Net::HTTP

HTTP клиентское API для Ruby

Net::HTTP предоставляет богатую библиотеку, которую можно использовать для создания пользовательских агентов HTTP. Более подробную информацию о HTTP смотрите в [RFC2616](*www.ietf.org/rfc/rfc2616.txt*).

Net::HTTP разработан для тесной работы с URI. URI::HTTP#host, URI::HTTP#port и URL::HTTP#request_uri предназначены для работы с Net::HTTP.

Если вы выполняете только несколько запросов GET, вам следует попробовать Openurl.

Простенькие примеры

Во всех примерах предполагается, что вы подключили Net::HTTP с помощью:

```
require 'net/http
```

Он также зависит от 'uri', поэтому вам не нужно подключать его отдельно.

Методы Net::HTTP, описанные в следующем разделе, не сохраняют соединения. Они не рекомендуются, если вы производите много HTTP-запросов.

GET

```
Net::HTTP.get('example.com', '/index.html') # => String
```

GET с помощью URI

```
uri = URI('http://example.com/index.html?count=10')
Net::HTTP.get(uri) # => String
```

GET с динамическими параметрами

```
uri = URI('http://example.com/index.html')
params = { :limit => 10, :page => 3 }
uri.query = URI.encode_www_form(params)

res = Net::HTTP.get_response(uri)
puts res.body if res.is_a?(Net::HTTPSuccess)
```

POST

```
uri = URI('http://www.example.com/search.cgi')
res = Net::HTTP.post_form(uri, 'q' => 'ruby', 'max' => '50')
puts res.body
```

POST с несколькими значениями

```
uri = URI('http://www.example.com/search.cgi')
res = Net::HTTP.post_form(uri, 'q' => ['ruby', 'perl'], 'max' => '50')
puts res.body
```

Как следует использовать Net::HTTP

Следующий пример кода может быть использован в качестве основы для пользовательского агента HTTP, который может выполнять различные типы запросов, используя постоянные соединения.

```
uri = URI('http://example.com/some_path?query=string')

Net::HTTP.start(uri.host, uri.port) do |http|
  request = Net::HTTP::Get.new uri

  response = http.request request # Net::HTTPResponse object
end
```

::start немедленно создает соединение с HTTP-сервером, которое остается открытым в течение всего периода блокировки. Соединение будет оставаться открытым для нескольких запросов в блоке, если сервер укажет, что поддерживает постоянные соединения.

Если вы хотите повторно использовать соединение для нескольких HTTP-запросов, не закрывая его автоматически, вы можете использовать ::now, а затем вызвать start и finish вручную.

Типы запросов, поддерживаемые Net::HTTP, перечислены ниже в разделе “Классы HTTP-запросов”.

Для всех объектов запроса Net::HTTP и методов быстрого запроса вы можете указать либо строку для пути запроса, либо URI, из которого Net::HTTP извлечет путь запроса.

Response значения

```
uri = URI('http://example.com/index.html')
res = Net::HTTP.get_response(uri)

# Headers
res['Set-Cookie']            # => String
res.get_fields('set-cookie') # => Array
res.to_hash['set-cookie']    # => Array
puts "Headers: #{res.to_hash.inspect}"

# Status
puts res.code       # => '200'
puts res.message    # => 'OK'
puts res.class.name # => 'HTTPOK'

# Body
puts res.body if res.response_body_permitted?
```

Following Redirection (перенаправление)

Каждый объект Net::HttpResponse принадлежит классу для своего кода ответа.

Например, все ответы 2XX являются экземплярами подкласса Net::HTTPSuccess, ответ 3XX является экземпляром подкласса Net::HTTPRedirection, а ответ 200 является экземпляром класса Net::HTTPOK. Подробные сведения о классах ответов приведены в разделе “Классы ответов HTTP” ниже.

Используя инструкцию «*case*», вы можете правильно обрабатывать различные типы ответов:

```
def fetch(uri_str, limit = 10)
  # You should choose a better exception.
  raise ArgumentError, 'too many HTTP redirects' if limit == 0

  response = Net::HTTP.get_response(URI(uri_str))

  case response
  when Net::HTTPSuccess then
    response
  when Net::HTTPRedirection then
```

```
    location = response['location']
    warn "redirected to #{location}"
    fetch(location, limit - 1)
  else
    response.value
  end
end

print fetch('http://www.ruby-lang.org')
```

