RZESZÓW
RUBY
USER
GROUP

Let’s cache it

Ruby on Rails cache in examples

RRUG #34, 24.04.2023

Mateusz Utkała

Agenda

- What is caching
- Caching strategies
- Rails cache stores
- Examples
- Tips

There are only two hard things in Computer Science:
***cache invalidation** and **naming things**.*

PHIL KARLTON

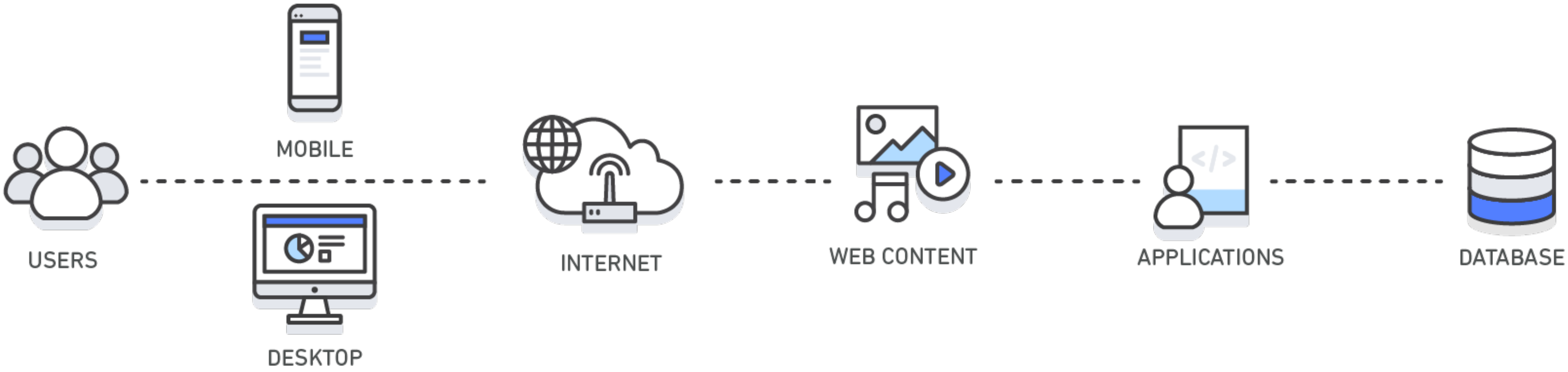
What is caching

Cache is a high-speed data storage layer which stores a subset of data, so that future requests for that data are served up faster than is possible by accessing the data's primary storage location.

Caching allows you to efficiently reuse previously computed data.

https://makeameme.org/meme/cache-cache-everywhere

Cache is everywhere



HTTP Cache Headers
Browser Cache

DNS

HTTP Cache Headers
CDNs, Reverse Proxies

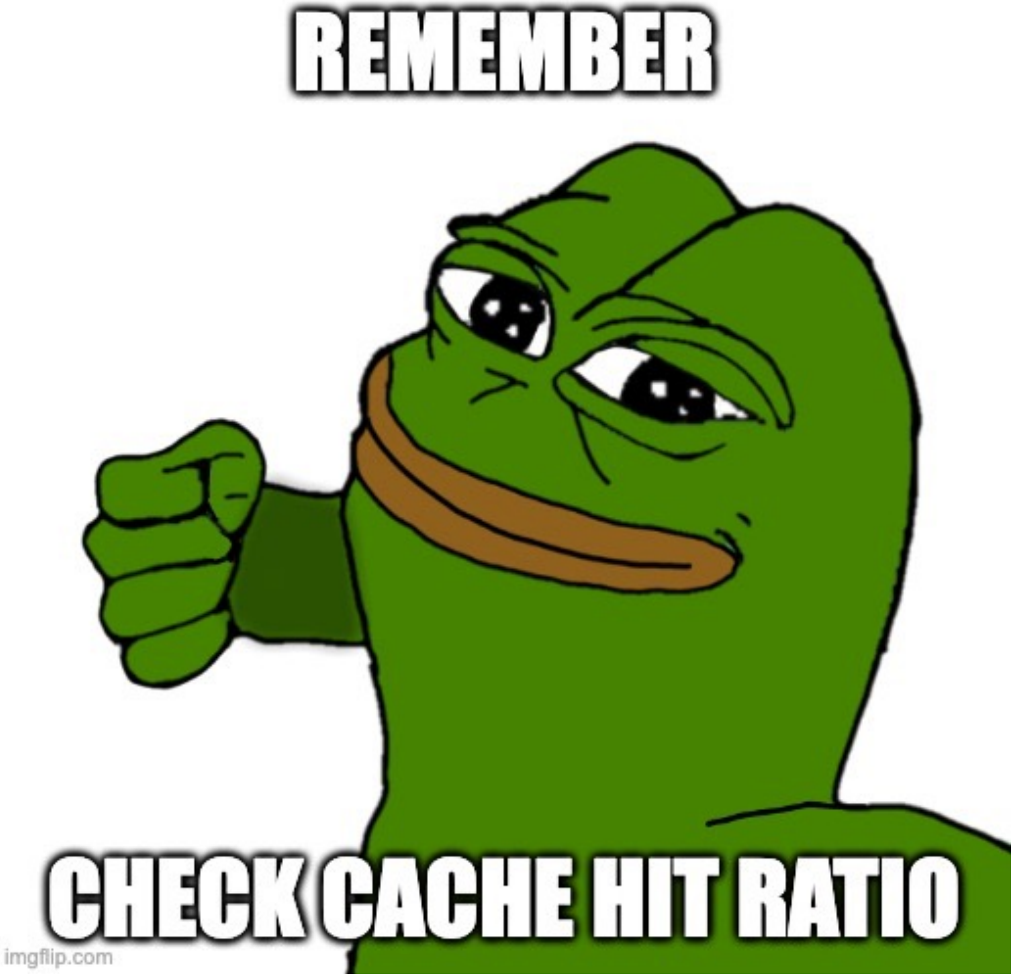
Key/Value stores
Local caches

Key/Value data stores
Database buffers

https://aws.amazon.com/caching/

Caching strategies – What to cache?

- Long running requests / queries
- Havy calculations
- Domain aggregated data
- Rarely changed data
  - Configurations



https://imgflip.com/i/7aj2lr

Caching strategies – What to cache?

$$\frac{\text{Number of cache hits}}{(\text{Number of cache hits} + \text{Number of cache misses})} = \text{Cache hit ratio}$$

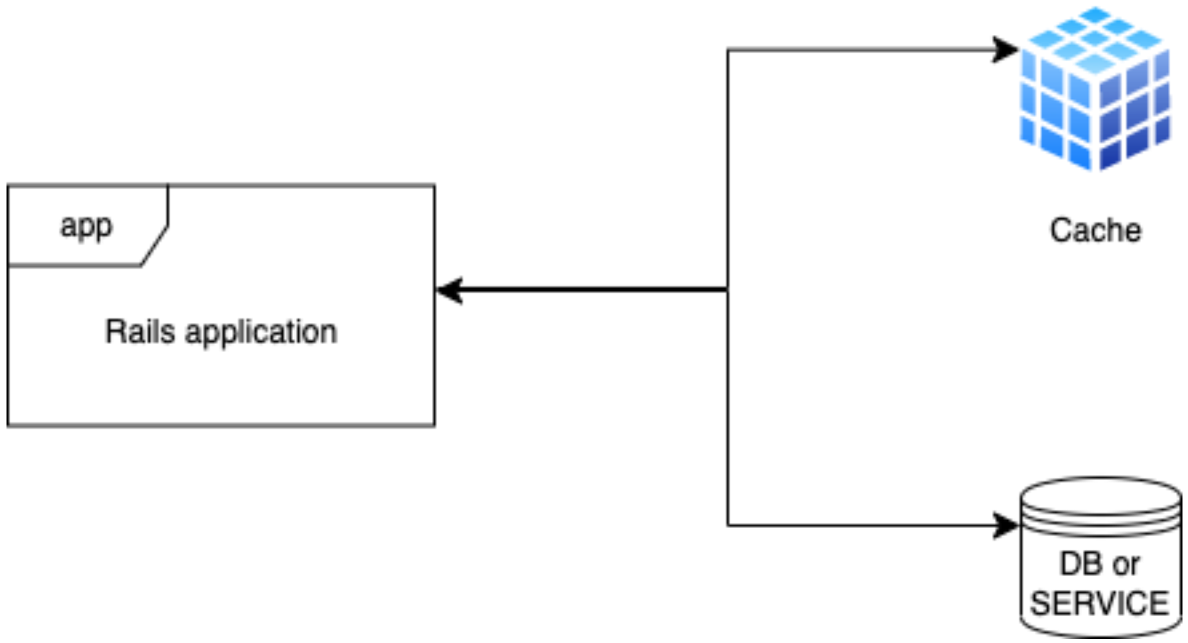
Caching strategies – Where to cache?

- In service memory
- Standalone service
- Distributed service

Caching strategies – When to cache?

- On Demand
- Pre-loading
- Lazy load

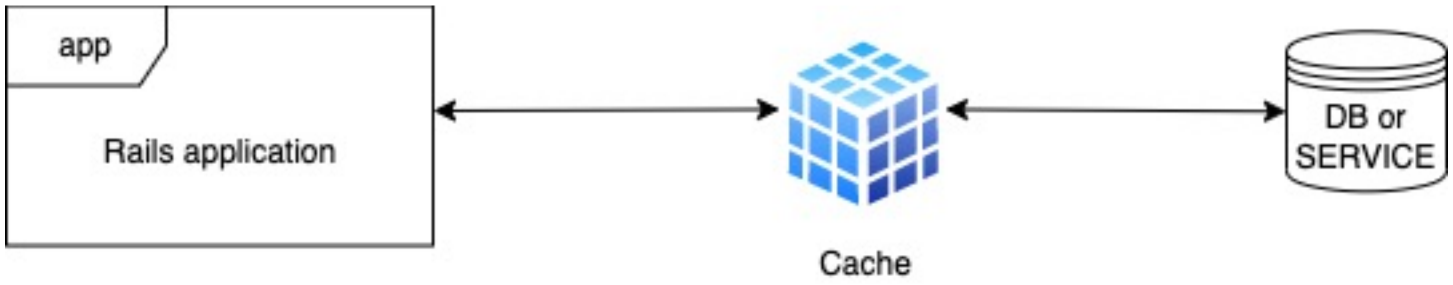
Caching strategies – How to cache?



Cache aside

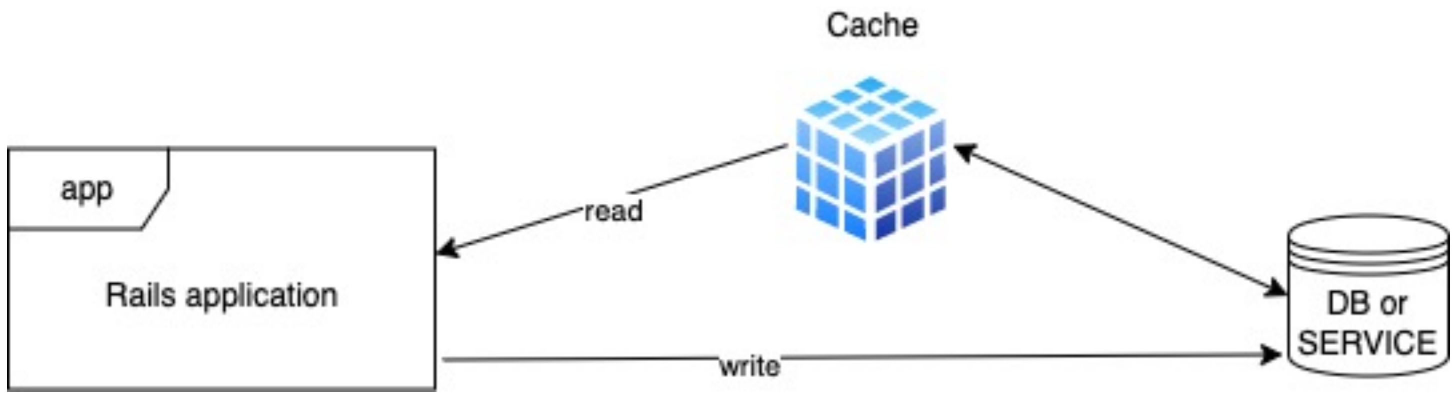
Caching strategies – How to cache?

Read/Write through



Caching strategies – How to cache?

