Урок 14

Асинхронный сервер tornado

pip install tornado

```
import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("Hello, world")

application = tornado.web.Application([
    (r"/", MainHandler),
])

if __name__ == "__main__":
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

Асинхронный сервер tornado

RequestHandler.initialize()

```
class ProfileHandler(RequestHandler):
    def initialize(self, database):
        self.database = database

    def get(self, username):
        ...

app = Application([
    (r'/user/(.*)', ProfileHandler, dict(database=database)),
    ])
```

RequestHandler.prepare() **RequestHandler.on_finish()**

Асинхронный сервер tornado

RequestHandler.get(**args*, ***kwargs*)

RequestHandler.post(**args*, ***kwargs*)

RequestHandler.put(**args*, ***kwargs*)

class **tornado.web.Application**(*handlers=None*, *default_host=''*, *transforms=None*, ***settings*)[source]

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

```
application = web.Application([
    (r"/", MainPageHandler),
])
http_server = httpserver.HTTPServer(application)
http_server.listen(8080)
ioloop.IOLoop.instance().start()
```

Примеры на tornado

```
import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("Hello, world")

application = tornado.web.Application([
    (r"/", MainHandler),
])

if __name__ == "__main__":
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

https://github.com/tornadoweb/tornado/tree/master/demos/blog

Вебсокеты. Чат на tornado

pip install sockjs-tornado

This is implementation of the SockJS realtime transport library on top of the Tornado framework.

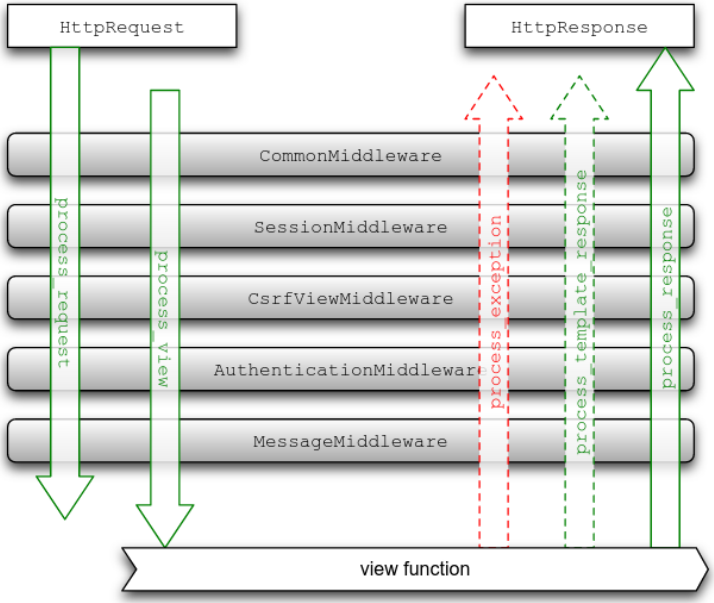
- websocket
- xhr-streaming
- iframe-eventsource
- iframe-htmlfile
- xhr-polling
- iframe-xhr-polling
- jsonp-polling

Запуск django с tornado

```
#! /usr/bin/env python
import os
import tornado.httpserver
import tornado.ioloop
import tornado.wsgi
import sys
import django.core.handlers.wsgi
from config.settings import BASE_DIR
from tornado import web
sys.path.append(BASE_DIR)
def main():
        os.environ['DJANGO_SETTINGS_MODULE'] = 'config.settings'
        application = django.core.handlers.wsgi.WSGIHandler()
        container = tornado.wsgi.WSGIContainer(application)
        http_server = tornado.httpserver.HTTPServer(container)
        http_server.listen(8888)
        tornado.ioloop.IOLoop.instance().start()
if __name__ == "__main__":
                main()
```

Middleware классы в django

Middleware is a framework of hooks into Django's request/response processing. It's a light, low-level "plugin" system for globally altering Django's input or output.



Middleware классы в django

Hooks and application order¶

During the request phase, before calling the view, Django applies middleware in the order it's defined in `MIDDLEWARE_CLASSES`, top-down. Two hooks are available:

- `process_request()`
- `process_view()`

During the response phase, after calling the view, middleware are applied in reverse order, from the bottom up. Three hooks are available:

- `process_exception()` (only if the view raised an exception)
- `process_template_response()` (only for template responses)
- `process_response()`

Middleware классы в django

middleware.py

```
from django.conf import settings

class BeforeFilter(object):
    def process_request(self, request):
        settings.my_var = 'Hello World'
        return None
```

views.py

```
from django.conf import settings
from django.http import HttpResponse

def myview(request):
    return HttpResponse(settings.my_var)
```

Контекстные процессоры

```
def media_url(request):
    from django.conf import settings
    return {'media_url': settings.MEDIA_URL}
TEMPLATE_CONTEXT_PROCESSORS = ('myapp.context_processors.media_url',)
from django.template import RequestContext
return render_to_response("my_app/my_template.html", {'some_var': 'foo'},
                          context_instance=RequestContext(request))
```

Что такое generic relations в Django.

```
>>> from django.contrib.contenttypes.models import ContentType
>>> user_type = ContentType.objects.get(app_label="auth", model="user")
>>> user_type
<ContentType: user>
>>> user_type.model_class()
<class 'django.contrib.auth.models.User'>
>>> user_type.get_object_for_this_type(username='Guido')
<User: Guido>
```

```
from django.db import models
from django.contrib.contenttypes.models import ContentType
from django.contrib.contenttypes import generic

class TaggedItem(models.Model):
    tag = models.SlugField()
    content_type = models.ForeignKey(ContentType)
    object_id = models.PositiveIntegerField()
    content_object = generic.GenericForeignKey('content_type', 'object_id')
```

Что такое generic relations в Django.

```
>>> from django.contrib.auth.models import User
>>> guido = User.objects.get(username='Guido')
>>> t = TaggedItem(content_object=guido, tag='bdfl')
>>> t.save()
>>> t.content_object
<User: Guido>
```