Admin

一、管理站点

A、class ModelAdmin

1、ModelAdmin objects

ModelAdmin类是模型在Admin界面中的表示形式：

```
class AuthorAdmin(admin.ModelAdmin):
    pass

admin.site.register(Author, AuthorAdmin)
```

若对默认的Admin满意,你无需自己定义ModelAdmin对象,可直接注册模型类而无需提供ModelAdmin的描述：

```
admin.site.register(Author)
```

2、注册装饰器

register(*models[, site=django.admin.sites.site])

可以用一个装饰器来注册您的ModelAdmin类：

```
@admin.register(Author)
class AuthorAdmin(admin.ModelAdmin):
    pass
```

如果不使用默认的 AdminSite ,可以提供一个或多个模块类来注册 ModelAdmin 并且一个选择性关键参数 site。

3、ModelAdmin options

ModelAdmin.actions
一个列表，包含自定义的actions
ModelAdmin.actions_on_top
是否在列表上方显示actions的下拉框，默认为True
ModelAdmin.actions_on_bottom
是否在列表下方显示actions的下拉框，默认为False。
ModelAdmin.actions_selection_counter
是否在actions下拉框右侧显示选中的对象的数量，默认为True，可改为False。
ModelAdmin.date_hierarchy

根据你指定的日期相关的字段，为页面创建一个时间导航栏，可通过日期过滤对象。

ModelAdmin.empty_value_display

指定空白所显示的内容。

如果你有些字段没有值（例如None，空字符串等等），默认情况下会显示破折号“-”。这个选项可以让你自定义显示什么，如下例就显示为“-empty-”：

```
class AuthorAdmin(admin.ModelAdmin):
    empty_value_display = '-empty-'
```

ModelAdmin.exclude

不显示指定的某些模型中的字段。

```
class AuthorAdmin(admin.ModelAdmin):
    # 一定注意了，值是个元组！一个元素的时候，最后的逗号不能省略。
    exclude = ('birth_date',)
```

ModelAdmin.fields

按你希望的顺序，显示指定的字段。

与exclude相对。但要注意与list_display区分。这里有个小技巧，你可以通过组合元组的方式，让某些字段在同一行内显示，例如下面的做法“url”和“title”将在一行内，而“content”则在下一行：

```
class FlatPageAdmin(admin.ModelAdmin):
    fields = (('url', 'title'), 'content')
```

ModelAdmin.fieldsets

这个功能其实就是根据字段对页面进行分组显示或布局。

fieldsets是一个二元元组的列表。每个二元元组代表一个fieldset,是整个form的一部分。

二元元组的格式为(name,field_options)，name是一个表示该filedset标题的字符串，field_options是一个包含在该filedset内的字段列表：

```
class FlatPageAdmin(admin.ModelAdmin):
    fieldsets = (
        (None, {
            'fields': ('url', 'title', 'content', 'sites')
        }),
        ('Advanced options', {
            'classes': ('collapse',),
            'fields': ('registration_required', 'template_name'),
        }),
    )
```

两个比较有用的样式是collaspe和wide，前者将fieldsets折叠起来，后者让它具备更宽的水平空间。

ModelAdmin.filter_horizontal
水平扩展多对多字段。
默认情况下，ManyTOManyField在admin的页面中会显示为一个select框。在需要选择大量对象时，这会有点困难。将ManyTOManyField添加到这个属性列表里后，页面就会对字段进行扩展，并提供过滤功能。
ModelAdmin.filter_vertical
与上面的类似，不过是改成垂直布置了。
ModelAdmin.form
默认情况下，admin系统会为你的模型动态的创建ModelForm，它用于创建你的添加/修改页面的表单。我们可以编写自定义的ModelForm，在"添加/修改"页面覆盖默认的表单行为。
注意：如果你的ModelForm和ModelAdmin同时定义了exclude选项，那么ModelAdmin中的具有优先权
ModelAdmin.formfield_overrides
设想一下我们自己写了个RichTextEditorWidget（富文本控件），然后想用它来代替传统的textarea（文本域控件）用于输入大段文字。我们可以这么做：

```
from django.db import models
from django.contrib import admin

# 从对应的目录导入我们先前写好的widget和model
from myapp.widgets import RichTextEditorWidget
from myapp.models import MyModel

class MyModelAdmin(admin.ModelAdmin):
    formfield_overrides = {
        models.TextField: {'widget': RichTextEditorWidget},
    }
```

