ARCH RAILS

#8: Blazing Fast Rails API

or

what to do when ActiveRecord sucks

Konza Investment Tech

www.konzainvestech.com

@vstegman

https://github.com/vstegman/speedy_api_demo

What We Will Cover

“ScoreKeeper” Data Model Overview

Users 1..n **Scores** n..1 **Games**

- value
- date

API end points

api/v1/users # index for Users

api/v1/games # index for Games

nested resources for:

:id/scores # all scores for User or Game

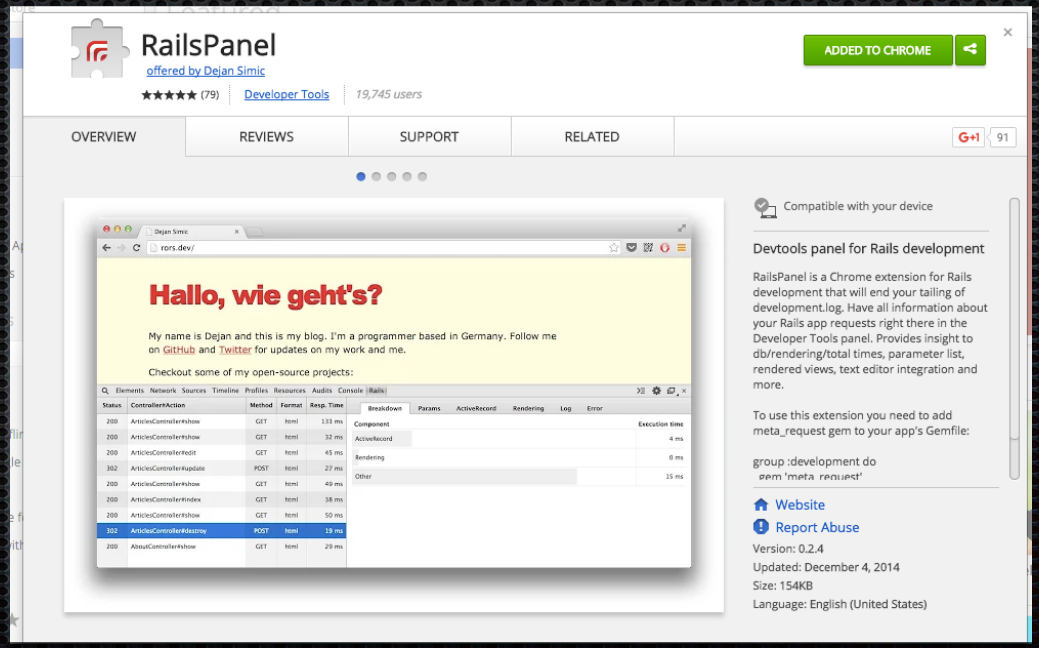
:id/stats # summary stats for User or Game

Gems

- ‘oj’ -> Optimized JSON
- ‘benchmark_suite’ -> Enhances standard ruby Benchmark
- ‘meta_request’ -> Supports RailsPanel in Chrome
- ‘descriptive-statistics’ -> Mean, Mode, Standard Deviation

RailsPanel

Chrome Add In
requires ‘meta_request’ Gem



Problem #1, Rendering

```ruby
class Api::V1::ScoresController < ApplicationController
  def index
    scores = Score.where(user_id: params[:user_id])
    hash = {
      meta: {user_id: params[:user_id], count: scores.length},
      scores: scores
    }
    render json: hash
  end
end
```


```
{
  “meta":
      {“user_id”:”5","count":10000},
  “scores":
      [
        {“value":44553,"date":"2002-09-10","user_id":5,"game_id":10}
        etc….
      ]
}
```


~/api/v1/users/2/scores

Request takes almost 3 seconds! Most time used in Rendering

Solution #1: Get Specific

```ruby
class Score < ActiveRecord::Base
  belongs_to :user
  belongs_to :game

  SCORE_STRUCT = Struct.new('Score', :value, :date, :user_id, :game_id)


  def self.as_struct(struct = nil)
    struct ||= SCORE_STRUCT
    arr = pluck(:value, :date, :user_id, :game_id)
    structs = []
    arr.each {|cf| structs << struct.new(cf[0],cf[1],cf[2], cf[3])}
    structs
  end

  def as_json(args)
    {value: value, date: date, user_id: user_id, game_id: game_id}
  end

end
```

