txZMQ Documentation

Release 0.8.0

Andrey Smirnov

Apr 21, 2017

Contents

txZMQ allows to integrate easily ØMQ sockets into Twisted event loop (reactor).

txZMQ supports both CPython and PyPy, and ØMQ library version 2.2.x or 3.2.x.

txZMQ introduces support for general 0MQ sockets by class *txzmq.ZmqConnection* that can do basic event loop integration, sending-receiving messages in non-blocking manner, scatter-gather for multipart messages.

txZMQ uses ØMQ APIs to get file descriptor that is used to signal pending actions from ØMQ library IO thread running in separate thread. This is used in a custom file descriptor reader, which is then added to the Twisted reactor.

From this class, one may implement the various patterns defined by ØMQ. For example, special descendants of the *txzmq.ZmqConnection* class, *txzmq.ZmqPubConnection* and *txzmq.ZmqSubConnection*, add special nice features for PUB/SUB sockets.

Request/reply pattern is achieved via DEALER/ROUTER sockets and classes *txzmq.ZmqREQConnection*, *txzmq.ZmqREPConnection*, which provide REQ-REP like semantics in asynchronous case.

Other socket types could be easily derived from *txzmq.ZmqConnection*.

Contents:

CHAPTER 1

Installation

Short version:

```
pip install txZMQ
```

Requirements

C libraries required:

- ØMQ library 2.2.x or 3.2.x

Python packages required:

- pyzmq >= 13
- Twisted

MacOS X

On Mac OS X with Homebrew, please run:

```
brew install --with-pgm zeromq
```

This would install ØMQ 2.2.0, for 3.2.x please run:

```
brew install --with-pgm --devel zeromq
```

Ubuntu/Debian

Install ØMQ library with headers:

```
apt-get install libzmq-dev
```

Package name could aslo be `libzmq3-dev` for version 3.x.

CHAPTER 2

Examples

Publish-Subscribe

Here is an example of using txZMQ with publish and subscribe (examples/push_pull.py):

```
#!env/bin/python

"""
Example txzmq client.

    examples/pub_sub.py --method=bind --endpoint=ipc:///tmp/sock --mode=publisher

    examples/pub_sub.py --method=connect --endpoint=ipc:///tmp/sock --mode=subscriber
"""
import os
import sys
import time
from optparse import OptionParser

from twisted.internet import reactor

rootdir = os.path.realpath(os.path.join(os.path.dirname(sys.argv[0]), '..'))
sys.path.append(rootdir)
os.chdir(rootdir)

from txzmq import ZmqEndpoint, ZmqFactory, ZmqPubConnection, ZmqSubConnection


parser = OptionParser("")
parser.add_option("-m", "--method", dest="method", help="0MQ socket connection:␣
↪bind|connect")
parser.add_option("-e", "--endpoint", dest="endpoint", help="0MQ Endpoint")
parser.add_option("-M", "--mode", dest="mode", help="Mode: publisher|subscriber")
parser.set_defaults(method="connect", endpoint="epgm://eth1;239.0.5.3:10011")
```

```
(options, args) = parser.parse_args()

zf = ZmqFactory()
e = ZmqEndpoint(options.method, options.endpoint)

if options.mode == "publisher":
    s = ZmqPubConnection(zf, e)

    def publish():
        data = str(time.time())
        print "publishing %r" % data
        s.publish(data)

        reactor.callLater(1, publish)

    publish()
else:
    s = ZmqSubConnection(zf, e)
    s.subscribe("")

    def doPrint(*args):
        print "message received: %r" % (args, )

    s.gotMessage = doPrint

reactor.run()
```

The same example is available in the source code. You can run it from the checkout directory with the following commands (in two different terminals):

```
examples/pub_sub.py --method=bind --endpoint=ipc:///tmp/sock --mode=publisher

examples/pub_sub.py --method=connect --endpoint=ipc:///tmp/sock --mode=subscriber
```

