Ruby Performance Tuning

RUBY

JAX+

Thank You Ascentis!

Prequel

- gem install ruby-prof
- brew install qcachegrind
- brew install graphviz

New Co-Location

- LionShare Cowork
- https://www.lionsharecowork.com/
- Contact: Sherrie Walker

WARNING

"Worry less about who you might offend and care more about who you might inspire."

–Mike Baxter

Goal - Demonstrate Solutions to Improve Ruby Performance

- Convert views from HAML to ERB (or Slim)
- Examine the data structures we are using
- Optimize our views by using Constants (UIStr)
- See how Ruby Prof can help us track down issues

One of the challenges we may encounter as an application grows in size and complexity is its' performance.

- NO, not this type of Performance Issue

- And not this type of Performance Issue

- A common area of performance issues can be with the database. In addition to latency issues, one might run into N+1 issues.
- Sometimes we need to look beyond the database. Especially if your app does not have a traditional database.

Not a Computer Scientist

- Not a Computer Scientist - at least not by college education
- Bachelor of Arts - Economics
- Just In Time Learner

What Is a JITL?

Simply put, one learns what they need to at the time they need the knowledge.

While working on the Auction Web Application, being tossed into the mix of existing code and fast deadlines, my immediate focus was on creating solutions for features and fixing bugs.

I was not thinking about performance, especially since I didn't notice any performance issues for the bits and pieces I was working on at the time.

It wasn't until we really started heavy load testing with our QA department that we noticed some performance issues. So off to the Internet for some Just In Time Learning!

Review the Views

As Web Developers, where would we most likely performance issues? Where would we notice the web application being sluggish?

If you said the Views, you are correct!

HAML.is_a?(HOG)

When I first started learning Rails, I only knew of ERB. But as I attended Meetups, Conference and some Code-n-Coffee sessions, I was introduced to HAML.

HAML was what all the cool kids were using.

Well, HAML is a HOG!

Lower numbers are better.

Row Data

Ruby Version	Rails Version	erb	erubis	fast erubis	temple erb	slim pretty	slim ugly	haml pretty	haml ugly	slim ugly vs. erb
1.9.3	3.2.17	4.33	3.65	3.65	6.62	9.13	5.16	26.64	24.99	19.17%
	4.0.4	4.31	3.63	3.7	6.2	9.07	5.15	26.39	24.85	19.49%
	4.1.0	4.11	3.47	3.63	6.09	8.87	5.06	26.02	24.4	23.11%
2.0.0	3.2.17	3.81	3.32	3.17	6.2	9.55	4.78	25.19	22.41	25.46%
	4.0.4	3.83	3.35	3.39	6.01	9.4	4.87	25.16	22.29	27.15%
	4.1.0	3.75	3.27	3.14	6.15	9.29	4.75	24.94	22.22	26.67%
2.1.1	3.2.17	3.12	2.74	2.81	5.21	8.41	4.31	22.31	20.29	38.14%
	4.0.4	3.15	2.75	2.77	5.24	8.51	4.33	21.72	19.9	37.46%
	4.1.0	3.11	2.69	2.72	5.08	8.36	4.29	21.75	20.01	37.94%

Source: https://sephinrothcn.wordpress.com/2014/04/14/slim-vs-haml-performance-perspective/

But I Hate Typing All That ERB!

Personally, I prefer ERB. I find it much easier to following along with design/layout examples where they normally post standard HTML and JavaScript markup.

Also, in my day job we make use of ERB templates to assist us with the creation of configuration and other system type files used in our automation.

But if you really hate the extra typing involved with ERB, then give SLIM a try. It is similar to HAML but at least your applications performance will not suffer.

