Subscribe

Keith R. Bennett's Technical Blog

Search

About

# class_eval, instance_eval, eval

By keithrbennett on January 28th, 2012

A couple of days ago I attended an interesting discussion of metaprogramming by Arild Shirazi at a meeting of the Northern Virginia Ruby User Group. Arild showed how he used metaprogramming (*class_eval* in particular) to generate functions whose names would only be known at runtime. His talk was very effective at reminding me that I don't know as much about metaprogramming as I thought!

(Feel free to offer suggestions and corrections, and I'll try to update the article accordingly.)

Dave Thomas, in his excellent Advanced Ruby training, emphasizes the value of knowing just who *self* is at any point in the code. (For a good time, bounce around an rspec source file and try to guess what *self* is in various places…).

*class_eval* provides an alternate way to define characteristics of a class. It should be used only when absolutely necessary. The only legitimate use I can think of is when the necessary code cannot be known until runtime.

Knowing very little about *class_eval*, I assumed that it changed self to be the class of the current value of self. I was wrong. class_eval doesn't change self at all; in fact, in this respect it functions identically to eval:

```
> class ClassEvalExample
>   class_eval "def foo; puts 'foo'; end"
> end
> ClassEvalExample.new.foo
foo
```

*eval* appears to do the exact same thing:

```
> class EvalExample
>   eval "def foo; puts 'foo'; end"
> end
> EvalExample.new.foo
foo
```

There is a difference, though, when you call them outside the class definition. For a class C, you can call C.class_eval, but not C.eval:

```
> class C1; end
> C1.class_eval "def foo; puts 'foo'; end"
> C1.new.foo
foo

> class C2; end
> C2.eval "def foo; puts 'foo'; end"
NoMethodError: private method `eval' called for (
    from (irb):2
    from :0
```

If class_eval could be used to define an instance method on a class in a class definition *outside* a function, what would happen if it were used *inside* a function, where self is no longer the class, but the instance of the class? Would it define a method on the singleton class (a.k.a. *eigenclass*)? Let's try it:

```
:001 > class D
 :002?>     def initialize
 :003?>         puts "In initialize"
 :004?>         class_eval "def foo; puts 'foo';
 :005?>       end
 :006?>   end
 => nil
 :007 >
 :008 >   D.new.foo
In initialize
NoMethodError: undefined method `class_eval' for
    from (irb):4:in `initialize'
    from (irb):8:in `new'
    from (irb):8
    from :0
```

No, this didn't work...but wait a minute, isn't class_eval a Kernel method? Let's find out:

```
> Kernel.methods.include? 'class_eval'
=> true
```

Alas, I was asking the wrong question. I should have asked if Kernel had an *instance* method named *class_eval*:

```
> Kernel.instance_methods.include? 'class_eval'
=> false
```

It doesn't, but *Class* does:

```
> Class.instance_methods.include? 'class_eval'
=> true
```

...which is why the Kernel.methods.include? above worked.

Although *class_eval* didn't work, *instance_eval* will work:

```
> class F
>   def initialize
>     instance_eval 'def foo; puts "object id is #
>   end
> end
> F.new.foo
object id is 2149391220
> F.new.foo
object id is 2149362060
```

To illustrate that foo has not been created as a class or member function on class F, but only on object f:

```
> F.methods(false).include? 'foo'
 => false
> F.instance_methods(false).include? 'foo'
 => false
> f = F.new
> f.methods(false).include? 'foo'
 => true
```

Could *eval* be substituted for *instance_eval* in the same way as it was for *class_eval*? Let's find out…

```
>   class F2
>     def initialize
>         eval 'def foo; puts "object id is #{obje
>     end
> end
> F2.new.foo
object id is 2149180440
```

Apparently, yes. However, similarly to *class_eval*, *instance_eval* can be called outside of a class definition, but *eval* cannot:

```
> class C; end
> c = C.new
> c.instance_eval 'def foo; puts "object id is #{
> c.foo
object id is 2149446940

> class D; end
> d = D.new
> d.eval 'def foo; puts "object id is #{object_ic
NoMethodError: private method `eval' called for #
    from (irb):7
    from :0
```

Hmmm, I wonder, if we can define a *function* using the eval methods, can we also declare an instance *variable*?:

```
# First, class_eval:
> class E
>   class_eval "@@foo = 123"
>   def initialize; puts "@@foo = #{@@foo}"; end
>   end
> E.new
@@foo = 123

# Next, instance_eval:
> o = Object.new
> o.instance_eval '@var = 456'
> o.instance_eval 'def foo; puts "@var = #{@var}'
> o.foo
```

```
@var = 456
```

What's interesting is that we created instance variable *var* in instance *o*, but its class Object knows nothing about this new variable. In the data storage world, this would be analogous to using a document store such as MongoDB and adding a variable to a single document, unlike in an RDBMS where you would have to add it to the table definition and include it in all rows of the table.

Techniques such as these are cool and powerful, but are not without cost. If your code accesses a function or variable that is not defined in a standard class definition, the reader may have a hard time tracking down the creation and meaning of that function or variable. We should be kind to our fellow developers and use these techniques only when absolutely necessary.

Categorized under: Uncategorized.

Tagged with: no tags.

## Leave a Response

You must be logged in to post a comment.

← Design by Contract, Ruby Style

Mailing Files Programmatically with GMail →

### Meta

- Register
- Log in
- Entries RSS
- Comments RSS
- WordPress.org

Search

### Archives

- November 2015
- November 2013
- August 2013
- January 2013
- December 2012
- November 2012
- September 2012
- July 2012
- January 2012
- June 2011
- May 2011
- March 2010
- July 2009
- March 2009
- February 2009
- November 2008

## Recent Activity

**Posts** **Comments**

- The Case for Nested Methods in Ruby
- Ruby’s inject/reduce and each_with_object
-  in Your System Prompt
- Using Oracle in JRuby with Rails and Sequel
- Copying (RVM) Data Between Hosts Using ssh, scp, and netcat
- Building A Great Ruby Development Environment and Desktop with Linux Mint 13 “Maya” Mate
- Intro to Functional Programming in Ruby
- WordPress Administration with Ruby
- Stealth Conditionals in Ruby
- Hello, Nailgun; Goodbye, JVM Startup Delays

## Archives

**Categories** **Tags** **Dates**

Authors

- Uncategorized

Powered by WordPress and the PressPlay Theme

Copyright © 2016 Keith R. Bennett's Technical Blog