Beijing Forest Studio
北京理工大学信息系统及安全对抗实验中心

Python Web编程-Django

崔成钢 硕士研究生

2021年01月24日

内容提要

- 基本概念
- Django 基础：MVT开发模式和Django admin
- Django 进阶：Django异步
- Django 总结

预期收获

- 了解Web基本编程结构
- 了解Django MVT开发模式
- 理解Django 异步操作
- 了解Django 的优缺点

Python Web

- Python Web：
  - 《使用python进行并发编程》
  - 后端服务器框架

高并发编程

	Tornado	Django
优点	1）轻量级web框架，功能少而精，注重性能优越 2）HTTP服务器 3）异步编程 4）WebSocket长连接 5）解决高并发	1）重量级web框架，功能大而全，注重高效开发,自带orm，template，view 2）内置管理后台 3）内置封装完善的ORM操作 4）session功能 5）后台管理
缺点	入门门槛较高	高耦合

名称	描述
Tornado	支持异步，有自己的服务器的Web框架，初学难度高
Webpy	轻量级Web框架，但可能已经停止更新
Flask	成熟的Web框架，生态齐备，使用率高但有一定学习成本
Japronto	2017年较新框架，性能很强，但生态还不齐备，当前版本还有诸多问题
Django	成熟的Web框架，生态齐备，学习成本低，易于快速上手

52

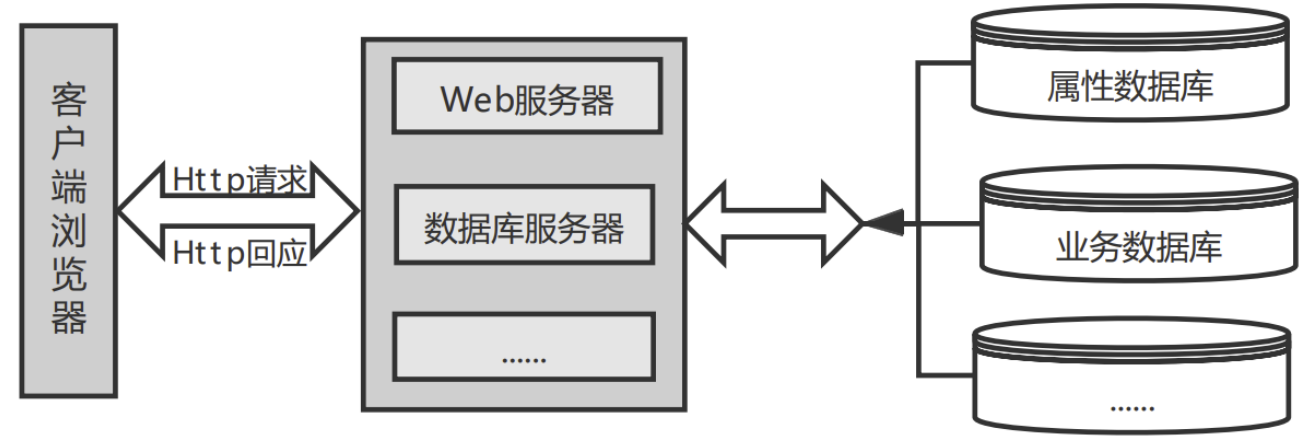
基本概念

基本概念

- **Web**
  - 定义： Web（World Wide Web）即全球广域网，也称为万维网，它是一种基于超文本和HTTP的全球性、动态交互的跨平台分布式图形信息系统。
  - 特点：
    - 图形化：将文字、图片等在一个结构下呈现出来；
    - 兼容：任何操作系统都可以通过浏览器预览使用；
    - 分布式：所有元素被安排在不同的磁盘中，防止因集中读取一台磁盘而出现瓶颈；
    - 动态：Web内信息动态可变，特别是对于大型更新速度快的网站；
    - 交互：通过Web内信息进行数据交互，例如私信交流。

