

--
Nobu Nakada

#3 - 01/16/2012 11:45 AM - trans (Thomas Sawyer)

> One of simplest ways I can think of would be:
>
> raise MyError.new('foo', 'bar')
>
> Wow, this works with your definition, doesn't it? ;)

Of course, *I* could do that, and that might be fine if it were only me and my program that would ever raise that error. But that's not the case. It may well be used by others. So to break the #raise "contract" would be very bad form. In other words, I am not going to try to explain to my end-developers that certain exceptions are "special" and can't be raised in standard fashion, but always must be done so with #new. And that's really the jist of my point. We shouldn't need to sacrifice one interface for the other.

#4 - 01/16/2012 11:56 AM - trans (Thomas Sawyer)

Actually, I just did something rather interesting that better illustrates the discrepancy:

trans@logisys:assertions$ irb

```
class MyError < RuntimeError
def initialize(*args)
p args
end
end
=> nil
raise MyError.new, "custom message"
[]
MyError: custom message
from (irb):6
from /home/trans/.rbenv/versions/1.9.3-rc1/bin/irb:12:in `'

raise MyError, "custom message"
["custom message"]
MyError: MyError
from (irb):7
from /home/trans/.rbenv/versions/1.9.3-rc1/bin/irb:12:in `'
```

The first case sets the message without needing to pass the message to the MyError#initialize, but the second does not.

#5 - 01/16/2012 12:23 PM - nobu (Nobuyoshi Nakada)

Hi,

(12/01/16 11:56), Thomas Sawyer wrote:

```
class MyError < RuntimeError

  def initialize(*args)
  p args
  super
  end
  end
```

--
Nobu Nakada

#6 - 01/17/2012 01:17 AM - trans (Thomas Sawyer)

How does adding #super help?

Maybe I've over explained (as I am wont to do) and I should just put it this way: It would be good if one could customize the #initialize method of an Exception subclass without causing #raise to break when used in the typical raise ErrorClass, "message" fashion.

#7 - 03/31/2012 10:38 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

#8 - 03/31/2012 11:05 AM - trans (Thomas Sawyer)

You can reject it, but this will eventually come up again b/c current behaviour is confusing, inconsistent and either overly limiting or not limiting enough depending on which side you come down on.

#9 - 03/31/2012 06:23 PM - nobu (Nobuyoshi Nakada)

If you can show the answer to the questions in comment #1([ruby-core:42143]) and it is reasonable, this may advance.