Django运行方式及处理流程总结

Django项目的运行方式和对Request的基本处理流程。

一、Django的运行方式

运行Django项目的方法很多，这里主要介绍一下常用的方法。一种是在开发和调试中经常用到runserver方法，使用Django自己的web server；另外一种就是使用fastcgi，uWSGIt等协议运行Django项目，这里以uWSGIt为例。

1、runserver方法

runserver方法是调试Django时经常用到的运行方式，它使用Django自带的WSGI Server运行，主要在测试和开发中使用，使用方法如下：

```
Usage: manage.py runserver [options] [optional port number, or ipaddr:port]
# python manager.py runserver    # default port is 8000
# python manager.py runserver 8080
```

看一下manager.py的源码，你会发现上面的命令其实是通过Django的execute_from_command_line方法执行了内部实现的runserver命令，那么现在看一下runserver具体做了什么。

看了源码之后，可以发现runserver命令主要做了两件事情：

1). 解析参数，并通过django.core.servers.basehttp.get_internal_wsgi_application方法获取wsgi handler;

2). 根据ip_address和port生成一个WSGIServer对象，接受用户请求

get_internal_wsgi_application的源码如下：

```
def get_internal_wsgi_application():
    """
    Loads and returns the WSGI application as configured by the user in
    ``settings.WSGI_APPLICATION``. With the default ``startproject`` layout,
    this will be the ``application`` object in ``projectname/wsgi.py``.

    This function, and the ``WSGI_APPLICATION`` setting itself, are only useful
    for Django's internal servers (runserver, runfcgi); external WSGI servers
    should just be configured to point to the correct application object
    directly.

    If settings.WSGI_APPLICATION is not set (is ``None``), we just return
    whatever ``django.core.wsgi.get_wsgi_application`` returns.

    """
    from django.conf import settings
    app_path = getattr(settings, 'WSGI_APPLICATION')
    if app_path is None:
        return get_wsgi_application()

    return import_by_path(
        app_path,
        error_prefix="WSGI application '%s' could not be loaded; " % app_path
```

通过上面的代码我们可以知道，Django会先根据settings中的**WSGI_APPLICATION**来获取handler；在创建project的时候，Django会默认创建一个wsgi.py文件，而settings中的WSGI_APPLICATION配置也会默认指向这个文件。看一下这个wsgi.py文件，其实它也和上面的逻辑一样，最终调用**get_wsgi_application**实现。

2、uWSGI方法

uWSGI+Nginx的方法是现在最常见的在生产环境中运行Django的方法，本人的博客也是使用这种方法运行，要了解这种方法，首先要了解一下WSGI和uWSGI协议。

WSGI，全称Web Server Gateway Interface，或者Python Web Server Gateway Interface，是为Python语言定义的Web服务器和Web应用程序或框架之间的一种简单而通用的接口，基于现存的CGI标准而设计的。WSGI其实就是一个网关(Gateway)，其作用就是在协议之间进行转换。(PS: 这里只对WSGI做简单介绍，想要了解更多的内容可自行搜索)

uWSGI是一个Web服务器，它实现了WSGI协议、uwsgi、http等协议。注意uwsgi是一种通信协议，而uWSGI是实现uwsgi协议和WSGI协议的Web服务器。uWSGI具有超快的性能、低内存占用和多app管理等优点。以我的博客为例，uWSGI的xml配置如下：

```
<uwsgi>
    <!-- 端口 -->
    <socket>:7600</socket>
    <stats>:40000</stats>
    <!-- 系统环境变量 -->
    <env>DJANGO_SETTINGS_MODULE=geek_blog.settings</env>
    <!-- 指定的python WSGI模块 -->
    <module>django.core.handlers.wsgi:WSGIHandler()</module>
    <processes>6</processes>
    <master />
    <master-as-root />
    <!-- 超时设置 -->
    <harakiri>60</harakiri>
    <harakiri-verbose/>
    <daemonize>/var/app/log/blog/uwsgi.log</daemonize>
    <!-- socket的监听队列大小 -->
    <listen>32768</listen>
    <!-- 内部超时时间 -->
    <socket-timeout>60</socket-timeout>
```

以上就是uWSGI xml配置的写法，也可以使用ini的方式。安装uWSGI和运行的命令如下：

```
sudo pip install uwsgi
```

uWSGI和Nginx一起使用的配置方法就不在这里说明了，网上教程很多，需要的可以自行搜索。

二、HTTP请求处理流程

Django和其他Web框架一样，HTTP的处理流程基本类似：接受request，返回response内容。Django的具体处理流程大致如下图所示：

1、加载project settings

在通过django-admin.py创建project的时候，Django会自动生成默认的settings文件和manager.py等文件，在创建WSGIServer之前会执行下面的引用：

