Tedu.cn
达内教育

Python开发

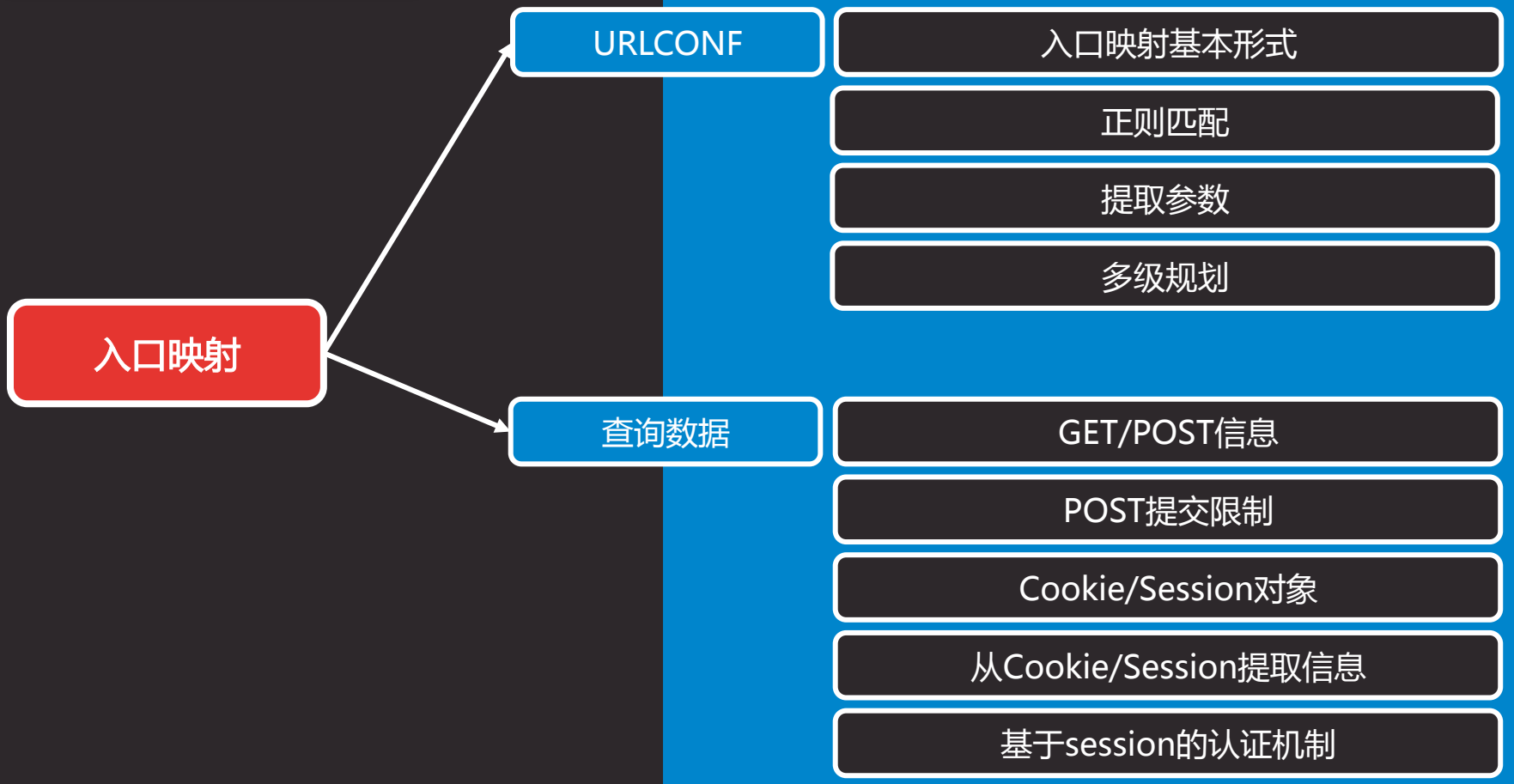
NSD DEVWEB

DAY06

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	入口映射
	10:30 ~ 11:20	
	11:30 ~ 12:00	模板详解
下午	14:00 ~ 14:50	
	15:00 ~ 15:50	模型详解
	16:10 ~ 17:00	
	17:10 ~ 18:00	总结和答疑