Tell the render json: command exactly what to do with an as_json method

Reduces request time by ~2 seconds

Solution #2: Parse Faster

```
1  class Api::V1::ScoresController < ApplicationController
2    def index
3      scores = Score.where(user_id: params[:user_id])
4      hash = {
5        meta: {user_id: params[:user_id], count: scores.length},
6        scores: scores
7      }
8      #render json: hash
9      render json: Oj.dump(hash, mode: :compat)
10   end
11 end
12
```

Oj is faster than the default JSON rendering

Reduces request time by another 50%

Solution #3: Bypass ActiveRecord

```ruby
class ScoreReader
  attr_reader :value, :date, :user_id, :game_id
  def initialize(args)
    @value = args[0]
    @date = args[1]
    @user_id = args[2]
    @game_id = args[3]
  end

  def self.build_from_relation(relation)
    arr = relation.pluck(:value, :date, :user_id, :game_id)
    obj = []
    arr.each {|r| obj << ScoreReader.new(r)}
    obj
  end

  def as_json
    {
      value: @value,
      date: @date,
      user_id: @user_id,
      game_id: @game_id
    }
  end

end
```

Use a PORO to encapsulate your data without the overhead of ActiveRecord

Place logic in specific objects and avoid a ‘God class’

Bypass ActiveRecord

```
1  class Api::V2::ScoresController < ApplicationController
2    def index
3      scores = ScoreReader.build_from_relation(
4        Score.where(user_id: params[:user_id])
5      )
6      hash = {
7        meta: {
8          user_id: params[:user_id],
9          count: scores.length
10       },
11       scores: scores
12     }
13     render json: Oj.dump(hash , mode: :compat)
14   end
15 end
~
~
~
```

V2 of our API incorporates our prior lessons plus ScoreReader

And we can still take advantage of ActiveRecord's querying interface

We've reduced response time from 3 seconds to 100~200ms for a 10,000 record request!

Next Challenge: Calculations

~/api/v1/users/4/stats

- For Each User / Game calculate the average (mean) score and…
- Create a “Skill Score” to show how consistently good the User is:

$$\frac{\text{average score}}{\text{standard deviation of scores}}$$

Benchmarking calculation options with Benchmark#ips

Class Options

- Hash
- PORO
- Struct

Sum Options

- #each with a sum variable
- #inject (this is how Rails implements Array#sum)
- #reduce

Benchmarking Conclusions

- When instantiating, Hashes are fast, Structs are faster, simple PORO similar
- Array#each is fast, consider using it instead of #reduce, #inject, or #map

AdvancedStat Class


```
{
    “meta”:{"user_id":"1"},
    “stats": [
      {
        “game_id”:10,
        “count":1000,
        “mean":57213.383,
        “skill_score”:9.840391
      },{
        “game_id”:9,
        …
    ]
}
```


```ruby
class AdvancedStat

  def initialize(relation)
    if(relation.is_a? ActiveRecord::Relation)
      @stats = DescriptiveStatistics::Stats.new(relation.pluck(:value))
    else
      @stats = DescriptiveStatistics::Stats.new(relation)
    end
  end

  def to_hash
    {
      count: @stats.length,
      mean: @stats.mean,
      skill_score: @stats.mean / @stats.standard_deviation
    }
  end
end
```

Class Method as_struct

Define the Struct in an initializer

```
Struct.new('Score', :value, :date, :user_id, :game_id)
```

Loads data into an array of faster, attribute only objects

```
def self.as_struct(struct = nil)
  arr = pluck(:value, :date, :user_id, :game_id)
  structs = []
  arr.each {|cf| structs << SCORE_STRUCT.new(cf[0],cf[1],cf[2], cf[3])}
  structs
end
```

Putting it all together: GroupedScore

```
class GroupedScore
  def initialize(relation)
    scores = relation.as_struct
    @scores = scores.group_by {|s| s.game_id}
    run_stats
  end

  def run_stats
    @stats = []
    @scores.each do |k,v|
      stat = AdvancedStat.new(v.map(&:value)).to_hash
      stat[:game_id] = k
      @stats << stat
    end
    @stats
  end

  def as_json
    @stats
  end

end
```

Now our Controller simply needs to pass an AR::Relation of Scores to a PORO to handle the business logic and presentation

More Resources

- http://brainspec.com/blog/2012/09/28/lightning-json-in-rails/
- http://www.ohler.com/oj/
- https://github.com/evanphx/benchmark-ips