上面引用在执行时，会读取os.environ中的DJANGO_SETTINGS_MODULE配置，加载项目配置文

件，生成settings对象。所以，在manager.py文件中你可以看到，在获取WSGIServer之前，会先将project的settings路径加到os路径中。

2、创建WSGIServer

不管是使用runserver还是uWSGI运行Django项目，在启动时都会调用django.core.servers.basehttp中的run()方法，创建一个django.core.servers.basehttp.WSGIServer类的实例，之后调用其serve_forever()方法启动HTTP服务。run方法的源码如下：

```
def run(addr, port, wsgi_handler, ipv6=False, threading=False):
    server_address = (addr, port)
    if threading:
        httpd_cls = type(str('WSGIServer'), (socketserver.ThreadingMixIn, WSGISe
    else:
        httpd_cls = WSGIServer
    httpd = httpd_cls(server_address, WSGIRequestHandler, ipv6=ipv6)
    # Sets the callable application as the WSGI application that will receive re
    httpd.set_app(wsgi_handler)
```

如上，我们可以看到：在创建WSGIServer实例的时候会指定HTTP请求的Handler，上述代码使用WSGIRequestHandler。当用户的HTTP请求到达服务器时，WSGIServer会创建WSGIRequestHandler实例，使用其handler方法来处理HTTP请求(其实最终是调用wsgiref.handlers.BaseHandler中的run方法处理)。WSGIServer通过set_app方法设置一个可调用(callable)的对象作为application，上面提到的handler方法最终会调用设置的application处理request，并返回response。

其中，WSGIServer继承自wsgiref.simple_server.WSGIServer，而WSGIRequestHandler继承自wsgiref.simple_server.WSGIRequestHandler，wsgiref是Python标准库给出的WSGI的参考实现。其源码可自行到wsgiref参看，这里不再细说。

3、处理Request

第二步中说到的application，在Django中一般是django.core.handlers.wsgi.WSGIHandler对象，WSGIHandler继承自django.core.handlers.base.BaseHandler，这个是Django处理request的核心逻辑，它会创建一个WSGIRequest实例，而WSGIRequest是从http.HttpRequest继承而来。

