Tedu.cn
达内教育

Python开发

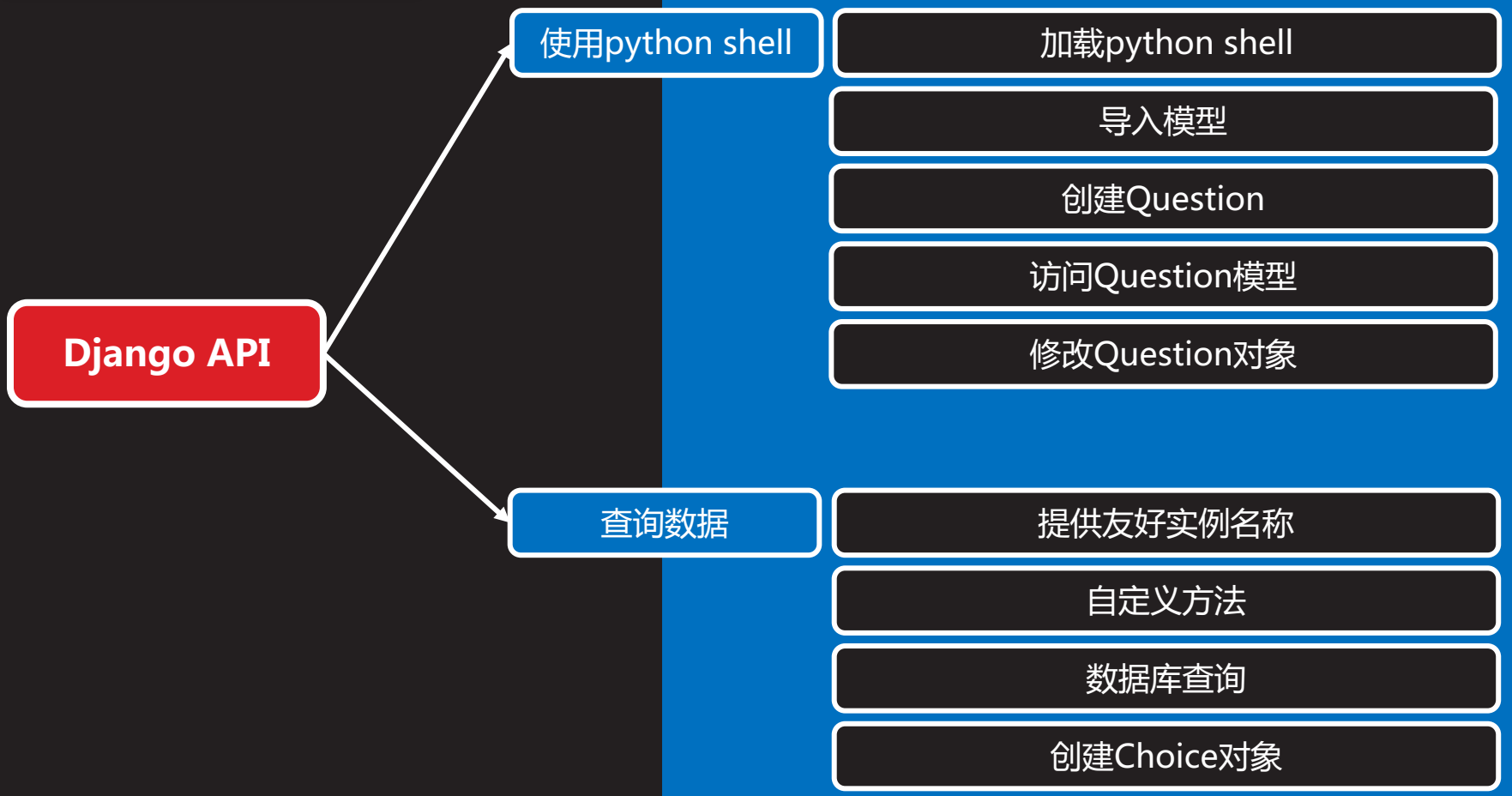
NSD DEVWEB

DAY04

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	Django API
	10:30 ~ 11:20	
	11:30 ~ 12:00	视图和模板
下午	14:00 ~ 14:50	
	15:00 ~ 15:50	使用表单
	16:10 ~ 17:00	
	17:10 ~ 18:00	总结和答疑

