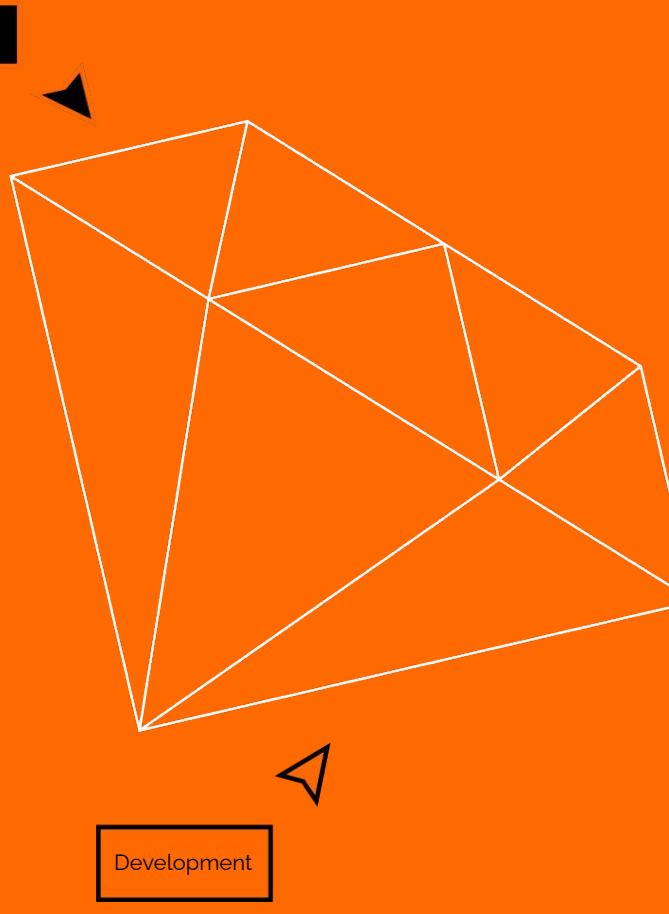
RNDSOFT



Руби'лово за память

RNDSOFT

Самойленко Юрий

- Вошёл в IT 20 лет назад
- Много лет писал на c/c++, Qt
- Был core linux developer - драйвера, библиотеки для программно-аппаратных комплексов для оборонки в области DSP
- Более 10 лет в области web, ruby/rails, асинхронных систем, docker...

В настоящее время работаю в **RNDSOFT** - техлид, архитектор, devops и еще много всего интересного.

Пишу корпоративный блог

blog.rnds.pro

О компании

RNDSOFT

10 лет

Компания RNDSOFT на рынке с 2014 года

70

Собственных продуктов, в том числе в области СМЭВ, ГОСУСЛУГИ, KYC

187+

Реализованных проектов

1000+

Клиентов в сфере BankTech, FinTech, B2B и B2G

Alfa-Bank · БКС БАНК · ЭТП ГПБ КОНСАЛТИНГ · МИГ КРЕДИТ · Деньгимигом · ДЕНЬГИ СРАЗУ · защита онлайн

ZAYMIGO · СРОЧНО деньги · LIME · ПРАВО ОНЛАЙН коллекторское агентство · ГРИН МАНИ · FINBRIDGE

RNDSOFT

Интеграционная Платформа «Агредатор»

