Resource-Oriented Web Apps

Or: *When Functional Programming Meets HTTP*

AutoCode • Functor • Filebase • Waves • Pages
JavaScript: DateJS • Behavioral Stylesheets

Dan Yoder

Director, R&D Solutions
dan@zeraweb.com

We're so totally hiring.

Waves::Vitals

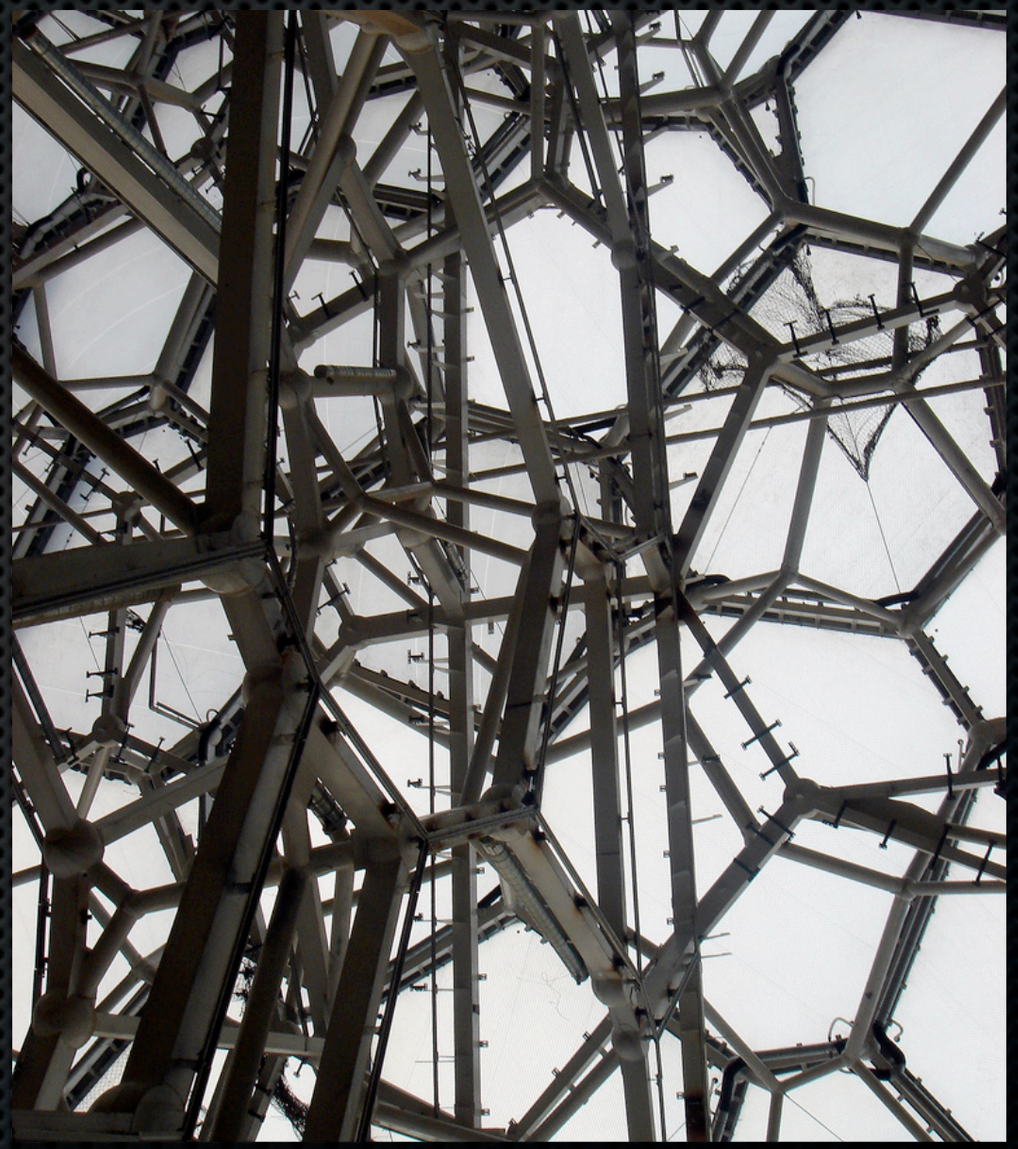
- Initial release Feb 08
- ~~Release 0.8.0 Available Now!~~
  Release 0.8.0 Available **October 6**
- Of course, it is on **git**.
  http://github.com/dyoder/waves
- Also, we have a Google Group.
  http://groups.google.com/group/rubywaves
- And, lest we forget, a Web site.
  http://rubywaves.com/