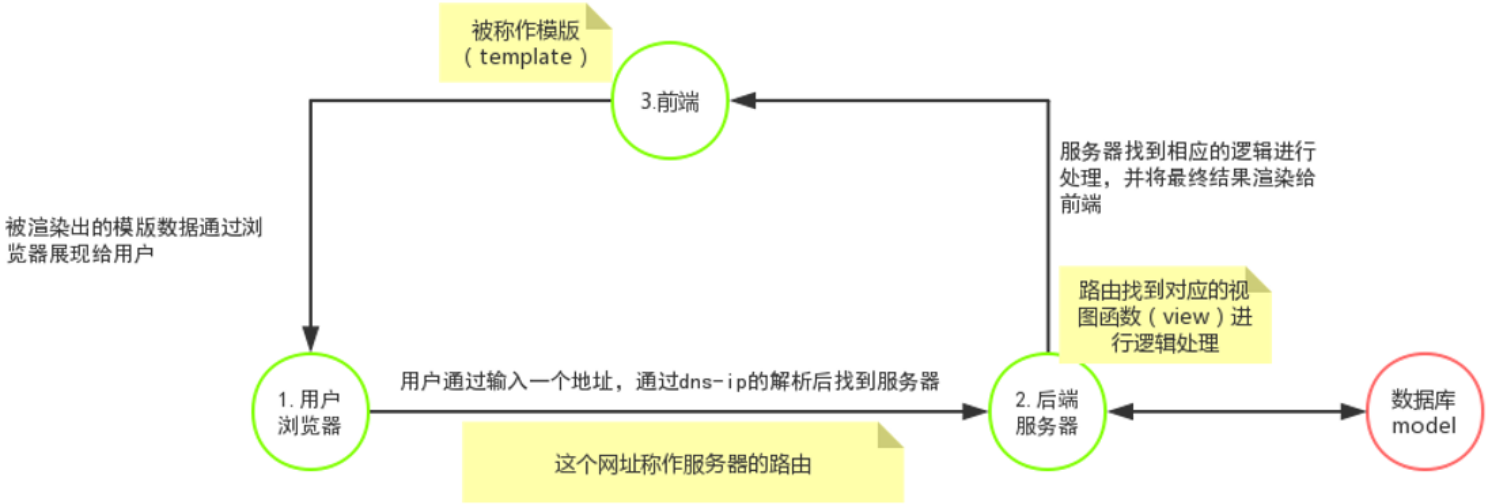
基本概念

- Web结构
  - 前端：用户肉眼看到的网站布局、内容以及一切可以直接接触与操作的部分。
    - 静态功能：不和后端服务器进行交互，仅在前端处理并响应用户；
    - 动态功能：前端发送给后端的指令，后端接到指令并处理后响应给前端。
  - 后端：业务逻辑，数据库IO，用户不可直接接触的部分。