Облачный сервис

Коробочная версия

10 Банки

БКС БАНК

10 Инвестиционные компании

РСХБ УПРАВЛЕНИЕ АКТИВАМИ

90 МФО и МКК

ЗАЙМЕР
РОБОТ ОНЛАЙН ЗАЙМОВ

10 Государственные учреждения

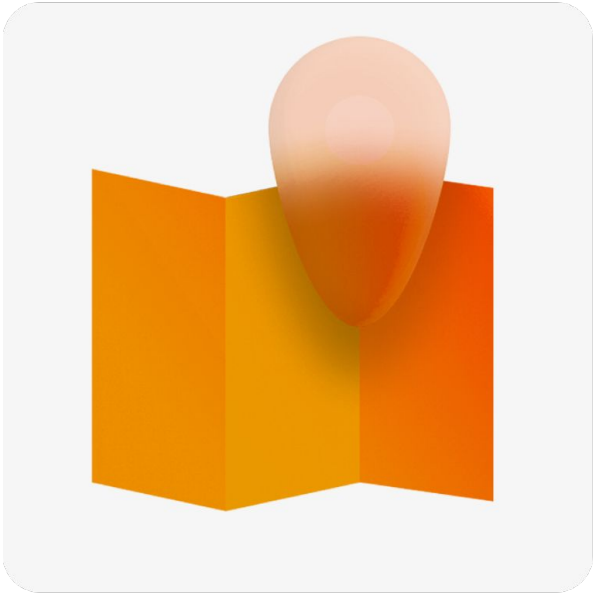
7 Страховые компании

ИНГОССТРАХ Ingosstrakh

О чём и для кого?

- О различных способах экономии памяти в экосистеме Ruby;
- Не про “скучные” способы - не про MALLOC_ARENA_MAX
- Для всех уровней Ruby разработчиков;
- Каждый кейс можно взять и применить у себя уже завтра;

План



- **Passenger → Puma**
  потребление памяти сервером приложений
- **DelayedJob**
  многопоточность и многопроцессность
- **Worker Killer**
  семь бед - один ответ
- **SAX парсеры**
- **Data Streaming**
  энумераторы в энумераторах

1. Смена сервера приложений

Phusion Passenger

Puma

1. Смена сервера приложений

Phusion Passenger

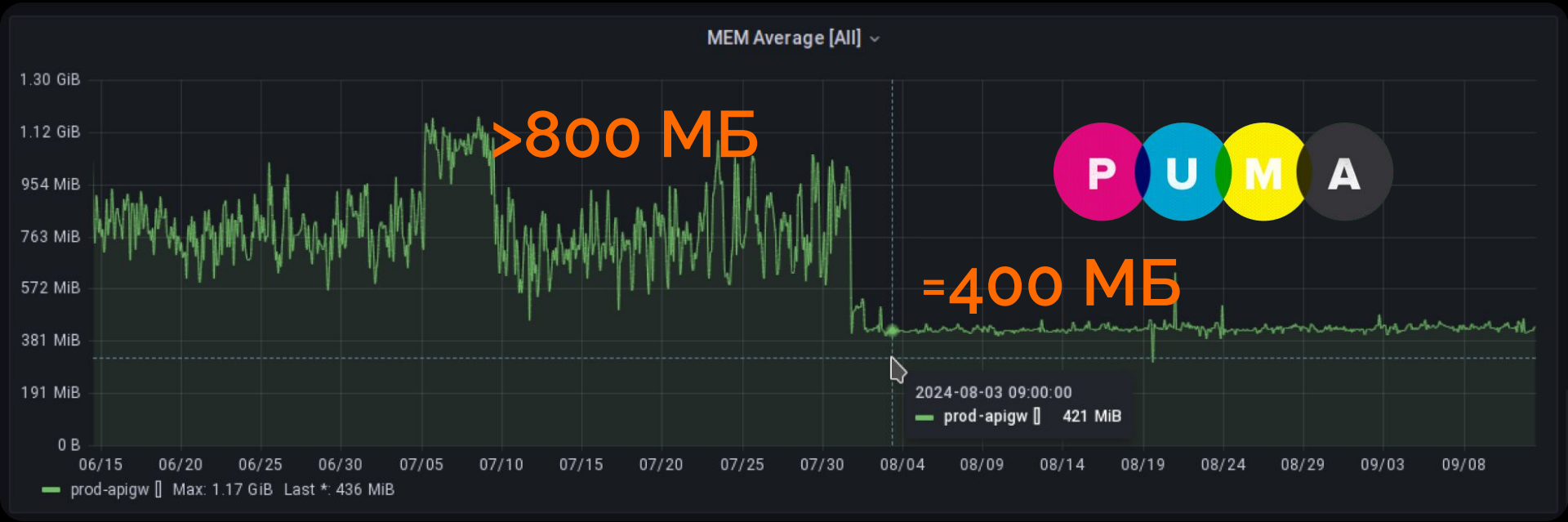
- Prefork сервер
- Copy on write
- Nginx под капотом
- Многопроцессность
- Многопоточность только в Enterprise!

