rack

rolling your own, tiny like, web thingoes

presentation by ryan allen - yeahnah.org (and etc)

the problem?

- each framework has to write it’s own handlers to mongrel, webrick, fast-cgi
- duplication among frameworks, lachie kills kittens
- 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
- based on python's WSGI, which was a good thing for python

the api

- can be satisfied with a lambda
- really, no, i’m serious

```
lambda { |env| [200, {}, ‘Hello World!’ }
```

the api, cont.

- 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’`

this `env` thing

- a hash of environment variables, things like `PATH_INFO`, `QUERY_STRING`, `REMOTE_ADDR`, `REQUEST_URI`, all that junk

things do to with `env`

- use `Rack::Request` to construct a request object, pull out GET and POST data
- Rack borrows code from Camping, Rails and IOWA in this department

```ruby
request = Rack::Request.new(env)
request.GET
request.POST
```

talking back, eloquently

- use `Rack::Response` to talk to clients about cookies
- additionally, you can use this instead of the `[200, {}, ‘’]` return value, see:

```ruby
response = Rack::Response.new('Hello World!')
response.set_cookie('sess-id', 'abcde')
response.finish
```

lots of love to mongrel

- we love mongrel to bits
- despite the fact we can plug into webrick and fast-cgi we really just love mongrel
- poor webrick and fast-cgi :(

```
%w(rubygems rack).each { |dep| require dep }
app = lambda { |env| [200, {}, 'Hello World!'] }
Rack::Handler::Mongrel.run(app, :Port => 3000)
```

oh, and that was a complete rack app

run it, it works, really!

```ruby
%w(rubygems rack).each { |dep| require dep }
app = lambda { |env| [200, {}, 'Hello World!'] }
Rack::Handler::Mongrel.run(app, :Port => 3000)
```

concurrency galore

- using the mongrel handler we can get crazy concurrency
- let’s have a look at this, using `ab`

```
%w(rubygems rack).each { |dep| require dep }
app = lambda { |env| sleep(5); [200, {}, 'Oi!'] }
Rack::Handler::Mongrel.run(app, :Port => 3000)
```

```
ab -c 100 -n 300 http://0.0.0.0:3000/
```

```
Benchmarking 0.0.0.0 (be patient)
Completed 100 requests
Completed 200 requests
Finished 300 requests
Concurrency Level:       100
Time taken for tests:    15.232 seconds
Complete requests:       300
Failed requests:         0
```

so?

- good like camping - very minimal, single file applications
- but better because camping isn’t multi-threaded, requests are wrapped in a mutex like rails, boo!

so??

- good like merb - multithreaded, non-blocking requests
- but better (in some scenarios) beacuse merb is a big framework mirroring a lot of what rails does

so???

- people use merb to write non-blocking uploader applications to compliment rails applications
- but, why aren't they using rack instead?
- rack is smaller, more lightweight and super easy to write and deploy!

time to break stuff

- question - ok, so how does `ActiveRecord` operate under concurrency?
- answer - creates a connection-per-thread when `allow_concurrency` is `true`
- exhaust `max-connections`, anyone?
- lets try!

```
%w(rubygems rack active_record).each { |dep| require dep }
AR = ActiveRecord::Base
AR.allow_concurrency = true
AR.establish_connection(
  :adapter  => 'mysql',
  :username => 'root',
  :database => 'flashden'
)
def user_count
  AR.connection.select_all('select * from users limit 1000')
end
app = lambda { |env| [200, {}, user_count.inspect] }
Rack::Handler::Mongrel.run(app, :Port => 3000)
```

```
ab -c 100 -n 100 http://0.0.0.0:3000/
```

```
Thu Oct 25 17:01:09 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:09 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:09 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:10 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:10 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:10 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:10 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:10 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:10 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:10 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:10 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:10 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:10 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:11 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:11 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:11 +1000 2007: ERROR: Too many connections
Thu Oct 25 17:01:11 +1000 2007: ERROR: Too many connections
```

yep, broke it (that's on max-connections=100) i dunno how to fix it

connection pooling?

yeah thanks for your feedback!

adding-on

- use `Rack::Builder` to add some more sophisticated behaviour, it's a DSL like thing to construct rack applications
- free logging, exception handling, url mapping, http authentication, directory serving...

```
app = Rack::Builder.new {
  use Rack::CommonLogger
  use Rack::ShowExceptions
  map "/" do
    use Rack::Lint
    run MyCustomAwesomeApp.new
  end
}
```

so, these bits and bobs can make life easier

- Rack::Reloader
- Rack::Static
- Rack::ShowExceptions
- Rack::CommonLogger
- There's a couple more, see docs!

links to stuff and etc

- http://rack.rubyforge.org/
- http://rack.rubyforge.org/doc/
- http://www.wsgi.org/

end!