# Improve Cookie-based Session with Decorator Pattern

@ ConFoo Montreal 2018-03-08
by Jian Weihang

# Bonjour!

# 簡煒航

## Jian Weihang

# @tonytonyjan

# Taiwan


Ireland
Kingdom
Belarus
Poland
Germany
Ukraine
Austria
France
Romania
Italy
Spain
Portugal
Greece
Turkey
Kazakhstan
Mongolia
Uzbekistan
Kyrgyzstan
Turkmenistan
Syria
Iraq
Iran
Afghanistan
Pakistan
China
South Korea
Sea of J
East China Sea
Nepal
India
Myanmar
(Burma)
Thailand
Philippine
Vietnam
Philippines
Gulf of
Thailand
Malaysia
Indonesia
Banda Sea
Great
Australian
Austra
WA
NT
SA
Tunisia
Morocco
Algeria
Libya
Egypt
Western
Sahara
Saudi Arabia
Oman
Yemen
Gulf of Aden
Arabian Sea
Bay of Bengal
Laccadive Sea
Mauritania
Mali
Niger
Chad
Sudan
Burkina
Faso
Guinea
Ghana
Nigeria
Gulf of Guinea
South Sudan
Ethiopia
Somalia
Kenya
Gabon
DR Congo
Tanzania
Angola
Zambia
Mozambique
Namibia
Zimbabwe
Botswana
Madagascar
South Africa
Indian
Ocean
North
Atlantic
Ocean
South
Atlantic
Ocean
United States
Mexico
Gulf of
Mexico
Cuba
Puerto Rico
Guatemala
Nicaragua
Caribbean Sea
Venezuela
Guyana
Suriname
Colombia
Ecuador
Peru
Brazil
Bolivia
Paraguay
Chile
Uruguay
Argentina
SK
NL
ON
QC
MT
ND
MN
WI
MI
NB
PE
NS
ME
NH
MA
NY
WY
SD
NE
IA
IL
IN
OH
PA
MD
DE
UT
MO
KY
WV
VA
AZ
NM
OK
AR
TN
NC
MS
AL
SC
GA
TX
LA
FL
AM
PA
MA
CE
RN
PB
PI
PE
AL
AC
RO
TO
BA
SE
MT
GO
MG
ES
MS
SP
RJ
PR
SC
RS
RR
AP


Ireland
Kingdom
Belarus
Poland
Germany
Ukraine
Austria
France
Romania
Italy
Spain
Portugal
Greece
Turkey
Kazakhstan
Mongolia
Uzbekistan
Kyrgyzstan
Turkmenistan
Syria
Iraq
Iran
Afghanistan
Pakistan
China
South Korea
Sea of J
East China Sea
Nepal
India
Myanmar (Burma)
Thailand
Vietnam
Philippines
Philippine S
Gulf of Aden
Arabian Sea
Bay of Bengal
Gulf of Thailand
Laccadive Sea
Malaysia
Indonesia
Banda Sea
Saudi Arabia
Oman
Yemen
Ethiopia
Somalia
Kenya
South Sudan
Sudan
Chad
Libya
Egypt
Tunisia
Morocco
Algeria
Western Sahara
Mauritania
Mali
Niger
Nigeria
Burkina Faso
Guinea
Ghana
Gulf of Guinea
Gabon
DR Congo
Tanzania
Angola
Zambia
Mozambique
Namibia
Zimbabwe
Botswana
Madagascar
South Africa
Indian Ocean
North Atlantic Ocean
South Atlantic Ocean
United States
Mexico
Gulf of Mexico
Cuba
Puerto Rico
Guatemala
Nicaragua
Caribbean Sea
Venezuela
Guyana
Suriname
Colombia
Ecuador
Peru
Brazil
Bolivia
Paraguay
Chile
Uruguay
Argentina
Great Australian