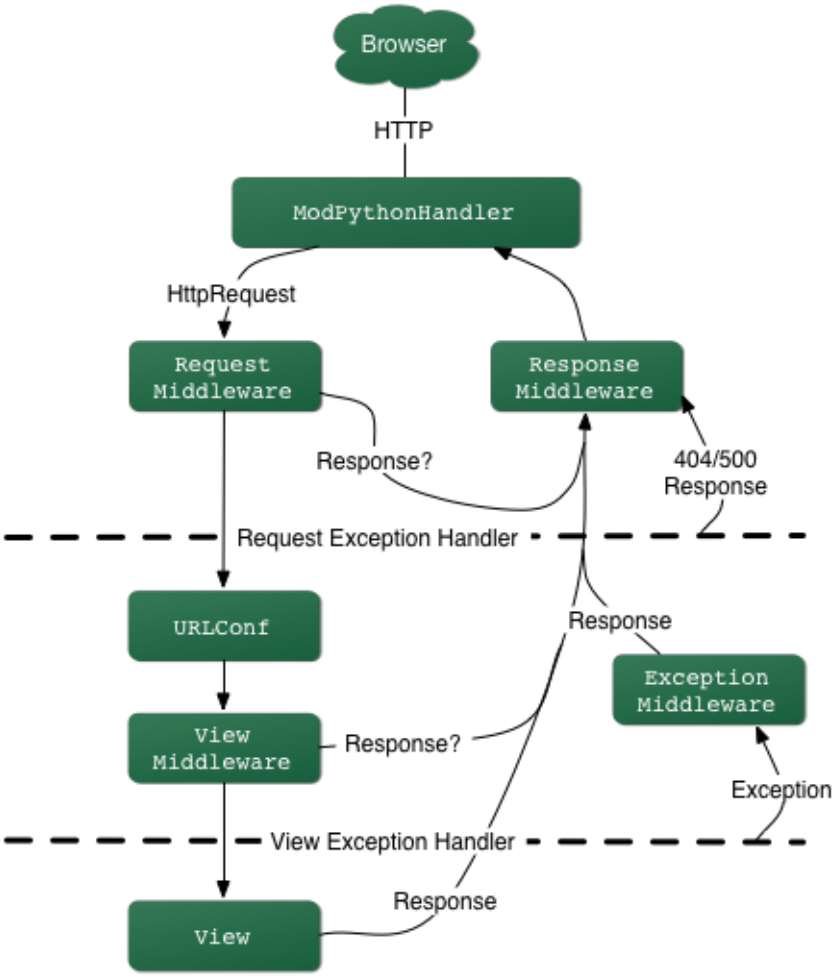
4、返回Response

上面提到的BaseHandler中有个get_response方法，该方法会先加载Django项目的ROOT_URLCONF，然后根据url规则找到对应的view方法(类)，view逻辑会根据request实例生成并返回具体的response。

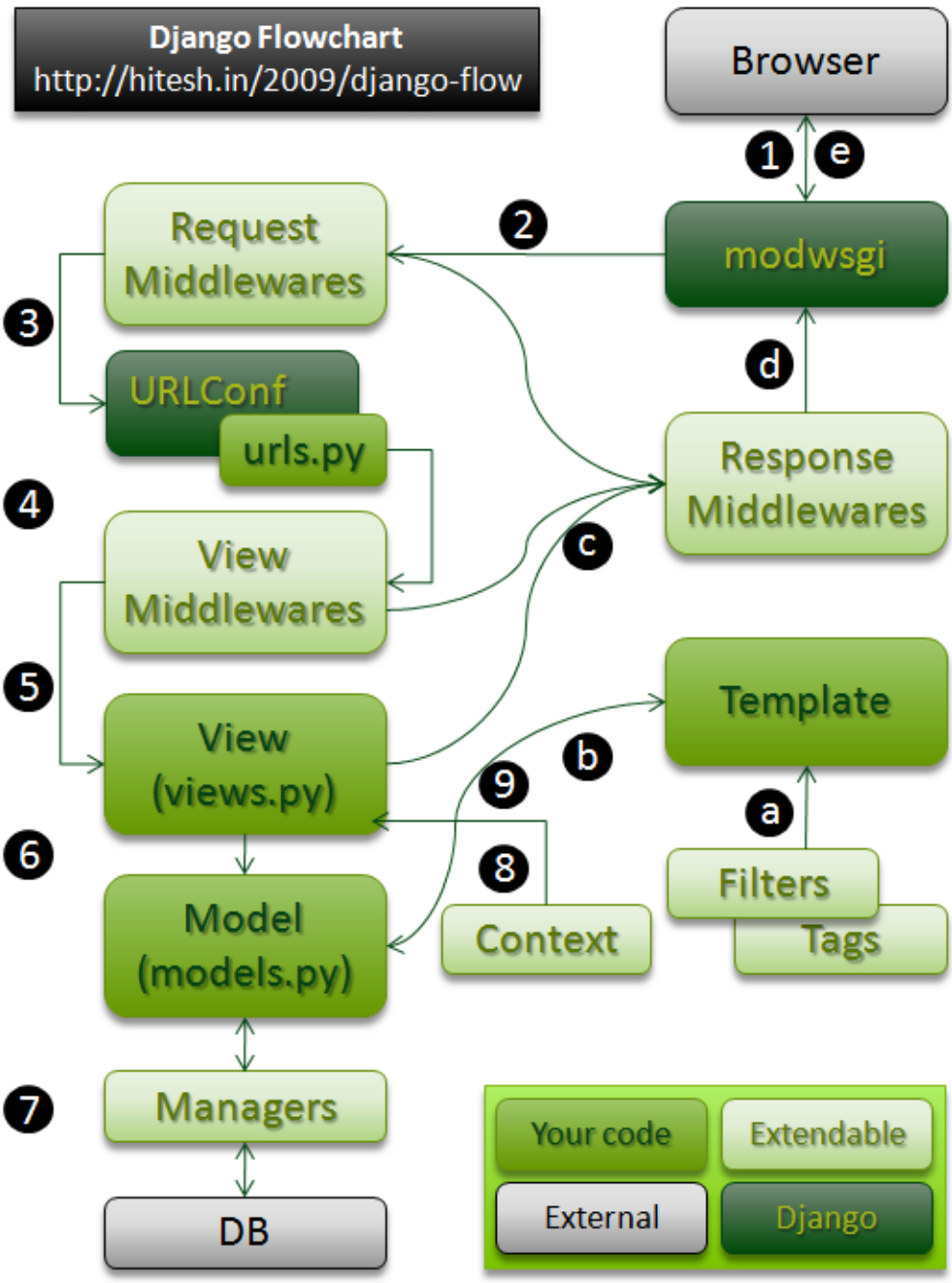
在Django返回结果之后，第二步中提到wsgiref.handlers.BaseHandler.run方法会调用finish_response结束请求，并将内容返回给用户。