HTML-To-HAML: http://html2haml.herokuapp.com/

HTML-To-SLIM: https://html2slim.herokuapp.com/

Example 1 - Performance Boost

SAMPLE - before converting from HAML to ERB:

```
I, [2016-01-20T12:56:56.726111 #11575]  INFO -- :   Rendered bidding/index.html.haml within layouts/loaded_area (923.1ms)
I, [2016-01-20T12:57:11.041835 #11575]  INFO -- :   Rendered bidding/index.html.haml within layouts/loaded_area (809.7ms)
I, [2016-01-20T12:58:50.452290 #11575]  INFO -- :   Rendered bidding/index.html.haml within layouts/loaded_area (745.8ms)
I, [2016-01-20T12:59:12.090701 #11575]  INFO -- :   Rendered bidding/index.html.haml within layouts/loaded_area (811.1ms)
```

SAMPLE - after converting from HAML to ERB:

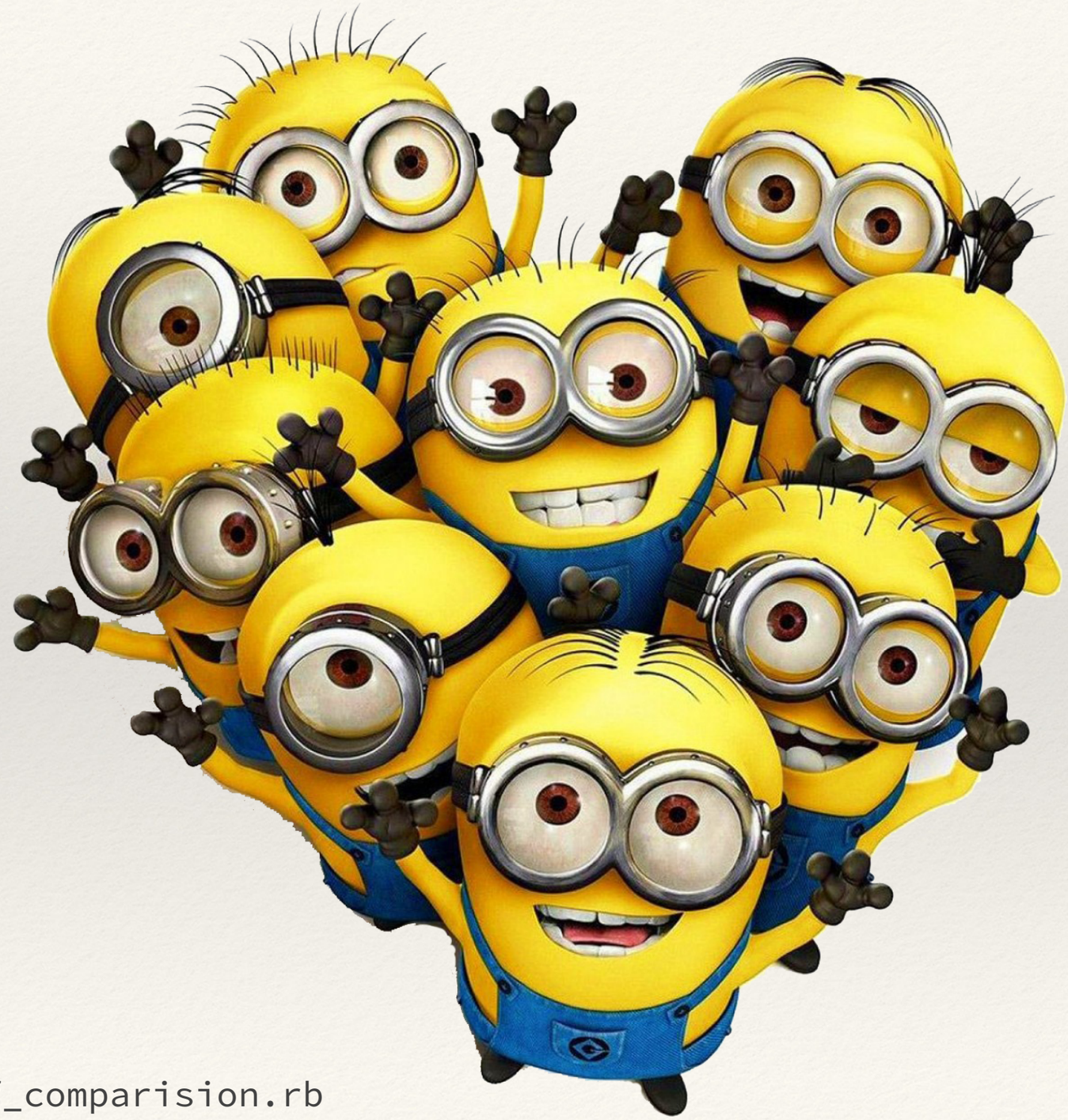
```
I, [2016-01-20T18:29:58.685644 #13821]  INFO -- :   Rendered bidding/index.html.erb within layouts/loaded_area (693.2ms)
I, [2016-01-20T18:30:15.869965 #13821]  INFO -- :   Rendered bidding/index.html.erb within layouts/loaded_area (716.7ms)
I, [2016-01-20T18:31:36.209626 #13821]  INFO -- :   Rendered bidding/index.html.erb within layouts/loaded_area (709.6ms)
I, [2016-01-20T18:32:10.981872 #13821]  INFO -- :   Rendered bidding/index.html.erb within layouts/loaded_area (690.3ms)
```

