一代版本一代神：利用Docker在Win10系统极速体验Django3.1真实异步(Async)任务

原创

v3u.cn 于 2020-09-30 20:04:31 发布 356 收藏 2

文章标签： django python 异步 async django3.1

刘悦的技术博客 v3u.cn

本文链接：https://blog.csdn.net/zcxey2911/article/details/108889754

一代版本一代神：利用Docker在Win10系统极速体验Django3.1真实异步(Async)任务

就在去年(2019年)，Django官方发布3.0版本，内核升级宣布支持Asgi，这一重磅消息让无数后台研发人员欢呼雀跃，弹冠相庆。大喜过望之下，小伙伴们兴奋的开箱试用，结果却让人大跌眼镜：非但说好的内部集成Websocket没有出现，就连原生的异步通信功能也只是个壳子，内部并未实现，很明显的换汤不换药，这让不少人转身投入了FastAPI的怀抱。不过一年之后，今天8月，Django3.1版本姗姗来迟，这个新版本终于一代封神，不仅支持原生的异步视图，同时也支持异步中间件，明显整了个大活。

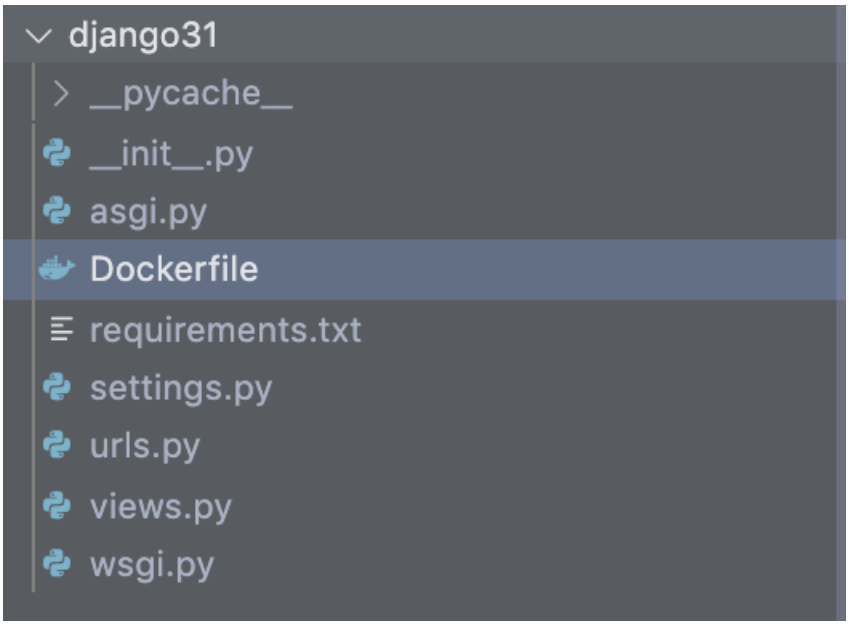
本次我们利用Docker制作一款基于Django3.1.1的项目镜像，实际体验一下Django原生异步的魅力。

首先在宿主机安装新版Django

```
pip install Django==3.1.1
```

新建一个项目，名字为django31

```
django-admin.py startproject django31 .
```



进入项目目录可以发现，熟悉的入口文件mange.py已经消失不见，新增了asgi.py文件用来启动项目，这里我们使用异步服务器uvicorn来启动新版Django，而uvicorn对windows系统支持不够友好，所以使用Docker来构建一个运行镜像，简单方便，进入django31目录，新建Dockerfile:

```
FROM python:3.7
WORKDIR /Project/django31

COPY requirements.txt ./
RUN pip install -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/simple

COPY . .
ENV LANG C.UTF-8
WORKDIR /Project
CMD ["uvicorn", "django31.asgi:application","--host","0.0.0.0"]
```

这里需要注意一点，Docker每创建一个容器，会在iptables中添加一个规则，每个容器都会在本机127.17.X.X范围内分配一个地址，容器绑定的主机端口会映射到本机的127.17.X.X的容器抛出端口上。所以容器内部的项目绑定的ip不能是127.0.0.1，要绑定为0.0.0.0，这样绑定后容器内部app的实际ip由Docker自动分配，所以这里uvicorn启动参数需要用host强制绑定为0.0.0.0。

随后在项目中创建依赖文件requirements.txt:

```
django==3.1.1
uvicorn
httpx
```

开始编译镜像文件：

```
docker build -t 'django31' .
```

编译成功后大概1g左右

```
liuyue:django31 liuyue$ docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
django31                    latest              e8afbbbb9305        30 minutes ago      919MB
```