United States
Mexico
Gulf of Mexico
Cuba
Puerto Rico
Caribbean Sea
Guatemala
Nicaragua
Venezuela
Guyana
Suriname
Colombia
Ecuador
Peru
Brazil
Bolivia
Paraguay
Chile
Uruguay
Argentina
North Atlantic Ocean
South Atlantic Ocean
Ireland
Kingdom
Germany
Poland
Belarus
Ukraine
Austria
France
Romania
Italy
Spain
Portugal
Greece
Turkey
Kazakhstan
Mongolia
Uzbekistan
Kyrgyzstan
Turkmenistan
Syria
Iraq
Iran
Afghanistan
Pakistan
China
South Korea
Sea of J
East China Sea
Nepal
India
Myanmar (Burma)
Thailand
Vietnam
Philippines
Philippine
Gulf of Thailand
Malaysia
Indonesia
Banda Sea
Bay of Bengal
Arabian Sea
Laccadive Sea
Indian Ocean
Tunisia
Morocco
Algeria
Libya
Egypt
Western Sahara
Mauritania
Mali
Niger
Chad
Sudan
Saudi Arabia
Oman
Yemen
Gulf of Aden
Burkina Faso
Guinea
Ghana
Nigeria
Gulf of Guinea
South Sudan
Ethiopia
Somalia
Kenya
Gabon
DR Congo
Tanzania
Angola
Zambia
Mozambique
Namibia
Zimbabwe
Botswana
Madagascar
South Africa
Austra
Great Australian

```
$ gem install taiwan
```

# Tech Leader

ROUND TAIWAN ROUND

# Ruby Developer

## since 2010

# Maintainer
## of exif and jaro_winkler

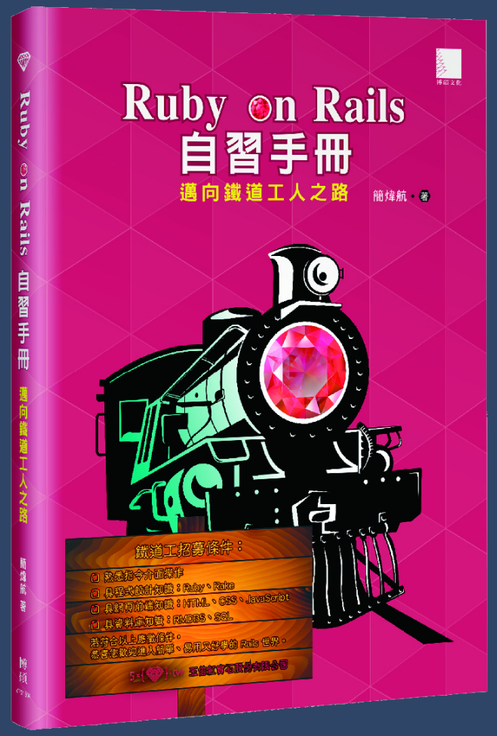
```
└λ gem query -red exif jaro_winkler | grep --color -E '^exif|^jaro_winkler|Jian Weihang|$'
exif (2.2.0)
    Author: Jian Weihang
    Homepage: https://github.com/tonytonyjan/exif

    Ruby EXIF reader written in C extension.
jaro_winkler (1.5.0, 1.4.0)
    Platforms:
        1.4.0: java
        1.5.0: ruby
    Author: Jian Weihang
    Homepage: https://github.com/tonytonyjan/jaro_winkler

    An implementation of Jaro-Winkler distance algorithm written \ in C
    extension which supports any kind of string encoding.
```

