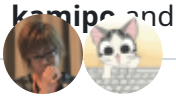
rack / lib / rack / session / abstract / id.rb



kamipo and ioquatix Fix to handle same_site option for session pool a0d57d4 · 5 years ago

523 lines (420 l…

```ruby
# frozen_string_literal: true

# AUTHOR: blink <blinketje@gmail.com>; blink#ruby-lang@irc.freenode.net
# bugrep: Andreas Zehnder

require_relative '../../../rack'
require 'time'
require 'securerandom'
require 'digest/sha2'

module Rack

  module Session

    class SessionId
      ID_VERSION = 2

      attr_reader :public_id

      def initialize(public_id)
        @public_id = public_id
      end

      def private_id
        "#{ID_VERSION}::#{hash_sid(public_id)}"
      end

      alias :cookie_value :public_id
      alias :to_s :public_id

      def empty?; false; end
      def inspect; public_id.inspect; end

      private

      def hash_sid(sid)
```

```ruby
      Digest::SHA256.hexdigest(sid)
    end
  end

  module Abstract
    # SessionHash is responsible to lazily load the session from store.

    class SessionHash
      include Enumerable
      attr_writer :id

      Unspecified = Object.new

      def self.find(req)
        req.get_header RACK_SESSION
      end

      def self.set(req, session)
        req.set_header RACK_SESSION, session
      end

      def self.set_options(req, options)
        req.set_header RACK_SESSION_OPTIONS, options.dup
      end

      def initialize(store, req)
        @store = store
        @req = req
        @loaded = false
      end

      def id
        return @id if @loaded or instance_variable_defined?(:@id)
        @id = @store.send(:extract_session_id, @req)
      end

      def options
        @req.session_options
      end
```

```ruby

        def inspect
          if loaded?
            @data.inspect
          else
            "#<#{self.class}:0x#{self.object_id.to_s(16)} not yet loaded>"
          end
        end

        def exists?
          return @exists if instance_variable_defined?(:@exists)
          @data = {}
          @exists = @store.send(:session_exists?, @req)
        end

        def loaded?
          @loaded
        end

        def empty?
          load_for_read!
          @data.empty?
        end

        def keys
          load_for_read!
          @data.keys
        end

        def values
          load_for_read!
          @data.values
        end

      private

        def load_for_read!
          load! if !loaded? && exists?
        end

        def load_for_write!
          load! unless loaded?
        end

        def load!
          @id, session = @store.send(:load_session, @req)
          @data = stringify_keys(session)
          @loaded = true
        end

        def stringify_keys(other)
          # Use transform_keys after dropping Ruby 2.4 support
```

```ruby
            hash = {}
            other.to_hash.each do |key, value|
              hash[key.to_s] = value
            end
            hash
          end
        end

        # ID sets up a basic framework for implementing an id based sessioning
        # service. Cookies sent to the client for maintaining sessions will only
        # contain an id reference. Only #find_session, #write_session and
        # #delete_session are required to be overwritten.
        #
        # All parameters are optional.
        # * :key determines the name of the cookie, by default it is
        #   'rack.session'
        # * :path, :domain, :expire_after, :secure, and :httponly set the related
        #   cookie options as by Rack::Response#set_cookie
        # * :skip will not a set a cookie in the response nor update the session state
        # * :defer will not set a cookie in the response but still update the session
        #   state if it is used with a backend
        # * :renew (implementation dependent) will prompt the generation of a new
        #   session id, and migration of data to be referenced at the new id. If
        #   :defer is set, it will be overridden and the cookie will be set.
        # * :sidbits sets the number of bits in length that a generated session
        #   id will be.
        #
        # These options can be set on a per request basis, at the location of
        # <tt>env['rack.session.options']</tt>. Additionally the id of the
        # session can be found within the options hash at the key :id. It is
        # highly not recommended to change its value.
        #
        # Is Rack::Utils::Context compatible.
        #
        # Not included by default; you must require 'rack/session/abstract/id'
        # to use.

        class Persisted
          DEFAULT_OPTIONS = {
            key: RACK_SESSION,
            path: '/',
            domain: nil,
            expire_after: nil,
            secure: false,
            httponly: true,
            defer: false,
            renew: false,
            sidbits: 128,
            cookie_only: true,
```

```ruby
233         class Persisted
250           def initialize(app, options = {})
251             @app = app
252             @default_options = self.class::DEFAULT_OPTIONS.merge(options)
253             @key = @default_options.delete(:key)
254             @cookie_only = @default_options.delete(:cookie_only)
255             @same_site = @default_options.delete(:same_site)
256             initialize_sid
257           end
258
259           def call(env)
260             context(env)
261           end
262
263           def context(env, app = @app)
264             req = make_request env
265             prepare_session(req)
266             status, headers, body = app.call(req.env)
267             res = Rack::Response::Raw.new status, headers
268             commit_session(req, res)
269             [status, headers, body]
270           end
271
272           private
273
274           def make_request(env)
275             Rack::Request.new env
276           end
277
278           def initialize_sid
279             @sidbits = @default_options[:sidbits]
280             @sid_secure = @default_options[:secure_random]
281             @sid_length = @sidbits / 4
282           end
283
284           # Generate a new session id using Ruby #rand.  The size of the
285           # session id is controlled by the :sidbits option.