Lower numbers are better.

Performance Boost #2

- Be mindful of which data structure you use
- In the Auction App we used View Models
- Assembled data for the view using OpenStructs
- OpenStructs are mega costly

Code Example Time - YEAH!



~/CodeSnippets/perf_comparision.rb

Review Example Results

Ruby 2.3.0

```
➜  CodeSnippets ./perf_comparision.rb
                  user     system      total        real
hash:         0.260000   0.000000   0.260000 (  0.271756)
openstruct:   9.050000   0.010000   9.060000 (  9.070562)
struct:       0.160000   0.000000   0.160000 (  0.152520)
class:        0.110000   0.000000   0.110000 (  0.115192)
➜  CodeSnippets
```

For the best performance, have some class!

Ruby 1.9.3-p551

```
➜  CodeSnippets rvm use ruby-1.9.3-p551
Using /Users/jfhogarty/.rvm/gems/ruby-1.9.3-p551
➜  CodeSnippets time ruby perf_comparision.rb | tee
                  user     system      total        real
hash:         0.360000   0.020000   0.380000 (  0.382459)
openstruct:   8.130000   0.050000   8.180000 (  8.183904)
struct:       0.190000   0.000000   0.190000 (  0.186375)
class:        0.130000   0.000000   0.130000 (  0.128261)
ruby perf_comparision.rb  8.83s user 0.08s system 99% cpu 8.965 total
tee  0.00s user 0.00s system 0% cpu 8.964 total
➜  CodeSnippets
```

perf_comparision.rb

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

require 'ostruct'
require 'benchmark'

COUNT = 500_000   # Note: 10_000_000 will take
some time
NAME = "Test Name"
EMAIL = "test@example.org"

class Person
  attr_accessor :name, :email
end

Benchmark.bm(13) do |x|   #arg to bm sets the
label width
  x.report("hash:") do
    COUNT.times do
      p = {name: NAME, email: EMAIL}
    end
  end

  x.report("openstruct:") do
    COUNT.times do
     p = OpenStruct.new
     p.name = NAME
     p.email = EMAIL
   end
  end
```

```
# continued **


  x.report("struct:") do
    PersonStruct = Struct.new(:name, :email)
    COUNT.times do
      p = PersonStruct.new
      p.name = NAME
      p.email = EMAIL
    end
  end

  x.report("class:") do
    COUNT.times do
      p = Person.new
      p.name = NAME
      p.email = EMAIL
    end
  end
end
```

Performance Boost #3 - Constantize Strings

- Use Constants - in Ruby constants are not garbaged collected
- In Views with with lots of strings, convert the static ones to constants
- When you have static strings that are used across many views files, implement a constants module - UiStr
- By using constants, we removed the need to create thousands of memory hungry string objects

DEMO UiStr

- In a Rails project, create the file ui_str.rb in the app/helpers directory
- The Constants can be accessed in any view file using a format like: `link_to image_tag(UiStr::FILETYPE_CSV)`

DEMO UiStr

```
module UiStr
  DISPLAY_NONE            = "display: none"
  DISPLAY_INLINE          = "display: inline-block"
  ERROR_TAG               = "Error"
  NEW_QTY_TAG             = "new_quantity"
  SCRN_UNAVAILABLE        = "Screen unavailable"

  WARNING_IMG             = "warning.png"
  EXCLAMATION_IMG         = "exclamation-red.png"
  SOFT_DASH_IMG           = "soft-dash.png"
  CHECK_GREEN_IMG         = "check-green.png"
  MAG_PLUS_IMG            = "mag-plus.png"
  MAG_MINUS_IMG           = "mag-minus.png"
  ICON_PRINTER_IMG        = "icons/printer.png"

