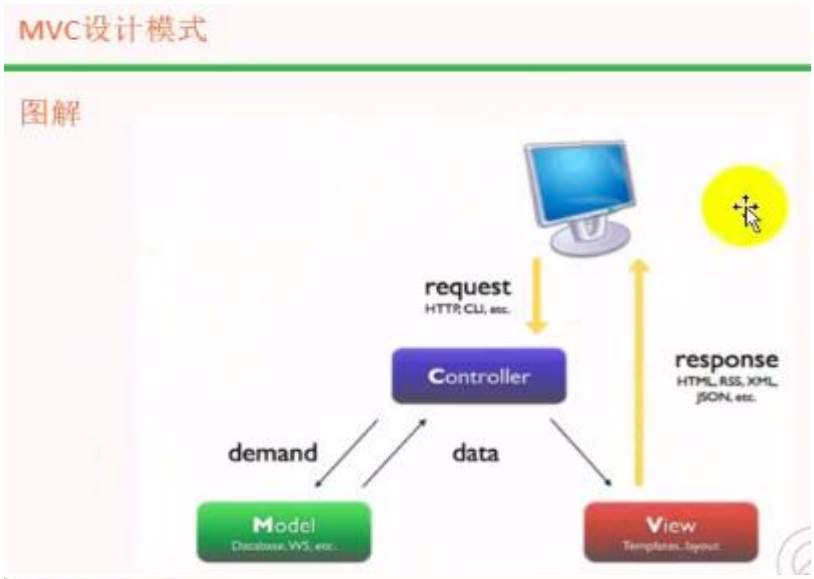
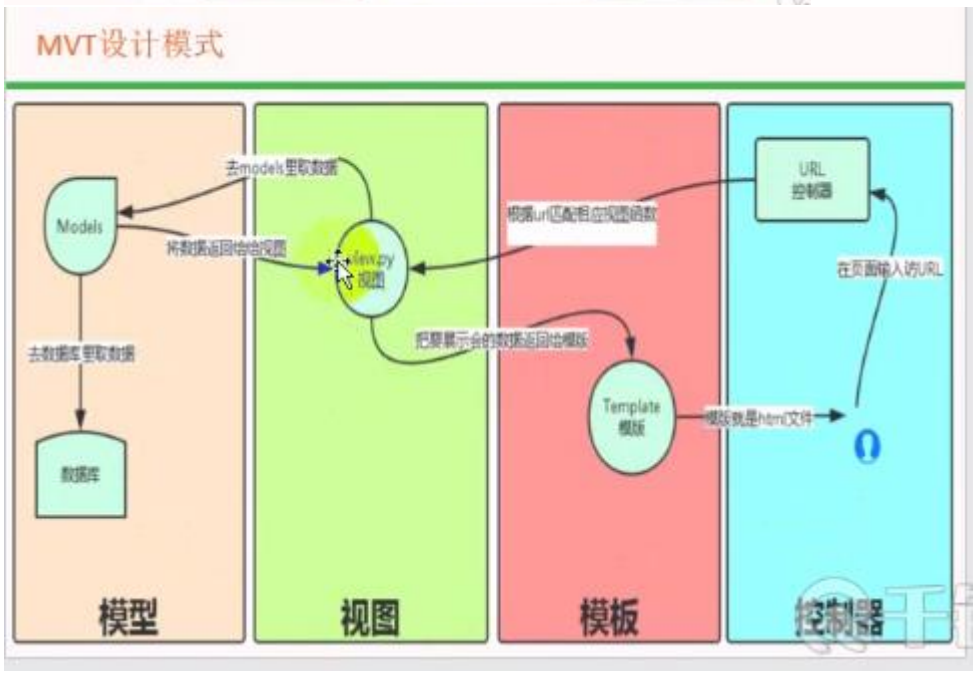
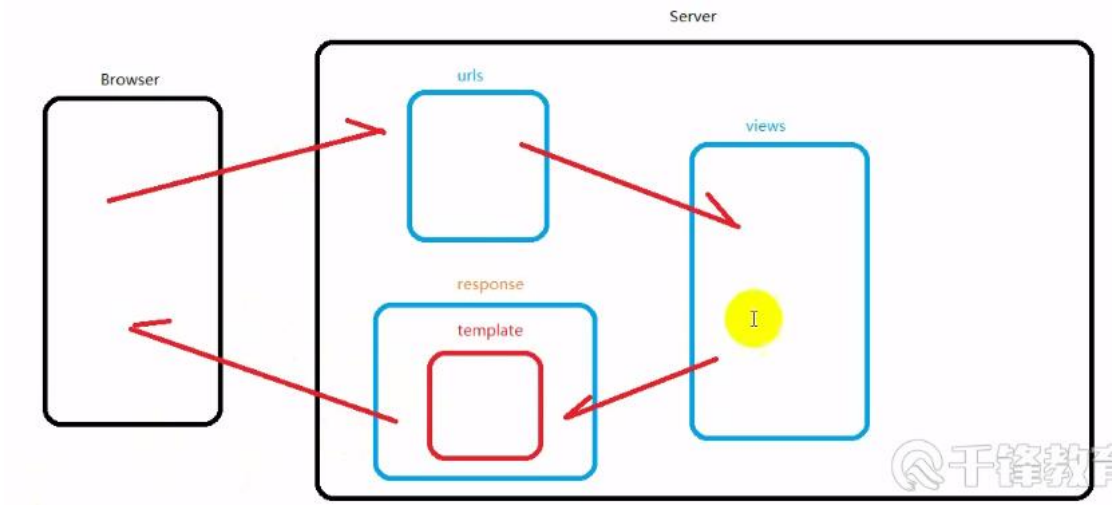
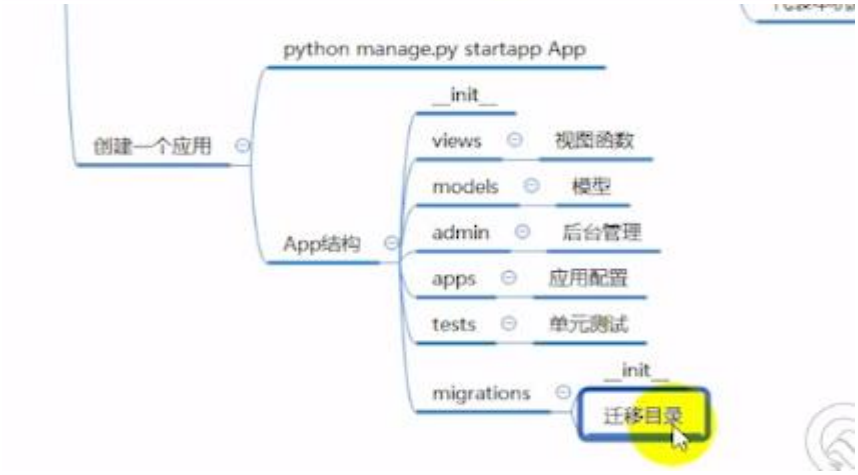
简介


MVC设计模式
图解
request
HTTP, CLI, etc.
Controller
response
HTML, RSS, XML, JSON, etc.
demand
data
Model
View