1. Смена сервера приложений

Puma

- Single / Cluster / Preload
- Copy on write
- Nio4r C-ext под капотом
- Многопроцессность
- Многопоточность

1. Смена сервера приложений



1. Смена сервера приложений

Falcon

- Он хорош
- Async вообще хорош
- НЕ умеет в graceful restart, а для нас это критично

2. Фоновые задачи

- ★ Delayed Job
- Solid Queue
- Sidekiq

2. Фоновые задачи

Это надо?

- Гибкое управление очередями
- Fair distribute
- Приоритизация
- Reverse order queue (lifo)

2. Фоновые задачи

Delayed Job

- Однопоточный - для гибкого управления очередями надо запускать контейнер
- Fair distribute
- Приоритизация
- Reverse order queue (lifo)
- Каждый процесс - это память, даже если он ничего не делает
- gem delayed_job_worker_pool
- Свой 🚲 патч! 🙈

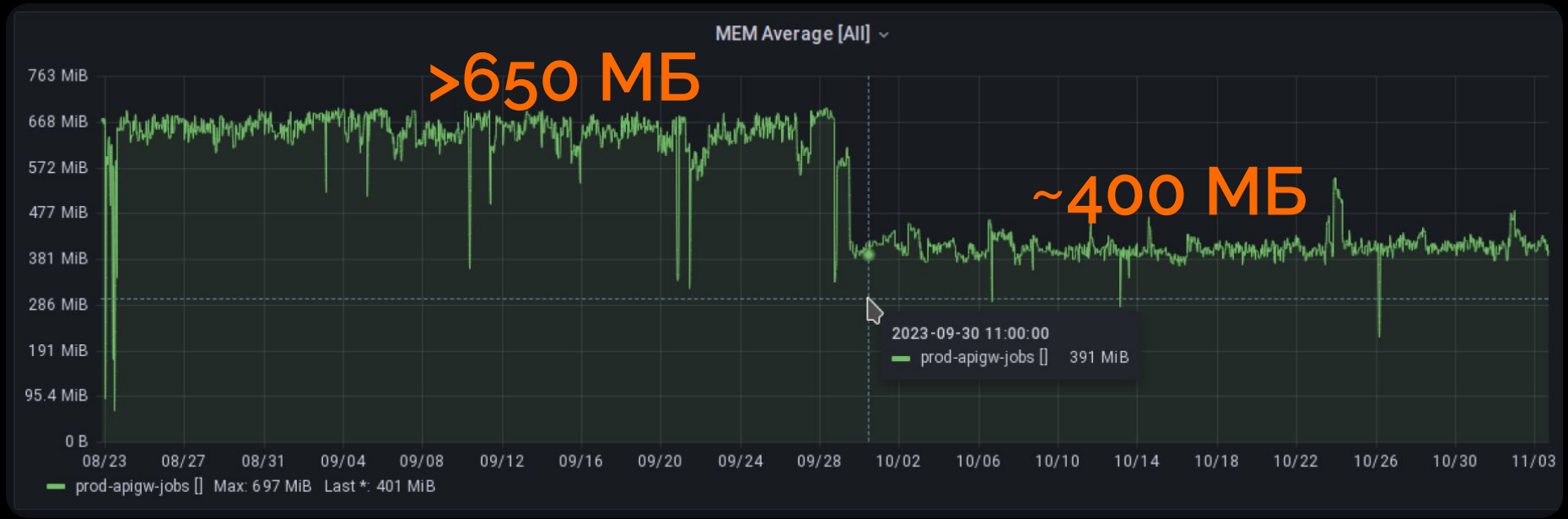
2. Фоновые задачи

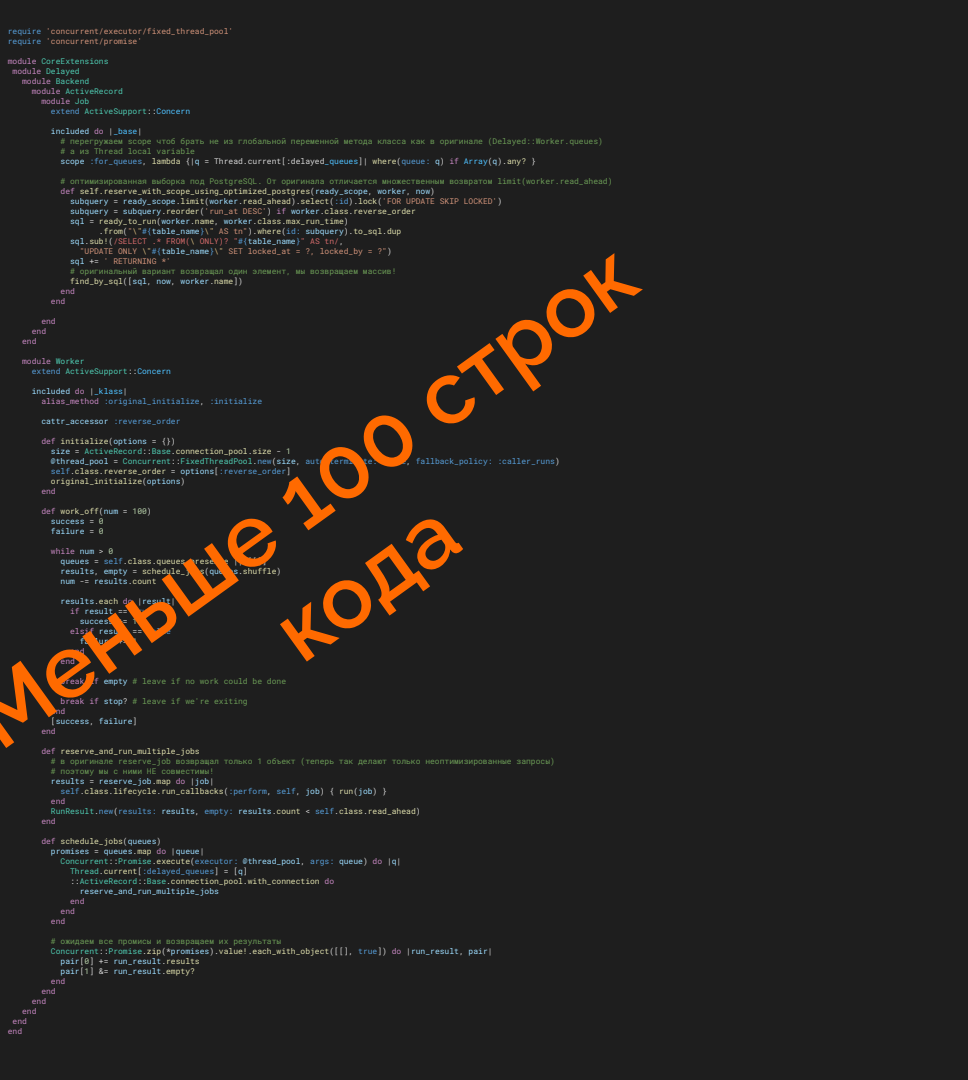
DelayedJob

- Сделали в потрохах пул потоков
- Сделали нормальный prefetch для при работе с AR, с использованием SELECT FOR UPDATE
- Запустили!

2. Фоновые задачи

Многопоточный Delayed Job




Меньше 100 строк
кода

2. Фоновые задачи

Что есть?

- Есть несколько форков, например Betterment/delayed
- Solid Queue - всё впереди!

3. Worker Killer

Утечки

Память действительно нужна!

Фрагментация

3. Worker Killer

Почему потребление памяти это плохо?

Оркестрация и расчёт ресурсов - где запускать?

soft_limit и hard_limit в оркестраторе - когда убивать?

Где спасение?

3. Worker Killer

Где спасение?

- Если приложение течёт - надо исправлять :)
- Gc.start
- Паттерн Worker Killer - периодический рестарт

Google

ruby worker killer

Все Картинки Видео Новости Покупки Книги Веб-версия Ещё Инструменты

GitHub
https://github.com › kzk › u... · Перевести эту страницу

kzk/unicorn-worker-killer: Automatically restart ...

unicorn-**worker-killer** gem provides automatic restart of Unicorn **workers** based on 1) max number of requests, and 2) process memory size (RSS), without affecting ...

GitHub
https://github.com › puma_w... · Перевести эту страницу

Automatically restart Puma cluster workers based on max ...

Puma **worker killer** can only function if you have enabled cluster mode or hybrid mode (threads + **worker** cluster). If you are only using threads (and not **workers**) ...