入口映射


入口映射
URLCONF
入口映射基本形式
正则匹配
提取参数
多级规划
查询数据
GET/POST信息
POST提交限制
Cookie/Session对象
从Cookie/Session提取信息
基于session的认证机制

URLCONF

入口映射基本形式

- 项目中已存在urls.py，说明如下：

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^hello/$', hello, name='aa', {'age':12,} ),
]
```

views.py：

```
def hello(request):
    return HttpResponse("hello world!!!")
```

知识讲解

正则匹配

- URL正则采用的是match操作
  - r'^hello':匹配以hello开头的任意URL,如:/helloabc
  - r'^hello/$':匹配hello且后面无信息的URL,如:/hello, /hello/
  - r'^$':匹配 / 即空URL,通常用来设定应用的根,即默认入口。如: http://IP:port或者http://IP:port/

知识讲解

提取参数

- URL本身可以用于传递信息,该信息不符合HTTP协议,是非 标准化化信息,需要依赖框架来解释。Django通过正则匹配 分组的方法,获取各段的信息

```
r'^student/(?P<name>\w+)/(?P<age>\w+)/$'
```

```
def test1(request, name='', age='')
```

知识讲解

多级规划

知识讲解

- 一般项目中包含多个相对独立的功能，比如学校管理系统,会包含学生、课程、教师等多种对象的管理。每个对象都需要数据库的增删改查动作。为此，希望的URL为：
  - /student/add/...
  - /student/update/...
  - /student/del/...
  - /student/find/...
- Django为这种设计提供了支持：通过建立应用级的urls映射,构建层级访问控制

多级规划（续1）

- 为将一个应用的映射关系添加到主映射，需要：
  - 添加app：在项目的setting中,在INSTALLED_APPS中添加一个应用名称。假定应 用名称为student。
  - 配置URLCONF：url(r ‘^student/’, include(‘student.urls’))，注意，这里不能加$，因为这只是个一级目录，是完全目录的一部分
  - 在应用student目录下的urls中添加：url(r ‘^add/(?P<name>.+)/(?P<age>.+)/$’,add), url(r ‘^edit/(?P<student_id>.+)/$’,add), ...

知识讲解

多级规划（续2）

- 最终构建的目录为：
  - /student/add/tom/12
  - /student/edit/4

知识讲解

Tedu.cn
达内教育

信息提取

GET/POST信息

- 浏览器应用本身发出的信息通常为GET/POST获取方法如下:

```
xm=request.GET.get['xm’]
xm=request.POST.get['xm']
```

- 如果获取不到，xm为None

知识讲解

POST提交限制

- Django对form表单的POST请求做了限制，需要特别处理一下。表单没有action,就意味着提交给自己

```
<form method=post>
{% csrf_token %}
    name:<input name=xm><br>
    age:<input name=age><br>
  <input type=submit value=go>
</form>
```

知识讲解

Cookie/Session对象

- 概念
  - Cookie：保存在浏览器端的键值对，与域名绑定，有存活时间，用户可以通过浏览器设置里的功能手动清除，或者禁止启用
  - Session：服务器端的存储对象，其可以存储任意类型的数据，但其索引保存在浏览器端的cookie中。由于浏览器端是区别用户的，因此session也是区别用户的

知识讲解

Cookie/Session对象（续1）

知识讲解

- 用途
  - cookie和session都可以用于保存特定用户的专有信息。因此，其值具有不同请求间的共享特性
  - 最常用的一个场景是用其保存用户登陆的状态

- 需要注意的是，session的实现机制依赖浏览器端，但不一定非要依赖cookie

从Cookie/Session提取信息

- Cookie存在于Request的请求head中，供读取。也存在于responpse的返回head中，供写入
- Session通常依赖Cookie而存在，但其主体信息保存在服务器上