MVT设计模式
去models里取数据
Models
将数据返回给视图
views.py
视图
根据url匹配相应视图函数
URL
控制器
在页面输入访问URL
把要展示的数据返回给模版
去数据库里取数据
Template
模版
模版就是html文件
数据库
模型
视图
模板
控制器


Server
Browser
urls
views
response
template

```
## 连接mysql驱动
- mysqlclient
    - python2,3都能直接使用
    - 致命缺点
        - 对mysql安装有要求，必须指定位置存在配置文件
- python-mysql
    - python2 支持很好
    - python3 不支持
- pymysql
    - python2，python3都支持
    - 它还可以伪装成前面的库
```



Model

- 在企业开发中，我们通常都是从数据开始开发的

ORM

- 对象关系映射
- 可以理解为翻译机
- 核心思想，解耦合
    - 将业务逻辑和SQL进行了解耦

逻辑删除
- 对于重要数据都做逻辑删除，不做物理删除，实现方法是定义isDelete属性，类型为BooleanField，默认值为False

字段类型
- AutoField
  - 一个根据实际ID自动增长的IntegerField，通常不指定如果不指定，一个主键字段将自动添加到模型中
- CharField(max_length=字符长度)
  - 字符串，默认的表单样式是 TextInput
- TextField
  - 大文本字段，一般超过4000使用，默认的表单控件是Textarea
- IntegerField
  - 整数
- DecimalField(max_digits=None, decimal_places=None)
  - 使用python的Decimal实例表示的十进制浮点数
  - 参数说明
    - DecimalField.max_digits
      - 位数总数
    - DecimalField.decimal_places
      - 小数点后的数字位数
- FloatField
  - 用Python的float实例来表示的浮点数
- BooleanField
  - true/false 字段，此字段的默认表单控制是CheckboxInput
- NullBooleanField
  - 支持null、true、false三种值
- DateField([auto_now=False, auto_now_add=False])
  - 使用Python的datetime.date实例表示的日期
  - 参数说明
    - DateField.auto_now
      - 每次保存对象时，自动设置该字段为当前时间，用于"最后一次修改"的时间戳，它总是使用当前日期，默认为false
    - DateField.auto_now_add
      - 当对象第一次被创建时自动设置当前时间，用于创建的时间戳，它总是使用当前日期，默认为false
  - 说明
    - 该字段默认对应的表单控件是一个TextInput。在管理员站点添加了一个JavaScript写的日历控件，和一个"Today"的快捷按钮，包含了一个额外的invali
  - 注意
    - auto_now_add, auto_now, and default 这些设置是相互排斥的，他们之间的任何组合将会发生错误的结果
- TimeField
  - 使用Python的datetime.time实例表示的时间，参数同DateField
- DateTimeField
  - 使用Python的datetime.datetime实例表示的日期和时间，参数同DateField
- FileField
  - 一个上传文件的字段
- ImageField
  - 继承了FileField的所有属性和方法，但对上传的对象进行校验，确保它是个有效的image

字段选项
- 概述
  - 通过字段选项，可以实现对字段的约束
  - 在字段对象时通过关键字参数指定
- null
  - 如果为True，Django 将空值以NULL 存储到数据库中，默认值是 False
- blank
  - 如果为True，则该字段允许为空白，默认值是 False
- 注意
  - null是数据库范畴的概念，blank是表单验证证范畴的
- db_column
  - 字段的名称，如果未指定，则使用属性的名称
- db_index
  - 若值为 True，则在表中会为此字段创建索引
- default
  - 默认值
- primary_key
  - 若为 True，则该字段会成为模型的主键字段
- unique
  - 如果为 True，这个字段在表中必须有唯一值

模型过滤

- filter
- exclude
- 连续使用
  - 链式调用
  - Person.objects.filter().filter().xxxx.eclude().exclude().yyyy

返回查询集的方法称为过滤器

all()	返回所有数据
filter()	返回符合条件的数据
exclude()	过滤掉符合条件的数据
order_by()	排序
values()	一条数据就是一个字典，返回一个列表

返回单个数据

get()：返回一个满足条件的对象
如果没有找到符合条件的对象，会引发 模型类.DoesNotExist异常
如果找到多个，会引发 模型类.MultiObjectsReturned 异常

first()：返回查询集中的第一个对象

last()：返回查询集中的最后一个对象

count()：返回当前查询集中的对象个数

exists()：判断查询集中是否有数据，如果有数据返回True没有反之

状态码

- 2xx
  - 请求成功
- 3xx
  - 转发或重定向
- 4xx
  - 客户端错误
- 5xx
  - 服务器错了
  - 后端开发人员最不想看到的

获取单个对象

- 查询条件没有匹配的对象，会抛异常，DoesNotExist
- 如果查询条件对应多个对象，会抛异常，MultipleObjectsReturned

first和last

- 默认情况下可以正常从QuerySet中获取
- 隐藏bug
  - 可能会出现 first和last获取到的是相同的对象
    - 显式，手动写排序规则

- 默认情况下可以正常从QuerySet中获取
- 隐藏bug
  - 可能会出现 first和last获取到的是相同的对象
    - 显式，手动写排序规则

切片

- 和python中的切片不太一样
- QuerySet[5:15] 获取第五条到第十五条数据
  - 相当于SQL中limit和offset

缓存集

- filter
- exclude
- all
- 都不会真正的去查询数据库
- 只有我们在迭代结果集，或者获取单个对象属性的时候，它才会去查询数据库
- 懒查询

- 懒查询
  - 为了优化我们结构和查询

查询条件

- 属性__运算符=值
- gt
- lt
- gte
- lte
- in 在某一集合中

忽略大小写

Attention! USE_TZ(use time zone) which is in "setting" file is complicated to handle especially in China! So we turned it off.

使用aggregate()函数返回聚合函数的值
Avg：平均值
Count：数量
Max：最大
Min：最小
Sum：求和

```
Student.objects().aggregate(Max('sage'))
```

F对象

可以使用模型的A属性与B属性进行比较

```
grades = Grade.objects.filter(ggirlnum__gt=F('gboynum') )
```

F对象支持算数运算

```
grades = Grade.objects.filter(ggirlnum__gt=F('gboynum') +10 )
```

F

- 可以获取我们属性的值
- 可以实现一个模型的不同属性的运算操作
- 还可以支持算术运算

Q

- 可以对条件进行封装
- 封装之后，可以支持逻辑运算
  - 与 & and
  - 或 | or
  - 非 ~ not

模型成员

类属性

显性: 自己写的那些

隐性: objects 是一个Manager类型的一个对象，作用于数据库进行交互

当模型类没有指定管理器的时候，Django会自动为我们创建模型管理器

当然我们也可以自定义管理器，

```
class Student(models.Model):
    stuManager = models.Manager()
```

当自定义模型管理器时，objects就不存在了，Django就不会为我们自动生成模型管理器

模型成员

- 显性属性
  - 开发者手动书写的属性
- 隐性属性
  - 开发者没有书写，ORM自动生成的

自定义管理器类

模型管理器是Django的模型与数据库进行交互的接口，一个模型可以有多个模型管理器

自定义模型管理器作用:

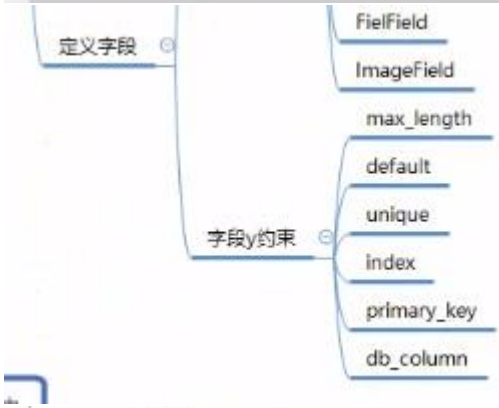
可以向管理器中添加额外的方法

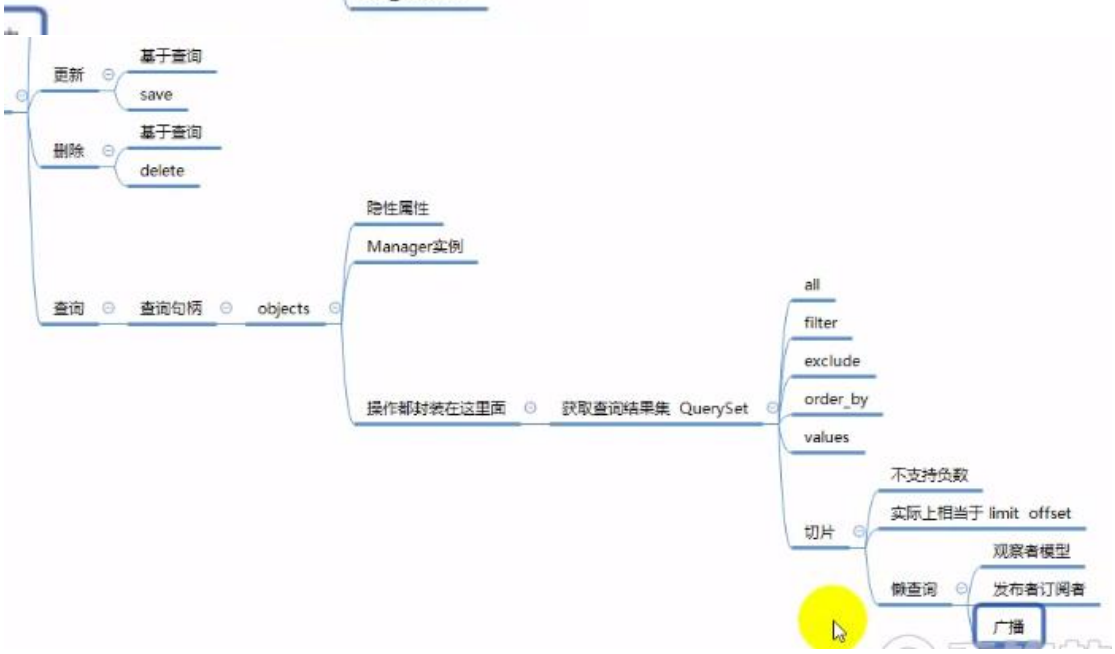
修改管理器返回的原始查询集

提供创建对象的方式

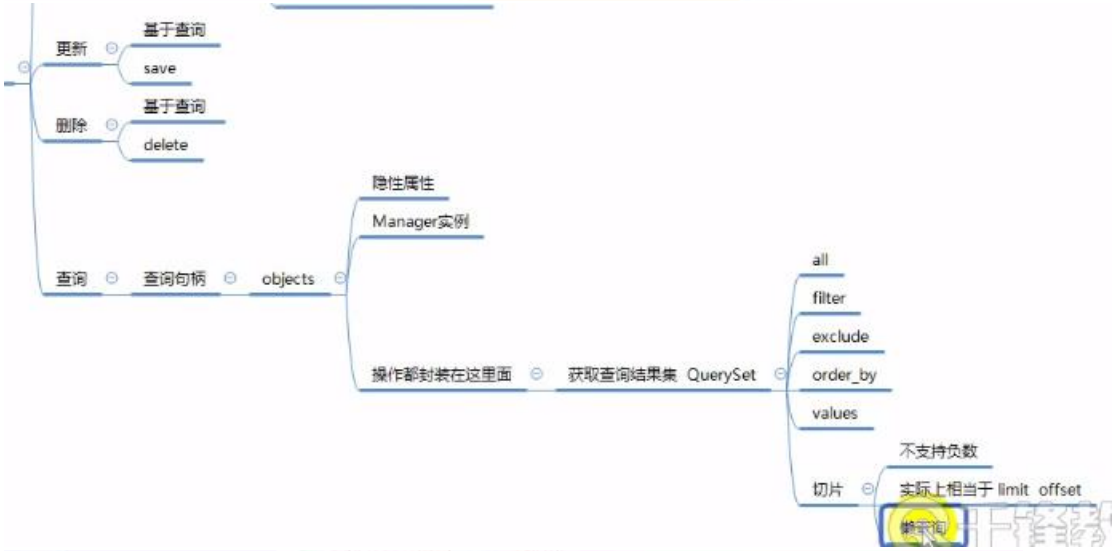
```
class StudentManager(models.Manager):
    def get_queryset(self):
        return super(StudentManager,self).get_queryset.filter(isDelete=False)

    def createStudent(self):
        stu = self.model()
        # 设置属性
        return stu
```