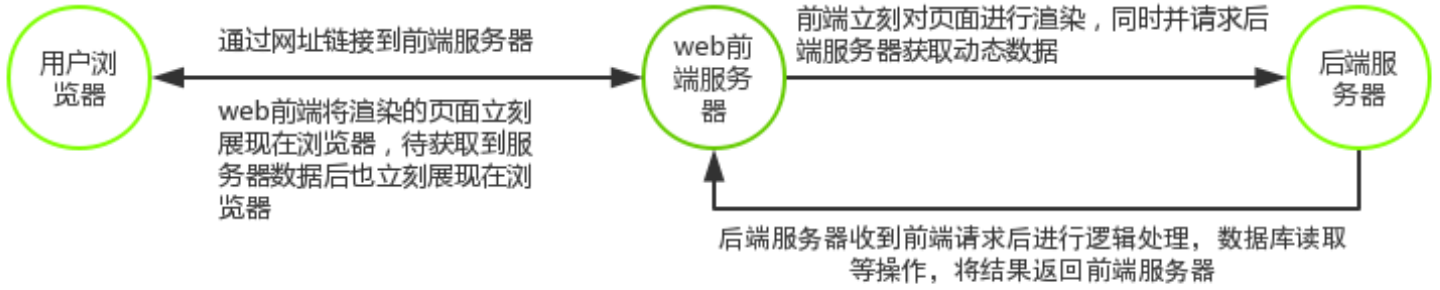


基本概念

- Web结构图
  - 前后端不分离：



  - 前后端分离：

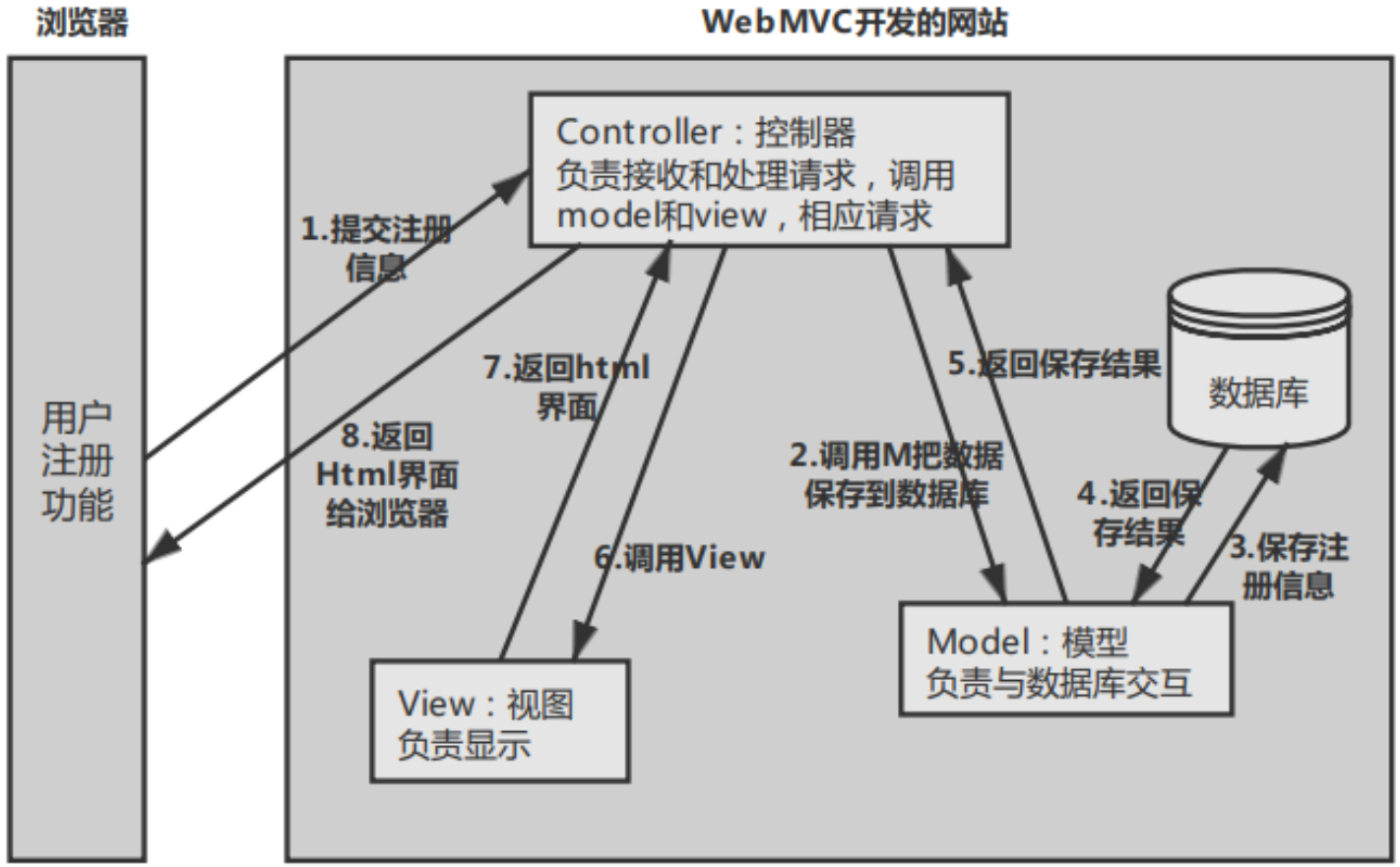


Django基础

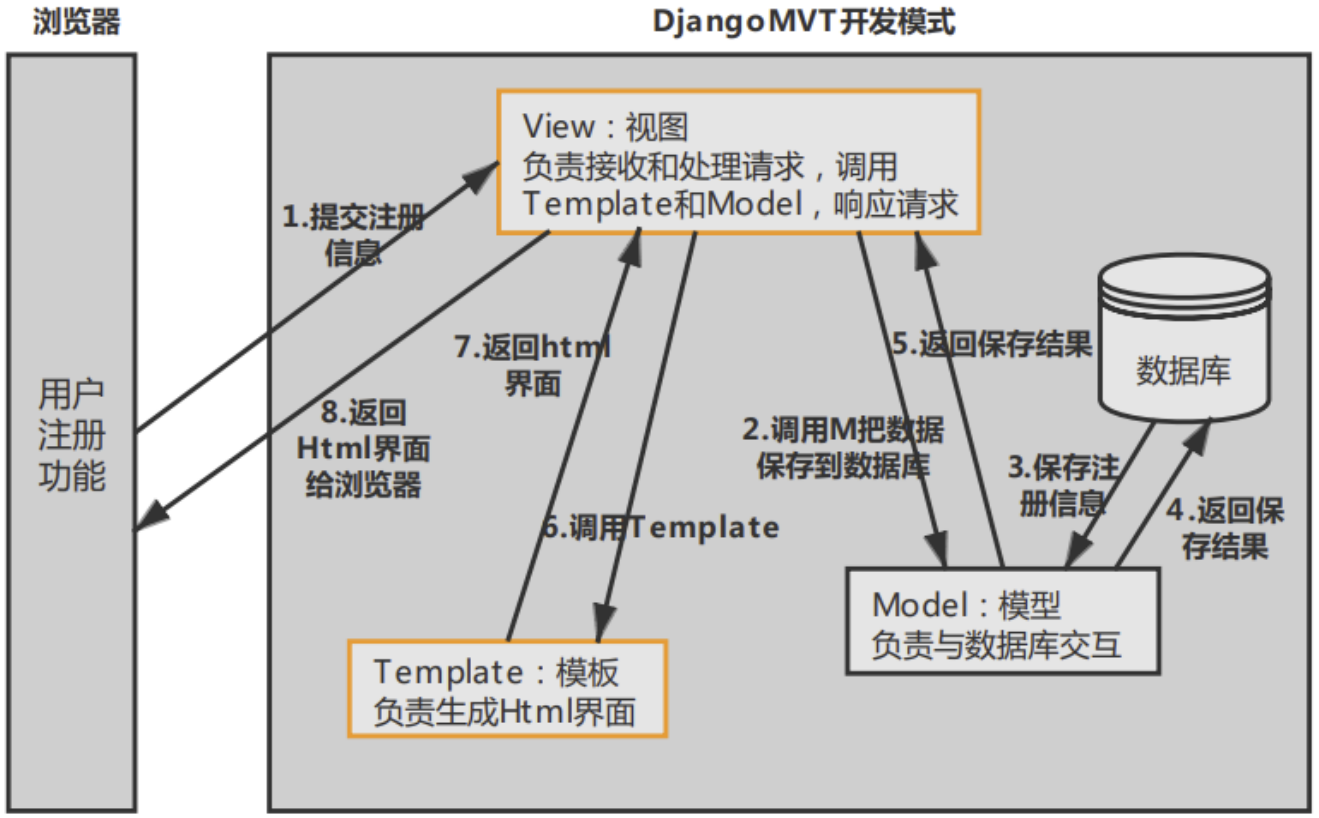
Django基础

- **Django基础介绍**
  - **Django是一个在2005年发布的开源Web应用框架，由Python编写；**
  - **拥有齐备的官方文档， 提供包括缓存、数据ORM、后台管理、等多项一站式功能。**
  - **官方文档地址：https://docs.djangoproject.com/**

- Web开发中MVC模式
  - MVC：Model-View-Controller 模型-视图-控制器；
  - M：数据处理；
  - V：界面显示；
  - C：逻辑处理。

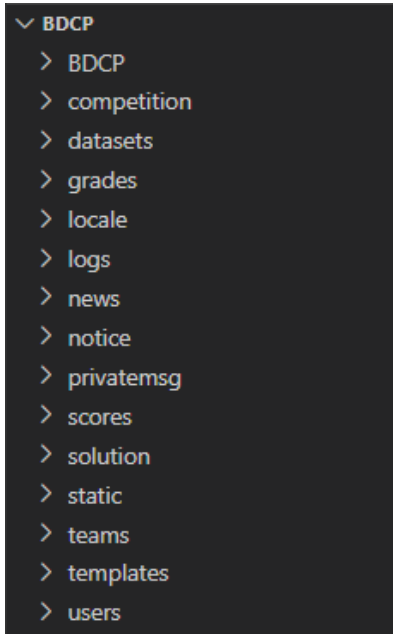


- Django中MVT模式
