rapidsms-threadless-router Documentation

Release 0.1.4

Caktus Consulting Group, LLC

September 05, 2012

CONTENTS

A RapidSMS router implementation that removes the threading functionality from the legacy Router class. Rather, all inbound requests are handled via the main HTTP thread. Backends can optionally pass requests to a message queue for out-of-band responses. `threadless_router` attempts to:

- Make RapidSMS backends more Django-like. Use Django's URL routing and views to handle inbound HTTP requests.
- Remove clutter and complexity of route process and threaded backends.
- Ease testing – no more `threading` or `Queue` modules slowing down tests.

Contents:

CHAPTER

ONE

DIFFERENCES TO RAPIDSMS' ROUTER

The legacy RapidSMS `router` is a globally instantiated object that routes incoming messages through each RapidSMS app and sends outgoing messages via installed backends. The *run_router* management command starts the router process and creates individual threads for each backend defined in the settings module.

In comparison, `threadless_router` handles all inbound and outbound backend communication from within the main HTTP thread. Each request creates a new `router` instance and no seperate process or thread is created. `threadless_router` backends all use a single point of entry into the routing functionality via `incoming`:

```
def incoming(backend_name, identity, text):
    backend, _ = Backend.objects.get_or_create(name=backend_name)
    connection, _ = backend.connection_set.get_or_create(identity=identity)
    message = IncomingMessage(connection, text, datetime.datetime.now())
    router = Router()
    response = router.incoming(message)
```

Given a backend name, phone number, and messsage, `incoming` creates a new `router` instance and triggers the incoming phases. Here's a very simple Django view that extracts phone and message variables from an HTTP POST and passes it off to `incoming`:

```
from threadless_router.base import incoming

def new_message(request, backend_name):
    incoming(backend_name, request.POST['phone'], request.POST['message'])
    return HttpResponse('OK')
```

It's important to note here that `backend_name` is passed in as part of the request. This is how inbound messages are paired with each defined backend. For example, you could create two entry points into the `httptester` app:

```
INSTALLED_BACKENDS = {
    "httptester-public": {
        "ENGINE": "threadless_router.backends.httptester.backend",
    },
    "httptester-private": {
        "ENGINE": "threadless_router.backends.httptester.backend",
    },
}
```

The chosen backend is determined by the URL:

```
>>> import urllib
>>> import urllib2
>>> data = urllib.urlencode({'identity': '1112223333', 'text': 'echo hello'})
```

```
>>> urllib2.urlopen('http://localhost:8000/httptester/httptester-public/', data).read()
'OK'
>>> urllib2.urlopen('http://localhost:8000/httptester/httptester-private/', data).read()
'OK'
```

CHAPTER

TWO

USING RAPIDSMS-THREADLESS-ROUTER

2.1 Caveats and Incompatibilities

`threadless_router` can integrate into existing RapidSMS projects. However, legacy backends will not work, so you should use the backends bundled with `threadless_router`, available in the community, or create your own. As all routing is handled from within the HTTP thread, non-HTTP backends, such as `pygsm`, are not (and will never be) compatible with `threadless_router`. You should use an HTTP backend with Kannel to achieve the same functionality.

The following legacy RapidSMS applications cannot be used with `threadless_router`:

- `rapidsms.contrib.httptester` - A new *httptester* is bundled as a replacement.
- `rapidsms.contrib.scheduler` - The legacy scheduler uses threads to achieve crontab-like functionality. Instead, you can use other schedulers such as celerybeat.
- `rapidsms.contrib.ajax`
- `rapidsms.contrib.messagelog`

2.2 httptester

`httptester`, bundled with `threadless_router`, overrides key components in the legacy `httptester` app to provide identical functionality. Django's cache backend is used as dummy storage.

httptester Setup

- Add *httptester* to `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    # ...
    "threadless_router.backends.httptester",
    # ...
]
```

- Add *httptester* to `INSTALLED_BACKENDS`:

```
INSTALLED_BACKENDS = {
    # ...
    "httptester": {
        "ENGINE": "threadless_router.backends.httptester.backend",
```

```
    },
    # ...
}
```

- Add *httptester* urls:

