Web System Development with Ruby on Rails

Day 9(22/Nov/2011)

Grammar of Ruby Language

Today's Theme

- Learn Grammatical Strucure of Ruby Language
- Class Definition, Method Definition
- Describing Array and Hash Array
- Symbol Description
- Usage of Iterators
- Regular Expression
- Rails API

Class Definition

- Under Rails Environment,
  - In the “models” folder, there is memo.rb
  - In the file, Relation is described
  - Inherit the Super Class: “ActiveRecord”
    - Major important Functions are defined in Super Class.
    - In Rails Environment, Super Classes are predefined.

```
class Question < ActiveRecord::Base
  belongs_to :quizset
  belongs_to :category
  validates_presence_of :code,:sentence,
         :answer, :choice1, :choice2,
         :choice3, :choice4
end
```

Super Classes in Object Oriented Environment

- "Basic Tools(Basic functions)" are defined in "super class Environment" those of which users often uses.

By inheriting those super class, descendants can use those functions.

- Ex, UIWindow Class in iOS Application
  - provides "Screen Function" in iPhone and iPad
- Ex, ActiveRecord in ruby
  - provides major functions for Database Access
- Ex, Application Controller in ruby
  - Defines methods to connect models and database

Definition of Method

```
def name(args)
end
```

- If no arguments are given, only the method name appears
- Examples in the right, defines two methods

```ruby
# GET /memos.json
def index
  @memos = Memo.all

  respond_to do |format|
    format.html # index.html.erb
    format.json { render json: @memos }
  end
end

# GET /memos/1
# GET /memos/1.json
def show
  @memo = Memo.find(params[:id])

  respond_to do |format|
    format.html # show.html.erb
    format.json { render json: @memo }
  end
end
```

String literals

- Both Double quotation” and 'single quotation’ work.
- Both quotations can contain the other type quotation among them, so they are used to contain the other quotation.
- Double Quotation can be used to realize the following function
  - Embed and show variables' value
  - Use the control characters

Embed variable reference in String

- Invoke irb on GNOME shell,
- Assign string literal to a variable,
- In another string enclosed by double quotations, #{name} will be developed to show the value of the variable by its 'name.'

```
[root@cisnote ~]# irb
irb(main):001:0> name='Ikuo'
=> "Ikuo"
irb(main):002:0> puts "Welcome, #{name}\n"
Welcome, Ikuo
=> nil
irb(main):003:0>
```

Specify filename to run ruby

- If a program has become too long to run in a command line, write the whole program into a file to run it.
- We often write a shebang, and encoding specification at the top lines.

- Type

  ruby FileName

- In the command prompt
- At the top two line, write

```
#!/usr/local/bin/ruby
# -*- coding: utf-8 -*-
```

so that UTF-8 characters are shown in the readable appearances.

Formatted String printouts

- The same with C language, printf can be used.

```
#! ruby -Ks
# -*- coding: Windows-31J -*-
e = 2.7182818284
f = 123456789.12
print "e= #{ e }\n"
print “2 by e makes #{ 2 * e }.\n"
prime100 = 541
print “100th prime number is #{prime100}.\n"

printf( "e = %5.3f\n", e )
printf( "f = %4.3f\n", f )
```

Elements of Array

- Array elements (objects) may belong to different Classes each other.
- Array Class can contain different Class instances, such as Integer, String, Array, and such. They are different but all belongs to the same Class, ‘Object’ at last.

```
irb(main):012:0> list = ['coffee', 3, 'tea', 4 ]
=> ["coffee", 3, "tea", 4]
irb(main):013:0> list.each do |l| puts l end
coffee
3
tea
4
=> ["coffee", 3, "tea", 4]
irb(main):014:0>
```

Sample of Array (1)

```
irb(main):005:0> animals = ['dog', 'cat', 'elephant' ]
=> ["dog", "cat", "elephant"]
irb(main):006:0> puts animals[0]
dog
=> nil
irb(main):007:0> puts animals[3]

=> nil
irb(main):008:0> animals[3] = 'whale'
=> "whale"
irb(main):009:0> puts animals[3]
whale
=> nil
irb(main):010:0> animals << 'mouse'
=> ["dog", "cat", "elephant", "whale", "mouse"]
irb(main):011:0> puts animals[4]
mouse
=> nil
irb(main):012:0>
```

Hash (Association Array)

- When we define Hash, "{ }" are used, but for referencing, it can be used as the same with ordinary array.
- Strings can be used for index.

```
population = {
  'France' => 60424213,
  'Germany' => 82424609,
  'Italy' => 58057477
}
puts "Italy: #{population['Italy']}"
population['Japan'] = 127767944
puts "Japan: #{population['Japan']}"
```

Conditional Branch

```
if condition1 then
    programs when condition1 is satisfied
elsif condition2 then
    programs when condition2 is satisfied
else
    neither condition 1 or 2 is satisfied
end
```

- “:” is used instead of “then”, when codes are written in 1 line.