Django API


Django API
使用python shell
加载python shell
导入模型
创建Question
访问Question模型
修改Question对象
查询数据
提供友好实例名称
自定义方法
数据库查询
创建Choice对象

使用python shell

加载python shell

- 使用如下命令来调用Python shell：

```
(django_env) [root@localhost mysite]# python manage.py shell
```

- 使用上述命令而不是简单地键入“python”进入python环境
- 因为manage.py 设置了DJANGO_SETTINGS_MODULE 环境变量，该环境变量告诉Django导入mysite/settings.py文件的路径

知识讲解

导入模型

- 导入前面编写的模型

```
>>> from polls.models import Question, Choice
```

- 查看所有的问题

```
>>> Question.objects.all()
<QuerySet []>
```

- 因为还没有创建任何问题，所以返回的是一个空查询集

知识讲解

创建Question

知识讲解

- Question模型中需要时间，可以使用django工具

```
>>> from django.utils import timezone
```

- 创建问题

```
>>> q = Question(question_text="你希望进入哪个公司工作?",
pub_date=timezone.now())
>>> q.save()
```

- 保存这个对象到数据库中。 必须显示地调用save()

访问Question模型

- 一旦创建好了对象，就可以对其进行访问了

```
>>> q.id
1
>>> q.question_text
'你希望进入哪个公司工作?'
>>> q.pub_date
datetime.datetime(2018, 5, 18, 14, 50, 11, 282269, tzinfo=<UTC>)
```

知识讲解

修改Question对象

- 通过改变属性来改变字段的值，然后调用save()

```
>>> q.question_text = "你期待哪个公司给你发offer?"
>>> q.save()
```

- 查询结果

```
>>> Question.objects.all()
<QuerySet [<Question: Question object>]>
```

知识讲解

Tedu.cn
达内教育

查询数据

提供友好实例名称

- <Question object> 完全是这个对象无意义的表示。
  接下来修复这个问题：

```
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    def __str__(self):
        return self.question_text

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
    def __str__(self):
        return self.choice_text
```

知识讲解

提供友好实例名称（续1）

- 修改完毕，重新加载

```
(django_env) [root@localhost mysite]# python manage.py shell
>>> from polls.models import Question, Choice
>>> Question.objects.all()
<QuerySet [<Question: 你期待哪个公司给你发offer?>]>
```

知识讲解

自定义方法

- 模型虽然与数据库具有映射关系，但是它也可以像普通的类一样进行方法的创建

```
import datetime
from django.db import models
from django.utils import timezone

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    def __str__(self):
        return self.question_text
    def was_published_recently(self):
        return self.pub_date >= timezone.now() - datetime.timedelta(days=1)
```

知识讲解

数据库查询

- Django 提供了丰富的数据库查询 API
- 通过关键字查询

```
>>> from polls.models import Question, Choice
>>> Question.objects.get(id=1)
<Question: 你期待哪个公司给你发offer?>
```

- 如果ID不存在，将引发一个异常

```
>>> Question.objects.get(id=2)
……
polls.models.DoesNotExist: Question matching query does not exist.
```

知识讲解

数据库查询（续1）

知识讲解

- 通过主键查询数据是常见的情况，因此 Django 提供了精确查找主键的快捷方式

```
>>> Question.objects.get(pk=1)
<Question: 你期待哪个公司给你发offer?>
```

- 使用自定义方法

```
>>> q = Question.objects.get(pk=1)
>>> q.was_published_recently()
True
```

数据库查询（续2）

- Django通过灵活的双下划线实现属性查找

```
>>> Question.objects.filter(id=1)
<QuerySet [<Question: 你期待哪个公司给你发offer?>]>
>>> Question.objects.filter(question_text__startswith='你')
<QuerySet [<Question: 你期待哪个公司给你发offer?>]>
>>> from django.utils import timezone
>>> current_year = timezone.now().year
>>> Question.objects.get(pub_date__year=current_year)
<Question: 你期待哪个公司给你发offer?>
```

