DjangoFutures Documentation

Release 0.1.0

Jeff Buttars

Oct 31, 2017

Contents

Contents:

CHAPTER 1

Install

The current version of Django Futures is a development build!

Django Futures is still a proof of concept. That said, please install it, try it, and give feedback.

There is package available and is the easiest way to install Django Futures. Simply install with `pip` as usual.

```
pip install djang-futures
```

If you prefer the bleeding edge, install using the sources from GitHub.

`pip` can also be used to install from source.

1. Clone the repo: `git clone https://github.com/jeffbuttars/django-futures`
2. `pip` the package sources: `pip install django-futures/src`

If you don't want to use `pip`, you can run the setup script directly. Clone the directory as in step one above then:

2. Change to the package source directory: `cd django-futures/src`
3. Run the `setup.py` script: `python ./setup.py install`

CHAPTER 2

Examples

Here is a simple asynchronous view that fetches a web page.

```
from tornado import gen
from django_futures.http_client import HttpClient
from django.views.generic import TemplateView
from core.views import BaseTemplateView

class TestAsyncHttpClient(BaseTemplateView):

    template_name = "test_async_httpclient.html"
    num_client_options = (1, 5, 10, 25, 50, 100)

    @gen.coroutine
    def get(self, request):
        """
        Here we make an asynchrounous web call using the asynchrounous
        aware web client available with Django Futures
        """

        # Go and grab a web page asynchronously
        http_client = HttpClient()

        res = yield http_client.get('http://yahoo.com')
        ctx = {
            'web_response': res
        }

        # Build a Django response
        myres = super(TestAsyncHttpClient, self).get(request, **ctx)

        # In an asynchronous view, we must render a Django response using
        # the render() method of the request object.
        request.render(myres)
    # get()
# TestAsyncHttpClient
```

CHAPTER 3

run_tornado

CHAPTER 4

HTTP Client

`django_futures.http_client.py`

`http_client.py` – Asynchronous HTTP Client

This is a 'smart' HTTPClient wrapper that also makes requests a bit simpler by borrowing conventions from the 'requests' module.

If a tornado IOLoop is running, then async requests are made. Otherwise, tornado's `run_sync()` method is used to make the calls synchronous.

class `django_futures.http_client.`**`HttpClient`**(**args*, ***kwargs*)

Bases: `object`

Docstring for HttpClient

classmethod **`configure`**(*impl*, ***kwargs*)

`get`(*url*, *params=None*, ***kwargs*)

GET wrapper. Always returns a future.

Parameters

- **`url`** (*`type description`*) – arg description
- **`kwargs`** (*`type description`*) – arg description

Returns

Return type

`post`(*url*, *data=None*, *params=None*, ***kwargs*)

todo: Docstring for post

Parameters

- **`url`** (*`type description`*) – arg description

- ****kwargs** (*type description*) – arg description

Returns

Return type

post_json(*url*, *data*, *params=None*, ***kwargs*)
todo: Docstring for post_json

Parameters

- **url** (*type description*) – arg description
- **data** (*type description*) – arg description
- **params** (*type description*) – arg description
- ****kwargs** (*type description*) – arg description

Returns

Return type

CHAPTER 5

Tasks

`django_futures.decorators.py`

`decorators.py` – Simplifiy Asynchronous Code

class `django_futures.decorators.`**`ttask`**`(`**args*, ***kwargs*`)`

Bases: `object`

Run a task as a tornado callback. Great for async background code. If tornado is not running, then things are run synchronously.

Example: We define the task `send_signup_confirmation()` using the `@ttask()` decorator. When the task is called on line 21 the call will return imediately and the task will run at a later time after the view has finished.

```
1  from ttasks.decorators import ttask
2
3  @ttask()
4  def send_signup_confirmation(req, emsg):
5
6      url = "https://api.myemailserver.example.com
7      hc = HTTPClient()
8      resp = hc.fetch(
9          url,
10         method='POST',
11         body=tornado.escape.json_encode(emsg),
12     )
13
14     logger.debug("email result: %s", resp)
15
16
17 def a_view(request):
18     # Process some stuff
19     ...
```

```
20      # Call the task
21      send_signup_confirmation(request)
22
23      # create and return a response
24      ...
25      return response
```

class django_futures.decorators.**ctask**(**args*, ***kwargs*)

Bases: object

Creates a *ttask* using a method/function that is also a Tornado coroutine.

This is a convenience decorator and is equivelant to decorting a function with @tornado.gen.coroutine and @ttask()

ctask will run the tornado.gen.coroutine on the decorated function first then decorate it with *ttask*.

For example, use this if you have a task that needs to make asynchronous http client calls.

CHAPTER 6

Sources

Github

Django Futures is an open source project hosted on GitHub. Browse, download and fork the sources at the Django Futures GitHub repository

Rendered

You can view the source files as they are rendered by Sphinx by visiting the links below.

`django_futures` – Top Level Module

`django_futures.management.commands` – Management Commands

`django_futures.core`

`django_futures.core.handlers`

`Tornado 4.0 Handlers`

`Tornado 3.2 Handlers`

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index