TomLan

Sinatra & Rack Tutorial

慕凡@NISRA 2013/03/30

主題

- Rack
- Sinatra
- REST

TomLan

RACK

A Ruby Webserver Interface

Rack provides a minimal interface between webservers supporting Ruby and Ruby frameworks.

TomLan

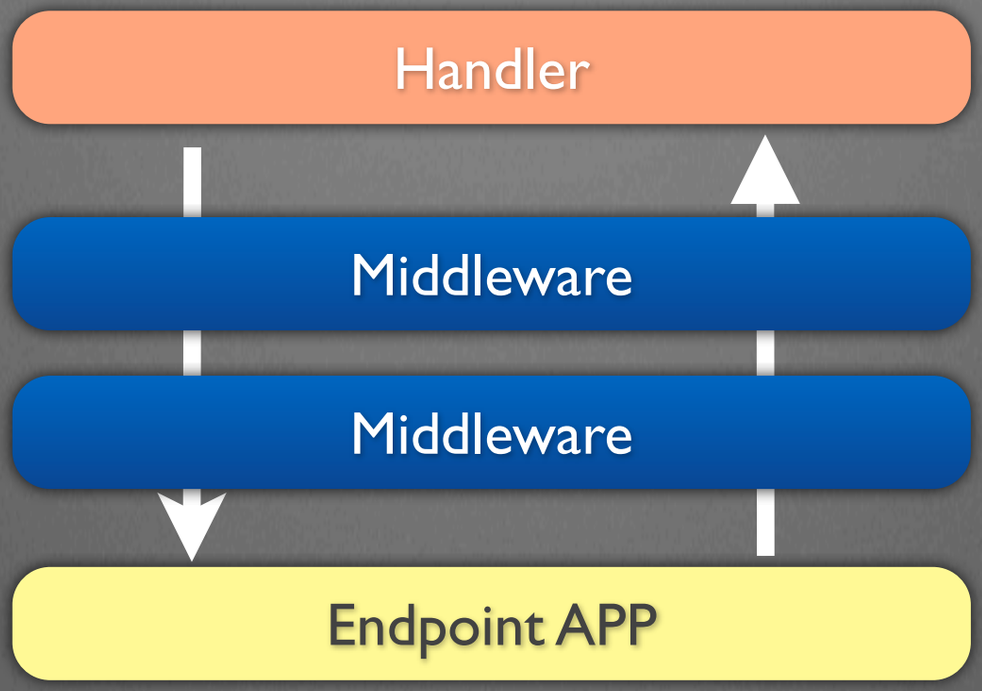
Why Rack

TomLan

Bunch of middlewares

讓不同的Application/framework可以接在一起並且可以共用中間件

What's Rack



Rack 組件類型

Handler

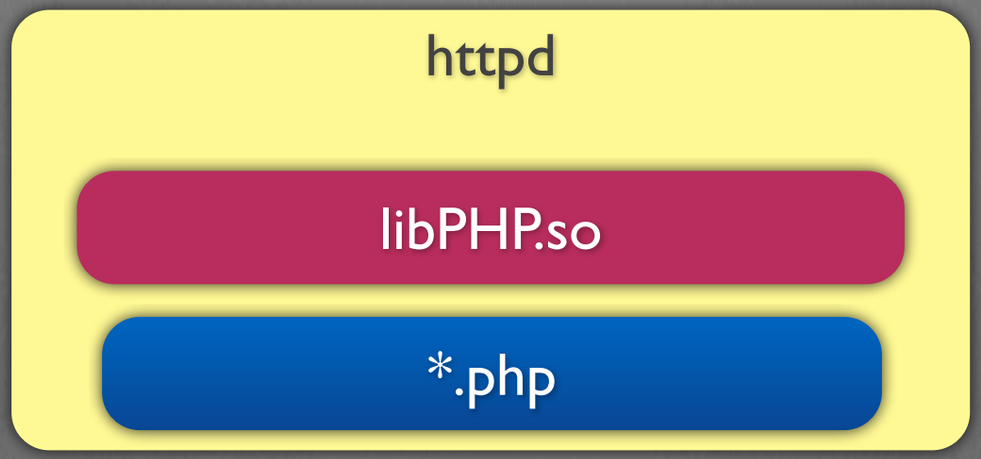
- 處理Application和Rack Application之間的I/O
- 通常屬於特定Applicaiton Server的一部份