知识讲解

创建Choice对象

- 由于存在外键关系，django通过Question对象可以反向得到Choice对象集

```
>>> q.choice_set.all()
<QuerySet []>
>>> q.choice_set.create(choice_text='阿里巴巴', votes=0)
<Choice: 阿里巴巴>
>>> q.choice_set.create(choice_text='华为', votes=0)
<Choice: 华为>
>>> c = q.choice_set.create(choice_text='达内', votes=0)
>>> c.question
<Question: 你期待哪个公司给你发offer?>
>>> q.choice_set.all()
<QuerySet [<Choice: 阿里巴巴>, <Choice: 华为>, <Choice: 达内>]>
>>> q.choice_set.count()
3
```

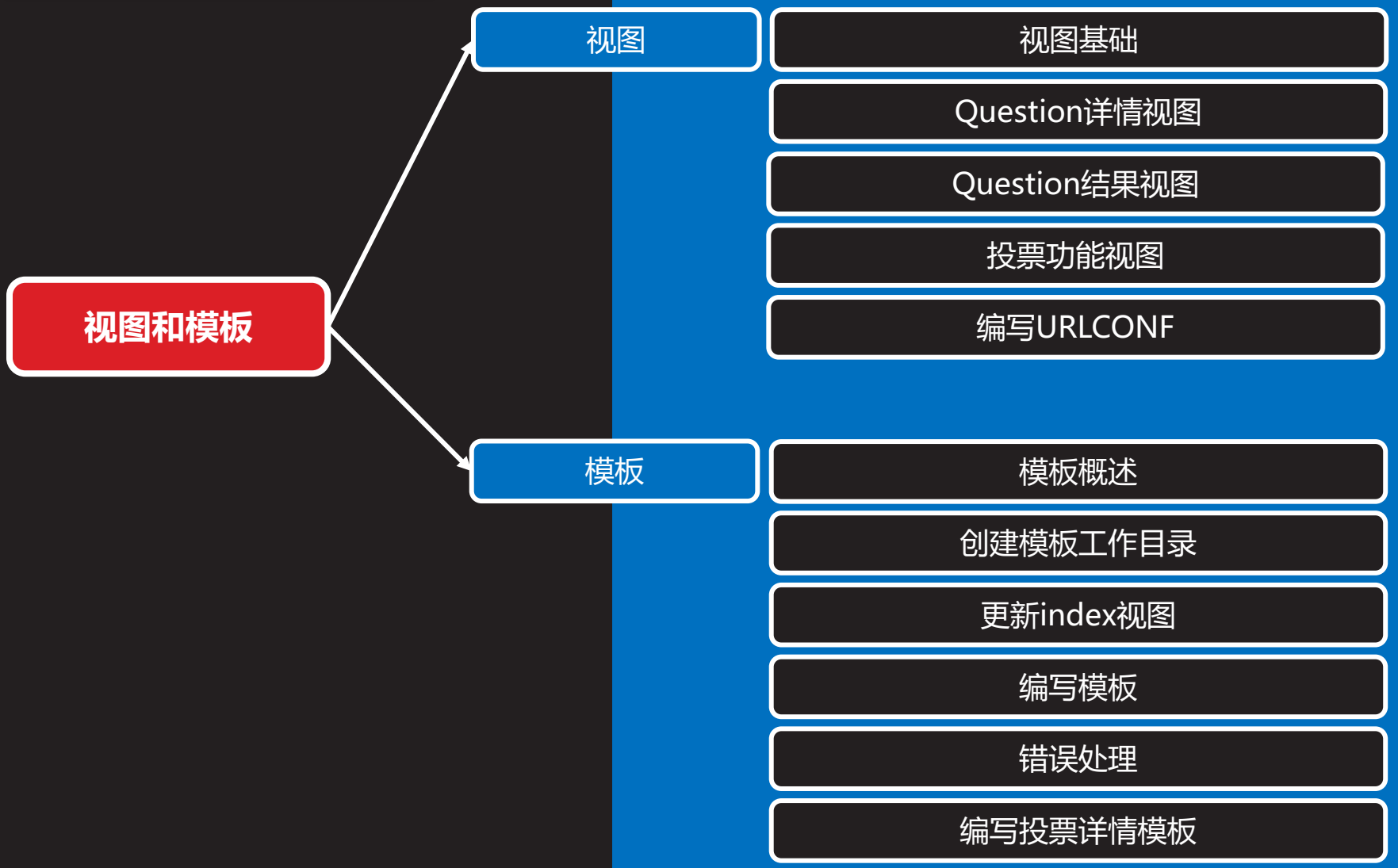
知识讲解

案例1：熟悉API

1. 进入python shell模式
2. 导入Question和Choice类
3. 创建Question和Choice对象
4. 通过id等条件查找对象

课堂练习

视图和模板


视图和模板
视图
视图基础
Question详情视图
Question结果视图
投票功能视图
编写URLCONF
模板
模板概述
创建模板工作目录
更新index视图
编写模板
错误处理
编写投票详情模板