ModelAdmin.inlines
参考InlineModelAdmin对象，就像ModelAdmin.get_formsets_with_inlines()一样。
ModelAdmin.list_display
指定显示在修改页面上的字段。这是一个很常用也是最重要的技巧之一。
在list_display中，你可以设置四种值：

1. 模型的字段名
2. 一个函数，它接收一个模型实例作为参数
3. 一个表示ModelAdmin的某个属性的字符串
4. 一个表示模型的某个属性的字符串

对list_display属性的一些特别提醒：

1. 对于Foreignkey字段，显示的将是其str()方法的值。
2. 不支持ManyToMany字段。如果你非要显示它，请自定义方法。
3. 对于BooleanField或NullBooleanField字段,会用on/off图标代替True/False。
4. 如果给list_display提供的值是一个模型的、ModelAdmin的或者可调用的方法，默认情况下会自动对返回结果进行HTML转义，这可能不是你想要的。

默认情况下，一个返回布尔值的方法在list_display中显示为True或者False的：
通常情况下，在list_display列表里的元素如果不是数据库内的某个具体字段，是不能根据它进行排序的。但是如果给这个字段添加一个admin_order_field属性，并赋值一个具体的数据库内的字段，则可以按这个字段对原字段进行排序。 **ModelAdmin.list_display_links**
指定用于链接修改页面的字段。
通常情况，list_display列表中的第一个元素被作为指向目标修改页面的超级链接点。但是，使用list_display_links可以帮你修改这一默认配置。
ModelAdmin.list_editable
这个选项是让你指定在修改列表页面中哪些字段可以被编辑。
指定的字段将显示为编辑框，可修改后直接批量保存。

注意：
一是不能将list_display中没有的元素设置为list_editable。
二是不能将list_display_links中的元素设置为list_editable。原因很简单，你不能编辑没显示的字段或者作为超级链接的字段。

ModelAdmin.list_filter

设置list_filter属性后，可以激活修改列表页面的右侧边栏，用于对列表元素进行过滤。

list_filter必须是一个元组或列表，其元素是如下类型之一：

BooleanField、CharField、DateField、DateTimeField、IntegerField、ForeignKey或者ManyToManyField

在这里，你可以利用双下划线进行跨表关联，如下例：

```
class PersonAdmin(admin.UserAdmin):
    list_filter = ('company__name',)
```

ModelAdmin.list_max_show_all

设置一个数值，当列表元素总数小于这个值的时候，将显示一个“show all”链接，点击后就能看到一个展示了所有元素的页面。该值默认为200.

ModelAdmin.list_per_page

设置每页显示多少个元素。Django自动帮你分页。默认为100。

ModelAdmin.list_select_related 如果设置了list_select_related属性，Django将会使用select_related()方法查询数据，这可能会帮助你减少一些数据库访问。

属性的值可以是布尔值、元组或列表，默认为False。当值为True时，将始终调用select_related()方法；如果值为False，Django将查看list_display属性，只对ForeignKey字段调用select_related()方法。

如果你需要更细粒度的控制，请赋值一个元组（或列表）。空元组将阻止select_related()方法，否则元组会被当做参数传递给select_related()方法。

ModelAdmin.ordering

设置排序的方式。

属性的值必须为一个元组或列表，格式和模型的ordering参数一样。如果不设置这个属性，Django将按默认方式进行排序。如果你想进行动态排序，请自己实现get_ordering()方法。

ModelAdmin.paginator

指定用于分页的分页器。

默认情况下，分页器用的是Django自带的django.core.paginator.Paginator。如果自定义分页器的构造函数接口和django.core.paginator.Paginator的不一样，那你还需要自己实现ModelAdmin.get_paginator()方法。

ModelAdmin.prepopulated_fields

设置预填充字段。

不接收DateTimeField、ForeignKey和ManyToManyField类型的字段。

ModelAdmin.preserve_filters

默认情况下，当你对目标进行创建、编辑或删除操作后，页面会依然保持原来的过滤状态。将preserve_filters设为False后，则会返回未过滤状态。

ModelAdmin.radio_fields

默认情况下，Django使用select标签显示ForeignKey或choices集合。如果将这种字段设置为radio_fields，则会以radio_box标签的形式展示。下面的例子假设group是Person模型的ForeignKey字段，

