Home

Pages Classes Methods

Search

Show/hide navigation

Parent

Object

Methods

#eval
#local_variable_defined?
#local_variable_get
#local_variable_set
#local_variables
#receiver
#source_location

class Binding

Objects of class `Binding` encapsulate the execution context at some particular place in the code and retain this context for future use. The variables, methods, value of `self`, and possibly an iterator block that can be accessed in this context are all retained. `Binding` objects can be created using `Kernel#binding`, and are made available to the callback of `Kernel#set_trace_func` and instances of `TracePoint`.

These binding objects can be passed as the second argument of the `Kernel#eval` method, establishing an environment for the evaluation.

```
class Demo
  def initialize(n)
    @secret = n
  end
  def get_binding
    binding
  end
end

k1 = Demo.new(99)
b1 = k1.get_binding
k2 = Demo.new(-3)
b2 = k2.get_binding

eval("@secret", b1)   #=> 99
eval("@secret", b2)   #=> -3
eval("@secret")       #=> nil
```

`Binding` objects have no class-specific methods.

Public Instance Methods

eval(string [, filename [,lineno]]) → obj

Evaluates the Ruby expression(s) in *string*, in the *binding*'s context. If the optional *filename* and *lineno* parameters are present, they will be used when reporting syntax errors.

```
def get_binding(param)
  binding
end
b = get_binding("hello")
b.eval("param")   #=> "hello"
```

local_variable_defined?(symbol) → obj

Returns `true` if a local variable `symbol` exists.

```
def foo
  a = 1
  binding.local_variable_defined?(:a) #=> true
  binding.local_variable_defined?(:b) #=> false
end
```

This method is the short version of the following code:

```
binding.eval("defined?(#{symbol}) == 'local-variable'")
```

local_variable_get(symbol) → obj

Returns the value of the local variable `symbol`.

```
def foo
  a = 1
  binding.local_variable_get(:a) #=> 1
  binding.local_variable_get(:b) #=> NameError
end
```

This method is the short version of the following code:

```
binding.eval("#{symbol}")
```

local_variable_set(symbol, obj) → obj

Set local variable named `symbol` as `obj`.

```
def foo
  a = 1
  bind = binding
  bind.local_variable_set(:a, 2) # set existing local variable `a'
  bind.local_variable_set(:b, 3) # create new local variable `b'
                                 # `b' exists only in binding

  p bind.local_variable_get(:a)  #=> 2
  p bind.local_variable_get(:b)  #=> 3
  p a                            #=> 2
  p b                            #=> NameError
end
```

This method behaves similarly to the following code:

```
binding.eval("#{symbol} = #{obj}")
```

if `obj` can be dumped in Ruby code.

local_variables → Array

Returns the names of the binding's local variables as symbols.

```
def foo
  a = 1
  2.times do |n|
    binding.local_variables #=> [:a, :n]
  end
end
```

This method is the short version of the following code:

```
binding.eval("local_variables")
```

receiver → object

Returns the bound receiver of the binding object.

source_location → [String, Integer]

Returns the Ruby source filename and line number of the binding object.

Validate
Generated by RDoc 6.4.0.
Based on Darkfish by Michael Granger.
Ruby-doc.org is provided by James Britt and Neurogami.
Maximum R+D.