第一讲 入门

1 开篇

Django 是新近出来的 Rails 方式的 web 开发框架。在接触 Django 之前我接触过其它几种 Python 下的 web framework, 但感觉 Karrigell 是最容易上手的。不过 Django 从我个人的感觉上来看，它的功能更强大，社区也很活跃，高手众多，发展也是极为迅速。

3 Django 的入门体验

但 Django 呢？如果说最简单的 web 体验 Hello, Django! 如何写呢？决不会象 Karrigell 那样简单，只从它提供的教程来看，你无法在安装后非常 Easy 地写出一个 Hello, Django! 的例子，因为有一系列的安装和准备工作要做。那么下面我把我所尝试写最简单的 Hello, Django! 的例子写出来。请注意，我测试时是在 Windows XP 环境下进行的。

3.1 安装

```
python setup.py install
```

参考文档 Django installed，一般地，Django 安装前还需要先安装 setuptools 包。可以从 PyPI 上搜到。目前最新的版本是 0.95 版，可以从 Django 的主页上面下载。如果你想从老的 0.91 迁移到最新版本，可以参阅 RemovingTheMagic 文档。安装后，建议检查 pythoninstalldir/Scripts 目录是否在你的 PATH 环境中，如果不在，建议将这个目录设置到 PATH 中。因为如果你采用标准的 Python 安装方法，那么 Django 会自动在 Scripts 目录下安装 django-admin.py 程序。这样，一旦你设置了 Scripts 在 PATH 中，就可以在命令行下任何目录中执行 django-admin.py 了。

3.2 生成项目目录

因为 Karrigell 可直接开发，因此放在哪里都可以。而 Django 是一个框架，它有特殊的配置要求，因此一般不需要手工创建目录之类的工作， Django 提供了 django-admin.py 可以做这件事。为使用 django-admin.py ，建议将 Python 的 Scripts 目录加入到 PATH 环境变量中去。

```
django-admin.py startproject newtest
```

这样就在当前目录下创建了一个 newtest 目录，进去入可以看到有四个文件：这个 newtest 将是我们以后工作的目录，许多讲解都是基于这个目录的。

__init__.py

表示这是一个 Python 的包

manage.py

提供简单化的 django-admin.py 命令，特别是可以自动进行

DJANGO_SETTINGS_MODULES 和 PYTHONPATH 的处理，而没有这个命令，处理上面环境变量是件麻烦的事情

settings.py 它是 django 的配置文件

uls.py url 映射处理文件，Django 的 url 映射是 url 对某个模块方法的映射，目前不能自动完成

在 0.91 版， django-admin.py startproject 会生成 apps 目录。但 0.95 版之后已经没有了。虽然 django-admin.py 为我们生成了许多东西， 而且这些东西在以后的开发中你都需要熟悉，但现在我们的目标是最简单的体验，就认为我们不需要知道它们都有什么用吧。

项目创建好了，那么我们可以启动服务器吗？ Django 为了开发方便，自带了一个用于开发的 web server。在 0.91 版，你需要至少修改一下 settings.py 中的 DATABASE_ENGINE ，如果你不改，那么 Django 会报错。在 0.95 版之后，不再需要设置了。

3.3.1 启动 web server:

别急呀，还没看见 Hello, Django! 在哪里呢。是的，我只是想看一看， Django 能否启动。

```
manage.py runserver
```

一旦出现:

```
Validating models...
```


```
0 errors found.
Starting server on port 8000 with settings module 'newtest.settings'.
Go to http://127.0.0.1:8000/ for Django.
Quit the server with CONTROL-C (Unix) or CTRL-BREAK (Windows).
```

说明 Django 真的启来了。在浏览器中看一下，有一个祝贺页面，说明成功了。

3.3.2 更改主机或端口

默认情况下， runserver 命令在 8000 端口启动开发服务器，且只监听本机连接。要想要更改服务器端口的话，可将端口作为命令行参数传入：

```
python manage.py runserver 8080
```

还可以改变服务器监听的 IP 地址。要和其他开发人员共享同一开发站点的话，该功能特别有用。下面的命令：

```
python manage.py runserver 0.0.0.0:8080
```

会让 Django 监听所有网络接口，因此也就让其它电脑可连接到开发服务器了

3.4 增加一个 helloworld 的 app 吗？

在 Django 中绝大多数应用都是以 app 形式存在的，但一定要加吗？其实并不一定。在 Django 中，每个 app 就是一个子包，真正调用时需要通过 URL Dispatch 来实现 url 与模块方

法的映射。这是 Django 的一大特色，但也是有些麻烦的地方。不用它，你无法发布一个功能，如果在 Django 中存在一种缺省的简单映射的方式，这样我想可以大大提高 Django 的入门体验度。不过在比较大的项目中，使用 app 还是一个非常好的方式，它可将项目功能进行分割，以便组织和代码的重用。因此根据 URL Dispatch 的机制，我们只要保证 Django 可以在正确的地方找到方法进行调用即可。那么我们就根本不去创建一个 app 了。在 newtest 目录下建一个文件 helloworld.py 内容为:

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hello, Django.")
```

0.95 版之后对许多 Django 模块都做了简化。具体可参考： NamespaceSimplification 文档。

3.5 修改 urls.py

```
from django.conf.urls.defaults import *
urlpatterns = patterns('',
    # Example:
    # (r'^newtest/',include('newtest.apps.foo.urls.foo')),
    (r'^$', 'newtest.helloworld.index'),
    # Uncomment this for admin:
     #(r'^admin/', include('django.contrib.admin.urls')),
)
```

上面的 r'^$' 是为了匹配空串，也就是形如: http://localhost:8000/ 。如果这时 web server 已经启动了，那么直接刷新页面就行了。现在觉得 Django 是不是简单多了，除了创建一个项目的操作，然后可能要修改两个配置文件。

4 结论

Django 本身的确是一种松散的框架组合，它既复杂又简单。复杂是因为如果你想使用它的自动化的、高级的功能你需要学习很多的东西，而且它的教程一上来就是以这种过于完整的例子进行展示，自然会让你觉得很麻烦。不过看了我的讲解之后，是不是觉得还是挺简单的。那么我们就先以无数据库的方式进行下去，一点点地发掘 Django 的功能特性吧。

第二讲 生成一个 web form 做加法的简单例子

1 引言

随着学习，我们的例子也开始复杂了，下一步我想实现一个简单的 web 加法器。界面会是这样：如何处理页面表格提交的数据，并且会对 URL Dispatch 作更进一步的解释。

2 创建 add.py 文件

我们先创建一个 add.py 文件。(由于我们还没有涉及到 Django 的模型，因此象 add.py 这样的东西叫什么呢？还是称其为 View 吧。因为在 django 中，View 是用来显示的，它代替了一般的 MVC 中的 Control 的作用，因为 Django 中不是 MVC 而是 MTV (Model Template View))

```
from django.http import HttpResponse
text = """<form method="post" action="/add/">
    <input type="text" name="a" value="%d"> + <input type="text" name="b"
value="%d">
    <input type="submit" value="="> <input type="text" value="%d">
</form>"""
def index(request):
    if request.POST.has_key('a'):
        a = int(request.POST['a'])
        b = int(request.POST['b'])
    else:
        a = 0
        b = 0
    return HttpResponse(text % (a, b, a + b))
```

请注意 action 为 /add/ ，在 Django 中链接后面一般都要有 '/' ，不然有可能得不到 POST 数据。有关更详细的关于常见问题可以参阅 NewbieMistakes 文档。

这里只有一个 index 方法。所有在 view 中的方法第一个参数都会由 Django 传入 request 对象，它就是请求数据对象，它是由 Django 自动生成。其中有 GET 和 POST 属性，分别保存两种不同的提交方式的数据，它们都可以象字典一样工作。更多关于 request 的内容参见 Request and response objects 文档。

那么我的想法就是：进入页面后(就是上面的效果)，页面上有两个输入文本框，一个是提交按钮，一个是显示结果的文本框。在两个输入文本框中输入整数，然后点击提交("="号按钮)，将返回相同的页面，但结果文本框中将显示两数相加的和。两个输入文本框分别定义为 a 和 b 。这里的逻辑就是：先判断 POST 数据中是否有变量 a ，如果没有则表示是第一次进入，则 a, b 初始为 0 ，然后返回页面。如果有变量 a ，则计算结果，返回页面。

3 修改 urls.py,增加 add 的 url 映射。