  CONFIRMED_TAG           = "Confirmed"
  UNCONFIRMED_TAG         = "Unconfirmed"
  NO_BID_REQUIRED_TAG     = "No bid required"
  MAG_PLUS_TAG            = "mag-plus"
  MAG_MINUS_TAG           = "mag-minus"
  NOTICE_TAG              = "Notice"
  UNSUBMITTED_TAG         = "Unsubmitted"
  PRINTER_TAG             = "printer"

  # continued ***

  STRING_TBLFRMT          = "string"
  EXACT_NUMBER_TBLFRMT    = "exact-number"
  CURRENCY_TBLFRMT        = "currency"
  NUMBER_TBLFRMT          = "number"
  PERCENTAGE_TBLFRMT      = "percentage"
  CHKBOX_TBLFRMT          = "checkbox"
  IMAGE_URL               = "image-url"
  LINK_TBLFRMT            = "link"

  PUBLISHER               = "Publisher"
  TITLE                   = "Title"
  RESERVE_PRICE           = "Reserve Price"
  SUPPLY                  = "Supply"
  UNITS                   = "Units"
  DEMAND                  = "Demand"
end
```

Example: ~/CodeSnippets/ui_str.rb

Performance Boost #4 - Strings (cont.)

- Avoid string parsing

Good	Bad
<%= @client.full_name %>	<%= “#{@client.full_name}” %>

#4 - Strings (cont.)

- When using parsed strings, use interpolated vs. concatenated

Good	Bad
“Good morning Mr. #{last_name}”	“Good morning Mr. “ << last_name

Performance Boost #5 - Iteration

Avoid costly iterators

- all?
- inject
- each_with_index

Although iterators are a great tool, time saver, and make our code more readable, they come at a cost.

`Source: Optimize Your Iterators, Chapter 2 of Ruby Performance Optimization`

#5 - Iteration (cont.)

Iterator	Enum†	Array	Range	Iterator	Enum†	Array	Range
all?	3	3	3	fill	0	—	—
any?	2	2	2	find	2	2	2
collect	0	1	1	find_all	1	1	1
cycle	0	1	1	grep	2	2	2
delete_if	0	—	0	inject	2	2	2
detect	2	2	2	map	0	1	1
each	0	0	0	none?	2	2	2
each_index	0	—	—	one?	2	2	2
each_key	—	—	0	reduce	2	2	2
each_pair	—	—	0	reject	0	1	0
each_value	—	—	0	reverse	0	—	—
each_with_index	2	2	2	reverse_each	0	1	1
each_with_object	1	1	1	select	0	1	0

Table 1—Number of additional T_NODE objects created by an iterator

† Enum is Enumerable

Source: Table 1 - Number of additional T_NODE objects created by an iterator