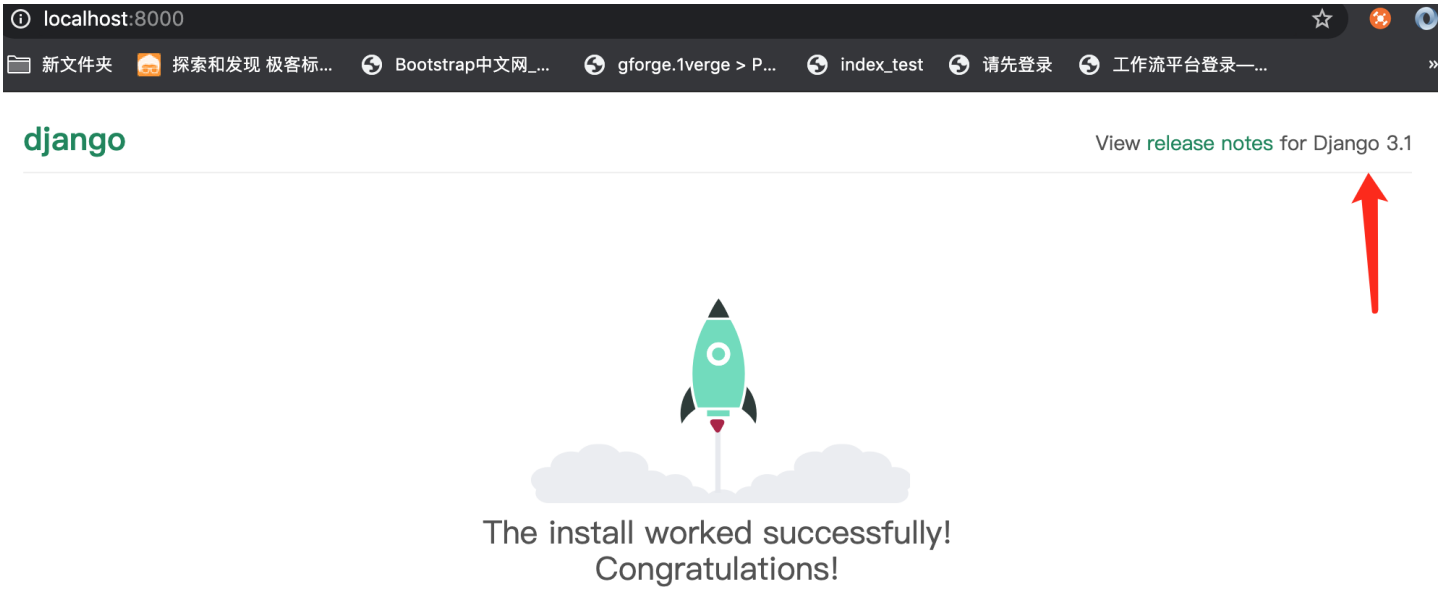
然后我们来启动项目：

```
docker run -it --rm -p 8000:8000 django31
```

后台显示启动顺利，绑定在容器内的0.0.0.0：

```
liuyue:django31 liuyue$ docker run -it --rm -p 8000:8000 django31
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     ASGI 'lifespan' protocol appears unsupported.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

浏览器访问：http://localhost:8000



熟悉的小火箭又起飞了，接下来我们来编写第一个异步视图views.py

```
from django.http import HttpResponse
async def index(request):
    return HttpResponse("异步视图")
```

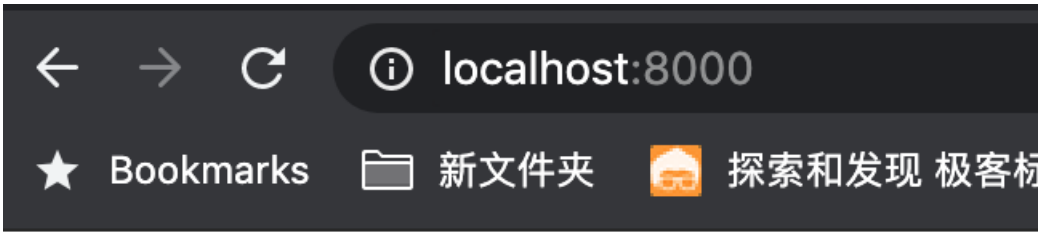
修改一下路由文件urls.py

```
from django.contrib import admin
from django.urls import path
from django31.views import index

urlpatterns = [
    path('admin/', admin.site.urls),
    path("", index)
]
```

重新编译镜像：

```
docker build -t 'django31' .
docker run -it --rm -p 8000:8000 django31
```

访问http://localhost:8000



异步视图

没有问题，还记得去年我们曾经使用Siege对Django2.0版本进行压力测试吗？现在我们再来测一下