Push-Pull

Example for push and pull socket is available in examples/push_pull.py.

```
#!env/bin/python

"""
Example txzmq client.

    examples/push_pull.py --method=bind --endpoint=ipc:///tmp/sock
    --mode=push

    examples/push_pull.py --method=connect --endpoint=ipc:///tmp/sock
    --mode=pull
"""
import os
import socket
import sys
import time
import zmq
from optparse import OptionParser
```

```
from twisted.internet import reactor

rootdir = os.path.realpath(os.path.join(os.path.dirname(sys.argv[0]), '..'))
sys.path.insert(0, rootdir)
os.chdir(rootdir)

from txzmq import ZmqEndpoint, ZmqFactory, ZmqPushConnection, ZmqPullConnection


parser = OptionParser("")
parser.add_option("-m", "--method", dest="method", help="0MQ socket connection:␣
↪bind|connect")
parser.add_option("-e", "--endpoint", dest="endpoint", help="0MQ Endpoint")
parser.add_option("-M", "--mode", dest="mode", help="Mode: push|pull")
parser.set_defaults(method="connect", endpoint="ipc:///tmp/txzmq-pc-demo")

(options, args) = parser.parse_args()

zf = ZmqFactory()
e = ZmqEndpoint(options.method, options.endpoint)

if options.mode == "push":
    s = ZmqPushConnection(zf, e)

    def produce():
        data = [str(time.time()), socket.gethostname()]
        print "producing %r" % data
        try:
            s.push(data)
        except zmq.error.Again:
            print "Skipping, no pull consumers..."

        reactor.callLater(1, produce)

    reactor.callWhenRunning(reactor.callLater, 1, produce)
else:
    s = ZmqPullConnection(zf, e)

    def doPrint(message):
        print "consuming %r" % (message,)

    s.onPull = doPrint

reactor.run()
```

CHAPTER 3

API Documentation

ZeroMQ integration into Twisted reactor.

Factory

All ØMQ connections should belong to some context, txZMQ wraps that into concept of factory that tracks all connections created and wraps context.

Factory could be used as an easy way to close all connections and clean up Twisted reactor.

class txzmq.**ZmqFactory**

I control individual ZeroMQ connections.

Factory creates and destroys ZeroMQ context.

Variables

- **reactor** – reference to Twisted reactor used by all the connections
- **ioThreads** (*int*) – number of IO threads ZeroMQ will be using for this context
- **lingerPeriod** (*int*) – number of milliseconds to block when closing socket (terminating context), when there are some messages pending to be sent
- **connections** (*set*) – set of instanciated *ZmqConnection*
- **context** – ZeroMQ context

__init__(*self*)

Constructor.

Create ZeroMQ context.

shutdown()

Shutdown factory.

This is shutting down all created connections and terminating ZeroMQ context. Also cleans up Twisted reactor.

registerForShutdown()
Register factory to be automatically shut down on reactor shutdown.

It is recommended that this method is called on any created factory.

Base Connection

ZmqConnection isn't supposed to be used explicitly, it is base for different socket types.

class txzmq.**ZmqEndpointType**
Endpoint could be "bound" or "connected".

bind = 'bind'
Bind, listen for connection.

connect = 'connect'
Connect to another endpoint.

class txzmq.**ZmqEndpoint**
ZeroMQ endpoint used when connecting or listening for connections.

Consists of two members: *type* and *address*.

Variables

- **type** – Could be either *ZmqEndpointType.bind* or *ZmqEndpointType.connect*.
- **address** (*str*) – ZeroMQ address of endpoint, could be IP address, filename, see ZeroMQ docs for more details.

class txzmq.**ZmqConnection**(*factory*, *endpoint=None*, *identity=None*)
Connection through ZeroMQ, wraps up ZeroMQ socket.

This class isn't supposed to be used directly, instead use one of the descendants like *ZmqPushConnection*.

ZmqConnection implements glue between ZeroMQ and Twisted reactor: putting polling ZeroMQ file descriptor into reactor, processing events, reading data from socket.

