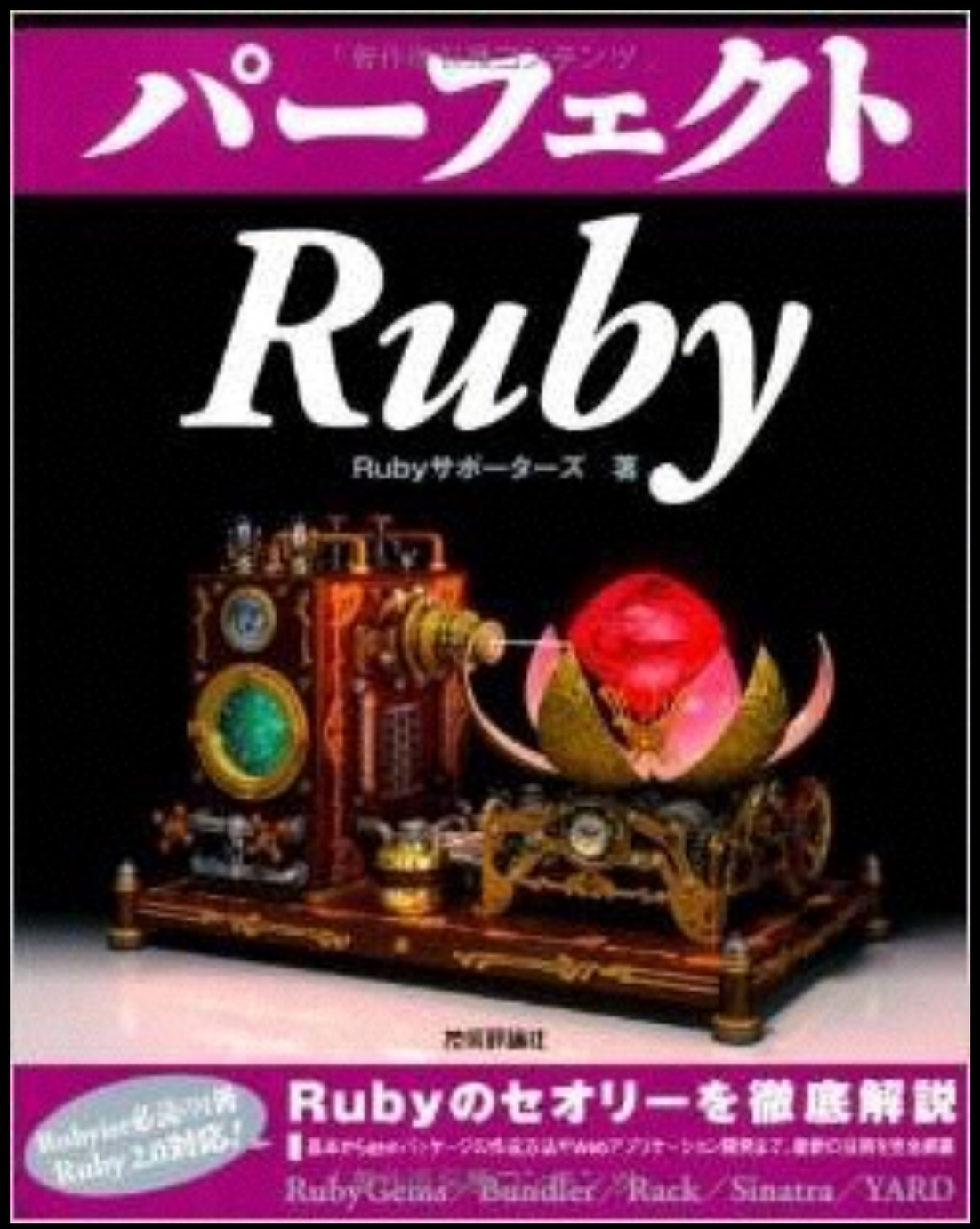