POST

Запись может быть сделана с использованием класса запросов Net::HTTP::Post. В этом примере создается текст записи, закодированный в URL:

```
uri = URI('http://www.example.com/todo.cgi')
req = Net::HTTP::Post.new(uri)
req.set_form_data('from' => '2005-01-01', 'to' => '2005-03-31')

res = Net::HTTP.start(uri.hostname, uri.port) do |http|
  http.request(req)
end

case res
when Net::HTTPSuccess, Net::HTTPRedirection
  # OK
else
  res.value
end
```

Чтобы отправить данные из нескольких частей/форм, используйте (заголовок)

Net::HTTPHeader #set_form:

```
req = Net::HTTP::Post.new(uri)
req.set_form([['upload', File.open('foo.bar')]], 'multipart/form-data')
```

Другие запросы, которые могут содержать тело, такое как PUT, могут быть созданы таким же образом, используя соответствующий класс запроса (Net::HTTP::Put).

Setting Headers

В следующем примере выполняется GET с использованием заголовка If-Modified-Since. Если файлы не были изменены с момента, указанного в заголовке, будет возвращен ответ без изменений. Более подробную информацию смотрите в разделе 9.3 RFC 2616.

```
uri = URI('http://example.com/cached_response'
file = File.stat 'cached_response'

req = Net::HTTP::Get.new(uri)
req['If-Modified-Since'] = file.mtime.rfc2822

res = Net::HTTP.start(uri.hostname, uri.port) {|http|
  http.request(req)
}

open 'cached_response', 'w' do |io|
  io.write res.body
end if res.is_a?(Net::HTTPSuccess)
```

Базовая аутентификация

Базовая аутентификация выполняется в соответствии с [RFC2617] (www.ietf.org/rfc/rfc2617.txt).

```
uri = URI('http://example.com/index.html?key=value')

req = Net::HTTP::Get.new(uri)
req.basic_auth 'user', 'pass'

res = Net::HTTP.start(uri.hostname, uri.port) {|http|
  http.request(req)
}
puts res.body
```

Streaming Response Bodies

По умолчанию Net::HTTP считывает весь ответ в память. Если вы работаете с большими файлами или хотите внедрить индикатор выполнения, вы можете вместо этого передать текст непосредственно в IO.

```
uri = URI('http://example.com/large_file')

Net::HTTP.start(uri.host, uri.port) do |http|
```

```
  request = Net::HTTP::Get.new uri

  http.request request do |response|
    open 'large_file', 'w' do |io|
      response.read_body do |chunk|
        io.write chunk
      end
    end
  end
end
```

HTTPS

HTTPS осуществляется для HTTP-соединения с помощью #use_ssl=.

```
uri = URI('https://secure.example.com/some_path?query=string')

Net::HTTP.start(uri.host, uri.port, :use_ssl => true) do |http|
  request = Net::HTTP::Get.new uri
  response = http.request request # Net::HTTPResponse object
end
```

В предыдущих версиях Ruby вам нужно было подключить "net/https" для использования HTTPS. Так больше не нужно делать.

Proxies

Net::HTTP автоматически создаст прокси-сервер из переменной окружения http_proxy, если она присутствует. Чтобы отключить использование http_proxy, укажите nil в качестве адреса прокси-сервера.

Вы также можете создать пользовательский прокси-сервер:

```
proxy_addr = 'your.proxy.host'
proxy_port = 8080

Net::HTTP.new('example.com', nil, proxy_addr, proxy_port).start { |http|
  # always proxy via your.proxy.addr:8080
}
```

Смотрите ::new для получения дополнительной информации и примеров, таких как прокси-серверы, для которых требуется имя пользователя и пароль.

Compression

Net::HTTP автоматически добавляет Accept-Encoding для сжатия текстов ответов и автоматически распаковывает gzip-файлы и откатывает ответы, если не был отправлен заголовок Range.

Сжатие можно отключить с помощью Accept-Encoding: identity header.

HTTP Request Classes

Вот иерархия классов HTTP-запросов.

