# Meta类属性汇总：

class Meta:

abstract = … : True or False, 标识本类是否为抽象基类

app\_label = … : 定义本类所属的应用，比如 app\_label=’myapp’

db\_table = … : 映射的数据表名， 比如 db\_table = ‘comments’

默认为app\_comments

db\_tablespace= …:映射的表空间名称。

default\_related\_name = …: 定义本模型的反向关系引用名称，默认与模型名一致。

get\_latest\_by = … : 定义按哪个字段排列以获得模型的开始和结束的记录，本属性通常指向一个日期或整数的模型字段

managed = … : True or False, 定义为Django的manage.py 命令行工具是否管理本模型。默认为True；若设为False，则运行python manage.py migrate 时将不会在数据库中生成本模型的数据表，所以需要手工维护数据库的定义。

order\_with\_respect\_to=… : 定义本模型可以按照某外键引用的关系排序。

ordering=… :本模型记录的默认排序字段，可以设置多个字段，默认以升序排列，如果以降序排列则需要在字段名前加“负号”。

class Meta:

ordering = [‘user\_name’, ‘-pub\_date’]

按user\_name升序和pub\_date降序排列

default\_permissions=…:模型的操作权限

默认为default\_permissions=(‘add’, ‘change’, ‘delete’)

proxy: True or False, 本模型及继承自本模型的子模型是否为代理模型

required\_db\_features:

required\_db\_vendor:

unique\_together:用来设置不重复的字段组合，必须唯一（可以将多个字段做联合唯一）

class Meta:

unique\_together = ((“user\_name”, “pub\_date”),)

定义每个user\_name在同一个pub\_date中只能有一条数据表记录

可以设置多个这样的唯一约束

index\_together:定义联合索引的字段，可以设置多个。

class Meta:

index\_together = [[“pub\_date”, “deadline”],]

verbose\_name:

verbose\_name\_plural:

# 常用字段参数

null: 默认为False，定义是否允许相应的数据库字段为null  
blank: 定义字段是否为空。

null与blank的区别：

null是一个数据库中的非空约束；

blank用于字段的HTML表单验证，即判断用户是否可以不输入数据

choices：定义字段为可选值

LEVEL = (

(‘1’, ‘Very good’),

(‘2’, ‘Good’),

)

在数据库中实际存储的是第1个值，而在HTML页面中显示的是第2个值

default: 设定默认值

help\_text: HTML页面中输入控件的帮助字符串

primary\_key: 定义字段是否为主键， 为True或false

unique: 是否为字段定义数据库的唯一约束。

on\_delete=models.CASCADE 对 OneToOneField和ForeignKey两字段有效。当被关联模型的记录被删除时，本模型的记录也被删除。

class Contact:

account = OneToOneField(“Account”, on\_delete=models.CASCADE)

….

a1 = Account(…)

a1.save()