```
from django.conf.urls.defaults import *
urlpatterns = patterns('',
    # Example:
    # (r'^testit/', include('newtest.apps.foo.urls.foo')),
    (r'^$', 'newtest.helloworld.index'),
    (r'^add/$', 'newtest.add.index'),
    # Uncomment this for admin:
#     (r'^admin/', include('django.contrib.admin.urls')),
)
```

4 启动 server

5 在浏览器测试:http://localhost:8000/add,你会看到和我相似的界面，然后输入整数试一试。

6 补充说明

1. 在 form 中的 method="post" 。你当然可以使用 get ，但是在 Django 的设计风格中认为，使用 POST 表示要对数据进行修改，使用 GET 则只是获取。
2. Django 提供了 URL Dispatch 文档，专门讲解有关 url 映射的东西。其中有一部分是关于 url 的正则表达式解析的。原本我认为象 Karrigell 中一样，定义在 form 中的变量会自动映射为方法的参数，但是我错了。方法中的参数是从 url 中通过正则表达式解析出来的，或者是在 url_conf(即 urls.py 文件)中指定的。因此它与 Karrigell 一点也不一样。因此，如果你想从 POST 或 GET 数据中得到值，那么象我一样去做好了。使用 request.POST 或 request.GET 或还有一个可以“统吃”的方法 request.REQUEST ，它们是一个字典数据，使用起来也算方便。

从这里我更想了解方法中参数的使用，当然这个例子并没有，有机会再使用吧。关于正则表达式解析参数在 blog 和 rss 中用得是非常多的。

第三讲 使用 Template 的简单例子

1 引言

从上一例我们看到，表格的生成是直接在 index() 函数中返回的 HTML 代码，这种混合方式对于大型开发非常不好，下面我们就学习模板的使用。 Django 自带有模板系统，但你可以不使用它，只要在 return 前使用自已喜欢的模板系统进行处理，然后返回即可。但 Django 自带的模板系统有很多特点，我不做过多的说明。我只是想使用它。现在我的问题就是：我有一个通讯录数据，我想使用一个表格来显示。为了方便，我们不需要使用数据库，因此我把它存在 view 文件中

2 创建 list.py

```
#coding=utf-8
from django.shortcuts import render_to_response
address = [
```

```
    {'name':'张三', 'address':'地址一'},

    {'name':'李四', 'address':'地址二'}
    ]
def index(request):
    return render_to_response('list.html', {'address': address})
```

我使用了汉字，并且字符的编码使用了 utf-8 ，请注意，而且以后如果不特别注明，所有的带汉字的文件，包括模板都将使用 utf-8 编码。

这里使用了一个新方法是 **render_to_response** ，它可以直接调用模板并返回生成好的文本。它接收两个参数，第一个是模板的文件名。在 Django 0.91 中，模板文件都是以 .html 结尾的，并且使用时是不带后缀的。但 0.95 版本取消了缺省模板后缀的设置，因此模板文件名必须是完整的，不再有默认后缀了。第二个参数是一个字典，这里只有一个 Key ，名字是 address ，它的值是一个字典的列表。只要注意模板所接收的就是这样的字典和包含字典的列表就行了。在 0.91 中 render_to_response 是在 django.core.extensions 中的，而到了 0.92 转变为 django.shortcuts 。

3 在 newtest 中创建 templates 目录:用来存放模板文件

4 修改 settings.py

```
TEMPLATE_DIRS = (
    # Put strings here, like "/home/html/django_templates".
    # Always use forward slashes, even on Windows.
    './templates',
)
```

如果有多个模板目录，加进去就行了。Django 会按顺序搜索的。Django 还支持在 App 中定义一个 templates 目录。这样 Django 在启动时会检查所有安装的的 App 的 templates 目录，如果存在，则将路径的搜索放在 TEMPLATE_DIRS 之后。这样就可以很方便地管理你的模板了。

5 创建 templates/list.html

```
<h2>通讯录</h2>
<table border="1">
 <tr><th>姓名</th><th>地址</th></tr>
 {% for user in address %}
 <tr>
 <td>{{ user.name }}</td>
 <td>{{ user.address }}</td>
 </tr>
{% endfor %}
</table>
```

生成了一个两列的表格。在 Django 模板中 {{}} 表示引用一个变量，{%%} 表示代码调用。在变量引用中，Django 还支持对变量属性的访问，同时它还有一定的策略，详细的建议查看 The Django template language 文档。这里我也使用了汉字，因此它也需要使用 utf-8 编码。

这里使用 for .. in 的模板 Tag 处理。因此 address 需要是一个集合。在我们的 View 代码中，address 为一个 list 值。每个 list 又是一个字典。因此 {{ user.name }} 和 {{ user.address }} 就是将此字典中的元素取出来。后面我们将了解更多的模板中的 Tag 标签的使用。且你会发现，Django 中的 Tag 很强大，可通过扩展形成庞大的 Tag 库方便模板的开发

6 修改 urls.py 增加了 list 的 url 映射。

```
from django.conf.urls.defaults import *
urlpatterns = patterns('',
    # Example:
    # (r'^testit/', include('newtest.apps.foo.urls.foo')),
    (r'^$', 'newtest.helloworld.index'),
    (r'^add/$', 'newtest.add.index'),
    (r'^list/$', 'newtest.list.index'),
    # Uncomment this for admin:
#     (r'^admin/', include('django.contrib.admin.urls')),
)
```

7 启动 server：http://127.0.0.1:8000/list,效果如这样：

第四讲 生成 csv 格式文件并下载

1 引言

经过前几节的学习，我想大家应该比较熟悉 Django 的大致开发流程了：

- 增加 view 方法
- 增加模板
- 修改 urls.py

剩下的就是挖掘 Django 提供的其它的能力。在我们还没有进入模型(model)之前还是再看一看外围的东西，再更进一步体验 Django 吧。在 Django 中我看到了一个生成 csv 格式的文档(Outputting CSV dynamically)，非常好，它没有数据库，正好用来做演示。现在我的需求就是提供 csv 格式文件的下载。我们会在原来 list(表格) 例子基础上进行演示，步骤就是上面的流程。

2 修改 templates/list.html

在文件最后增加:

```
<p><a href="/csv/address/">csv 格式下载</a></p>
```

它将显示为一个链接，它所指向的链接将用来生成 csv 文件。

3 在 newtest 下增加 csv_test.py

```
#coding=utf-8
from django.http import HttpResponse
from django.template import loader, Context
address = [
```

```
    ('张三', '地址一'),

    ('李四', '地址二')
    ]
def output(request, filename):
    response = HttpResponse(mimetype='text/csv')
    response['Content-Disposition'] = 'attachment; filename=%s.csv' %
filename
    t = loader.get_template('csv.html')
    c = Context({'data': address,})
    response.write(t.render(c))
    return response
```

loader, Context 是从 django.core.template 导入的，而 0.95 为 django.template 。具体可以参考： NamespaceSimplification 文档。以后类似的地方不再详述。

这里使用的东西多了一些。这里没有 render_to_response 了，而是演示了一个完整的从头进行模板解析的处理过程。为什么需要这样，因为我们需要修改 response 对象的值，而 render_to_response 封装了它使得我们无法修改。从这里我们也可以看到，在调用一个方法时，Django 会传入一个 request 对象，在返回时，你需要将内容写入 response ，必要时修改它的某些属性。更详细的建议你参考 django 所带的 request_response 文档，里面详细描述了两个对象的内容，并且还可以在交互环境下进行测试，学习非常方便。response 对象也是由 Django 自动维护的。具体的内容参见 Request and response objects 文档。

这里 address 不再是字典的列表，而是 tuple 的列表。让人高兴的是， Django 的模板除了可以处理字典，还可以处理序列，而且可以处理序列中的元素。一会在模板定义中我们会看到。

这里 output() 是我们希望 Django 调用的方法，不再是 index() 了。而且它与前面的 index() 不同，它带了一个参数。这里主要是想演示 url 的参数解析。因此你要注意，这个参数一定是放在 url 上的。它表示输出文件名。