```
cookie = request.COOKIES.get('is_logined')
response.set_cookie("is_logined", "100")

username = request.session.get('username')
request.session['username'] = 'Tom'
```

知识讲解

基于session的认证机制

- 鉴权动作序列：
  1. 浏览器访问任意入口，检测session变量(logined)，如果没有设置就跳转/login入口，展示登陆页面
  2. 用户在登陆页面填写账号、密码信息，提交给/login入口，使用数据库鉴定是否是合法用户。如果合法，设置session变量(logined)为任意值，然后跳转到原始路径。
  3. 如果在任意入口检测logined变量存在，则正常显示页面。

知识讲解

基于session的认证机制（续1）

- 通过一个session对象，受保护页面检测是否登陆，如果没有,就去登陆，登陆成功后设置该对象

```
def home(request):
    return render(request, 'polls/home.html')

def login(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        pwd = request.POST.get('pwd')
    if username == 'zhangsan' and pwd == '123456':
        request.session['IS_LOGIN'] = True
        return redirect('index')
    return render(request, 'polls/login.html')
```

知识讲解

基于session的认证机制（续2）


```
def protected(request):
    is_login = request.session.get('IS_LOGIN', False)
    if is_login:
        return HttpResponse('OK')
    return redirect('home')
```

知识讲解

基于session的认证机制（续3）

- URLCONF如下：

```
urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^home/$', views.home, name='home'),
    url(r'^login/$', views.login, name='login'),
    url(r‘^protected/$’, views.protected, name='protected'),  ……
```

知识讲解

基于session的认证机制（续4）

- 模板home.html如下：