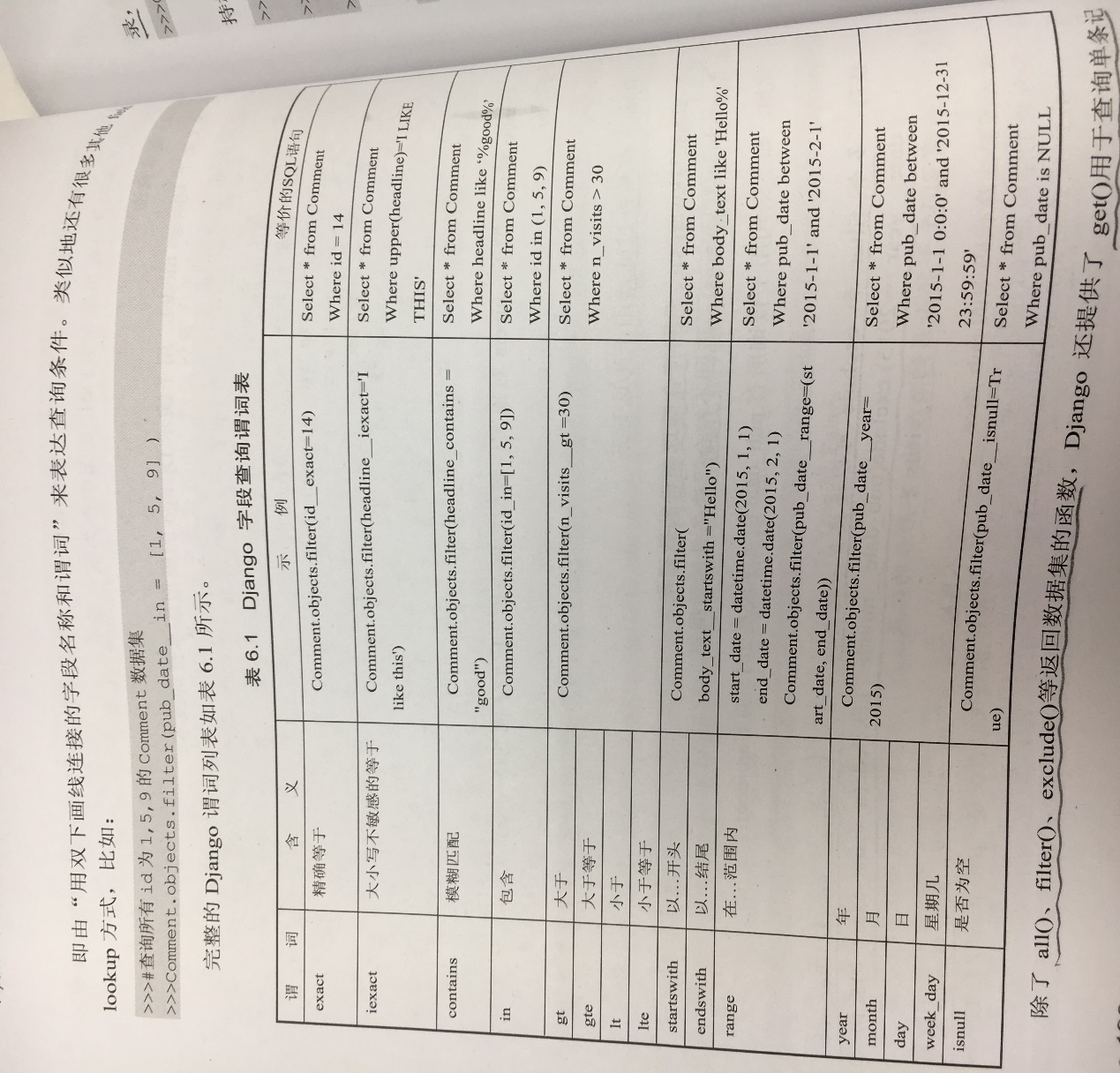
c1 = Contact(account=a1)

c1.save()

a1.delete()#此时a1与c1同时被删除

ForeignKey同理

# Django字段查询谓词表



Model.objects.get(…)

Model.objects.filter(…)

Model.objects.all(…)

Model.objects.order\_by(…)

Model.objects.exclude(…)

关系操作

## 一对一关系：OneToOneField

account = models.OneToOneField(“Account”, on\_delete=models.CASCADE,primary\_key=True)

on\_delete:参数定义，当被关联模型Account的记录被删除时，本模型的记录如何处理

models.CASCADE:用于定义本记录也被删除

a1=Account(..)

a1.save()

c1=Contatct(account=a1,…)

c1.save()

a1.contact 等价于 c1

c1.account 等价于 a1

## 一对多关系：ForeignKey

class Contact：

account = ForeignKey(‘Account’, on\_delete=models.CASCADE)

a1=Account(..)