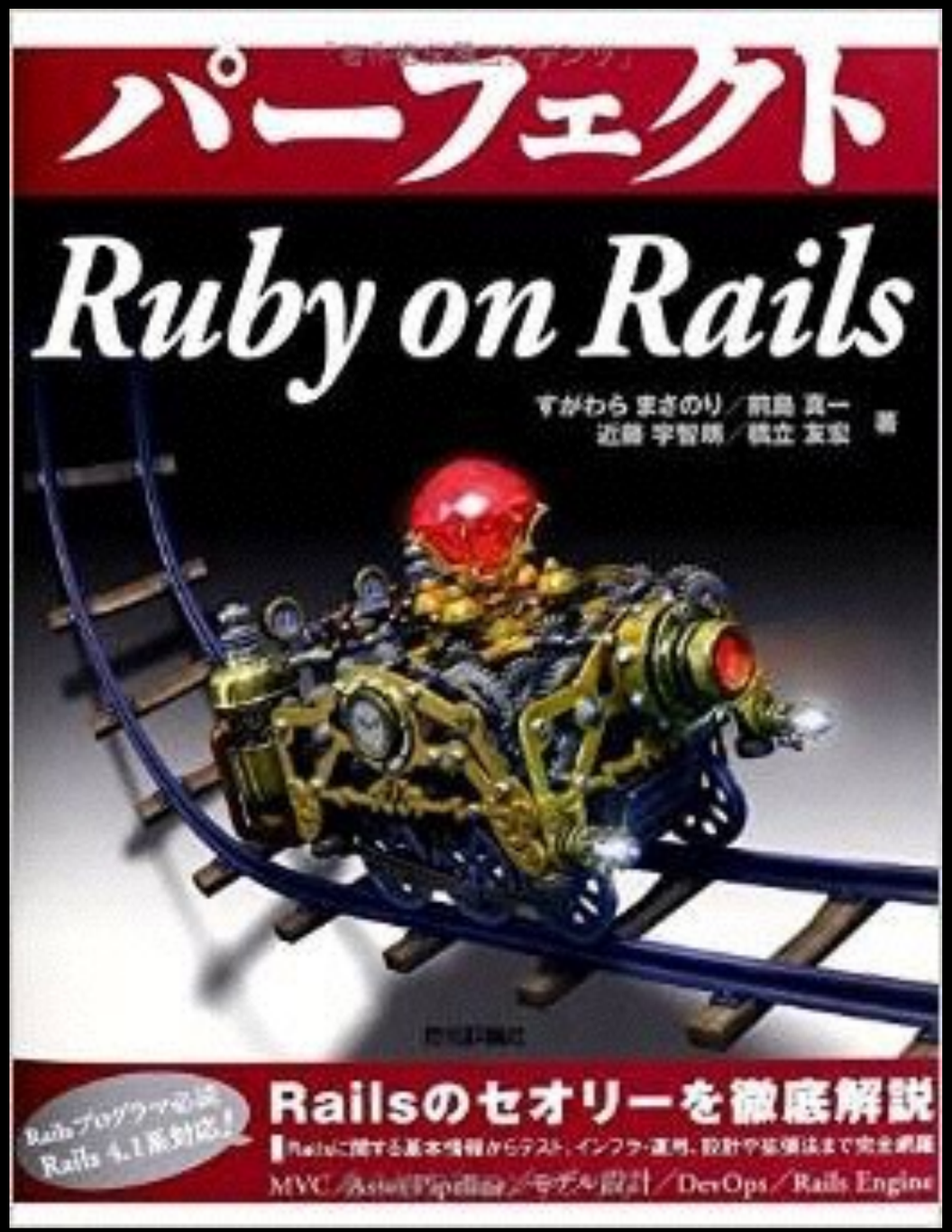
Railsパフォーマンス基本のキ