Net::HTTPRequest

- Net::HTTP::Get
- Net::HTTP::Head
- Net::HTTP::Post
- Net::HTTP::Patch
- Net::HTTP::Put
- Net::HTTP::Proppatch
- Net::HTTP::Lock
- Net::HTTP::Unlock
- Net::HTTP::Options
- Net::HTTP::Propfind
- Net::HTTP::Delete
- Net::HTTP::Move
- Net::HTTP::Copy
- Net::HTTP::Mkcol
- Net::HTTP::Trace

HTTP Response Classes

HTTPUnknownResponse

For unhandled HTTP extensions

HTTPInformation

1xx

HTTPContinue

100

HTTPSwitchProtocol

101

HTTPProcessing

102

HTTPEarlyHints

103

HTTPSuccess

2xx

HTTPOK

200

HTTPCreated

201

HTTPAccepted

202

HTTPNonAuthoritativeInformation

203

HTTPNoContent

204

HTTPResetContent

205

HTTPPartialContent

206

HTTPMultiStatus

207

HTTPAlreadyReported

208

HTTPIMUsed

226

HTTPRedirection

3xx

HTTPMultipleChoices

300

HTTPMovedPermanently

301

HTTPFound

302

HTTPSeeOther

303

HTTPNotModified

304

HTTPUseProxy

305

HTTPTemporaryRedirect

307

HTTPPermanentRedirect

308

HTTPClientError

4xx

HTTPBadRequest

400

HTTPUnauthorized

401

HTTPPaymentRequired

402

HTTPForbidden

403

HTTPNotFound

404

HTTPMethodNotAllowed

405

HTTPNotAcceptable

406

HTTPProxyAuthenticationRequired

407

HTTPRequestTimeOut

408

HTTPConflict

409

HTTPGone

410

HTTPLengthRequired

411

HTTPPreconditionFailed

412

HTTPRequestEntityTooLarge

413

HTTPRequestURITooLong

414

HTTPUnsupportedMediaType

415

HTTPRequestedRangeNotSatisfiable

416

HTTPExpectationFailed

417

HTTPMisdirectedRequest

421

HTTPUnprocessableEntity

422

HTTPLocked

423

HTTPFailedDependency

424

HTTPUpgradeRequired

426

HTTPPreconditionRequired

428

HTTPTooManyRequests

429

HTTPRequestHeaderFieldsTooLarge

431

HTTPUnavailableForLegalReasons

451

HTTPServerError

5xx

HTTPInternalServerError

500

HTTPNotImplemented

501

HTTPBadGateway

502

HTTPServiceUnavailable

503

HTTPGatewayTimeOut

504

HTTPVersionNotSupported

505

HTTPVariantAlsoNegotiates

506

HTTPInsufficientStorage

507

HTTPLoopDetected

508

HTTPNotExtended

510

HTTPNetworkAuthenticationRequired

511

Constants

ENVIRONMENT_VARIABLE_IS_MULTIUSER_SAFE
SSL_ATTRIBUTES
SSL_IVNAMES
STATUS_CODES

Attributes

proxy_address[R]

Address of proxy host. If Net::HTTP does not use a proxy, nil.

proxy_pass[R]

User password for accessing proxy. If Net::HTTP does not use a proxy, nil.

proxy_port[R]

Port number of proxy host. If Net::HTTP does not use a proxy, nil.

proxy_user[R]

User name for accessing proxy. If Net::HTTP does not use a proxy, nil.

address[R]

The DNS host name or IP address to connect to.

ca_file[RW]

Sets path of a CA certification file in PEM format.
The file can contain several CA certificates.

ca_path[RW]

Sets path of a CA certification directory containing certifications in PEM format.

cert[RW]

Sets an OpenSSL::X509::Certificate object as client certificate. (This method is appeared in Michal Rokos's OpenSSL extension).

cert_store[RW]

Sets the X509::Store to verify peer certificate.

ciphers[RW]

Sets the available ciphers. See OpenSSL::SSL::SSLContext#ciphers=

close_on_empty_response[RW]

continue_timeout[R]

Seconds to wait for 100 Continue response. If the HTTP object does not receive a response in this many seconds it sends the request body. The default value is `nil`.

extra_chain_cert[RW]

Sets the extra X509 certificates to be added to the certificate chain. See OpenSSL::SSL::SSLContext#extra_chain_cert=

keep_alive_timeout[RW]

Seconds to reuse the connection of the previous request. If the idle time is less than this Keep-Alive Timeout, Net::HTTP reuses the TCP/IP socket used by the previous communication. The default value is 2 seconds.

key[RW]

Sets an OpenSSL::PKey::RSA or OpenSSL::PKey::DSA object. (This method is appeared in Michal Rokos's OpenSSL extension.)

local_host[RW]

The local host used to establish the connection.

local_port[RW]

The local port used to establish the connection.

max_retries[R]

max_version[RW]

Sets the maximum SSL version. See OpenSSL::SSL::SSLContext#max_version=

min_version[RW]

Sets the minimum SSL version. See OpenSSL::SSL::SSLContext#min_version=

open_timeout[RW]