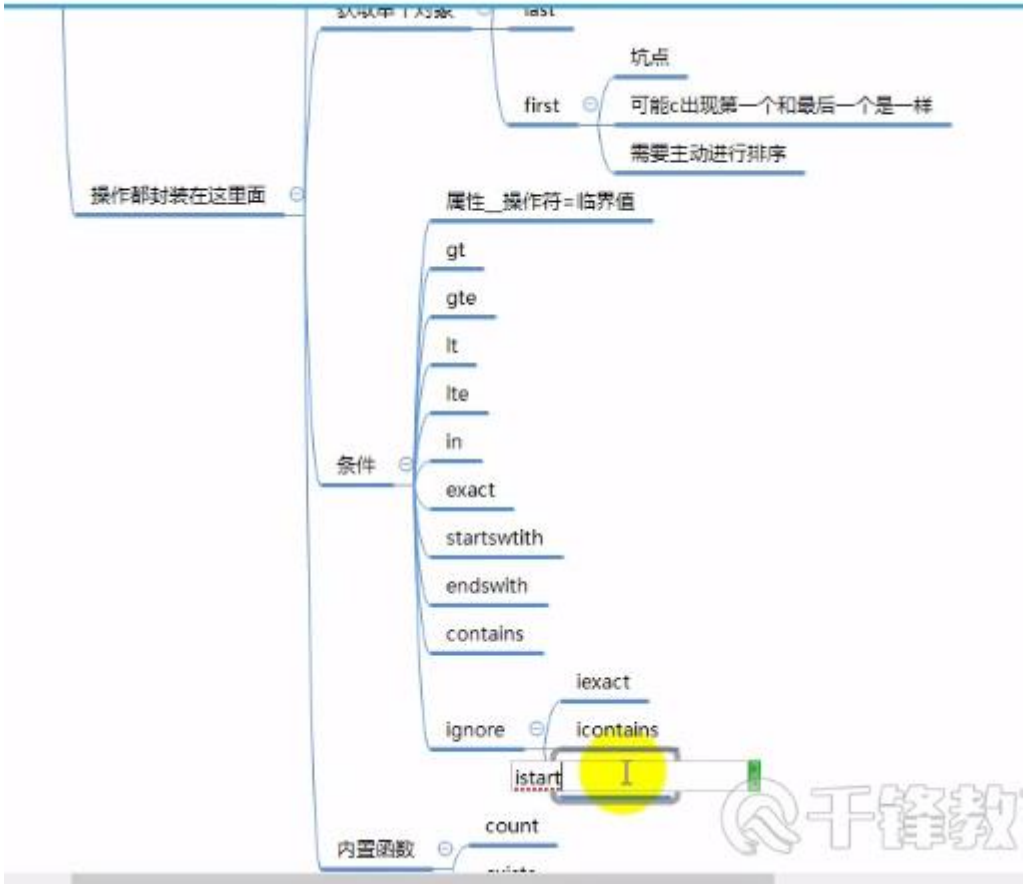

class Meta: 元信息
继承自Model
db_table 表名
定义模型
CharField
TextField
IntegerField
FloatField
BooleanField
字段类型
DecimalField
NullBooleanrField
AutoField
FielField
定义字段


FielField
定义字段
ImageField
max_length
default
unique
字段y约束
index
primary_key
db_column


更新
基于查询
save
删除
基于查询
delete
隐性属性
Manager实例
查询
查询句柄
objects
all
filter
exclude
order_by
操作都封装在这里面
获取查询结果集 QuerySet
values
不支持负数
切片
实际上相当于 limit offset
观察者模型
做查询
发布者订阅者
广播


CRUD
CREATE
save
创建对象
直接实例化对象，设置属性
创建对象，传入属性
使用 Model.objects.create()
自己封装类方法创建
在Manager中封装方法创建


更新
基于查询
save
删除
基于查询
delete
查询
查询句柄
objects
隐性属性
Manager实例
操作都封装在这里面
获取查询结果集 QuerySet
all
filter
exclude
order_by
values
切片
不支持负数
实际上相当于 limit offset

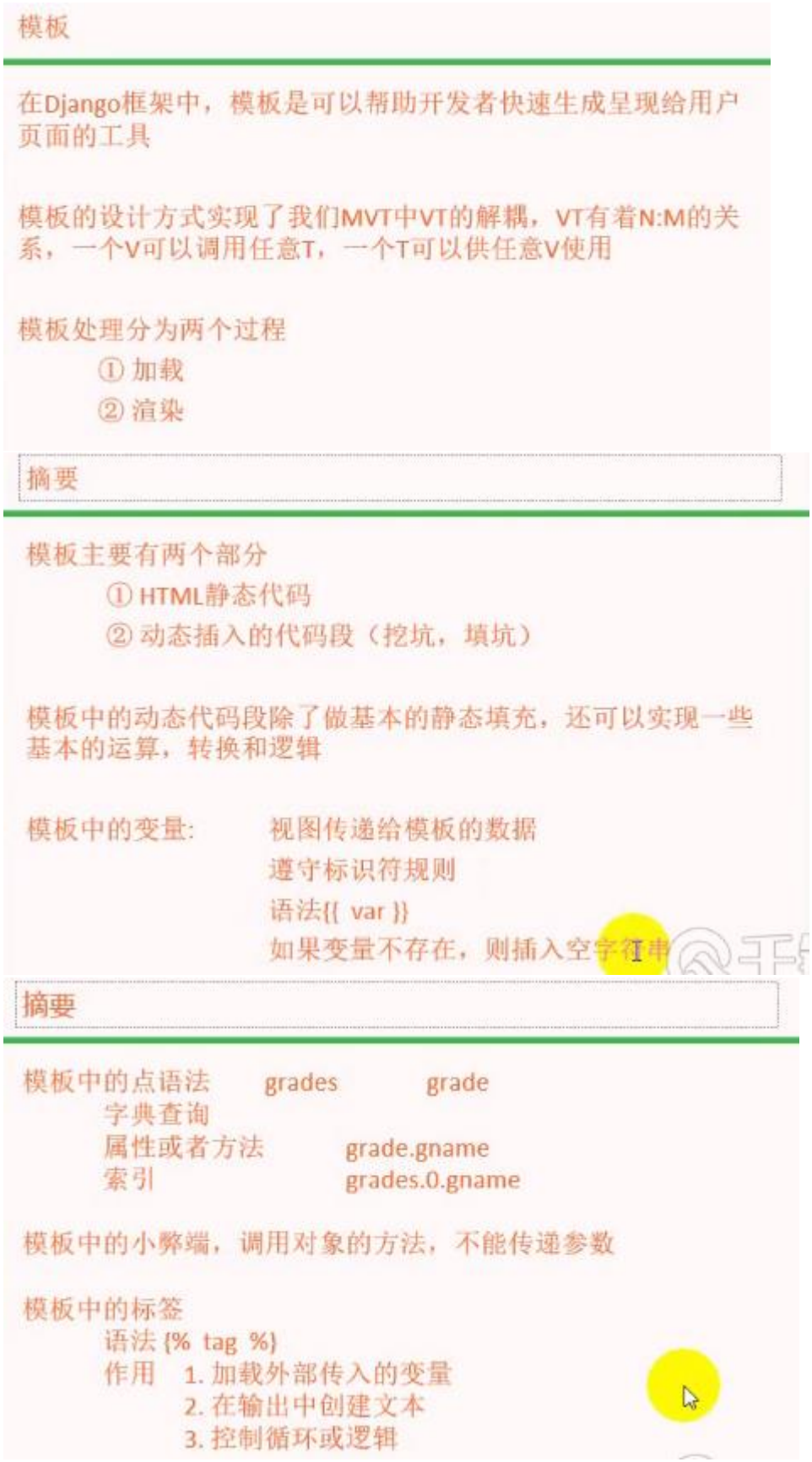

操作都封装在这里面
first
坑点
可能c出现第一个和最后一个是一样
需要主动进行排序
属性__操作符=临界值
条件
gt
gte
lt
lte
in
exact
startswtith
endswith
contains
ignore
iexact
icontains
istart
内置函数
count


