Rack Inside

James Zhan

The Problem?

- *each framework has to write it's own handlers to mongrel, webrick, fast-cgi, etc...*
- *duplication among frameworks*
- *new frameworks are required to write even more duplicate code, boring!*

A solution - Rack

- *super simple API for writing web apps.*
- *single API to connect to mongrel, fast-cgi, webrick...*
- *base on Python's WSGI.*

Rack Interface

- *an object that responds to call and accepts one argument: env, and returns:*
- *a status, i.e. 200*
- *the headers, i.e. {'Content-Type' => 'text/html'}*
- *an object that responds to each, i.e. 'some random string'*

Rack Spec

- *The Environment*

  - The environment must be an instance of Hash that includes CGI-like headers. The application is free to modify the environment.

  - The environment is required to include these variables (adopted from PEP333), except when they'd be empty.

  - Rack-specific variables

  - The server or the application can store their own data in the environment, too. The keys must contain at least one dot, and should be prefixed uniquely. The prefix `rack.` is reserved for use with the Rack core distribution and other accepted specifications and must not be used otherwise. The environment must not contain the keys `HTTP_CONTENT_TYPE` or `HTTP_CONTENT_LENGTH` (use the versions without `HTTP_`). The CGI keys (named without a period) must have String values. There are the following restrictions:

Concepts in Web Server

- *Server*
- *Service*
- *Application*
- *Connection*
- *Connector*

HTTP Request

- *<request-line>*
- *<headers>*
- *<blank line>*
- *[<request-body>]*

Request Line

- *<HTTP-METHOD> <Resource Path> <HTTP-Protocol>*
- *Example:* GET /index.html HTT1.1
- *GET, POST, PUT, DELETE, HEAD, TRACE, OPTIONS, CONNECT*

Request Headers

```
POST / HTTP/1.1
Host: www.wrox.com
User-Agent: Mozilla/5.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 40
Connection: Keep-Alive

name=Professional%20Ajax&publisher=Wiley
```

Blank Line

```
POST / HTTP/1.1
Host: www.wrox.com
User-Agent: Mozilla/5.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 40
Connection: Keep-Alive
<\r\n>
name=Professional%20Ajax&publisher=Wiley
```

Request Body

```
POST / HTTP/1.1
Host: www.wrox.com
User-Agent: Mozilla/5.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 40
Connection: Keep-Alive

name=Professional%20Ajax&publisher=Wiley
```

HTTP Response

- *<status-line>*
- *<headers>*
- *<blank line>*
- *[<response-body>]*

Status Line

- *<http protocol> <status code> <status description>*
- *Examples:*HTTP/1.1 200 OK
- *We will details talk about status code mean.*

Response Headers

```
HTTP/1.1 200 OK
Date: Sat, 31 Dec 2005 23:59:59 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 122

<html>
<head>
<title>Wrox Homepage</title>
</head>
<body>
<!-- body goes here -->
</body>
</html>
```

Response Body

```
HTTP/1.1 200 OK
Date: Sat, 31 Dec 2005 23:59:59 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 122

<html>
<head>
<title>Wrox Homepage</title>
</head>
<body>
<!-- body goes here -->
</body>
</html>
```

HTTP Status Code

状态行包含HTTP版本、状态代码、与状态代码对应的简短说明信息。在大多数情况下，除了Content-Type之外的所有应答头都是可选的。但Content-Type是必需的，它描述的是后面文档的MIME类型。虽然大多数应答都包含一个文档，但也有一些不包含，例如对HEAD请求的应答永远不会附带文档。有许多状态代码实际上用来标识一次失败的请求，这些应答也不包含文档（或只包含一个简短的错误信息说明）。

1xx - 信息提示
2xx - 成功
3xx - 重定向
4xx - 客户端错误
5xx - 服务器错误

HTTP Status Code

1xx - 信息提示
这些状态代码表示临时的响应。客户端在收到常规响应之前，应准备接收一个或多个 1xx 响应。
• 100 - 继续。
• 101 - 切换协议。

2xx - 成功
这类状态代码表明服务器成功地接受了客户端请求。
• 200 - 确定。客户端请求已成功。
• 201 - 已创建。
• 202 - 已接受。
• 203 - 非权威性信息。
• 204 - 无内容。
• 205 - 重置内容。
• 206 - 部分内容。

3xx - 重定向
客户端浏览器必须采取更多操作来实现请求。例如，浏览器可能不得不请求服务器上的不同的页面，或通过代理服务器重复该请求。
• 302 - 对象已移动。
• 304 - 未修改。
• 307 - 临时重定向。