```
<form action="/polls/login/" method="post">
{% csrf_token %}
   用户名：<input type="text" name="username"><br>
   密码：<input type="text" name="pwd">
<input type="submit" value="提交">
</form>
```

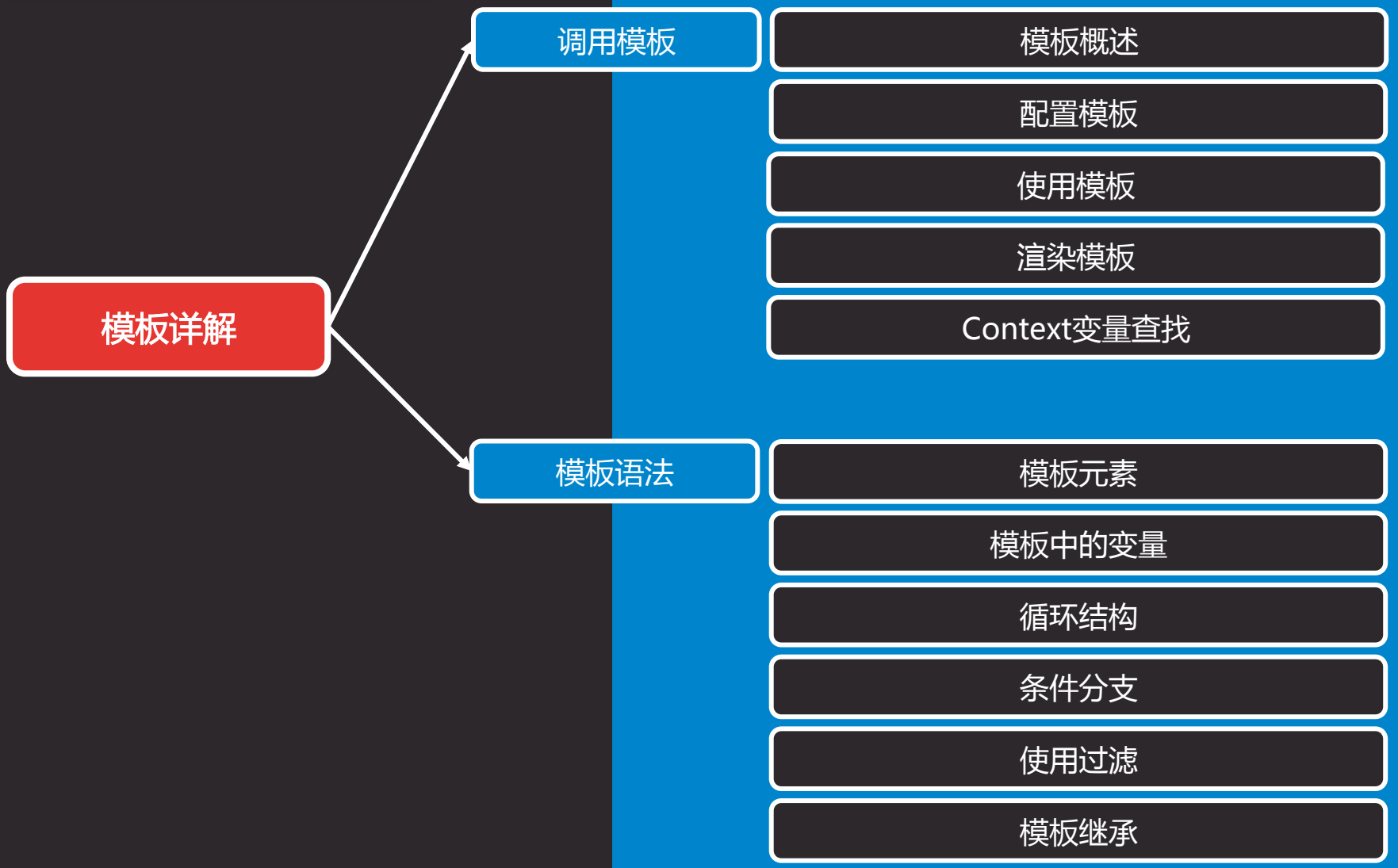
知识讲解

案例1：实现鉴权

1. 编写登陆页面模板
2. 编写三个视图，分别用于登陆页、验证登陆以及受保护页面
3. 如果用户密码正确给出登陆成功，否则重定向到登陆页
4. 编写URLCONF，实现入口

课堂练习

模板详解


模板详解
调用模板
模板概述
配置模板
使用模板
渲染模板
Context变量查找
模板语法
模板元素
模板中的变量
循环结构
条件分支
使用过滤
模板继承

Tedu.cn
达内教育

调用模板

模板概述

- 作为Web 框架，Django 需要一种很便利的方法以动态地生成HTML。 最常见的做法是使用模板
- 模板包含所需HTML 输出的静态部分，以及一些特殊的语法，描述如何将动态内容插入
- Django 项目可以配置一个或多个模板引擎。Django的模板系统自带内建的后台 —— 称为Django 模板语言（DTL），以及另外一种流行的Jinja2

知识讲解

配置模板

- 模板引擎通过TEMPLATES 设置来配置。 它是一个设置选项列表，与引擎一一对应。 默认的值为空。 由startproject 命令生成的settings.py 定义了一些有用的值

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            # ... some options here ...
        },
    },
]
```

知识讲解

配置模板（续1）

- BACKEND 是一个指向实现了Django模板后端API的模板引擎类的带点的Python路径
- 内置的后端有：
  - django.template.backends.django.DjangoTemplates
  - django.template.backends.jinja2.Jinja2

知识讲解

配置模板（续2）

- 由于绝大多数引擎都是从文件加载模板的，所以每种模板引擎都包含两项通用设置
  - DIRS 定义了一个目录列表，模板引擎按列表顺序搜索这些目录以查找模板源文件
  - APP_DIRS 告诉模板引擎是否应该进入每个已安装的应用中查找模板

