8章 模块

作者 bluetea
网站:https://github.com/bluetea

8.5 模块是什么？
如果说类表现的是实物的实体（数据）及其行为（处理），那么模块表现的就只是实物的行为部分，模块与类有以下2点不同：
1.模块不能拥有实例
2.模块不能被继承

模块提供命名空间
例如 在FileTest的模块中存在与获取文件信息相关的方法，我们可以使用 “模块名.方法名” 的形式来调用模块中定义的方法，这样的的方法成为 模块函数

```
1  p FileTest.exist?("/usr/bin/ruby")
2  p FileTest.size("/usr/bin/ruby")
```

如果没有定义 模块内的 方法，变量等同名的名称，也可以省略模块名，通过include可以把模块内的方法名，变量合并到当前的命名空间，不用每次都带着模块名调用了

```
[1] pry(main)> p Math::PI
3.141592653589793
=> 3.141592653589793
[2] pry(main)> p Math.sqrt(2)
1.4142135623730951
=> 1.4142135623730951
[3] pry(main)> include Math
=> Object
[4] pry(main)> p PI
3.141592653589793
=> 3.141592653589793
[5] pry(main)> p sqrt(2)
1.4142135623730951
=> 1.4142135623730951
```

8.6.2 Mix_in扩展功能
就是在定义class的时候 通过include 模块，这样模块内的方法，常量都能使用了
Mix_in 有点和类的集成类似，但是灵活的满足一下的问题和需求
1.虽然下面2个类有相似功能，但是不希望把它们作为相同的类（class）来考虑问题
2.Ruby不支持父类的多重继承，因此无法对阴茎继承的类添加共通的功能的时候

```
1  module MyModule
2    def print_hello
3      puts "hello module"
4    end
5  end
6
7  class MyClass1
8    include MyModule
9  end
10
11 class MyClass2
12   include MyModule
13 end
14 MyClass1.new.print_hello
```

注意声明为模块函数，需要显式声明，这个模块函数在被class包含后，会变为class的private实例方法，就无法显示调用了

eigenclass.rb

```ruby
module Mod
  def one
  "this is one"
  end
  module_function :one #被包含之后，会变为class的私有实例方法
end
class MyClass
  include Mod
  def cls
    one
  end
end
a = MyClass.new
p Mod.one
p a.cls
p a.one
```

Run eigenclass

```
C:\Ruby200-x64\bin\ruby.exe -e $stdout.sync=true;$stderr.sync=true;load($0=ARGV.shift) C:/Users/wangmjc/Documents/ruby/eigenclass.
"this is one"
"this is one"
C:/Users/wangmjc/Documents/ruby/eigenclass.rb:16:in `<top (required)>': private method `one' called for #<MyClass:0x00000002db9f78
    from -e:1:in `load'
    from -e:1:in `<main>'
```

```ruby
module MyModule
  Version = 1.0
  def hello
    puts "hello module"
  end
  module_function :hello  #必须显式的将方法声明为 module_function才行

end

class MyClass1
  include MyModule
end

class MyClass2
  include MyModule
end

MyModule.hello
p MyModule::Version

include MyModule

p Version
hello
```

test.rb

```ruby
module MyModule
  def foo
    p "foo module"
    p self
  end

end

class C
  include MyModule
end

a = C.new
a.foo
p C.include?(MyModule) #可以测试一个类是否包含某个模块
```

当类C的实例在调用方法时，Ruby会按照 C, MyModule, Object(C的superclass)的顺序来查找 C -> MyModule -> Object ，MyModule虽然不是C的父类，但是作用类似，所以算C的虚拟父类

用 C.ancestors 的类方法可以查看C类的继承关系

```
[20] pry(main)> C.ancestors
=> [C, MyModule, Object, PP::ObjectMixin, Kernel, BasicObject]
[21] pry(main)> C.superclass
=> Object
```

```
2.0.0-p598 :019 >   C.ancestors
 => [C, MyModule, Object, Kernel, BasicObject]
2.0.0-p598 :020 > C.superclass
 => Object
```

注意:

1.PP::ObjectMixin是 pry软件家出来的，下面的是真正的Irb输出

2.Kernel是Ruby内部的一个核心模块，Ruby程序运行时所需的共通函数都封装在此模块中，例如p方法，raise方法。

当include了Module后，实例的方法查找规则

1.同继承关系一样，原类中已经定义了同名的方法时，优先使用类内的方法

```ruby
module M
  def meth
    "M - meth"
  end
end

class C
  include M
  def meth
    "C - meth"
  end
end

C.new.meth  #输出为 "C - meth"

```

2.在同一个类中包含多个模块时，优先使用最后一个包含的模块

```ruby
module M1
end

module M2
end

class C
  include M1
  include M2
end

p C.ancestors
```

输出结果

```
[12] pry(main)> p C.ancestors
[C, M2, M1, M, Object, PP::ObjectMixin, Kernel, BasicObject]
=> [C, M2, M1, M, Object, PP::ObjectMixin, Kernel, BasicObject]
```

3.套嵌include时，查找顺序也是线性的，如下

```ruby
module M1
end

module M2
end

module M3
  include M2
end

class C
  include M1
  include M3
end

p C.ancestors
```

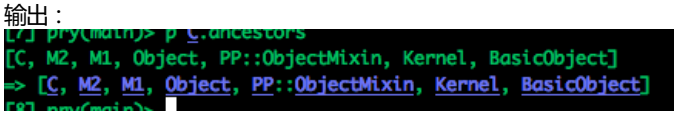
输出结果如下：

```
[21] pry(main)> p C.ancestors
[C, M3, M2, M1, M, Object, PP::ObjectMixin, Kernel, BasicObject]
=> [C, M3, M2, M1, M, Object, PP::ObjectMixin, Kernel, BasicObject]
```

4.相同域名被包含两次以上时，第2次会被省略

```ruby
module M1
end

module M2
end

class C
  include M1
  include M2
  include M1
end

p C.ancestors

```

输出：


[C, M2, M1, Object, PP::ObjectMixin, Kernel, BasicObject]
=> [C, M2, M1, Object, PP::ObjectMixin, Kernel, BasicObject]