RubyGems
https://rubygems.org › gems · Перевести эту страницу

unicorn-worker-killer 0.4.5

13 апр. 2021 г. — The **Ruby** community's gem hosting service. Instantly publish your gems and then install them. Use the API to find out more about available gems.

GitFlic
https://gitflic.ru › project › w... · Перевести эту страницу

rndsoft/worker-killer: Kill any workers by memory and

worker-killer gem provides automatic restart of Web-server and/or background job processor based on 1) max number of requests, and 2) process memory size (RSS).

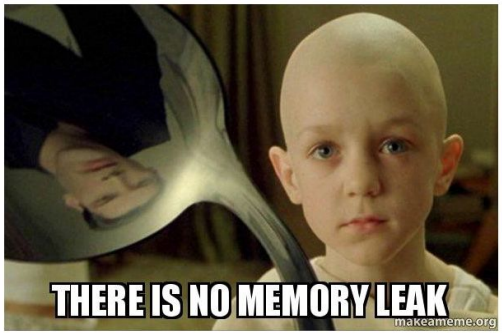
Stack Overflow

RNDSOFT Technology Blog
@rnds

rails
9 апреля

There is no memory leak! Again!

Утечки ресурсов и/или памяти, а также её фрагментация являются обычной проблемой для всех языков программирования. Неважно есть там сборщик мусора или нет, компилируемый язык или интерпретируемый. Ruby не является исключением и сегодня мы немного поговорим про эти проблемы, варианты их решения и даже напишем своё собственное.



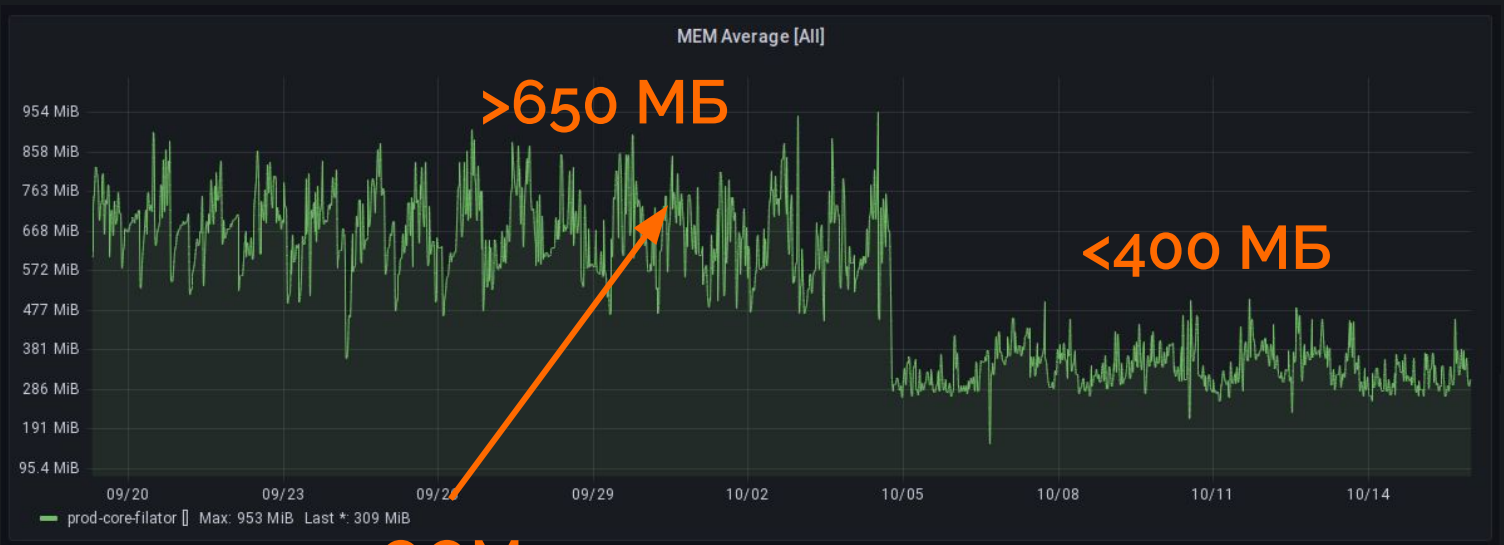
3. Worker Killer

Паттерн

- Zero cost restart
- Поддержка разных подсистем:
  - Phusion Passenger
  - Puma
  - Delayed Job
  - Sidekiq
  - Solid Queue