#5 - Iteration (cont.)

Reduce iteration over similar data

- This may sound easier than it is. How does one know what data to focus on?
- It is not always easy to see reduction opportunities by casually browsing your code.
- We need to the big boy tools to help us - RubyProf!

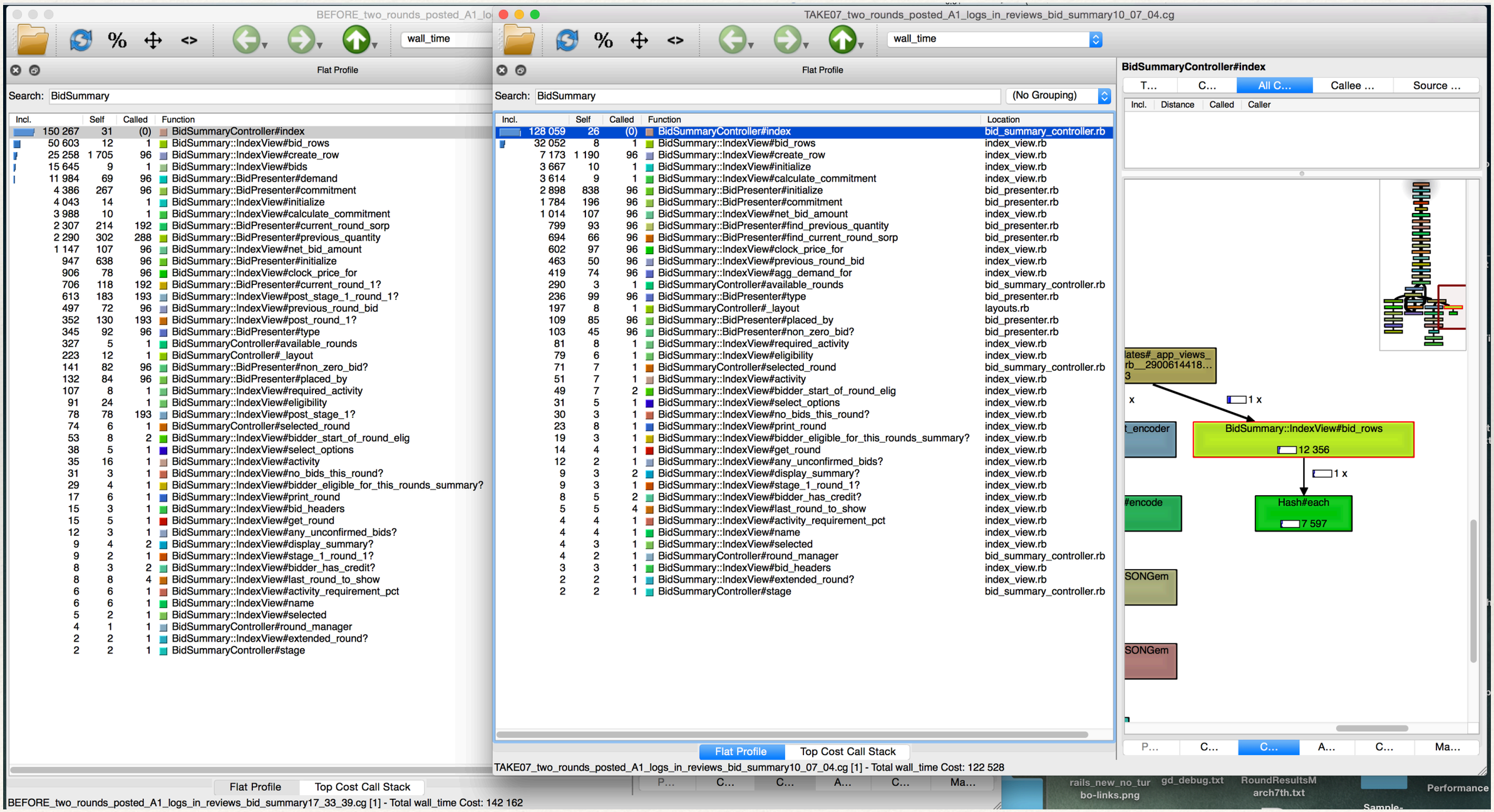
Profiling with RubyProf

- Profiling could be a few hour talk on its own - we will but graze the surface
- RubyProf is a gem that will help us profile our application
- Along with RubyProf we will also need Qcachegrind (or Kcachegrind)
- Graphviz

RubyProf Preparation

- These tools are available for both OS X and Linux but we will only cover OS X here
- gem install ruby-prof
- brew install qcachegrind
- brew install graphviz

DEMO Qcachegrind Output



How To Use RubyProf

- You have installed the gem and the qcachegrind and graphviz tools
- Pick a controller of one of your sluggish components
- Add RubyProf to the controller

Example: Controller with RubyProf

```
class BidSummaryController < ApplicationController
require 'ruby-prof'

  def index
    RubyProf.start                                          # starts the profiling process

    products = Product.all
    bidders = Bidder.all
    bids = Bid.all

    @bid_summary = BidSummary::IndexView.new(products, bidders, bids)

    render 'index'
    profile_result = RubyProf.stop                          # stops profiling, storing in variable
    RubyProf::FlatPrinter.new(profile_result).print(STDOUT)  # prints out to your server console

    # Create a .cg file of the gathered data so we can view with qcachegrind
    stampit = DateTime.now.strftime("%H_%M_%S")             # create a time stamp
    RubyProf::CallTreePrinter.new(profile_result).print(profile: "data/perf_profile/BEFORE/bid_summary-
#{stampit}")
  end
end
```

How To Use RubyProf

- Start the rails server in PRODUCTION mode
- Visit the slow view page and hit refresh about 10 times
- The reason we hit refresh at least 10 times is so that we can get a good sample of data
- We are capturing the ‘Before’ state

Example Production Log - Before Optimization

