小回顾

1.路由与视图

2.模板

3.项目状态

4.接下来要完成的功能---发布会签到

路由的分发

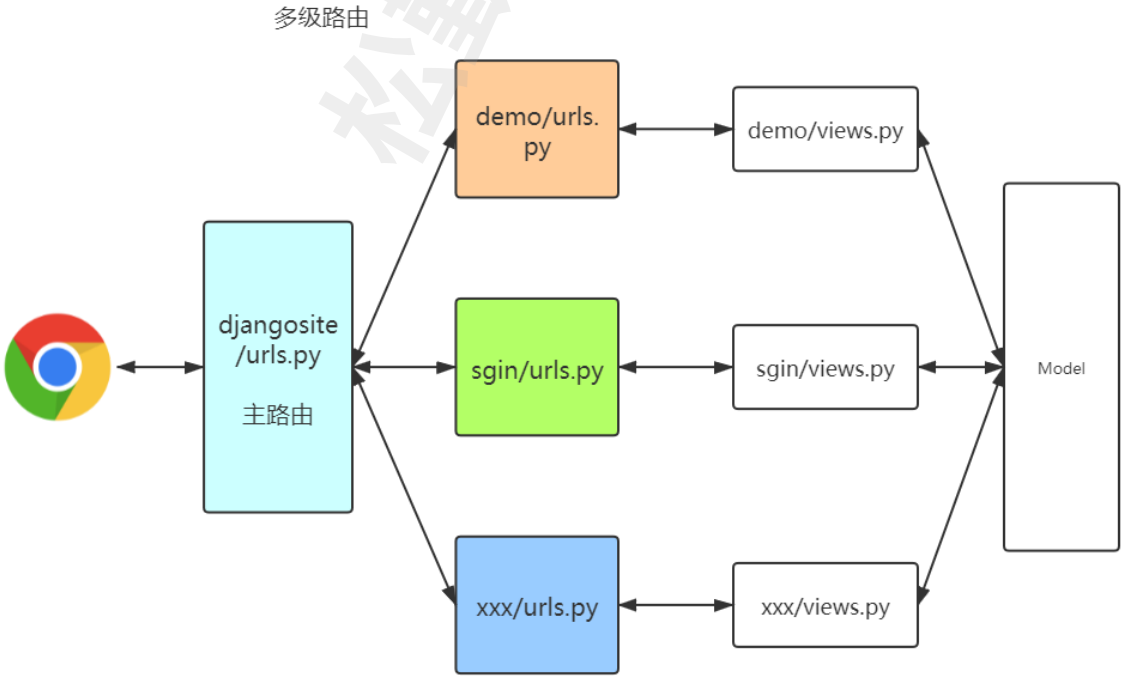
对比下我们目前的路由文件djangosite/urls.py

```
#第一步导入视图函数
from demo import views
from sgin import views as sgin_view
urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/',views.index),  #第二部，配置路由

    #签到系统路由
    path('sgin/events',sgin_view.events),
    path('sgin/event_detail',sgin_view.event_detail),
]
```

不同的app路由都统一集中到了一块，如果项目简单还好，项目复杂，url文件就会变得特别臃肿。

所以为了后期好维护，我们可以将路由按照以下逻辑逐层分解。



我们可以用django的include函数完成这个操作

```
from django.urls import path,include
path('1级路由/',include(视图模块))
```

在app目录下新建分路由文件urls.py(这个文件名可以自定义，一般叫这个名字)

demo/urls.py

```
from django.urls import path,include
from .  import views
urlpatterns = [
    path('index/',views.index),
]
```

sgin/urls.py

```
from django.urls import path,include
from .  import views
urlpatterns = [
    path('events/',views.events),
    path('event_detail/',views.event_detail),
]
```

修改主路由，开头导入include

```
from django.contrib import admin
from django.urls import path,include #导入include

#第一步导入路由
from demo import urls as demo_view
from sgin import urls as sgin_view
urlpatterns = [
    path('admin/', admin.site.urls),
    #路由分发
    path('demo/',include(demo_view)),
    # 签到系统路由
    path('sgin/',include(sgin_view)),
]
```


数据操作--ORM模型

解决发布会假数据的问题　返回的数据都是一样的

保存数据的途径---数据库?