获取字段信息
F
通常用在模型的自我属性比较
支持算术运算
条件升级
条件封装
Q
与或非
&
支持逻辑运算
|
~
内置函数
count
exists


显性属性
开发者手动声明的属性
属性
没有声明，父类中也不存在，动态产生出来的
隐性属性
如果开发者主动声明了这些属性，隐性属性自己就不再生成了
逻辑字段
重要数据处理
is_delete
自定义Manager实现统一封装
重写get_queryset


exists
操作列
Max
聚合函数
Avg
Min
Sum
Count

模板 template


模板
在Django框架中，模板是可以帮助开发者快速生成呈现给用户页面的工具
模板的设计方式实现了我们MVT中VT的解耦，VT有着N:M的关系，一个V可以调用任意T，一个T可以供任意V使用
模板处理分为两个过程
① 加载
② 渲染
摘要
模板主要有两个部分
① HTML静态代码
② 动态插入的代码段（挖坑，填坑）
模板中的动态代码段除了做基本的静态填充，还可以实现一些基本的运算，转换和逻辑
模板中的变量:
视图传递给模板的数据
遵守标识符规则
语法{{ var }}
如果变量不存在，则插入空字符串
摘要
模板中的点语法
grades
grade
字典查询
属性或者方法
grade.gname
索引
grades.0.gname
模板中的小弊端，调用对象的方法，不能传递参数
模板中的标签
语法{% tag %}
作用
1. 加载外部传入的变量
2. 在输出中创建文本
3. 控制循环或逻辑

if

格式: {% if 表达式 %}
语句
{% endif %}

{% if 表达式 %}
语句
{% else %}
语句
{% endif %}
{% if 表达式 %}
语句
{% elif 表达式 %}
语句
{% endif %}

for

{% for 变量 in 列表 %}
语句1
{% empty %}
语句2
{% endfor %}

当列表为空或不存在时,执行empty之后的语句
{{ forloop.counter }} 表示当前是第几次循环，从1数数
{{ forloop.counter0}}表示当前是第几次循环，从0数数
{{ forloop.revcounter}}表示当前是第几次循环，倒着数数，到1停
{{ forloop.revcounter0}}表示当前第几次循环，倒着数，到0停
{{ forloop.first }} 是否是第一个 布尔值
{{ forloop.last }} 是否是最后一个 布尔值

```
{% comment %}
    <ul>
        <li>xxx</li>
    </ul>
{% endcomment %}
```

注释

单行注释

{# 被注释掉的内容 #}

多行注释

{% comment %}

内容

{% endcomment %}

乘除

{% widthratio 数 分母 分子 %}

整除 {% if num|divisibleby:2 %}

ifequal 如果相等

{% ifequal value1 value2 %}

语句

{% endifequal %}

ifnotequal 如果不相等

url: 反向解析

{% url 'namespace:name' p1 p2 %}

csrf_token 用于跨站请求伪造保护的

格式 {% csrf_token %}

摘要

过滤器: {{ var|过滤器 }}

作用：在变量显示前修改

add {{ p.page | add : 5 }}

没有减法过滤器，但是加法里可以加负数

{{ p.page | add : -5}}

lower

{{ p.pname|lower }}

upper

摘要

过滤器可以传递参数，参数需要使用引号引起来
比如join：　　{{ students|join '=' }}

默认值:default，格式 {{var|default value}}
如果变量没有被提供或者为False，空，会使用默认值

根据指定格式转换日期为字符串，处理时间的
就是针对date进行的转换
　　{{ dateVal | date:'y-m-d' }}

HTML转义
　　将接收到的数据当成普通字符串处理还是当成HTML代码来渲染的一个问题

渲染成html:{{ code|safe}}

```
{% autoescape off%}
　　code
{% endautoescape %}
```

不想渲染

```
{% autoescape on%}
　　code
{% endautoescape %}
```

CSRF

跨站请求伪造
　　某些恶意网站包含链接，表单，按钮，js利用登陆用户在浏览器中的认证信息，进行非法操作，攻击服务，破坏数据

在表单中添加
{% csrf_token %}

在settings中的中间件MIDDLEWARE中配置打开
　　'django.middleware.csrf.CsrfViewMiddleware',

模板继承

模板也可以继承

关键字block:挖坑

```
{% block XXX%}
    code
{% endblock %}
```

extends 继承，写在开头位置

```
{% extends '父模板路径' %}
```

include: 加载模板进行渲染

格式{% include '模板文件' %}

结构标签

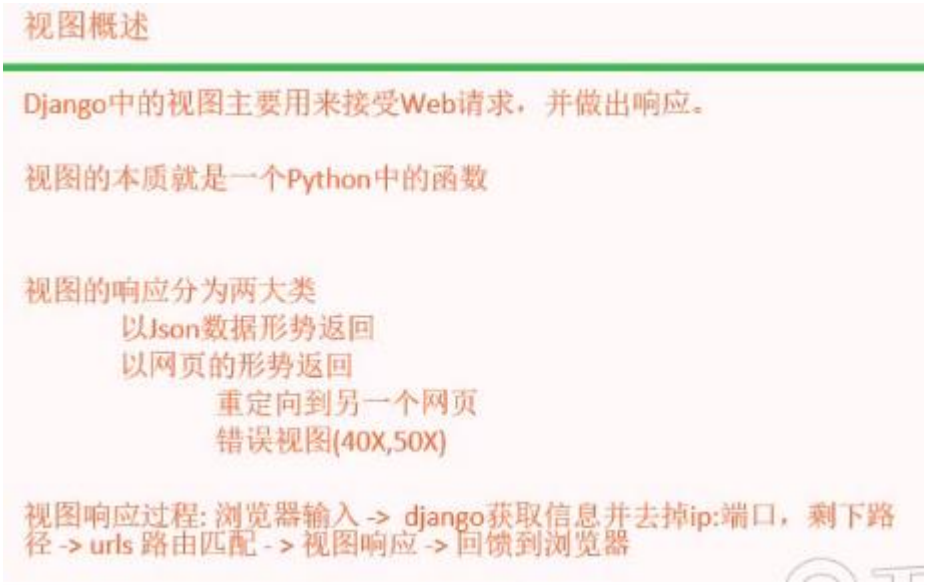
- block
  - 块
  - 用来规划我们的布局 （挖坑）
  - 首次出现，代表规划
  - 第二次出现，代表填充以前的规划
  - 第三次出现， 代表填充以前的规划，默认动作是覆盖
    - 如果不想覆盖，可以添加 {{ block.super }}
    - 这样就实现了增量式操作
- extends
  - 继承
  - 可以获取父模板中的所有结构
- block + entends
  - 化整为零

- include
  - 包含
  - 可以将页面作为一部分，嵌入到其它页面中
- include + block
  - 由零聚一
- 三个标签也可以混合使用
- 能用block + extends搞定的 就尽量不要使用include

静态资源

- 动静分离
- 创建静态文件夹
- 在settings中注册 STATICFILES_DIRS=[]
- 在模板中使用
  - 先加载静态资源 {% load static %}
  - 使用 {% static 'xxx' %} xxx相对路径
- 坑点
  - 仅在debug模式可以使用
  - 以后需要自己单独处理

视图 view



urls

- 路由器
  - 按照列表的书写顺序进行匹配的
  - 从上到下匹配，没有最优匹配的概念
- 路由规则编写
  - 我们通常直接指定以 ^ 开头
  - 在结尾处直接添加反斜线
- 路由路径中的参数使用 () 进行获取
  - 一个圆括号对应视图函数中的一个参数

