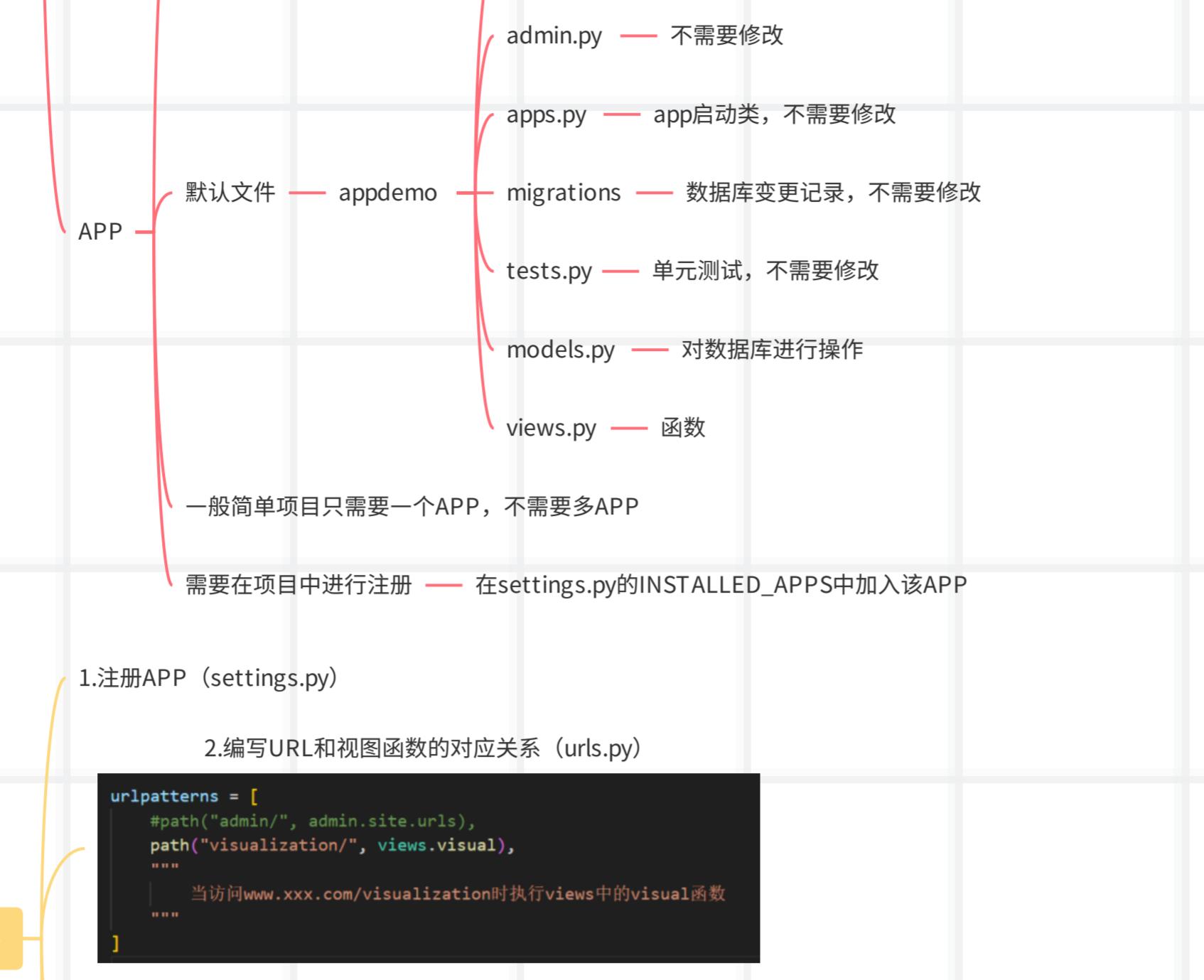




项目创建



1. 注册APP (settings.py)

2. 编写URL和视图函数的对应关系 (urls.py)

```

urlpatterns = [
    path("admin/", admin.site.urls),
    path("visualization/", views.visualization),
    ...
]
    当访问www.xxx.com/visualization时执行views中的visualization函数

```

3. 编写视图函数(APP->views.py)

```

def index(request):
    return HttpResponse("欢迎使用")