Tomohiro Hashidate
(@joker1007)

p self
=> @joker1007

パRuby パRails





at Crowdworks as freelance

Crowd Works

株式会社クラウドワークス

21世紀の新しいワークスタイルを提供する日本最大級のクラウドソーシング「クラウドワークス」のエンジニアチームです！

161 投稿	6017 ストックされた回数

この内５１００が俺

最近の仕事

- Redshiftをバックエンドにした分析基盤
- esminc/adhoq を弄ってプルリクを出す
- Railsアプリのリファクタリング/改善

あるページの
表示までの時間

7sec over

(development)

これはひどい

他人事ではない


Railsは遅くないが
何も考えてないと
「遅くなる」
https://flic.kr/p/9U8CDz

遅くなる要因

- クソクエリ
- 無駄なクエリ発行とN+1
- ActiveRecordのインスタンス化コスト
- 多過ぎるpartial view
- そもそもビジネスロジックの実装が(ry

例を見ていく

```
<b><span class="emphasis"><%= (User.employer_count / 10000.0).round %>
```

クソクエリ

ビューからカジュアルに
Userをcountする

```ruby
def self.search_with_db(source, options, order = nil)
  # 以下でoptionsの中身を追加/削除しても
  # paramsに影響を与えないように
  options = Marshal.load(Marshal.dump(options))

  enable_option = !options[:disable_options] && true

  # order を指定すると、注目オプションは無効になる
  if order.present?
    enable_option = false
  else
    order = options[:payment_type] == 'competition' ? '-expires' : '-new'
  end

  sorts = []
  sorts <<
    case order
    when 'new'
      'released_at DESC'
    when '-new'
      'released_at ASC'
    when 'popular'
      case options[:payment_type]
      when 'competition'
        'job_offers.num_proposal_products DESC'
      when 'task'
        'job_offers.num_completed_tasks DESC'
      else
        'job_offers.num_application_conditions DESC'
      end
    when '-popular'
      case options[:payment_type]
      when 'competition'
        'job_offers.num_proposal_products ASC'
      when 'task'
        'job_offers.num_completed_tasks ASC'
      else
        'job_offers.num_application_conditions ASC'
      end
    when 'price'
      'sort_amount'
    when '-price'
      'sort_amount DESC'
    when 'expires'
      'expired_on DESC'
    when '-expires'
      'expired_on ASC'
    else
      'released_at DESC' # 不正なパラメーター中回避へのアクセスに対応
    end

  query = self.includes(:summary, :attachments, :options, :competition, :task, :job_offer_partner_users, :category, :skills, user: { profile: :picture }, manual_skills: :skill).
    joins("JOIN job_offer_summaries ON job_offers.id = job_offer_summaries.job_offer_id").
    joins(:user).merge(User.active)

  job_offers = JobOffer.arel_table
  users = User.arel_table

  base_conditions = job_offers[:status].in(options[:released_only] ? [STATUS_RELEASED] : [STATUS_RELEASED, STATUS_CLOSED]).and(job_offers[:closed_flag].eq(false))

  query = query.where("? <= job_offer_summaries.min_budget", options[:price_min].to_i) if options[:price_min].to_i > 0
  query = query.where("job_offer_summaries.max_budget <= ?", options[:price_max].to_i) if options[:price_max].to_i > 0

  if options[:keywords].present?
    query = query.joins(:summary).where('MATCH(full_text_keywords.text) AGAINST (? IN BOOLEAN MODE)', CrowdWorks::Service::SearchService.search_text(options[:keywords]))
  end

  if source == 'partners'
    base_conditions = base_conditions.and(job_offers[:partners_only_flag].eq(true))
    # 一部のパートナーだけを限定している仕事は表示しない
    user_id = options[:user_id].to_i
    query = query.joins("JOIN (SELECT id FROM job_offers WHERE id NOT IN (SELECT job_offer_id FROM job_offer_partner_users) OR id IN (SELECT job_offer_id FROM job_offer_partner_users WHERE user_id = #{user_id})) AS jo ON job_offers.id = jo.id")

    if options[:partners_only]
      query = query.joins("JOIN partnerships ON partnerships.employer_user_id = job_offers.user_id").merge(Partnership.of_user_as_employee(user_id))
    end
  else
    base_conditions = base_conditions.and(job_offers[:partners_only_flag].eq(false))
  end

  if source == 'watchlist'
    if options[:user_id].blank? && options[:watchlist_session_id].blank?
      raise ArgumentError, "user_id OR watchlist session_id is required."
    end
```

クソクエリ

poor man’s search engine

```ruby
users = User.employer
users.each |u|
  u.skills.any? do |skill|
    skill.skill_group.name == "Ruby"
  end
end
```

無駄クエリ

N+1

```ruby
def category_root
  return self if depth == 1

  parent.category_root
end

def developer?
  category_root == "developer"
end
```

無駄クエリ

再帰呼び出しの中でクエリ発行

```ruby
competitions = JobOffer.competitions

competitions.released.each do |c|
  p c.title
end

competitions.opened do |c|
  p c.title
end

competitions.closed.each do |c|
  p c.title
end
```

無駄クエリ

少しづつ違うRelationを何度も呼ぶ

```ruby
journeys.all? do |departure_schedule, destination_schedule|
  departure_schedule.is_punctual? && destination_schedule.is_punctual?
end

def is_punctual?
  (actual_time.to_i - estimate_time.to_i) < service_provider.schedule_buffer
end
```