a1.save()

c1=Contatct(account=a1,…)

c1.save()

c2=Contatct(account=a1,…)

c2.save()

c1.account 与 c2.account 等价于 a1

a1.contact\_set.all() 返回c1与c2

a1.contact\_set.count() 返回2

a1.delete() c1与c2同时删除

## 多对多关系：ManyToManyField

class Contact：

accounts = ManyToManyField(‘Account’)

a1 = Account()

a1.save()

c1 = Contact(….)

c1.save()

c1.accounts.add(a1)

a2 = Account()

a2.save()

a2.contact\_set.add(c1)

# 面向对象ORM

## 1．抽象类继承

在多个表有若干相同的字段时，可以使开发者将这些字段统一定义在抽象基类中。数据库中不会有MessageBase表

class MessageBase(models.Model):

….

class Meta:

abstract = True

class Moments(MessageBase):

….

## 2．多表继承

class MessageBase(models.Model):

….

class Moments(MessageBase):

….

## 3. 代理模型继承

继承子类只用于管理父类的数据，而不实际存储数据。

通过在子类的Meta中定义proxy=True属性来实现。

# URL映射

## URL路径用正则表达式定义

‘^’: 以。。。为开始

‘$’: 以。。。为结束

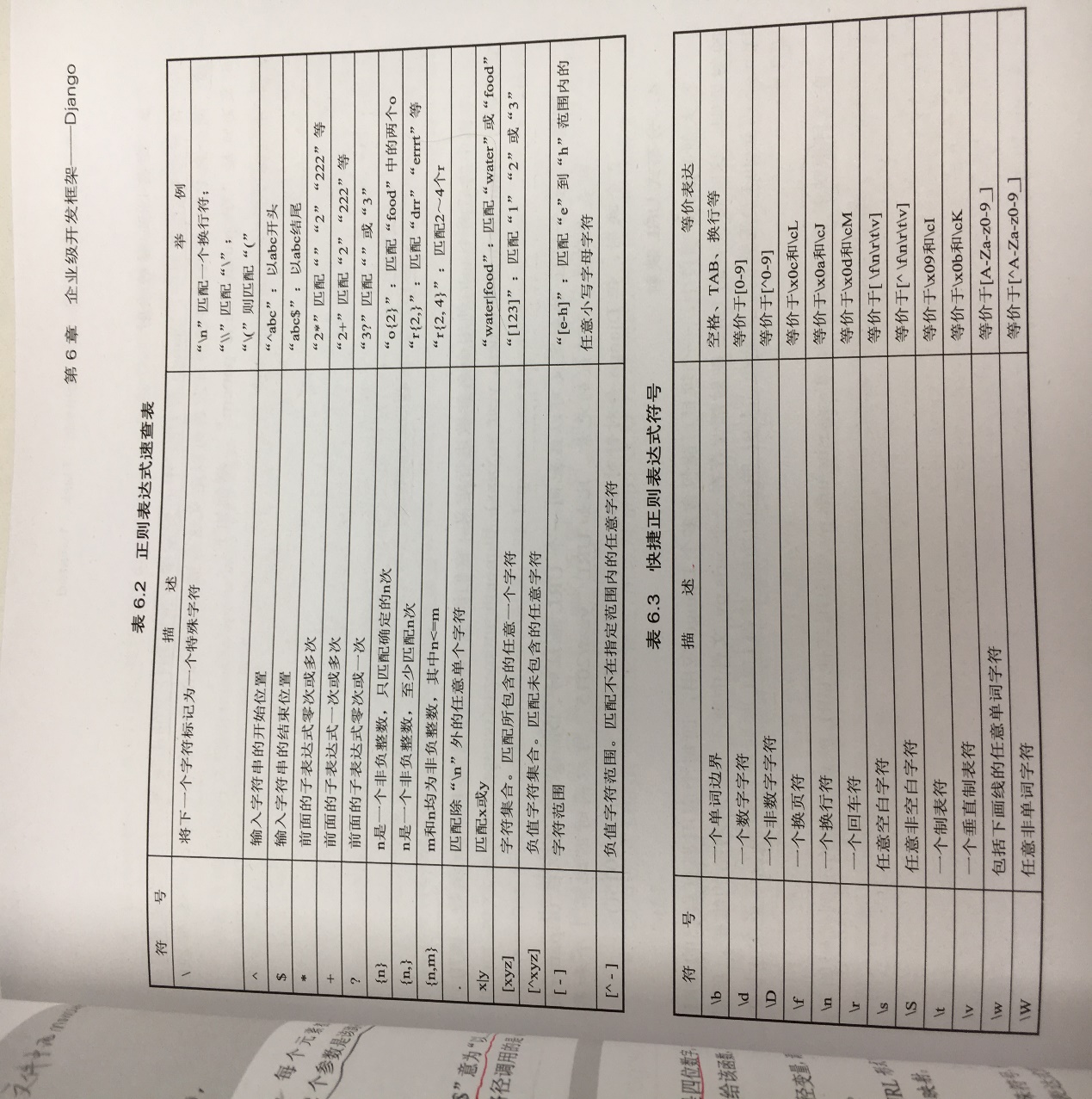
Eg: url(r’^/year/2015/$’, views.moments\_2015)

