Báo cáo về Ruby on Rails môn Vấn đề hiện đại của công nghệ phần mềm

Nhóm 7
Đỗ Văn Toàn
Lê Văn Hưng
Lê Hồng Cẩm
Nguyễn Ngọc Thoại

23/11/2013

Mục lục

1 Giới thiệu về Ruby on Rails **5**
- 1.1 Sơ lược về Rails . . . 5
- 1.2 Các triết lý của Ruby on Rails . . . 6
  - 1.2.1 Don't Repeat Yourself . . . 6
  - 1.2.2 Convention Over Configuration . . . 6
- 1.3 Rails là một Agile framework . . . 6

2 Giới thiệu về ngôn ngữ lập trình Ruby **8**
- 2.1 Sơ lược về Ruby . . . 8
- 2.2 Ruby là ngôn ngữ hướng đối tượng hoàn toàn . . . 9
  - 2.2.1 Quy ước đặt tên . . . 9
  - 2.2.2 Các phương thức . . . 10
- 2.3 Các kiểu dữ liệu . . . 11
  - 2.3.1 Mảng và kiểu băm . . . 11
- 2.4 Các logic chương trình . . . 12
  - 2.4.1 Cấu trúc block và iterator . . . 12
- 2.5 Cấu trúc tổ chức . . . 13
  - 2.5.1 Các class . . . 14
  - 2.5.2 Các module . . . 15

3 Kiến trúc của framework Ruby on Rails **16**
- 3.1 Mô hình Model-Controler-View . . . 16
- 3.2 Model trong Rails . . . 17
  - 3.2.1 Object-Relational Mapping . . . 17
  - 3.2.2 Active Record . . . 18
- 3.3 View và Controller . . . 19
  - 3.3.1 View . . . 19
  - 3.3.2 Controller . . . 19

4 Các thành phần trong Ruby on Rails **20**
4.1 Các thành phần chính trong Rails 20
4.2 Model trong Rails 20
4.2.1 Ánh xạ model với các bảng dữ liệu 21
4.2.2 Data migrations 22
4.2.3 Quan hệ giữa các model 23
4.3 Controller trong Rails 24
4.4 View trong Rails 25
4.5 Routing trong Rails 26

A Hướng dẫn cài đặt **29**

Mở đầu

Trong lĩnh vực công nghệ phần mềm hiện nay đang xuất hiện rất nhiều các xu hướng, phương pháp phát triển phần mềm khác nhau như điện toán đám mây, di động, big data và phương pháp phát triển phần mềm linh hoạt agile hay test-driven development... Cùng với sự phát triển của hạ tầng mạng, các phần mềm ứng dụng ngày càng được xây dựng và triển khai trên nền web nhiều hơn, thay vì được triển khai cục bộ trên các máy tính.

Để hỗ trợ việc xây dựng các ứng dụng web trở lên dễ dàng, nhanh chóng và hiệu quả hơn, rất nhiều công cụ đã được tạo ra, đó là các web framework.Trên thực tế, hiện nay có rất nhiều các web framework, được xây dựng dựa trên các ngôn ngữ khác nhau, một số ví dụ điển hình như Java với các framework như Spring, Struts, PHP có Zend framework, Yii, Python có Django... Các framework được tạo ra phù hợp với một số loại ứng dụng nhất định, mỗi framework đều có các tư tưởng, triết lý riêng và các điểm mạnh điểm yếu, không có cái nào được cho là viên đạn bạc.

Với tư duy của người phát triển, với mỗi miền ứng dụng nhất định thì nên cân nhắc và lựa chọn phù hợp, ví dụ : để xây dựng một hệ thống lớn, có nghiệp vụ phức tạp, đòi hỏi có kiến trúc tốt thì sử dụng Java và các framework hỗ trợ được cho là một giải pháp tốt, nhưng để xây dựng một trang web thông thường, ít mang tính thương mại thì sử dụng một framework nhỏ gọn là một giải pháp hợp lý hơn.

Ruby on Rails là một web framework được viết bằng ngôn ngữ lập trình Ruby, là một framework có rất nhiều các ưu điểm như thời gian phát triển nhanh, chi phí thấp, hướng tới người phát triển, áp dụng các tư tưởng của phương pháp agile và test-driven development. Trên thực tế Ruby on Rails đã được sử dụng thành công trong các ứng dụng như github hay twitter.

Dưới sự hướng dẫn của thầy **TS.Trương Anh Hoàng**, tài liệu này sẽ hướng tới việc tìm hiểu về framework Ruby on Rails cũng như cách sử dụng để xây dựng ứng dụng.

Tài liệu gồm các phần sau:
Chương 1 : Giới thiệu về framework Ruby on Rails
Chương 2 : Giới thiệu về ngôn ngữ lập trình Ruby
Chương 3 : Kiến trúc của Ruby on Rails
Chương 4 : Các thành phần của Ruby on Rails
Và một số phụ lục đính kèm.

Chương 1

Giới thiệu về Ruby on Rails

Chương này sẽ giới thiệu một cách tổng quát, sơ lược về framework Ruby on Rails.