Waves::Features

- Resource-oriented: hello, HTTP?!
- Heavy functional programming influence
- Layered architecture: not very opinionated
- Transparent approach: magic, not mystery
- Modular, designed to encourage reuse
- Emphasis on Ruby, not the framework
- Intelligent class and module loading
- Meta-framework, designed for hacking

Why Use Waves?

Because you want more control over your apps and love REST, FP, and Ruby.



Enough preliminaries.

What Is HTTP, Really?

A protocol that calls a method on a resource.

That sounds like Ruby to me!

```
class Resource

  attr_accessor :request

  def initialize( request )
    @request = request
  end

  def get
    # ...
  end

  def put
    # ...
  end

  def post
    # ...
  end

  def delete
    # ...
  end

end
```

HTTP Requests: Method Signatures?

Example

get(*URI*: Content-Type, *Params*: Query-String) : Accept

But We Don't Want This ...

```
def get
  case request.path
  when %r{^/$}
    ...
  when %r{^/stories$}
    ...
  when $r{^/story/}
    ...
  default
    { :status => 404, :body => “Sorry, charlie!” }
  end
end
```

Functor To The Rescue

Functor gives us argument pattern matching in Ruby.

A Quick Intro To Functor

```
fib = Functor.new do
  given( 0 ) { 1 }
  given( 1 ) { 1 }
  given( Integer ) do |n|
    self.call( n - 1 ) + self.call( n - 2 )
  end
end

fib.call( 7 ) # => 21
```

Functor Methods

```
class Story < Resource::Base

  include Functor::Method

  functor( :get, %r{ ^/$ } ) do
    ...
  end
  functor( :get, %r{ ^/stories$ } ) do
    ...
  end
  functor( :get, $r{ ^/story/ } ) do
    ...
  end

  ...

end
```

Functor Methods

```
class Story < Resource:Base

  include Functor::Method

  functor( :get, %r{ ^/$ } ) do
    ...
  end
  functor( :get, %r{ ^/stories$ } ) do
    ...
  end
  functor( :get, $r{ ^/story/ } ) do
    ...
  end

  ...

end
```

Bad!

Sadly, It's Not That Simple

HTTP Requests - er, method signatures, are more than just pretty URIs.

Fortunately, Functors Take Lambdas As Arguments

```
# Evil smile ...

match_feed = lambda do | request |
  request.path.match(%r{^/blog/(\w+)\.rss$}) and
    request.accept.include?( :rss )
end

functor( :get, match_feed ) do |request|
  ...
end
```

Hmm .... (cue Waves music)

Waves Makes This Easy

- Simple DSL For Declaring Complex Constraints
- Resource “mounts” to support multiple resources within an application
- Layer-based Helpers to make it easy to write request methods

The One File Waves App

```
require 'rubygems'
require 'waves'

module BoogieBoard

  include Waves::Foundations::Simple

  module Configurations
    class Production < Waves::Configurations::Default
      host '0.0.0.0' ; port 3000 ; debug false ; synchronize? false
      handler ::Rack::Handler::Mongrel, :Host => host, :Port => port
      resources { mount :hello }
      application { run ::Waves::Dispatchers::Default.new }
    end
  end

  module Resources
    class Hello < Waves::Resources::Base
      on( :get, [] ) { "Hello #{query.name}"  }
    end
  end

end

Waves << BoogieBoard
```

```ruby
require 'rubygems'
require 'waves'

module BoogieBoard

  include Waves::Foundations::Simple

  module Configurations
    class Production < Waves::Configurations::Default
      host '0.0.0.0' ; port 3000 ; debug false ; synchronize? false
      handler ::Rack::Handler::Mongrel, :Host => host, :Port => port
      resources { mount :hello }
      application { run ::Waves::Dispatchers::Default.new }
    end
  end

  module Resources
    class Hello < Waves::Resources::Base
      on( :get, [] ) { "Hello #{query.name}"  }
    end
  end

end

Waves << BoogieBoard
```