url(r’^/year/[0-9]{4}/$’, views.year\_moments) 匹配路径是“year/xxxx/”，其中要求xxxx是4位数字，并且把四位数字作为变量传递给该函数调用参数形式是year\_moments(request,xxxx)

url(r’^/year/[0-9]{4}/[0-9]{2}/$’, views.month\_moments) : month\_moments(request, xxxx, yy)

url(r’^/year/[0-9]{4}/[0-9]{2}/([0-9]+)$’, views.single) : single(request, xxxx, yy, zz)

## 正则表达式



## 命名URL参数映射

“?P<param\_name>pattern”

Eg: url(r‘^year/?P<year>([0-9]{4})/$’, views.year\_moments) : year\_moments(request, year=xxxx)

## 分布式URL映射

include 的使用

eg: url(r’^moments/’, include(‘djangosite.app.urls’))

以moments开头的URL被转接到djangosite.app.urls包中

## 反向解析

利用映射名代替很多需要些绝对URL路径的地方，提高了代码的可维护性。

模板文件（HTML）中用{% url %}标签来反向解析

Python程序中利用django.core.urlresolvers.reverse()函数调用反向解析

Eg：

url(r‘^year/2015/$’, views.year\_moments, name = ‘moments\_2015’),

模板文件中

<a href=”{%url ‘moments\_2015%}”>查看2015年消息</a>

Python代码与模板文件中

return HttpResponseRedirect(reverse(‘moments\_2015’))

## 带参数的反向解析

url(r’^year/?P<year>([0-9]{4}/$)’, views.year\_moments, name = “moments”)

在模板文件中

<a href=”{% url ‘moments’, 2014 %}”>查看2014年信息</a>

Python代码中

Return HttpResponseRedirect(reverse(‘moments’, args=(2014, )))

# 视图函数

## 直接构造HTML页面 HttpResponse

now = datetime.datetime.now()

return HttpResponse(now)

## 用数据渲染HTML模板文件render()

m = Moment.objects.get(id=moment\_id)

return render(request, ‘templetes/moment.html’, {‘headline’: m.headline, ‘user’: m.user\_name})

render()第一个参数是HTTP request,第二个参数是模板文件名，第三个参数是以字典形式表达的模板参数

## 返回HTTP错误

return HttpResponse(status=404)

## HttpResponse子类：

HttpResponseRedirect: 返回Status302，用于URL重定向，需要将重定向的目标地址作为参数传给该类

参数经常用URL反向映射函数reverse()获得，这样可以避免在更新网站url.py内容的时候维护视图函数中的代码

HttpResponseNotModified: 返回Status 304，用于指示浏览器用其上次请求时的缓存结果作为页面内容显示。

HttpResponsePermanentRedirect: 返回Status 301， 与HttpResponseRedirect类似，但是告诉浏览器这是一个永久重定向