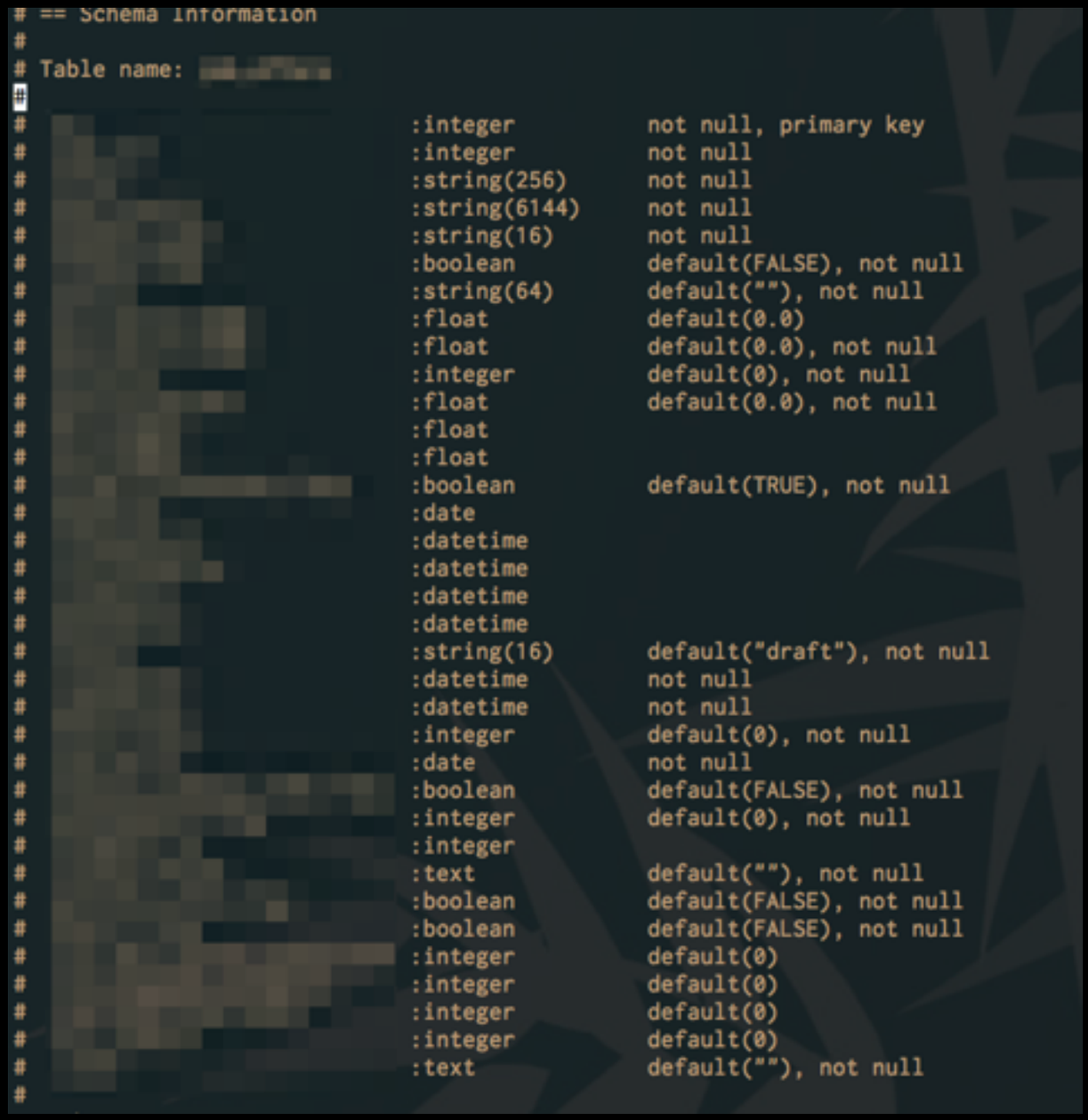
無駄クエリ

複数の子から呼ばれる親

```ruby
class ArModel < ActiveRecord::Base
  serialize :col1, Array
  serialize :col2, Array
  serialize :col3, Hash
  serialize :col4, Hash
  serialize :col5, Hash
  serialize :col6, Hash
  serialize :col7, Array
  serialize :col8, Array
  serialize :col9, Array
end
```