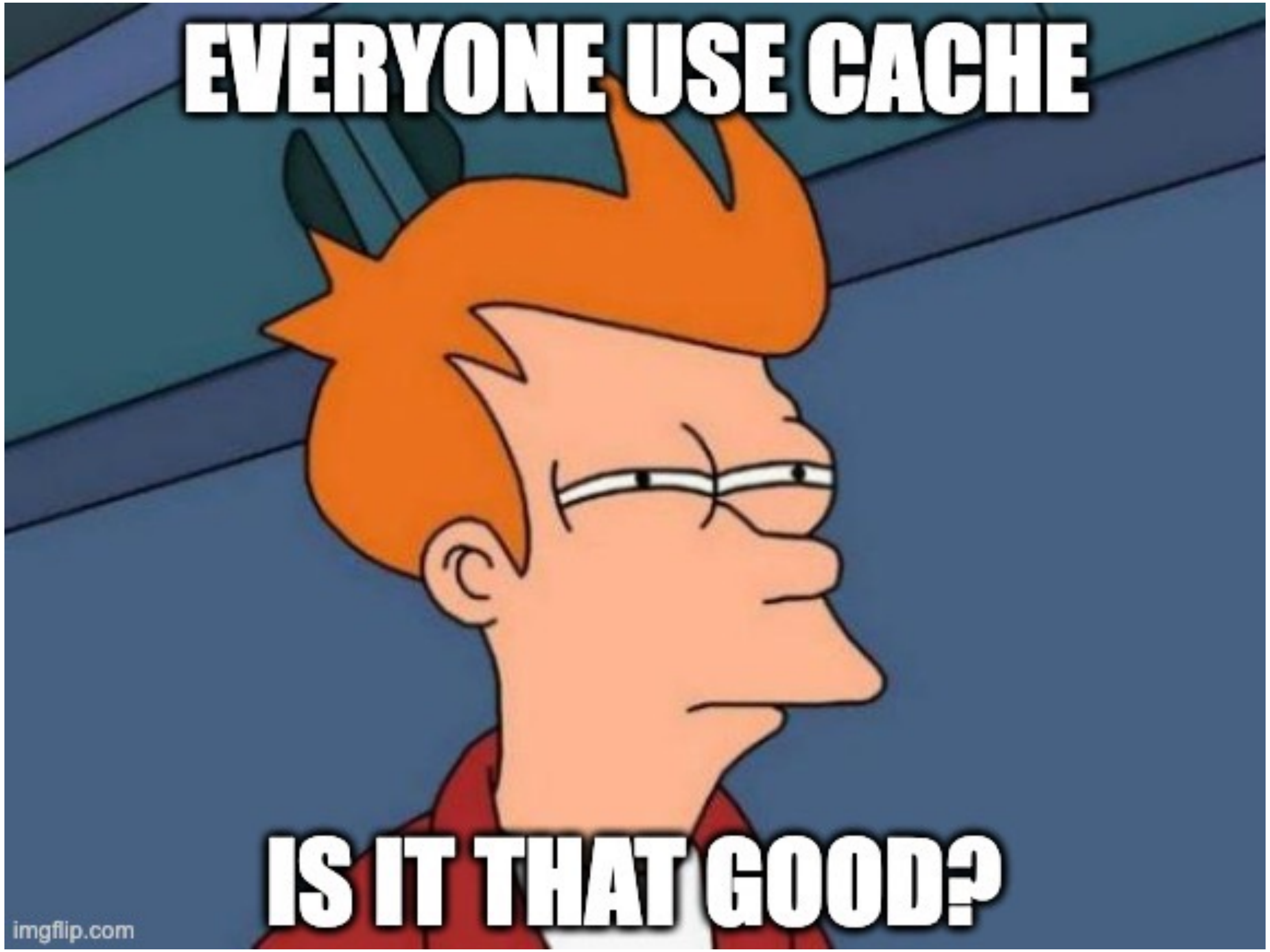
Read through



Caching strategies – How long to cache?

Never treat your cache as your database
even if it super reliable with great clustering features

Is it that good?



https://imgflip.com/i/7aiomg

Is it that good?

PROS

✓ Accelerate data retrieval

✓ Reduce the load on backend services

✓ Fast respond to the user = good user experience

✓ Respond to a lot of users = easy to scale

CONS

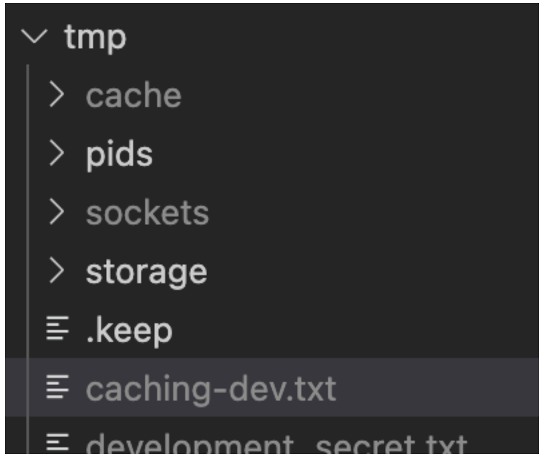
○ Cache invalidation problem (stale data)

○ Storage consumption

○ Security risks

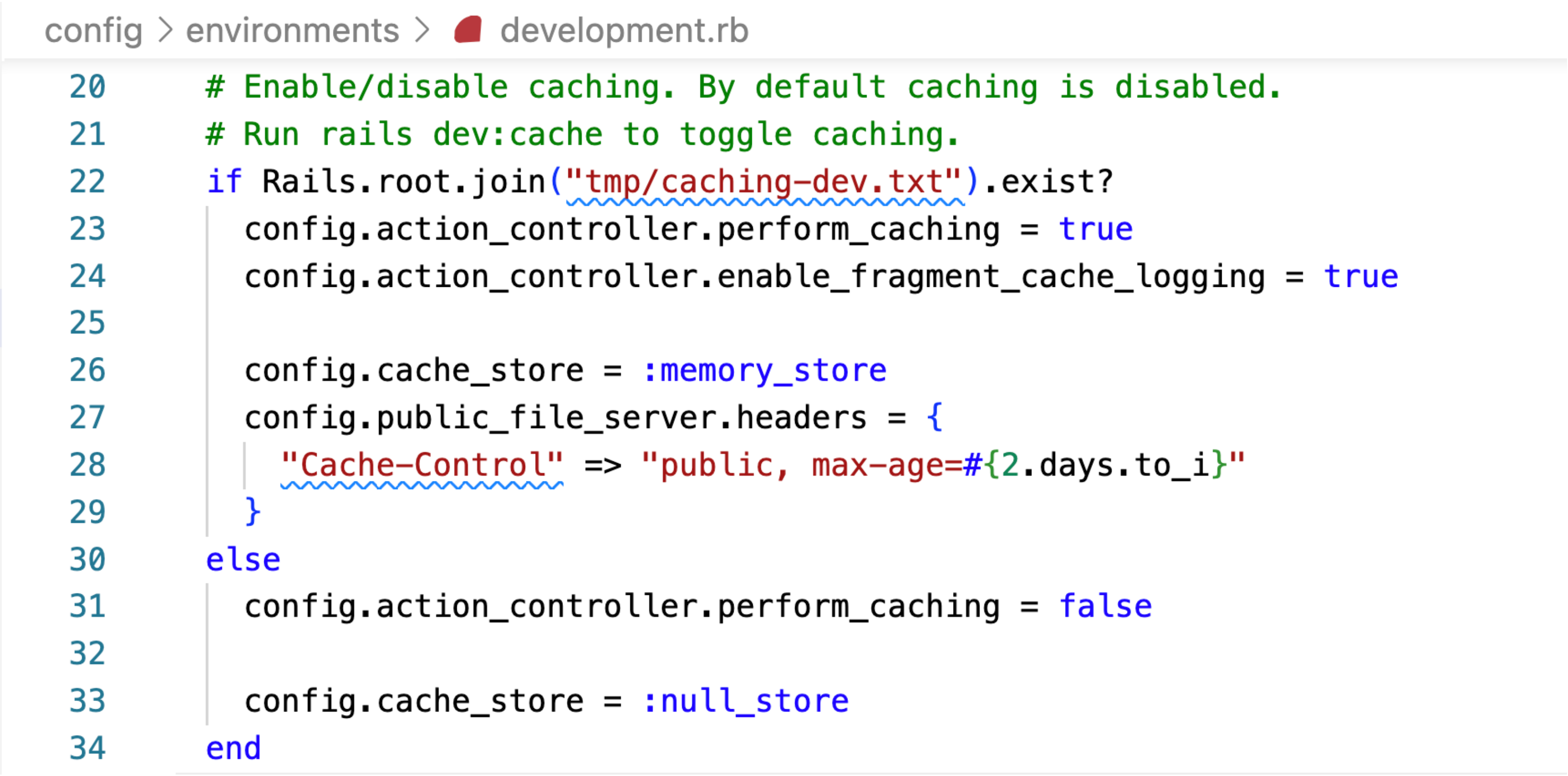
Rails cache

rails dev:cache



```
tmp
  cache
  pids
  sockets
  storage
  .keep
  caching-dev.txt
  development_secret.txt
```

Rails cache



```ruby
  # Enable/disable caching. By default caching is disabled.
  # Run rails dev:cache to toggle caching.
  if Rails.root.join("tmp/caching-dev.txt").exist?
    config.action_controller.perform_caching = true
    config.action_controller.enable_fragment_cache_logging = true

    config.cache_store = :memory_store
    config.public_file_server.headers = {
      "Cache-Control" => "public, max-age=#{2.days.to_i}"
    }
  else
    config.action_controller.perform_caching = false

    config.cache_store = :null_store
  end
```

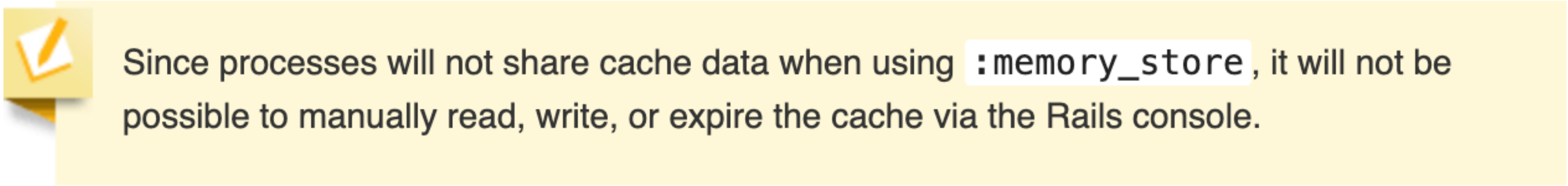
Rails cache store

ActiveSupport::Cache::Store

The main API methods are **read, write, delete, exist?**, and **fetch**.

Rails memory store

```
config.cache_store = :memory_store, { size: 32.megabytes }
```



Since processes will not share cache data when using `:memory_store`, it will not be possible to manually read, write, or expire the cache via the Rails console.

Rails file store

```
config.cache_store = :file_store, "#{root}/tmp/cache/"
```

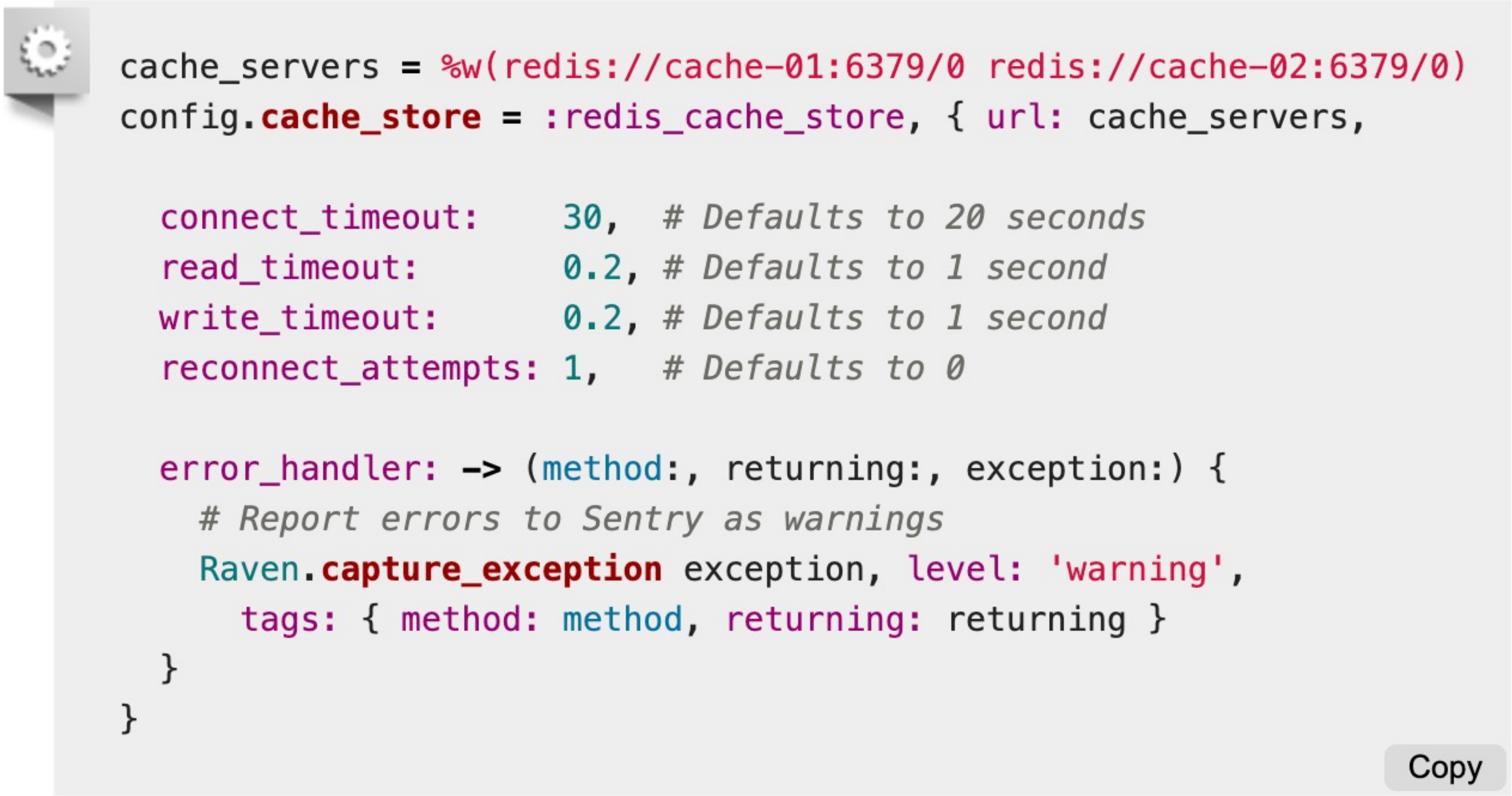
As the cache will grow until the disk is full, it is recommended to periodically clear out old entries.