1.1 Sơ lược về Rails

Ruby on Rails, hay Rails, là một web framework mã nguồn mở được viết bằng ngôn ngữ lập trình Ruby. Rails là một full-stack framework : cung cấp đầy đủ các kĩ thuật, công cụ cần thiết để xay dựng một ứng dụng web hiện đại (web 2.0). Rails được xây dựng bởi một lập viên trẻ người Đan Mạch, với bản phát hành đầu tiên 1.0 vào năm 2005, phiên bản hiện nay của Rails là 4.0, và được duy trì và phát triển bởi Rails Core Team.

Tất cả các ứng dụng viết bằng Rails được triển khai sử dụng kiến trúc Model-View-Controller (M-V-C). Rails cũng hỗ trợ việc sinh code tự đông, lập trình viện chỉ cần sử dụng các lệnh cho để tạo các thành phần của ứng dụng một cách chuẩn nhất, gồm model, view, controller... Để hỗ trợ việc test ứng dụng, với mỗi thành phần được sinh ra bằng các lệnh hỗ trợ, Rails cũng tự động sinh ra các test stubs, lập trình viên chỉ cần cài đặt các test stubs này thì đảm bảo các thành phần của ứng dựng đều được test.

Rails được viết bằng Ruby, một ngôn ngữ lập trình hiện đại, hướng đối tượng hoàn toàn. Ruby có cú pháp súc tích, rõ ràng, gần với ngôn ngữ tự nhiên khiến cho chương trình viết bằng Ruby ngắn ngọn, dễ đọc dễ bảo trì, giúp cho việc phát triển ứng dựng bằng Rails thuận lợi hơn.

1.2 Các triết lý của Ruby on Rails

Ruby on Rails bao gồm 2 triết lý giúp các chương trình ngắn gọn, dễ đọc và bảo trì hơn : don't repeat yourself và convention over configuration, giúp công sức bỏ ra của người phát triển được giảm đi đáng kể.

1.2.1 Don't Repeat Yourself

Triết lý này gợi ý rằng nên tránh việc viết các đoạn code trùng lặp nhau, Rails hỗ trợ việc này bằng cách mỗi đoạn code viết ra đều được đặt vào đúng 1 vị trí xác định duy nhất, và được tổ chức rất hợp lý và chặt chẽ, giúp việc quản mã nguồn tốt hơn từ đó trách được việc lặp lại code, với các framework khác, khi thay đổi tổ chức của mã nguồn sẽ kéo theo việc thay đổi của rất nhiều thứ khác.

1.2.2 Convention Over Configuration

Đây là triết lý làm lên đặc trưng của Rails, Rails đưa ra một số các quy ước mặc định, nếu lập trình viên tuân theo các quy ước này thì sẽ phải viết ít code hơn, chủ yếu là các dòng code để thiết lập hệ thống và cơ sở dữ liệu, ví dụ trong khi các lập trình viên Java phải viết rất nhiều các file cấu hình bằng XML, rất dài dòng và tốt công sức. Bên cạnh đó Rails còn giúp việc phát triển dễ dàng hơn khi đã tích hợp sẵn một số thành phần như Ajax hay RESTful web service, các phần này sẽ được trình bày ở các phần sau của tài liệu.

1.3 Rails là một Agile framework

Agile là phương pháp phát triển phần mềm linh hoạt, với mục đích có được sản phẩm phần mềm nhanh nhất. Tuyên ngôn của agile được phát biểu như sau :

- Các cá nhân và sự tương tác hơn là công cụ và quy trình
- Phần mềm chạy được hơn là tài liệu đầy đủ
- Cộng tác với khác hàng hơn là hợp đồng ràng buộc
- Phản ứng với thay đổi hơn là làm theo một bản kế hoạch

Thực tế, Rails hướng tới các cá nhân và sự tương tác, không có các bộ công cụ phát triển cồng kềnh, không có các cấu hình phức tạp, không có các quy trình phát triển chi tiết. Rails không phản đối việc sử dụng tài liệu phát triển, nhưng quy trình phát triển ứng dựng bằng Rails không được định hướng bởi tài liệu. Rails làm cho việc tạo tài liệu trở lên dễ dàng bằng việc hỗ trợ tự sinh tài liệu dạng HTML cho toàn bộ code của ứng dụng.

Rails đặc biệt coi trọng việc test ứng dụng gồm unit test và functional test, các test stubs được sinh ra cùng với các chức năng được tạo ra, khi mỗi thành phần được xây dựng và test theo cách này, ứng dụng sẽ luôn chạy được và có thể dùng để giao tiếp với khách hàng, từ đó có được phản hồi sớm nhất và dễ dàng thay đổi nhanh chóng và phù hợp. Sản phẩm cuối cùng được hoàn thành khi các chức năng được triển khai hết.

Chính vì tính linh hoạt, nhanh chóng trong việc xây dựng ứng dụng, Rails được sử dụng rất nhiều để xây dựng các trang web dạng start-up và các ứng dụng mobile, khi yêu cầu có được sản phẩm sớm nhất và dễ dàng thay đổi, nâng cấp được đặt lên hàng đầu.

Tổng kết

Rails là một framework đã được chứng minh tính hiệu quả trong thực tế. Để phát triển được ứng dụng bằng Ruby on Rails, người lập trình cần có một số kĩ năng lập trình Ruby, thực tế Rails chỉ sử dựng chủ yếu một số các cú pháp nổi bật của Ruby và đặt trong một phạm vi giới hạn giúp tối ưu việc lập trình, chương sau của tài liệu sẽ giới thiệu về ngôn ngữ lập trình Ruby và các thành phần được sử dựng nhiều trong Rails.