Tedu.cn
达内教育

视图

视图基础

- 在Django中，网页的页面和其他内容都是由视图来传递的（视图对WEB请求进行回应）
- 每个视图都是由一个简单的Python函数(或者是基于类的视图的方法)表示的
- Django通过检查请求的URL（准确地说，是URL里域名之后的那部分）来选择使用哪个视图

知识讲解

Question详情视图

- 编写一个视图，显示单个Question的具体内容，不显示该议题的当前投票结果，而是提供一个投票的表单

```
def detail(request, question_id):
    return HttpResponse("你正在查看的问题是：%s。" % question_id)
```

知识讲解

Question结果视图

- 编写视图，显示特定的Question的投票结果

```
def result(request, question_id):
    response = "你正在查看问题[%s]的结果。"
    return HttpResponse(response % question_id)
```

知识讲解

投票功能视图

- 编写视图，处理对Question中Choice的投票

```
def vote(request, question_id):
    return HttpResponse("您正在为[%s]投票。")
```

知识讲解

编写URLCONF

知识讲解

- 将视图和polls.urls模块关联
- 当客户端向你的网站请求一个页面（ 例如“/polls/34/”）时，Django将加载mysite.urls Python模块
- 因为它被指向ROOT_URLCONF设置， 它寻找名为urlpatterns 的变量并按顺序匹配其中的正则表达式
- 在 ‘^polls/’ 找到匹配后，它将取消匹配的文本（“polls/”），并发送剩余的文本（ “34/”）到'polls.urls'URLconf进行进一步处理

编写URLCONF（续1）

- 编辑polls/urls.py

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^(?P<question_id>[0-9]+)/$', views.detail, name='detail'),
    url(r'^(?P<question_id>[0-9]+)/results/$', views.result, name='result'),
    url(r'^(?P<question_id>[0-9]+)/vote/$', views.vote, name='vote'),
]
```

知识讲解

案例2：编写视图

1. 为投票系应用编写投票功能视图、问题详情视图以及问题结果视图
2. 为第1步的三个视图编写URLCONF，通过相应的URL可以调用对应的视图函数

课堂练习

Tedu.cn
达内教育

模板

模板概述

- 前面的例子，HTML是硬编码在python代码中的。这会带来不少问题：
  - 任何页面的改动会牵扯到Python代码的改动
  - 写后台Python代码与设计HTML是不同的工作，更专业的Web开发应该将两者分开
  - 程序员写Python代码同时页面设计者写HTML模板会更高效，而不是一个人等待另一个人编辑同样的文件

知识讲解

模板概述（续1）

- 为了解决硬码的这些问题，可以模板：
  - 网站的设计改动会比Python代码改动更频繁，所以如果我们将两者分离开会更方便
  - 页面设计者和HTML/CSS程序员不应该编辑Python代码，他们应该与HTML打交道
  - 使用Django的模板系统分离设计和Python代码会更干净更易维护

知识讲解

创建模板工作目录

知识讲解

- 默认的设置文件settings.py配置了一个DjangoTemplates后端，其中将APP_DIRS选项设置为True
- DjangoTemplates在 INSTALLED_APPS所包含的每个应用的目录下查找名为"templates"子目录
- 模板应该位于polls/templates/polls/目录下

```
[root@localhost mysite]# mkdir -p polls/templates/polls
```

更新index视图

- 修改polls/views.py的index函数

知识讲解

```
from django.shortcuts import render
from .models import Question

def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    context = {'latest_question_list': latest_question_list}
    return render(request, 'polls/index.html', context)
```

编写模板

- 创建模板文件polls/templates/polls/index.html，模板语言后续课程中有详细介绍

```
{% if latest_question_list %}
    <ul>
    {% for question in latest_question_list %}
        <li><a
href="/polls/{{ question.id }}/">{{ question.question_text }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