操作数据库的方法----1.原生sql　2.ORM模型

ORM的概念

Object Relational Mapping(对象关系映射)

其主要作用是在编程中，把面向对象的概念跟数据库中表的概念对应起来。 ... 当然这只是从对象到SQL的映射，还有从SQL到对象的映射，也是类似的过程。

有了ORM，我们不需要再写繁琐的sql语句，一切以python语法为准，通过ORM提供的API就可以操作数据库

主流的数据库都被Django 官方支持：

- PostgreSQL
- MariaDB
- MySQL
- Oracle
- SQLite

模型定义

以Event（发布会）为例，继承django自带的model类，并定义相应的字段

每个字段类都对应一种数据库字段格式

- 名称

  字符串　最大长度256　非空

- 开始时间

  datetime格式时间

- 地址

  字符串　最大长度256　非空

- 人数

  整数

- 状态

  布尔　默认 true

- 创建时间

  datetime格式时间

常用的数据库字段类型（ORM）

CharField

最常用的类型，字符串类型。必须接收一个max_length参数，表示字符串长度不能超过该值。

BooleanField

布尔值类型。默认值是None

DateTimeField

日期时间类型。Python的datetime.datetime的实例。

IntegerField

整数类型，范围-2147483648 到 2147483647的值在 Django 支持的所有数据库中都是安全的。

附录：字段类型参考：https://docs.djangoproject.com/zh-hans/3.1/ref/models/fields/#field-types

常用字段选项（ORM）

null

如果是 True， Django 将在数据库中存储空值为 NULL。默认为 False。

default

该字段的默认值。可以是一个值或者是个可调用的对象，不能是一个可更改的对象（模型实例、list、set 等）。

unique

如果设置为 True，这个字段必须在整个表中保持值唯一。默认为False。若为True该字段可以成为一个唯一索引

verbose_name

字段的一个人类可读名称，如果没有给定详细名称，Django 会使用字段的属性名自动创建，并将下划线转换为空格。

剩余通用字段参考:https://docs.djangoproject.com/zh-hans/3.1/ref/models/fields/#field-options

模型字段定义代码参考：

```
#定义发布会模型   --模型名称对应表明   app_class
class Event(models.Model):  #继承django自带的模型基类
    #发布会名称              字符串    最大长度256    非空。
    name = models.CharField(max_length=256)
    # 发布会地点           字符串    最大长度256    非空。
    address = models.CharField(max_length=256)
    #发布会人数    #默认值100
    limits = models.IntegerField(default=100)
    # 发布会状态   布尔类型，默认为True。
    status = models.BooleanField(default=True)
    # 开始时间    日期时间类型。Python的datetime.datetime的实例。
    start_time = models.DateTimeField(null=True)
```

字段命名约束：

Django不允许下面两种字段名：

- 与Python关键字冲突。这会导致语法错误。例如：

```
class Example(models.Model):
    pass = models.IntegerField() # 'pass'是Python保留字!
```

- 字段名中不能有两个以上下划线在一起，因为两个下划线是Django的查询语法。例如：

```
class Example(models.Model):
    foo__bar = models.IntegerField() # 'foo__bar' 有两个下划线在一起!
```

- 字段名不能以下划线结尾，原因同上。

由于你可以自定义表名、列名，上面的规则可能被绕开，但是请养成良好的习惯，一定不要那么起名。

模型方法

可以修改模型自带的**str**方法，这样在查询的时候，就能看到对应的数据信息了

```
#定义发布会模型  --模型名称对应表明  app_class
class Event(models.Model):  #继承django自带的模型基类
    #发布会名称              字符串    最大长度256    非空。
    name = models.CharField(max_length=256)
    # 发布会地点           字符串    最大长度256    非空。
    address = models.CharField(max_length=256)
    #发布会人数    #默认值100
    limits = models.IntegerField(default=100)
    # 发布会状态   布尔类型，默认为True。
    status = models.BooleanField(default=True)
    # 开始时间    日期时间类型。Python的datetime.datetime的实例。
    start_time = models.DateTimeField(null=True)

    # 修改模型自带方法   __str__   调用该对象自动返回对象的name值
    def __str__(self):
        return self.name
```

对比：修改前

```
>>> from sgin.models import Event
>>> Event.objects.all()
<QuerySet [<Event: Event object (1)>]>
```

对比：修改后：

```
>>> from sgin.models import Event
>>> Event.objects.all()
<QuerySet [<Event: 测开发布会>]>
```

激活模型与数据迁移

激活模型

settings.py中添加应用名称以关联app

配置数据库信息

默认使用的是sqlite，不需要配置，若要改成其他数据库，可以参考以下配置修改：如改成mysql

```
# settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',     #数据库引擎
        'NAME': 'sqtp',                           #数据库名称
        'USER': 'root',                           #用户名
        'PASSWORD': 'devops',                     #密码
        'HOST': '192.168.21.140',                 #数据库IP
        'PORT': '3306',                           #数据库端口
    }
}
```