```
response = HttpResponse(mimetype='text/csv')
```

```
response['Content-Disposition'] = 'attachment; filename=%s.csv' %
filename
```

这两行是用来处理输出类型和附件的。它表明返回的是一个 csv 格式的文件。

```
t = loader.get_template('csv.html')
c = Context({'data': address,})
response.write(t.render(c))
```

这几行就是最原始的模板使用方法。先通过 loader 来找到需要的模板，然后生成一个 template 对象，再生成一个 Context 对象，它就是一个字典集。然后 t.render(c) 这个用来对模板和提供的变量进行合并处理，生成最终的结果。最后调用 response.write() 将内容写入。

4 增加 templates/csv.html

```
{% for row in data %}"{{ row.0|addslashes}}", "{{ row.1|addslashes}}",
{% endfor %}
```

使用了一个 for 循环。这里 data 与上面的 Context 的 data 相对应。因为 data 是一个列表，它的每行是一个 tuple ，因此 row.0, row.1 就是取 tuple 的第一个和第二个元素。| 是一个过滤符，它表示将前一个的处理结果作为输入传入下一个处理。因此 Django 的模板很强大，使用起来也非常直观和方便。 addslashes 是 Django 模板内置的过滤 Tag ，它用来将结果中的特殊字符加上反斜线。同时我们注意到，每个 {{}} 前后都有一个双引号，这样就保证每个字符串使用双引号引起来。然后在第一个与第二个元素之间还使用了逗号分隔。最后 endfor 在下一行，表示上面每行模板后有一个回车。Django 还允许你自定义 Tag ，在 The Django template language: For Python programmers 文档中有描述。

5 修改 urls.py

```
from django.conf.urls.defaults import *
urlpatterns = patterns('',
    # Example:
    # (r'^testit/', include('newtest.apps.foo.urls.foo')),
```

```
    (r'^$', 'newtest.helloworld.index'),
    (r'^add/$', 'newtest.add.index'),
    (r'^list/$', 'newtest.list.index'),
    (r'^csv/(?P<filename>\w+)/$', 'newtest.csv_test.output'),
    # Uncomment this for admin:
#     (r'^admin/', include('django.contrib.admin.urls')),
)
```

增加了 csv 的 url 映射。上面的正则表达式有些复杂了，因为有参数的处理在里面。(?P<filename>\w+) 这是一个将解析结果起名为 filename 的正则表达式，它完全符合 Python 正则表达式的用法。在最新的 Django 中，还可以简化一下：(\w+)。但这样需要你的参数是按顺序传入的，在一个方法有多个参数时一定要注意顺序。关于 url 的配置问题在 URL configuration 文档中有说明。还记得吗？我们的链接是写成 /csv/address/ ，因此上面实际上会变成对 csv.output(filename='address') 的调用。

6 启动 server:看一下结果吧。点击链接，浏览器会提示你保存文件的。很简单吧。但这里面的内容其实也不少，而且许多地方都有很大的扩展空间。

第五讲 session 和数据库

1 引言

现在我们就学习一下 session 吧。session 可以翻译为“会话”，做过 web 的可能都知道。它就是为了实现页面间的数据交换而产生的东西，一般有一个 session_id ，它会保存在浏览器的 cookie 中，因此如果你的浏览器禁止了 cookie ，下面的试验是做不了的。在 Django 中的 session 也非常简单，它就存在于 request 对象的 session 属性中。你可以把它看成一个字典就可以了。下面我们做一个非常简单的功能：首先当用户进入某个页面，这个页面会显示一个登录页面，上面有一个文本框用来输入用户名，还有一个提交按钮用来提交数据。当用户输入用户名，然

后点提交，则显示用户已经登录，并且打印出用户的姓名来，同时还提供一个“注销”按钮。然后如果用户再次进入这个页面，则显示同登录成功后的页面。如果点击注销则重新进入未登录的页面。

2 在 newtest 下创建 login.py

```
from django.http import HttpResponseRedirect
from django.shortcuts import render_to_response
def login(request):
    username = request.POST.get('username', None)
    if username:
        request.session['username'] = username
    username = request.session.get('username', None)
    if username:
        return render_to_response('login.html', {'username':username})
    else:
        return render_to_response('login.html')
def logout(request):
    try:
        del request.session['username']
    except KeyError:
        pass
    return HttpResponseRedirect("/login/")
```

两个方法： login() 和 logout()。login() 用来提供初始页面、处理提供数据和判断用户是否登录。而 logout() 只是用来从 session 中删除用户名，同时将页面重定向到 login 画面。这里我仍然使用了模板，并且根据传入不同的字典来控制模板的生成。是的，因为 Django 的模块支持条件判断，所以可以做到。

在 login() 中的判断逻辑是：

A. 先从 POST 中取 username (这样 username 需要由模板的 form 来提供)，如果存在则加入到 session 中去。加入 session 很简单，就是一个字典的 Key 赋值。

B. 然后再从 session 中取 username ，有两种可能：一种是上一步实现的。还有一种可能是直接从以前的 session 中取出来的，它不是新产生的。而这里并没有细分这两种

情况。因此这个判断其实对应两种页面请求的处理：一种是提交了用户姓名，而另一种则是处理完用户提交姓名之后，用户再次进入的情况。而用户再次进入时，由于我们在前面已经将他的名字保存在 session 里面了，因此可以直接取出来。如果 session 中存在，则表示用户已经登录过，则输出 login.html 模板，同时传入了 username 字典值。而如果 session 中不存在，说明用户从来没有登录过，则输出 login.html 模板，这次不带值。

因此对于同一个 login.html 模板传入的不同值，后面我们会看到模板是如何区分的。在 logout() 中很简单。先试着删除 session ，然后重定向页面到 login 页面。这里使用了 HttpResponseRedirect 方法，它是从以前我们看到的 HttpResponse 派生来的子类。更多的派生子类和关于 response 的内容要参考 Request and response objects 文档。

3 创建 templates/login.html

```
{% if not username %}
<form method="post" action="/login/">
   用户名：<input type="text" name="username" value=""><br/>

   <input type="submit" value="登录">
</form>
{% else %}
你已经登录了！{{ username }}<br/>
<form method="post" action="/logout/">
   <input type="submit" value="注销">
</form>
{% endif %}
```

整个是一个 if 语句。在 Django 模板中的 if 可以象 Python 一样使用，如使用 not，and，or 。象 if not username 表示什么呢？它表示如果 username 不存在，或为空，或是假值等等。而此时我们利用了 username 不存在这种判断。上面的逻辑表示，如果 username 不存在，则

显示一个表单，显示用户名输入文本框。如果存在，则显示已经登录信息，同时显示用户名和注销按钮。而这个注销铵钮对应于 logout() 方法。

4 修改 urls.py：增加了 login 和 logout 两个 url 映射。

```
from django.conf.urls.defaults import *
urlpatterns = patterns('',
    # Example:
    # (r'^testit/', include('newtest.apps.foo.urls.foo')),
    (r'^$', 'newtest.helloworld.index'),
    (r'^add/$', 'newtest.add.index'),
    (r'^list/$', 'newtest.list.index'),
    (r'^csv/(?P<filename>\w+)/$', 'newtest.csv_test.output'),
    (r'^login/$', 'newtest.login.login'),
    (r'^logout/$', 'newtest.login.logout'),
    # Uncomment this for admin:
     #(r'^admin/', include('django.contrib.admin.urls')),
)
```

5 启动 server 运行

但我要说，你一定会报错。而且我的也在报错。从这一刻起，我们就要进入有数据库的环境了。因为在 django 中 session 是存放在数据库中的。所以在这里要进行数据库的初始化了。

6 修改 settings.py，主要修改以下地方:

```
DATABASE_ENGINE = 'sqlite3'
DATABASE_NAME = './data.db'
DATABASE_USER = ''
DATABASE_PASSWORD = ''
DATABASE_HOST = ''
DATABASE_PORT = ''
```

这里我使用 sqlite3 。在使用数据库时，需要安装相应的数据库处理模块。对 sqlite3 只需要修改两项： DATABASE_ENGINE 和 DATABASE_NAME 。这里数据文件名我使用了相对路径，在实际情况下可能使用绝对路径为好。sqlite3 对应的是 pysqlite2 模块，从

http://pysqlite.org/ 下载到最新版本。如果你使用 utf-8，要注意缺省字符编码应为 utf-8。同时对于 DATABASE_PORT 字段，Django 目前允许使用数字或字符串表示了。

7　初始化数据库：改了配置还不够，还要执行相应的建库、建表的操作，使用 django-admin.py 或 manage.py 都可以:

```
manage.py syncdb
```

它可以自动创建已经安装的 App 中，在数据库中不存在的 Model。并且会自动将表的权限赋与超级用户。并且把超级用户的创建也结合到了一起。

8　启动 server，这次再进入试吧：http://localhost:8000/login/，从此我们要进入数据库的世界了，当然目前还没有用到，而 Django 提供的许多自动化的高级功能都是需要数据库支持的。

第六讲 wiki 的例子

1　引言

以后的例子可能会越来越复杂，下面我们按照 TurboGears 的 20 Minute Wiki Tutorial 的例子仿照一个，我们要用 Django 来做 wiki。我不会按 TurboGears 的操作去做，只是实现一个我认为的最简单的 wiki。现在我的要求是：做一个简单的 wiki，要可以修改当前页面，即在页面下面提供一个编辑的按钮。然后还要识别页面中的两个开头大写的单词为页面切换点，可以进入一个已经生成好的页面，或提示创建一个新页面。下面我们将开始创建 Django 中的 app 了。

先说一下。如果你看过官方版的教程，它就是讲述了一个 Poll 的 app 的生成过程。那么一个 app 就是一个功能的集合，它有自已的 model ，view 和相应的模板，还可以带自已的 urls.py 。那么它也是一个独立的目录，这样一个 app 就可以独立地进行安装，你可以把它安装到其它的

Django 服务器中去。因此采用 app 的组织形式非常有意义。而且 adango-admin.py 也提供了一个针对 app 的命令，一会我们就会看到。而且 Django 提供一些自动功能也完全是针对于 app 这种结构的。Model, Template, View 就合成了 MTV 这几个字母。 Model 是用来针对数据库，同时它可以用来自动生成管理界面， View 在前面我们一直都用它，用来处理请求和响应的相当于 MVC 框架中的 Controller 的作用， Template 用来生成界面。

2 创建 wiki app

```
manage.py startapp wiki
```

在 0.91 版， **app 都是放在 apps 目录下的**。不过到了 0.95 版，apps 目录不自动创建了。因此你就可以直接放在项目目录下了。这样在 wiki 子目录下有以下文件:

__init__.py 表示 wiki 目录是一个包。

views.py 用来放它的 view 的代码。

models.py 用来放 model 代码。

3 编辑 wiki/models.py

```
from django.db import models
# Create your models here.
class Wiki(models.Model):
    pagename = models.CharField(max_length=20, unique=True)
    content = models.TextField()
```

每个 model 其实在 Django 中就是一个表，你将用它来保存数据。在实际的应用中，一般都要与数据库打交道，如果不想用数据库，那么原因可能就是操作数据库麻烦，创建数据库环境也麻烦。但通过 Django 的 model 处理，它是一种 ORM (Object Relation Mapping, 对象与关系的映射)，可以屏蔽掉底层数据库的细节，同时提供以对象的形式来处理数据。非常方便。而且

Django 的 model 层支持多种数据库，如果你改变数据库不是问题，这也为以后的数据库迁移带来好处。

Wiki 是 model 的名字，它需要从 models.Model 派生而来。它定义了两个字段，一个是字段是 pagename ，用来保存 wiki 页面的名字，它有两个参数，一个是最大长度(从这点上不如 SQLAlchemy 方便, SQLAlchemy 并不需要长度，它会根据有无长度自动转为 TEXT 类型)，目前 CharField 需要这个参数；另一个是 unique 表示这个字段不能有重复值。还有一个字段是 content ，用来保存 wiki 页面的内容，它是一个 TextField 类型，它不需要最大长度。

在运行时， Django 会自动地为这个 model 增加许多数据操作的方法。关于 model 和 数据库操作 API 的详细内容参见 Model reference 和 Database API reference 的文档。

4 修改 settings.py, 安装 app

虽然我们的其它工作没有做完，但我还是想先安装一下 app 吧。每个 app 都需要安装一下。安装一般有两步：

1. 修改 settings.py

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'newtest.wiki',
)
```

2. 执行(在 newtest 目录下)

```
manage.py syncdb
```

以前是使用 install wiki 。现在也可以使用，不过使用 syncdb 要更简单得多。如果没有报错就是成功了。这一步 Django 将根据 model 的信息在数据库中创建相应的表。

5 在命令行下加入首页(FrontPage)

我们假设首页的名字为 FrontPage ，并且我们将在命令行下增加它，让我们熟悉一下命令行的使用，进入 newtest 目录，然后:

```
manage.py shell
```

进入 python

```
>>> from newtest.wiki.models import Wiki
>>> page = Wiki(pagename='FrontPage', content='Welcome to Easy Wiki')
>>> page.save()
>>> Wiki.objects.all()
[<Wiki object>]
>>> p = Wiki.objects.all()[0]
>>> p.pagename
'FrontPage'
>>> p.content
'Welcome to Easy Wiki'
```

因为在写这篇教程时是在 magic-removal 分枝下进行的操作，因此有些 API 并不稳定。象 objects 的方法以前是沿用 model 的方法，但后来进行了简化，比如 get_list() 变为 all() 。还有一系统的变化。具体的可参见 Removing The Magic 文档中关于 Descriptor fields 的说明。

在 Django 中，对于数据库的记录有两种操纵方式，一种是集合方式，一种是对象方式。集合方式相当于表级操作，在 0.92 中可以使用 model.objects 来处理。objects 对象有一些集合方式的操作，如 all() 会返回全部记录， filter() 会根据条件返回部分记录。而象插入新记录则需要使用记录方式来操作。

6 修改 wiki/views.py

```
#coding=utf-8
from newtest.wiki.models import Wiki
from django.template import loader, Context
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import render_to_response
def index(request, pagename=""):
    """显示正常页面，对页面的文字做特殊的链接处理"""
    if pagename:
        #查找是否已经存在页面
            #pages = Wiki.objects.get_list(pagename__exact=pagename)
        pages = Wiki.objects.filter(pagename=pagename)
        if pages:
            #存在则调用页面模板进行显示
            return process('wiki/page.html', pages[0])
        else:
            #不存在则进入编辑画面
            return render_to_response('wiki/edit.html',
{'pagename':pagename})
    else:
            #page = Wiki.objects.get_object(pagename__exact='FrontPage')
        page = Wiki.objects.get(pagename='FrontPage')
        return process('wiki/page.html',page)
def edit(request, pagename):
    """显示编辑存在页面"""
      #page = Wiki.objects.get_object(pagename__exact=pagename)
    page = Wiki.objects.get(pagename=pagename)
  return render_to_response('wiki/edit.html', {'pagename':pagename, 'content':page.content})
def save(request, pagename):
    """保存页面内容，老页面进行内容替换，新页面生成新记录"""
    content = request.POST['content']
      #pages = Wiki.objects.get_list(pagename__exact=pagename)
    pages = Wiki.objects.filter(pagename=pagename)
    if pages:
        pages[0].content = content
        pages[0].save()
    else:
        page = Wiki(pagename=pagename, content=content)
        page.save()
    return HttpResponseRedirect("/wiki/%s" % pagename)
import re
r = re.compile(r'\b(([A-Z]+[a-z]+){2,})\b')
```

```
def process(template, page):
    """处理页面链接，并且将回车符转为<br>"""
    t = loader.get_template(template)
    content = r.sub(r'<a href="/wiki/\1">\1</a>', page.content)
    content = re.sub(r'[\n\r]+', '<br>',content)
    c = Context({'pagename':page.pagename,'content':content})
    return HttpResponse(t.render(c))
```