Chương 2

Giới thiệu về ngôn ngữ lập trình Ruby

Chương này sẽ giới thiệu về các thành phần, đặc điểm nổi bật của ngôn ngữ lập trình Ruby

2.1 Sơ lược về Ruby

Ruby là một ngôn ngữ lập trình động, hướng đối tượng, đa mục đích được thiết kế bởi một lập trình người Nhật là Yukihiro Matsumoto. Phiên bản đầu tiên của Ruby 1.0 xuất hiện vào năm 1995, phiên bản ổn định hiện nay là 2.0.

Ruby được cho là một ngôn ngữ lập trình có cú pháp ngắn gọn, gần với ngôn ngữ tự nhiên, hướng tới người lập trình với năng suất cao, mang lại sự hài lòng cho lập trình viên thay vì hướng tới hiệu năng của chương trình.

Ruby trở lên thực sự nổi tiếng vào năm 2005 nhờ sự ra đời của framework Ruby on Rails, một framework dùng để phát triển các ứng dụng web nhanh chóng.

2.2 Ruby là ngôn ngữ hướng đối tượng hoàn toàn

Ruby là một ngôn ngữ hướng đối tượng hoàn toàn, mọi thứ trong Ruby đều là đối tượng, kiểu số trong Ruby cũng được biểu diễn bởi các đối tượng, đoạn code 2.1 mô tả việc sử dụng các đối tượng trong Ruby.

Listing 2.1: Các đối tượng trong Ruby

```
# Ruby has dynamic types
a = 10
# Print type of a, class Fixnum
print a.class()
# Object id of a
print a.object_id()
# A number is an object, a.next() = 11
print a.next()
# Now a is a string object
a = "Ruby␣is␣fun"
a.class()
# a now has another object id
print a.object_id()
```

2.2.1 Quy ước đặt tên

Ruby đưa ra một số quy ước đặt tên cho các thành phần như các biến, các method, các class.

Đặt tên biến trong Ruby các biến được chia ra thành các loại : biến cục bộ, biến thực thể, biến lớp và hằng số. Tất cả các loại biến đều được đặt tên bằng chữ cái thường, các số và dấu gạch dưới để ngăn cách các từ, riêng hằng số được đặt tên bằng chữ cái in hoa. Biến cục bộ bắt đầu bằng chữ cái in thường, biến thực thể bắt đầu bằng 1 kí tự @, biến lớp được bắt đầu bằng 2 kí tự @, đoạn code 2.2 chỉ ra cách đặt tên trong Ruby.

Các method được đặt tên bằng chữ cái thường, các từ được phân cách bằng dấu gạnh dưới. Các class trong ruby phải bắt đầu bằng chữ cái in hoa, các từ tiếp theo thì chữ cái đầu được viết hoa.

Listing 2.2: Cách đặt tên biến trong Ruby

```
# Local variable
a = 10
# Constant
CONST = 100

# Class name
class StudentClass
# Class variable
    @@count = 1
# method name
  def set_name(name)
  # Instance variable
    @name = name
  end
end
```

2.2.2 Các phương thức

Các phương thức trong Ruby có thể tồn tại ở 2 vị trí, bên trong class và bên ngoài class. Method bên trong chia ra làm 2 mức là static method và instance method. Method trong Ruby không cần khai báo kiểu giá trị trả về, hơn thế nữa có thể trả về nhiều giá trị cho một method. Một biến cuối cùng trong thân method sẽ được trả về nếu không có biến nào được trả về trực tiếp bằng từ khóa **return**, và dấu ngoặc đơn bao quanh các tham số của 1 method có thể lược bỏ đi, đoạn code 2.3 mô tả cách dùng các method trong Ruby.

Listing 2.3: Cách dùng method trong Ruby

```
# Outside class method
def say_hello name
    puts name
end
# Use method
say_hello "Ruby"

# Inside class method
```

```
class Student
    @@count = 10
# Static method
    def self.get_count
    # Return @@count
      @@count
    end
# Instance method
def say_hello name
    puts name
end
# Use static method
puts Student.get_count
# Use instance method
s = Student.new # Initiate
s = say_hello "Rails"
```

2.3 Các kiểu dữ liệu

Mọi kiểu dữ liệu trong Ruby đều được mô tả bằng các class, các giá trị cụ thể được biểu diễn dưới dạng các đối tượng, ở đây chỉ đề cập tới một số kiểu dữ liệu đặc trưng trong Ruby.

2.3.1 Mảng và kiểu băm

Mảng và kiểu băm là các tập hợp được đánh chỉ số, dùng để lưu các đối tượng, và được truy cập bằng khóa. Với mảng khóa là các số nguyên, trong khi đó kiểu băm cho phép khóa thuộc bất kì kiểu đối tượng nào. Cả mảng và kiểu băm trong Ruby đều cho phép truy cập, sửa đổi, thêm, xóa các phần tử, đoạn code 2.4 mô tả cách sử dụng mảng và kiểu băm.