附录：数据库配置 https://docs.djangoproject.com/zh-hans/3.1/intro/tutorial02/

模型迁移3步曲

- 1编辑 `models.py` 文件，改变模型。
- 2运行 `python manage.py makemigrations` 为模型的改变生成迁移文件,类似以下输出。

```
(venv) D:\Course\course_django\djangosite>python manage.py makemigrations
Migrations for 'sgin':
  sgin\migrations\0003_auto_20210311_1848.py
    - Alter field phone on guest
```

- 3运行 `python manage.py migrate` 来应用数据库迁移 ,类似以下输出

```
(venv) D:\Course\course_django\djangosite>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, sgin
Running migrations:
  Applying sgin.0002_guest... OK
  Applying sgin.0003_auto_20210311_1848... OK
```

迁移的内部过程

makemigrations命令---做语法预检查，生成操作数据库的命令，并不操作数据库，此时发生的错误都和数据库无关

migrate命令 将前一步的命令落实到数据库，此时发生的错误都和数据库有关

数据的增删改查API

调用模型API--- 通过模型的管理器

语法：模型类.objects.方法

增加

Model.objects.create(**kws)

修改

数据对象.字段=值

数据对象.save()　　#调用save才会保存到数据库

#千万不要用　Model.objects.update(**kws) 会把所有数据记录都更改

删除

数据对象.delete()　#这里用对象调用删除方法，模型管理器没有，想想为什么?

查询---最丰富的操作

Model.objets.all()　---- 获取所有查询结果，返回QuerySet对象

Model.objets.filter(**kw) --- 条件过滤查询，返回QuerySet对象,不加参数效果和all()一样

Modle.objets.get() --- 获取单个数据对象　#没有匹配结果或者匹配结果超过1个都会报错

查询是web中最频繁也是最丰富的操作

ORM执行查询参考：https://docs.djangoproject.com/zh-hans/3.1/topics/db/queries/

测试API--体验Django自带的命令行

```
python manage.py shell
```

Django自带的后台数据管理

用API手动创建的方式还是太麻烦，后续我们可以用API编写到接口里面

现在，我们先用Django自带的后台来管理数据

1. 模型注册到admin.py
2. 创建管理员用户　python manage.py createsuperuser
3. 新增数据

创建数据--发布会

功能迭代--修改视图

修改event视图

从数据库读取数据

```
event_list = Event.objects.all()
```

修改发布会详情视图 event_detail

修改视图event_detail

```
def event_detail(request,event_id):
    event = Event.objects.get(pk=event_id)
    return render(request,'sgin/event_detail.html',{'event':event})
```

发布会详情页url可以设计成`event_detail/<int:event_id>`

django会把event_detail/2类型的url 解析成`event_detail/<int:event_id>`

event_id 可作为event_detail视图函数的参数，注意参数名需要相同

此为django解析foo/path/数字 类型的url的规则 若变化的值为字符串，将int改为str即可

修改url,增加ID

```
path('sgin/event_detail/<int:event_id>',views.event_detail),
```

修改发布会详情页 event_detail.html

更新样式

```
{% extends 'base.html' %}
{% block content %}
    <div class="panel panel-info">
    <div class="panel-heading"> 发布会详情页 </div>
    <div class="panel-body">
        <p>发布会名称: {{ event.name }}</p>
        <p>发布会时间: {{ event.start_time }}</p>
        <p>发布会地址: {{ event.address }}</p>
        <p><a href="/sgin/events" class="btn btn-info">返回列表</a></p>
    </div>
    </div>
{% endblock %}
```

修改列表页的URL，events.html

```
<a href="/sgin/event_detail/{{ event.id }}">    #加上{{ event.id }}
```