HttpResponseBadRequest: 返回Status 400，请求内容错误

HttpResponseForbidden: 返回Status 403，禁止访问错误

HttpResponseNotAllowed: 返回Status 405， 用不允许的方法（Get、POST、Head等）访问本页面

HttpResponseServerError: 返回 Status 500，服务器内部错误，比如无法处理的异常等

# 模板语法

{% extends base.html %} #继承base.html页面

{% block title %} #块内容，用于模板的继承

{{section.title}} #变量的替换，动态内容

{% end block%} #块内容，结束

{% block content%}

{% for moment in moment\_list %} #流程控制——for循环

<h2>

{{moment.headline|upper}} #带过滤器的变量替换

#变量与过滤器通过管道符号“|”连接，表示将过滤器应用在moment.headline中，作用是制定以大写的方式输出moment.headline

</h2>

{% endfor%}

{% endblock %}

## 过滤器

add: 给value加上一个数值，比如{{123| add:”5”}},返回128

addslashes:单引号加上转义号

capfirst:第1个字母大写，比如{{ “good” | capfirst }},返回Good

center:输出指定长度的字符串，把变量居中，比如{{“abcd” | center:”50” }}

cut: 删除指定的字符串，比如 {{ value | cut:”not”}}

date: 格式化日期

default: 如果值不存在，则使用默认值代替， 比如{{ value | default: “(N/A)”}}

default\_if\_none: 如果值为none，则使用默认值代替，方法与default类似

dictsort:按某字段排序，变量必须是一个dictionary

{% for moment in moments | dictsort:”id” %}

\*{{ moment.headline }}

{% endfor %}

dictsortreversed: 按某字段倒序排序，变量必须是一个dictionary

divisibleby: 判断是否可以被某数字整除。

escape：按HTML转意，比如讲“<”转换为“&lt”，将 “>”转换为“&gt”

filesizeformat: 增加数字的可读性，转换结果为13KB、89MB、3Bytes等

first: 返回列表的第1个元素，变量必须是一个列表，比如{{ [‘English’, ‘Chinese’, ‘Japanse’] | first }}

floatformat:转换为制定精度的小数，默认保留1位小数

{{ 34.23234 | floatformat}} #返回34.2

{{ 34.23234 | floatformat: 3}} #返回34.232

{{ 34.23234 | floatformat: 4}} #返回34.2323

get\_digit:从个位数开始截取指定位置的数字，比如{{ 23456 |get\_digit:”1”}}返回6

join: 用指定分隔符连接列表，比如{{ [‘abc’, ‘45’] | join: “\*”}}，返回abc\*45

length:返回列表中元素的个数或字符串的长度

length\_is: 检查列表、字符串的长度是否符合指定的值，比如{{ “hello” | length\_is: “3”}}返回false

line\_breaks: 用<p>或<br />标记包裹变量，其中单独的换行被替换为<br/>，空行前后被分割为<p>

{{ “Hi\n\nDavid” |line\_breaks }} #返回“<p>Hi</p> <p>David</p>”

linebreakfor:用<br/>标记代替换行符

linenumbers:为变量中的每一行加上行号

ljust:输出制定长度的字符串，变量左对齐。比如{{ “ab” |ljust:5 }} 返回“ab ”

lower:字符串变换为小写，比如{{ “ABCD” |lower }}

make\_list: 将字符串转换为列表，比如{{ “abc” |make\_list }}返回 [‘a’, ‘b’, ‘c’]

pluralize:根据数字确定是否输出英文复数符号

you have {{ num\_message }} message{{ num\_message | pluralize}}.