Rails memcached store

Bundled ’dalli gem’

```
config.cache_store = :mem_cache_store, "cache-1.com”, "cache-2.com”
```

Rails redis store

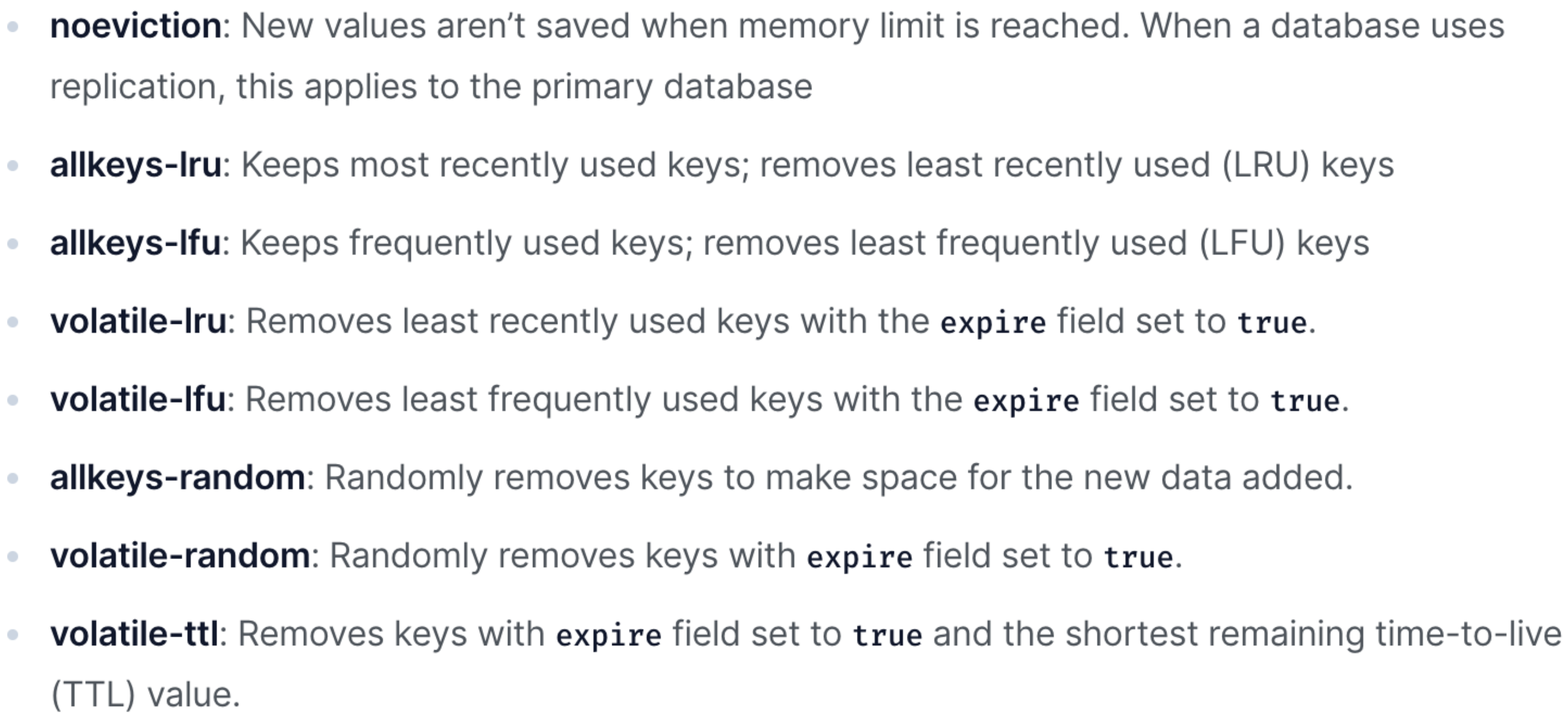


```ruby
cache_servers = %w(redis://cache-01:6379/0 redis://cache-02:6379/0)
config.cache_store = :redis_cache_store, { url: cache_servers,

  connect_timeout:    30,  # Defaults to 20 seconds
  read_timeout:       0.2, # Defaults to 1 second
  write_timeout:      0.2, # Defaults to 1 second
  reconnect_attempts: 1,   # Defaults to 0

  error_handler: -> (method:, returning:, exception:) {
    # Report errors to Sentry as warnings
    Raven.capture_exception exception, level: 'warning',
      tags: { method: method, returning: returning }
  }
}
```

https://guides.rubyonrails.org/caching_with_rails.html#activesupport-cache-rediscachestore

Rails redis store

maxmemory-policy



- **noeviction**: New values aren't saved when memory limit is reached. When a database uses replication, this applies to the primary database
- **allkeys-lru**: Keeps most recently used keys; removes least recently used (LRU) keys
- **allkeys-lfu**: Keeps frequently used keys; removes least frequently used (LFU) keys
- **volatile-lru**: Removes least recently used keys with the `expire` field set to `true`.
- **volatile-lfu**: Removes least frequently used keys with the `expire` field set to `true`.
- **allkeys-random**: Randomly removes keys to make space for the new data added.
- **volatile-random**: Randomly removes keys with `expire` field set to `true`.
- **volatile-ttl**: Removes keys with `expire` field set to `true` and the shortest remaining time-to-live (TTL) value.

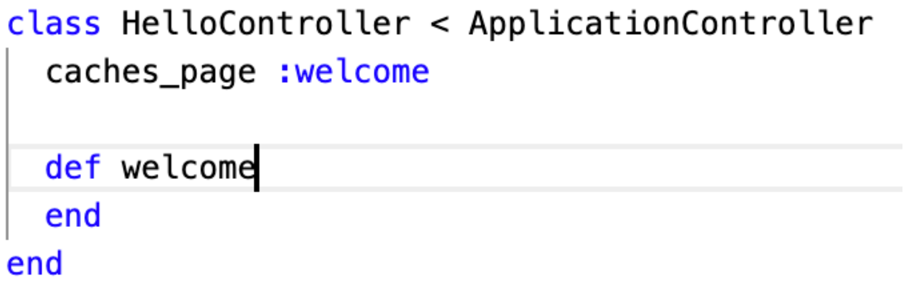
https://redis.io/docs/reference/eviction/

Rails null store

```
config.cache_store = :null_store
```

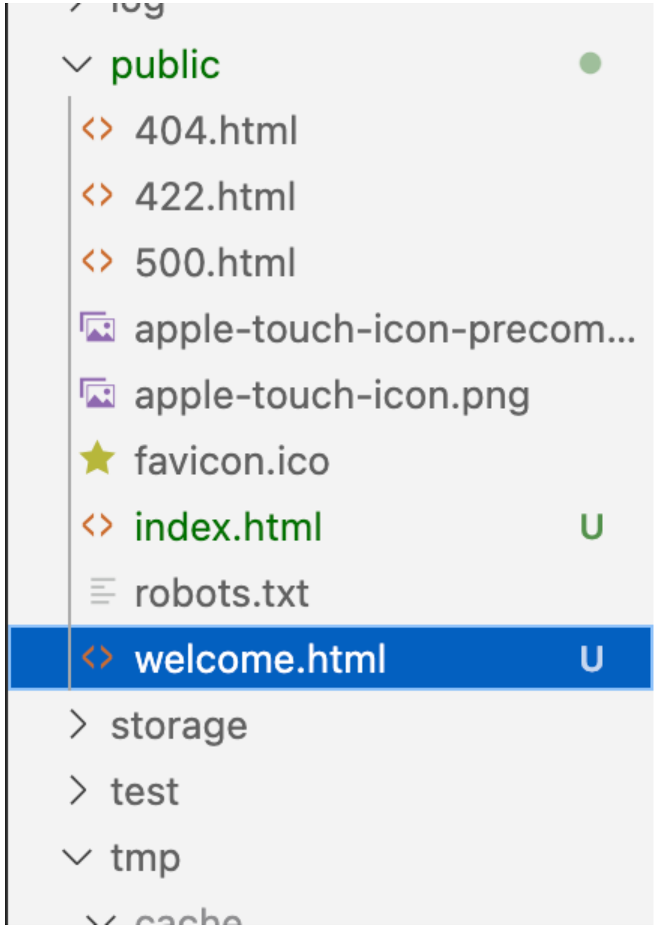
Examples – Page caching

gem "actionpack-page_caching"

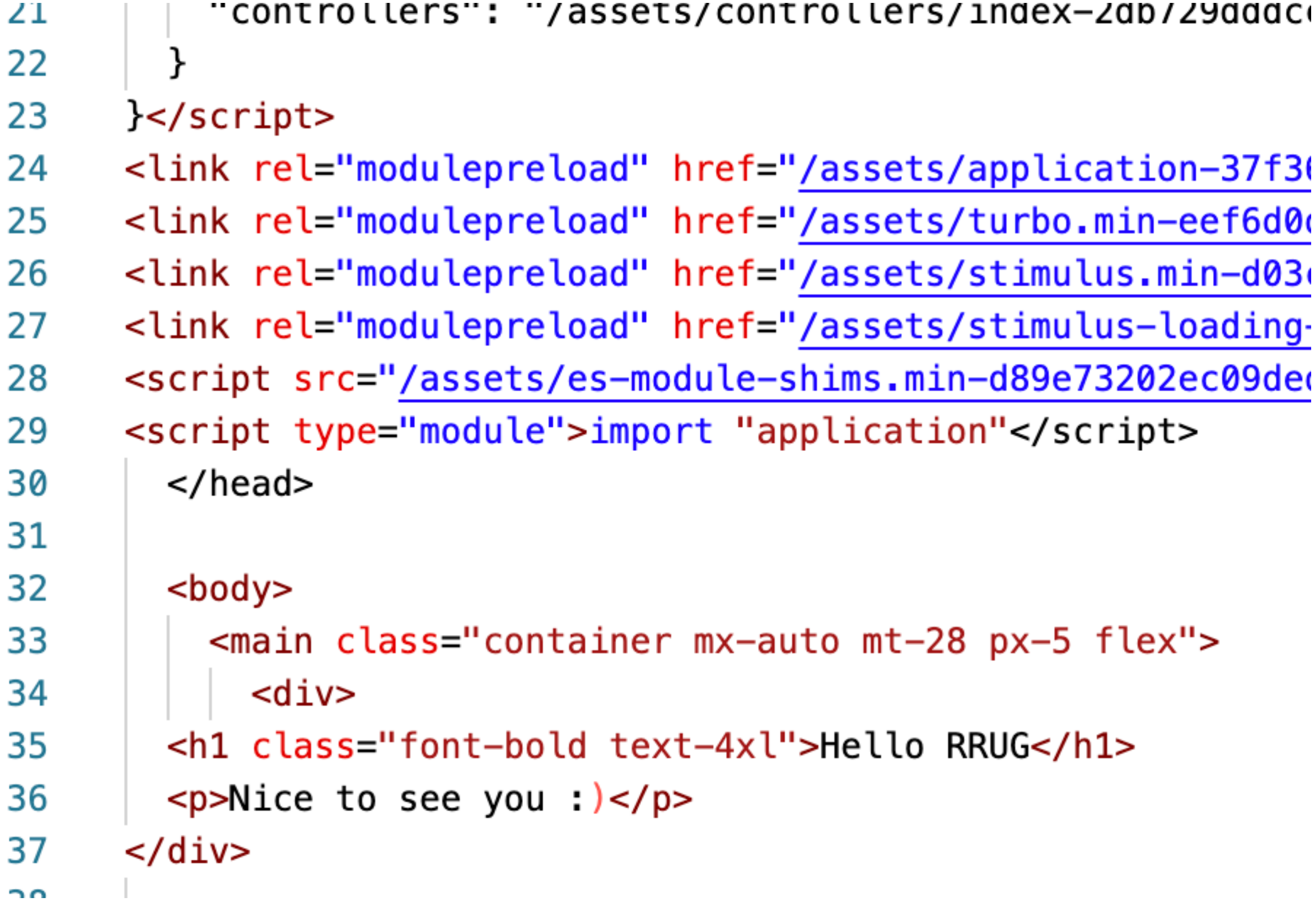


```
class HelloController < ApplicationController
  caches_page :welcome

  def welcome
  end
end
```