将原来老的 model 方法加了注释，目前改用最新的 API 了。代码有些长，有些地方已经有说明和注释了。简单说一下：

- index() 用来显示一个 wiki 页面。它需要一个参数就是页面的名称。如果在数据库中找得到，则调用 process()方法(process()方法是一个自定义方法，主要用来对页面的文本进行处理，比如查找是否有满足 wiki 命名规则的单词，如果有则替换成链接。再有就是将回车转为
)。如果没有找到，则直接调用编辑模板显示一个编程页面。当然，这个页面的内容是空的。只是它的页面名字就是 pagename 。如果 pagename 为空，则进入 FrontPage 页面。 Wiki.objects 对象有 filter() 方法和 get() 方法，一个返回一个结果集，一个返回指定的对象。这里为什么使用 filter() 呢，因为一旦指定文件不存在，它并不是返回一个 None 对象，而是抛出异常，而我没有使用异常的处理方式。通过 filter() 如果存在则结果中应有一个元素，如果不存在则应该是一个 [] 。这样就知道是否有返回了。

filter() 中使用的参数与一般的 db-api 是一样的，但如果是比较相等，可以为: pagename__exact=pagename 也可以简化为 pagename=pagename 。在 Django 中，一些字段的比较操作比较特殊，它是在字段名后加 __ 然后是比较条件。这样看上去就是一个字符串。具体的参见 The Database API 。回车转换的工作其实可以在模板中使用 filter 来完成。

- 从对模板的使用 (wiki/edit.html) 可以猜到在后面我们要在 templates 中创建子目录了。的确，对于不同的 app ，我们可以考虑将所有的模板都放在统一的 templates 目录下，但为了

区分方便，一般都会针对 app 创建不同的子目录。当然也可以不这样，可以放在其它的地方，只要修改 settings.py ，将新的模板目录加进去就行了。

因为我们在设计 model 时已经设置了 pagename 必须是唯一的，因此一旦 filter() 有返回值，那它只能有一个元素，而 pages[0] 就是我们想要的对象。

- page = wikis.get(pagename='FrontPage')，是表示取出 pagename 为 FrontPage 的页面。你可能要说，为什么没有异常保护，是的，这也就是为什么我们要在前面先要插条记录在里面的原因。这样就不会出错了。再加上我要做的 wiki 不提供删除功能，因此不用担心会出现异常。
- edit() 用来显示一个编辑页面，它直接取出一个页面对象，然后调用 wiki/edit.html 模板进行显示。也许你还是要问，为什么不考虑异常，因为这里不会出现。为什么？因为 edit() 只用在已经存在的页面上，它将用于存在页面的修改。而对于不存在的页面是在 index() 中直接调用模板来处理，并没有直接使用这个 edit() 来处理。也许你认为这样可能不好，但由于在 edit() 要重新检索数据库，而在 index() 已经检索过一次了，没有必要再次检索，因此象我这样处理也没什么不好，效率可能要高一些。当然这只是个人意见。
- save() 用来在编辑页面时用来保存内容的。它先检查页面是否在数据库中存在，如果不存在则创建一个新的对象，并且保存。注意，在 Django 中，对对象处理之后只有调用它的 save() 方法才可以真正保存到数据库中去。如果页面已经存在，则更新页面的内容。处理之后再重定向到 index() 去显示这个页面。

7 在 templates 中创建 wiki 子目录

8 编辑 templates/wiki/page.html

```
<h2>{{ pagename }}</h2>
<p>{{ content }}</p>
```

```
<hr/>
<p>
<form method="POST" action="/wiki/{{ pagename }}/edit/">
<input type="submit" value="编辑">
</form></p>
```

它用来显示页面，同时提供一个“编辑”按钮。当点击这个按钮时将调用 view 中的 edit() 方法。

9 编辑 templates/wiki/edit.html

```
<h2>编辑:{{ pagename }}</h2>
<form method="POST" action="/wiki/{{pagename}}/save/">
<textarea name="content" rows="10"
cols="50">{{ content }}</textarea><br/>
<input type="submit" value="保存">
</form>
```

它用来显示一个编辑页面，同时提供“保存”按钮。点击了保存按钮之后，会调用 view 中的 save() 方法。

10 修改 urls.py

```
from django.conf.urls.defaults import *
urlpatterns = patterns('',
    # Example:
    # (r'^testit/', include('newtest.apps.foo.urls.foo')),
    (r'^$', 'newtest.helloworld.index'),
    (r'^add/$', 'newtest.add.index'),
    (r'^list/$', 'newtest.list.index'),
    (r'^csv/(?P<filename>\w+)/$', 'newtest.csv_test.output'),
    (r'^login/$', 'newtest.login.login'),
    (r'^logout/$', 'newtest.login.logout'),
    (r'^wiki/$', 'newtest.wiki.views.index'),
    (r'^wiki/(?P<pagename>\w+)/$', 'newtest.wiki.views.index'),
    (r'^wiki/(?P<pagename>\w+)/edit/$', 'newtest.wiki.views.edit'),
    (r'^wiki/(?P<pagename>\w+)/save/$', 'newtest.wiki.views.save'),
    # Uncomment this for admin:
    #(r'^admin/', include('django.contrib.admin.urls')),
)
```

增加了 wiki 等 4 个 url 映射。一般一个 wiki ，我们访问它的一个页面可能为：

wiki/pagename。我设计对 index() 方法的调用的 url 为:r'^wiki/(?P<pagename>\w+)/$'

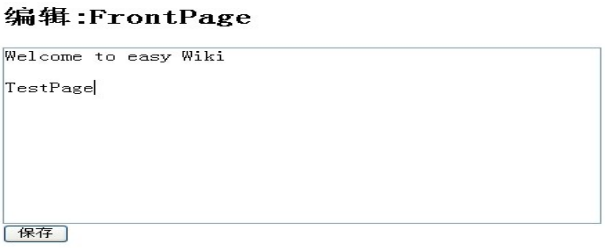
也就是把 wiki/后面的解析出来作为 pagename 参数。但这样就带来一个问题，如果我想实现 wiki/edit.html 表示修改， pagename 作为一个参数通过 POST 来提交好象就不行了。因为上面的解析规则会“吃”掉这种情况。因此我采用 Zope 的表示方法：把对象的方法放在对象的后面。我可以把 pagename 看成为一个对象， edit，save 是它的方法，放在它的后面，也简单，也清晰。当然如果我们加强上面的正则表达式，也可以解析出 wiki/edit.html 的情况。因此 wiki/pagename 就是显示一个页面，wiki/pagename/edit 就是编辑这个页面，wiki/pagename/save 就是保存页面。而 pagename 解析出来后就是分别与 index()，edit()，save() 的 pagename 参数相对应。

下面你可以运行了。

11 启动 server: manage.py runserver，进入 http://localhost:8000/wiki

首先进入这个页面：

然后你点编辑，则进入 FrontPage 的编辑界面：



然后我们加上一个 TestPage ，它符合 wiki 的名字要求，两个首字母大写的单词连在一起。然后点击保存。

看见了吧。页面上的 TestPage 有了链接。点击它将进入：

这是 TestPage 的编辑页面。让我们输入中文，然后输入 FrontPage 。然后保存。

好了，剩下的你来玩吧。点击 FrontPage 将回到首页。

第七讲 通讯录

1 引言

敢问路在何方，路在脚下。如果你坚持下来，一定会有收获的。直到目前我们已经学了：

- settings.py 的设置
- url dispatcher
- 模板
- session
- app
- model

其实在某些方面，使用 Django 还可以更加方便。而且我们还有许多东西没有学，一点点跟着我学吧。我有一个通讯录，它是保存在 Excel 文件中的，我不想每次到目录下去打开它，我希望用 Django 做一个 web 上的简单应用，如何做呢？