知识点

- locals
  - 内置函数
  - 将局部变量，使用字典的方式进行打包
  - key是变量名，value是 变量中存储的数据

获取url路径上的参数

如果需要从url中获取一个值，需要对正则加小括号

```
url(r'^grade/(\d+)$',views.getStudents),
```

注意，url匹配中添加了 () 取参，在请求调用的函数中必须接收

```
def getStudents(request,classId):
```

如果需要获取url路径中的多个参数，那就添加多个括号，默认按照顺序匹配路径名字

```
url(r'^news/(\d{4})/(\d)+/(\d+)$',views.getNews),
```

匹配年月日

```
def getNews(request,year,month,day):
```

参数也可以使用关键字参数形势

```
url(r'^news/(?P<year>\d{4})/(?P<month>\d)+/(?P<day>\d+)$',views.getNews),
```

对 current_datetime 的一次赋值操作：

```
1  def current_datetime(request):
2    now = datetime.datetime.now()
3    return render_to_response('current_datetime.html', {'current_date': now})
```

很多时候，就像在这个范例中那样，你发现自己一直在计算某个变量，保存结果到变量中（比如前面代码中的 now ），然后将这些变量发送给模板。 尤其喜欢偷懒的程序员应该注意到了，不断地为临时变量和临时模板命名有那么一点点多余。 不仅多余，而且需要额外的输入。

如果你是个喜欢偷懒的程序员并想让代码看起来更加简明，可以利用 Python 的内建函数 locals() 。它返回的字典对所有局部变量的名称与值进行映射。 因此，前面的视图可以重写成下面这个样子：

```
1  def current_datetime(request):
2    current_date = datetime.datetime.now()
3    return render_to_response('current_datetime.html', locals())
```

url反向解析

url反向解析:
在根urls中
url(r'^views/', include('ViewsLearn.urls',namespace='view')),
在子urls中
url(r'^hello/(\d+)',views.hello,name='sayhello'),
在模板中使用
Hello
year 的位置如果不指定名称按顺序算，指定名称按=算
在视图中使用
HttpResponseRedirect(reverse('view:sayhello',kwargs={}))
kwargs是字典

使用反向解析优点
如果在视图，模板中使用硬编码连接，在url配置发生改变时，需要变更的代码会非常多，这样导致我们的代码结构不是很容易维护，使用反向解析可以提高我们代码的扩展性和可维护性。

- 反向解析
  - 根据根路由中注册的namespace和在子路由中注册name，这两个参数来动态获取我们的路径
  - 在模板中使用 {% url 'namespace:name' %}
  - 如果带有位置参数 {% url 'namespace:name' value1 value2 [valuen...] %}
  - 如果带有关键字参数 {% url 'namespace:name' key1=value1 key2=value2 [keyn=valuen....] %}