ModelAdmin.raw_id_fields

这个属性会改变默认的ForeignKey和ManyToManyField的展示方式，它会变成一个输入框，用于输入关联对象的主键id。对于ManyToManyField，id以逗号分隔。并且再输入框右侧提供一个放大镜的图标，你可以点击进入选择界面。

```
class PersonAdmin(admin.ModelAdmin):
    raw_id_fields = ("group",)
```

ModelAdmin.readonly_fields

该属性包含的字段在页面内将展示为不可编辑状态。它还可以展示模型或者ModelAdmin本身的方法的返回值，类似ModelAdmin.list_display的行为。

ModelAdmin.save_as

默认情况下，它的值为False。如果设置为True，那么右下角的“Save and add another”按钮将被替换成“Save as new”，意思也变成保存为一个新的对象。

ModelAdmin.save_as_continue

默认值为True, 在保存新对象后跳转到该对象的修改页面。但是如果这时save_as_continue=False，则会跳转到元素列表页面。

ModelAdmin.save_on_top

默认为False。 设为True时，页面的顶部会提供同样的一系列保存按钮。

ModelAdmin.search_fields

设置这个属性，可以为admin的修改列表页面添加一个搜索框。

被搜索的字段可以是CharField或者TextField文本类型，也可以通过双下划线进行ForeignKey或者ManyToManyField的查询，格式为search_fields = ['foreign_key__related_fieldname'].

例如：如果作者是博客的ForeignKey字段，下面的方式将通过作者的email地址来查询对应的博客，也就是email地址是查询值的作者所写的所有博客。

```
search_fields = ['user__email']
```

ModelAdmin.show_full_result_count

用于设置是否显示一个过滤后的对象总数的提示信息，

ModelAdmin.view_on_site

这个属性可以控制是否在admin页面显示“View site”的链接。这个链接主要用于跳转到你指定的URL页面

4、ModelAdmin methods

ModelAdmin.save_model (request, obj, form, change)
save_model 方法被赋予HttpRequest ,模型实例, ModelForm 实例和布尔 值,基于它是添加还是更改对象。在这里您可以执行任何预保存或后保存操作。 例如,在保存之前将 request.user 附加到对象:

```
from django.contrib import admin

class ArticleAdmin(admin.ModelAdmin):
    def save_model(self, request, obj, form, change):
        obj.user = request.user
        obj.save()
```

ModelAdmin.delete_model (request, obj)
delete_model 方法给出了 HttpRequest 和模型实例。使用此方法执行预删除或 后删除操作。

ModelAdmin.save_formset (request, form, formset, change)
save_formset 方法是给予HttpRequest ,父 ModelForm 实例和基于是否添加 或更改父对象的布尔值。

ModelAdmin.get_ordering (request)
get_order方法接受一个请求作为参数，并期望返回一个列表或tuple，以便与排序属性类似。

```
class PersonAdmin(admin.ModelAdmin):
    def get_ordering(self, request):
        if request.user.is_superuser:
            return ['name', 'rank']
        else:
            return ['name']
```

ModelAdmin.get_search_results (request, queryset, search_term)
get_search_results 方法将显示的对象列表修改为与提供的搜索项匹配的对象。

它接受请求,应用当前过滤器的查询集以及用户提供的搜索项。它返回一个包 含被修改以实现搜索的查询集的元组,以及一个指示结果是否可能包含重复项的布尔值。

默认实现搜索在 ModelAdmin.search_fields 中命名的字段。

ModelAdmin.save_related (request, form, formsets, change)
save_related 方法给出了 HttpRequest ,父 ModelForm 实例,内联表单列表 和一个布尔值,添加或更改。

在这里,您可以对与父级相关的对象执行任何预保存 或后保存操作。请注意,此时父对象及其形式已保存。

ModelAdmin.get_readonly_fields (request, obj=None)
返回将以只读形式显示的字段名称 的 list 或 tuple ,如上面在 ModelAdmin.readonly_fields 部分中所述。

ModelAdmin.get_prepopulated_fields (request, obj=None)
返回 dictionary ,如上面 在 ModelAdmin.prepopulated_fields 部分中所述。

ModelAdmin.get_list_display (request)
返回字段名称的 list 或 tuple 显示在如上所述的 ModelAdmin.list_display 部分中的 changelist视图上。

ModelAdmin.get_list_display_links (request, list_display)
返回 更改列表上将链接到更改视图的字段名称的 None 或 list 或 tuple ,如上所述在 ModelAdmin.list_display_links 部分中。