HTTP Status Code

4xx - 客户端错误

发生错误，客户端似乎有问题。例如，客户端请求不存在的页面，客户端未提供有效的身份验证信息。

- 400 - 错误的请求。
- 401 - 访问被拒绝。IIS 定义了许多不同的 401 错误，它们指明更为具体的错误原因。这些具体的错误代码在浏览器中显示，但不在 IIS 日志中显示：
  - 401.1 - 登录失败。
  - 401.2 - 服务器配置导致登录失败。
  - 401.3 - 由于 ACL 对资源的限制而未获得授权。
  - 401.4 - 筛选器授权失败。
  - 401.5 - ISAPI/ 应用程序授权失败。
  - 401.7 – 访问被 Web 服务器上的 URL 授权策略拒绝。这个错误代码为 IIS 6.0 所专用。
- 403 - 禁止访问：Web Server可以定义了许多不同的 403 错误，它们指明更为具体的错误原因：
  - 403.1 - 执行访问被禁止。
  - 403.2 - 读访问被禁止。
  - 403.3 - 写访问被禁止。
  - 403.4 - 要求 SSL。
  - 403.5 - 要求 SSL 128。
  - 403.6 - IP 地址被拒绝。
  - 403.7 - 要求客户端证书。
  - 403.8 - 站点访问被拒绝。
  - 403.9 - 用户数过多。
  - 403.10 - 配置无效。
  - 403.11 - 密码更改。
  - 403.12 - 拒绝访问映射表。
  - 403.13 - 客户端证书被吊销。
  - 403.14 - 拒绝目录列表。
  - 403.15 - 超出客户端访问许可。
  - 403.16 - 客户端证书不受信任或无效。
  - 403.17 - 客户端证书已过期或尚未生效。

HTTP Status Code

- 404 - 未找到。
  - 404.0 -（无） – 没有找到文件或目录。
  - 404.1 - 无法在所请求的端口上访问 Web 站点。
  - 404.2 - Web 服务扩展锁定策略阻止本请求。
  - 404.3 - MIME 映射策略阻止本请求。
- 405 - 用来访问本页面的 HTTP 谓词不被允许（方法不被允许）
- 406 - 客户端浏览器不接受所请求页面的 MIME 类型。
- 407 - 要求进行代理身份验证。
- 412 - 前提条件失败。
- 413 – 请求实体太大。
- 414 - 请求 URI 太长。
- 415 – 不支持的媒体类型。
- 416 – 所请求的范围无法满足。
- 417 – 执行失败。
- 423 – 锁定的错误。

HTTP Status Code

5xx - 服务器错误
服务器由于遇到错误而不能完成该请求。
- 500 - 内部服务器错误。
  - 500.12 - 应用程序正忙于在 Web 服务器上重新启动。
  - 500.13 - Web 服务器太忙。
  - 500.15 - 不允许直接请求 Global.asa。
  - 500.16 – UNC 授权凭据不正确。这个错误代码为 IIS 6.0 所专用。
  - 500.18 – URL 授权存储不能打开。这个错误代码为 IIS 6.0 所专用。
  - 500.100 - 内部 APP 错误。
- 501 - 页眉值指定了未实现的配置。
- 502 - Web 服务器用作网关或代理服务器时收到了无效响应。
  - 502.1 - CGI 应用程序超时。
  - 502.2 - CGI 应用程序出错。
- 503 - 服务不可用。这个错误代码为 IIS 6.0 所专用。
- 504 - 网关超时。
- 505 - HTTP 版本不受支持。

Rack Middleware

- *Rack middleware is a way to filter a request and response coming into your application*

- *Rack::Static, Rack::CommonLogger, Rack::ShowExceptions, Rack::Lint, Rack::Config, Rack::Cascade, Rack::Lock, Rack::Head, Rack::Reloader, Rack::Runtime, Rack::ContentLength, Rack::Deflater*

Quick Demo

```
require 'rack'

Rack::Server.start(
    :app => lambda do |env|
        [200, {'Content-Type' => 'text/html'}, ['hello world']]
    end,
    :server => 'thin'
)
```

Quick Demo

```
require 'rack'
require 'rack/lobster'
app = Rack::Builder.app do
      use Rack::CommonLogger
      use Rack::ShowExceptions
      map "/lobster" do
            use Rack::Lint
            run Rack::Lobster.new
      end
end

Rack::Handler::Thin.run(app, :Port => 3000)
```

Use Rack DSL

```
#config.ru
require 'rack/lobster'
require 'sinatra'

use Rack::ShowExceptions
use Rack::Auth::Basic, "Lobster 2.0" do |username, password|
  'secret' == password
end
run Rack::Lobster.new

#rackup
#rackup config.ru
```

Rack Prototype

```
module Rack
    class Builder
        def initialize()
                @run, @middlewares = [], nil
        end
        def use(middleware, *args, &block)
                @middlewares << lambda{|app| middleware.new(app, *args, &block)}
        end
        def run(app)
                @run = app
        end
        def to_app
                app = @run
                @middlewares.reverse.reduce{|app, middleware|
                        middleware[app]
                }
        end
        def call(env)
              to_app.call(env)
        end
    end
end
```

rackup

```
#!/usr/bin/env ruby

require "rack"
Rack::Server.start
```