```

4. 启动项目 —— 命令行启动 python manage.py runserver

如何找html: 根据app的注册顺序, 在每个app下的templates目录中寻找

开发过程中一般将img,css,js当作静态文件处理

在根目录下创建static文件夹, 配置settings.py文件中的static路径

```

settings.py x urls.py manage.py
visualization > settings.py > ...
119     STATIC_URL = "/static/"
120
121     STATICFILES_DIRS = [
122         os.path.join(BASE_DIR, "static"),
123     ]
124
125

```

引用静态文件: {% static '/img/1.png' %}

本质: html中写一些占位符, 由数据对占位符进行替换

在views.py的函数中return写入字典, html通过{{}}读取字典的值

```

def tpl(request):
    name = "韩超"
    roles = ["管理员", "CEO", "保安"]
    return render(request, 'tpl.html', {"n1": name, "n2": roles})

```

1. html中的for循环

```

<div>
    {% for item in n2 %}
        <span>{{ item }}</span>
    {% endfor %}
</div>

```

2. 根据索引访问列表: list.index

```

def tpl(request):
    name = "韩超"
    roles = ["管理员", "CEO", "保安"]
    return render(request, 'tpl.html', {"n1": name, "n2": roles})

```

```

<ul>
    {% for item in n3.keys %}
        <li>{{ item }}</li>
    {% endfor %}
</ul>

```

```

<ul>
    {% for item in n3.values %}
        <li>{{ item }}</li>
    {% endfor %}
</ul>

```

1. for循环

```

<ul>
    {% for k,v in n3.items %}
        <li>{{ k }} = {{ v }}</li>
    {% endfor %}
</ul>

```

2. 根据key值

```

<hr/>
{{ n3 }}
{{ n3.name }}
{{ n3.salary }}
{{ n3.role }}

```

html访问字典

```

user_info = {"name": "郭留", "salary": 100000, "role": "CTO"}

```

```

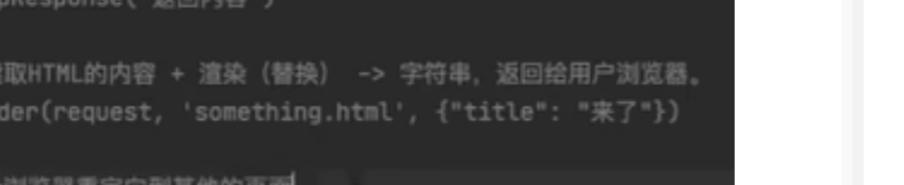
return render(request, 'tpl.html', {"n1": name, "n2": roles, "n3": user_info})

```

```

{% if n1 == "韩超" %}
    <h1>哒哒哒哒哒</h1>
{% elif n1 == "XXX" %}
    <h1>哔哔哔</h1>
{% else %}
    <h1>嘟嘟嘟嘟</h1>
{% endif %}

```



根据这段的render内部
1. 读取含有模板语法的HTML文件
2. 内部进行渲染(模板语法执行并替换数据)

最终得到, 只包含HTML标签的字符串。

3. 将渲染(替换)完成的字符串还给用户浏览器。

request是一个对象, 封装了用户发送过来的所有请求相关的数据

```
# 1. 获取请求方式 GET/POST
print(request.method)
```

```
# 2. 在URL上传递值 /something/?n1=123&n2=999
print(request.GET) I
```

```
# 3. 在请求体中提交数据
print(request.POST)
```

```
# 4. 【响应】HttpResponse("返回内容"), 内容字符串内容返回给请求者。
# return HttpResponse("返回内容")
```

```
# 5. 【响应】读取HTML内容 + 渲染(替换) -> 字符串, 返回给用户浏览器。
# return render(request, 'something.html', {"title": "来了"})
```

```
# 6. 【响应】让浏览器重定向到其他的页面
# return redirect("https://www.baidu.com")
```

内部提供ORM框架, 更加方便

ORM的功能 —— 创建, 修改, 删除数据库中的表, 但是无法创建数据库

操作表中的数据

1. 需要安装第三方模块mysqlclient

2. 自己创建数据库

3. 在settings.py中配置, 连接数据库

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'gx_day15', # 数据库名字
        'USER': 'root',
        'PASSWORD': 'root123',
        'HOST': '127.0.0.1', # 那台机器安装了MySQL
        'PORT': 3306,
    }
}
```