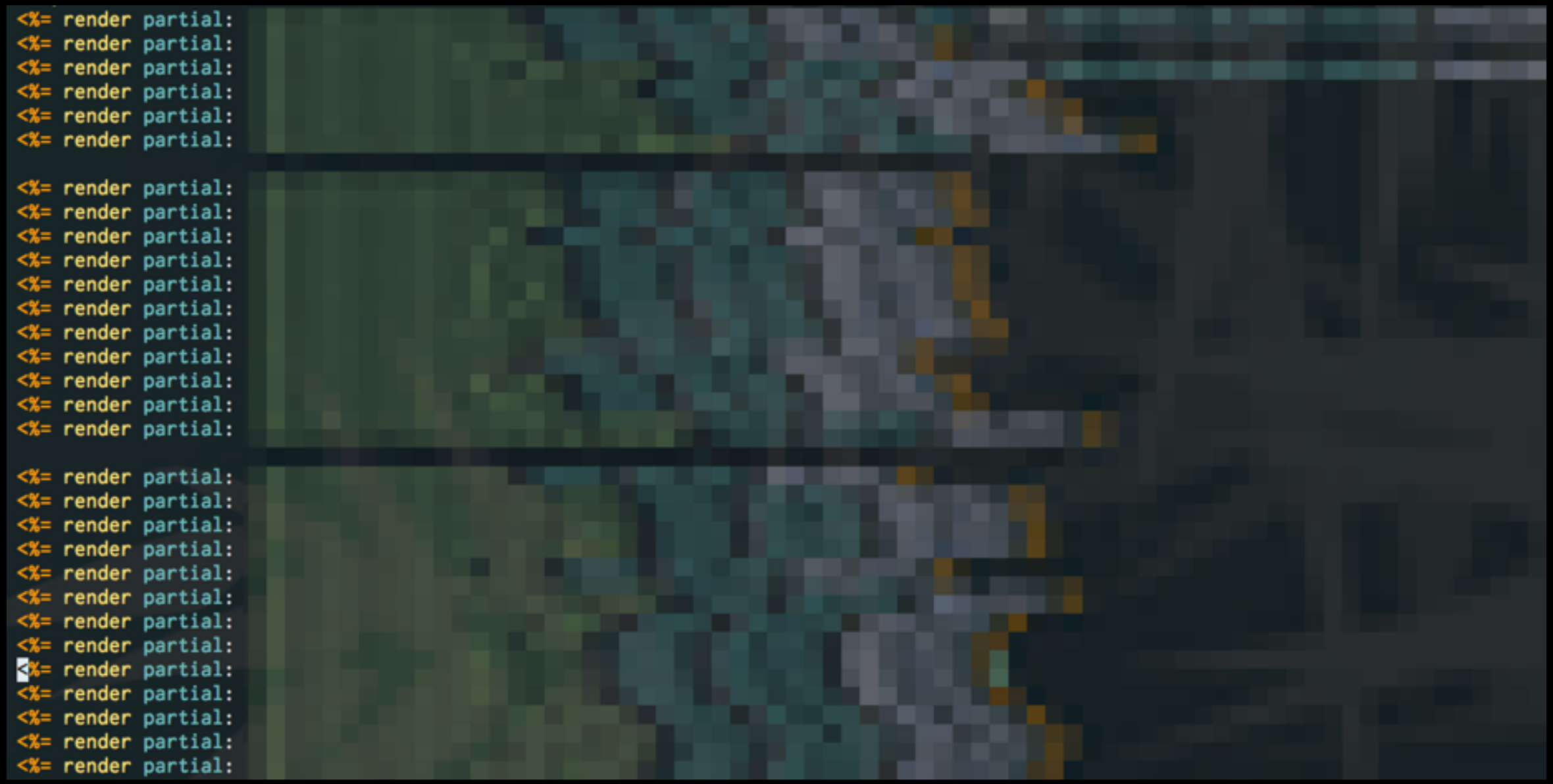
ARのインスタンス化コスト

serializeの罠



ARのインスタンス化コスト

大量のカラムと不要なインスタンス化



過剰なpartial view

不要なものまで全部renderして隠す

過剰なpartial view

jbuilderが割と重い

一つ一つは

数十msの無駄が

積み重なると数百msに

ちなみに

最近で一番やばかったのは

```ruby
class ArModel < ActiveRecord::Base
  serialize :col1, Array
  serialize :col2, Array
  serialize :col3, Hash
  serialize :col4, Hash
  serialize :col5, Hash
  serialize :col6, Hash
  serialize :col7, Array
  serialize :col8, Array
  serialize :col9, Array
end
```

ARのインスタンス化コスト

serializeの罠

```
 13.6ms +      0.5ms ( 1122)
 26.7ms +      1.3ms (  935)

727.7ms +     31.5ms ( 3468)
```

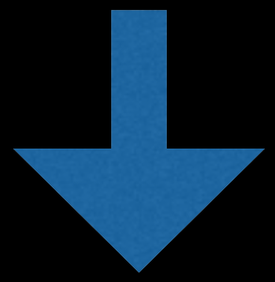
```
def hourly_wage_budget
  if leaf? && metainfo.present? && metainfo.min
    I18n.t("job_category_hourly_wage_buget.#{me
  end
end

def payment_types
  metainfo.payment_types.try(:map, &:to_sym) ||
end
```

```
 12.8ms +      0.5ms ( 1122)
 24.1ms +      0.5ms (  935)

  5.8ms +      0.7ms ( 2312)
```

```
def hourly_wage_budget
  if leaf? && metainfo.present? && metainfo.min_hourly_wage_
    I18n.t("job_category_hourly_wage_buget.#{metainfo.min_ho
  end
end

def payment_types
  metainfo.payment_types.try(:map, &:to_sym) || []
end
```

```
  39.7ms +      2.6ms ( 1244)
2181.2ms +     88.8ms ( 1202)
```

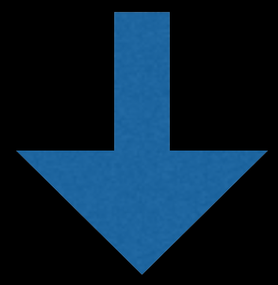
```
def title_helper
  if leaf? && metainfo.present?
    metainfo.title_helper.presence
  end
end
```



```
36.5ms +      1.5ms ( 1244)
 1.6ms +      0.2ms (  601)
```

```
def title_helper
  if leaf? && metainfo.present?
    metainfo.title_helper.presence || []
  end
end
```

