

Writing Networked Applications in Python with Twisted

Tom Clark
entropymedia

What is Twisted?

- A set of libraries and helper programs for writing event driven network servers and clients
- Supports various basic protocols like TCP, UDP, and Unix sockets
- Includes classes for common application protocols like HTTP, ssh, IMAP, etc.
- Development is community supported through the Twisted Software Foundation
- MIT Licensed

Advantages of Twisted

- Well-tested, robust
- Long running and active development
- Easy to use
- Useful for both clients and servers

Let's get started

Three key ideas:

- Reactors
- Factories
- Protocols

Reactors

Twisted applications are event based and the event loop is called the *reactor*

The second-simplest Twisted app:

```
from twisted.internet import reactor

def hello():
    print "Hello, world!"

reactor.callLater(1, hello)
reactor.run()
```

Factories

- Listen for incoming connections or connect to remote hosts
- Handle errors in connections
- Clean up after connections are closed
- Hand off control to Protocol objects when a successful connection is made

Protocols

- Handle the client/server interaction
- Provide the interface between the body of your application and the network
- Subclass Protocol classes that handle low level details
- Override needed parent methods

An example: Insults

- Server
 - Set server to listen on a given port.
 - When a client connects and supplies a topic the server will respond with an insult about the topic.
 - Disconnect when the client sends “quit”.
- Client
 - Connect to the server.
 - Pass input entered via stdin to the server.
 - Print the server's responses to stdout.

Server: set up and run the reactor

```
port = 8080
```

```
greetings = ['Welcome to the CPOSC Insult Server.  
Enter a topic to receive an insult.',  
'Enter "quit" to disconnect.']
```

```
reactor.listenTCP(port, InsultServerFactory(greetings))  
reactor.run()
```

Server: provide the Factory

```
class InsultServerFactory(Factory):  
    protocol = InsultServerProtocol  
    def __init__(self, greetings):  
        self.greeting_lines = greetings
```

Server: other Factory methods

- `startFactory()`
Called just before the factory starts listening
- `stopFactory()`
Called just after the factory stops listening

These are good places to open/close files and connect to databases.

Server: set up the Protocol

```
class InsultServerProtocol(LineReceiver):  
    def __init__(self):  
        self.insults = Insults()  
  
    def connectionMade(self):  
        for line in self.factory.greeting_lines:  
            self.sendLine(line)
```

Server: more Protocol

```
def lineReceived(self, line):  
    line = line.lower().strip()  
    if line == "quit":  
        self.sendLine("Goodbye, loser.")  
        self.transportloseConnection()  
    elif line == "help":  
        self.getHelp()  
    else:  
        self.sendLine(self.insults.getInsult(line))
```

Run the server

Client: Reactor

```
host = "127.0.0.1"
```

```
port = 8080
```

```
reactor.connectTCP(host, port, InsultClientFactory())
```

```
reactor.run()
```

Client: Factory

```
class InsultClientFactory(ClientFactory):  
    protocol = InsultClientProtocol  
  
    def clientConnectionLost(self, transport, reason):  
        reactor.stop()  
  
    def clientConnectionFailed(self, transport, reason):  
        print reason.getErrorMessage()  
        reactor.stop()
```


Client: Protocol

```
class InsultClientProtocol(Protocol):  
    def dataReceived(self, data):  
        data = data.strip()  
        print data  
        if data == "Goodbye, loser." :  
            return  
        else:  
            input = raw_input(">").strip()  
            self.transport.write(input)  
            self.transport.write("\r\n")
```

Try the client

Taking it up a notch

Using *twisted* to run our server as a daemon

twistd allows us to

- Run servers in the background
- Log events and errors
- Run as an unprivileged user
- Run in a *chroot* environment

The plan

- Protocol and ProtocolFactory objects don't need to change
- Add an new class that implements `twisted.application.service.IService`
- Write a script that provides a `service.Application` object named “application”

Service Class

[illegible]

Script for *twistd*

```
from twisted.application import service
import insultserver_twistd

port = 8080

greetings = ['Welcome to the CPOSC Insult Server.
              Enter a topic to receive an insult.',
              'Enter "quit" to disconnect.']

application = service.Application("InsultServer")

insult_service = insultserver_twistd.InsultService(port,
greetings)

insult_service.setServiceParent(application)
```

Run the script

```
twistd -y insultserver_app.py
```

- Runs in the background
- Stores the pid in twistd.pid
- Logs to twistd.log

Running as another user

Change

```
application = service.Application("InsultServer")
```

to

```
application = service.Application("InsultServer",  
                                   uid=UID, gid=GID)
```

Running in a chroot environment

- Invoke twistd with the `–chroot dirname` option
- Works nicely with virtualenv

Resources

- Twisted web site
 - <http://twistedmatrix.com/trac/>
 - <http://twistedmatrix.com/documents/current/core/howto/index.html>
 - <http://twistedmatrix.com/documents/current/api/t>
- This presentation
 - <http://github.com/tclark/cposc2010>