```
tails -f log/production.log

I, [2016-03-10T17:32:04.828281 #90561]  INFO -- : Started GET "/test416/bid_summary" for 127.0.0.1 at
2016-03-10 17:32:04 -0500
I, [2016-03-10T17:32:04.830102 #90561]  INFO -- : Processing by BidSummaryController#index as HTML
I, [2016-03-10T17:32:04.830147 #90561]  INFO -- :   Parameters: {"area_id"=>"test416"}
I, [2016-03-10T17:32:04.868982 #90561]  INFO -- :   Rendered shared/_info_element.html.erb (0.2ms)
I, [2016-03-10T17:32:04.869672 #90561]  INFO -- :   Rendered shared/_info_element.html.erb (0.1ms)
I, [2016-03-10T17:32:04.870213 #90561]  INFO -- :   Rendered shared/_info_element.html.erb (0.1ms)
I, [2016-03-10T17:32:04.870662 #90561]  INFO -- :   Rendered shared/_info_element.html.erb (0.1ms)
I, [2016-03-10T17:32:04.872660 #90561]  INFO -- :   Rendered bid_summary/_info_bar.html.haml (10.6ms)
I, [2016-03-10T17:32:04.991354 #90561]  INFO -- :   Rendered shared/_sort_filter_table.html.haml (5.2ms)
I, [2016-03-10T17:32:04.991640 #90561]  INFO -- :   Rendered bid_summary/index.html.erb within layouts/
loaded_area (138.3ms)
I, [2016-03-10T17:32:04.996141 #90561]  INFO -- :   Rendered layouts/_round_info.html.erb (0.6ms)
I, [2016-03-10T17:32:05.003536 #90561]  INFO -- :   Rendered /Users/jfhogarty/.rvm/gems/ruby-2.2.2/
bundler/gems/auction_works_framework-053d471f2c24/app/views/layouts/_browser_notice.html.erb (0.5ms)
I, [2016-03-10T17:32:05.005162 #90561]  INFO -- :   Rendered layouts/_round_info.html.erb (0.2ms)
I, [2016-03-10T17:32:05.017440 #90561]  INFO -- :   Rendered layouts/_footer.html.erb (0.4ms)
I, [2016-03-10T17:32:05.023126 #90561]  INFO -- :   Rendered /Users/jfhogarty/.rvm/gems/ruby-2.2.2/
bundler/gems/auction_works_framework-053d471f2c24/app/views/layouts/base.html.erb (4.8ms)
I, [2016-03-10T17:32:05.396708 #90561]  INFO -- : Completed 200 OK in 566ms (Views: 186.9ms)
```

Example Production Log - After Optimization