Listing 2.4: Cách dùng method trong Ruby

```
# Using arrays
# Create an array
a = [1, 2, "ruby", "rails"]
# Access an element
print a[0] # => 1
```

```
a[1] = 100
a.delete_at(2)
# Read all elements
for e in a
    puts e
end

# Using hashes
# Create a hash
hash = {1 => 'one', 'r' => 'Ruby',
       :symbol => 'symbols are immutable strings' }
print hash[1] # => 'one'
# Symbol object as a key of hash
# :symbol is identical to 'symbol'
print hash[:symbol] # => 'symbols are immutable strings'
```

Trong Ruby, Symbol là một class đặc biệt, được dùng để lưu trữ những string mà chỉ được tạo một lần, lưu trữ bên trong trình thông dịch của Ruby, gọi là table symbol một đối tượng thuộc class Symbol một khi được tạo, các lần tạo sau nếu đối tượng được tạo có cùng giá trị với đối tượng trước, thì đối tượng sau không được mới mà sử dụng lại đối tượng đã được tạo và lưu trữ trước đó.

2.4 Các logic chương trình

Ruby cung cấp một số cấu trúc đặc trưng để điều khiển logic của chương trình, ngoài các cấu trúc thông thường như lặp hay rẽ nhánh, trong Ruby còn có một số cấu trúc đặc trưng khác.

2.4.1 Cấu trúc block và iterator

Block là một cấu trúc nhóm một số đoạn code lại với nhau, block trong Ruby được dùng phổ biến nhất với các method, như một cách giúp mở rộng thêm các xử lý của một method, có thể truyền một block vào một method giống như một tham số. Đoạn code 2.5 chỉ ra cách dùng block với method.

Listing 2.5: Các dùng block với method trong Ruby

```
# A block
{ puts "block" }
# A block
do
    puts "another␣way␣to␣creates␣block"
end

# Method uses block
def foo(&block)
    block.call "block␣and␣method"
end
# Pass block to method
foo do |x|
    puts x
end
```

Iterator là cách để duyệt các phần tử trong một tập hợp như mảng hay kiểu băm trong ruby, đoạn code 2.7 chỉ ra cách dùng iterator để duyệt mảng và kiểu băm.

Listing 2.6: Cách dùng iterator để duyệt mảng và kiểu băm

```
# Iterate an array
a = [1, 2, "ruby", "rails"]
a.each do |e|
    puts e
end

# Iterate a hash
h = {:r => "ruby", :j => "java", :p => "python"}
h.each do |k, v|
    puts k, v
end
```

2.5 Cấu trúc tổ chức

Các method trong Ruby có thể được nhóm vào trong các class hoặc các module.

2.5.1 Các class

Một class là một bản mô tả cho cách tạo các đối tượng, gồm các thuộc tính và các method. Method được chia thành 2 mức, mức đối tượng (instance method) và mức class (static method). Thuộc tính cũng được chia mức tương tự. Các method được phân định giới hạn truy cập ở 3 mức, public, protected, private. Mặc định nếu không khai báo gì thêm thì các method là public. Các giới hạn truy cập này không áp dụng cho các thuộc tính, các thuộc tính chỉ được truy cập thông quy các method, setter và getter. Đoạn code **??** chỉ ra cách dùng class trong Ruby.

Listing 2.7: Cách dùng class trong Ruby

```
# Declare a class
class Student
    @@count = 0
    # Default constructor's name
    def initialize name, id
        @name = name
        @id = id
    public
        def set_name name
          @name = name
        end

       def get_name
         @name
       end
     private
       def private_method
         puts "private␣method"
       end
end

# Using class
s = Student.new "Ruby", "r"
s.get_name # => "Ruby"
s.private_method # => Error raised
```

2.5.2 Các module

Module trong Ruby được dùng với 2 mục đích, sử dụng để quản lý mã nguồn như các namespace, và dùng để thêm các method vào trong một class một cách linh hoạt. Module nhóm các class, các method, các biến vào bên trong. Đoạn code 2.8 mô tả cách dùng của module.

Listing 2.8: Cách dùng module trong Ruby

```
# Declare a module
module IT
class Programmer
    def program
      puts "I␣can␣program"
    end
end

class Hacker < IT::Programmer
   def hack
     puts "I␣can␣hack"
        end
end

hacker = Hacker.new
hacker.program # => "I can program"
hacker.hacker # => "I can hack"
```