Examples – Page caching

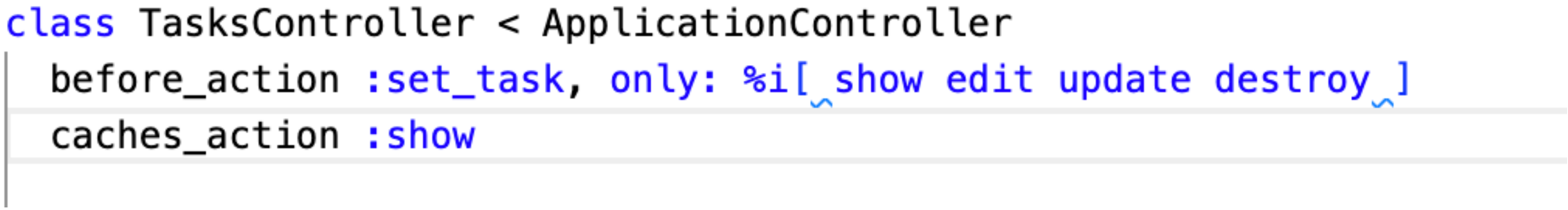


- log
- public
  - 404.html
  - 422.html
  - 500.html
  - apple-touch-icon-precom...
  - apple-touch-icon.png
  - favicon.ico
  - index.html U
  - robots.txt
  - welcome.html U
- storage
- test
- tmp
  - cache

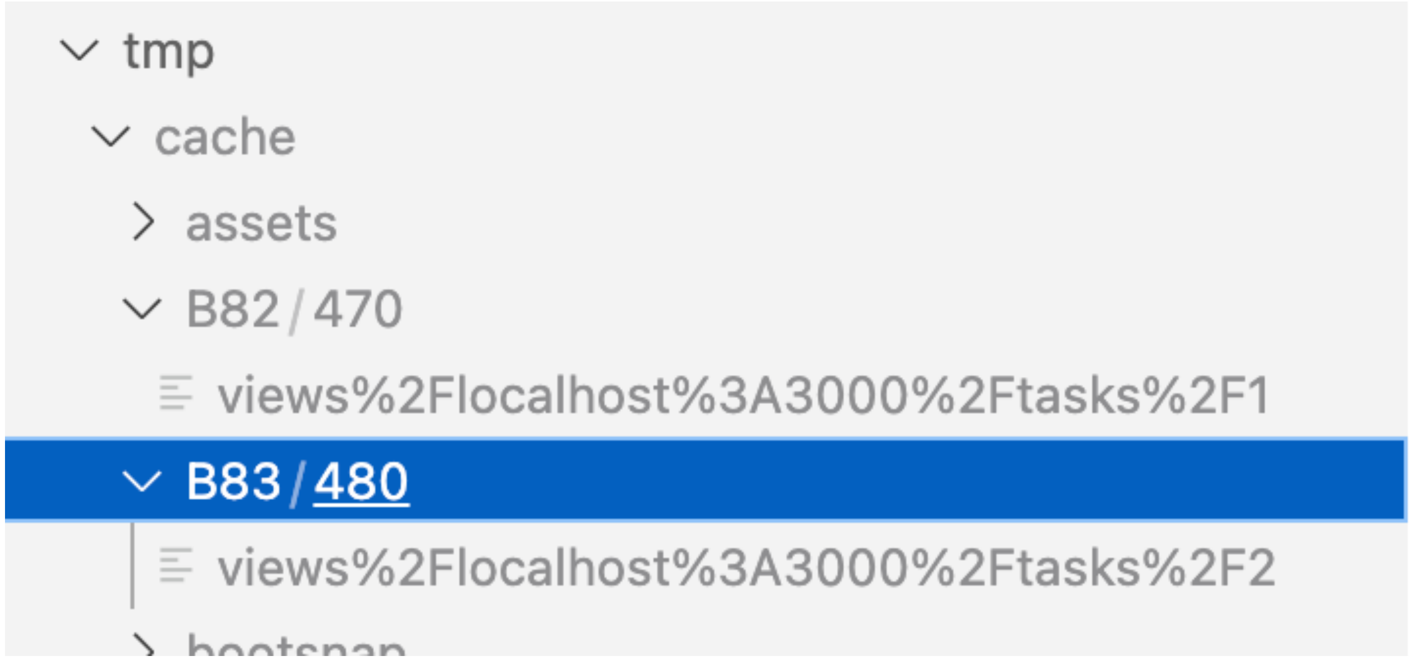


```
    "controllers": "/assets/controllers/index-2ab729ddac
  }
}</script>
<link rel="modulepreload" href="/assets/application-37f3
<link rel="modulepreload" href="/assets/turbo.min-eef6d0
<link rel="modulepreload" href="/assets/stimulus.min-d03
<link rel="modulepreload" href="/assets/stimulus-loading-
<script src="/assets/es-module-shims.min-d89e73202ec09de
<script type="module">import "application"</script>
  </head>

  <body>
    <main class="container mx-auto mt-28 px-5 flex">
      <div>
  <h1 class="font-bold text-4xl">Hello RRUG</h1>
  <p>Nice to see you :)</p>
</div>
```

Examples – Action caching

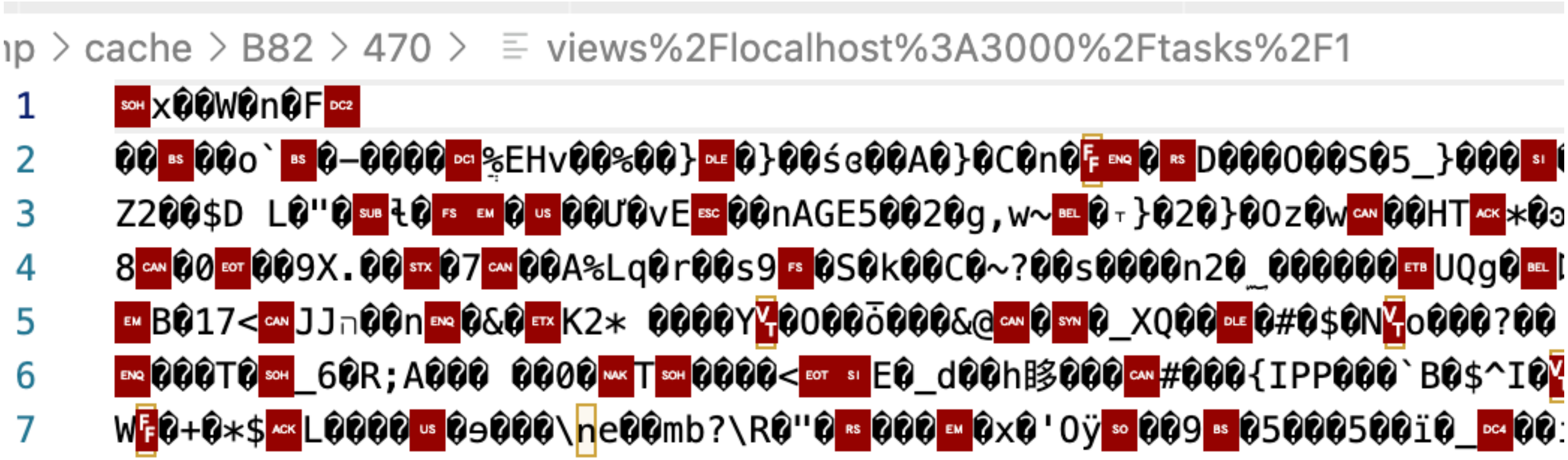


```ruby
class TasksController < ApplicationController
  before_action :set_task, only: %i[ show edit update destroy ]
  caches_action :show
```



```
tmp
  cache
    assets
    B82/470
      views%2Flocalhost%3A3000%2Ftasks%2F1
    B83/480
      views%2Flocalhost%3A3000%2Ftasks%2F2
    bootsnap
```

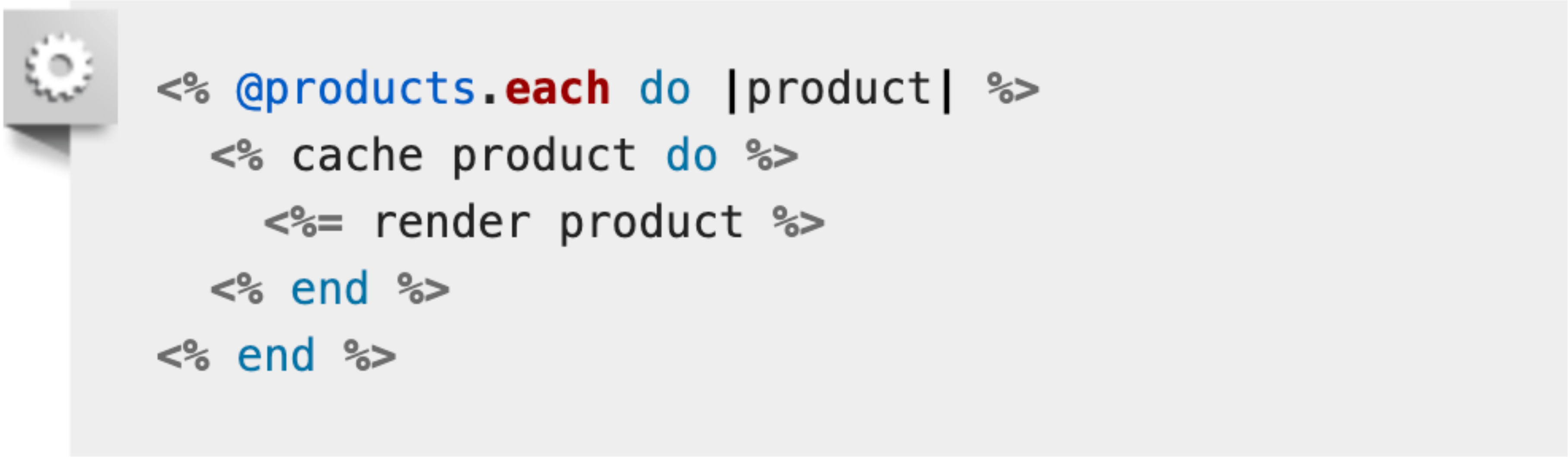
Examples – Action caching



ıp > cache > B82 > 470 > views%2Flocalhost%3A3000%2Ftasks%2F1

```
SOHx��W�n�FDC2
��BS��o`BS�–����DC1%EHv��%��}DLE�}��śɜ��A�}�C�n�FFENQ�RSD����0��S�5_}���SI�
Z2��$D L�"�SUBƚ�FS EM�US��Ư�vEESC��nAGE5��2�g,w~BEL�ᅮ}�2�}�0z�wCAN��HTACK*�ɜ
8CAN�0EOT��9X.��STX�7CAN��A%Lq�r��s9FS�S�k��C�~?��s����n2�_������ETBUQg�BEL[
EMB�17<CANJJ¬��nENQ�&�ETXK2* ����YVT�0���ō����&@CAN�SYN�_XQ��DLE�#�$�NVTo����?��
ENQ���T�SOH_6�R;A��� ��0�NAKTSOH����<EOT SIE�_d��h䫛���CAN#���{IPP���`B�$^I�V
WFF�+�*$ACKL����US�ɘ���\ne��mb?\R�"�RS���EM�x�'0ÿSO��9BS�5���5��ï�_DC4��:
```