ModelAdmin.get_fields (request, obj=None)
返回字段列表,如上面在 ModelAdmin.fields 部分中所述。

ModelAdmin.get_fieldsets (request, obj=None)
返回二元组列表。

ModelAdmin.get_list_filter (request)
返回与 list_filter 属性相同类型的序列类型。

ModelAdmin.get_search_fields (request)
返回与 search_fields 属性相同类型的序列类型。

ModelAdmin.get_inline_instances (request, obj=None)
返回 list 或 tuple 的 InlineModelAdmin 对象,如下面的 InlineModelAdmin 部分所述。

ModelAdmin.get_urls()
ModelAdmin 的 get_urls 方法返回ModelAdmin 将要用到的URLs,方式与URLconf 相同。因此,你可以用URL 调度器中所述的方式扩展它们。

ModelAdmin.get_form (request, obj=None, **kwargs)
返回Admin中添加和更改视图使用的 ModelForm 类,请参阅 add_view() 和change_view() 。 其基本的实现是使用 modelform_factory() 来子类化 form ,修改如 fields 和 exclude 属性。

ModelAdmin.get_formsets_with_inlines (request, obj=None)
产量(FormSet , InlineModelAdmin)对用于管理添加和更改视图。

ModelAdmin.formfield_for_foreignkey (db_field, request, **kwargs)
允许覆盖外键字段的默认窗体字段。 **ModelAdmin.formfield_for_manytomany (db_field, request, **kwargs)**
可以覆盖 formfield_for_manytomany 方法来更改多对多字段的默认窗体字段。

ModelAdmin.formfield_for_choice_field (db_field, request, **kwargs)
可以覆盖 formfield_for_choice_field 方法更改已声明选择的字段的默认窗体字 段。

ModelAdmin.get_changelist (request, **kwargs)
返回要用于列表的 Changelist 类。默认情况下,使用django.contrib.admin.views.main.ChangeList 。通过继承此类,您可以 更改列表的行为。

ModelAdmin.get_changelist_form (request, **kwargs)
返回 ModelForm 类以用于更改列表页面上的 Formset 。

ModelAdmin.get_changelist_formset (request, **kwargs)

如果使用 list_editable ,则返回ModelFormSet类以在更改列表页上使用。
ModelAdmin.has_add_permission (request)
如果允许添加对象,则应返回 True ,否则返回 False 。

ModelAdmin.has_change_permission (request, obj=None)

如果允许编辑obj,则应返回 True ,否则返回 False

ModelAdmin.has_delete_permission (request, obj=None)

如果允许删除obj,则应返回 True ,否则返回 False 。

ModelAdmin.has_module_permission (request)

如果在管理索引页上显示模块并允许访问模块的索引页,则应返回 True ,否 则 False 。默认情况下使用 User.has_module_perms() 。

ModelAdmin.get_queryset (request)

ModelAdmin 上的 get_queryset 方法会返回管理网站可以编辑的所有模型实例 的 QuerySet 。

ModelAdmin.message_user (request, message, level=messages.INFO, extra_tags='', fail_silently=False)

使用 django.contrib.messages 向用户发送消息。

ModelAdmin.get_paginator(queryset,per_page,orphans=0,allow_empty_first_page=True)

返回要用于此视图的分页器的实例。默认情况下,实例化 paginator 的实例。

ModelAdmin.response_add (request, obj, post_url_continue=None)

为 add_view() 阶段确定 HttpResponse 。 response_add 在管理表单提交后,在对象和所有相关实例已创建并保存之后调 用。

ModelAdmin.response_change (request, obj)

确定 change_view() 阶段的 HttpResponse 。 response_change 在Admin 表单提交并保存该对象和所有相关的实例之后调 用。

ModelAdmin.response_delete (request, obj_display, obj_id)

为 delete_view() 阶段确定 HttpResponse 。 在对象已删除后调用 response_delete 。您可以覆盖它以在对象被删除后更改默 认行为。 obj_display 是具有已删除对象名称的字符串。 obj_id 是用于检索要删除的对象的序列化标识符。

ModelAdmin.get_changeform_initial_data (request)

用于管理员更改表单上的初始数据的挂钩。默认情况下,字段从 GET 参数给出初 始值。

其他方法

这五个方法实际上被设计为从管 理应用程序URL调度处理程序调用为Django视图,以呈现处理模型实例的页面 CRUD操作。因此,完全覆盖这些方法将显着改变管理应用程序的行为。覆盖这些方法的一个常见原因是增加提供给呈现视图的模板的上下文数据。

ModelAdmin.add_view (request, form_url='', extra_context=None)
Django视图为模型实例添加页面。