286           # Monkey patch this to use custom methods for session id generation.
287
288           def generate_sid(secure = @sid_secure)
289             if secure
290               secure.hex(@sid_length)
291             else
292               "%0#{@sid_length}x" % Kernel.rand(2**@sidbits - 1)
293             end
294           rescue NotImplementedError
296           end
297
298           # Sets the lazy session at 'rack.session' and places options and session
299           # metadata into 'rack.session.options'.
300
```

```ruby
        def prepare_session(req)
          session_was               = req.get_header RACK_SESSION
          session                   = session_class.new(self, req)
          req.set_header RACK_SESSION, session
          req.set_header RACK_SESSION_OPTIONS, @default_options.dup
          session.merge! session_was if session_was
        end

        # Extracts the session id from provided cookies and passes it and the
        # environment to #find_session.

        def load_session(req)
          sid = current_session_id(req)
          sid, session = find_session(req, sid)
          [sid, session || {}]
        end

        # Extract session id from request object.

        def extract_session_id(request)
          sid = request.cookies[@key]
          sid ||= request.params[@key] unless @cookie_only
          sid
        end

        # Returns the current session id from the SessionHash.

        def current_session_id(req)
          req.get_header(RACK_SESSION).id
        end

        # Check if the session exists or not.

        def session_exists?(req)
          value = current_session_id(req)
          value && !value.empty?
        end

        # Session should be committed if it was loaded, any of specific options like :renew, :dr
        # or :expire_after was given and the security permissions match. Skips if skip is given.

        def commit_session?(req, session, options)
          if options[:skip]
            false
          else
            has_session = loaded_session?(session) || forced_session_update?(session, options)
            has_session && security_matches?(req, options)
          end
        end

        def loaded_session?(session)
          !session.is_a?(session_class) || session.loaded?
        end
```

```ruby
        def forced_session_update?(session, options)
          force_options?(options) && session && !session.empty?
        end

        def force_options?(options)
          options.values_at(:max_age, :renew, :drop, :defer, :expire_after).any?
        end

        def security_matches?(request, options)
          return true unless options[:secure]
          request.ssl?
        end

        # Acquires the session from the environment and the session id from
        # the session options and passes them to #write_session. If successful
        # and the :defer option is not true, a cookie will be added to the
        # response with the session's id.

        def commit_session(req, res)
          session = req.get_header RACK_SESSION
          options = session.options

          if options[:drop] || options[:renew]
            session_id = delete_session(req, session.id || generate_sid, options)
            return unless session_id
          end

          return unless commit_session?(req, session, options)

          session.send(:load!) unless loaded_session?(session)
          session_id ||= session.id
          session_data = session.to_hash.delete_if { |k, v| v.nil? }

          if not data = write_session(req, session_id, session_data, options)
            req.get_header(RACK_ERRORS).puts("Warning! #{self.class.name} failed to save session
          elsif options[:defer] and not options[:renew]
            req.get_header(RACK_ERRORS).puts("Deferring cookie for #{session_id}") if $VERBOSE
          else
            cookie = Hash.new
            cookie[:value] = cookie_value(data)
            cookie[:expires] = Time.now + options[:expire_after] if options[:expire_after]
            cookie[:expires] = Time.now + options[:max_age] if options[:max_age]
```

```
450           end
451         end
452
453         class PersistedSecure < Persisted
454           class SecureSessionHash < SessionHash
455             def [](key)
456               if key == "session_id"
457                 load_for_read!
458                 id.public_id if id
459               else
```

```ruby
                super
              end
            end
          end

          def generate_sid(*)
            public_id = super

            SessionId.new(public_id)
          end

          def extract_session_id(*)
            public_id = super
            public_id && SessionId.new(public_id)
          end

          private

          def session_class
            SecureSessionHash
          end

          def cookie_value(data)
            data.cookie_value
          end
        end

        class ID < Persisted
          def self.inherited(klass)
            k = klass.ancestors.find { |kl| kl.respond_to?(:superclass) && kl.superclass == ID }
            unless k.instance_variable_defined?(:"@_rack_warned")
              warn "#{klass} is inheriting from #{ID}.  Inheriting from #{ID} is deprecated, pleas
              k.instance_variable_set(:"@_rack_warned", true)
            end
            super
          end

          # All thread safety and session retrieval procedures should occur here.
          # Should return [session_id, session].
          # If nil is provided as the session id, generation of a new valid id
          # should occur within.

          def find_session(req, sid)
            get_session req.env, sid
          end

          # All thread safety and session storage procedures should occur here.
          # Must return the session id if the session was saved successfully, or
          # false if the session could not be saved.

          def write_session(req, sid, session, options)
            set_session req.env, sid, session, options
          end
```

```
513
514             # All thread safety and session destroy procedures should occur here.
515             # Should return a new session id or nil if options[:drop]
516
517             def delete_session(req, sid, options)
518               destroy_session req.env, sid, options
519             end
520           end
521         end
522       end
523     end
```