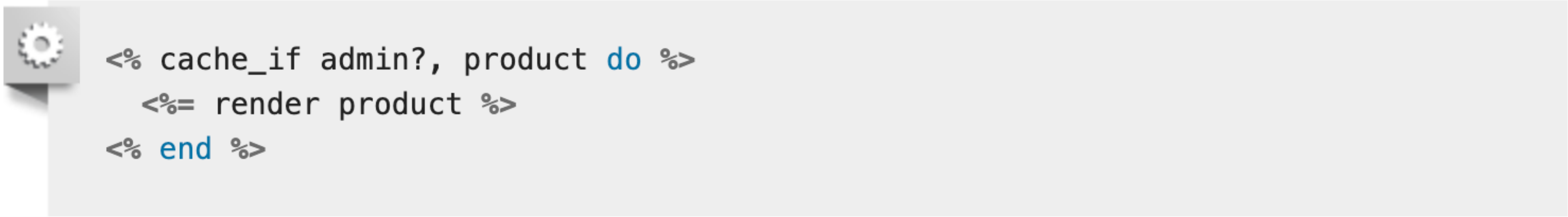
Examples – Fragment caching



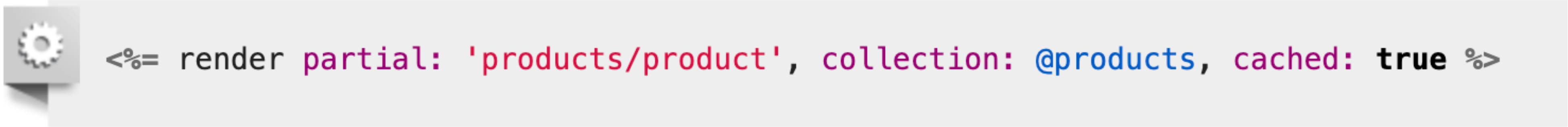
```
<% @products.each do |product| %>
  <% cache product do %>
    <%= render product %>
  <% end %>
<% end %>
```

views/products/index:bea67108094918eeba42cd4a6e786901/products/1

Examples – Fragment caching

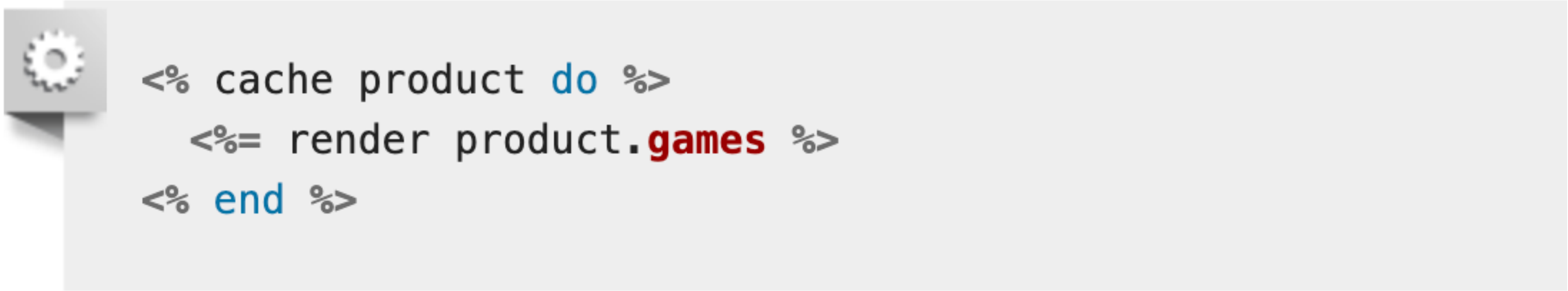


```
<% cache_if admin?, product do %>
  <%= render product %>
<% end %>
```

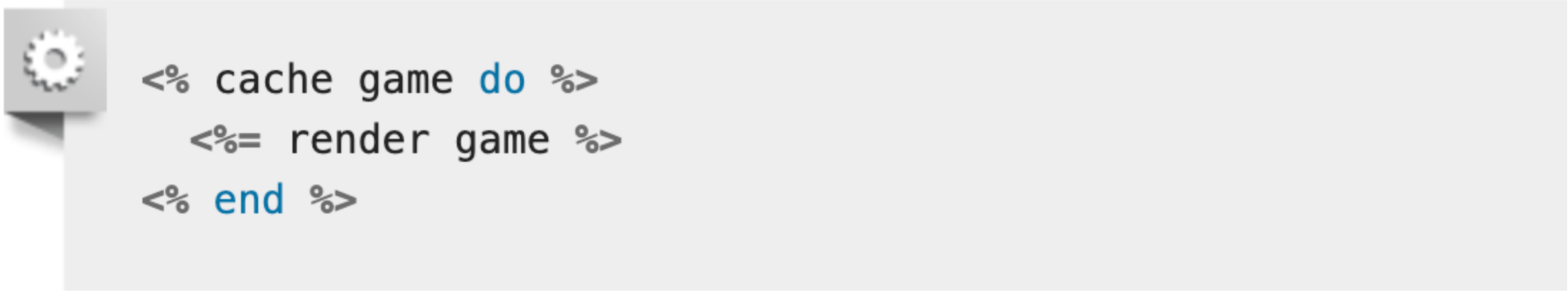


```
<%= render partial: 'products/product', collection: @products, cached: true %>
```

Examples – Nested fragment caching

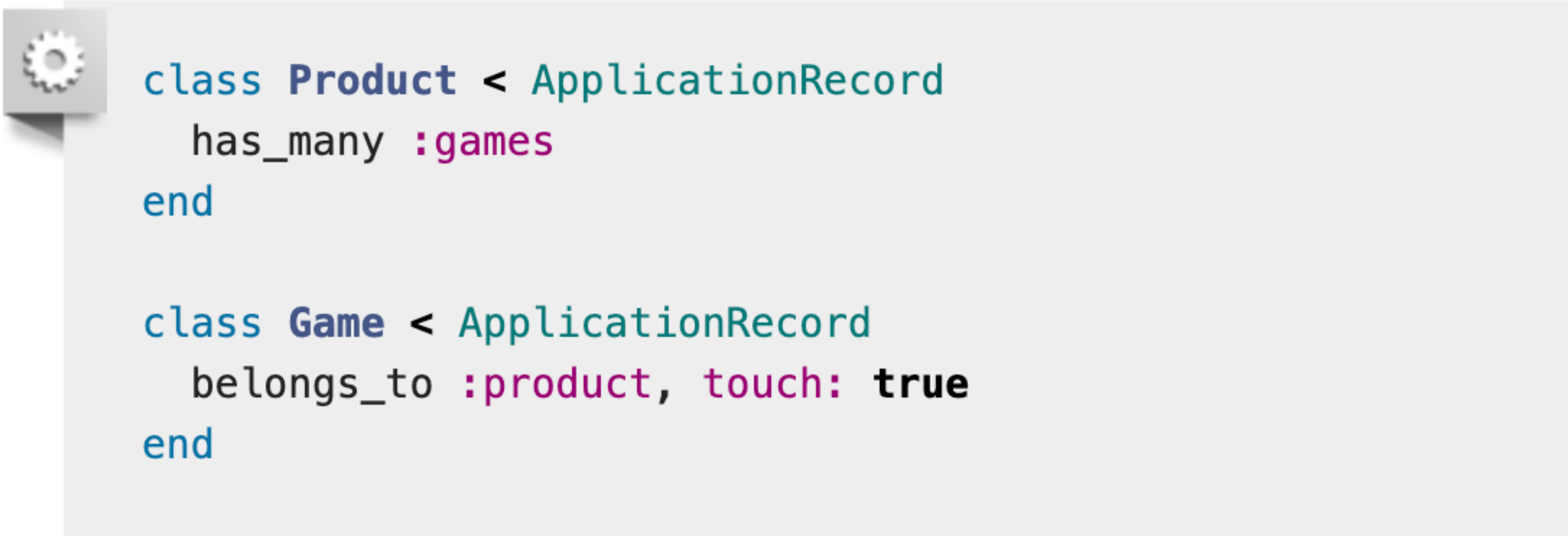


```
<% cache product do %>
  <%= render product.games %>
<% end %>
```



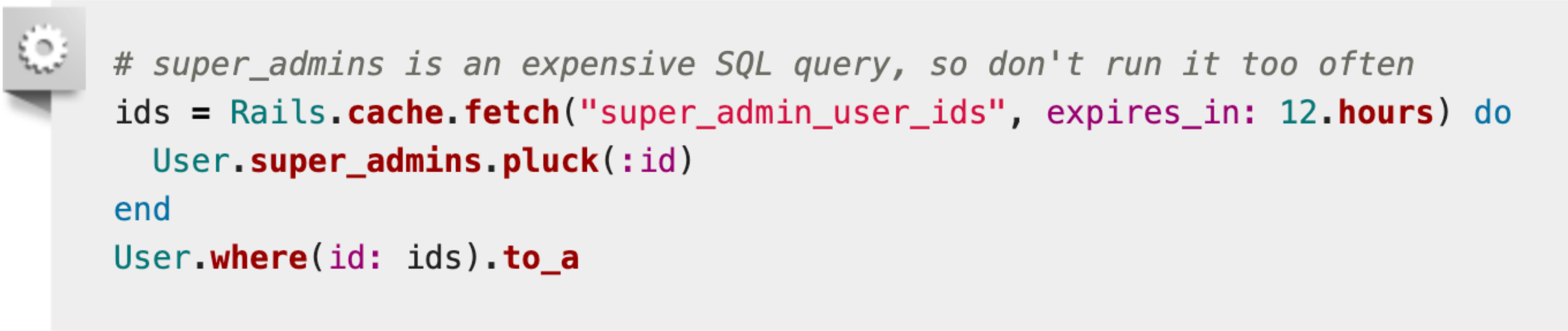
```
<% cache game do %>
  <%= render game %>
<% end %>
```

Examples – Nested fragment caching



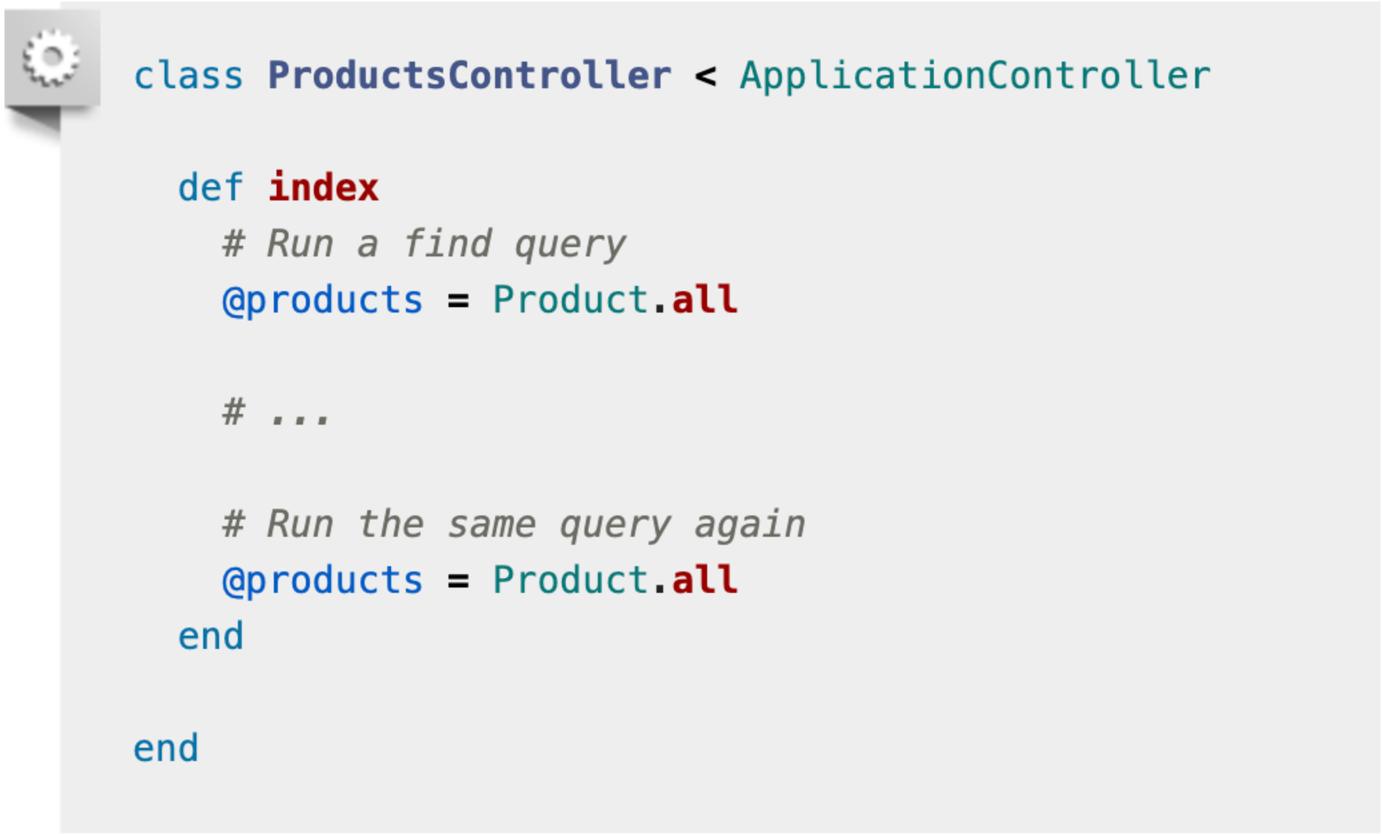
```ruby
class Product < ApplicationRecord
  has_many :games
end

class Game < ApplicationRecord
  belongs_to :product, touch: true
end
```

Examples – Low-level caching



```ruby
# super_admins is an expensive SQL query, so don't run it too often
ids = Rails.cache.fetch("super_admin_user_ids", expires_in: 12.hours) do
  User.super_admins.pluck(:id)
end
User.where(id: ids).to_a
```

Examples – SQL caching



```ruby
class ProductsController < ApplicationController

  def index
    # Run a find query
    @products = Product.all

    # ...

    # Run the same query again
    @products = Product.all
  end

end
```

Examples – SQL caching
Recyclable cache keys in Rails

Rails < 5.2

cache_key = *{model_name}/{id}-{update_at}*

”products/1-20230227080152975653"