Match Components

```
on( :get, [ ‘admin’ ] )
```

Wildcard

```
on( :get, [ ‘image’, true ] )
```

Capture Component

```
on( :post, [ :name, ‘edit’ ] )
```

Extract Wildcard

```
on( :get, [ ‘image’,
  { :path => true } ] )
```

Default Component

```
on( :get, [ { :name => ‘home’ } ] )
```

Match Query

```
on( :get, [ ‘location’ ],
  :query => { :lat => /\d{4}/,
    :long => /\d{4}/ } )
```

Path With Accept

```
on( :get, [ ‘blog’, :name ],
  :accept => :rss )
```

... and much much more!

Here’s a real example.

```
module Pages
  module Resources
    class Default
      include Waves::Resources::Mixin

      with( :visitor ) do
        on( :get, :show => [{ :name => 'home' }] ) do
          view.show( singular => controller.find( query.name ) )
        end
      end

      with( :author ) do
        before do
          redirect( paths( :site ).login ) unless session[:user]
        end
        on( :post, :update =>  [ :name ] ) do
          controller.update( query.name ) and redirect( paths( :site ).admin )
        end
        on( :get, :edit => [ :name ] ) do
          view.editor( singular => controller.find( query.name ) )
        end
        on( :put, :add =>  [ :name ] ) do
          controller.create and redirect( paths.show )
        end
        on( :delete, :delete =>  [ :name ] ) do
          controller.delete( query.name ) and redirect( paths( :site ).main )
        end
      end

    end
  end
end
```

Look ... It's Just Ruby

```
pages $ waves-console

Pages::Resources::Story.instance_methods &
  %w( get put post delete )

# => ["delete", "post", "put", "get"]
```

Some Implications

Or: *Why Are Resource Classes Better Than Routes?*

Resource Classes

- **Inheritance.** I can now just inherit from a base class to provide common REST features.
- **Relative paths.** I can specify relative paths that will automatically change if the mount point changes.
- **Modularity.** Request-handling is broken down into logically discrete chunks, just like normal classes.

Mount Points

- **Reuse.** I can reuse Resources from other applications by simply giving them distinct mount points.

```
mount Blog::Resources::Entry, [ ‘blog’, { :rest => true } ]
```

- **Performance.** I don’t have to go through each possible match - just those for the matching resource.