ModelAdmin.change_view (request, object_id, form_url='', extra_context=None)
Django视图为模型实例版本页。

ModelAdmin.changelist_view (request, extra_context=None)
Django视图为模型实例更改列表/操作页面。

ModelAdmin.delete_view (request, object_id, extra_context=None)
模型实例删除确认页面的Django 视图。

ModelAdmin.history_view (request, object_id, extra_context=None)

显示给定模型实例的修改历史的页面的Django视图。

5、ModelAdmin asset definitions

有时候你想添加一些CSS和/或JavaScript到添加/更改视图。这可以通过 在 ModelAdmin 上使用 Media 内部类来实现:

```
class ArticleAdmin(admin.ModelAdmin):
   class Media:
        css = {
            "all": ("my_styles.css",)
        }
        js = ("my_code.js",)
```

6、jQuery

Django管理JavaScript使用jQuery库。

为了避免与用户提供的脚本或库冲突,Django的jQuery(版本1.11.2)命名为 django.jQuery 。如果您想在自己的管理JavaScript中使用jQuery而不包含第 二个副本,则可以使用更改列表上的 django.jQuery 对象和添加/编辑视图。

默认情况下, ModelAdmin 类需要jQuery,因此除非有特定需要,否则不需要向您 的 ModelAdmin 的媒体资源列表添加jQuery。

向管理员添加自定义验证

在管理员中添加数据的自定义验证是很容易的。自动管理界面重用 django.forms ,并且 ModelAdmin 类可以定义您自己的形式:

```
class ArticleAdmin(admin.ModelAdmin):
      form = MyArticleAdminForm
```

MyArticleAdminForm 可以在任何位置定义,只要在需要的地方导入即可。现 在,您可以在表单中为任何字段添加自己的自定义验证:

```
class MyArticleAdminForm(forms.ModelForm):
    def clean_name()self:
        # do something that validates your data
        return self.cleaned_data["name"]
```

B、InlineModelAdmin objects

```
class InlineModelAdmin
class TabularInline
class StackedInline
```

此管理界面能够在一个界面编辑多个Model。这些称为内联。

可以通过在 ModelAdmin.inlines 中指定模型来为模型添加内联:

```
from django.contrib import admin

  class BookInline(admin.TabularInline):
       model = Book

   class AuthorAdmin(admin.ModelAdmin):
       inlines = [
           BookInline,
        ]
```

Django提供了两个 InlineModelAdmin 的子类如下:

- TabularInline
- StackedInline

这两者之间仅仅是在用于呈现他们的模板上有区别。

1、InlineModelAdmin options

InlineModelAdmin 与 ModelAdmin 具有许多相同的功能,并添加了一些自己的功能(共享功能实际上是在 BaseModelAdmin 超类中定义的)。

- 形成
- fieldets
- 字段
- formfield_overrides
- 排除
- filter_horizontal
- filter_vertical
- 订购
- prepopulated_fields
- get_queryset()
- radio_fields
- readonly_fields
- raw_id_fields
- formfield_for_choice_field()
- formfield_for_foreignkey()
- formfield_for_manytomany()
- has_add_permission()
- has_change_permission()
- has_delete_permission()
- has_module_permission()

InlineModelAdmin.model 内联正在使用的模型。这是必需的。

InlineModelAdmin.fk_name 模型上的外键的名称。在大多数情况下,这将自动处理,但如果同一父模型有多个外键,则必须显式指定 fk_name 。

InlineModelAdmin.formset 默认为 BaseInlineFormSet 。使用自己的表单可以给你很多自定义的可能性。

InlineModelAdmin.form form 的值默认为 ModelForm 。这是在为此内联创建表单集时传递 到 inlineformset_factory() 的内容。

InlineModelAdmin.extra 这控制除初始形式外,表单集将显示的额外表单的数量。 **InlineModelAdmin.max_num** 这控制在内联中显示的表单的最大数量。这不直接与对象的数量相关,但如果值足够小,可以。 **InlineModelAdmin.min_num** 这控制在内联中显示的表单的最小数量。 InlineModelAdmin.get_min_num() 还允许您自定义显示的表单的最小数量。 **InlineModelAdmin.raw_id_fields** raw_id_fields是您希望更改为ForeignKey或ManyToManyField 的 Input 窗口小部件的字段列表:

```
class BookInline(admin.TabularInline):
    model = Book
    raw_id_fields = ("pages",)
```

