In this lecture, we will discuss…

- Web Caching
- Client Caching

Web Caching

- Temporary storage of documents to reduce bandwidth usage and server load, and improve performance
- Improve perceived performance by doing less and not repeating ourselves

Client Caching

- Caching Headers
  - `ETag`
  - `Last-Modified`
  - `cache-control`

```
> response=HTTParty.head("http://localhost:3000/movies/12345")
> pp ["cache-control","etag","last-modified"].map {|h| {h=>response.header[h]}}
[{"cache-control"=>"max-age=0, private, must-revalidate"},
 {"etag"=>"W/\"ea6bd9165ddcfb2be59b079aee9fcfca\""},
  {"last-modified"=>nil}]
```

Etag: Using `cache_key`

- Value derived from the database - `cache_key` – same as returned in the response

```
def show
  headers["ETag"]=Digest::MD5.hexdigest(@movie.cache_key)
end
```

```
> HTTParty.head("http://localhost:3000/movies/12345.json").headers["ETag"]
  => "428600c9adc835d6feaefbab903ab72e"
```

Etag: Using `cache_key`

- Value derived from the database - `cache_key` – same as returned in the response

```
def show
  headers["ETag"]=Digest::MD5.hexdigest(@movie.cache_key)
end
```

```
> HTTParty.head("http://localhost:3000/movies/12345.json").headers["ETag"]
  => "428600c9adc835d6feaefbab903ab72e"
```

Etag: Using `cache_key`

- Calculate an ETag manually using the Active Model `cache_key`

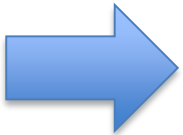
```
> Movie.find("12345").cache_key
 => "movies/12345-20160112171635666000000"

> Digest::MD5.hexdigest(Movie.find("12345").cache_key)
 => "428600c9adc835d6feaefbab903ab72e"
```

Last-Modified

- Control `Last-Modified` as well

```
> Movie.find("12345").updated_at.httpdate
 => "Tue, 12 Jan 2016 17:16:35 GMT"
```



```
def show
  headers["ETag"]=Digest::MD5.hexdigest(@movie.cache_key)
  headers["Last-Modified"]=>@movie.updated_at.httpdate
end
```

```
> HTTParty.head("http://localhost:3000/movies/12345.json").headers["Last-Modified"]
 => "Tue, 12 Jan 2016 17:16:35 GMT"
```

fresh_when

- Rails provides convenient methods that perform the roles discussed
  - **`fresh_when`**

```
def show
#    headers["ETag"]=Digest::MD5.hexdigest(@movie.cache_key)
#    headers["Last-Modified"]=@movie.updated_at.httpdate
    fresh_when(@movie)
  end
```

fresh_when

- last-modified – updated (based on content)

```
> response=HTTParty.head("http://localhost:3000/movies/12345"); #...
[{"cache-control"=>"max-age=0, private, must-revalidate"},
 {"etag"=>"\"2ce808f07c5ff27a4698a87d73cf0d3b\""},
 {"last-modified"=>"Tue, 12 Jan 2016 17:16:35 GMT"}]
```

Summary

- Web caching is a technology that can significantly enhance your Web browsing experience and reduce bandwidth usage

What's Next?

- Cache Revalidation Headers

In this lecture, we will discuss…

- ✧ Cache Revalidation Headers
- ✧ Custom Logic
  - `If-None-Match`
  - `If-Modified-Since`

Cache Revalidation Headers

- Validate if what we have is current or stale using conditional cache validation headers:
  - `If-Not-Match` : (Etag)
  - `If-Modified-Since` : (Timestamp)

Cache Revalidation Headers

- If the resource has not changed
  - enable server-side to do less processing because client does not need a new copy
  - report the resource has not changed to the client
  - enable client-side to do less processing because nothing has changed

ETag and Last-Modified: Storing

- Get a current copy of ETag and Last-Modified and store

```
> response=HTTParty.get("http://localhost:3000/movies/12345",
                        headers:{"Accept"=>"application/json"})
> response.response
 => #<Net::HTTPOK 200 OK readbody=true>
> etag=response.header["etag"]
 => "\"dd7543eb8124a81a065c2d0629222e2c\""
> last_modified=response.header["last-modified"]
 => "Tue, 12 Jan 2016 17:16:35 GMT"
```