```
siege -c150 -t60S -v -b 127.0.0.1:8000
```

150个并发持续一分钟，看看新版Django的抗压能力怎么样：

```
liuyue:~ liuyue$ siege -c150 -t60S -v -b 127.0.0.1:8000

{ "transactions":          10517,
 "availability":         100.00,
 "elapsed_time":          59.70,
 "data_transferred":          0.12,
 "response_time":          0.84,
 "transaction_rate":        176.16,
 "throughput":            0.00,
 "concurrency":         148.58,
 "successful_transactions":        10517,
 "failed_transactions":             0,
 "longest_transaction":          1.13,
 "shortest_transaction":          0.45
}
liuyue:~ liuyue$
```

从测试结果看，整体性能虽然没有质的提高，但是也还算是差强人意，乞丐级主机在uvicorn的加持下单机200个左右并发还是能抗住的。

接下来我们来体验一下真正的技术，Django内置的原生异步任务，分别同步和异步两种方式使用httpx来请求接口，方法中人为的阻塞10秒钟：

```
from django.http import HttpResponse

import asyncio
from time import sleep
import httpx

#异步请求
async def http_call_async():
    for num in range(10):
        await asyncio.sleep(1)
        print(num)
    async with httpx.AsyncClient() as client:
        r = await client.get("https://v3u.cn")
        print(r)

#同步请求
def http_call_sync():
    for num in range(10):
        sleep(1)
        print(num)
    r = httpx.get("https://v3u.cn")
    print(r)
```

再分别通过同步和异步视图进行调用：

```
async def async_view(request):
    loop = asyncio.get_event_loop()
    loop.create_task(http_call_async())
    return HttpResponse("非阻塞视图")


def sync_view(request):
    http_call_sync()
    return HttpResponse("阻塞视图")
```

修改路由：

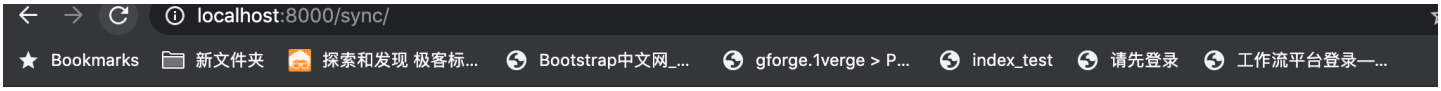
```
from django.contrib import admin
from django.urls import path
from django31.views import index, async_view, sync_view