2 创建 address app

manage.py startapp address ；这样就创建好了 address 相关的目录了。

3 修改 address/models.py

```
#coding=utf-8
from django.db import models
# Create your models here.
class Address(models.Model):
    name = models.CharField('姓名', max_length=6, unique=True)

    gender = models.CharField('性别', choices=(('M', '男'), ('F', '女')),
        max_length=1)#, radio_admin=True)
    telphone = models.CharField('电话', max_length=20)

    mobile = models.CharField('手机', max_length=11)
```

这回 model 复杂多了。在上面你可以看到我定义了四个字段： name，gender，telpnone，mobile 。其中 gender 表示性别，它可以从一个 tuple 数据中进行选取。并且在后面的 radio_admin=True 表示在 admin 的管理界面中将使用 radio 按钮来处理。

Django 提供了许多的字段类型，有些字段类型从数据的取值范围来讲没有什么区别，但之所以有这种区别，是因为：Django 的数据类型不仅仅用于创建数据库，进行 ORM 处理，还用于 admin 的处理。一方面将用来对应不同的 UI 控件，另一方面提供对不同的数据类型将进行不同的数据校验的功能。在 Django 中每个字段都可以有一个提示文本，它是第一个参数，如果没有则会使用字段名。因此我定义的每个字段为了方便都有一个对应的汉字提示文本。

因为本节主要是讲 admin 的使用。admin 是 Django 提供的一个核心 app(既然是 app 就需要安装，一会就看到了)，它可以根据你的 model 来自动生成管理界面。我为什么要用它，因为有了这个管理界面，对于通讯录的增加、删除、修改的处理界面完全可以通过 admin 来自动生成，我不用自已写。不相信吗？我们就会看到了。那么 admin 到底可以带来些什么好处呢？它的功能很强大，不仅界面漂亮，还能对数据提供操作记录，提供搜索。特别是它是在用户权限控制之下，你都可以不用考虑安全的东西了。并且它本身就是一个非常好的学习的东西，特别是界面自动生成方面，学习它的代码可以用在我们自已的定制之中。当然，也许你用不上 admin ，它的确有一定的适应范围，不过对于大部分工作来说它可能足够了。对于那些交互性强的功能，你可能要自已实现许多东西，对于管理集中，主要以发布为主的东西，使用它可以节省你大量的时间。至于怎么使用，你要自已去权衡。但这一点对于快速实现一个 web 应用，作用非常大，这是 Django 中的一个亮点。

4 修改 settings.py

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.admin',
    'newtest.wiki',
    'newtest.address',
)
```

这里我们加入了两个 app ，一个是 address ，还有一个是 django.contrib.admin 。admin 也是一个应用，需要加入才行，后面还要按添加 app 的方式来修改 url 映射和安装 admin app。这些与标准的 app 的安装没有什么不同。

5 安装 admin app

manage.py syncdb (username：admin 3814264)，这样将在数据库中创建 admin 相关的表。

6 修改 urls.py

```
from django.conf.urls.defaults import *
urlpatterns = patterns('',
    # Example:
    # (r'^testit/', include('newtest.apps.foo.urls.foo')),
    (r'^$', 'newtest.helloworld.index'),
    (r'^add/$', 'newtest.add.index'),
    (r'^list/$', 'newtest.list.index'),
    (r'^csv/(?P<filename>\w+)/$', 'newtest.csv_test.output'),
    (r'^login/$', 'newtest.login.login'),
    (r'^logout/$', 'newtest.login.logout'),
    (r'^wiki/$', 'newtest.wiki.views.index'),
    (r'^wiki/(?P<pagename>\w+)/$', 'newtest.wiki.views.index'),
    (r'^wiki/(?P<pagename>\w+)/edit/$', 'newtest.wiki.views.edit'),
    (r'^wiki/(?P<pagename>\w+)/save/$', 'newtest.wiki.views.save'),
    # Uncomment this for admin:
    (r'^admin/', include('django.contrib.admin.urls')),
)
```

缺省的 urls.py 中在最后已经加上了 admin 的映射，不过是一个注释，把注释去掉就好了。这里要注意，它使用了一个 include 方式。对于这种 URL 的解析 Django 是分段的，先按 r'^admin/' 解析(这里没有 $)，匹配了则把剩下的部分丢给 django.contrib.admin.urls.admin 去进行进一步的解析。使用 include 可以方便移植，每个 app 都可以有独立的 urls.py ，然后可以与主 urls.py 合在一起使用。配置起来相对简单。而且可以自由地在主 urls.py 中修改应用 URL 的前缀，很方便。

7 增加超级用户：进入 http://localhost:8000/admin

admin 功能是有用户权限管理的，因此一个 admin 替你完成了大量的工作：用户的管理和信息的增加、删除、修改这类功能类似，开发繁琐的东西。那么我们目前还没有一个用户，因此可以在命令下创建一个超级用户，有了这个用户，以后就可以直接在 admin 界面中去管理了。

```
manage.py shell
>>> from django.contrib.auth.create_superuser import createsuperuser
>>> createsuperuser()
```

它会让你输入用户名，邮件地址和口令。如果你使用了 syncdb 的话，应该在运行的最后，当没有超级用户时会提示你创建的。因此这一步可能会省略掉。如果想直接创建可以使用这种方法。这种方法与 authentication 所讲述的不完全一致，原因就是这种方法不用设置 PYTHONPATH 和 DJANGO_SETTING_MODULE 环境变量，所以要简单一些。这回再进去看一下吧。

上面已经有一些东西了，其中就有用户管理。但如何通过 admin 增加通讯录呢？别急，我们需要在 model 文件中增加一些与 admin 相关的东西才可以使用 admin 来管理我们的 app 。因此是否启用 admin 管理取决于你。只要在 model 中增加 admin 相关的部分，我们的应用才可以在 admin 中被管理。

8 修改 address/models.py

```
#coding=utf-8
from django.db import models
```

```
# Create your models here.
class Address(models.Model):
    name = models.CharField('姓名', maxlength=6, unique=True)

    gender = models.CharField('性别',choices=(('M','男'),('F','女')),
        maxlength=1, radio_admin=True)
    telphone = models.CharField('电话', maxlength=20)

    mobile = models.CharField('手机', maxlength=11)
    class Admin: pass
```

在 0.91 版 Admin 内部类需要写成

```
class META:
    admin = meta.Admin()
```

在 0.95 版发生了变化。有了这个东西，你就可以在 admin 中看到 adress 这个 app 了。再到浏览器中看一下是什么样子了。

看见了吧。上面有增加和删除的按钮，先让我们点击一下增加吧。

这个自动生成的界面是不是很不错。增加一条保存起来了。不过我发现当我输入 limodou 时，只能输入 limodo 好象 u 输不进去。为什么？因为我把姓名按汉字算最多 6 个就够了，一旦我使用英文的名字可能就不够。因此这是一个问题，一会要改掉。

怎么新增的记录叫 <Address object>这样看上去很别扭。为什么会这样，因为没有定义特殊的方法。下面就让我们定义一下。

9 修改 address/models.py

```
#coding=utf-8
from django.db import models
# Create your models here.
class Address(models.Model):
    name = models.CharField('姓名',maxlength=6,unique=True)

    gender = models.CharField('性别', choices=(('M', '男'), ('F', '女')),
        maxlength=1, radio_admin=True)
    telphone = models.CharField('电话', maxlength=20)

    mobile = models.CharField('手机', maxlength=11)
    def __str__(self):
        return self.name
    class Admin: pass
```

改好了，再刷新下页面。这次看见了吗？增加了一个 __str__ 方法。这个方法将在显示 Address 实例的时候起作用。我们就使用某个联系人的姓名就行了。曾经在 0.91 使用 __repr__ 也可以，但后来只能使用 __str__ 了。要注意。

你记得吗？Model 是与数据库中的表对应的，为什么我们改了 model 代码，不需要重新对数据库进行处理呢？因为只要不涉及到表结构的调整是不用对表进行特殊处理的。不过，我们马上要修改表结构了。

10 修改 address/models.py：姓名留短了真是不方便，另外我突然发现需要再增加一个房间字段。