Application Server

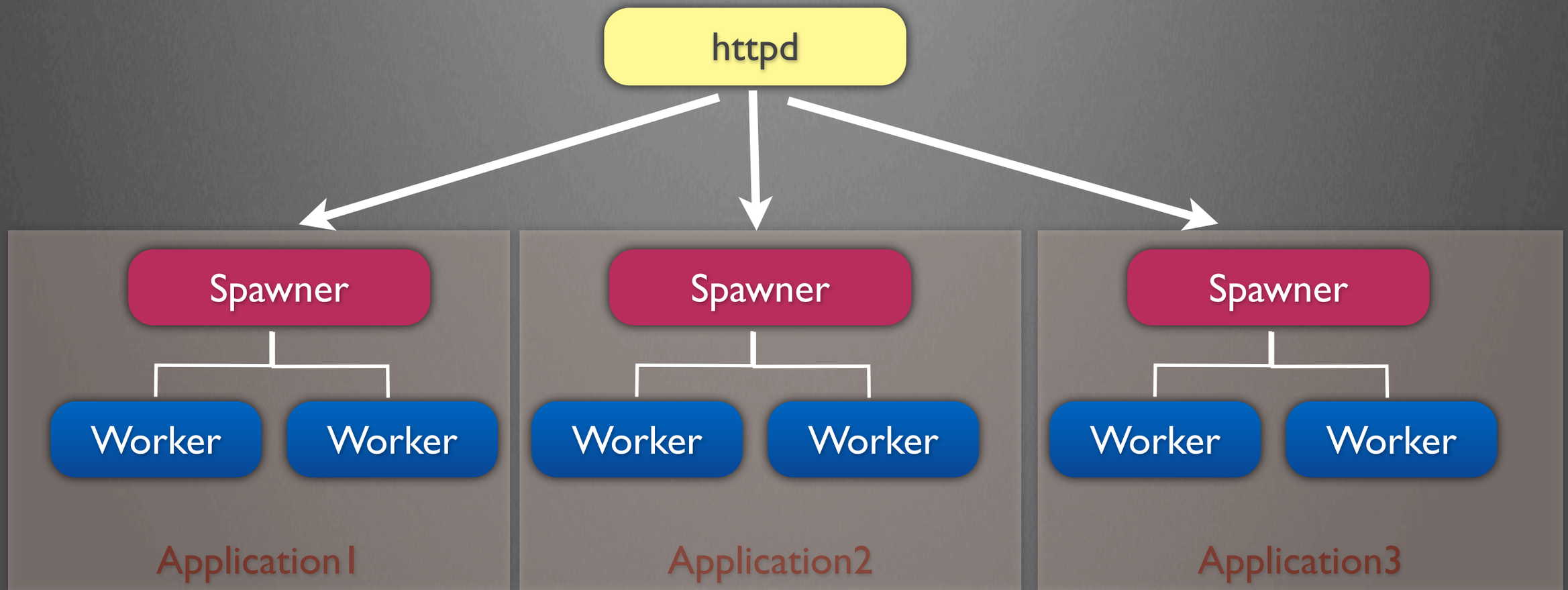
- 和PHP/ASP不同
- 通常不會隨後端server一同啟動
- Ruby的世界有數十種
- Rack是公約數