Number of seconds to wait for the connection to open. Any number may be used, including Floats for fractional seconds. If the HTTP object cannot open a connection in

this many seconds, it raises a Net::OpenTimeout exception. The default value is 60 seconds.

port[R]

The port number to connect to.

proxy_address[W]

proxy_from_env[W]

proxy_pass[W]

proxy_port[W]

proxy_user[W]

read_timeout[R]

Number of seconds to wait for one block to be read (via one read(2) call). Any number may be used, including Floats for fractional seconds. If the HTTP object cannot read data in this many seconds, it raises a Net::ReadTimeout exception. The default value is 60 seconds.

ssl_timeout[RW]

Sets the SSL timeout seconds.

ssl_version[RW]

Sets the SSL version. See OpenSSL::SSL::SSLContext#ssl_version=

verify_callback[RW]

Sets the verify callback for the server certification verification.

verify_depth[RW]

Sets the maximum depth for the certificate chain verification.

verify_hostname[RW]

Sets to check the server certificate is valid for the hostname. See OpenSSL::SSL::SSLContext#verify_hostname=

verify_mode[RW]

Sets the flags for server the certification verification at beginning of SSL/TLS session. OpenSSL::SSL::VERIFY_NONE or OpenSSL::SSL::VERIFY_PEER are acceptable.

write_timeout[R]

Number of seconds to wait for one block to be written (via one write(2) call). Any number may be used, including Floats for fractional seconds. If the HTTP object cannot write data in this many seconds, it raises a Net::WriteTimeout exception. The default value is 60 seconds. Net::WriteTimeout is not raised on Windows.

Public Class Methods

Proxy(p_addr = :ENV, p_port = nil, p_user = nil, p_pass = nil)

Создает HTTP-прокси-класс, который ведет себя как Net::HTTP, но выполняет весь доступ через указанный прокси.

Этот класс устарел. Вы можете передать эти же параметры непосредственно в ::new. Подробные сведения об аргументах см. в ::new.

```
              # File net/http.rb, line 1105
def HTTP.Proxy(p_addr = :ENV, p_port = nil, p_user = nil, p_pass = nil)
  return self unless p_addr

  Class.new(self) {
    @is_proxy_class = true

    if p_addr == :ENV then
      @proxy_from_env = true
      @proxy_address = nil
      @proxy_port    = nil
    else
      @proxy_from_env = false
      @proxy_address = p_addr
      @proxy_port    = p_port || default_port
    end

    @proxy_user = p_user
    @proxy_pass = p_pass
  }
end
```

default_port()

Порт по умолчанию, используемый для HTTP-запросов; по умолчанию он равен 80.

```
              # File net/http.rb, line 559
def HTTP.default_port
  http_default_port()
end
```

get(uri_or_host, path_or_headers = nil, port = nil)

Отправляет GET-запрос адресату и возвращает HTTP-ответ в виде строки. Адресат может быть указан либо как (uri, заголовки), либо как (хост, путь, порт = 80); таким образом:

```
print Net::HTTP.get(URI('http://www.example.com/index.html'))
```

или:

```
print Net::HTTP.get('www.example.com', '/index.html')
```

вы также можете указать заголовки запросов:

```
Net::HTTP.get(URI('http://www.example.com/index.html'), { 'Accept' => 'text/html' })
```

```
              # File net/http.rb, line 472
def HTTP.get(uri_or_host, path_or_headers = nil, port = nil)
  get_response(uri_or_host, path_or_headers, port).body
end
```

get_print(uri_or_host, path_or_headers = nil, port = nil)

Получает основной текст из целевого объекта и выводит его в $stdout. Целевой объект может быть указан либо как (uri, headers), либо как (host, path, port = 80); таким образом:

```
Net::HTTP.get_print URI('http://www.example.com/index.html')
```

или:

```
Net::HTTP.get_print 'www.example.com', '/index.html'
```

вы также можете указать заголовки запросов:

```
Net::HTTP.get_print URI('http://www.example.com/index.html'), { 'Accept' => 'text/html' }
```

```
              # File net/http.rb, line 449
def HTTP.get_print(uri_or_host, path_or_headers = nil, port = nil)
  get_response(uri_or_host, path_or_headers, port) {|res|
    res.read_body do |chunk|
      $stdout.print chunk
    end
  }
  nil
end
```

get_response(uri_or_host, path_or_headers = nil, port = nil, &block)

Отправляет получателю запрос GET и возвращает HTTP-ответ в виде объекта Net::HttpResponse. Целевой объект может быть указан либо как (uri, headers), либо как (host, path, port = 80); таким образом,:

```
res = Net::HTTP.get_response(URI('http://www.example.com/index.html'))
print res.body
```