7sec over



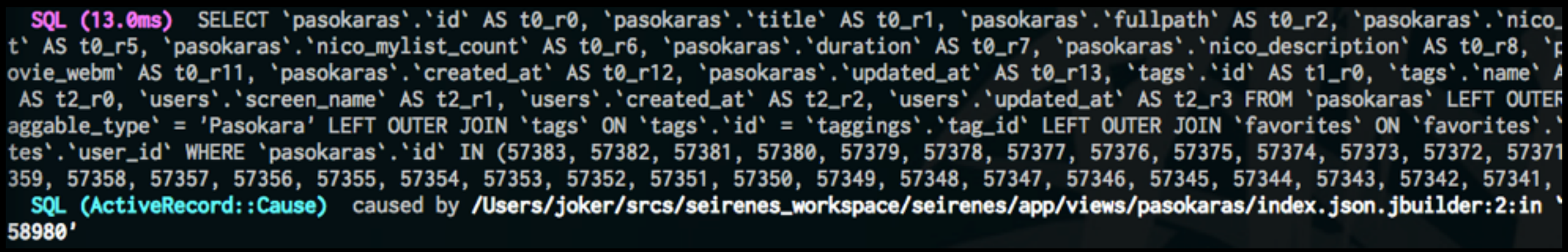
1.5sec

無駄と戦うために

DBの

気持ちを知る

検知と計測のための
ツールを知る



activerecord-cause

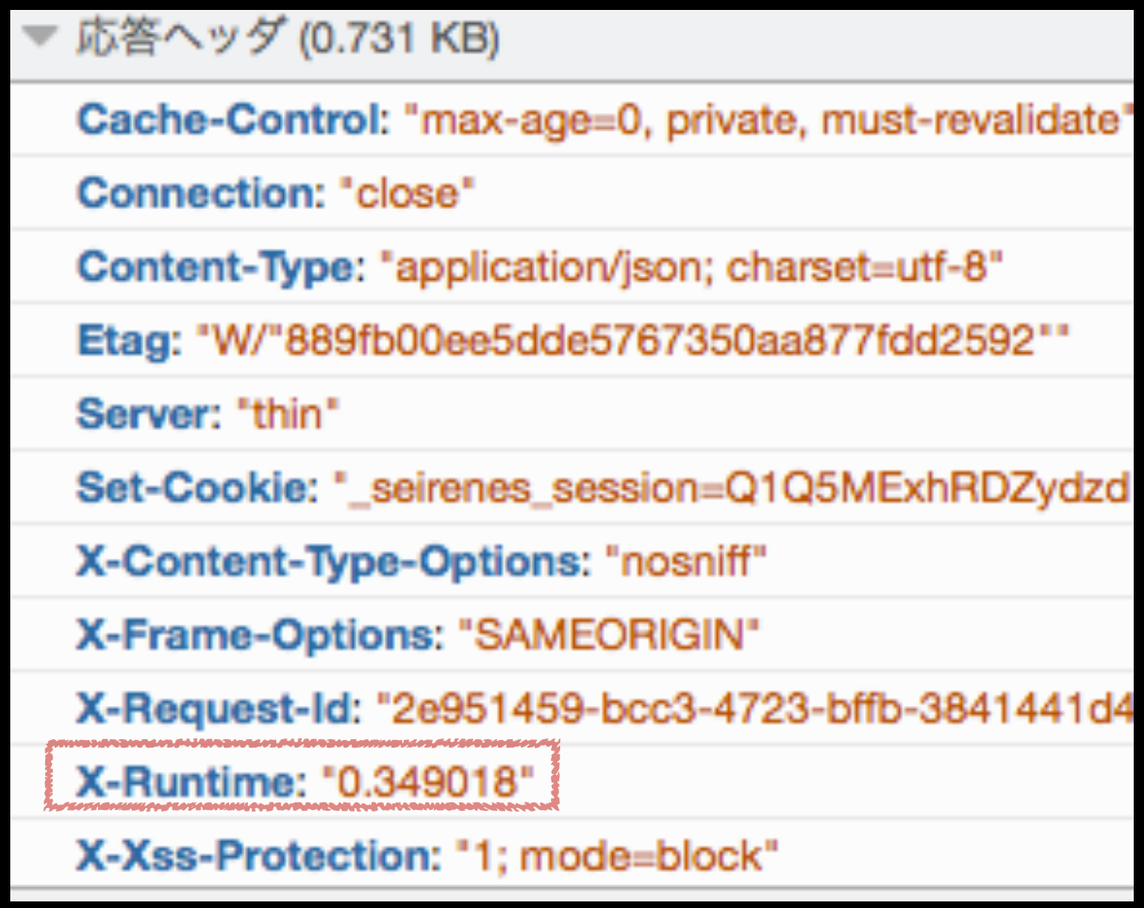
SQLが実際に発行された場所のバックトレースを出す

- N+1 Query:

```
2009-08-25 20:40:17[INFO] N+1 Query: PATH_INFO: /posts;    model: Post =>
Add to your finder: :include => [:comments]
2009-08-25 20:40:17[INFO] N+1 Query: method call stack:·
/Users/richard/Downloads/test/app/views/posts/index.html.erb:11:in `_run_
/Users/richard/Downloads/test/app/views/posts/index.html.erb:8:in `each'
/Users/richard/Downloads/test/app/views/posts/index.html.erb:8:in `_run_e
/Users/richard/Downloads/test/app/controllers/posts_controller.rb:7:in `i
```