InlineModelAdmin.template 用于在页面上呈现内联的模板。

InlineModelAdmin.verbose_name 覆盖模型的内部 Meta 类中找到的 verbose_name 。

InlineModelAdmin.verbose_name_plural 覆盖模型的内部 Meta 类中的 verbose_name_plural 。 **InlineModelAdmin.can_delete** 指定是否可以在内联中删除内联对象。默认为 True 。 **InlineModelAdmin.show_change_link** 指定是否可以在admin中更改的内联对象具有指向更改表单的链接。默认 为 False 。 **InlineModelAdmin.get_formset (request, obj=None, **kwargs)** 返回 BaseInlineFormSet 类,以在管理员添加/更改视图中使用。请参阅 ModelAdmin.get_formsets_with_inlines 的示例。 **InlineModelAdmin.get_extra (request, obj=None, **kwargs)** 返回要使用的其他内联表单的数量。默认情况下,返 回 InlineModelAdmin.extra 属性。 **InlineModelAdmin.get_max_num (request, obj=None, **kwargs)** 返回要使用的额外内联表单的最大数量。 **InlineModelAdmin.get_min_num (request, obj=None, **kwargs)** 返回要使用的内联表单的最小数量。 覆盖此方法以编程方式确定最小内联表单数。

2、使用具有两个或多个外键的模型到同一个父模型

```
from django.db import models
    class Friendship(models.Model):
        to_person = models.ForeignKey(Person, related_name="friends")
        from_person = models.ForeignKey(Person, related_name="from_friends")
```

如果您想在 Person 管理员添加/更改页面上显示内联,则需要明确定义外键,因 为它无法自动执行:

```
from django.contrib import admin
from myapp.models import Friendship

 class FriendshipInline(admin.TabularInline):
   model = Friendship
   fk_name = "to_person"

class PersonAdmin(admin.ModelAdmin):
    inlines = [
       FriendshipInline,
    ]
```

3、使用多对多模型

默认情况下,多对多关系的管理窗口小部件将显示在包含 ManyToManyField 的实 际引用的任何模型上。 假设我们有以下模型:

```
from django.db import models

class Person(models.Model):
   name = models.CharField(max_length=128)

class Group(models.Model):
   name = models.CharField(max_length=128)
    members = models.ManyToManyField(Person, related_name='groups')
```

如果要使用内联显示多对多关系,可以通过为关系定义 InlineModelAdmin 对象 来实现:

```
from django.contrib import admin

class MembershipInline(admin.TabularInline):
   model = Group.members.through

class PersonAdmin(admin.ModelAdmin):
   inlines = [MembershipInline,]

class GroupAdmin(admin.ModelAdmin):
    inlines = [MembershipInline,]
    exclude = ('members',)
```

这个例子中有两个值得注意的特征。 **首先** - MembershipInline 类引用 Group.members.through 。through 属性 是对管理多对多关系的模型的引用。在定义多对多字段时,此模型由Django自动创 建。 **其次**, GroupAdmin 必须手动排除 members 字段。

4、使用多对多中介模型

当您使用 ManyToManyField 的 through 参数指定中介模型时,管理员将不会默 认显示窗口小部件。这是因为该中间模型的每个实例需要比可以在单个小部件中显 示的更多的信息,并且多个小部件所需的布局将根据中间模型而变化。 但是,我们仍然希望能够内联编辑该信息。

5、使用泛型关系作为内联

略

C、重写admin模板

相对重写一个admin站点的各类页面,直接在admin站点默认templates上直接进行修改是件相对简单的事。

1、设置项目的admin模板目录

Admin模板文件位于 contrib/admin/templates/admin 目录中。 如要覆盖一个或多个模板,首先在你的项目的 templates 目录中创建一个 admin 目录。它可以是你在 TEMPLATES 设置的 DjangoTemplates 后端 的 DIRS 选项中指定的任何目录。

如果你已经自定义 'loaders' 选项,请确保 'django.template.loaders.filesystem.Loader' 出现在 'django.template.loaders.app_directories 之前。Loader',以便在包 含 django.contrib.admin 的模板之前,模板加载系统可以找到您的自定义模 板。 在 admin 目录下, 以你的应用名创建子目录. 在应用名的目录下,以你模型层的 名字创建子目录. 注意:admin应用会以小写名的形式在目录下查找模型, 如果你想 在大小写敏感的文件系统上运行app,请确保以小写形式命名目录. 为一个特定的app重写admin模板, 需要拷贝 django/contrib/admin/templates/admin 目录到你刚才创建的目录下, 并且修改它们。