# Published a book

## in 2015

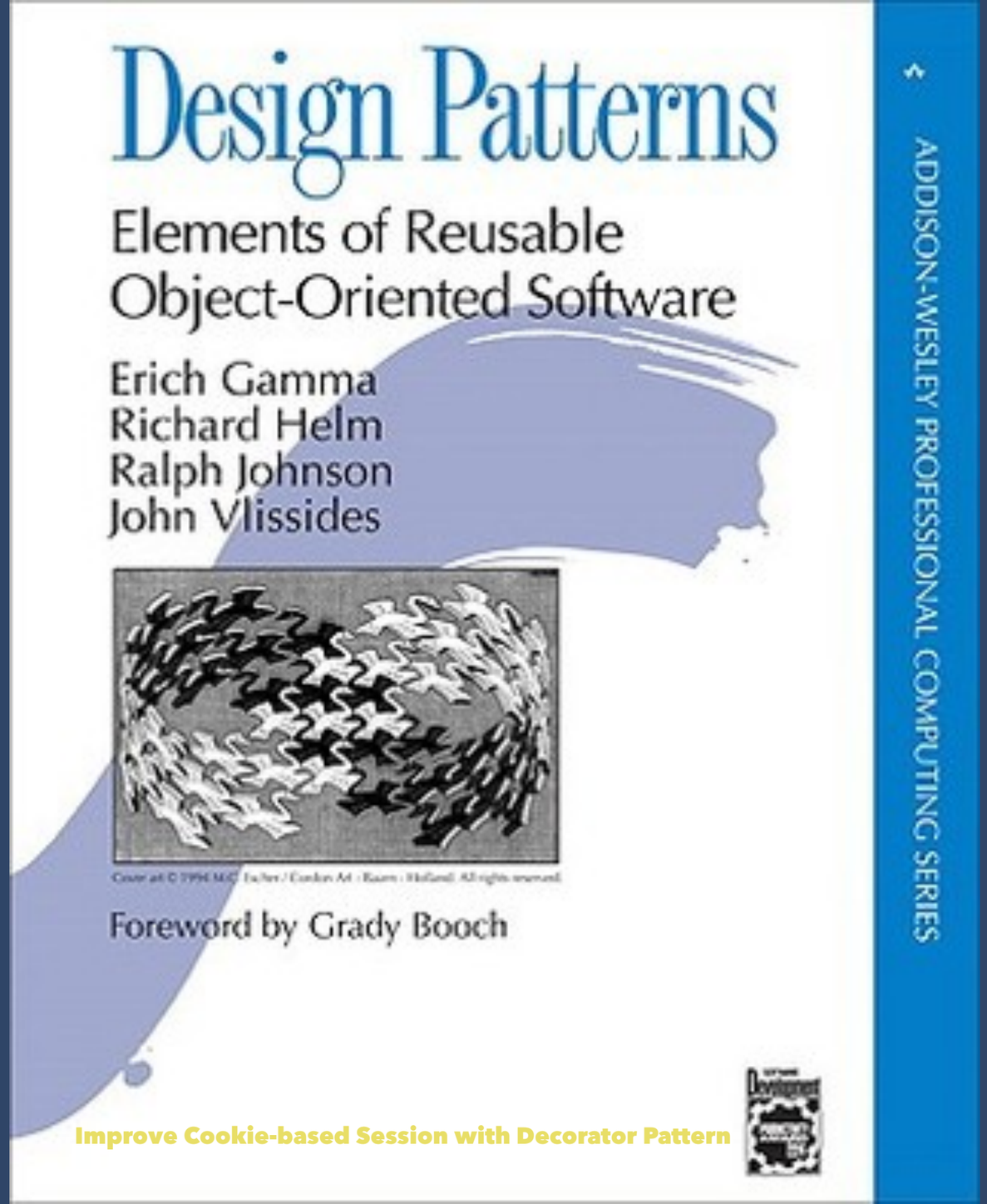


# Improve Cookie-based Session with Decorator Pattern

# Outline

- Introduction of Decorator Pattern
- Security of `Rack::Session::Cookie`
- Encryption of `Rack::Session::Cookie`

# Decorator Pattern




# Decorator Pattern

*Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.*

– Design Patterns by the Gang of Four

# Using Inheritance

```
class CoffeeWithSugar < Coffee
  def cost
    super + 0.2
  end
end

class CoffeeWithMilkAndSugar < Coffee
  def cost
    super + 0.4 + 0.2
  end
end
```

# What's the problem?

- Cannot customize during runtime.
  - Cannot control how and when to decorate a component.
  - ex. double milk?
- It is tightly coupled.