random:返回列表的随机一项

removetags:删除字符串中指定的HTML标记

{{ value| removetags:”h1 h2”}}

当value是“<h1>Good morning</h1><h3>David</h3>”时，输出为”Good morning<h3>David</h3>”

rjust:输出制定长度的字符串，变量右对齐

slice：切片操作，即返回列表、字符串的一部分。比如{{ [3,9,1] | slice:”:2” }}返回[3,9]

slugify: 在字符串中留下减号和下划线，其他符号删除，空格用减号替换

stringformat:字符串格式化，使用python的字符串格式化方法

time: 返回日期的时间部分

timesince：以“到现在为止过了多长时间”的形式显示时间变量，可能结果如45days，3hours等。

timeuntil: 与timesince类似，但是比较的是当前时间与之后的某个时间。

title: 每个单词的首字母大写

truncatewords:将字符串转换为省略表达方式，传入参数表达保留的单词个数，比如

{{ “This is a lovely cat” | truncatewords:”3” }} 返回“This is a …”

truncatewords\_html：与truncatewords类似，但保留其中的HTML标签，比如

{{ “<p>This is a lovely cat</p>” |truncatewords\_html: “3”}返回”<p>This is a … </p>”

upper:转换为全部大写的形式

urlencode: 将字符串中的特殊字符转换为URL兼容的表达方式

{{ “https://www.example.org/foo?a=b&c=d” |urlencode}}

返回“https%3A//www.example.org/foo%3Fa%3Db%26c%3Dd”

urlize: 将变量字符串中的URL由纯文本变为可单击的链接

{{ “点击 [www.django.com](http://www.django.com)” |urlize}}

返回“点击<a href=”www.django.com” rel=”nofollow”>www.django.com</a>”

wordcount:返回变量字符串中的单词数

yesno：将布尔变量变换为字符串yes、no或者maybe，也可以在参数中制定变量的结果。

{{ true|yesno:”Yes,No,Perhaps”}} #返回Yes

{{ false|yesno}} #返回no

{{None| yseno:”Yes,No,Perhaps”}} #返回perhaps

## 2. 流程控制

for:

{% for moment in moments %}

…

{% endfor %}

if:

{% if moment.id <10 %}

…

{% elif moment.id <20 %}

…

{% else %}

…

{% endif%}

## 3. 模板继承

# 使用Django表单

## 表单绑定状态

is\_bound

f = MomentForm()

print(f.is\_bound) #False

f = MomentForm({‘headline’: ‘hello’})

print(f.is\_bound) #True

## 表单数据验证

在服务器端用Python代码验证表单中数据的合法性

* 字段属性验证

通过model中字段的约束完成

is\_valid()函数，在代码中获得表单验证是否通过的信息。用errors获得错误的提示信息

f = MomentForm({‘user\_name’: ‘David’})

print(f.is\_valid) #False

print (f.error) #{‘content’: [‘This field is required’]}

在视图函数开发中的重要作用

def viewer(request):

if request.method == “POST”:

form = XXXForm(request.POST)

if form.is\_valid() :

#此处编写正常的表单提交的业务逻辑

#处理完成后用redirect重定向页面

else:

#此处编写提交数据不完全的业务逻辑，比如显示特定的错误信息等

#用render()函数渲染表单页面

* 自定义逻辑验证

进行表数据验证时判断自定义的复杂逻辑，则可以通过重载Form子类的clean()函数进行定义。修改MomentForm定义如下：

class MomentForm(ModelForm):

class Meta:

model = Moment

fields = '\_\_all\_\_'

def clean(self):

cleaned\_data = super(MomentForm, self).clean()

content = cleaned\_data.get("content")

if content is None:

raise ValidationError("请输入Content内容")

elif content.find("ABCD") >=0:

raise ValidationError("不能输入敏感字ABCD！")

return cleaned\_data

clean()函数在开发者调用Form.is\_valid()函数时自动被Django调用

## 检查变更字段

表单函数has\_changed()来判断用户是否修改过表单的数据

changed\_data中有更改的字段名称

def moments\_input(request):  
 if request.method == 'POST':  
 data = {  
 'content': 'Please input the content',  
 'user\_name': '匿名',  
 'kind': 'Python技术',  
 }  
 form = MomentForm(request.POST, initial=data)  
 if form.has\_changed():  
 print('如下字段进行了修改：')  
 for field in form.changed\_data:  
 print(field)