知识讲解

编写模板（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

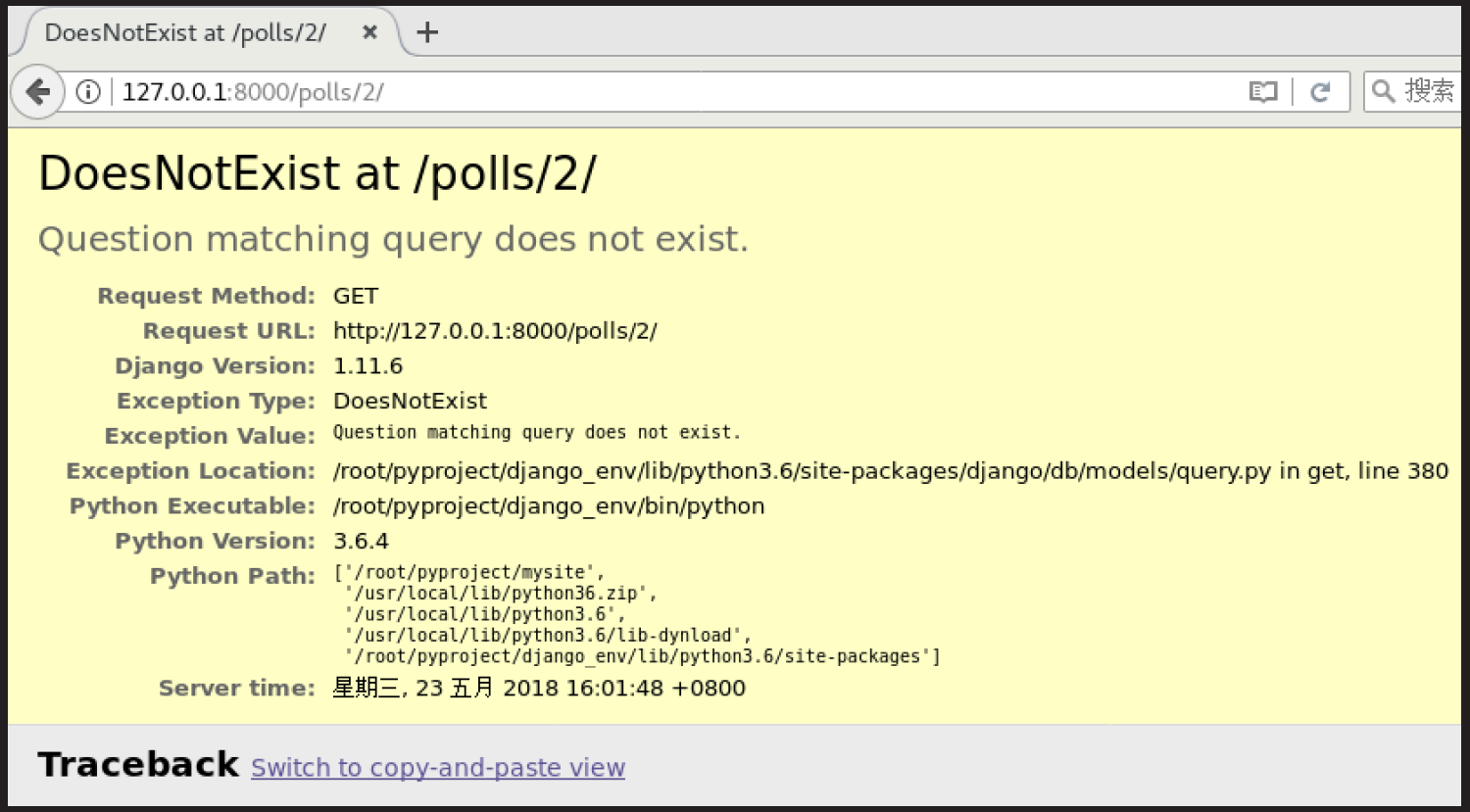
编写模板（续2）

- 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

知识讲解

错误处理

- 当请求一个不存在的对象时，django将会抛出异常



知识讲解

错误处理（续1）

- 一种常见的习惯是使用get()并在对象不存在时引发Http404。 Django为此提供一个快捷方式

```
from django.shortcuts import render, get_object_or_404

def detail(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/detail.html', {'question': question})
```