или:

```
res = Net::HTTP.get_response('www.example.com', '/index.html')
print res.body
```

вы также можете указать заголовки запросов:

```
Net::HTTP.get_response(URI('http://www.example.com/index.html'), { 'Accept' => 'text/html' })
```

```
              # File net/http.rb, line 492
def HTTP.get_response(uri_or_host, path_or_headers = nil, port = nil, &block)
  if path_or_headers && !path_or_headers.is_a?(Hash)
    host = uri_or_host
    path = path_or_headers
    new(host, port || HTTP.default_port).start {|http|
      return http.request_get(path, &block)
    }
  else
    uri = uri_or_host
    headers = path_or_headers
    start(uri.hostname, uri.port,
          :use_ssl => uri.scheme == 'https') {|http|
      return http.request_get(uri, headers, &block)
    }
  end
end
```

https_default_port()

Порт по умолчанию, используемый для HTTPS-запросов; по умолчанию он равен 443.

new(address, port = nil, p_addr = :ENV, p_port = nil, p_user = nil, p_pass = nil, p_no_proxy = nil)

Создает новый объект Net::HTTP без открытия TCP-соединения или HTTP-сеанса.

Адрес должен быть именем хоста DNS или IP-адресом, а порт - это порт, на котором работает сервер. Если порт не указан, используется порт по умолчанию для HTTP или HTTPS.

Если ни один из p_arguments не указан, хост и порт прокси-сервера берутся из переменной среды http_proxy (или ее эквивалента в верхнем регистре), если они присутствуют. Если прокси-сервер требует аутентификации, вы должны указать это вручную. Смотрите URI::Generic#find_proxy для получения подробной информации об обнаружении прокси-

сервера в среде. Чтобы отключить обнаружение прокси-сервера, установите значение p_addr равным нулю.

Если вы подключаетесь к пользовательскому прокси-серверу, p_addr указывает DNS-имя или IP-адрес прокси-сервера, p_port - порт, который будет использоваться для доступа к прокси-серверу, p_user и p_pass - имя пользователя и пароль, если для использования прокси-сервера требуется авторизация, и p_no_proxy - хосты, которые не используют прокси-сервер.

```
              # File net/http.rb, line 653
def HTTP.new(address, port = nil, p_addr = :ENV, p_port = nil, p_user = nil,
p_pass = nil, p_no_proxy = nil)
  http = super address, port

  if proxy_class? then # from Net::HTTP::Proxy()
    http.proxy_from_env = @proxy_from_env
    http.proxy_address  = @proxy_address
    http.proxy_port     = @proxy_port
    http.proxy_user     = @proxy_user
    http.proxy_pass     = @proxy_pass
  elsif p_addr == :ENV then
    http.proxy_from_env = true
  else
    if p_addr && p_no_proxy && !URI::Generic.use_proxy?(p_addr, p_addr, p_port,
p_no_proxy)
      p_addr = nil
      p_port = nil
    end
    http.proxy_address = p_addr
    http.proxy_port    = p_port || default_port
    http.proxy_user    = p_user
    http.proxy_pass    = p_pass
  end

  http
end
```

new(address, port = nil)

Создает новый объект Net::HTTP для указанного адреса сервера, не открывая TCP-соединение и не инициализируя сеанс HTTP. Адрес должен быть именем хоста DNS или IP-адресом.

```ruby
              # File net/http.rb, line 681
def initialize(address, port = nil)
  @address = address
  @port    = (port || HTTP.default_port)
  @ipaddr = nil
  @local_host = nil
  @local_port = nil
  @curr_http_version = HTTPVersion
  @keep_alive_timeout = 2
  @last_communicated = nil
  @close_on_empty_response = false
  @socket  = nil
  @started = false
  @open_timeout = 60
  @read_timeout = 60
  @write_timeout = 60
  @continue_timeout = nil
  @max_retries = 1
  @debug_output = nil

  @proxy_from_env = false
  @proxy_uri      = nil
  @proxy_address  = nil
  @proxy_port     = nil
  @proxy_user     = nil
  @proxy_pass     = nil

  @use_ssl = false
  @ssl_context = nil
  @ssl_session = nil
  @sspi_enabled = false
  SSL_IVNAMES.each do |ivname|
    instance_variable_set ivname, nil
  end
end
```

newobj(address, port = nil, p_addr = :ENV, p_port = nil, p_user = nil, p_pass = nil, p_no_proxy = nil)

Alias for: *new*