urlpatterns = [
    path('admin/', admin.site.urls),
    path("", index),
    path("async/", async_view),
    path("sync/", sync_view),
]
```

重新编译：

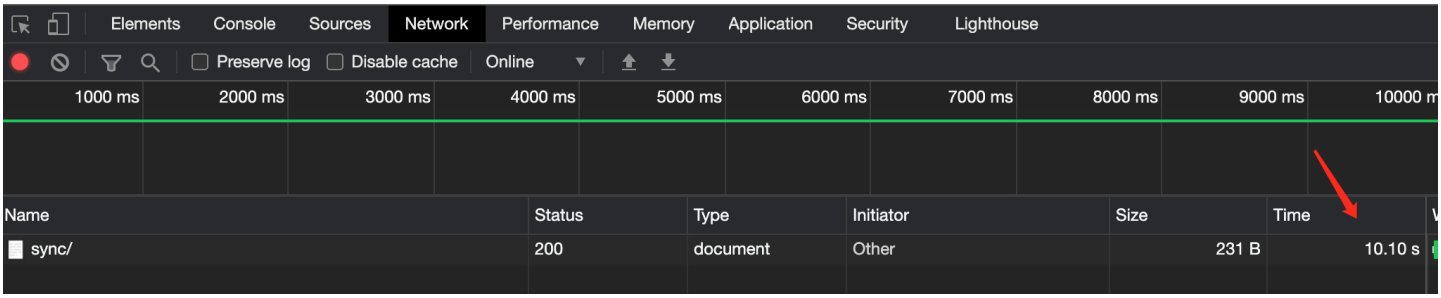
```
docker build -t 'django31' .
docker run -it --rm -p 8000:8000 django31
```

访问 http://localhost:8000/sync/ 看看同步的效率

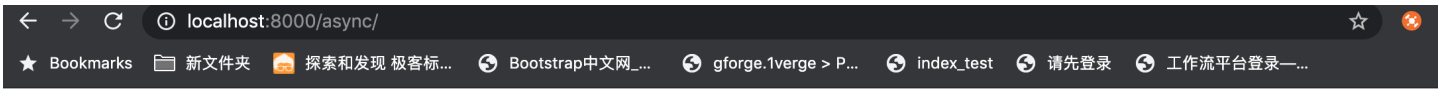
很明显过程中阻塞了10秒，然后我们才等到页面结果：



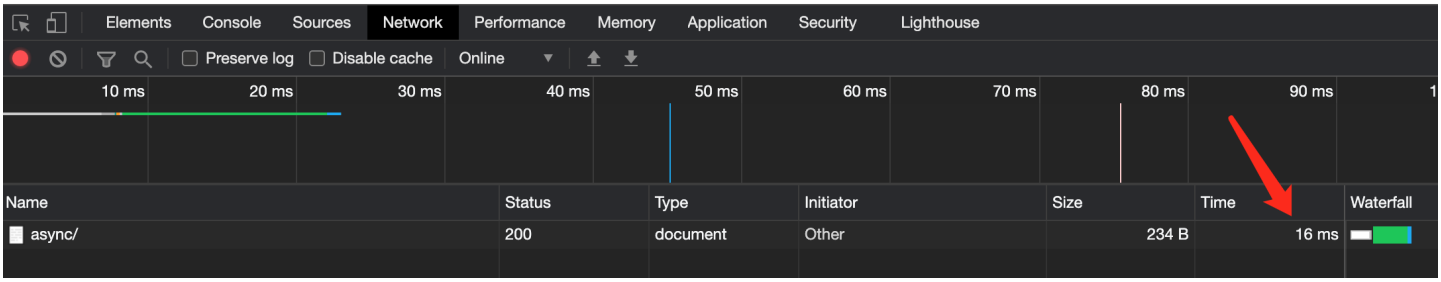
阻塞视图



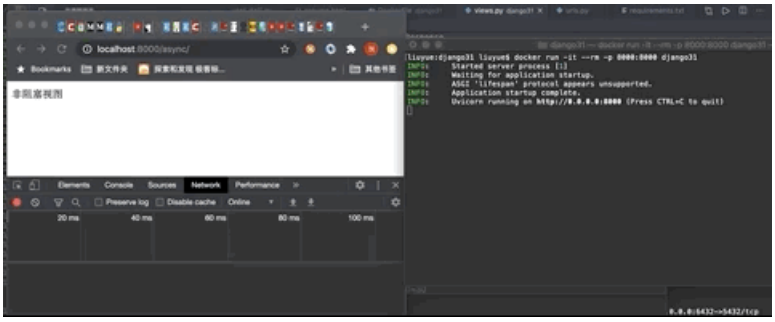
再来试试不一样的，访问http://localhost:8000/async/



非阻塞视图



16毫秒，无视阻塞，瞬间响应。



通过动图我们可以发现，后端还在执行阻塞任务，但是前段已经通过异步多路复用将请求任务结果返回至浏览器了。

虽然这已经很不错了，但是稍有遗憾的是，目前Django内置的ORM还是同步机制，也就是说当我们读写数据库的时候还是阻塞状态，此时的场景就是异步视图内塞入了同步操作，这该怎么办呢？可以使用内置的sync_to_async方法进行转化：

```
from asgiref.sync import sync_to_async
async def async_with_sync_view(request):
    loop = asyncio.get_event_loop()
    async_function = sync_to_async(http_call_sync)
    loop.create_task(async_function())
    return HttpResponse("(via sync_to_async)")
```

由此可见，Django3.1在异步层面真的开始秀操作了，这就带来另外一个问题，既然原生异步任务已经做得这么牛逼了，我们到底还有没有必要使用Celery?

其实关于Django的异步视图只是提供了类似于任务或消息队列的功能，但功能上并没有Celery强大。如果你正在使用(或者正在考虑)Django3.1，并且想做一些简单的事情(并且不关心可靠性)，异步视图是一种快速、简单地完成这个任务的好方法。如果你需要执行重得多的、长期运行的后台进程，你还是要使用Celery。

简而言之，Django3.1的异步任务目前仅仅是解决Celery过重的一个简化方案而已。

结语：假如我们说，新世纪以来在Python在Web开发界有什么成就，无疑的，我们应该说，Django和Flask是两个颠扑不破的巨石重镇，没有了它们，Python的web开发史上便要黯然失光，Django作为第一web开发框架，要文档有文档，要功能有功能，腰斩对手于马下，敏捷开发利器。Django3.1的发布仿佛把我们又拉回到了Django一统江湖的年代，那个美好的时代，让无数人午夜梦回。

原文转载自「刘悦的技术博客」 https://v3u.cn/a_id_177