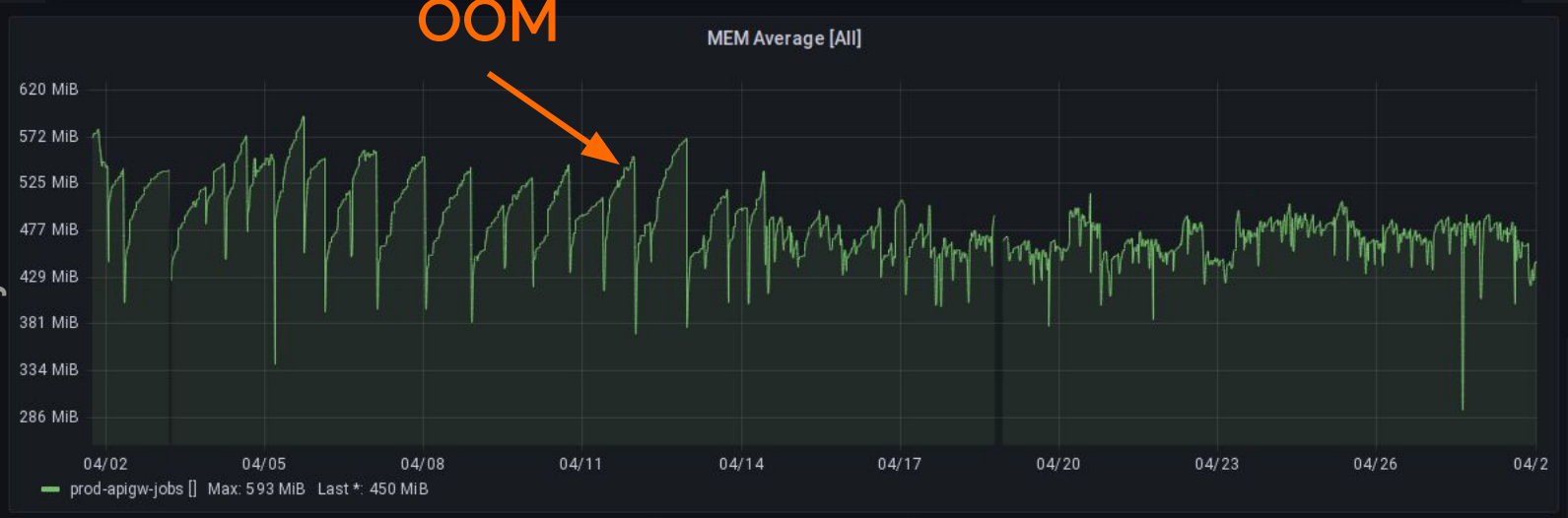
HTTP

(Phusion Passenger)



Jobs

(Delayed::Job)



4. SAX Парсеры

XML

JSON

4. SAX Парсеры

Зачем грузить полностью?

- Документ в памяти занимает ОЧЕНЬ много места
- 80mb JSON ➜ **600**mb MEM

4. SAX Парсеры

```ruby
xml_io = File.open('huge.xml')

result_data = SimpleXmlExtractor.new(xml_io).extract('Person.*.Id', 'Person.*.DocType', 'Person.*.DocDate')

result_data => {
  'Person.*.Id'       => [15,            20,          35],
  'Person.*.DocType'  => ['passport',   'drive',      'passport'],
  'Person.*.DocDate'  => ['30.09.1988', '18.03.2023', '23.03.2006'],
}
```

4. SAX Парсеры - поток

```
xml_io = File.open('huge.xml')

result_stream = SimpleXmlExtractor.new(xml_io).extract_stream('Person.*.Id', 'Person.*.DocType',
'Person.*.DocDate')

result_stream.next => ['Person.*.Id', ['Users', 'Person', 'Document', 'Id'], 15]
result_stream.next => ['Person.*.DocType', ['Users', 'Person', 'Document', 'DocType'], 'passport']
result_stream.next => ['Person.*.DocDate', ['Users', 'Person', 'Document', 'DocDate'], '30.09.1988']

result_stream.next => ['Person.*.Id', ['Users', 'Person', 'Document', 'Id'], 20]
result_stream.next => ['Person.*.DocType', ['Users', 'Person', 'Document', 'DocType'], 'drive']
result_stream.next => ['Person.*.DocDate', ['Users', 'Person', 'Document', 'DocDate'], '18.03.2023']

...
...
```