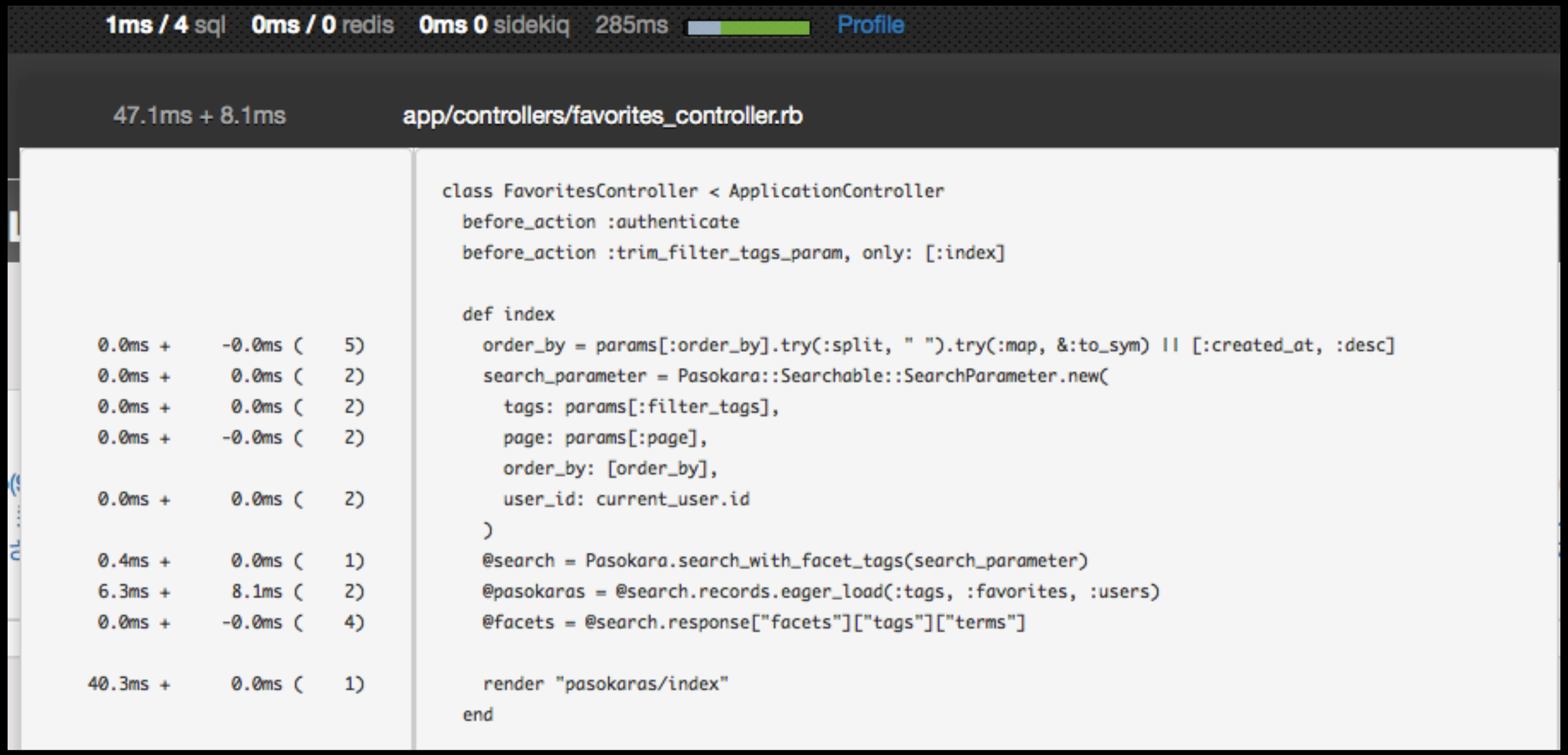
bullet

N+1や使われていないeager load,
counter cacheを検知する



rack-runtime

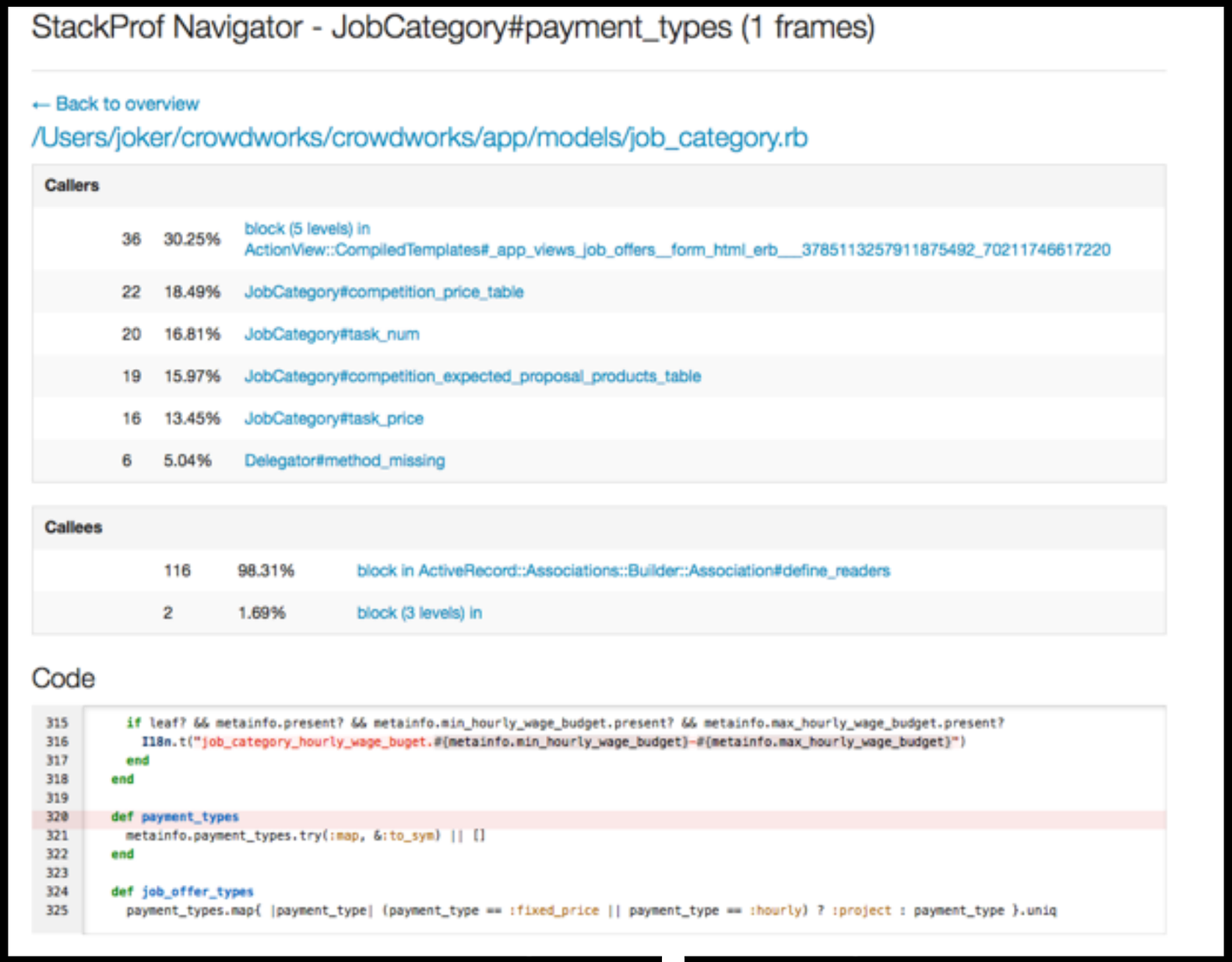
rack層でリクエストにかかった時間を計測して返す



rblineprof & peek

アクセスを処理する際の行毎の処理時間を表示する

rack-lineprofもある



stackprof-webnav

どの処理がどこから呼ばれて、どれくらいの時間を使っているのか
スタックをどんどん掘って見ていくことができる

ざっくりまとめ

- DBアクセスの基本を守る
  - 無駄なクエリを呼ばないこと
- cacheに頼るのもいいが根本的に解決できるならその方が良い
  - モデル層のパフォーマンスはあらゆる箇所に影響する
- serialize(特にYAML)はヤバイから気を付けて使うべし
- 計測が大事。計測ツールをすぐ使えるように覚えておこう

最も大事なこと

割れた窓を放置しない