Rails >= 5.2

cache_key = *{model_name}/{id}*

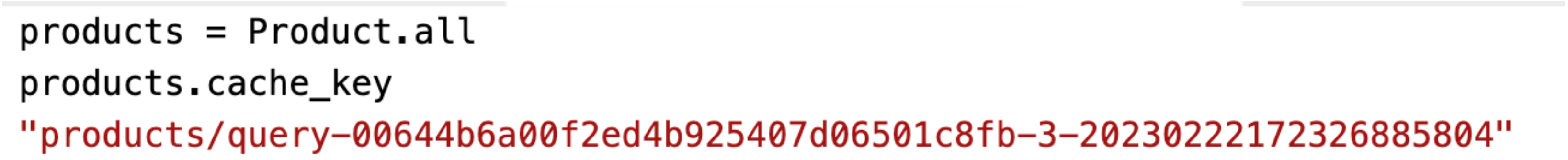
"products/1"

cache_version = *{update_at}*

"20230227080152975653"

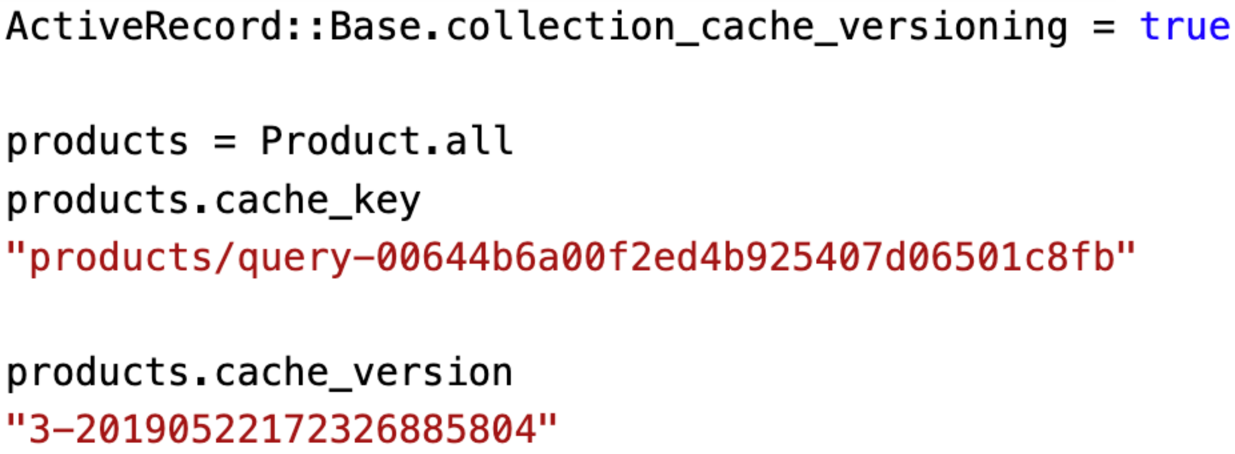
Examples – SQL caching
Collection cache versioning

Rails < 6.0



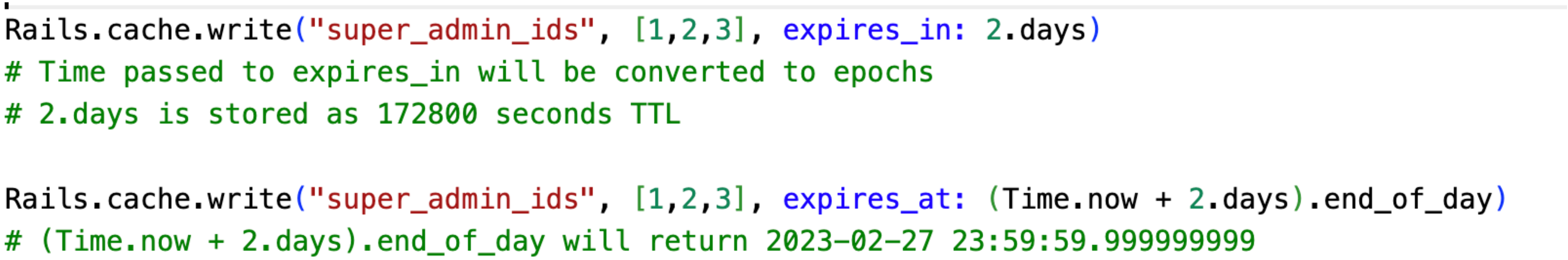
```
products = Product.all
products.cache_key
"products/query-00644b6a00f2ed4b925407d06501c8fb-3-2023022172326885804"
```

Rails >= 6.0

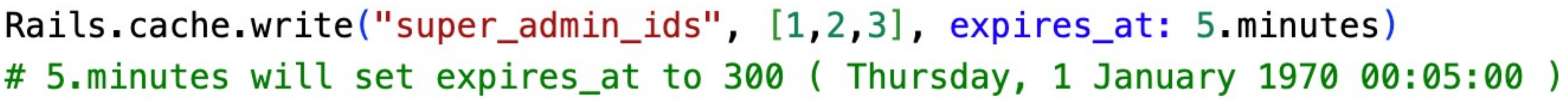


```
ActiveRecord::Base.collection_cache_versioning = true

products = Product.all
products.cache_key
"products/query-00644b6a00f2ed4b925407d06501c8fb"

products.cache_version
"3-20190522172326885804"
```

Tips – expires_in vs expires_at in Rails 7



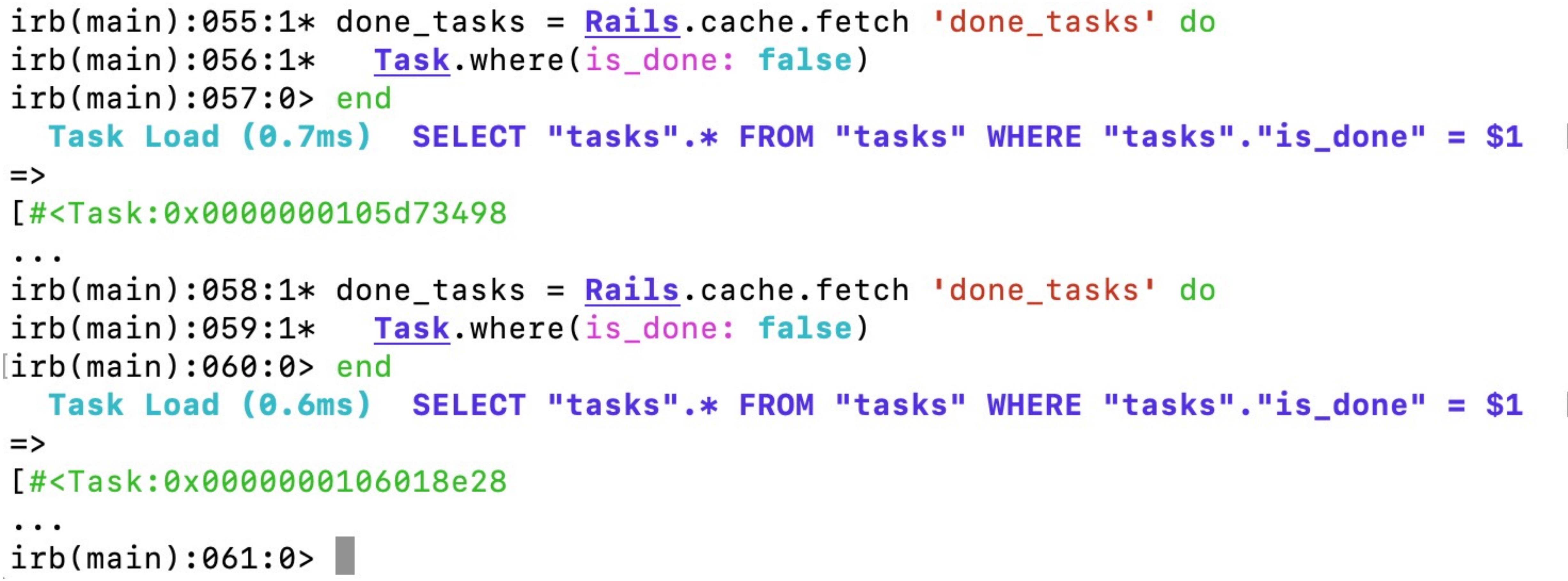
```
Rails.cache.write("super_admin_ids", [1,2,3], expires_in: 2.days)
# Time passed to expires_in will be converted to epochs
# 2.days is stored as 172800 seconds TTL

Rails.cache.write("super_admin_ids", [1,2,3], expires_at: (Time.now + 2.days).end_of_day)
# (Time.now + 2.days).end_of_day will return 2023-02-27 23:59:59.999999999
```



```
Rails.cache.write("super_admin_ids", [1,2,3], expires_at: 5.minutes)
# 5.minutes will set expires_at to 300 ( Thursday, 1 January 1970 00:05:00 )
```

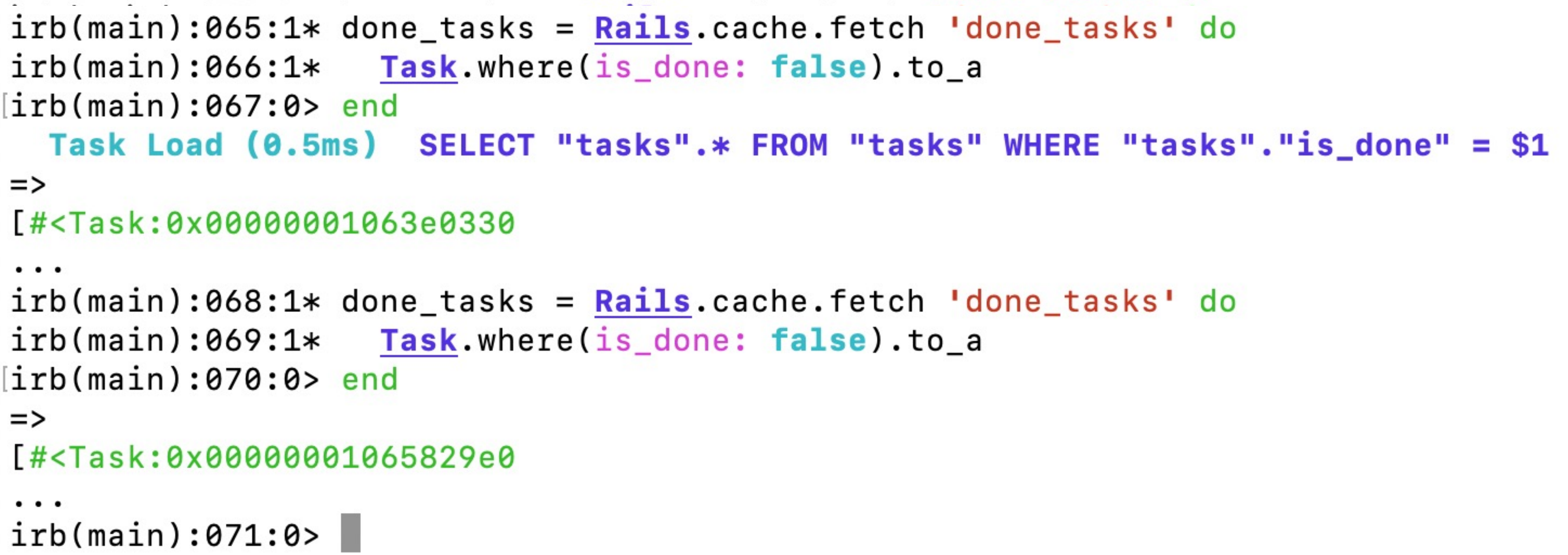
If both the expires_in and expires_at are set, expires_at gets priority.