TomLan

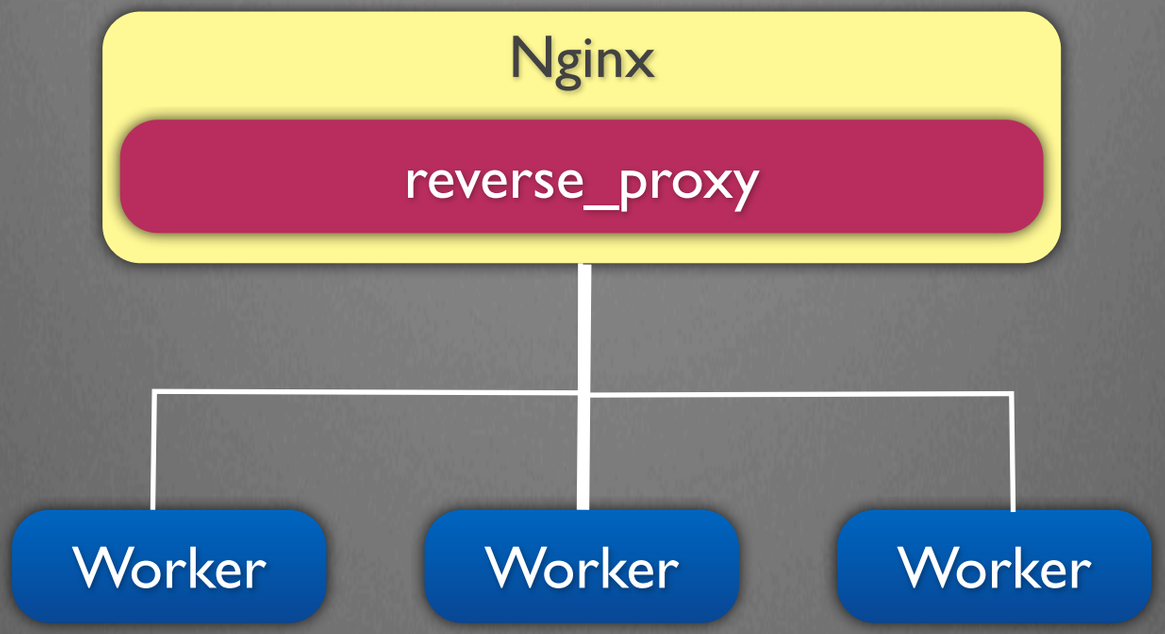
libPHP之場合



Passenger


httpd
Spawner
Spawner
Spawner
Worker
Worker
Worker
Worker
Worker
Worker
Application1
Application2
Application3

Thin



Middleware

- 通常用來處理環境/headers等等
- 所謂中間件

EndPoint App

- 其實也是middleware
- response body內容的產生者

Middleware的條件

```ruby
class Middle1

  def initialize(app = nil)
    @app = app
  end

  def call(env)
    status, headers, resp = @app.call(env)
    resp = resp.first + "<p>123</p>"
    [status, headers, [resp]]
  end

end
```

Middleware的條件

- initialize 接受app參數
- call 接受env參數, 並且回傳

1. http status code
2. response headers
3. response body(respond_to each)

env參數

- hash

  1. request headers
  2. middleware間傳遞的變數

Rackup file

```
require './end_point'
require './middle1'
require './middle15'

use Middle1
use Middle15

run EndPoint.new
```

Rackup file

- 設定middleware初始參數
- 設定middlewares的順序
- 設定各EndPoint的掛載點

範例1：single endpoint

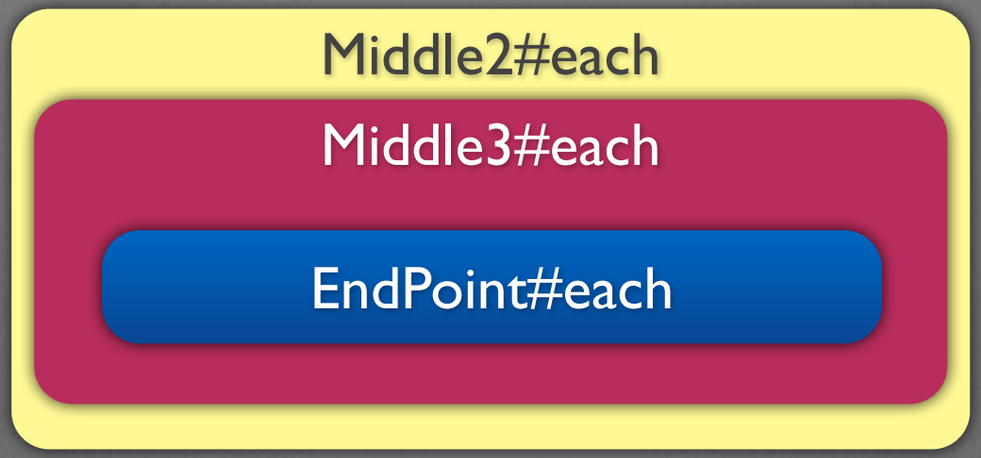
範例2：Middleware + Endpoint

範例3：Middleware + Endpoint with each implementation

執行順序

```
require './end_point2'
require './middle1'
require './middle2'
require './middle3'
use Middle2
use Middle3
run EndPoint2.new
```

each graph



執行順序

1. initialize：上到下
2. call：上到下
3. response body(each)：下到上

顺序非常重要

TomLan

Middleware基本類型

修改response body

- defalter
- file

改變執行環境

- CommonLogger

修改header/env

- method_override
- request
- session
- etag
- warden

Sianatra

Sinatra

README
DOCUMENTATION
CONTRIBUTE
CODE
CREW
ABOUT

Put this in your pipe

```
require 'sinatra'

get '/hi' do
  "Hello World!"
end
```

and smoke it

```
$ gem install sinatra
$ ruby -rubygems hi.rb
== Sinatra has taken the stage ..
>> Listening on 0.0.0.0:4567
```