Comparison and Logic operators

- Comparison operators used in a conditional branch:

==, ===, !=, >, >=, <, <=, <=>, =~, !~ etc.

- left and right is not symmetric in "===" operator.
- =~ is for "regular form"

- Logic operators used in a conditional branch:

&&, ||, !, and, or, not, etc.

- Comparing && and ||, "&&" has higher priority while "and" and "or" have the same priority.

Example of regular form

```
greeting = 'Good Morning.'
if /[Mm]orning/ =~ greeting then
    reply = 'Good Morning.'
else
    reply= 'Good Day.'
end
puts reply
```

- Execute the above example
- In the above example, it is judged if "Morning" or "morning" is included in the string.

Regular Expression Patterns

Pattern	Description
^	Matches beginning of line.
$	Matches end of line.
.(dot)	Matches any single character except newline.(Wildcard) Using m option allows it to match newline as well.
re*	Matches 0 or more occurrences of preceding expression.
re+	Matches 1 or more occurrence of preceding expression.
re?	Matches 0 or 1 occurrence of preceding expression.
[...]	Matches any single character in brackets.
[^...]	Matches any single character not in brackets.
a|b	Matches either a or b.
[0-9]	Matches any digit; same as /[0123456890]/
[a-z]	Match any lowercase ASCII letter

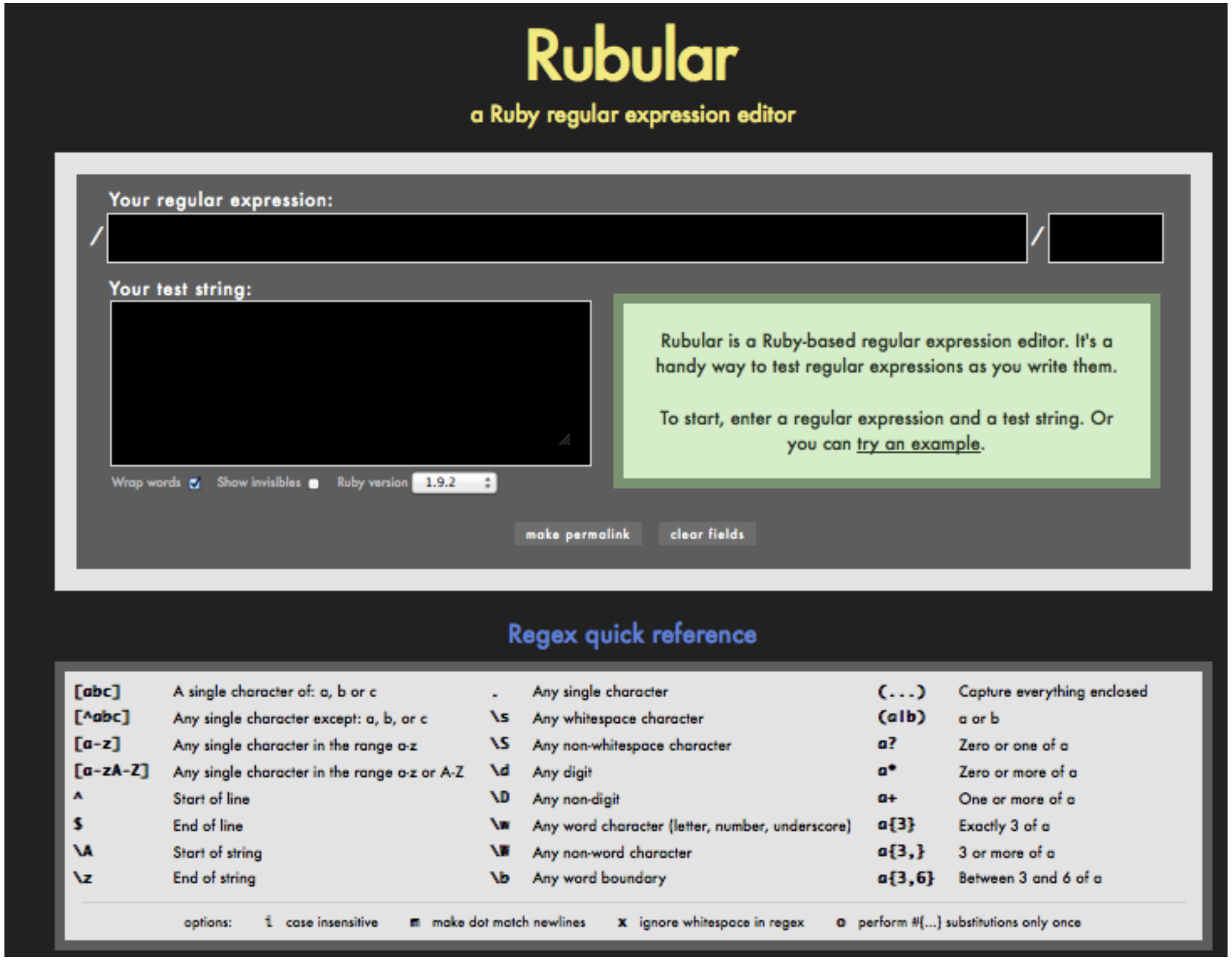
Regular Expression Patterns(2)