三、Django处理Request的详细流程

上述的第三步和第四步逻辑只是大致说了一下处理过程，Django在处理request的时候其实做了很多事情，下面我们详细的过一下。首先给大家分享两个网上看到的Django流程图：


Browser
HTTP
ModPythonHandler
HttpRequest
Request Middleware
Response Middleware
Response?
404/500 Response
Request Exception Handler
URLConf
Response
Exception Middleware
View Middleware
Response?
Exception
View Exception Handler
Response
View

Django流程图1



Django流程图2

上面的两张流程图可以大致描述Django处理request的流程，按照流程图2的标注，可以分为以下几个步骤：

1. 用户通过浏览器请求一个页面

2. 请求到达Request Middlewares，中间件对request做一些预处理或者直接response请求

3. URLConf通过urls.py文件和请求的URL找到相应的View

4. View Middlewares被访问，它同样可以对request做一些处理或者直接返回response

5. 调用View中的函数

6. View中的方法可以选择性的通过Models访问底层的数据

7. 所有的Model-to-DB的交互都是通过manager完成的

8. 如果需要，Views可以使用一个特殊的Context

9. Context被传给Template用来生成页面

a. Template使用Filters和Tags去渲染输出

b. 输出被返回到View

c. HTTPResponse被发送到Response Middlewares

d. 任何Response Middlewares都可以丰富response或者返回一个完全不同的response

e. Response返回到浏览器，呈现给用户

上述流程中最主要的几个部分分别是：Middleware(中间件，包括request, view, exception, response)，URLConf(url映射关系)，Template(模板系统)，下面一一介绍一下。

1、Middleware(中间件)

Middleware并不是Django所独有的东西，在其他的Web框架中也有这种概念。在Django中，Middleware可以渗入处理流程的四个阶段：request，view，response和exception，相应的，在每个Middleware类中都有rocess_request，process_view， process_response 和process_exception这四个方法。你可以定义其中任意一个活多个方法，这取决于你希望该Middleware作用于哪个处理阶段。每个方法都可以直接返回response对象。

Middleware是在Django BaseHandler的**load_middleware**方法执行时加载的，加载之后会建立四个列表作为处理器的实例变量：

_request_middleware：process_request方法的列表

_view_middleware：process_view方法的列表

_response_middleware：process_response方法的列表

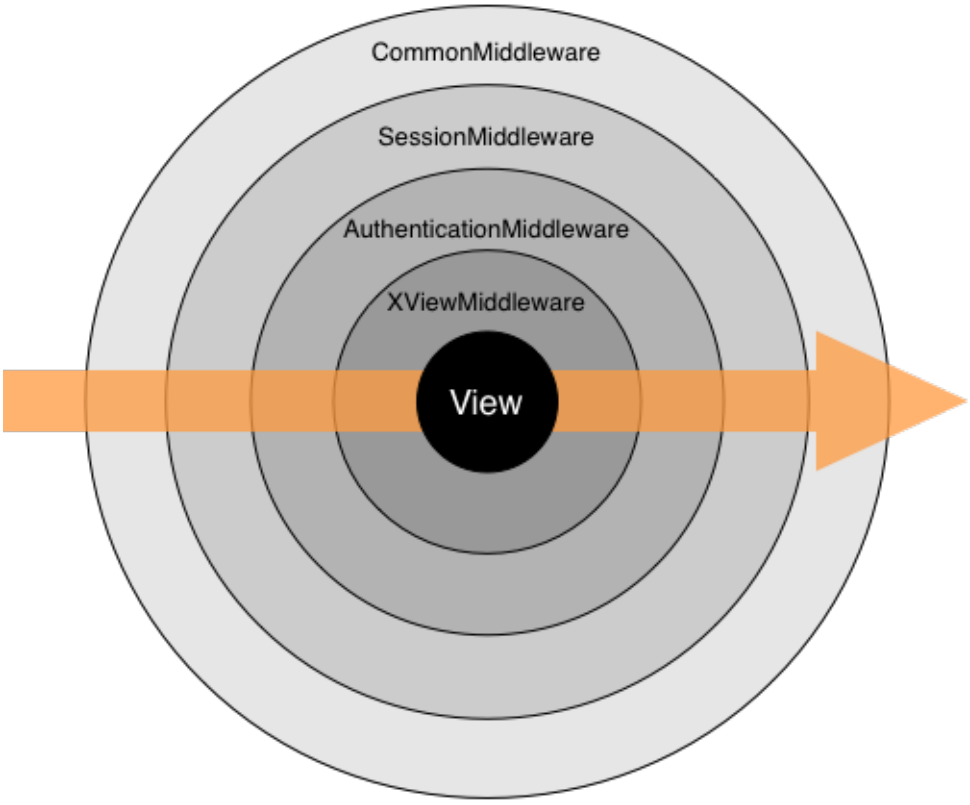
_exception_middleware：process_exception方法的列表