TomLan

- very micro web framework(少於1600行)
- Like CodeIgniter of PHP
- pure Ruby/Rack
- DSL化路由設計
- 最新版1.4

Micro vs Full Stack

- 自己選擇的ORM
- 自己選擇template engine
- 相對單純之route
- 從多變少或從少變多

TomLan

Why you should learn Sinatra

時勢所趨

http://robbinfan.com/blog/40/ruby-off-rails

移动时代，Web服务将取代Web网站

随着最近几年智能手机的迅速普及，如今来自智能手机和移动设备的总体Web访问和服务请求量已经超过了传统的PC，这意味着Web时代主流的Browser/Server的架构重新回到了Mobile Client/Server的架构。在B/S架构下，在服务器端生成完整的HTML页面，我们需要开发一个完整的Website；但在移动时代，服务器端的功能大大简化了，退化成了Web API调用接口提供者，而复杂的界面构造、交互和运算都是在移动客户端完成的。

传统的Website将越来越让位于Web Service，移动客户端无论是iOS，Android还是HTML5都通过API调用获取服务器端的json/xml格式的数据，无需服务器端生成HTML页面了。这种B/S架构重新往C/S架构的迁移，也意味着full-stack Web框架将越来越没有用武之地了。

Web服务器端并发常见的三种应用场景：

- Website：传统Web网站
- Web Service：Web服务端提供API调用接口
- real-time：Web实时推送

并发场景	业务逻辑	界面构造	数据格式	IO并发
Website	复杂，功能多	服务端组装页面	HTML页面	很低
Web Service	简单，功能少	客户端组装页面	json/xml	高
real-time	单一，功能极少	客户端实时响应	json/xml	极高

我们看Linkedin和Iron.io的案例，都是非常典型的Web Service的应用场景：Linkedin使用Rails开发了移动服务器的API网关，而Iron.io用Rails开发了搜集客户设备数据的后台服务，这些都不是Rails最擅长的开发website的场景，所以最终Rails被抛弃，并不是一个很意外的结果。

Rails为何不适合做Web Service?

TomLan

了解Rails魔法是怎麼練成的

第一隻app

```
# myapp.rb
require 'sinatra'
get '/' do
  'Hello world!'
end
```

return值即為response body

Routing Basic

- HTTP method + URL expression然後 block
- 允許同樣URL expression & http method的 block
- 返回值會有
  1. HTTP Status Code(可省略)
  2. Headers (可省略)

Named Route Params

```
get '/rooms/:id/index.html' do
  # matches '/rooms/123/
index.html?width=500&height=300'
  params[:id] # => 123
  params[:width]# => 500
  params[:height]# => 300
end
```

Splat Param Route

```
  get '/books/*.*' do
  # matches /books/ruby-guide.html
  params["splat"]# => ["ruby-guide", "html"]
end
```

RegExp

```ruby
  get %r{/posts/name-([\w]+)} do
  # => get /posts/name-abc, params[:captures][0] = 'abc'
  "Hello, #{params[:captures].first}!"
end

get %r{/posts/([\w]+)} do |pid|
  # => put match content to block param(s)
  # => matches 「([\w]+)」 to 「pid」
end
```

TomLan

Routing Block Variables

```
  get '/thread/:tid' do |tid|
  # => tid == params[:tid]
end

get '/pictures/*.*' do |filename, ext|
  # => GET '/pictures/abc.gif' then filename = "abc" and
ext = "gif"
  "filename = #{filename}, ext = #{ext}"
end

get %r{/posts/([\w]+)} do |pid|
  # => put match content to block param(s)
  # => matches 「([\w]+)」 to 「pid」
end
```

Conditional route

```
  get '/foo', :agent => /MSIE\s(\d.\d+)/ do
  "You're using IE version #{params[:agent][0]}"
  # => IE特製版
end
get '/', :host_name => /^admin\./ do
  "Admin Area, Access denied!"
end
```

params hash

- params hash可能包含

1. url query string
2. form post body
3. Named Route Params

ERB Template

- Code Block的最返回erb 樣板名稱
- 樣板名稱必需是symbol
- 樣板檔名是erb結尾
- 預設在views目錄下

```
get '/' do
  @page_title  = "輸入程式碼!!"
  erb :index  #用index.erb輸出
end
```