知识讲解

编写投票详情模板

- 创建polls/templates/polls/detail.html

```
<h1>{{ question.question_text }}</h1>
<ul>
{% for choice in question.choice_set.all %}
    <li>{{ choice.choice_text }}</li>
{% endfor %}
</ul>
```

知识讲解

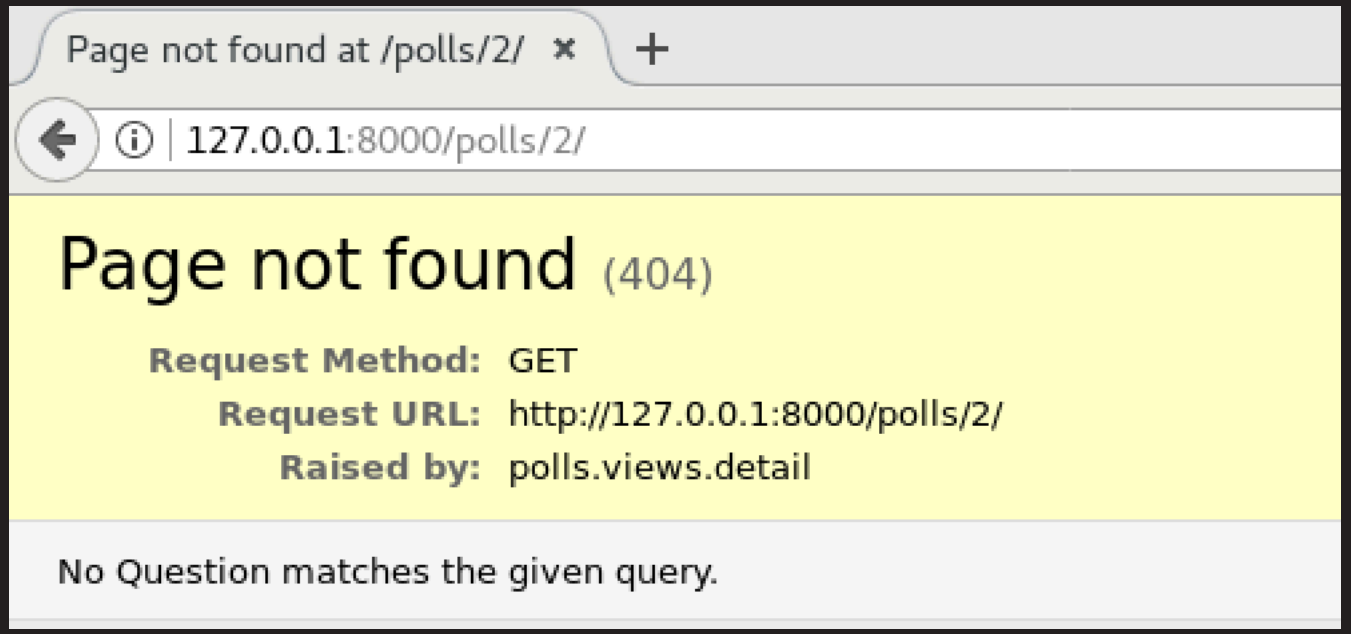
编写投票详情模板（续1）

- 没有发生异常时的页面

知识讲解

编写投票详情模板（续2）

- 异常发生时的页面



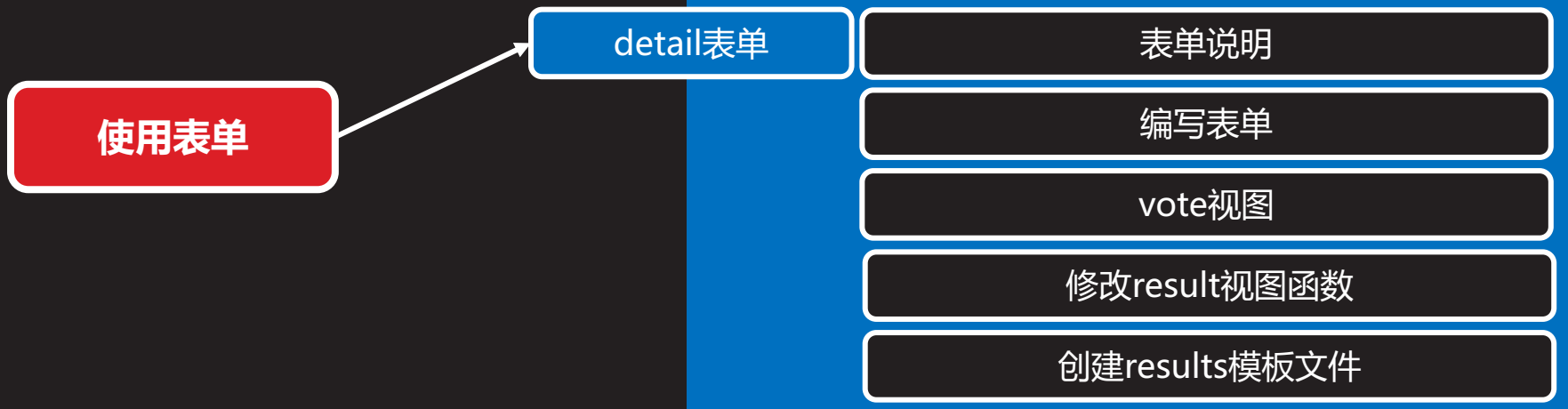
知识讲解

案例3：创建模板

1. 为投票、投票结果、问题详情编写视图
2. 为投票、投票结果、问题详情编写模板
3. 访问相应url，观察是否是用到了正确的模板

课堂练习

使用表单


使用表单
detail表单
表单说明
编写表单
vote视图
修改result视图函数
创建results模板文件

detail表单

表单说明

- 在detail网页模板中，我们为Question对应的每个Choice都添加了一个单选按钮用于选择。 每个单选按钮的value属性是对应的各个Choice的ID。
- 每个单选按钮的name是“choice”。 这意味着，当选择一个单选按钮并提交表单提交时，它将发送一个POST数据choice=#，其中#为选择的Choice的ID

知识讲解

编写表单

- 更新投票详细页面的模板detail.html

```
<h1>{{ question.question_text }}</h1>

{% if error_message %}<p><strong>{{ error_message }}</strong></p>{%
endif %}
<form action="/polls/{{ question.id }}/vote/" method="post">
{% csrf_token %}
{% for choice in question.choice_set.all %}
    <input type="radio" name="choice" id="choice{{ forloop.counter }}"
value="{{ choice.id }}" />
    <label
for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br />
{% endfor %}
<input type="submit" value="投票" />
</form>
```

知识讲解

编写表单（续1）

- forloop.counter指示for标签已经循环多少次
- 由于我们创建一个POST表单（它具有修改数据的作用），所以我们需要小心跨站点请求伪造