# What's the problem?

- **Cannot customize during runtime.**
  - Cannot control how and when to decorate a component.
  - ex. double milk?
- It is tightly coupled.

# What's the problem?

- Cannot customize during runtime.
  - Cannot control how and when to decorate a component.
  - ex. double milk?
- **It is tightly coupled.**

# Decorator Pattern in Ruby

```ruby
class Milk
  def initialize(coffee); @coffee = coffee end
  def cost; @coffee.cost + 0.4 end
end

class Sugar
  def initialize(coffee); @coffee = coffee end
  def cost; @coffee.cost + 0.2 end
end

coffee = Coffee.new # coffee.cost = 2.0
Sugar.new(Milk.new(coffee)).cost  # 2.6
Sugar.new(Sugar.new(coffee)).cost # 2.4
```

# Benefits

- Plain Old Ruby Object.
- Can be wrapped infinitely.
  - Can use same decorator more than once on component.
- Can customize in runtime.

# Benefits

- **Plain Old Ruby Object.**
- Can be wrapped infinitely.
  - Can use same decorator more than once on component.
- Can customize in runtime.

# Benefits

- Plain Old Ruby Object.
- **Can be wrapped infinitely.**
  - Can use same decorator more than once on component.
- Can customize in runtime.

# Benefits

- Plain Old Ruby Object.
- Can be wrapped infinitely.
  - Can use same decorator more than once on component.
- **Can customize in runtime.**

```ruby
class Additive
  def initialize(coffee)
    @coffee = coffee
  end

  def cost
    raise NotImplementedError
  end
end

class Salt < Additive
  def cost
    @coffee.cost + 0.1
  end
end
```

# Rack

# Is `Rack::Session::Cookie` secure?

# Simple HTTP Server

```ruby
require 'rack'

app = lambda do |env|
  session = env['rack.session']
  session[:name] = 'tonytonyjan'
  session[:age] = 28
  [200, {}, ['it works']]
end

app = Rack::Builder.app(app) do
  use Rack::Session::Cookie, secret: 'secret'
end

Rack::Handler::WEBrick.run app, Port: ARGV[0]
```

# Experiment

# Decode

```
Marshal.load(
  Base64.decode64(
    URI.decode_www_form_component(cookie)
      .split('--')
     .first
  )
)
```

# Structure of Rack Cookie Session

```
+-------------- uri encode ---------------+
| +------ base64 ------+      +- hex -+ |
| | +- Marshal.dump -+ |      |       | |
| | |     object     | | "--" |  mac  | |
| | +----------------+ |      |       | |
| +--------------------+      +-------+ |
+----------------------------------------+
```

# Decode and Verify

```
def decode_cookie(cookie, secret)
  cookie = URI.decode_www_form_component(cookie)
  data, hmac = cookie.split('--')
  computed_hmac = OpenSSL::HMAC.hexdigest(
    OpenSSL::Digest::SHA1.new, secret, data
  )
  raise 'invalid hmac' unless computed_hmac == hmac
  Marshal.load(Base64.decode64(data))
end
```

# Is `Rack::Session::Cookie` secure?

# Not Exactly

# Rack::Session::Cookie

- Sign with HMAC-SHA1.
- No encryption.

# Rack::Session::Cookie

- **Sign with HMAC-SHA1.**
- No encryption.

# Rack::Session::Cookie

- Sign with HMAC-SHA1.
- **No encryption.**

# Sinatra

# Sinatra Official Document

## `:sessions` – enable/disable cookie based sessions

Support for encrypted, cookie-based sessions are included with Sinatra but are disabled by default. Enable them with:

```
set :sessions, true
```

Sessions are implemented by inserting the `Rack::Session::Cookie` component into the application's middleware pipeline.

```ruby
require 'sinatra'

set :sessions, true
set :session_secret, 'set secret'

get '/' do
  session = env['rack.session']
  session[:name] = 'tonytonyjan'
  session[:age] = 28
  'Hello world!'
end
```