如果我们只想为名为“Page”的特定模型添加一个工具到更改列表视图,我们将把同 一个文件复制到我们项目的 templates/admin/my_app/page 目录。

2、覆盖于替换管理模板

由于管理模板的模块化设计,通常既不必要也不建议替换整个模板。最好只覆盖模板中需要更改的部分。 如果我们将此文件放在 templates/admin/my_app 目录中,我们的链接将出现在my_app中所有模型更改表单上。

3、每个应用或模型中可以被重写的模板

不是 contrib/admin/templates/admin 中的每个模板都可以在每个应用或每个 模型中覆盖。以下可以:

```
app_index.html
change_form.html
change_list.html
delete_confirmation.html
object_history.html
```

对于那些不能以这种方式重写的模板,你可能仍然为您的整个项目重写它们。只需要将新版本放在你的 templates/admin 目录下。这对于要创建自定义的404 和 500 页面特别有用。

4、Root and login模板

如果你想要更改主页、 登录或登出页面的模板,你最后创建你自己的 AdminSite 实例(见下文),并更改 AdminSite.index_template 、 AdminSite.login_template和 AdminSite.logout_template 属性。

D、AdminSite objects

Django 的一个Admin 站点通过 django.contrib.admin.sites.AdminSite 的一 个实例表示;默认创建的这个类实例是 django.contrib.admin.site ,你可以通过它注册自己的模型和 ModelAdmin 实例。

当构造 AdminSite 的实例时,你可以使用 name 参数给构造函数提供一个唯一 的实例名称。这个实例名称用于标识实例,尤其是反向解析Admin URLs 的时候。

1、AdminSite attributes

AdminSite.site_header 每个Admin 页面顶部的文本,形式为 <h1> (字符串)。默认为 “Django administration”。

AdminSite.site_title 每个Admin 页面底部的文本,形式为 <title> (字符串)。默认 为“Django site admin”。

AdminSite.site_url 每个Admin 页面顶部"View site" 链接的URL。默认情况下, site_url 为 / 。设 置为 None 可以删除这个链接。

AdminSite.index_title Admin 主页顶部的文本(一个字符串)。默认为 “Site administration”。

AdminSite.index_template Admin 站点主页的视图使用的自定义模板的路径。

AdminSite.login_template Admin 站点登录视图使用的自定义模板的路径。

AdminSite.login_form Admin 站点登录视图使用的 AuthenticationForm 的子类。

AdminSite.logout_template Admin 站点登出视图使用的自定义模板的路径。

AdminSite.password_change_template Admin 站点密码修改视图使用的自定义模板的路径。

AdminSite.password_change_done_template Admin 站点密码修改完成视图使用的自定义模板的路径。

2、AdminSite methods

AdminSite.ach_context (request) 返回一个字典,包含将放置在Admin 站点每个页面的模板上下文中的变量。 包含以下变量和默认值:

- site_header : AdminSite.site_header
- site_title : AdminSite.site_title
- site_url : AdminSite.site_url
- has_permission : AdminSite.has_permission()

AdminSite.has_permission (request) 对于给定的 HttpRequest ,如果用户有权查看Admin 网站中的至少一个页面,则返回 True 。

3、绑定AdminSite

设置Django Admin 的最后一步是放置你的 AdminSite 到你的URLconf 中。通过 指向给定的URL 到 AdminSite.urls 方法来执行此操作。

```
# urls.py
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
   url(r'^admin/', include(admin.site.urls)),
]
```

4、自定义AdminSite

如果你想要建立你自己的具有自定义行为Admin 站点,你可以自由地子类化 AdminSite 并重写或添加任何你喜欢的东西。你只需创建 AdminSite 子类的实例(方式与你会实例化任何其它Python 类相同)并注册你的模型和ModelAdmin 子类与它而不是默认的站点。最后,更新 myproject/urls.py 来引用你的 AdminSite 子类。

5、相同URLConfzhong有多个Admin站点

在Django 构建的同一Web 站点上创建Admin 站点的多个实例非常容易。只需要创 建 AdminSite 的多个实例并将每个实例放置在不同的URL 下。

6、向管理网站添加视图

与 ModelAdmin 一样, AdminSite 提供了一个 get_urls() 方法,可以重写该方法以定义网站的其他视图。要向您的管理网站添加新视图,请扩展基本 get_urls() 方法,为新视图添加模式。