知识讲解

使用模板

- 在Python代码中使用模板系统的步骤如下：
  - 用模板代码创建一个Template对象。Django提供指定模板文件路径的方式创建Template对象
  - 使用一些给定变量context调用Template对象的render()方法。这将返回一个完全渲染的模板，它是一个string，其中所有的变量和块标签都会根据context得到值

知识讲解

渲染模板

知识讲解

- 一旦拥有一个Template对象，就可以通过给一个context来给它传递数据
- context是一个变量及赋予的值的集合，模板使用它来得到变量的值，或者对于块标签求值
- 这个context由django.template模块的Context类表示
- 它的初始化函数有一个可选的参数：一个映射变量名和变量值的字典

Context变量查找

- 模板系统可以优雅的处理复杂的数据结构，如列表，字典和自定义对象
- 在Django模板系统中处理复杂数据结构的关键是使用(.)字符
- 使用小数点来得到字典的key，属性，对象的索引和方法

知识讲解

Context变量查找（续1）

- 当模板系统遇到变量名里有小数点时会按以下顺序查找：
  - 字典查找，如foo["bar"]
  - 属性查找，如foo.bar
  - 方法调用，如foo.bar()
  - 列表的索引查找，如foo[bar]

知识讲解

Tedu.cn
达内教育

模板语法

模板元素

知识讲解

- 模板中可以使用的元素有：
  - 变量,使用 {{ variable }} 的格式
  - 标签/指令,使用 {% ... %}的格式
  - 字符串:{ } 之外的任何东西,都当做字符串处理
- 上述内容的执行顺序通常就是出现的顺序。在模板里的代码并不是python，也无法支持python的缩进，因此其编程是受限的

模板中的变量

- 简单变量：{{cname}}
- 对象变量：{{ student . cname}}
- 列表对象：{{ student . 1 }}
- 字典对象：{{ student . cname }}

知识讲解

循环结构

- 循环遍历与Python语法很像,但需要endfor配对

```
<table border="1">
    {% for row in data %}
        <tr>
            <td>{{ row.name }}</td> <td>{{ row.age }}</td>
        </tr>
    {% endfor %}
</table>
```

知识讲解

条件分支

- 条件分支，也需要endif配对

```
{% if score >= 90 %}
优秀
{% elif score >= 80 %}
良好
{% elif score >= 70 %}
一般
{% elif score >= 60 %}
需要努力
{% else %}
不及格!
{% endif %}
```

知识讲解

使用过滤

- 过滤函数可以完成各种数据转换功能

{{ msg }}
首字母大写：{{ msg | capfirst }}
转义：{{ msg | addslashes }}
显示行号：{{ msg | linenumbers }}
大写+行号：{{ msg | upper| linenumbers }}
指定日期格式：{{ date | date:"F j, Y" }}
url编码：{{ url | urlencode }}
列表连接：{{ cities |join:", " }}

知识讲解

模板继承

- 一个项目中有很多页面。这些页面总体的样子是相同的，比如页眉、页脚等
- 重复制作每个页面相同的部分显得非常笨拙
- 可以先制作一个基础页面，包含页眉、页脚。其他页面，只要继承基础页面，其他不同内容单独制作

知识讲解

模板继承（续1）

- 基础页base.html示例如下：

```
<html lang="en">
<head>
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    <h1>My helpful timestamp site</h1>
    {% block content %}{% endblock %}
    {% block footer %}
    <hr>
    <p>Thanks for visiting my site.</p>
    {% endblock %}
</body>
</html>
```

知识讲解

模板继承（续2）

- 继承页面date.html示例如下：

```
{% extends "base.html" %}

{% block title %}The current time{% endblock %}

{% block content %}
<p>It is now {{ current_date }}.</p>
{% endblock %}
```