# Experiment

sinatra / sinatra.github.com

Watch 22 | Star 278 | Fork 198

Code | Issues 3 | Pull requests 1 | Projects 0 | Insights

# `Rack::Session::Cookie` won't encrypt #220

Edit

Merged zzak merged 1 commit into `sinatra:master` from `tonytonyjan:patch-1` on Dec 25, 2016

Conversation 1 | Commits 1 | Files changed 1

Changes from all commits | Jump to... | +1 −1

Unified | Split | Review changes

2 configuration.markdown

View

```
@@ -91,7 +91,7 @@ The environment can be set explicitly:
91  91
92  92  ### `:sessions` - enable/disable cookie based sessions
93  93
94      -Support for encrypted, cookie-based sessions are included with Sinatra but
    94  +Support for signed (but not encrypted), cookie-based sessions are included with Sinatra but
95  95   are disabled by default. Enable them with:
96  96
97  97       set :sessions, true
```

rubymonsters / webapps-for-beginners

Watch 17 | Star 11 | Fork 10

Code | Issues 3 | Pull requests 2 | Projects 0 | Wiki | Insights

# Actually, it's neither encrypted nor signed. #10

Edit

Merged **svenfuchs** merged 1 commit into `rubymonsters:main` from `tonytonyjan:patch-1` on Jan 17, 2017

Conversation 1 | Commits 1 | Files changed 1

Changes from **all commits** | Jump to... | +2 −2

Unified | Split | Review changes

4 source/12-sessions/02-sinatra_sessions.md

View

```
@@ -46,7 +46,7 @@ Ok, cool.
 The `session` looks like a simple Ruby hash, but if we store something to it
 then Sinatra will set a cookie for us. It does so by sending a `Set-Cookie`
 header along the reponse. This header will have a long, messy looking,
-encrypted string as a value.
+encoded string as a value.

 In my browser it looks like this:

@@ -90,7 +90,7 @@ How does this work?
 In our `post` route we store the message to the session hash. This is
 something Sinatra provides to us as developers. When we enable this
 feature Sinatra will, after every request, store this hash to a cookie
-with the name `rack.session`, in the encrypted form that you saw above.
+with the name `rack.session`, in the encoded form that you saw above.

 We say the hash is being <a href="http://en.wikipedia.org/wiki/Serialization">serialized</a>,
 which is a fancy way of saying it is turned into some kind of format that
```

# Rails

# Convention over Configuration

# Structure of Rails Cookie

```
+-------------------- uri encode --------------------+
| +------------- base64 -------------+      +- hex -+ |
| | +- base64 -+      +- base64 -+ |      |       | |
| | |   JSON   | "--" |    iv    | | "--" |  mac  | |
| | +----------+      +----------+ |      |       | |
| +--------------------------------+      +-------+ |
+-----------------------------------------------------+
```

# Decode with ActiveSupport

```ruby
require 'uri'
require 'json'
require 'active_support'

def verify_and_decrypt_session_cookie(cookie, secret_key_base)
  cookie = URI.decode_www_form_component(cookie)
  salt = 'encrypted cookie'
  signed_salt = 'signed encrypted cookie'
  key_generator = ActiveSupport::KeyGenerator.new(secret_key_base, iterations: 1000)
  secret = key_generator.generate_key(salt)[0, ActiveSupport::MessageEncryptor.key_len]
  sign_secret = key_generator.generate_key(signed_salt)
  encryptor = ActiveSupport::MessageEncryptor.new(secret, sign_secret, serializer: JSON)
  encryptor.decrypt_and_verify(cookie)
end
```

Pure Ruby Version: https://goo.gl/vuQPkr

# Encryption in `Rack::Session::Cookie`

```ruby
require 'rack'
require 'action_dispatch'

secret_key_base = '...secret_key_base...'
key_generator = ActiveSupport::KeyGenerator.new(secret_key_base, iterations: 1000)

app = lambda do |env|
  env['action_dispatch.secret_key_base'] = secret_key_base
  env['action_dispatch.cookies_serializer'] = :json
  env['action_dispatch.signed_cookie_salt'] = 'signed cookie'
  env['action_dispatch.encrypted_cookie_salt'] = 'encrypted cookie'
  env['action_dispatch.encrypted_signed_cookie_salt'] = 'signed encrypted cookie'
  env['action_dispatch.key_generator'] = key_generator
  session = env['rack.session']
  session[:name] = 'tonytonyjan'
  session[:age] = 28
  [200, {}, ['it works']]
end

app = Rack::Builder.app(app) do
  use ActionDispatch::Cookies
  use ActionDispatch::Session::CookieStore, key: '_myapp_session'
end
```