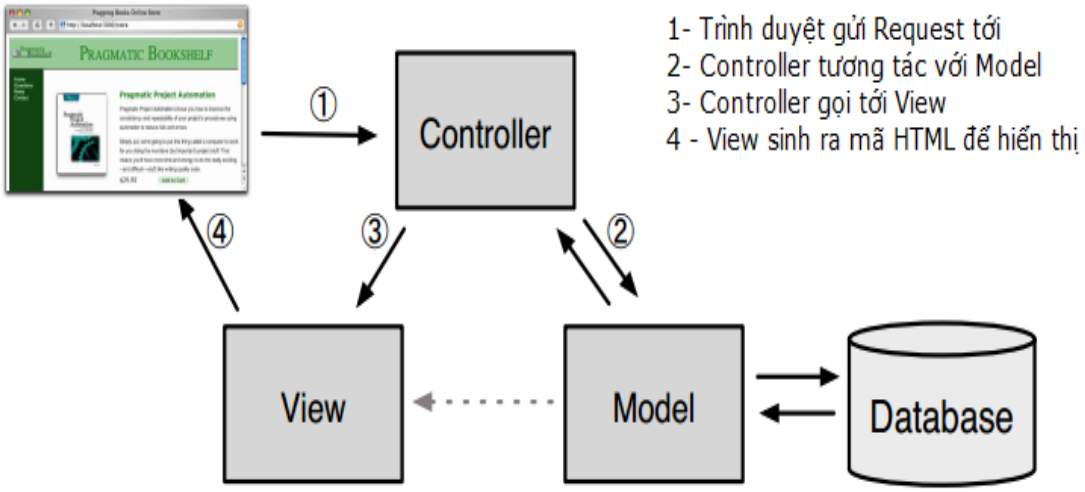
Chương 3

Kiến trúc của framework Ruby on Rails

Giới thiệu kiến trúc tổng quan của Rails

3.1 Mô hình Model-Controler-View

Các thành của ứng dựng Rails được chia thành ba thành phần gồm : model, controller và view.
Model có nhiệm vụ duy trì trạng thái của ứng dụng, đó có thể là các thông tin, dữ liệu tương tác với người dùng, có nhưng thông tin là tạm thời và mất đi sau một vài tương tác với người dùng, những thông tin được lưu trữ lâu dài bên ngoài ứng dung, thông thường trong cơ sở dữ liệu.
Model không chỉ lưu trữ dữ liệu mà còn chứa một số các quy tắc thương mại áp dụng trên các dữ liệu đó. Ví dụ, một model là tên là Account chứa các thông tin về một tài khoản ngân hàng, gồm số tiền trong tài khoản và độ tuổi của khác hàng, tài khoản chỉ được mở khi số tiền ban đầu là nạp vào là 100 000 VND và độ tuổi của khách hàng phải từ 18 trở lên. Model Account có thể chứa các ràng buộc để các quy tắc trên được đáp ứng.
View có nhiệm vụ tạo ra giao diện người dùng, thông thường dựa trên model, các dữ liệu được duy trì trong model, và được hiển thị tới người dùng thông qua view dưới dạng mã HTML. View và model không giao tiếp trực tiếp với nhau mà phải thông qua controller, hình 3.1 mô tả cách các thành phần trong Rails tương tác với nhau.
Controller có nhiệm vụ điều phối giữa các thành phần của ứng dụng, con-



Hình 3.1: Kiến trúc M-V-C trong Rails

troller nhận các request từ người dùng, tương tác mới model và sau đó gọi tới một view tương ứng để hiển thị thông tin từ model. Model được lấy ra từ database nhờ một cơ chế hỗ trợ là Object-Relational Mapping, phần này sẽ trình bày ở phần tiếp theo.

3.2 Model trong Rails

Trên thực tế Rails là web framework hướng cơ sở dữ liệu, có nghĩa là database sẽ là phần quan trọng nhất của ứng dựng, chính vì thế Rails có một số thành phần hỗ trợ cho việc xử lý cơ sở dữ liệu và model trở lên dễ dàng hơn.

3.2.1 Object-Relational Mapping

Object-Relational Mapping (ORM) là một cơ chế giúp ánh xạ một bảng dữ liệu trong cơ sở dữ liệu quan hệ với một class trong một ngôn ngữ lập trình hướng đối tượng, các dòng trong bảng thành các đối tượng, các cột thành các thuộc tính của đối tượng, và ngược lại.
Để tránh phải viết nhiều các câu lênh SQL khiến ứng dụng khó bảo trì cơ chế ORM hỗ trợ các phương thức để giao tiếp với cơ sở dữ liệu. Đối với một

số framework khác việc cấu hình giữa cơ sở dữ liệu và model sẽ yêu cầu lập trình viên phải thực hiện, thông thường phải viết và duy trì khá nhiều file cấu hình dạng XML. Trong Rails việc cấu hình sẽ không cần phải thực hiện nhờ sự hỗ trợ của thành phần gọi là Active Record.

3.2.2 Active Record

Active Record là một mẫu thiết kế sử dụng trong Object-Relational Mapping, mẫu thiết kế Active Record đưa ra các phương thức cơ bản giúp việc giao tiếp với cơ sở dữ liệu như create, read, update, delete - CRUD.
Trong Rails có một thành phần là Active Record hỗ trợ cơ chế ORM. Về cơ bản thành phần này tuần theo mô hình ORM chuẩn : bảng ánh xạ với class, dòng ánh xạ với đối tượng, cột ánh xạ với thuộc tính. Nhưng cách cấu hình để liên kết giữa cơ sở dữ liệu và model khác biệt. Bằng cách dựa trên các cấu hình quy ước mặc định, Active Record giảm thiểu việc phải viết các file cấu hình. Chỉ cần một khai báo một class model theo chuẩn của Rails thì model đó sẽ được tự động ánh xạ với một bảng dữ liệu trong cơ sở dữ liệu, cũng như cung cấp ngay các phương thức để giao tiếp với cơ sở dữ liệu. Đoạn code ở Listing 3.1 sẽ mô tả việc tạo môt model trong Rails cũng như sử dụng các phương thức để thực hiện các thao tác như lưu một bản ghi hay tìm một bản ghi.