Variables

- **socketType** – socket type, from ZeroMQ
- **allowLoopbackMulticast** (*bool*) – is loopback multicast allowed?
- **multicastRate** (*int*) – maximum allowed multicast rate, kbps
- **highWaterMark** (*int*) – hard limit on the maximum number of outstanding messages 0MQ shall queue in memory for any single peer
- **tcpKeepalive** (*int*) – if set to 1, enable TCP keepalive, otherwise leave it as default
- **tcpKeepaliveCount** (*int*) – override TCP_KEEPCNT socket option (where supported by OS)
- **tcpKeepaliveIdle** (*int*) – override TCP_KEEPCNT(or TCP_KEEPALIVE on some OS) socket option(where supported by OS).
- **tcpKeepaliveInterval** (*int*) – override TCP_KEEPINTVL socket option(where supported by OS)
- **reconnectInterval** (*int*) – set reconnection interval
- **reconnectIntervalMax** (*int*) – set maximum reconnection interval

- **factory** (*ZmqFactory*) – ZeroMQ Twisted factory reference
- **socket** (*zmq.Socket*) – ZeroMQ Socket
- **endpoints** (list of *ZmqEndpoint*) – ZeroMQ addresses for connect/bind
- **fd** (*int*) – file descriptor of zmq mailbox
- **queue** (*deque*) – output message queue

__init__(*self*, *factory*, *endpoint=None*, *identity=None*)
Constructor.

One endpoint is passed to the constructor, more could be added via call to *addEndpoints()*.

Parameters

- **factory** (*ZmqFactory*) – ZeroMQ Twisted factory
- **endpoint** (*ZmqEndpoint*) – ZeroMQ address for connect/bind
- **identity** (*str*) – socket identity (ZeroMQ), don't set unless you know how it works

addEndpoints(*endpoints*)
Add more connection endpoints.

Connection may have many endpoints, mixing ZeroMQ protocols (TCP, IPC, ...) and types (connect or bind).

Parameters **endpoints** (list of *ZmqEndpoint*) – list of endpoints to add

shutdown()
Shutdown (close) connection and ZeroMQ socket.

fileno()
Implementation of IFileDescriptor.

Returns ZeroMQ polling file descriptor.

Returns The platform-specified representation of a file descriptor number.

connectionLost(*reason*)
Called when the connection was lost.

Implementation of IFileDescriptor.

This is called when the connection on a selectable object has been lost. It will be called whether the connection was closed explicitly, an exception occurred in an event handler, or the other end of the connection closed it first.

doRead()
Some data is available for reading on ZeroMQ descriptor.

ZeroMQ is signalling that we should process some events, we're starting to receive incoming messages.

Implementation of IReadDescriptor.

logPrefix()
Implementation of ILoggingContext.

Returns Prefix used during log formatting to indicate context.

Return type str

send(*message*)
Send message via ZeroMQ socket.

Sending is performed directly to ZeroMQ without queueing. If HWM is reached on ZeroMQ side, sending operation is aborted with exception from ZeroMQ (EAGAIN).

After writing read is scheduled as ZeroMQ may not signal incoming messages after we touched socket with write request.

Parameters **message** (*str or list of str*) – message data, could be either list of str (multipart message) or just str

messageReceived(*message*)
Called when complete message is received.

Not implemented in *ZmqConnection*, should be overridden to handle incoming messages.

Parameters **message** – message data

Publish-Subscribe

For information on publish-subscribe in ØMQ, please read either reference or guide (look for publish-subscribe).

Note: These classes use PUB and SUB sockets from ØMQ. Special framing is implemented to support sending tag: tag and message are separated by zero byte and sent over as single message. This is related to the way PUB-SUB works with PGM (UDP multicast): multipart messages are sent as multiple datagrams and they get mixed together if several publishers exist in the same broadcast domain.

class txzmq.**ZmqPubConnection**(*factory*, *endpoint=None*, *identity=None*)
Bases: txzmq.connection.ZmqConnection

Publishing in broadcast manner.

publish(*message*, *tag=''*)
Publish *message* with specified *tag*.