ERB樣板語言

- <%%>間為執行程式
- <%=%>間除執行程式外，會把結果直接layout在頁面上

inline Template

```
   require 'sinatra'
get '/' do
  erb :index
end
__END__
@@index
<!DOCTYPE html>
<html>
<head>
   <title>Sinatra app</title>
</head>
<body>
  <% 10.times do %>
    <p><%= "Hello #{params[:name]}!" %></p>
  <% end %>
</body>
</html>
```

partial view

```
   ....
__END__
@@index
<!DOCTYPE html>
<html>
<head>
   <title>Sinatra app</title>
</head>
<body>
  <% 10.times do %><%=erb :name, layout: false%><% end %>
</body>
</html>
@@name
<p><%= "Hello #{params[:name]}!" %></p>
```

View Layout

- 樣板間共用的部份
- HTML的上下文
- 預設layout.erb
- erb命令時可用:layout => :layout 做設定

Layout Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Backend</title>
  <%= stylesheet_link_tag    "application"%>
  <%= javascript_include_tag "application"%>
  <%= csrf_meta_tags %>
</head>
<body>
<%= yield %> <!-/* yield代表view要render的部份*/->
</body>
</html>
```

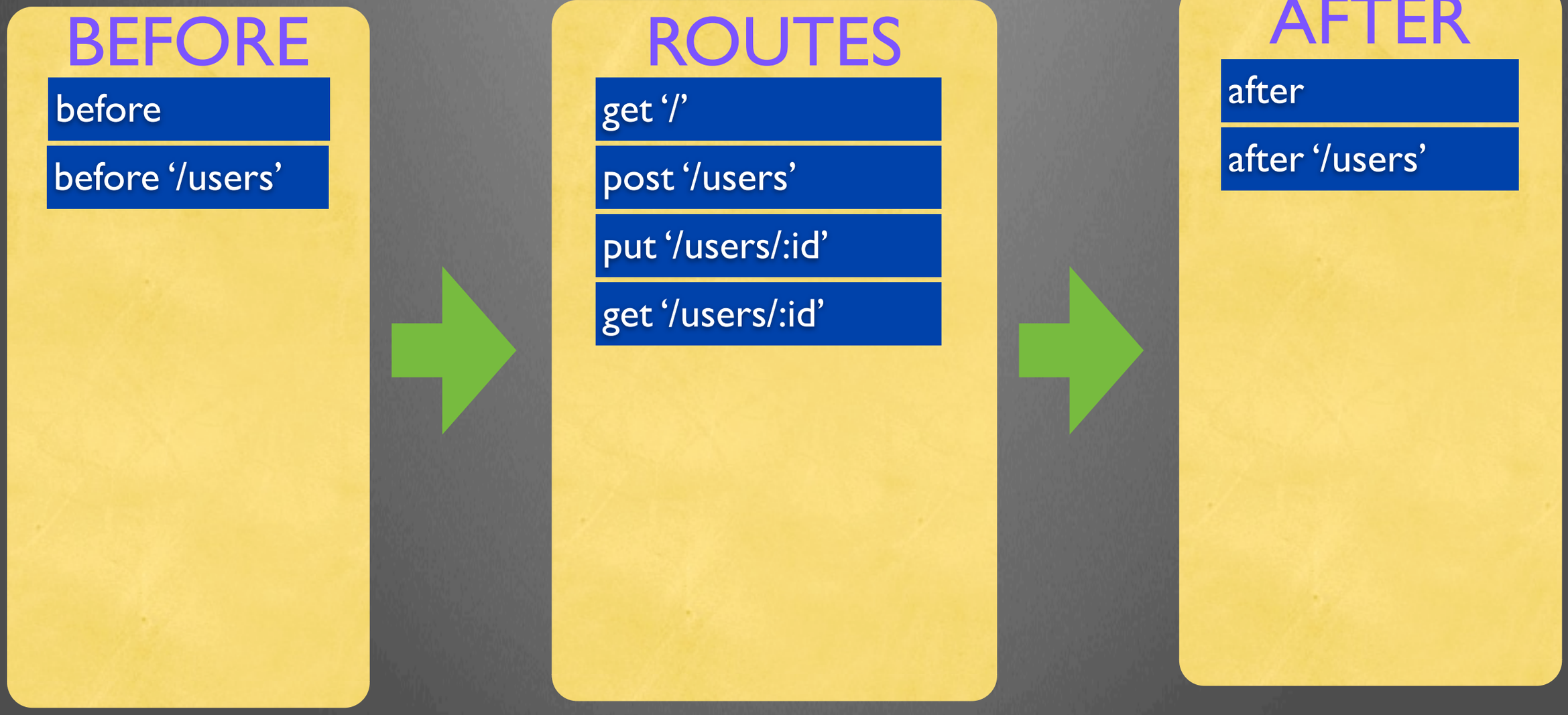
Helpers