7、加入一个密码重置的特性

想admin site加入密码重置功能只需要在url配置文件中简单加入几行代码即可。 体操作就是加入四个正则规则:

```
from django.contrib.auth import views as auth_views

url(r'^admin/password_reset/$', auth_views.password_reset,
name='admin_password_reset'),
url(r'^admin/password_reset/done/$', auth_views.password_reset_done,
name='password_reset_done'),
url(r'^reset/(?P<uidb64>[0-9A-Za-z_\-]+)/(?P<token>.+)/$',
auth_views.password_reset_confirm,           name='password_reset_confirm'),
url(r'^reset/done/$', auth_views.password_reset_complete,
name='password_reset_complete'),
```

E、反向解析的AdminURL

略

二、管理操作

简而言之,Django管理后台的基本流程是,“选择一个对象并改变它”。

A、编写操作

首先,我们需要定义一个函数,当后台操作被点击触发的时候调用。操作函数,跟普通的函数一样,需要接收三个参数:

- 当前的 ModelAdmin
- 表示当前请求的 HttpRequest
- 含有用户所选的对象集合的 QuerySet

我们 可以提供一个更好、更人性化的名称,通过向 make_published 函数添加 short_description 属性:

```
def make_published(modeladmin, request, queryset):
    queryset.update(status='p')
    make_published.short_description = "Mark selected stories as published"
```

B、添加操作到ModelAdmin

把操作告诉 ModelAdmin 。它和其他配置项的工作方式相同。 所以,带有操作及其注册的完整的 admin.py 看起来像这样:

```
from django.contrib import admin
from myapp.models import Article

def make_published(modeladmin, request, queryset):
    queryset.update(status='p')
    make_published.short_description = "Mark selected stories as published"

class ArticleAdmin(admin.ModelAdmin):
    list_display = ['title', 'status']
    ordering = ['title']
    actions = [make_published]

admin.site.register(Article, ArticleAdmin)
```

C、操作的高级技巧

1、ModelAdmin上的操作 ModelAdmin

由于操作与 Article 紧密耦 合,不如将操作直接绑定到 ArticleAdmin 对象上更有意义。 首先注意,我们将 make_published 放到一个方法中,并重命名 modeladmin 为 self ,其次,我们现在将'make_published' 字符串放进 了 actions ,而不是一个直接的函数引用。这样会让 ModelAdmin 将这个操作视为方法。

2、提供中间页面的操作

在执行操作之后,用户会简单地通过重定向返回到之前的修改列表页面中。然而,一些操作,尤其是更加复杂的操作,需要返回一个中间页面。例如,内建的删除操作,在删除选中对象之前需要向用户询问来确认。要提供中间页面,只要从你的操作返回 HttpResponse (或其子类)就可以了。

```
response = HttpResponse(content_type="application/json")
```

3、在整个站点应用操作

你可以使用 AdminSite.add_action() 让一个操作在全局都可以使用。

```
Syntax:
AdminSite.add_action (action[, name])

Code:
admin.site.add_action(export_selected_objects)
```

4、禁用操作

1、如果你需要禁用站点级操作 ,你可以调用 AdminSite.disable_action() 。例如,你可以使用这个方法来移除内建的“删除选中的对象”操作:

```
admin.site.disable_action('delete_selected')
```

2、如果你想批量移除所提供 ModelAdmin 上的所有操作,可以 把 ModelAdmin.actions 设置为 None :

```
 class MyModelAdmin(admin.ModelAdmin):
    actions = None
```

3、可以通过覆写 ModelAdmin.get_actions() ,对每个请求(每个用户) 按需开启或禁用操作。

```
ModelAdmin.get_actions (request)
```

三、管理文档生成器

在某种程度上,你可以使用 admindocs 来快为你自己的代码生成文档。这个应用的功能十分有限,然而它主要用于文档模板、模板标签和过滤器。

要启用 admindocs ,你需要执行以下步骤:

- 向 INSTALLED_APPS 添加 django.contrib.admindocs 。 向你的 urlpatterns 添加(r'^admin/doc/' ,
- include('django.contrib.admindocs.urls'))。 确保它在 r'^admin/' 这一项 之前包含,以便 /admin/doc/ 的请求不会被后面的项目处理。
- 安装 docutils Python 模块 (http://docutils.sf.net/)。
- 可选的: 使用 admindocs 的书签功能需要安
- 装 django.contrib.admindocs.middleware.XViewMiddleware 。