需求2实现嘉宾列表和详情页

1.创建嘉宾数据模型对象

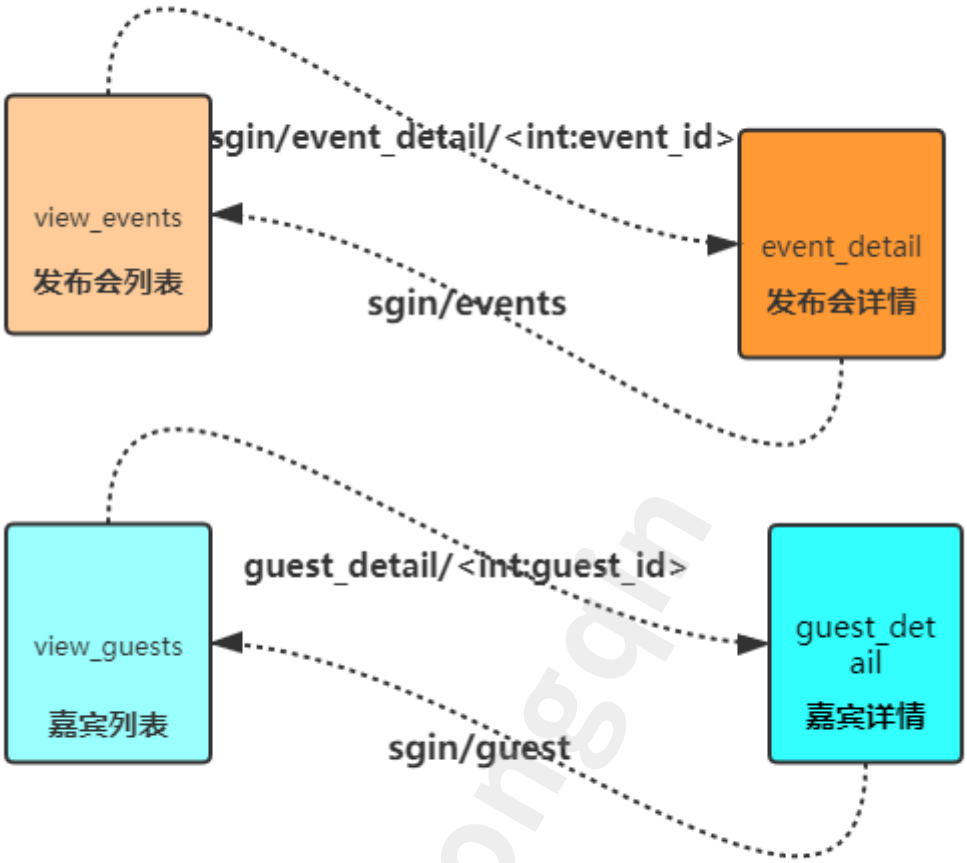
至少包括手机号，姓名，邮箱，和关联的发布会

2.模仿发布会列表和详情页，创建嘉宾视图路由以及模板页面

/sgin/guests/　　#嘉宾列表

/sgin/guest_details/int:id　#嘉宾详情页

现在的视图逻辑关系



按照需求，嘉宾需要与发布会进行关联，这个就涉及到了数据库表关联的关系。

数据库表关联

后端系统开发中， 数据库设计是重中之重。

特别是前后端分离的系统， 后端的职责基本就是数据管理， 开发的代码几乎都是围绕数据操作的。

虽然，我们教程不是专门讲数据库的， 但也必须讲解常用的 数据库表 和 表之间的关系 的设计 。

目前 使用的数据库系统 主要还是 `关系型数据库` 。

什么是关系型数据库？就是建立在关系模型基础上的数据库。

大家耳熟能详的mysql、oracle、 sqlserver、SQLite 都是，而 mongodb、Cassandra不是

而关系型数据库，设计的一个难点就是 `各种表之间的关联关系` 。

常见的3种关联关系就是： `一对多` ， `一对一` ， `多对多`

例子

1对多： 1个老师可以上多门课程，对应多门课程

```
models.ForeignKey       # 定义在多方
```

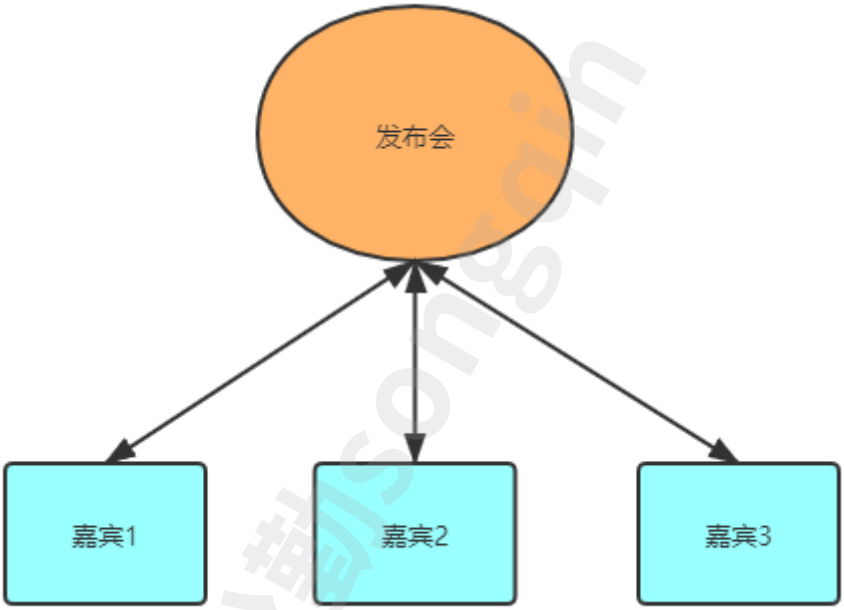
多对多：1个学生可以对应多门课程，1门课程也可以对应多个学生

```
models.ManyToManyField  #定义在任意1方，只能定义在1方，不能双方都定义
```