```
tails -f log/production.log

I, [2016-03-15T10:08:00.902370 #97746]  INFO -- : Started GET "/test416/bid_summary" for 127.0.0.1 at
2016-03-15 10:08:00 -0400
I, [2016-03-15T10:08:00.903222 #97746]  INFO -- : Processing by BidSummaryController#index as HTML
I, [2016-03-15T10:08:00.903273 #97746]  INFO -- :   Parameters: {"area_id"=>"test416"}
I, [2016-03-15T10:08:00.913852 #97746]  INFO -- :   Rendered shared/_info_element.html.erb (0.2ms)
I, [2016-03-15T10:08:00.914504 #97746]  INFO -- :   Rendered shared/_info_element.html.erb (0.1ms)
I, [2016-03-15T10:08:00.915098 #97746]  INFO -- :   Rendered shared/_info_element.html.erb (0.2ms)
I, [2016-03-15T10:08:00.915576 #97746]  INFO -- :   Rendered shared/_info_element.html.erb (0.1ms)
I, [2016-03-15T10:08:00.917767 #97746]  INFO -- :   Rendered bid_summary/_info_bar.html.haml (4.8ms)
I, [2016-03-15T10:08:01.000096 #97746]  INFO -- :   Rendered shared/_sort_filter_table.html.haml (1.6ms)
I, [2016-03-15T10:08:01.000382 #97746]  INFO -- :   Rendered bid_summary/index.html.erb within layouts/
loaded_area (89.4ms)
I, [2016-03-15T10:08:01.004161 #97746]  INFO -- :   Rendered layouts/_round_info.html.erb (0.2ms)
I, [2016-03-15T10:08:01.009996 #97746]  INFO -- :   Rendered /Users/jfhogarty/.rvm/gems/ruby-2.2.2/
bundler/gems/auction_works_framework-053d471f2c24/app/views/layouts/_browser_notice.html.erb (0.4ms)
I, [2016-03-15T10:08:01.011407 #97746]  INFO -- :   Rendered layouts/_round_info.html.erb (0.2ms)
I, [2016-03-15T10:08:01.021012 #97746]  INFO -- :   Rendered layouts/_footer.html.erb (0.4ms)
I, [2016-03-15T10:08:01.026250 #97746]  INFO -- :   Rendered /Users/jfhogarty/.rvm/gems/ruby-2.2.2/
bundler/gems/auction_works_framework-053d471f2c24/app/views/layouts/base.html.erb (4.4ms)
I, [2016-03-15T10:08:01.325639 #97746]  INFO -- : Completed 200 OK in 422ms (Views: 116.9ms)
```

The In-Between

The previous two slides demonstrated the rendering performance of the targeted view before any optimization and then afterwards. But surely there is more to the story.

I didn't have a magic wand that I could wave to make things all better. I had to learn, the hard way through trial and error, how to read the output of the CG files and investigate what the pretty visualization was trying to tell me.

I did have help though. Stuart, my lead on this particular project helped me get the tools setup and provided a quick and dirty lesson.

Let's take a look at that example output again that we saw earlier.

Live Demo

BEFORE - Optimization

- Benchmark current application
- Need to start rails server in production mode
- Need to complete a few pre-steps

Preparation

- Prepare Database: RAILS_ENV=production rake db:migrate
- Prepare Assets: RAILS_ENV=production rake assets:precompile
- Populate Database: RAILS_ENV=production DISABLE_DATABASE_ENVIRONMENT_CHECK=1 rake dev:prime
- Start Web Server: RAILS_ENV=production rails s
- Monitor the logs: tail -f logs/production.log
- Visit Bid Summary Page, refresh 10+ times

Assets Issues - Fix

- Edit config/environments/production.rb
- RAILS_ENV=production rake assets:clean
- RAILS_ENV=production rake assets:clobber
- RAILS_ENV=production rake assets:precompile

Implement UiStr

- git checkout add_ui_str_class
- Start Web Server: RAILS_ENV=production rails s
- Monitor the logs: tail -f logs/production.log
- Visit Bid Summary Page, refresh 10+ times
- Take note of response time:
- `Completed 200 OK in 153ms (Views: 147.7ms | ActiveRecord: 4.3ms)`

Ruby Prof Time

- git checkout add_ruby_prof
- Start Web Server: RAILS_ENV=production rails s
- Monitor the logs: tail -f logs/production.log
- Visit Bid Summary Page, refresh 10+ times
- Examine RubyProf files in: `data/perf_profile/`

Summary

- Convert view files from HAML to ERB
- Use efficient data structures - Class > OpenStruct
- Implement a constants module for common strings
- By using constants, we removed the need to create thousands of memory hungry string objects
- Avoid String Parsing

Resources

- Information related to comparing OpenStruct to Hash, Struct and Class: http://palexander.posthaven.com/ruby-data-object-comparison-or-why-you-should-never-ever-use-openstruct
- Information related to comparing erb to haml and slim: https://sephinrothcn.wordpress.com/2014/04/14/slim-vs-haml-performance-perspective/
- Information related to mem_measure and Ruby Prof: Book: 'Ruby Performance Optimization: Why Ruby is Slow and How to Fix it' by Alexander Dymo
- 20 Ruby Performance Tips http://www.monitis.com/blog/2012/02/08/20-ruby-performance-tips
- http://blog.endpoint.com/2012/05/profile-ruby-with-ruby-prof-and.html

In Closing…

- John Hogarty (john_hogarty@ognt.io)
- Twiter/Github: hogihung
- Blog: http://ognt.io -OR- http://oldguynewtrick.com
- Repo: https://github.com/hogihung/bidder-up

The End