# Cons

- `ActionDispatch` is fat.
- Stack too deep.

# Cons

- `ActionDispatch` **is fat.**
- Stack too deep.

# Cons

- `ActionDispatch` is fat.
- **Stack too deep.**

# Custom Coder

```
coder.respond_to?(:encode) # => true
coder.respond_to?(:decode) # => true

app = Rack::Builder.app(app) do
  use(Rack::Session::Cookie,
    coder: coder,
    let_coder_handle_secure_encoding: true
  )
end
```

# Custom Coder

```
class CustomCoder
  def encode(obj)
    Base64.encode64(Marshal.dump(obj))
  end

  def decode(obj)
    Marshal.decode64(Base64.load(obj))
  end
end
```

# Cons

- Lack of Flexibility
- Tight coupling

# Building Coder with Decorator Pattern

```ruby
class Coder
  def initialize(coder = nil)
    @coder = coder
  end

  def encode(obj)
    raise NotImplementedError
  end

  def decode(obj)
    raise NotImplementedError
  end
end
```

# Usage

```
class JsonCoder < Coder
  def encode(obj)
    JSON.dump(@coder.encode(obj))
  end

  def decode(str)
    @coder.decode(JSON.parse(str))
  end
end
```

# Usage

```
class Base64Coder < Coder
  def encode(obj)
    Base64.encode64(@coder.encode(obj))
  end

  def decode(str)
    @coder.decode(Base64.decode64(str))
  end
end
```

# A Coder Behaves like Rack::Session::Cookie

```
coder = HMACCoder.new(
  Base64Coder.new(MarshalCoder.new),
  secret: 'secret', digest: 'SHA1'
)
```

# What about Rails?

# A Coder Behaves like Rails

```
Coders::HMAC.new(
  Coders::Cipher.new(
    Coders::JSON.new,
    secret: 'secret'
  ),
  secret: 'secret'
)
```

```
gem 'coder_decorator'
```

# Demo

# Can they be part of Rack core?

# Built-in coders in Rack

- `Rack::Session::Cookie::Base64::Marshal`
- `Rack::Session::Cookie::Base64::JSON`
- `Rack::Session::Cookie::Base64::ZipJSON`
- `Rack::Session::Cookie::Identity`

# Bad Designs

- They are implemented via inheritance
- It repeats itself
- Force to encode in base64 in the end
- The namespace makes no sense

# Bad Designs

- **They are implemented via inheritance**
- It repeats itself
- Force to encode in base64 in the end
- The namespace makes no sense

# rack-2.0.4/lib/rack/session/cookie.rb

```
class Base64
  def encode(str); [str].pack('m') end

  def decode(str); str.unpack('m').first end

  class Marshal < Base64
    def encode(str); super(::Marshal.dump(str)) end

    def decode(str)
      return unless str
      ::Marshal.load(super(str)) rescue nil
    end
  end
end
```

# Bad Designs

- They are implemented via inheritance
- **It repeats itself**
- Force to encode in base64 in the end
- The namespace makes no sense

## rack-2.0.4/lib/rack/session/cookie.rb

```
class JSON < Base64
  def encode(obj)
    super(::JSON.dump(obj))
  end

  def decode(str)
    return unless str
    ::JSON.parse(super(str)) rescue nil
  end
end
```

# rack-2.0.4/lib/rack/session/cookie.rb

```
class ZipJSON < Base64
  def encode(obj)
    super(Zlib::Deflate.deflate(::JSON.dump(obj)))
  end

  def decode(str)
    return unless str
    ::JSON.parse(Zlib::Inflate.inflate(super(str)))
  rescue
    nil
  end
end
```