  - MTV：Model-View-Template；
  - Model：与MVC中的M功能相同，负责与数据库交互，处理数据，内嵌ORM框架；
  - View：与MVC中的C功能相同，负责接收HttpRequest，业务逻辑处理，返回HttpResponse；
  - Template：与MVC中的V功能相同，负责封装构造返回的html，内嵌了模板引擎。

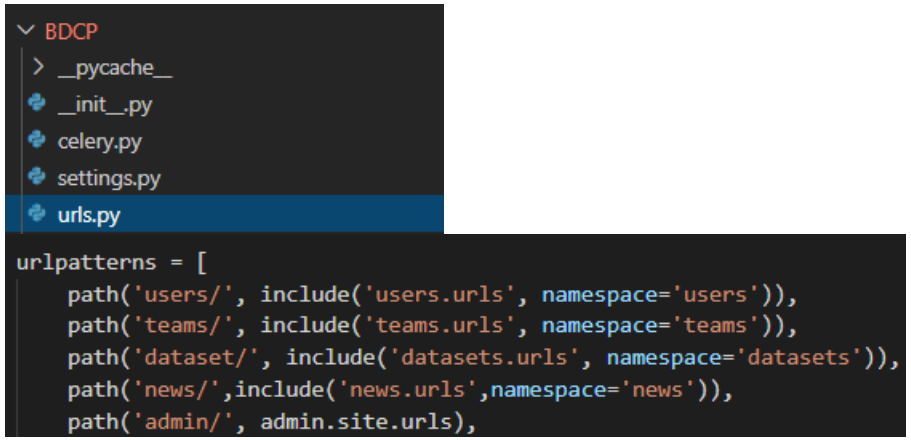


Django基础

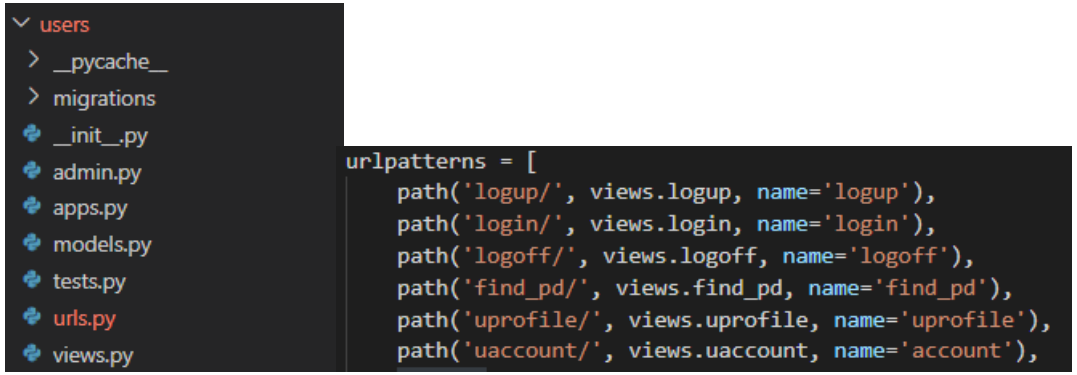
- 项目创建
  - 基础环境：python3（Virtualenv）
  - 项目创建：
    - django-admin startproject 项目名 创建一个django项目；
    - python manage.py startapp 应用名 项目中创建一个应用。
  - 项目运行：
    - python manage.py runserver 0.0.0.0:8000 启动开发服务器。