- Django已经拥有一个用来防御它的非常容易使用的系统
- 简而言之，所有针对内部URL的POST表单都应该使用{% csrf_token %}模板标签

知识讲解

vote视图

- 修改vote函数，使之可以真正投票

```
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        choices = question.choice_set.get(pk=request.POST['choice'])
    except(KeyError, Choice.DoesNotExist):
        return render(request, 'polls/detail.html', {
            'qustion': question,
            'error_message': '您没有做出任何选择',
        })
    else:
        choices.votes += 1
        choices.save()
        return redirect('result', question_id=question_id)
```

知识讲解

vote视图（续1）

- request.POST 是一个类似字典的对象，可以通过关键字的名字获取提交的数据
- request.POST['choice'] 以字符串形式返回选择的Choice的ID。request.POST 的值永远是字符串
- 如果在POST数据中没有提供request.POST['choice']，choice将引发一个KeyError

知识讲解

修改result视图函数

- 修改vote函数，使之可以真正投票

```
def result(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/results.html', {'question': question})
```

知识讲解

创建results模板文件

- 创建polls/templates/polls/results.html模板文件

```
<h1>{{ question.question_text }}</h1>

<ul>
{% for choice in question.choice_set.all %}
    <li>{{ choice.choice_text }} : {{ choice.votes }}</li>
{% endfor %}
</ul>

<a href="/polls/">投票首页</a>
```

知识讲解

案例4：完成投票系统

- 为投票应用增加表单，使用户可以完成投票操作
- 修改投票的视图函数
- 修改模板文件
- 使投票应用成为真正可用的程序

课堂练习

Tedu.cn
达内教育

总结和答疑