```
 helpers do
def current_member
  @member ||= Member.find(session[:sid])
end
def div_for(content, klass)
  "<div class='#{klass}>#{content}</div>"
  end
end
get '/members/:id.json' do
  #如果@member為空則返回nil
  return nil unless current_member
  erb :show
end
__END__
@@show
<%=div_for current_member.name, 'member'%>
#輸出一個div tag
```

Helper可用在

- filters
- routes
- templates

流程控制


BEFORE
before
before '/users'
ROUTES
get '/'
post '/users'
put '/users/:id'
get '/users/:id'
AFTER
after
after '/users'

一個request的一生

- before filter(maybe)
- routing區塊處理
- render layout & view
- after filter(maybe)
- response to client

before & after filter

- URL expression和route相同
- 實體變數可互通
- 從URL expression中取得之變數不互通
- 一次可match多個filter
- 順序為在code中的順序

halt

```
  helpers do
  def current_member
    @member ||= Member.find(session[:sid])
  end
end
before do #如果沒有登入就在這裡終止
  halt 401 unless current_member
end
get '/membercp' do
  erb :show
end
```

pass

```
   require 'sinatra'
get '/' do
  pass
  erb :index
end
get '/' do
  '學rails也可以很謙虛'
end
```

redirect

```
redirect '/do-it' #重導至 /do-it
```

request object

```
   # 在 http://example.com/example 上運行的應用
get '/foo' do
  request.body                # request body
  request.scheme              # "http"
  request.script_name         # "/example"
  request.path_info           # "/foo"
  request.port                # 80
  request.request_method      # "GET"
  request.query_string        # "" 查詢參數
  request.content_length      # request.body的長度
  request.media_type          # request.body的媒體類型
end
```

session

```
   enable :sessions
post '/sessions' do
  @current_user = User.auth(params[:account],
params[:passwd])
  session[:uid] = @current_user.id if @current_user
end
delete '/sessions' do
  session[:uid] = nil
  redirect '/'
end
```

cookies

```
  before do
  request.cookies['xd'] = 'Lorem ipsum'
end

get '/' do
  request.cookies.inspect
end
```

http methods(verbs)

- get -> read
- post -> create
- put/patch -> update
- delete -> delete

method override

```
<div class="navbar navbar-fixed-top">
<div class="container-fluid admin_yield">
  <form id="edit_member_51574" class="simple_form
  edit_member" novalidate="novalidate" method="post" action="/admin/members
  /51574" accept-charset="UTF-8">
    <div style="margin:0;padding:0;display:inline">
      <input type="hidden" value="√" name="utf8">
      <input type="hidden" value="put" name="_method">
      <input type="hidden" value="Nf3xo5+h/TE1N8qZvZ6fTDV2caeKiTyPbbbVIHK32
      mo=" name="authenticity_token">
    </div>
```

TomLan

Restful Style

http method on 動作+類別名稱(單or複數)

GET /viewthread.php?tid=356875

v.s.

GET /threads/356875

POST /topicadmin.php?tid=12345&action=delete

v.s.

DELETE /topic/12345

URI	單/複	ACTION	METHOD	用途
/posts	複	x	GET	Get all posts
/posts	複	x	POST	Create New Post
/posts/:id	單	x	GET	Get one post
/posts/:id	單	x	DELETE	Delete one
/posts/:id	單	x	PUT	Update one Post
/posts/:id/edit	單	edit	GET	Get edit form
/posts/new	複	new	GET	Get New Form

組合方式:

/物件名(複數)/id(單數才有)/action(可能沒有)

好處

- 不用想網址
- WEB幾乎所有行為都是CRUD
- 物件模式/操作和網址有同樣pattern

TomLan

Live Coding:

偽CSV資料庫操作

Q&A