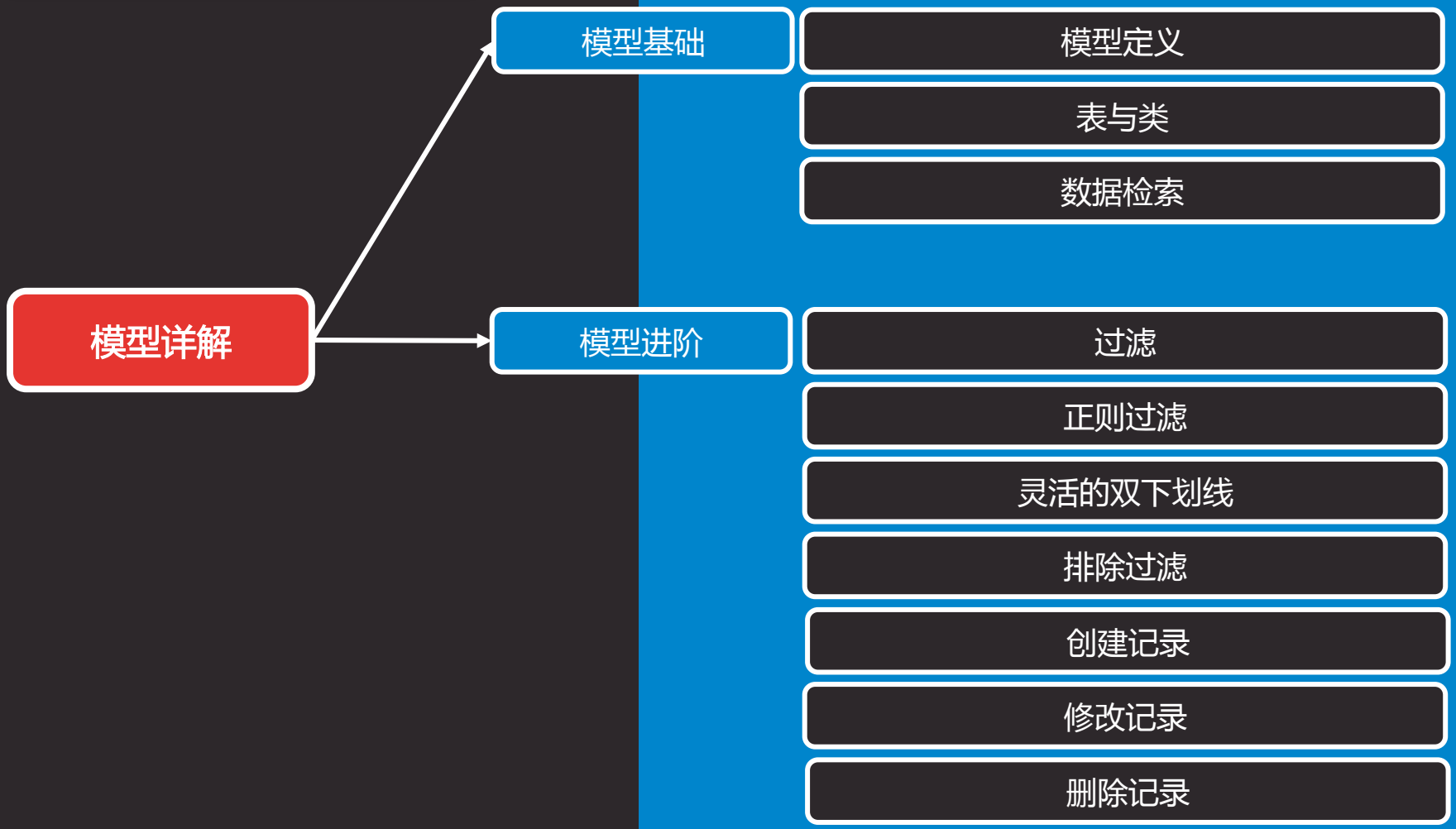
知识讲解

案例2：修改模板

1. 为投票、投票结果、问题详情修改模板
2. 创建一个基础页面
3. 其他模板文件继承于基础页面

课堂练习

模型详解


模型详解
模型基础
模型定义
表与类
数据检索
模型进阶
过滤
正则过滤
灵活的双下划线
排除过滤
创建记录
修改记录
删除记录

Tedu.cn
达内教育

模型基础

模型定义

- 数据库模型编程又称ORM编程。采用ORM，数据库编程更具面向对象的OOP特征
- 模型是数据库表的Python对象表达
- 模型提供了数据库操作的基本功能
- 模型提供了数据库表数据的存储功能
- 模型对单表数据库操作简单易用，但在多表联查，复杂查询时，表达并不简洁，对数据库方面的约束会很多

知识讲解

表与类

- 模型的定义非常类似于数据库表的字段定义。其字段类具有定义、校验的作用

```
class Student(models.Model):
    id = models.IntegerField(primary_key=True)
    cname = models.CharField(
            unique=True,
            max_length=128,
            blank=True,
            null=True
            )
    cage = models.TextField(blank=True, null=True)
```

知识讲解

数据检索

- 获取表的所有记录：
  - Student.objects.all()
- 获取特定条件的记录：
  - Person.objects.get(cname="Alice")
- 获取前10条记录：
  - Person.objects.all()[:10]
- 排序：
  - Book.objects.order_by('name')

知识讲解

过滤

- 严格匹配
  - Person.objects.filter(name="abc")
  - Person.objects.filter(name__exact="abc")
- 不区别大小写
  - Person.objects.filter(name__iexact="abc")
- 姓名中包含
  - Person.objects.filter(name__contains="abc")
- 姓名中包含且不区别大小写
  - Person.objects.filter(name__icontains="abc")

知识讲解

正则过滤

- 采用正则进行过滤
  - Person.objects.filter(name__regex="^abc")
- 采用正则表达式不区分大小写
  - Person.objects.filter(name__iregex="^abc")