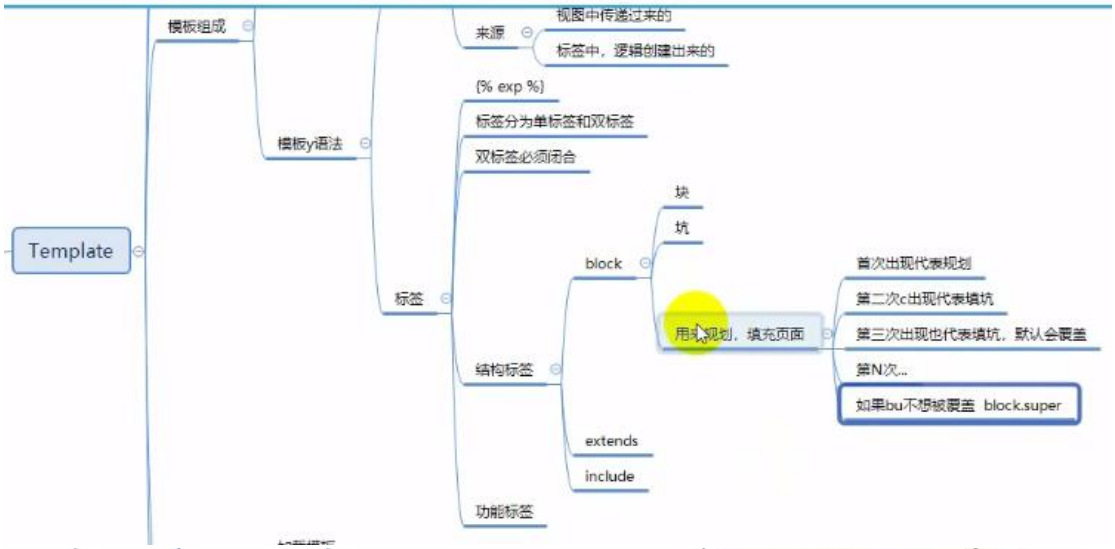
HttpRequest

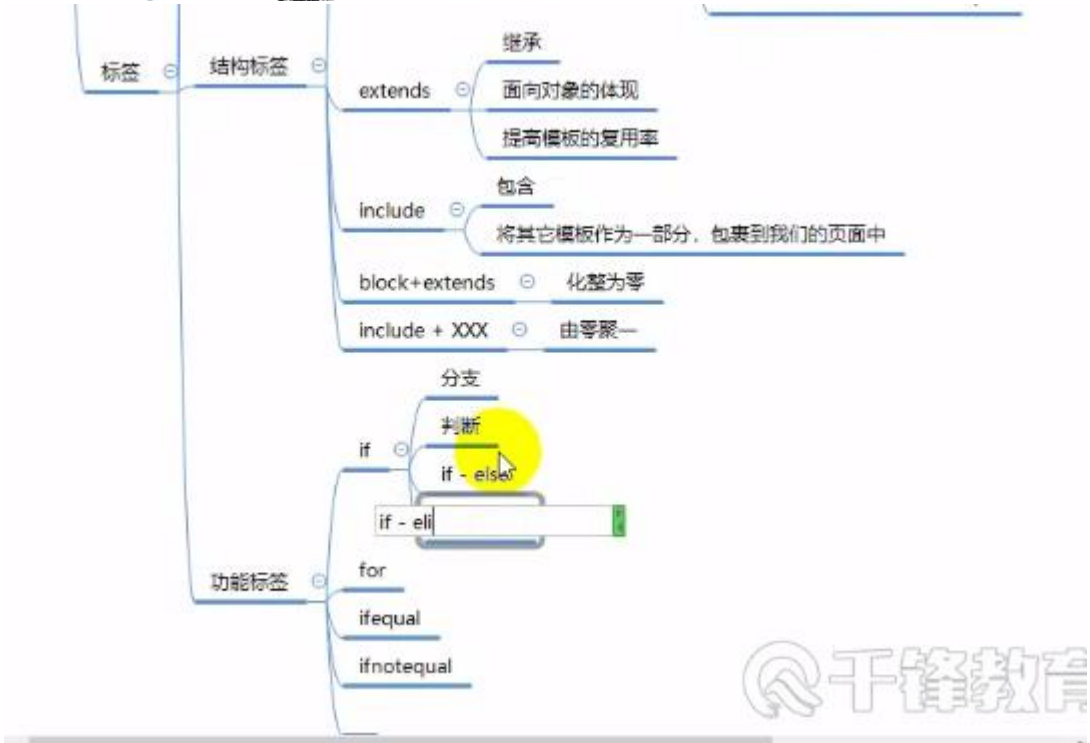
服务器在接收到Http请求后，会根据报文创建HttpRequest对象

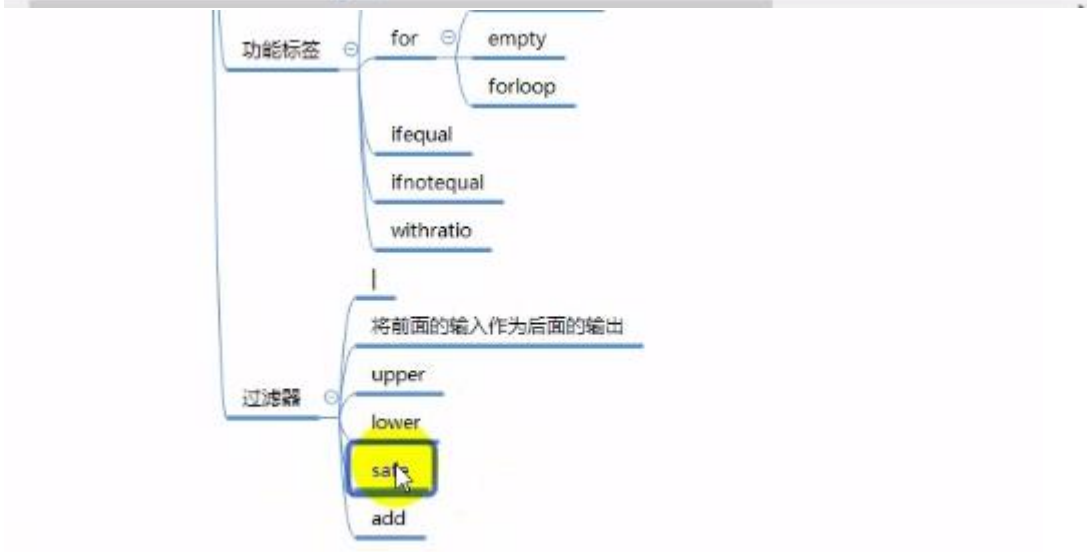
视图中的第一个参数就是HttpRequest对象

Django框架会进行自己的包装，之后传递给视图

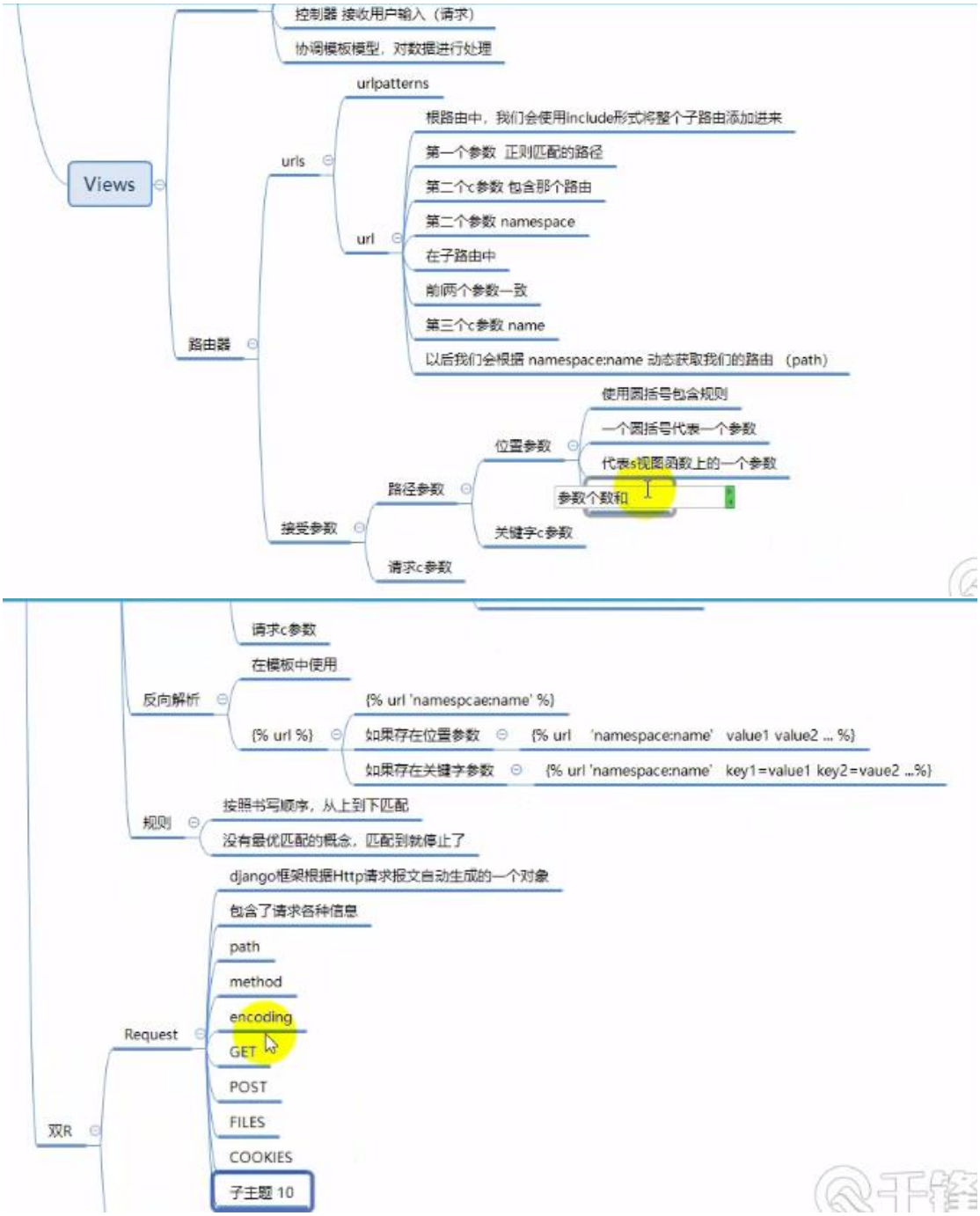
属性:	path	请求的完整路径
	method	请求的方法，常用GET,POST
	encoding	编码方式，常用utf-8
	GET	类似字典的参数，包含了get的所有参数
	POST	类似字典的参数，包含了post所有参数
	FILES	类似字典的参数，包含了上传的文件
	COOKIES	字典，包含了所有COOKIE
	session	类似字典，表示会话
方法:	is_ajax()	判断是否是ajax()，通常用在移动端和JS中


Template
模板组成
来源
视图中传递过来的
标签中，逻辑创建出来的
模板y语法
{% exp %}
标签分为单标签和双标签
双标签必须闭合
标签
结构标签
block
块
坑
用来规划，填充页面
首次出现代表规划
第二次c出现代表填坑
第三次出现也代表填坑，默认会覆盖
第N次...
如果bu不想被覆盖 block.super
extends
include
功能标签