Listing 3.1: Active Record trong Rails

```
# Declares a model
class User < ActiveRecord::Base
end
# Initiates an user and stores into database
user = User.new
user.save
# Finds a user whose id is 1
user = User.find(1)
```

Active Record là một thành phần (một module) nền tảng trong kiến trúc M-V-C của Rails.

3.3 View và Controller

View và Controller là 2 thành phần có quan hệ khá chặt chẽ với nhau, controller chuyển dữ liệu từ model để hiển thị trong View, View gửi các sự kiện tới Controller để được xử lý, bởi các sự tương tác này View và Controller được đặt vào trong một thành phần gọi là Action Pack.

3.3.1 View

View trong Rails đơn giản chứa các đoạn mã HTML cho các nội dung cố định, để hiển thị các thông tin động được lấy từ cơ sở dữ liệu, View có thể nhúng các đoạn mã Ruby vào bên trong, được gọi là Embedded Ruby với phần mở rộng là .html.erb. Việc nhúng mã Ruby vào trong View có thể phá vỡ cấu trúc của mô hình MVC nếu các đoạn mã đó chứa các xử lý logic của ứng dụng, vì vậy các đoạn code Ruby nhúng vào chỉ nên dùng để hiển thị thông tin trong Model được cung cấp bởi Controller, các xử lý logic nên được đưa hết vào bên trong Controller và Model.

3.3.2 Controller

Controller là phần trung tâm xử lý các logic của ứng dụng, cộng tác các tương tác giữa người dùng, View, Model với nhau. Việc xây dựng ứng dụng Rails tập trung vào mức xây dựng từng chức năng hoàn chỉnh, việc này giúp việc phát triển và bảo trì rất dễ dàng, mỗi chức năng sẽ được định nghĩa trong Controller.

Chương 4

Các thành phần trong Ruby on Rails

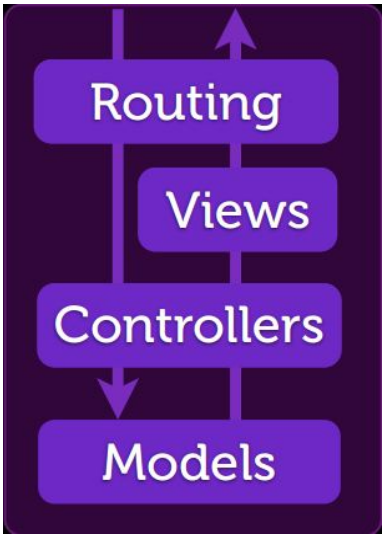
Chương này trình bày chi tiết về các thành phần chính trong framework Ruby on Rails.

4.1 Các thành phần chính trong Rails

Các thành chính trong Rails gồm : Model, View, Controller và Routing, hình 4.1 chỉ ra vị trí của 4 thành phần trong Rails. Các thành phần tương tác, quan hệ chặt chẽ với nhau. Việc cấu hình để các thành phần này hoạt động với nhau được Rails đơn giản hóa đi rất nhiều bằng cách cung cấp quy ước cấu hình, đúng theo triết lý Convention over Configuration. Bên cạnh đó Rails cũng cung cấp các lệnh tương tác với shell để tạo ra các thành phần trên một cách thuận lợi và đúng quy chuẩn nhất.

4.2 Model trong Rails

Model là một thành phần được cho là quan trọng nhất trong các ứng dụng Ruby on Rails, Rails hỗ trợ rất nhiều các thành phần để việc làm việc với Model trở lên dễ dàng và tiện lợi, các thành phần hỗ trợ cho Model được đặt trong một module là Active Record. Các model trong Rails kế thừa lớp Base từ module ActiveRecord. Các model được đặt trong thư mục model trong thư mục app của 1 project Rails.



Hình 4.1: 4 thành phần chính trong Rails

4.2.1 Ánh xạ model với các bảng dữ liệu

Một bảng dữ liệu trong cơ sở dữ liệu quan hệ được biểu diện bởi một model. Các model trong Rails được cài đặt theo mẫu thiết kế Active Record giúp việc thao tác với các bảng dữ liệu trở lên dễ dàng thông qua các method được cung cấp sẵn, gồm các thao tác Creat, Read, Update, Delete - CRUD. Model được tạo tự động bằng câu lệnh "rails generate model". Doạn code 4.1 chỉ ra các tạo một model từ một câu lệnh, và thao tác với cơ sở dữ liệu

Listing 4.1: Model class trong Rails

```
# rails generate model User name:string phone:string
# A model class
class User < ActiveRecord::Base
end

# CRUD with model
# Create and store a model into database
User.create(:name => "user␣1")

# Get a model from database by id
```

```
user = User.find(1)

# Update a model
user.update_attributes(:name => "USER␣1")

# Delete a User by id
User.destroy(1)
```