Tips – Avoid direct cache AR queries



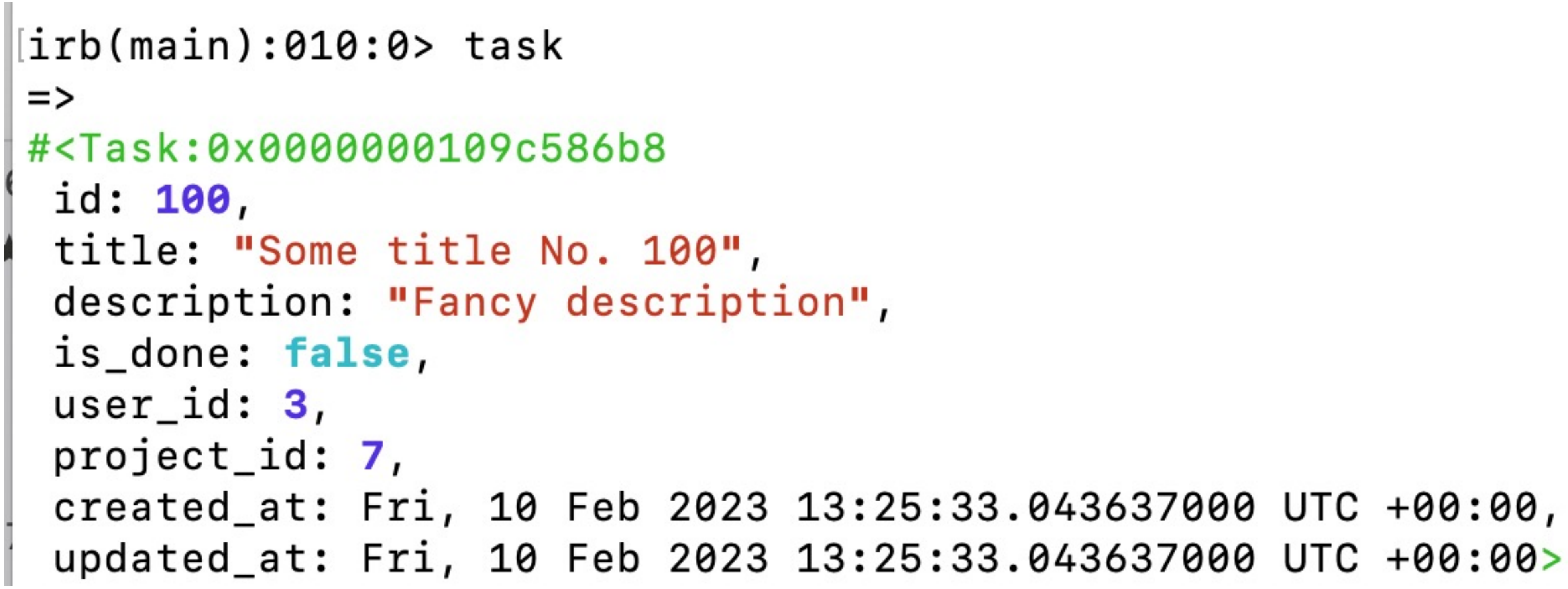
```
irb(main):055:1* done_tasks = Rails.cache.fetch 'done_tasks' do
irb(main):056:1*   Task.where(is_done: false)
irb(main):057:0> end
  Task Load (0.7ms)  SELECT "tasks".* FROM "tasks" WHERE "tasks"."is_done" = $1
=>
[#<Task:0x0000000105d73498
...
irb(main):058:1* done_tasks = Rails.cache.fetch 'done_tasks' do
irb(main):059:1*   Task.where(is_done: false)
irb(main):060:0> end
  Task Load (0.6ms)  SELECT "tasks".* FROM "tasks" WHERE "tasks"."is_done" = $1
=>
[#<Task:0x0000000106018e28
...
irb(main):061:0>
```

Tips – Avoid direct cache AR queries



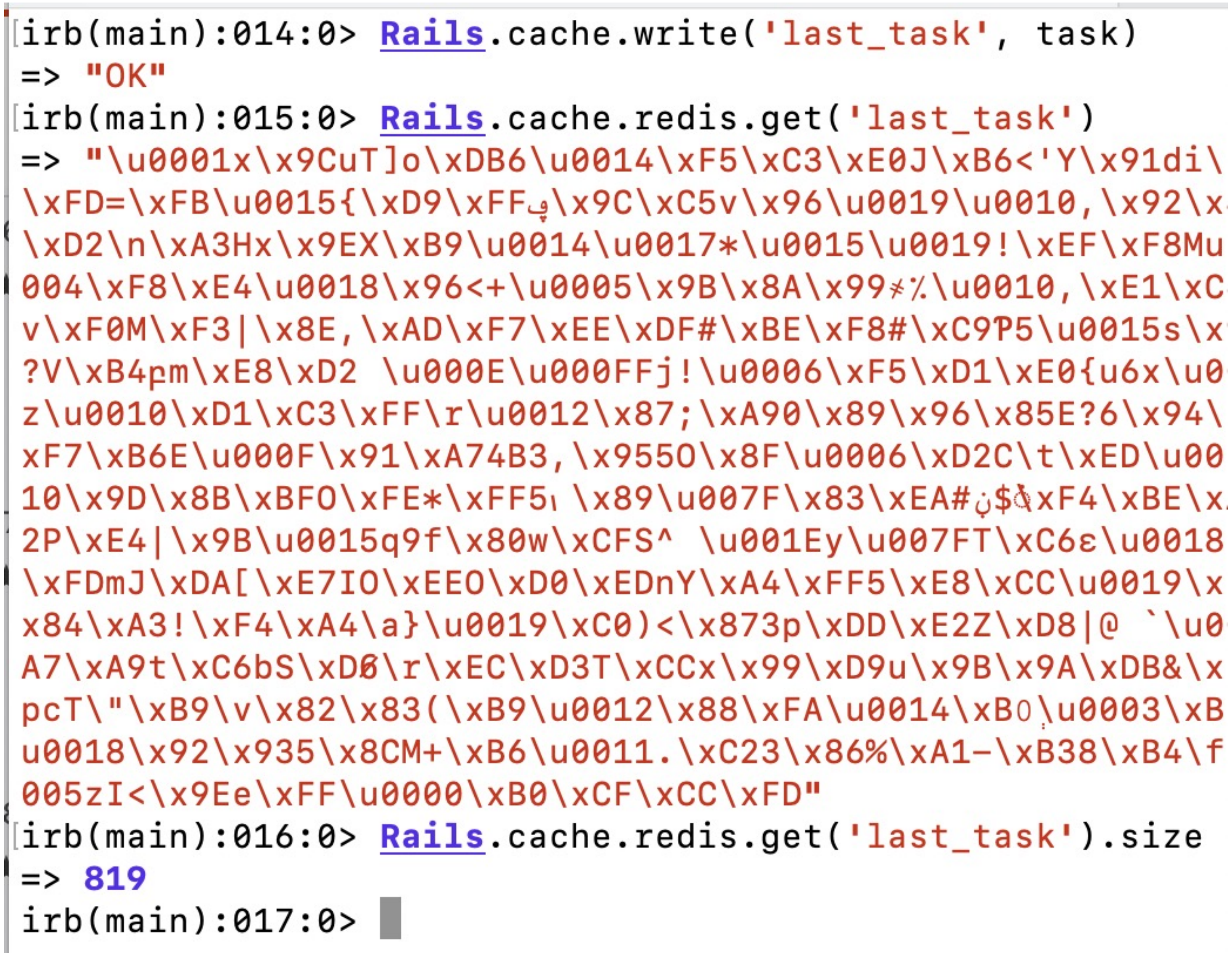
```
irb(main):065:1* done_tasks = Rails.cache.fetch 'done_tasks' do
irb(main):066:1*   Task.where(is_done: false).to_a
irb(main):067:0> end
  Task Load (0.5ms)  SELECT "tasks".* FROM "tasks" WHERE "tasks"."is_done" = $1
=>
[#<Task:0x00000001063e0330
...
irb(main):068:1* done_tasks = Rails.cache.fetch 'done_tasks' do
irb(main):069:1*   Task.where(is_done: false).to_a
irb(main):070:0> end
=>
[#<Task:0x00000001065829e0
...
irb(main):071:0>
```

Tips – Reduce cache size for AR objects



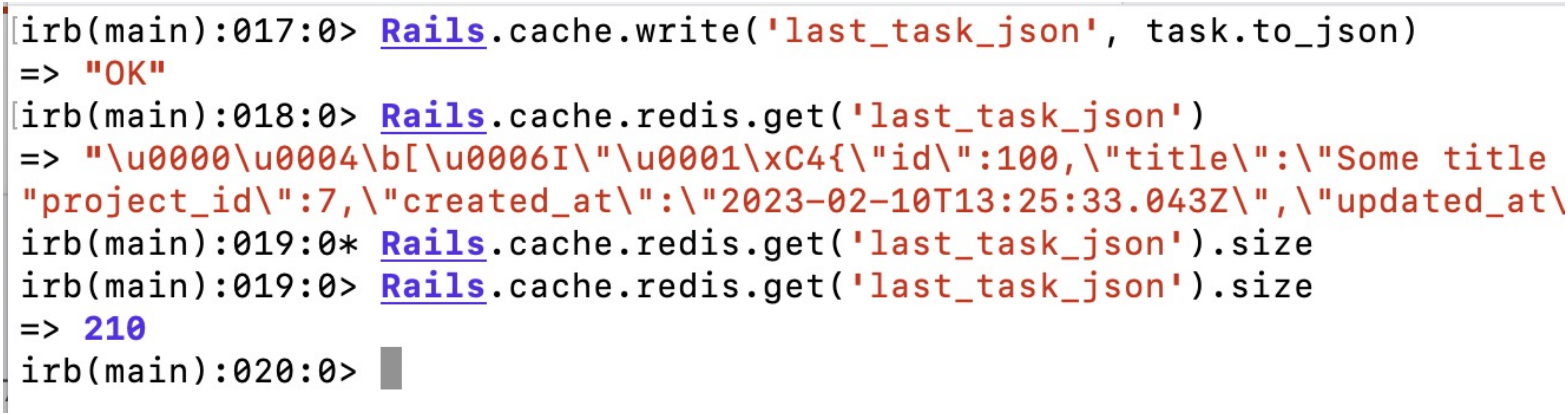
```
irb(main):010:0> task
=>
#<Task:0x0000000109c586b8
 id: 100,
 title: "Some title No. 100",
 description: "Fancy description",
 is_done: false,
 user_id: 3,
 project_id: 7,
 created_at: Fri, 10 Feb 2023 13:25:33.043637000 UTC +00:00,
 updated_at: Fri, 10 Feb 2023 13:25:33.043637000 UTC +00:00>
```

Tips – Reduce cache size for AR objects



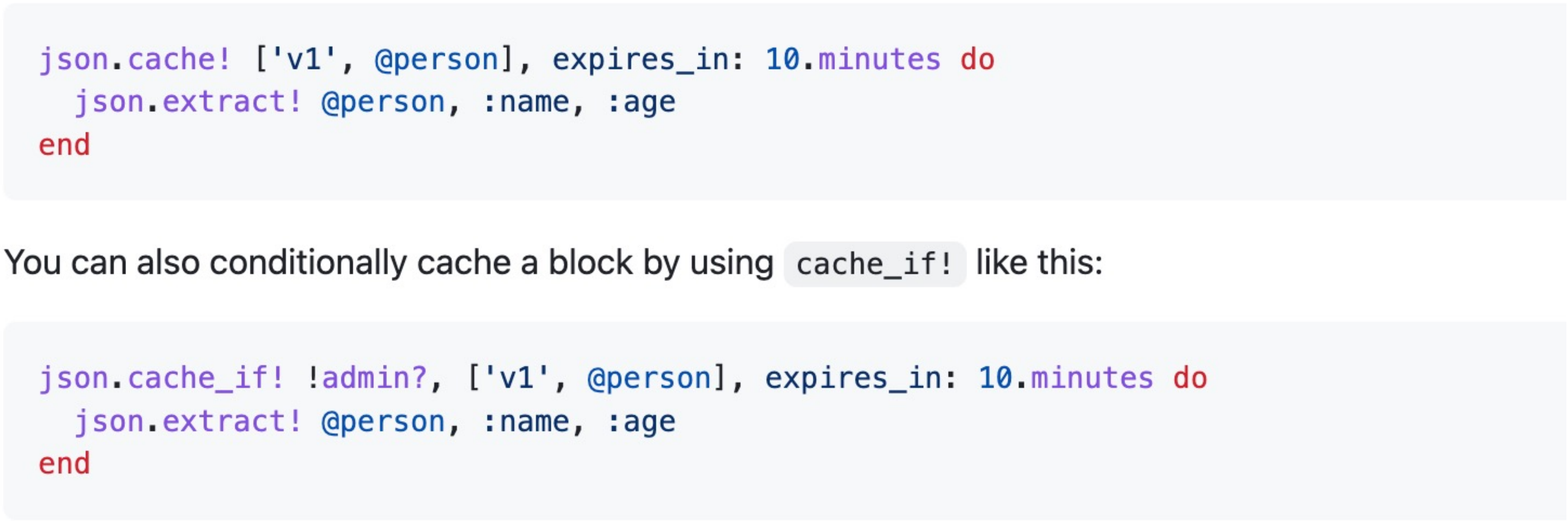
```
irb(main):014:0> Rails.cache.write('last_task', task)
=> "OK"
irb(main):015:0> Rails.cache.redis.get('last_task')
=> "\u0001x\x9CuT]o\xDB6\u0014\xF5\xC3\xE0J\xB6<'Y\x91di\
\xFD=\xFB\u0015{\xD9\xFFپ\x9C\xC5v\x96\u0019\u0010,\x92\x
\xD2\n\xA3Hx\x9EX\xB9\u0014\u0017*\u0015\u0019!\xEF\xF8Mu
004\xF8\xE4\u0018\x96<+\u0005\x9B\x8A\x99≠%\u0010,\xE1\xC
v\xF0M\xF3|\x8E,\xAD\xF7\xEE\xDF#\xBE\xF8#\xC9Ᵽ5\u0015s\x
?V\xB4ꝑm\xE8\xD2 \u000E\u000FFj!\u0006\xF5\xD1\xE0{u6x\u0
z\u0010\xD1\xC3\xFF\r\u0012\x87;\xA90\x89\x96\x85E?6\x94\
xF7\xB6E\u000F\x91\xA74B3,\x955O\x8F\u0006\xD2C\t\xED\u00
10\x9D\x8B\xBFO\xFE*\xFF5ı \x89\u007F\x83\xEA#ڹ$ۀxF4\xBE\x
2P\xE4|\x9B\u0015q9f\x80w\xCFS^ \u001Ey\u007FT\xC6ε\u0018
\xFDmJ\xDA[\xE7IO\xEEO\xD0\xEDnY\xA4\xFF5\xE8\xCC\u0019\x
x84\xA3!\xF4\xA4\a}\u0019\xC0)<\x873p\xDD\xE2Z\xD8|@ `\u0
A7\xA9t\xC6bS\xD6\r\xEC\xD3T\xCCx\x99\xD9u\x9B\x9A\xDB&\x
pcT\"\xB9\v\x82\x83(\xB9\u0012\x88\xFA\u0014\xB0\u0003\xB
u0018\x92\x935\x8CM+\xB6\u0011.\xC23\x86%\xA1-\xB38\xB4\f
005zI<\x9Ee\xFF\u0000\xB0\xCF\xCC\xFD"
irb(main):016:0> Rails.cache.redis.get('last_task').size
=> 819
irb(main):017:0> 
```