```
#coding=utf-8
from django.db import models
# Create your models here.
class Address(models.Model):
    name = models.CharField('姓名', maxlength=20, unique=True)

    gender = models.CharField('性别', choices=(('M', '男'), ('F', '女')),
        maxlength=1, radio_admin=True)
    telphone = models.CharField('电话', maxlength=20)

    mobile = models.CharField('手机', maxlength=11)

    room = models.CharField('房间', maxlength=10)
    def __repr__(self):
        return self.name
    class Admin: pass
```

这回表结构要改变了，怎么做呢？

11 修改表结构

目前 Django 没有一个特别的命令可以直接更新表结构。为什么呢？在 Django 看来修改表结构并不是件很容易的事情，主要的问题是数据库中现有的数据怎么办，因此为了使旧的数据可以平滑迁移到新的表结构中，这步操作还是手工来做好一些。但现在我们正在开发中，因此很有可能表结构要经常发生变化，每次手工做多麻烦呀。Django 有一个命令行命令： sqlreset 可以生成 drop 表，然后创建新表的 SQL 语句，因此我们可以先调用这个命令，然后通过管道直接导入数据库的命令行工具中。这里我使用的是 sqlite3 ，因此我这样做:

```
manage.py sqlreset address|sqlite3 data.db
```

sqlreset 后面是要处理的 app 的名字，因此它只会对指定的 app 有影响。但这样，这个 app 的所有数据都丢失了。如果想保留原有数据，你需要手工做数据切换的工作。另外 django-amdin.py 还提供了更为简单的命令 manage.py reset address ，效果同上面是一样。对于其它的数据库，在数据库命令行可能是不同的，这个你自已去掌握吧。同时对于 sqlite3 ，有人可能想：直接把数据库文件删除了不就行了。但是你一定要清楚，如果存在其它的 app 的话，它们的数据是否还有用，如果没用删除当然可以，不过相应的 app 都要再重新 install 一遍以便初始化相应的表。如果数据有用，这样做是非常危险的，因此还是象上面的处理为好，只影响当前的 app 。

12 进入 admin

我们可以再次进入 admin 了，增加，删除，修改数据了。用了一会，也许你会希望：能不能有汉化版本的界面呢？答案是肯定的，而且已做好了。

13 修改 settings.py

把 LANGUAGE_CODE 由 'en' 改为 'zh-cn' ， TIME_ZONE 建议改为 'CCT'，刷新下界面，是不是变成汉字了。国际化支持在 Django 中做得是非常的出色，程序可以国际化，模板可以国

际化，甚至 js 都可以国际化。这一点其它的类似框架都还做不到。而国际化的支持更是 RoR 的一个弱项，甚至在 Snakes and Rubies 的会议上，RoR 的作者都不想支持国际化。但 Django 却做得非常出色，目前已经有二十多种语言译文。在增加，删除，修改都做完了，其实还剩下什么呢？显示和查询。那么实现它则需要写 view 和使用模板了。这个其实也没什么，最简单的，从数据库里查询出所有的数据，然后调用模板，通过循环一条条地显示。不错是简单。但是在做之前，先让我们想一想，这种处理是不是最常见的处理方法呢？也许我们换成其它的应用也是相似的处理。如果很多这样的处理，是不是我们需要每次都做一遍呢？有没有通用的方便的方法。答案是：有！Django 已经为我们想到了，这就是 Generic views 所做的。它把最常见的显示列表，显示详细信息，增加，修改，删除对象这些处理都已经做好了一个通用的方法，一旦有类似的处理，可以直接使用，不用再重新开发了。但在配置上有特殊的要求。具体的可以看 Generic views 文档。从这里我有一点想法，我认为 view 这个名称特别容易让人产生误解，为什么呢？因为 view 可以译为视图，给人一种与展示有关的什么东西。但实际上 Django 中的 view 相当于一个 Controller 的作用，它是用来收集数据，调用模板，真正的显示是在模板中处理的。因此我倒认为使用 Controller 可能更合适，这样就称为 MTC 了。另外， Generic views 产生的意义在于 Django 的哲学理含 DRY (Don't repeat yourself, 不要自已重复)，目的是重用，减少重复劳动。还有其它的哲学理含参见 Design philosophies 文档。因此可以知道 view 可以省掉，但模板却不能省， Django 在这点上认为：每个应用的显示都可能是不同的，因此这件事需要用户来处理。但如果有最简单的封装，对于开发人员在测试时会更方便，但目前没有，因此模板我们还是要准备，而且还有特殊的要求，一会就看到了。

对于目前我这个简单的应用来说，我只需要一个简单的列表显示功能即可，好在联系人的信息并不多可以在一行显示下。因此我要使用 django.views.generic.list_detail 模块来处理。

14 增加 address/urls.py，

对，我们为 address 应用增加了自已的 urls.py。

```
from django.conf.urls.defaults import *
from newtest.address.models import Address
info_dict = {
#    'model': Address,
   'queryset': Address.objects.all(),
}
urlpatterns = patterns('',
   (r'^/?$', 'django.views.generic.list_detail.object_list',info_dict),
)
```

info_dict 存放着 object_list 需要的参数，它是一个字典。不同的 generic view 方法需要不同的 info_dict 字典(这个变量你可以随便起名)。对于我们要调用的 object_list 它只要一个 queryset 值即可。但这个值需要一个 queryset 对象。因此在第二句从 newtest.address.models 中导入了 Address 。并且使用 Address.objects.all() 来得到一个全部记录的 queryset 。在 0.91 版，需要两个参数 app_name 和 module_name 。但在 0.95 版之后， module_name 取消了。代替为 model_name 的小写形式。而 info_dict 也变成了一个 model 值了。但最新的变化是 model 也不要了，取而代之的是 queryset 。这样会更方便。只是我的代码要改来改去的。

前面已经谈到：使用 generic view 只是减少了 view 的代码量，但对于模板仍然是必不可少的。因此要创建符合 generic view 要求的模板。主要是模板存放的位置和模板文件的名字。使用 object_list() 需要的模板文件名为： app_label/model_name_list.html ，这是缺省要查找的模板名。

15 创建 templates/address 目录

16 创建 templates/address/address_list.html

```
<h1>通讯录</h1>
<hr>
<table border="1">
<tr>
 <th>姓名</th>
```

```
 <th>性别</th>

 <th>电话</th>

 <th>手机</th>

 <th>房间</th>
</tr>
{% for person in object_list %}
<tr>
 <td>{{ person.name }}</td>
 <td>{{ person.gender }}</td>
 <td>{{ person.telphone }}</td>
 <td>{{ person.mobile }}</td>
 <td>{{ person.room }}</td>
</tr>
{% endfor %}
</table>
```

17 修改 urls.py，将我们的应用的 urls.py include 进去。