#返回修改的字段，content、user\_name、kind  
 if form.is\_valid():

#数据处理

# 个性化管理员站点

## 模型

class MomentAdmin(admin.ModelAdmin):  
 empty\_value\_display = "空值"  
  
admin.site.register(Moment, MomentAdmin)

empty\_value\_display定义了模型管理界面中对空值的显示方式

## 常用的管理类属性描述:

date\_hierarchy:设置一个日期类型字段，使其出现在日期导航找模型实例的界面中

empty\_value\_display:设置一个字符串，定义空值的显示方式，还可以按字段配置（python3不可用）

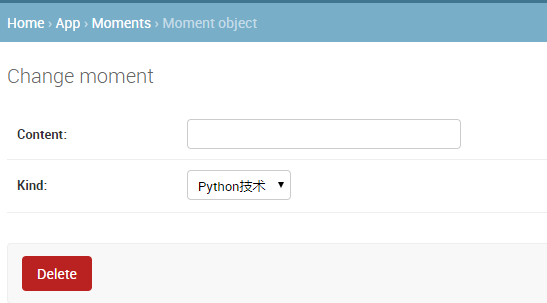
fields和exclude：分别用于设置需要管理的字段和排除管理的字段。对如下模型Moment来说：

models.py

class Moment(models.Model):  
 content = models.CharField(max\_length=300, null=False)  
 user\_name = models.CharField(max\_length=20, default='匿名')  
 kind = models.CharField(max\_length=20, choices=KIND\_CHOICES, default=KIND\_CHOICES[0])

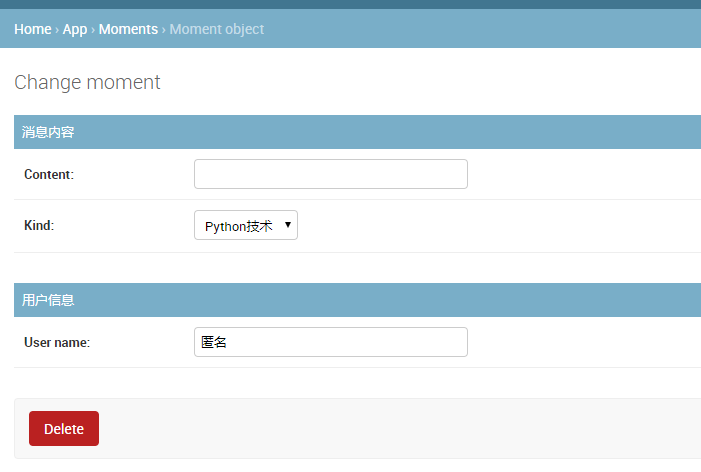
admin.py

class MomentAdmin(admin.ModelAdmin):  
 #需要管理的字段  
 fields = ('content', 'kind',)  
 #排除管理的字段,和上一句功能一样  
 # exclude = ('user\_name',)  
  
admin.site.register(Moment, MomentAdmin)



fieldsets:配置字段分组，美化管理界面

class MomentAdmin(admin.ModelAdmin):  
 fieldsets = (  
 ("消息内容", {  
 'fields': ('content', 'kind')  
 }),  
 ("用户信息", {  
 'fields': ('user\_name',)  
 }),  
 )



list\_editable: 设置字段列表，指定模型中哪些字段可以编辑

必须配合list\_display进行使用。且list\_display中的第一个元素不可在list\_editable中

class MomentAdmin(admin.ModelAdmin):  
 list\_editable = ('kind', 'content')  
 list\_display = ('pk', "content", "user\_name", 'kind',)



list\_per\_page:设置一个整数，指定每一页显示的实例数量，默认为100

search\_fields: 设置字段列表，出现搜索页面

ordering: 设置排序方式