Tips – Reduce cache size for AR objects



```
irb(main):017:0> Rails.cache.write('last_task_json', task.to_json)
=> "OK"
irb(main):018:0> Rails.cache.redis.get('last_task_json')
=> "\u0000\u0004\b[\u0006I\"\u0001\xC4{\"id\":100,\"title\":\"Some title
"project_id\":7,\"created_at\":\"2023-02-10T13:25:33.043Z\",\"updated_at\
irb(main):019:0* Rails.cache.redis.get('last_task_json').size
irb(main):019:0> Rails.cache.redis.get('last_task_json').size
=> 210
irb(main):020:0> 
```

Tips – Jbuilder cache

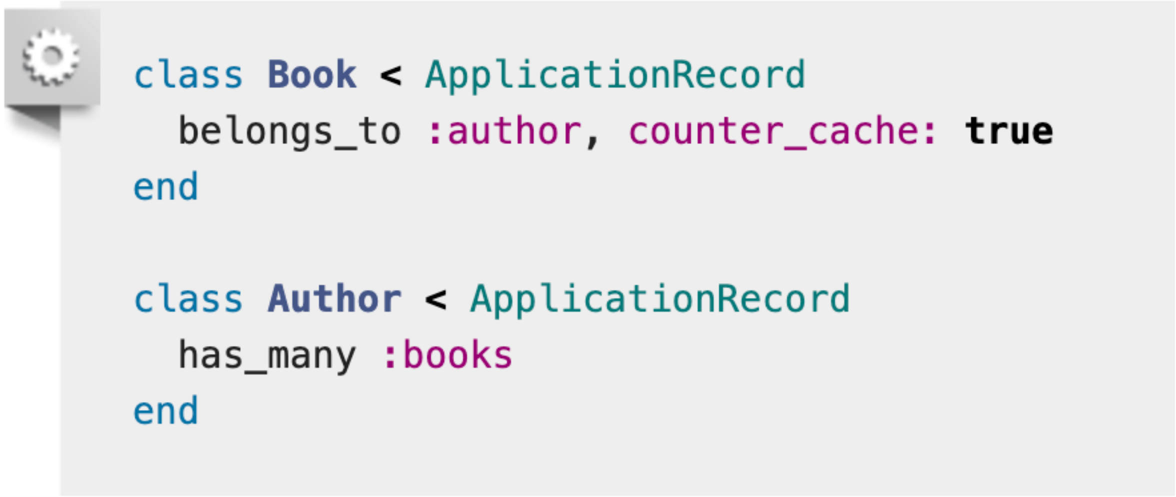


```ruby
json.cache! ['v1', @person], expires_in: 10.minutes do
  json.extract! @person, :name, :age
end
```

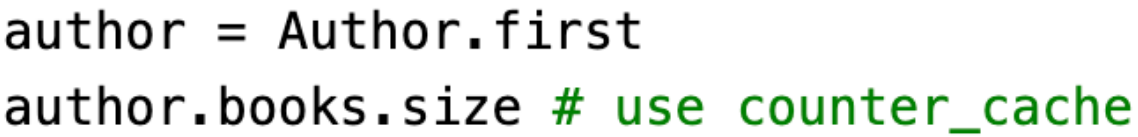
You can also conditionally cache a block by using `cache_if!` like this:

```ruby
json.cache_if! !admin?, ['v1', @person], expires_in: 10.minutes do
  json.extract! @person, :name, :age
end
```

Tips – Counter cache

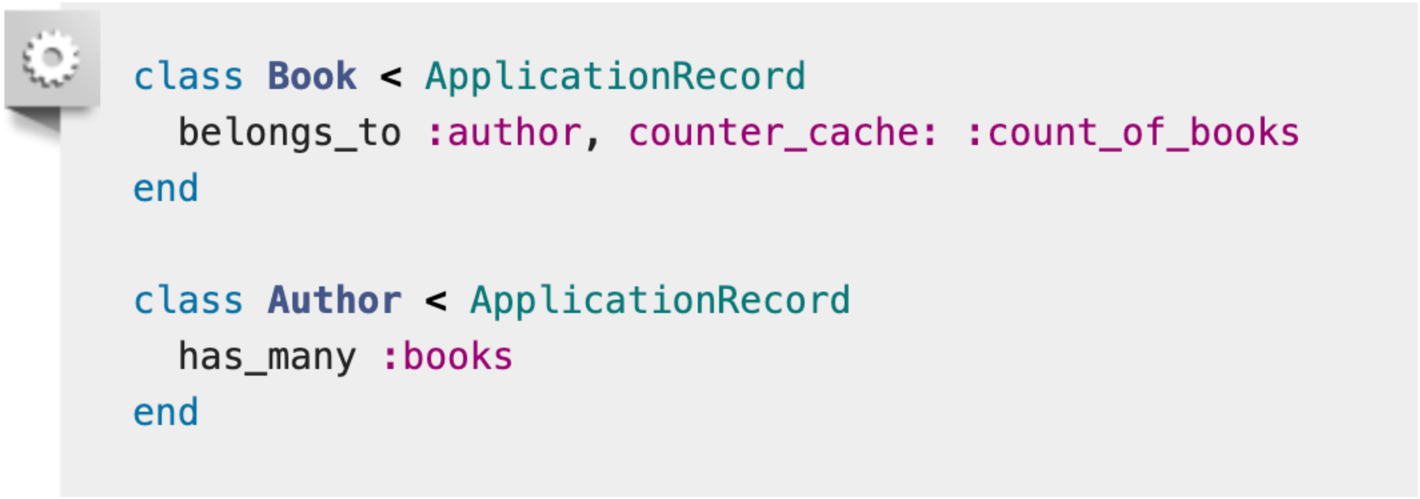


```ruby
class Book < ApplicationRecord
  belongs_to :author, counter_cache: true
end

class Author < ApplicationRecord
  has_many :books
end
```

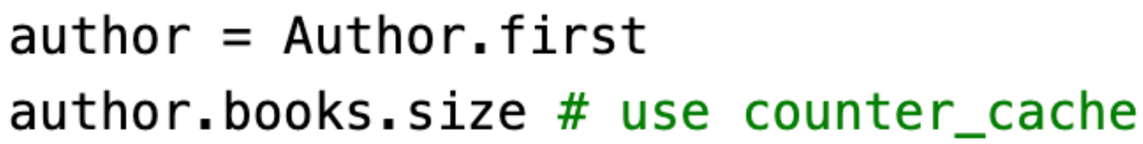


```ruby
author = Author.first
author.books.size # use counter_cache
```

Tips – Counter cache

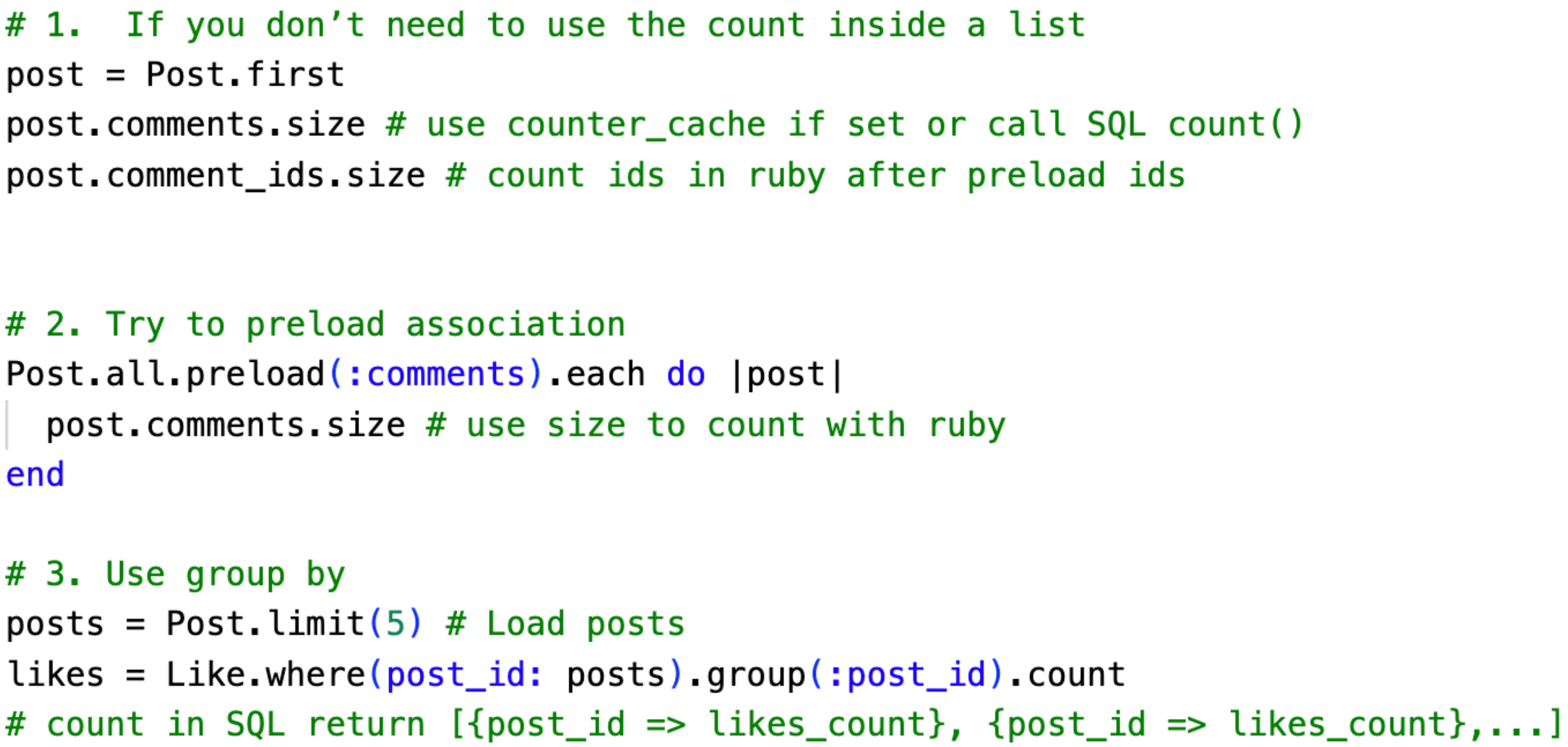


```ruby
class Book < ApplicationRecord
  belongs_to :author, counter_cache: :count_of_books
end

class Author < ApplicationRecord
  has_many :books
end
```



```ruby
author = Author.first
author.books.size # use counter_cache
```

Tips – Counter cache or query size



```
# 1.  If you don’t need to use the count inside a list
post = Post.first
post.comments.size # use counter_cache if set or call SQL count()
post.comment_ids.size # count ids in ruby after preload ids


# 2. Try to preload association
Post.all.preload(:comments).each do |post|
  post.comments.size # use size to count with ruby
end

# 3. Use group by
posts = Post.limit(5) # Load posts
likes = Like.where(post_id: posts).group(:post_id).count
# count in SQL return [{post_id => likes_count}, {post_id => likes_count},...]
```

Good cache practices

- It’s usually better that the service implements cache, rather than the client
- Any playform that uses cache should be able to run completely without it
- In SQL case the best way to implement caching is to avoid it. Always double-check if adding a database index cannot save you from developing a complex cache expiration strategy.

https://imgflip.com/i/7aizl6

RZESZÓW
RUBY
USER
GROUP

Sources:

- https://guides.rubyonrails.org/caching_with_rails.html
- https://www.nacnez.com/caching-in-microservices.html
- https://aws.amazon.com/caching/
- https://www.bigbinary.com/blog/rails-adds-support-for-recyclable-cache-keys
- https://pawelurbanek.com/rails-active-record-caching
- https://github.com/rails/jbuilder
- https://bhserna.com/what-can-you-try-before-using-a-counter-cache-in-rails.html