- View视图层
  - 功能：负责处理用户的请求和生成相应内容，然后传输给页面或其它文件中。
  - 路由：决定端到端路径的网络范围的进程。
    - 根路由：根目录项目中的urls.py文件，根路由可以集合所有应用路由；
    - 应用路由：每个应用下创建的urls.py，属于每个应用的独有路由，集成或绑定到根路由中进行使用。



根路由



应用路由

- View视图层
  - 用户的请求 request；
    - 浏览器向服务器发送的请求对象，包含用户信息，请求内容和请求方法；
  - 对用户请求的逻辑处理 handler；
  - 将处理后的数据返回 response。

```python
def login(request):
    if request.session.get('is_login', None):  # 已经登录
        return redirect('/index/')  # 使用相对URL，redire

    if request.method == 'POST':
        username = request.POST.get('username').strip()
        password = request.POST.get('password')  # 获取密
```

请求

```python
if email_check(username):  # 判断是否为邮箱
    if not User.objects.filter(email=username):
        infor = {'message': "邮箱不存在！", 'username': username}
        return render(request, 'users/login.html', infor)
    else:
        username = User.objects.get(email=username).username  # 邮
elif not username_check(username):  # 用户名检查
    infor = {'message': "用户名为2-10位中文字母或数字！"}
    return render(request, 'users/login.html', infor)

if not password_check(password):
    infor = {'message': "密码长度为6-20位!"}
    return render(request, 'users/login.html', infor)
```

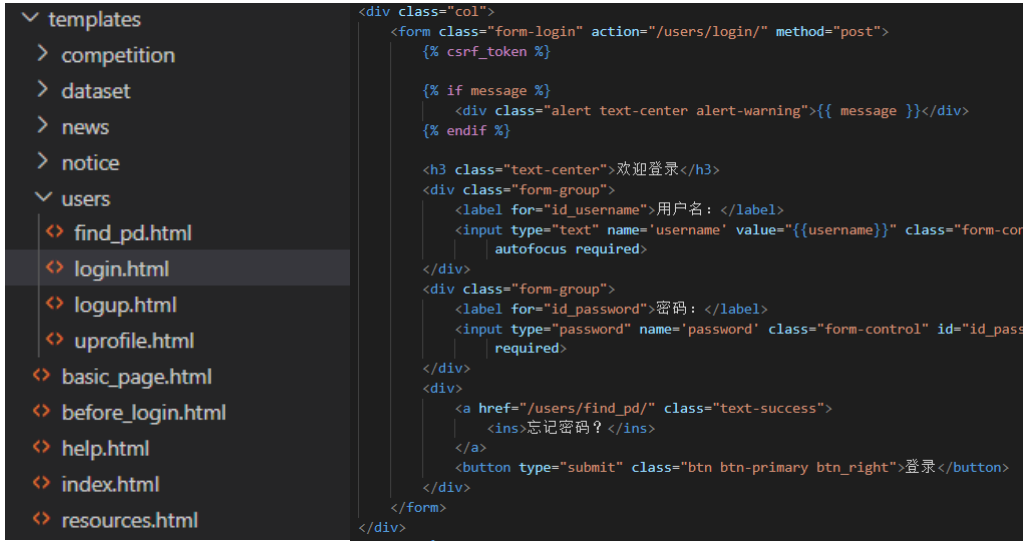
```python
return HttpResponse('该链接超出有效时间，请重新注册获取验证链接！')
```

处理与返回

- Template模板层
  - 功能：动态生成HTML网页，包括HTML代码和一些特殊的语法等。
  - 使用方法：
    - Settings里配置Template路径；
    - 创建目录，确定HTML、CSS等代码。

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'users.views.global_setting',
            ],
        },
    },
]
```

Template路径



目录及代码

- Model模型层
  - 功能：负责和数据库交互，处理数据。
  - ORM：object relational mapping（对象关系映射）
    - 作用：通过把表映射成类，把行映射成实例，把字段映射成属性，把对应的对象操作转换成数据库的原生语句来使用Python完成数据库的开发。
    - 优点：
      - 使用简单：将数据库语法封装，直接使用相应的方法即可操作；
      - 性能好：提升业务编程方便性的同时，将ORM转化为SQL性能消耗极低；
      - 兼容性好：适配多数关系型数据库，例如MySQL、SQLite等。

- Model模型层
  - 使用方法：
    - Settings.py设置创建好的数据库信息；
    - 在每个应用的models.py中以类的形式定义Model（主键、外键、表关系）；
    - 通过migrate和migration指令将Model中的信息迁移到数据库；
    - 在View中对Model操作实现对数据库操作。

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'BDCP',
        'USER': 'root',
        'PASSWORD': 'ccg',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

数据库配置

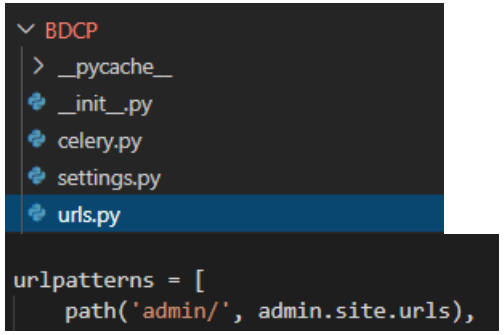
```
class UserProfile(models.Model):