Parameters

- **message** (*str*) – message data
- **tag** (*str*) – message tag

class txzmq.**ZmqSubConnection**(*factory*, *endpoint=None*, *identity=None*)
Bases: txzmq.connection.ZmqConnection

Subscribing to messages published by publishers.

Subclass this class and implement *gotMessage()* to handle incoming messages.

subscribe(*tag*)
Subscribe to messages with specified tag (prefix).

Function may be called several times.

Parameters **tag** (*str*) – message tag

unsubscribe(*tag*)
Unsubscribe from messages with specified tag (prefix).

Function may be called several times.

Parameters **tag** (*str*) – message tag

messageReceived(*message*)
Overridden from *ZmqConnection* to process and unframe incoming messages.

All parsed messages are passed to *gotMessage()*.

Parameters **message** – message data

gotMessage(*message*, *tag*)
Called on incoming message recevied by subscriber.

Should be overridden to handle incoming messages.

Parameters

- **message** – message data
- **tag** – message tag

Push-Pull

For information on push and pull sockets in ØMQ, please read either reference or guide (look for pull or push).

class txzmq.**ZmqPushConnection**(*factory*, *endpoint=None*, *identity=None*)
Bases: txzmq.connection.ZmqConnection

Pushing messages to the socket.

Wrapper around ZeroMQ PUSH socket.

push(*message*)
Push a message L{message}.

Parameters **message** (*str*) – message data

class txzmq.**ZmqPullConnection**(*factory*, *endpoint=None*, *identity=None*)
Bases: txzmq.connection.ZmqConnection

Pull messages from a socket.

Wrapper around ZeroMQ PULL socket.

Subclass and override *onPull()*.

messageReceived(*message*)
Called on incoming message from ZeroMQ.

Parameters **message** – message data

onPull(*message*)
Called on incoming message received by puller.

Parameters **message** – message data

Request-Reply and Router-Dealer

For information on these socket types in ØMQ, please read either reference or guide (look for router/dealer and request/reply).

class txzmq.**ZmqREQConnection**(**args*, ***kwargs*)
Bases: txzmq.connection.ZmqConnection

A Request ZeroMQ connection.

This is implemented with an underlying DEALER socket, even though semantics are closer to REQ socket.

Socket mimics request-reply behavior by sending each message with unique uuid and recording Deferred associated with the message. When reply comes, it uses that Deferred to pass response back to the caller.

Variables **defaultRequestTimeout** – default timeout for requests, disabled by default (seconds)

sendMsg(**messageParts*, ***kwargs*)
Send request and deliver response back when available.

Parameters

- **messageParts** (*tuple*) – message data
- **timeout** (*float*) – as keyword argument, timeout on request

Returns Deferred that will fire when response comes back

messageReceived(*message*)
Called on incoming message from ZeroMQ.

Dispatches message to back to the requestor.

Parameters **message** – message data

class txzmq.**ZmqREPConnection**(**args*, ***kwargs*)
Bases: txzmq.connection.ZmqConnection

A Reply ZeroMQ connection.

This is implemented with an underlying ROUTER socket, but the semantics are close to REP socket.

reply(*messageId*, **messageParts*)
Send reply to request with specified messageId.

Parameters

- **messageId** (*str*) – message uuid
- **messageParts** (*list*) – message data

messageReceived(*message*)
Called on incoming message from ZeroMQ.

Parameters **message** – message data

gotMessage(*messageId*, **messageParts*)
Called on incoming request.

Override this method in subclass and reply using *reply()* using the same messageId.

Parameters

- **messageId** (*str*) – message uuid
- **messageParts** – message data

class txzmq.**ZmqRouterConnection**(*factory*, *endpoint=None*, *identity=None*)
Bases: txzmq.router_dealer.ZmqBase

Raw ZeroMQ ROUTER connection.

class txzmq.**ZmqDealerConnection**(*factory*, *endpoint=None*, *identity=None*)
Bases: txzmq.router_dealer.ZmqBase

Raw ZeroMQ DEALER connection.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

Symbols

A

B

C

D

F

G

L

M

O

P

R

S

T

U

Z