多对1：1个男盆友对应1个女盆友..

```
models.OneToOneField    # 定义在多方
```

嘉宾的与发布会的关系可先定为多对1，参考下图



模型定义代码参考

```
# 定义发布会关联嘉宾
class Guest(models.Model):
    # django会自动帮你创建一个id字段作为主键
    # 关联发布会
    event = models.ForeignKey(Event,on_delete=models.CASCADE)  #CASCADE 如果删除了关联的发布会，该嘉宾也会被删除
    # 姓名  字符串  64  唯一
    name = models.CharField(max_length=64,unique=True)
    # 手机号  字符串  11  唯一
    phone = models.CharField(max_length=11,unique=True)
    # 邮箱  邮箱格式 xxx@yyy.zz
    email = models.EmailField()
    # 加入时间 --创建数据的时候就自动取当前时间 auto_now_add=True
    join_time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name
```

不要忘记运行命令同步到数据库。同时更新url和view.

urls.py

```
# 嘉宾列表
    path('guests/',views.guests),
    # 嘉宾详情
    # 将guest_id/2类型的url 解析成guest_detail/<int:guest_id>
    # guest_id 可作为guest_detail视图函数的参数，注意参数名需要相同
    path('guest_detail/<int:guest_id>',views.guest_detail),
```

views.py

```
# 嘉宾列表
def guests(request):
    # 从数据库获取所有嘉宾
    guest_list = Guest.objects.filter()
    # 返回页面并填充数据
    return render(request,'guests.html',{'guest_list':guest_list})


# 嘉宾详情
def guest_detail(request,guest_id):
    # 根据获取的ID取对应的嘉宾
    guest = Guest.objects.get(pk=guest_id)
    # 返回页面并且填充嘉宾
    return render(request,'guest_detail.html',{'guest':guest})
```

最后：网页模板

guests.html

```
{% extends 'base.html' %}
{% block content %}
    <ul class="list-group">
        {% for guest in guest_list %}
            <li class="list-group-item text-center"><a
href="/sgin/guest_detail/{{ guest.id }}">{{ guest.name }}</a></li>
        {% endfor %}
    </ul>

{% endblock %}
```

guest_detail.html

```
{% extends 'base.html' %}
{% block content %}
    <div class="panel-info panel">
        <div class="panel-heading"> 嘉宾详情页 </div>
        <div class="panel-body">
        <p>参与发布会：{{ guest.event }}</p>
        <p>姓名：{{ guest.name }}</p>
        <p>手机号：{{ guest.phone }}</p>
```

```
        <p>邮箱：{{ guest.email }}</p>
        <p>加入时间：{{ guest.join_time }}</p>
        <p><a href="/sgin/guests" class="btn btn-info">返回列表</a></p>
    </div>
    </div>
{% endblock %}
```

网页模板更新了样式，以实际代码为准。

知识点梳理

1.数据库ORM 模型定义---字段类型与选项---需要牢记

2.数据库增删改查API

3.数据库表关联

4.Django自带后台管理系统

1，2，3 为重点，需要牢记

附录

模型字段参考：https://docs.djangoproject.com/zh-hans/3.1/ref/models/fields/#field-types

数据库配置：https://docs.djangoproject.com/zh-hans/3.1/intro/tutorial02/

数据库访问优化：https://docs.djangoproject.com/zh-hans/3.1/topics/db/optimization/

字段命名限制：https://docs.djangoproject.com/zh-hans/3.1/topics/db/models/#field-name-restrictions

模型方法：https://docs.djangoproject.com/zh-hans/3.1/topics/db/models/#model-methods

模型管理器：https://docs.djangoproject.com/zh-hans/3.1/topics/db/managers/

ORM执行查询：https://docs.djangoproject.com/zh-hans/3.1/topics/db/queries/

QuerySetAPI参考：https://docs.djangoproject.com/zh-hans/3.1/ref/models/querysets/