    user = models.OneToOneField(User, on_delete=models.CASCADE, related_name='profile')
    course = models.ManyToManyField(Course, blank=True, verbose_name='选课信息')
    subject = models.ManyToManyField(Competition, blank=True, verbose_name='选题信息')
    name = models.CharField('姓名', max_length=128, blank=True)
    classnumber = models.CharField('班级', max_length=20, blank=True)
    schoolnumber = models.CharField('学号', max_length=20, blank=True)
    coursename = models.CharField('选课', max_length=20, blank=True)
    subjectname = models.CharField('选题', max_length=20, blank=True)
    mod_date = models.DateTimeField('最后修改日期', auto_now=True)
```

类定义Model

```
def get_profile(username):
    user = User.objects.get(username=username)
    courses = Course.objects.all()
    my_course = user.profile.course.all()
    if my_course:
        course_id = my_course[0].id
    else:
        course_id = '0'
```

实现对数据库操作

- Django admin后台管理
  - 特点：
    - Django自带，只需配置参数，无需手动编写界面；
    - 权限控制，只有管理员可以访问；
    - 系统数据库的可视化监控与操作。
  - 配置与访问：
    - 项目根路由下配置admin；
    - 在对应的根IP地址下添加/admin访问。

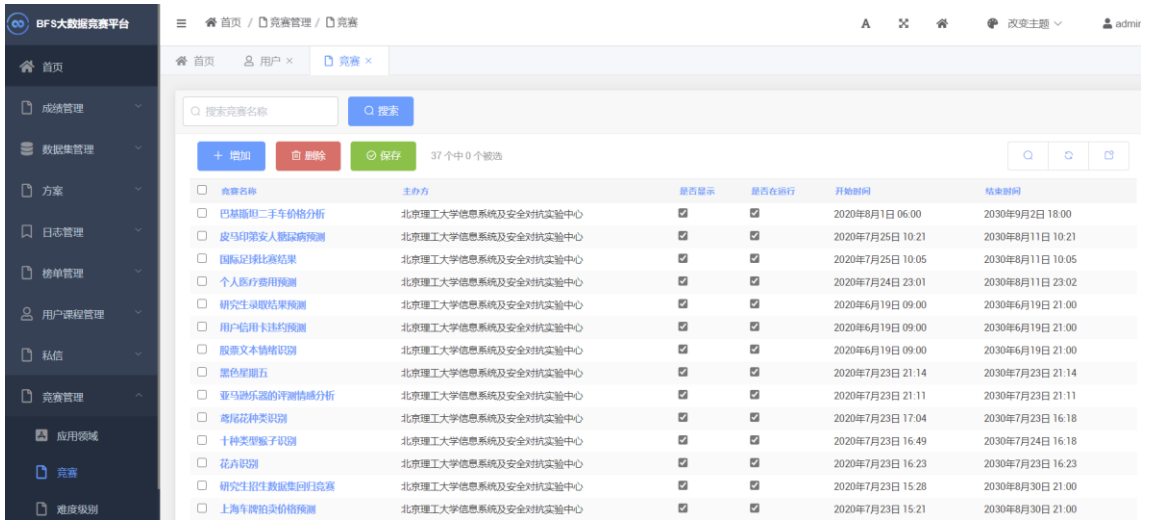


根路由

admin访问

- Django admin后台管理
  - 使用方法：
    - 在应用目录下创建admin.py文件；
    - 使用register注册对应Model中的类；
    - 选择类中监控和可编辑的对象。

admin注册

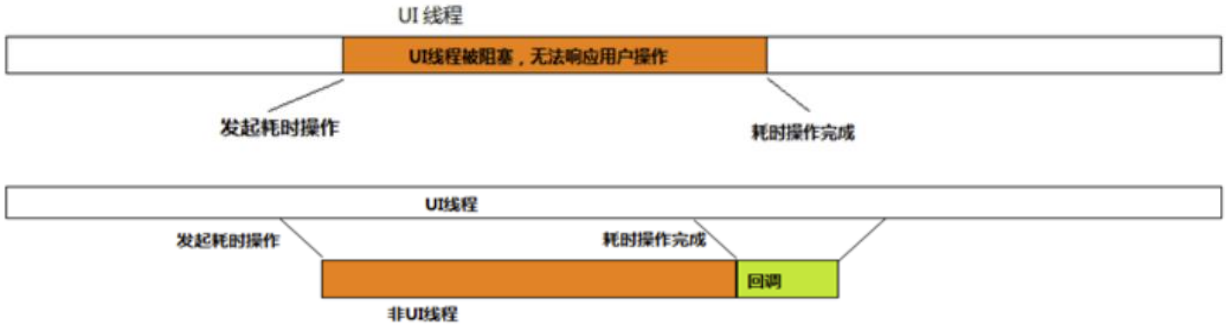


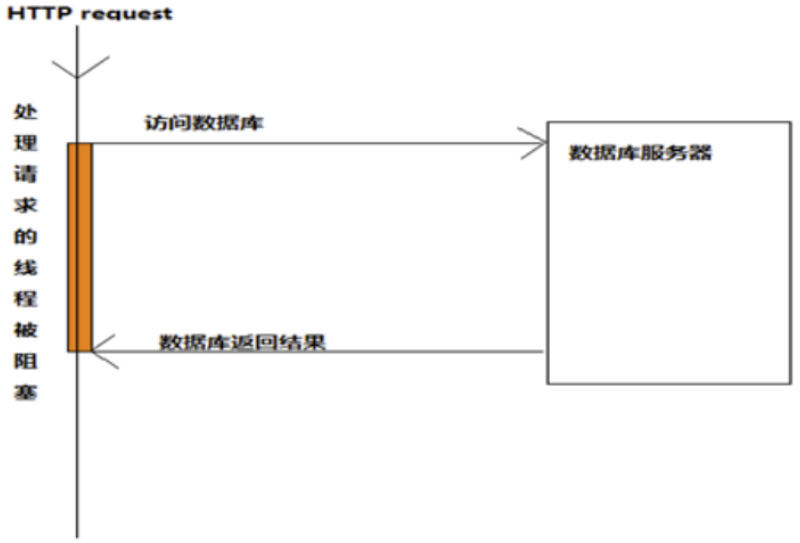
后台管理

Django 进阶

异步编程

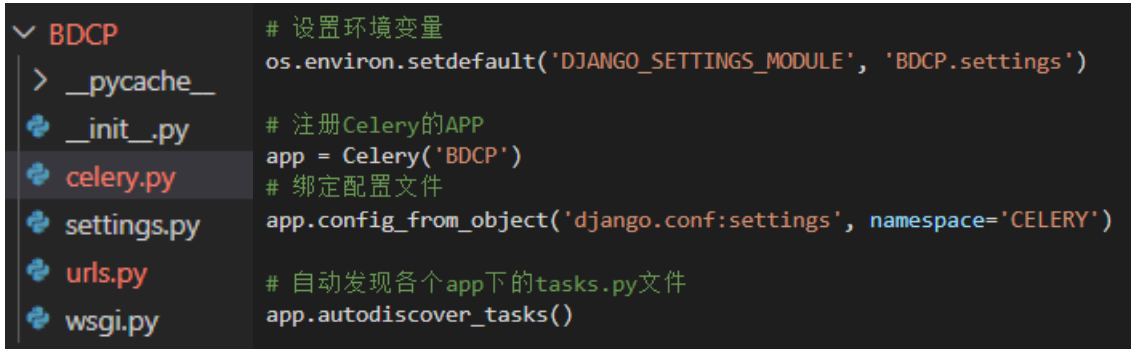
- 异步编程
  - 定义：如果程序调用某个方法，等待其执行全部处理后才能继续执行称其为同步的。相反，在处理完成之前就返回调用方法则是异步。
  - 优点：
    - 快速响应的用户界面；
    - 更高的伸缩性。



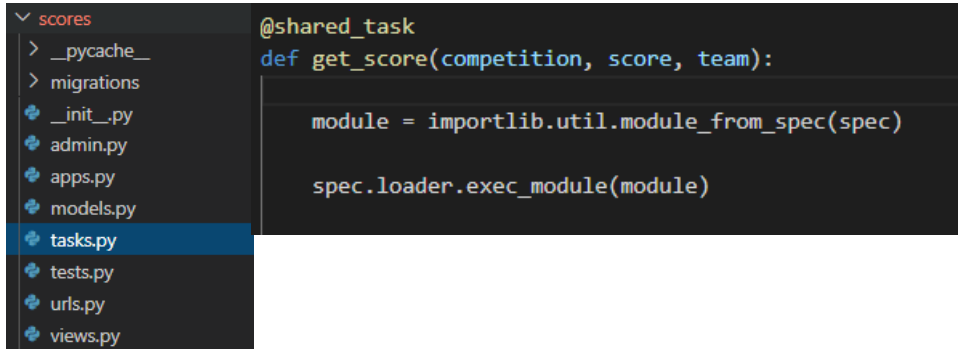


Django异步

- **Django 2.x异步编程**
  - **Celery：Python开发的分布式任务调度模块，支持RabbitMQ、Redis、Mysql等数据库。**
  - **使用方法：**
    - **创建celery.py配置文件；**
    - **编写异步任务task.py任务文件。**



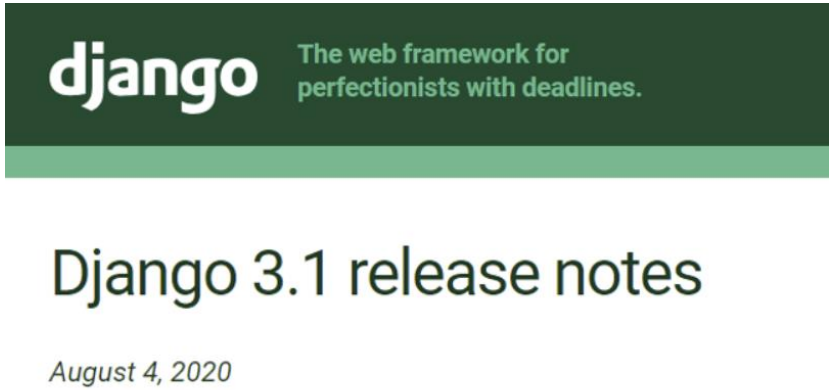
celery.py



task.py

Django异步

- **Django 3**
  - 背景：各技术的异步化趋势，CPU的多核发展特点。
  - 简介：Django 3.1于2020年8月4日正式发布，主要增加的功能为
    - 对异步请求视图和中间件的支持；
    - 新增JSONField字段，支持更多数据库作为后端；
    - 使用pathlib代替原来的os.path更好处理文件路径；
    - 多语言切换的国际化功能优化；
    - ……



Django异步

- Django 3异步编程
  - 支持异步
    - Asynchronous views（异步视图）；
    - Asynchronous middleware（异步中间件）；
    - Asynchronous tests and test client（异步测试和测试客户端）。
  - 使用方法
    - 在普通视图前使用async字段。