Django的中间件是在其配置文件(settings.py)的MIDDLEWARE_CLASSES元组中定义的。在MIDDLEWARE_CLASSES中，中间件组件用字符串表示：指向中间件类名的完整Python路径。例如GeekBlog项目的配置：

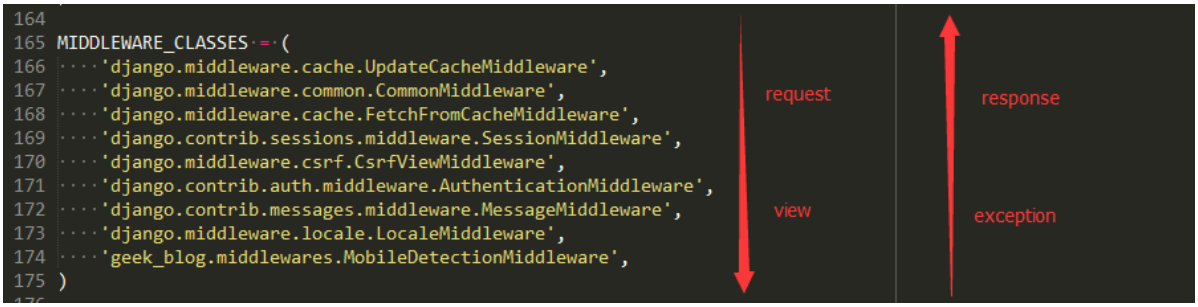
```
1  MIDDLEWARE_CLASSES = (
2      'django.middleware.cache.UpdateCacheMiddleware',
3      'django.middleware.common.CommonMiddleware',
4      'django.middleware.cache.FetchFromCacheMiddleware',
5      'django.contrib.sessions.middleware.SessionMiddleware',
6      'django.middleware.csrf.CsrfViewMiddleware',
```

```
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    'geek_blog.middlewares.MobileDetectionMiddleware',    # 自定义的Middleware
```

Django项目的安装并不强制要求任何中间件，如果你愿意，MIDDLEWARE_CLASSES可以为空。中间件出现的顺序非常重要：在request和view的处理阶段，Django按照MIDDLEWARE_CLASSES中出现的顺序来应用中间件，而在response和exception异常处理阶段，Django则按逆序来调用它们。也就是说，Django将MIDDLEWARE_CLASSES视为view函数外层的顺序包装子：在request阶段按顺序从上到下穿过，而在response则反过来。以下两张图可以更好地帮助你理解：



Django Middleware流程1



Django Middleware流程2

2、URLConf(URL映射)

如果处理request的中间件都没有直接返回response，那么Django会去解析用户请求的URL。URLconf就是Django所支撑网站的目录。它的本质是URL模式以及要为该URL模式调用的视图函数之间的映射表。通过这种方式可以告诉Django，对于这个URL调用这段代码，对于那个URL调用那段代码。具体的，在Django项目的配置文件中有ROOT_URLCONF常量，这个常量加上根目录”/“ ，作

为参数来创建django.core.urlresolvers.RegexURLResolver的实例，然后通过它的resolve方法解析用户请求的URL，找到**第一个**匹配的view。

其他有关URLConf的内容，这里不再具体介绍，大家可以看DjangoBook了解。

3、Template(模板)

大部分web框架都有自己的Template(模板)系统，Django也是。但是，Django模板不同于Mako模板和jinja2模板，在Django模板不能直接写Python代码，只能通过额外的定义filter和template tag实现。由于本文主要介绍Django流程，模板内容就不过多介绍。

参考文章：

uWSGI Web服务器介绍

wsgiref源码分析

用Python写一个简单的Web框架

Django 结构及处理流程分析

PS: 以上代码和内容都是基于**Django 1.6.5**版本，其他版本可能与其不同，请参考阅读。