4. SAX Парсеры

Плюсы и минусы

- Полностью отсутствует потребление памяти
- ВОЗМОЖНО оно чуть медленнее будет работать
- Нельзя пройтись по данным несколько раз
- Иногда бывает только одна попытка

5. Data Streaming

Enumerators

Generators

Формирование “на лету”

5. Data Streaming

- Огромная таблица в БД
- Нам надо выгрузить данные
- Формат CSV
- Результат CSV превышает 6GB
- Никаких временных файлов!

Пример

SQL ➔ CSV ➔ ZIP ➔ S3 upload

5. Data Streaming - БД и CSV

```
def csv_stream
 sql = Request.where('created_at < ?', Time.now).to_sql
 query = "COPY (#{sql}) TO STDOUT WITH (FORMAT CSV, HEADER
TRUE);"

 Enumerator.new do |y|
   conn = ActiveRecord::Base.connection.raw_connection
   conn.copy_data(query) do
     while (chunk = conn.get_copy_data)
       y << chunk
     end
   end
 end
end

stream = csv_stream() # вот наш поток
```

5. Data Streaming - Сжатие

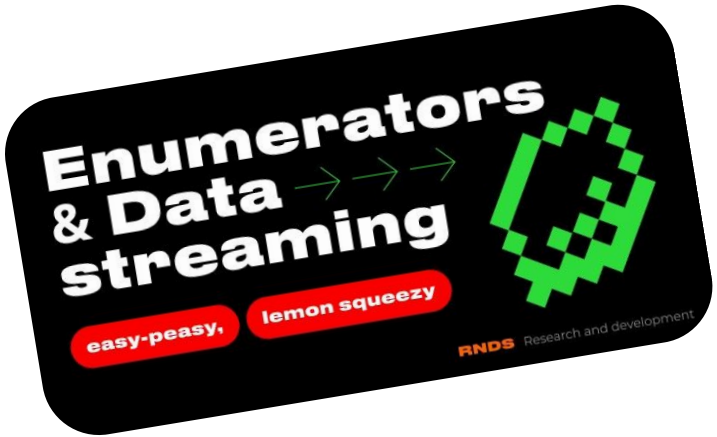
```
def zip_stream(stream)
 ZipTricks::Streamer.output_enum do |zip|
   file = "requests.csv"
   zip.write_deflated_file(file) do |sink|
     stream.each do |line|
       sink.write(line)
     end
   end
 end
end

stream = zip_stream(csv_stream)
```

5. Data Streaming - Upload

```
def upload_stream(stream, filename)
 uri = URI('http://local.http.storage')

 # преобразуем интерфейс потока
 # добавляем ему методы типа filename и read (который на самом деле each)
 io = ReadableEnumerator.new(stream, filename)
  request = Net::HTTP::Post.new(uri, {'Transfer-Encoding' => 'chunked'})
 form_data = [['body', io, { filename: filename }]]
 request.set_form form data, 'multipart/form-data'

 response = Net::HTTP.start(uri.hostname, uri.port, use_ssl: false) do
|http|
   http.request(request) # наконец! Тёпленькая пошла!
 end

 JSON.parse(response.body)
end

ok = upload_stream(zip_stream(csv_stream), 'requests.zip')
```

5. Data Streaming

Итог



- Нет потребление памяти
- Однопроходность
- Моя статья “Enumerators, Data Streaming и другие модные слова в Ruby”

- Не надо бояться “патчить” или менять существующие решения
- Всегда можно найти решение лучше и проще
- Можно покодить для души и принести неплохую пользу

Всегда рады вас видеть

RNDSOFT

Спасибо

t.me/jerry_ru

https://github.com/RND-SOFT

blog.rnds.pro

RNDSOFT

Оцените доклад

Материалы доклада