```python
from django.http import HttpResponse


async def index(request):
    return HttpResponse("Hello, async Django!")
```

Django异步

- Django 3异步编程
  - 应用案例：
    - 分别创建同步和异步的任务：
      - 每隔一秒输出一个数字然后返回http状态。
    - 查看结果：

```python
# 异步任务
async def http_call_async():
    for num in range(1, 6):
        await asyncio.sleep(1)
        print(num)
    async with httpx.AsyncClient() as client:
        r = await client.get("https://httpbin.org/")
        print(r)

# 同步任务
def http_call_sync():
    for num in range(1, 6):
        sleep(1)
        print(num)
    r = httpx.get("https://httpbin.org/")
    print(r)
```

```python
# 异步视图 - 调用异步任务
async def async_view(request):
    loop = asyncio.get_event_loop()
    loop.create_task(http_call_async())
    return HttpResponse("Non-blocking HTTP request")

# 同步视图 - 调用普通同步任务
def sync_view(request):
    http_call_sync()
    return HttpResponse("Blocking HTTP request")
```

同步和异步任务

```
INFO:     127.0.0.1:60374 - "GET /async/ HTTP/1.1" 200 OK
1
2
3
4
5
```

异步结果

```
2
3
4
5
<Response [200 OK]>
INFO:     127.0.0.1:60375 - "GET /sync/ HTTP/1.1" 200 OK
```

同步结果

Django异步

- Django 3异步编程
  - Celery比较：
    - 代码的编写：
      - Django 3代码量远少于Celery。
    - 应用场景：
      - Django 3适用于较为简单任务，Celery可用于大量或长时间运行的后台程序。
  - 串联使用：
    - Django 3使用异步视图发送电子邮件或对数据库进行一次性修改；
    - Celery每晚在计划的时间清理数据库或生成并发送客户报告。

Django 总结

Django总结

- Django优点：
  - 易于上手：
    - 使用Python的语法规则在Django中创建应用程序；
  - 丰富的框架和资源库：
    - 使用MVT框架加快开发速度并简化应用程序的代码；
    - 不断更新的Django社区提供开发库，例如Django REST框架（构建API）和Django CMS（管理网站内容）等；
    - 通过celery和自身机制支持异步开发；
  - 自带后台管理：
    - 提供一个以Model为中心的后台管理界面；
  - 安全和可靠性检验：
    - 通过CSRFtoken等验证方式使用内置安全功能保护其应用免受攻击，例如跨站点脚本编写，SQL注入等。

Django总结

- Django缺点：
  - 不适用于小型Web应用：
    - Django的附加功能在执行轻量级Web应用时增加不必要的资源消耗；
  - 组件之间的联系性：
    - Django组件的限制导致如果不遵守规则，可能无法实现某些可靠功能；
  - 依靠自己的ORM系统：
    - 框架使用的ORM系统不具有其它广泛使用的ORM系统所提供的强大功能，例如SQLalchemy。

参考资料

- **Django MVT框架：**
  - **https://www.cnblogs.com/lowbi/p/10776600.html**
- **Django 3：**
  - **https://www.jianshu.com/p/5745f8b94289**
- **Celery：**
  - **https://blog.csdn.net/showgea/article/details/109362025**
- **Django框架的选择：**
  - **https://zhuanlan.zhihu.com/p/324399612**

道德经

大成若缺，其用不弊。
大盈若冲，其用不穷。
大直若屈。大巧若拙。
大辩若讷。静胜躁，寒
胜热。清静为天下正。

谢谢！