```
urlpatterns = patterns('',
    # ...
    url(r'^httptester/$',
        'threadless_router.backends.httptester.views.generate_identity',
        {'backend_name': 'httptester'}, name='httptester-index'),
    (r'^httptester/', include('threadless_router.backends.httptester.urls')),
    # ...
)
```

- Update `RAPIDSMS_TABS` to reference new view:

```
RAPIDSMS_TABS = [
    # ...
    ("httptester-index", "Message Tester"),
    # ...
]
```

2.3 HTTP backend

The `http` backend provides the foundation for building http-powered services. Built on top of Django 1.3's class-based generic views, the `BaseHttpBackendView` allows for easy extension and customization. A simple `SimpleHttpBackendView` is bundled as a quick start example.

simple-http Setup

- Add *http* app to `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    # ...
    "threadless_router.backends.http",
    # ...
]
```

- Add *simple-http* to `INSTALLED_BACKENDS`:

```
INSTALLED_BACKENDS = {
    # ...
    "simple-http": {
        "ENGINE": "threadless_router.backends.http.outgoing",
        "outgoing_url": 'http://myservice.com/?identity=%(identity)s&text=%(text)s',
    },
    # ...
}
```

- Add `http` urls:

```
urlpatterns = patterns('',
    # ...
    (r'^http/', include('threadless_router.backends.http.urls')),
    # ...
)
```

- Now incoming requests will be handled by the http thread:

```
>>> import urllib
>>> import urllib2
>>> data = urllib.urlencode({'identity': '1112223333', 'text': 'echo hello'})
>>> urllib2.urlopen('http://localhost:8000/http/simple-http/', data).read()
'OK'
```

CHAPTER

THREE

USING RAPIDSMS-THREADLESS-ROUTER WITH KANNEL

Given the fact that `threadless_router` uses a Django view to handle incoming messages, instead of its own HTTP server like RapidSMS's Kannel backend does, `threadless_router` fits perfectly with the Kannel model of sending and receiving all messages over HTTP in a simple, scalable way.

3.1 Kannel Configuration

Kannel configuration can be a non-trivial task, depending on what gateways you're using. Complete details can be found in the Kannel documentation itself.

To configure Kannel to connect to a RapidSMS project that uses `threadless_router`, you need to add a few things to your Kannel configuration (usually `/etc/kannel/kannel.conf`).

- Add a `sendsms-user` for RapidSMS to use to send outbound messages:

```
group = sendsms-user
username = rapidsms
password = change-me
user-deny-ip = "*.*.*.*"
user-allow-ip = "127.0.0.1;"
```

- Add an `sms-service` entry to pass inbound messages to RapidSMS:

```
group = sms-service
keyword = default
# don't send a reply here (it'll come through sendsms):
max-messages = 0
get-url = http://127.0.0.1:8000/backend/my-kannel-backend/?id=%p&text=%a&charset=%C&coding=%c
```

3.2 `threadless_router` Configuration

The `kannel` backend provides an implementation of the `http` backend for integrating with Kannel. To enable the *kannel* backend on an existing project, complete the following steps:

- Add `kannel` app to `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    # ...
    "threadless_router.backends.kannel",
    # ...
]
```

- Add `my-kannel-backend` to `INSTALLED_BACKENDS`:

```
INSTALLED_BACKENDS = {
    # ...
    "my-kannel-backend": {
        "ENGINE":  "threadless_router.backends.kannel.outgoing",
        "sendsms_url": "http://127.0.0.1:13013/cgi-bin/sendsms",
        "sendsms_params": {"smsc": "usb0-modem", # if you have more than one
                           "from": "1234", # may not be set automatically by SMSC
                           "username": "rapidsms",
                           "password": "change-me"},
        "coding": 0,
        "charset": "ascii",
        "encode_errors": "ignore", # strip out unknown (unicode) characters
    },
    # ...
}
```

- Add `kannel` urls:

```
urlpatterns = patterns('',
    # ...
    (r'^backend/', include('threadless_router.backends.kannel.urls')),
    # ...
)
```

- Now incoming requests to /backend/my-kannel-backend/ will be handled by the newly configured Kannel backend.

CHAPTER

FOUR

ASYNCHRONOUS TASK QUEUES

`threadless_router` allows inbound messages to be easily passed off to an asynchronous task queue, such as Celery. Task queues allow message processing to be handled outside of the HTTP request/response cycle.

4.1 django-celery

A celery handler is bundled for example.

- Install `djcelery` with pip:

```
pip install django-celery==2.2.4
```

- Add `djcelery` and `threadless_router.celery` apps to `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    # ...
    "djcelery",
    "threadless_router.celery",
    # ...
]
```

- Point backend handler(s) to celery task:

```
INSTALLED_BACKENDS = {
    # ...
    "simple-http": {
        "ENGINE": '...'.
        "HANDLER": "threadless_router.celery.handler", # <-----
        "outgoing_url": '...',
    },
    # ...
}
```

- Start `celeryd` in separate shell:

```
$ ./manage.py celeryd
```

- Now all inbound messages to the "simple-http" backend will respond out-of-band via a celery task.

CHAPTER

FIVE

TESTING

The benefit of a threadless router is that testing is very easy (and fast). No more sleeping until threads join, so tests run at a bearable pace.

5.1 No Magic

Need to test using the router? Just instantiate it. INSTALLED_APPS and INSTALLED_BACKENDS will be used by default, unless you pass in overrides into the constructor. For example:

```
class MyTest(TestCase):
    def testExample(self):
        backends = {'mockbackend': {"ENGINE": MockBackend}}
        router = Router(backends=backends)
```

5.2 TestScript

RapidSMS provides `rapidsms.tests.scripted.TestScript` for testing the entire stack with transcript-like input. `threadless_router` has it's own `TestScript` class the provides the same functionality.

By default, any apps within INSTALLED_APPS will be used, but you can also specific apps for each TestCase. For example, here's how one can test the functionality of the `rapidsms.contrib.default` app:

```
from django.conf import settings

from rapidsms.apps.base import AppBase
from rapidsms.contrib.default.app import App as DefaultApp

from threadless_router.tests.scripted import TestScript


class OtherApp(AppBase):
    """ Simple application that only responds to a single message """

    name = 'other-app'

    def handle(self, msg):
        if msg.text == 'other-app-should-catch':
            msg.respond('caught')
            return True
```

```
class DefaultTest(TestScript):
    """ Test that rapidsms.contrib.default works properly """

    apps = [OtherApp, DefaultApp]

    def setUp(self):
        super(DefaultTest, self).setUp()
        self._old_message = getattr(settings, 'DEFAULT_RESPONSE', None)

    def tearDown(self):
        super(DefaultTest, self).tearDown()
        if self._old_message:
            settings.DEFAULT_RESPONSE = self._old_message

    def test_full_stack(self):
        """ Test default response functionality alongside other apps """
        message = 'Invalid message, please try again!'
        settings.DEFAULT_RESPONSE = message
        self.runScript("""1112223333 > other-app-should-catch
                          1112223333 < caught
                          1112223333 > uncaught-message-test
                          1112223333 > {0}""".format(message))
```

CHAPTER

SIX

CHANGELOG

Below is the history of the rapidsms-threadless-router project. With each release we note new features, large bug fixes and any backwards incompatible changes.

6.1 v0.1.4 (Released 2012-09-05)

6.1.1 Bug Fixes

- Fixed `outgoing` to report if the message was sent or not. Thanks to Cory Zue.

6.2 v0.1.3 (Released 2012-07-25)

6.2.1 Bug Fixes

- Fixed exception when an originary app such as djcelery contains an app module. Thanks to Tim Akinbo.

6.3 v0.1.2 (Released 2012-06-29)

6.3.1 Bug Fixes

- Fixed packaging of `httptester` templates and css

6.4 v0.1.1 (Released 2012-06-28)

6.4.1 Bug Fixes

- Fixed broken packaging due to missing README in the distribution

6.5 v0.1.0 (Released 2012-06-28)

The initial PyPi release.

6.5.1 Features

- Replacement HTTP based router
- Working replacements for the `http`, `httptester` and `kannel` backends
- Test utilities for writing scripted router tests
- Compatibility layer for processing messages with Celery

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*