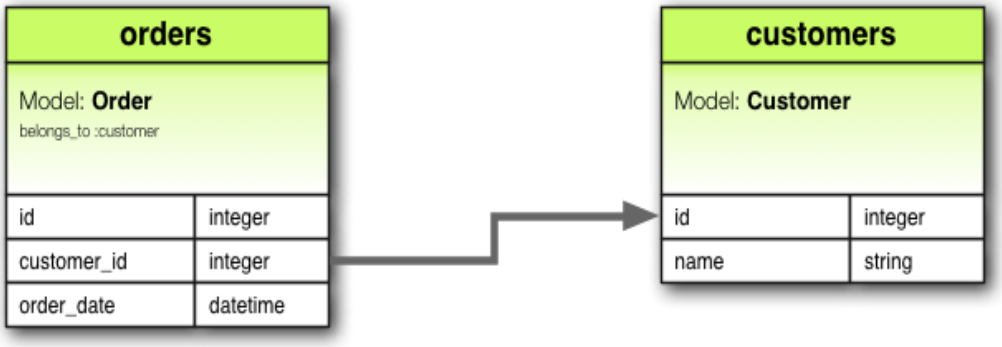
4.2.2 Data migrations

Data migrations là một cách để tạo, thay đổi các bảng dữ liệu trong Rails, thay vì phải viết các lệnh SQL để tạo cấu trúc của các bảng dữ liệu, migrations là các Ruby class, ở mức trừu tượng cao hơn, tọa thuận lợi cho việc viết mã cũng như dễ đọc và bảo trì và quản lý. Migrations độc lập với cơ sử dữ liệu vật lý sử dụng bên dưới, tùy thuộc vào hệ quản trị cơ sở dữ liệu sử dụng mà migrations được dịch thành các mã SQL phù hợp. Bên cạnh đó migration cũng là cho việc quản lý các phiên bản của cơ sở dữ liệu một cách thuận lợi bằng việc đánh phiên bản cho mỗi lần thay đổi cơ sở dữ liệu sử dụng migration, từ đó có thể chuyển đổi các phiên bản tùy ý.

Mỗi model được tạo ra bằng lệnh của Rails, thì 1 class migration được tạo ra tương ứng với model đó. Các migration class kế thừa class Migration từ model ActiveRecord. Để tạo bảng dữ liệu từ migration class dùng lệnh rake db:migrate. Đoạn code 4.2 mô tả một migration class.

Listing 4.2: Migration class trong Rails

```
# Migration class for creating users table
# Generated from User's generation
class CreateUsers < ActiveRecord::Migration
    def change
        create_table :users do |t|
            t.string :name
            t.string :phone
            t.timestamps
        end
    end
end
```



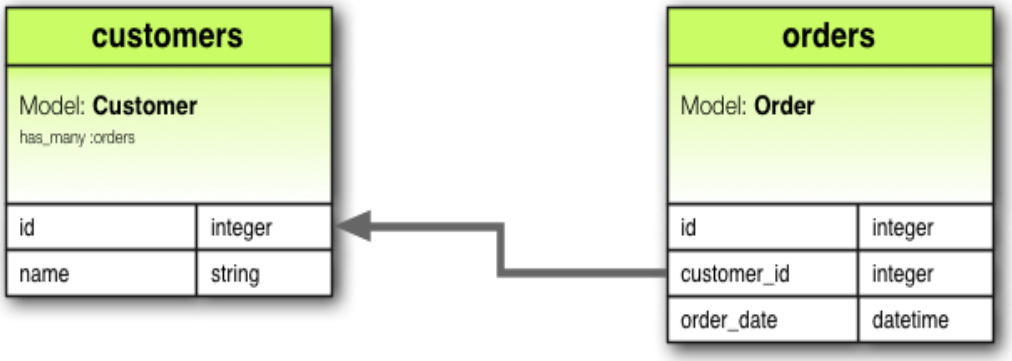

```
class Order < ActiveRecord::Base
  belongs_to :customer
end
```

Hình 4.2: 1 Order thuộc về 1 Customer

4.2.3 Quan hệ giữa các model

Các bảng dữ liệu trong hệ cơ sở dữ liệu quan hệ cũng có các quan hệ với nhau như 1-nhiều, 1-1, nhiều-nhiều, các model biểu diễn các bảng dữ liệu nên giữa các model cũng có những quan hệ này. Rails cung cấp một số method để việc tạo các quan hệ này giữa các model trở lên dễ dàng, chỉ phải thêm các method cho các quan hệ tương ứng và bên trong phần định nghĩa của các model class thì các quan hệ được thiết lập. Việc này giúp cho việc lập trình trở lên cực kì thuận tiện và dễ dàng, thay vì phải tạo các file cấu hình rất dài, đối với các framework khác.

Ví dụ tạo quan hệ giữa 2 model Order và Customer, một Order thuộc về 1 Customer được biểu diễn bằng method `belongs_to`, Customer có nhiều Order, được biểu diễn bằng method `has_many`. Lúc này order có thể biết được customer và customer có thể quản lý được các order của mình đã đặt. Hình 4.2 và 4.3 và mô tả việc tạo quan hệ giữa 2 model Order và Customer.




```
class Customer < ActiveRecord::Base
  has_many :orders
end
```