GET: Same Action

- GET request, same payload
- 200 ID - returned

```
> response=HTTParty.get("http://localhost:3000/movies/12345",
                        headers:{"Accept"=>"application/json"})
> response.response
 => #<Net::HTTPOK 200 OK readbody=true>
```

With Headers

- Add `If-None-Match` header with the `ETag` or `If-Modified-Since` with the `Last-Modified` timestamp
- `304/NOT_MODIFIED` - returned

```
> response=HTTParty.get("http://localhost:3000/movies/12345",
                        headers:{"Accept"=>"application/json",
                                 "If-None-Match"=>etag})
> response.response
 => #<Net::HTTPNotModified 304 Not Modified readbody=true>
> response.body
 => nil
```

Conditional Logic

- Fire only if the caller is not getting `304/NOT_MODIFIED`

```
def show
#    fresh_when(@movie)
    @movie.movie_accesses.create(:action=>"show")
    if stale? @movie
      @movie.movie_accesses.create(:action=>"show-stale")
      #do some additional, expensive work here
    end
  end
```

- **Note**: `stale?` calls `fresh_when` under the covers

Conditional Logic

```
> response=HTTParty.get("http://localhost:3000/movies/12345",
                        headers:{"Accept"=>"application/json",
                                 "If-Modified-Since"=>last_modified})
> response.response
 => #<Net::HTTPNotModified 304 Not Modified readbody=true>
> pp Movie.find("12345").movie_accesses.pluck(:created_at, :action).to_a
[[2016-01-12 18:37:08 UTC, "show"]]
```

- `last_modified` value is being set to the most recent change in the collection

```
def index
  @movies = Movie.all
  fresh_when last_modified: @movies.max(:updated_at)
end
```

Conditional Logic

- ✧ Provide both `If-Modified-Since` and `If-None-Match` in the header
  - If either fires, our conditional logic will get triggered

```
> response=HTTParty.get("http://localhost:3000/movies/12345",
                        headers:{"Accept"=>"application/json",
                                 "If-Modified-Since"=>last_modified,
                                 "If-None-Match"=>"123"})
> response.response
 => #<Net::HTTPOK 200 OK readbody=true>
> pp Movie.find("12345").movie_accesses.pluck(:created_at, :action).to_a
[[2016-01-12 18:37:08 UTC, "show"],
 [2016-01-12 18:45:49 UTC, "show"],
 [2016-01-12 18:45:49 UTC, "show-stale"]]
```

Browser Test

- Chrome → Developer Tools → Network (Preserve Logs)
- http://localhost:3000/movies/12345.json (Status → 200/OK)
- Hit refresh (Status → 304/Not Modified)
  - `If-Modified-Since` and `If-None-Match` headers were supplied
- Click disable-cache at the top of the Network tab and hit refresh
  - The conditional headers are not sent to the Rails server and the full response is returned using a 200/OK

Cache Revalidation Headers

Demo

Summary

- Cache Validation Headers add value and improve performance when it comes to caching

What's Next?

- Cache-Controls

In this lecture, we will discuss…

- Cache Control
- Delegation of Responsibility
- “`expires`”

Cache Control

- *Used to specify directives that must be obeyed by all caching mechanisms along the request-response chain*
- Provide better hints to the client as to how long the information is good

Demo

- Request Movie 10 times (rapid fire)

```
> 10.times.each {HTTParty.head("http://localhost:3000/movies/12345")}
 => 10
> pp MovieAccess.where(:movie_id=>"12345",:action=>"show-stale").pluck(:created_at, :action)
[[2016-01-12 19:26:20 UTC, "show-stale"],
 [2016-01-12 19:26:20 UTC, "show-stale"],
 [2016-01-12 19:26:20 UTC, "show-stale"],
 [2016-01-12 19:26:20 UTC, "show-stale"],
 [2016-01-12 19:26:20 UTC, "show-stale"],
 [2016-01-12 19:26:20 UTC, "show-stale"],
 [2016-01-12 19:26:21 UTC, "show-stale"],
 [2016-01-12 19:26:21 UTC, "show-stale"],
 [2016-01-12 19:26:21 UTC, "show-stale"],
 [2016-01-12 19:26:21 UTC, "show-stale"]]
```

- Each call results in a database access (no headers)

Delegate Responsibility

- Update the show method to include two caching headers:
  - Expires and Cache-Control
  - Overlap in meaning and if they ever conflict, `Cache-Control` is supposed to take precedence