知识讲解

灵活的双下划线

- __exact：精确等于，like 'aaa'
- __iexact：精确等于，忽略大小写，ilike ‘aaa’
- __contains：包含，like '%aaa%'
- __icontains：包含，忽略大小写，ilike '%aaa%'
- __gt：大于
- __gte：大于等于
- __lt：小于
- __lte：小于等于

知识讲解

灵活的双下划线

- __in：存在于一个list范围内
- __startswith：以...开头
- __istartswith：以...开头，忽略大小写
- __endswith：以...结尾
- __iendswith：以...结尾，忽略大小写
- __range：在...范围内
- __year：日期字段的年份
- __month：日期字段的月份
- __day：日期字段的日
- __isnull=True/False

排除过滤

- 排除包含
  - Person.objects.exclude(name__contains="Tom")
- 过滤+排除的连续操作
  - Person.objects.filter(
    name__contains="abc").exclude(age =23)

知识讲解

创建记录

- 通过create方法

```
Person.objects.create(name='Tom',age=12)
```

- 创建实例

```
p = Person(name="Tom", age=23)
p.save()
```

- 修改属性

```
p = Person(name="Tom")
p.age = 23
p.save()
```

知识讲解

创建记录（续1）

- 还有一种方法可以防止重复，返回一个元组，第一个为Person对象，第二个为True或False，新建时返回的是True，已经存在时返回 False

```
Person.objects.get_or_create(name="bob", age=23)
```

知识讲解

修改记录

- 通过save方法

```
s = Student(id=4, cname='Tom', age=12)
s.save()
```

- 通过update方法

```
Student.objects.filter(id=4).update(cname='Tom', age=33)
```

知识讲解

删除记录

- 直接删除

```
student = Student()
student.id = 13
student.delete()
```

- 查找对象后删除

```
s=Student.objects.get(id=13)
s.delete()
```

知识讲解

案例3：熟悉模型

- 进入python shell
- 导入模型
- 对模型进行检索、增删改查操作

课堂练习

Tedu.cn
达内教育

总结和答疑