HTTP Adapers

Polyglot Programming DC 2015

Brock Wilcox
awwaiid@thelackthereof.org

Web Applications

Request → Response

Let's go BACK....

to 1993

CGI

Common Gateway Interface

Request → Response

... Unix!

Request → Response

ENV + STDIN → STDOUT

ENV == Headers (params)
STDIN == POST
STDOUT == Headers + Body

```
#!/bin/sh

# A lovely cgi application

echo "Content-type: text/html"
echo
echo "Hello, world! Query:"
echo $QUERY_STRING
```

Parse the query string if you want

cgi programs
rise from ash then burn again
upon each request

Long running applications

Kinda long-running

mod_perl
mod_php
mod_python
...

Kinda long-running

FastCGI

Just run your own HTTPD

Also annoying

Same webserver language,
but unlink from webserver.

Web App Frameworks

```
func app(request) → response

response = [ status, headers, body ]
```

Common Lisp - Clack
Clojure - Ring
Elixir - Plug
Haskell - Hack2
Java - Servlets?
Javascript - JSGI
Lua - WSAPI
Perl - PSGI
Perl6 - P6SGI
Python - WSGI
Ruby - Rack
Scala - SSGI
* - uWSGI

```python
# Python WSGI

def application(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/plain')])
    yield 'Hello, world!'
```

```ruby
# Ruby Rack

app = lambda do |env|
    [200, {"Content-Type" => "text/plain"}, ["Hello, world!"]]
end
```

```perl
# Perl PSGI

my $app = sub {
    my ($env) = @_;
    return [200, ['Content-Type' => 'text/plain'], ["Hello, world!"]];
}
```

```
// Javascript JSGI

require("jsgi-node").start(function(request){
  return {
    status:200,
    headers:{},
    body:["Hello, world!"]
  };
});
```

```elixir
# Elixir Plug

defmodule MyPlug do
  import Plug.Conn

  def init(options) do
    # initialize options
    options
  end

  def call(conn, _opts) do
    conn
    |> put_resp_content_type("text/plain")
    |> send_resp(200, "Hello, world!")
  end
end
```

```clojure
; Clojure Ring

(ns hello-world.core)

(defn handler [request]
  {:status 200
   :headers {"Content-Type" "text/html"}
   :body "Hello World"})
```

```
# Perl6 P6SGI

sub app(%env) {
    start {
        200, [ Content-Type => 'text/plain' ], [ 'Hello World!' ]
    }
}
```

```
// Scala SSGI

package org.scalatra.ssgi
package examples.servlet

import scala.xml.NodeSeq

class HelloWorldApp extends Application {
  def apply(v1: Request) = Response(body = <h1>Hello, world!</h1>)
}
```

Lua WSAPI

```lua
function hello(wsapi_env)
  local headers = { ["Content-type"] = "text/html" }

  local function hello_text()
    coroutine.yield("Hello, world!")
  end

  return 200, headers, coroutine.wrap(hello_text)
end
```

```java
// Java Servlet

// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

  private String message;

  public void init() throws ServletException
  {
      // Do required initialization
      message = "Hello, world!";
  }

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // Set response content type
      response.setContentType("text/html");

      // Actual logic goes here.
      PrintWriter out = response.getWriter();
      out.println("<h1>" + message + "</h1>");
  }

  public void destroy()
  {
      // do nothing.
  }
}
```

```haskell
{-# Haskell #-}

{-# LANGUAGE OverloadedStrings #-}

import Hack2
import Hack2.Contrib.Response (set_body_bytestring)
import Hack2.Handler.SnapServer

app :: Application
app = env ->
  return $
    Response
      200 [ ("Content-Type", "text/plain") ] "Hello, world!"

main = run app
```

Streaming

response = [status, headers, callback]

Keep invoking callback until done

Streaming

response = [status, headers, promise]

Wait for promise results, maybe chained

Streaming / Websockets ... not unified

Middleware

app: request → response

middleware: app → (request → response)

```ruby
def middlin(app)
  lambda do |request|
    # ... mess with request
    app_response = app(request)
    # ... mess with app_response
    app_response
  end
end
```

```ruby
def log_middleware(app)
  lambda do |request|
    File.open("middleware.log", "w+") do |f|
      f.puts request
    end
    app_response = app(request)
    app_response
  end
end

app = lambda do |env|
    [200, {"Content-Type" => "text/plain"}, ["Hello, world!"]]
end

new_app = log_middleware(app)
```

Middleware Examples

Logging
Authenticating
Proxy
Multi-app routing
Session tracking
Debugging
Content filtering
Content translation**
Content injection
Static serving

THE END