R = 5, M = 5
R x M = 25 methods
R + M = 10 patterns
60% improvement

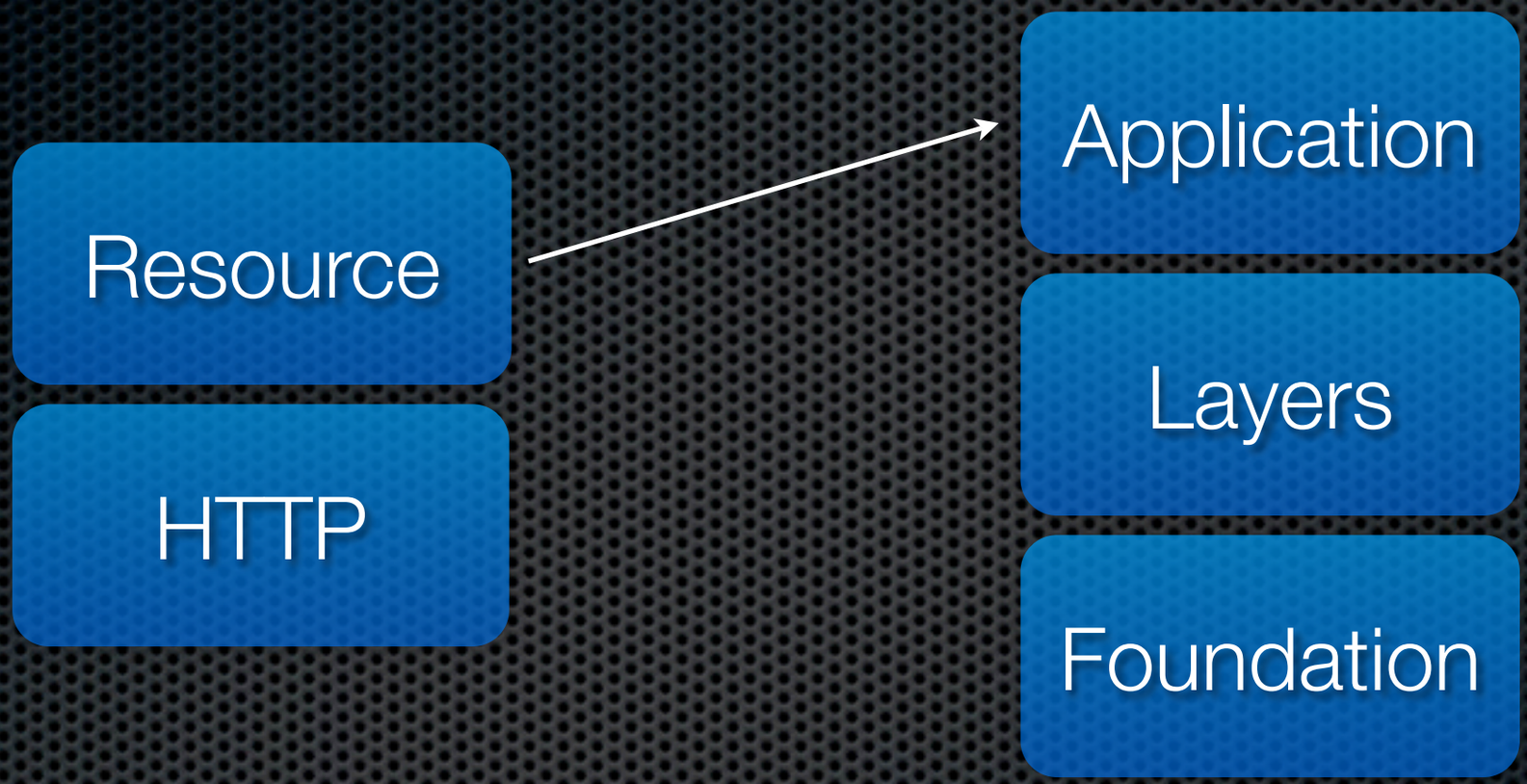
Mount Points

```
module Pages
  module Configurations
    class Default < Waves::Configurations::Default

      resources do

        mount :image, :accept => :image
        mount :media, :accept => [ :css, :javascript ]
        mount :blog, :accept => :rss

        mount true, [ 'admin', :resource, { :rest => true }], :as => :author
        mount :site, [ 'admin' ], :as => :author

        mount true, [ :resource, { :rest => true }], :as => :visitor
        mount :story, :as => :visitor


      end

    end
  end
end
```

A Word About Architecture

Waves Supports Layered Architecture Through **Foundations** And **Layers**.

Resources decouple your application from the HTTP request processing.



Layers mix-in features (like MVC support) to help build your application.

A Foundation is a Layer that provides at least the minimal features required for a Waves application.

The Default Foundation

```
include Waves::Layers::Inflect::English
include Waves::Layers::Simple
include Waves::Layers::MVC
include Waves::Layers::DefaultError
```

But you can define your own ...

Roadmap

TODO: this is a hack ...

In No Particular Order ...

- Update documentation and tutorials
- Caching API
- Improved sample apps: Blog, Wiki, CMS
- Optimize functor using state-machine approach
- “Certified” JRuby version
- Improved Accept header handling
- Integrate LiveConsole, Cassandra, and Erector
- Passenger Handler
- RIA and SOA Layers

Waves Needs You!

This stuff is hard. We get confused and get headaches. Please help!

Thanks!

Photo credits: All photos from Flickr.com under Creative Commons License.
http://www.flickr.com/photos/neilharvey/343068501/sizes/l/
http://www.flickr.com/photos/baldiri/496892841/sizes/o/
http://www.flickr.com/photos/walkingthedeepfield/2281168070/sizes/l/
http://www.flickr.com/photos/kanaka/1798470406/sizes/l/
http://www.flickr.com/photos/fuseman/262082281/sizes/o/
http://www.flickr.com/photos/xiaming/484440654/sizes/l/
http://www.flickr.com/photos/smithco/118770617/sizes/l/
http://www.flickr.com/photos/brainfarts/97676505/sizes/l/
http://www.flickr.com/photos/xfp/2830064227/sizes/o/