# Bad Designs

- They are implemented via inheritance
- It repeats itself
- **Force to encode in base64 in the end**
- The namespace makes no sense

## rack-2.0.4/lib/rack/session/cookie.rb

```
class Base64
  class Marshal < Base64
  end

  class JSON < Base64
  end

  class ZipJSON < Base64
  end
end
```

# Bad Designs

- They are implemented via inheritance
- It repeats itself
- Force to encode in base64 in the end
- **The namespace makes no sense**

# Built-in coders in Rack

- `Rack::Session::Cookie::Base64::Marshal`
- `Rack::Session::Cookie::Base64::JSON`
- `Rack::Session::Cookie::Base64::ZipJSON`
- `Rack::Session::Cookie::Identity`

# A Better Namespace to Put Coders

- `Rack::Cokers::Coder`
- `Rack::Cokers::JSON`
- `Rack::Cokers::Marshal`
- `Rack::Cokers::HMAC`

# Improve Rack!

# Difficulty

- Backward compatibility
- Signature is not controlled by coder

# Difficulty

- **Backward compatibility**
- Signature is not controlled by coder

# Difficulty

- Backward compatibility
- **Signature is not controlled by coder**

# rack-2.0.4/lib/rack/session/cookie.rb

```ruby
module Rack::Session::Cookie
  def initialize
    @secrets = options.values_at(:secret, :old_secret).compact
  end

  def write_session(req, session_id, session, options)
    if @secrets.first
      session_data << "--#{generate_hmac(session_data, @secrets.first)}"
    end
  end
end
```

# Add built-in coders in `Rack::Session::Cookie::Coder` using decorator pattern. #1134

Edit

Open **tonytonyjan** wants to merge 5 commits into `rack:master` from `tonytonyjan:patch-coder`

Conversation 4 | Commits 5 | Files changed 2 | +188 −39

**tonytonyjan** commented on Dec 8, 2016 — Contributor

I saw there has been some built-in coder, such as `Base64::Marshal`, `Base64::JSON`, `Base64::ZipJSON`. However, since they are all built on top of `Rack::Cookie::Base64`, it makes them not flexible, reusable and unable to be used without base64.

With this patch, we can build a complex coder by wrapping multiple coders, for example, to encode with Marshal and Base64:

```
coder = Coder::Base64.new(Coder::Marshal.new, strict: true)
use Session::Cookie, coder: coder
```

Encode with JSON and Zip:

```
coder = Coder::Base64.new(Coder::Marshal.new)
use Session::Cookie, coder: coder
```

## Advice Wanted

1. `Rack::Session::Cookie::Coder` can be used stand-alone, and seems not part of `Rack::Session::Cookie`'s responsibility. I am considering whether I should extract those code to another module, maybe `Rack::Util`?
2. I've also written `Encrypt` and `HMAC` coders, It's very useful when we want a signed, encrypted secure cookie like Rails did, for example:

**Reviewers**
No reviews

**Assignees**
No one assigned

**Labels**
None yet

**Projects**
None yet

**Milestone**
No milestone

**Notifications**
Unsubscribe
You're receiving notifications because you authored the thread.

**3 participants**

Allow edits from maintainers.

https://github.com/rack/rack/pull/1134

# Conclusion

- source codes never lie
- mature project can still have some bad design

# Happy Coding

# THANKS FOR YOUR LISTENING

@tonytonyjan

# References

- https://www.wikiwand.com/en/Design_Patterns
- https://github.com/tonytonyjan/coder_decorator/
- https://gist.github.com/tonytonyjan/d71f040fe1085dfcf1d4#file-rails5with*pure*ruby-rb
- https://github.com/rack/rack/pull/1134
- https://github.com/sinatra/sinatra.github.com/pull/220
- https://github.com/rubymonsters/webapps-for-beginners/

# https://tonytonyjan.net/slides

# Q&A