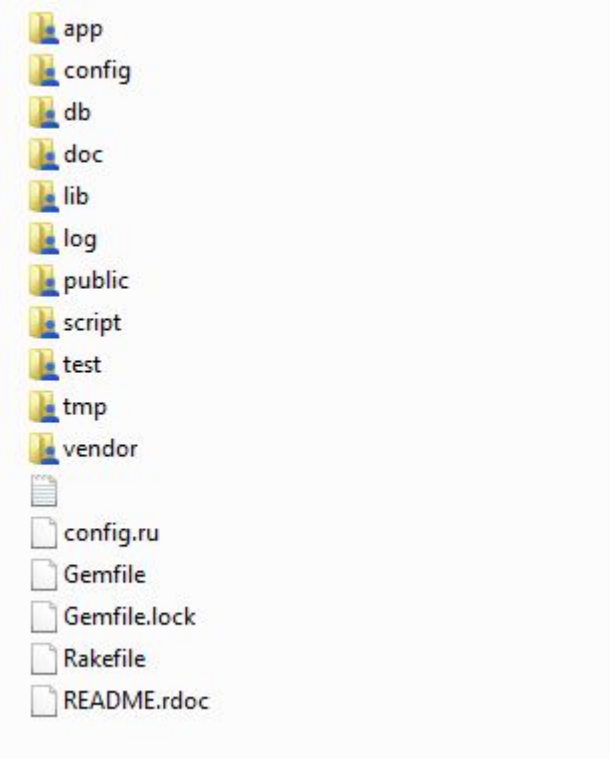
Hình 4.3: 1 Customer có nhiều Order

4.3 Controller trong Rails

Controller là thành phần điều phối tất các thành phần còn lại như Model và View. Controller trong Rails kế thừa lớp ActionController::Base, các controller được đặt trong thư mục controller của thư mục app, giúp việc quản lý trở lên thuận tiện. Một controller có nhiều các method, mỗi phương thức tương ứng với một action sẽ thực hiện bởi các request từ người dùng. Rails cũng cung cấp một câu lệnh chuẩn để tạo controller. Đoạn code 4.3 chỉ ra các tạo một model trong Rails.

Listing 4.3: Controller trong Rails

```
# rails generate controller welcome index
# A controller with a method and view both named index
class WelcomeController < ApplicationController
    # GET welcome/index
    def index
    end
end
```



Hình 4.4: Cấu trúc 1 project Rails

Mỗi method trong conttroller sẽ được gọi tới bằng môt HTTP request, và mỗi method sẽ được gắn với một request xác định được quy định trong thành phần định tính của ứng dụng Rails.

4.4 View trong Rails

View là thành phần dùng để hiển thị dữ liệu từ model được cung cấp bởi controller, view là thành phần mà các triết lý của Rails được thể hiện rõ nhất, bằng việc hỗ trợ tự động sinh mã sử dụng method render. Mỗi view được biểu diễn bởi một file có phần mở rộng .html.erb và tương ứng với một method trong controller tương ứng, tất cả các view của một controller được đặt trong một thư mục có tên là controller đó nằm trong thư mục app/view, hình 4.4 là cấu trúc của một project Rails chuẩn.

Thư mục app là thư mục chứa các controller, view, model. Thư mục db

chứa các thành phần về cơ sở dữ liệu, thư mục test chứa các thành phần để test ứng dụng.

View trong rails được tạo theo các mẫu có sẵn hay gọi là các template. Ví dụ có thể sinh tự động các view từ model bằng cách dùng câu lệnh rails g scaffold, đoạn code 4.4 mô tả việc tạo các view cho một model.

Listing 4.4: Tạo form tự động cho việc tạo model post

```
# rails generate scaffold post
# name:string title:string content:text
<%= form_for(@post) do |f| %>
  <div class="field">
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </div>
  <div class="field">
    <%= f.label :title %><br />
    <%= f.text_field :title %>
  </div>
  <div class="field">
    <%= f.label :content %><br />
    <%= f.text_area :content %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

4.5 Routing trong Rails

Routing có nhiệm vụ đưa các request từ người dùng tới các method trong controller một cách phù hợp. Mỗi HTTP request trước khi được gửi tới các method trong controller được xử lý, sẽ được kiểm tra trong bảng định tuyến của thành phần routing, nếu request đó tương ứng với một method trong controller nào đó, thì method đó sẽ được gọi. Hình 4.5 là bảng định tuyến của một controller là posts với các method tương ứng trong bảng.

Việc định tuyến được quy định trong file routes.rb trong thư mục config, Rails cung cấp một các dễ dàng để tạo một bảng định tuyến như hình 4.5

HTTP Method	Path	action	named helper
GET	/posts	index	posts_path
GET	/posts/new	new	new_post_path
POST	/posts	create	posts_path
GET	/posts/:id	show	post_path(:id)
GET	/posts/:id/edit	edit	edit_post_path(:id)
PUT	/posts/:id	update	post_path(:id)
DELETE	/posts/:id	destroy	post_path(:id)

Hình 4.5: Bảng định tính của controller posts

bằng cách thêm dòng lệnh resources :posts trong file routes.rb, để xem tất cả các định tuyến trong 1 ứng dụng sử dụng câu lệnh rake routes.

Tổng kết

Các thành phần trong Ruby on Rails đều được thiết kế giúp việc xây dựng ứng dụng trở lên nhanh gọn và thuận tiện nhất, mang đúng tư tưởng của phương pháp phát triển phần mềm agile, giúp cho người lập trình tăng năng suất và sự hài lòng.

Phụ lục A

Hướng dẫn cài đặt

hướng dẫn cài đặt

Tài liệu tham khảo

[1] The Ruby Programming Language 2008 by David Flanagan and Yukihiro Matsumoto - Ruby's creator

[2] Agile Web Development with Rails Fourth Edition by Sam Ruby, Dave Thomas and David H. Hansson - Rails' creator