Pattern	Description
re{ n}	Matches exactly n number of occurrences of preceding expression.
re{ n,}	Matches n or more occurrences of preceding expression.
re{ n, m}	Matches at least n and at most m occurrences of preceding expression.
(re)	Groups regular expressions and remembers matched text.
\d	Matches digits. Equivalent to [0-9].
\s	Matches whitespace. Equivalent to [\t\n\r\f]./
Example:	/(foo){1}/ # => "foo"
	/(foo){2,}/ # => "foofoofoo"
	/(foo){1,2}/ # => "foofoo"
	/\(\d{3}\)\s(\d{3})-(\d{4}/ # => (123) 456-7890

http://www.tutorialspoint.com/ruby/ruby_regular_expressions.htm

Ruby Regular Expression Tester

Visit the site: http://rubular.com/



Repetition in loop

- Both "for", "while", and "until" can be used, but iterators are often used.

```
Array1 = [ 'Nakano' , 'Mitaka' , 'Tachikawa'  ]
i = 0
while array1[i]
   puts array1[i]
   i += 1
end
```

Repetition with iterator

- times, upto, each, each_with_index, and such methods are used as iterators.

```
10.times do { |i| print i, ', ' }

Array1 = [ 'Nakano' , 'Mitaka' , 'Tachikawa' ]
array1.each do |item|
  print item + ','
end
```

Syntax of Ruby

Visit the following site:

http://web.njit.edu/all_topics/Prog_Lang_Docs/html/ruby/yacc.html

YACC is a tool to design the compiler; stands for “Yet Another Compiler compiler,” and is used together with LEX.

LEX is a lexical analyzer, to design the BNF of the language.

BNF stands for Backus-Naur Form. It is used to discribe the grammar of languages.

Ruby Language Reference Manual

Visit the following site:

http://web.njit.edu/all_topics/Prog_Lang_Docs/html/ruby/index.html

Now I refer to the above site.

Reserved Words

BEGIN	class	ensure	nil	self	when
END	def	false	not	super	while
alias	defined	for	or	then	yield
and	do	if	redo	true	
begin	else	in	rescue	undef	
break	elsif	module	retry	unless	
case	end	next	return	until	

Modules and recursive call

We can define a module to write some methods.

```
module Foo
  def test
      :
  end
    :
end
```

Examples:

```
  def fact(n)
      if n== 1 then
          1
      else
          n * fact(n-1)
      end
  end
```

Method names end with '?'

Ruby has some method names end with '?'.

defined?, empty?, exited?, any?, all?, include?, coredump?, etc.

Also some method names end with '!'.

reject!, next!, delete!, etc.

```
"abc".empty? # ==> false
"".empty?    # ==> true
```

Yield and Block as an argument

Ruby can hand a block as an argument.

Yield call the block, and the procedure does not appear in the argument list.

```
def bar( x )
   p block_given?
   return x + 2
end

p bar( 3 )
p bar( 5 ) { p “zot” }
```

```
def bar( x, &proc )
   proc.call if  block_given?
   return x + 2
end

p bar( 3 )
p bar( 5 ) { p “zot” }
```

```
def bar( x )
   yield if  block_given?
   return x + 2
end

p bar( 3 )
p bar( 5 ) { p “zot” }
```

Scope of methods

Just like c++, C#, and Java, there are three scope types in Ruby;

private, protected, and public

Public methods have no limit in accessing.

Protected methods can be called only from the same class methods or its subclass.

Private methods can be called from the same class methods or its subclass, too!

When we define a class method in a subclass, it can override the method of the same name in a parent class.

Ruby's method or Rails' method?

When we read Ruby on Rails generated source code, we should be careful if the methods are defined in Ruby language or Rails environment.

We can use .blank? method in rails environment, but it is not defined in Ruby.

Also, we need to learn the Superclass methods in the generated source codes.

Rails Document (Practice)

To read the Rails generated source code, first, we have to check the superclass of the generated classes.

Visit the following site:

http://api.rubyonrails.org/

Here are documents for super classes.

Check the following words in this API reference;

attr_accessible, has_one, redirect_to, and such.

Further more…

- Two lecture days are too short to learn one language.
- What we have learnt during those two lecture days were the tutorial background knowledge to read the automatically generated source program of Ruby on Rails.
- When necessary, grammatical explanations are added to the lecture slides.

Practice

1) Write the regular expression of mail address.
2) Find add_column (Migration) methods in the API reference, and find what other migration methods are available.

No Report is requested for this practice.