Delegate Responsibility

- Document will expire at a certain time
- Document is not specific to an individual caller
  - You may cache this document for other callers as well
  - If this information was specific to the caller (e.g., a personal bank statement), then `Cache-Control` would either be set to `nocache` or private to keep the resource from being served to other clients
- The maximum time to cache = 10 seconds

expires

- Rails method - set the Cache-Control response header

```
  def show
    @movie.movie_accesses.create(:action=>"show")
    if stale? @movie
      @movie.movie_accesses.create(:action=>"show-stale")
      #do some additional, expensive work here
    secs=10
    response.headers["Expires"] = secs.seconds.from_now.httpdate
#    response.headers["Cache-Control"] = "public, max-age=#{secs}"
    expires_in secs.seconds, :public=>true
    end
```

Sample Call: return message

- Sample response

```
etag:
- '"dd7543eb8124a81a065c2d0629222e2c"'
last-modified:
- Tue, 12 Jan 2016 17:16:35 GMT
expires:
- Tue, 12 Jan 2016 19:52:25 GMT
cache-control:
- max-age=10, public
```

Changes

- Add gems
  - gem 'httparty'
  - gem 'dry_ice‘
- app/services/

```ruby
# app/services/cached_ws.rb
class CachedWS
  include HTTParty
  include HTTParty::DryIce
#  debug_output $stdout
  base_uri "http://localhost:3000"
  cache Rails.cache
end
```

Demo

- ✧ Script – DB is polled every 9 to 12 seconds
  - 3 second sleep and 10 second cache timeout

```
> 10.times.each do |x|
   p "look=#{x}, accesses=#{Movie.find("12345").movie_accesses.where(:action=>"show").count}"
   CachedWS.get("/movies/12345.json").parsed_response
   sleep(3.seconds)
   end
"look=0, accesses=0"
"look=1, accesses=1"
"look=2, accesses=1"
"look=3, accesses=1"
"look=4, accesses=1"
"look=5, accesses=2"
"look=6, accesses=2"
"look=7, accesses=2"
"look=8, accesses=2"
"look=9, accesses=3"
```

Service Side Caching

Demo

Summary

- Cache Control techniques can be done to offload some work that may not have to be done during steady-state

What's Next?

- Server Caching

In this lecture, we will discuss…

- Server Caching
- Page Caching

Server Caching

- Focus is on the server – how to make it efficient
- Various types of caching on the server side
- Can be turned on or off – globally

```
# config/environments/development.rb
  config.action_controller.perform_caching = true
```

Server Caching - Types

- Rails several levels of caching
  - Page Caching
  - Action Caching
  - Fragment Caching
  - Low Level Caching

Page Caching

- ✧ Page Cache
  - writes static files to directory
  - lazily updates files only when accessed
  - invalidates/removes files on events like updates
  - directory cleared of stale content using sweeper

Page Caching

- Web Server
  - Serves a public single URI
  - Looks for content first in static content directory
  - Makes request to Rails server if static content is missing

Page Caching - Properties

- ✧ Fast - pre-rendered views being served
- ✧ Good for
  - dynamic content that stays stable for periods of time
  - content served without regard to caller

Page Caching - Properties

- ✧ Not appropriate for
  - content that varies per user (e.g., login, preferences)
  - content that is very dynamic
- ✧ Separate gem
  - Gemfile: gem 'actionpack-page_caching'

Caching Setup

- Turn on caching

```
# config/environments/development.rb
  config.action_controller.perform_caching = true
  config.action_controller.page_cache_directory = "#{Rails.root.to_s}/public/page_cache"
```

- Add `caches_page`

```
class MoviePagesController < ApplicationController
  before_action :set_movie, only: [:show, :edit, :update, :destroy]
  caches_page :index, :show
```

Caching Setup - expiration

- Page Expiration

```
  def update
    respond_to do |format|
      if @movie.update(movie_params)
        expire_page action: "show", id:@movie, format: request.format.symbol
        expire_page action: "index", format: request.format.symbol
...
  def destroy
    @movie.movie_accesses.create(:action=>"destroy")
    @movie.destroy
    expire_page action: "show", id:@movie, format: request.format.symbol
    expire_page action: "index", format: request.format.symbol
...
```

caches folder

- The rendered content is written to files in the public directory based on the URI.
  - Result of calling index and show methods

```
public/page_cache/
|-- movie_pages
|   `-- 12345.json
`-- movie_pages.json
```

Service side Caching

Demo

Summary

- Caching can be at both the client side or on the server side.

What's Next?

- Web Services Security