

基于版本1.0版本

**tornado.options** — Command-line parsing

tornado/options.py里面

有个全局的options = \_Options.instance() #用来存储\_Option对象

类

class **\_Options**(dict)

class **\_Option**(object)

def **define**(name, default=None, type=str, help=None, metavar=None,

multiple=False):

options[name] = \_Option(name, ...) #存储定义对象

def **parse\_command\_line**(args=None):

for arg in arg[1:]:

name, equals, value = arg.partition("=")

option = options[name]

option.parse(value) 转换option对象的值

tornado.web

绑定路径和回调函数

class **Application**(object):

def **\_\_init\_\_**(*self*, handlers=None, default\_host=*""*, transforms=None,

wsgi=False, \*\*settings):

self.handlers = []

for spec in handlers:

pattern , handler ,kwargs = spec

spec = URLSpec(pattern, handler, kwargs)

handlers.append(spec)

self.handlers.append((re.compile(".\*$"), handlers))

def **\_\_call\_\_**(*self*, request):

*"""Called by HTTPServer to execute the request."""*

handlers = *self*.\_get\_host\_handlers(request)

for spec in handlers:

match = spec.regex.match(request.path)

if match:

handler = spec.handler\_class(*self*, request, \*\*spec.kwargs)

break

handler.\_execute(transforms, \*args, \*\*kwargs)

return handler

class **RequestHandler**(object):

def **\_\_init\_\_**(*self*, application, request, transforms=None):

*self*.\_finished = False

*self*.\_auto\_finish = True

def **\_execute**(*self*, transforms, \*args, \*\*kwargs):

*"""Executes this request with the given output transforms."""*

if not *self*.\_finished:

getattr(*self*, *self*.request.method.lower())(\*args, \*\*kwargs)

if *self*.\_auto\_finish and not *self*.\_finished:

*self*.finish()

def **finish**(*self*, chunk=None):

*"""Finishes this response, ending the HTTP request."""*

assert not *self*.\_finished

。。。。

*self*.\_finished = True

* [**tornado.iostream** — Convenient wrappers for non-blocking sockets](http://www.tornadoweb.org/documentation/iostream.html)

tornado.httpserver

class **HTTPServer**(object):

def **\_\_init\_\_**(*self*, request\_callback, no\_keep\_alive=False, io\_loop=None, xheaders=False, ssl\_options=None):

def **listen**(*self*, port, address=*""*):

*self*.bind(port, address)

*self*.start(1)

def **start**(*self*, num\_processes=1):

if not *self*.io\_loop:

*self*.io\_loop = ioloop.IOLoop.instance()

*self*.io\_loop.add\_handler(*self*.\_socket.fileno(),

*self*.\_handle\_events,

ioloop.IOLoop.READ)

def **\_handle\_events**(*self*, fd, events):

while True:

try:

connection, address = *self*.\_socket.accept()

except socket.error, e:

if e[0] in (errno.EWOULDBLOCK, errno.EAGAIN):

return

raise

stream = iostream.IOStream(connection, io\_loop=*self*.io\_loop)

HTTPConnection(stream, address, *self*.request\_callback,

*self*.no\_keep\_alive, *self*.xheaders)

* [**tornado.ioloop** — Main event loop](http://www.tornadoweb.org/documentation/ioloop.html)

tornado.ioloop

class **IOLoop**(object):

def **\_\_init\_\_**(*self*, impl=None):

*self*.\_impl = impl or \_poll()

*self*.\_handlers = {}

*self*.\_events = {}

r, w = os.pipe()

*self*.\_waker\_reader = os.fdopen(r, *"r"*, 0)

*self*.\_waker\_writer = os.fdopen(w, *"w"*, 0)

*self*.add\_handler(r, *self*.\_read\_waker, *self*.READ)

def **add\_handler**(*self*, fd, handler, events):

*"""Registers the given handler to receive the given events for fd."""*

*self*.\_handlers[fd] = handler

*self*.\_impl.register(fd, events | *self*.ERROR)

def **add\_callback**(*self*, callback):

*"""Calls the given callback on the next I/O loop iteration."""*

*self*.\_callbacks.add(callback)

*self*.\_wake()

def **start**(*self*):

while True:

poll\_timeout = 0.2

callbacks = list(*self*.\_callbacks)

for callback in callbacks:

if callback in *self*.\_callbacks:

*self*.\_callbacks.remove(callback)

*self*.\_run\_callback(callback)

if *self*.\_callbacks:

poll\_timeout = 0.0

if *self*.\_timeouts:

now = time.time()

while *self*.\_timeouts and *self*.\_timeouts[0].deadline <= now:

timeout = *self*.\_timeouts.pop(0)

*self*.\_run\_callback(timeout.callback)

if *self*.\_timeouts:

milliseconds = *self*.\_timeouts[0].deadline - now

poll\_timeout = min(milliseconds, poll\_timeout)

event\_pairs = *self*.\_impl.poll(poll\_timeout)

*self*.\_events.update(event\_pairs)

while *self*.\_events:

fd, events = *self*.\_events.popitem()

*self*.\_handlers[fd](fd, events)

#处理回调

#处理定时器

#处理epoll

**IOLoop**

def **start**(*self*):

def **add\_handler**(*self*, fd, handler, events):

*"""A collection of request handlers that make up a web application.*

**Application**

*"""A non-blocking, single-threaded HTTP server*

**HTTPServer**(object):

def **listen**(*self*, port, address=*""*)