标签
结构标签
extends
继承
面向对象的体现
提高模板的复用率
include
包含
将其它模板作为一部分，包裹到我们的页面中
block+extends
化整为零
include + XXX
由零聚一
功能标签
if
分支
判断
if - else
if - eli
for
ifequal
ifnotequal


功能标签
for
empty
forloop
ifequal
ifnotequal
withratio
过滤器
将前面的输入作为后面的输出
upper
lower
add

模板


Views
控制器 接收用户输入（请求）
协调模板模型，对数据进行处理
路由器
urls
urlpatterns
根路由中，我们会使用include形式将整个子路由添加进来
url
第一个参数 正则匹配的路径
第二个c参数 包含那个路由
第二个参数 namespace
在子路由中
前俩个参数一致
第三个c参数 name
以后我们会根据 namespace:name 动态获取我们的路由 （path）
接受参数
路径参数
位置参数
使用圆括号包含规则
一个圆括号代表一个参数
代表s视图函数上的一个参数
参数个数和
关键字c参数
请求c参数
请求c参数
反向解析
在模板中使用
{% url %}
{% url 'namespcae:name' %}
如果存在位置参数
{% url 'namespace:name' value1 value2 ... %}
如果存在关键字参数
{% url 'namespace:name' key1=value1 key2=vaue2 ...%}
规则
按照书写顺序，从上到下匹配
没有最优匹配的概念，匹配到就停止了
双R
Request
django框架根据Http请求报文自动生成的一个对象
包含了请求各种信息
path
method
encoding
GET
POST
FILES
COOKIES
子主题 10

HttpResponse

服务器返回给客户端的数据

HttpResponse由程序猿自己创建

1. 不使用模板，直接HttpResponse()
2. 调用模板，进行渲染
   - 2.1 先load模板，再渲染
   - 2.2 直接使用render一步到位

render(request,template_name[,context])

request	请求体对象
template_name	模板路径
context	字典参数，用来填坑

HttpResponse

属性:

content	返回的内容
charset	编码格式
status_code	响应状态码(200,3xx,404,5xx)
content-type	MIME类型

方法

init	初始化内容
write(xxx)	直接写出文本
flush()	冲刷缓冲区
set_cookie(key,value='xxx',max_age=None,exprise=None)	
delete_cookie(key)	删除cookie，上面那个是设置

```python
def get_ticket(request):

    # if random.randrange(10) > 5:
    #     return HttpResponseRedirect('/app/hello/')
    #
    # return HttpResponse("恭喜您抢到十一期家特票")

    url = reverse('app:hello')

    return HttpResponseRedirect(url)
```

HttpResponse子类

HttpResponseRedirect
响应重定向:可以实现服务器内部跳转
return HttpResponseRedict('/grade/2017')
使用的时候推荐使用反向解析

JsonResponse
返回Json数据的请求，通常用在异步请求上
JsonResponse（dict）
也可以使用__init__（self.data）设置数据

Content-type是application/json

Json

- JsonObject
  - { }
  - key - value
- JsonArray
  - [ ]
  - 列表中可以是普通数据类型，也可以是JsonObject
- JsonObject和JsonArray可以嵌套
- 给移动端的 Json
- 给Ajax
  - 前后端
  - DRF
- Google Chrome
  - JsonFomatter
  - JsonView

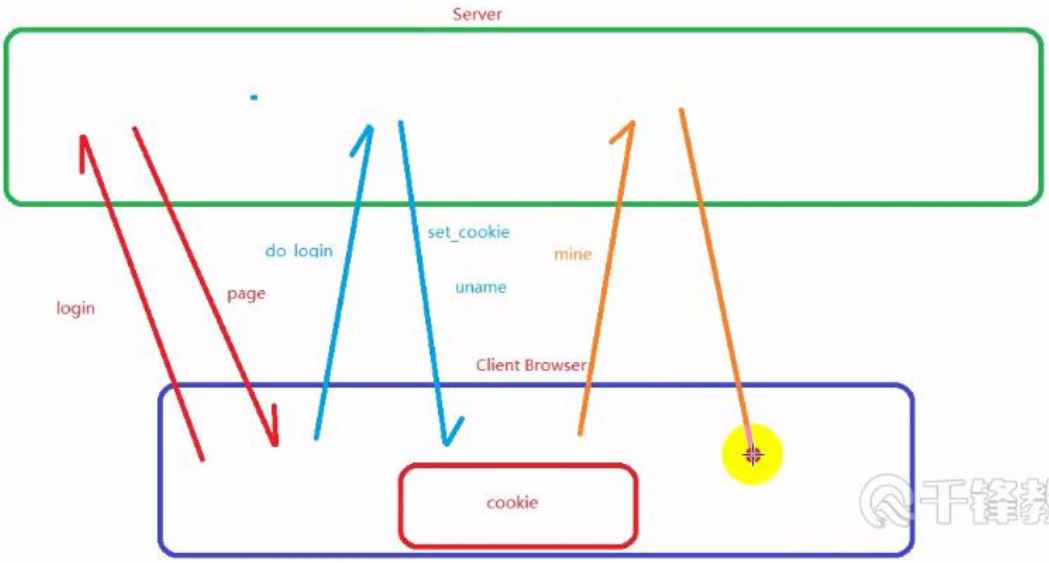
HttpResponse

- HttpResponseRedirect
  - 重定向，暂时
  - 302
  - 简写 redirect
- JsonResponse
  - 以Json格式返回数据
  - 重写了__init__，序列化Json数据，指定content_type 为application/json
- HttpResponsePermanentRedirect
  - 重定向，永久性
  - 301
- HttpResponseBadRequest
  - 400
- HttpResponseNotFound
  - 404
- HttpResponseForbidden

会话技术

- 出现场景
  - 服务器如何识别客户端
  - Http在Web开发中基本都是短连接
- 请求生命周期
  - 从Request开始
  - 到Response结束
- 种类
  - Cookie
  - Session
  - Token

- 种类
  - Cookie
    - 客户端会话技术
      - 数据存储在客户端
    - 键值对存储
    - 支持过期时间
    - 默认Cookie会自动携带，本网站所有Cookie
    - Cookie跨域名，跨网站
    - 通过HttpResponse
    - Cookie默认不支持中文
    - 可以加盐
      - 加密
      - 获取的时候需要解密
  - Session
  - Token



COOKIE

浏览器端的会话技术

cookie本身由浏览器生成，通过Response将cookie写到浏览器上，下一次访问，浏览器会根据不同的规则携带cookie过来

Response.set_cookie(key,value[,max_age=None,exprise=None)]

request.GET.get(key,defaultvalue)

cookie不能跨浏览器

COOKIE

response.set_cookie(key,value,max_age=None,exprise=None)

max_age: 整数，指定cookie过期时间
expries : 整数，指定过期时间，还支持是一个datetime或timedelta，可以指定一个具体日期时间
max_age和expries两个选一个指定

过期时间的几个关键时间
max_age 设置为 0 浏览器关闭失效
设置为None永不过期
expires=timedelta(days=10) 10天后过期