```
from django.conf.urls.defaults import *
urlpatterns = patterns('',
    # Example:
    # (r'^testit/', include('newtest.apps.foo.urls.foo')),
    (r'^$', 'newtest.helloworld.index'),
    (r'^add/$', 'newtest.add.index'),
    (r'^list/$', 'newtest.list.index'),
    (r'^csv/(?P<filename>\w+)/$', 'newtest.csv_test.output'),
    (r'^login/$', 'newtest.login.login'),
    (r'^logout/$', 'newtest.login.logout'),
    (r'^wiki/$', 'newtest.wiki.views.index'),
    (r'^wiki/(?P<pagename>\w+)/$', 'newtest.wiki.views.index'),
    (r'^wiki/(?P<pagename>\w+)/edit/$', 'newtest.wiki.views.edit'),
    (r'^wiki/(?P<pagename>\w+)/save/$', 'newtest.wiki.views.save'),
    (r'^address/', include('newtest.address.urls')),
    # Uncomment this for admin:
     (r'^admin/', include('django.contrib.admin.urls')),
)
```

可以看到 r'^address/' 没有使用 $ ，因为它只匹配前部分，后面的留给 address 中的 urls.py 来处理。

18 启动 server 看效果

通讯录

姓名	性别	电话	手机	房间
limodou	M	1111	11111111111	1111

第八讲 为通讯录增加文件导入和导出功能

1 引言

上一讲的确很长，但如果看代码你会发现，代码主要在 model 的调整中，``urls.py`` 的工作不多，而连一行 view 的代码都没有写。是不是非常方便呢！那么让我们来继续完善这个通讯录吧。现在我想完成的是：增加批量导入和导出功能.为什么要批量导入呢？因为一般情况下，我一定是已经有了一个通讯录文件(象以前我说过的 Excel 文件)，那么现在需要转到 web 上来，难道要我一条条全部手工录入吗？能不能上传文件，自动插入到数据库中去呢？那么就让我们实现一个文件上传的处理吧。为了简化，我采用 csv 格式文本文件(这个文件在 svn 中有一个例子 data.csv ，不然就自行生成好了)。

```
abc,M,11,11,11,
bcd,M,11,11,11,
ass,M,11,11,11,
dfsdf,F,11,11,11,
sfas,F,11,11,11,
```

2 修改 templates/address/address_list.html

```
<h1 id="title">通讯录</h1>

<hr>
<form enctype="multipart/form-data" method="POST"
action="/address/upload/">
```

```
上传通讯录文件： <input type="file" name="file"/><br/>

<input type="submit" value="上传文件"/>
</form>
<table border="1">
<tr>
 <th>姓名</th>

 <th>性别</th>

 <th>电话</th>

 <th>手机</th>

 <th>房间</th>
</tr>
{% for person in object_list %}
<tr>
 <td>{{ person.name }}</td>
 <td>{{ person.gender }}</td>
 <td>{{ person.telphone }}</td>
 <td>{{ person.mobile }}</td>
 <td>{{ person.room }}</td>
</tr>
{% endfor %}
</table>
```

3 修改 address/views.py

```
#coding=utf-8
# Create your views here.
from newtest.address.models import Address
from django.http import HttpResponseRedirect
from django.shortcuts import render_to_response
def upload(request):
    file_obj = request.FILES.get('file', None)
    if file_obj:
        import csv
        import StringIO
        buf = StringIO.StringIO(file_obj['content'])
        try:
            reader = csv.reader(buf)
        except:
            return render_to_response('address/error.html',
```

```
            {'message':'你需要上传一个 csv 格式的文件！'})
        for row in reader:
#           objs = Address.objects.get_list(name__exact=row[0])
            objs = Address.objects.filter(name=row[0])
            if not objs:
                obj = Address(name=row[0], gender=row[1],
                    telphone=row[2], mobile=row[3], room=row[4])
            else:
                obj = objs[0]
                obj.gender = row[1]
                obj.telphone = row[2]
                obj.mobile = row[3]
                obj.room = row[4]
            obj.save()
        return HttpResponseRedirect('/address/')
    else:
        return render_to_response('address/error.html', {'message':'你需要上传一个文件！'})
```

这里有一个 upload() 方法，它将使用 csv 模块来处理上传的 csv 文件。首先查找姓名是否存在于数据库中，如果不存在则创建新记录。如果存在则进行替换。如果没有指定文件直接上传，则报告一个错误。如果解析 csv 文件出错，则也报告一个错误。报造错误使用了一个名为 error 的模板，我们马上要创建。

4 创建 templates/error.html

```
<h2>出错</h2>
<p>{{ message }}</p>
<hr>
<p><a href="/address/">返回</a></p>
```

5 修改 address/urls.py，增加一个 upload 的 url 映射。

```
from django.conf.urls.defaults import *
from newtest.address.models import Address
info_dict = {
#    'model': Address,
    'queryset': Address.objects.all(),
}
```

```
urlpatterns = patterns('',
    (r'^/?$', 'django.views.generic.list_detail.object_list', info_dict),
    (r'^upload/$', 'address.views.upload'),
)
```

6 启动 server 测试

这样导入功能就做完了。那导出呢？很简单了，参考 csv 的例子去做就可以了。不过，并不全是这样，仍然有要修改的地方，比如 csv.html 模板，它因为写死了处理几个元素，因此需要改成一个循环处理。

7 修改 templates/csv.html

```
{% for row in data %}{% for i in row %}"{{ i|addslashes }}",{% endfor %}
{% endfor %}
```

将原来固定个数的输出改为循环处理。

8 修改 templates/address/address_list.html，增加生成导出的 csv 文件的链接

```
<h1 id="title">通讯录</h1>
<hr>
<form enctype="multipart/form-data" method="POST"
action="/address/upload/">
上传通讯录文件： <input type="file" name="file"/><br/>

<input type="submit" value="上传文件"/>
</form>
<hr>
<p><a href="/address/output/">导出为 csv 格式文件</a></p>
<table border="1">
<tr>
 <th>姓名</th>
```

```
 <th>性别</th>

 <th>电话</th>

 <th>手机</th>

 <th>房间</th>
</tr>
{% for person in object_list %}
<tr>
 <td>{{ person.name }}</td>
 <td>{{ person.gender }}</td>
 <td>{{ person.telphone }}</td>
 <td>{{ person.mobile }}</td>
 <td>{{ person.room }}</td>
</tr>
{% endfor %}
</table>
```

9 修改 apps/address/views.py

```
#coding=utf-8
# Create your views here.
from newtest.address.models import Address
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import render_to_response
from django.template import loader, Context
def upload(request):
    file_obj = request.FILES.get('file', None)
    if file_obj:
        import csv
        import StringIO
        buf = StringIO.StringIO(file_obj['content'])
        try:
            reader = csv.reader(buf)
        except:
            return render_to_response('address/error.html',
                {'message':'你需要上传一个 csv 格式的文件！'})
        for row in reader:
#            objs = Address.objects.get_list(name__exact=row[0])
            objs = Address.objects.filter(name=row[0])
            if not objs:
                obj = Address(name=row[0], gender=row[1],
                    telphone=row[2], mobile=row[3], room=row[4])
```

```
            else:
                obj = objs[0]
                obj.gender = row[1]
                obj.telphone = row[2]
                obj.mobile = row[3]
                obj.room = row[4]
            obj.save()
        return HttpResponseRedirect('/address/')
    else:
        return render_to_response('address/error.html',
            {'message':'你需要上传一个文件！'})

def output(request):
    response = HttpResponse(mimetype='text/csv')
    response['Content-Disposition'] = 'attachment; filename=%s' %
'address.csv'
    t = loader.get_template('csv.html')
#    objs = Address.objects.get_list()
    objs = Address.objects.all()
    d = []
    for o in objs:
        d.append((o.name, o.gender, o.telphone, o.mobile, o.room))
    c = Context({
        'data': d,
    })
    response.write(t.render(c))
    return response
```

在开始处增加了对 HttpResponse, loader, Context 的导入。然后增加了用于输出处理的 output() 方法。

10 修改 address/urls.py，增加了对 output 方法的 url 映射。

```
from django.conf.urls.defaults import *
from newtest.address.models import Address
info_dict = {
#    'model': Address,
    'queryset': Address.objects.all(),
}
urlpatterns = patterns('',
    (r'^/?$', 'django.views.generic.list_detail.object_list', info_dict),
    (r'^upload/$', 'address.views.upload'),
    (r'^output/$', 'address.views.output'),
)
```

11 启动 server 测试

第九讲通讯录的美化，使用嵌套模板，静态文件，分页处理等

1 引言

不知道大家有没有对这个通讯录感到厌烦了，希望没有，因为还有一些东西没有讲完呢。最让我感觉不满意的就是通讯录的显示了，的确很难看，希望可以美化一下。那么主要从这几方面：

- 对姓名进行排序
- 生成分页结果
- 增加 css 和一些图片

2 修改 address/models.py 实现排序

可以在 model 中增加一个叫 Meta 的内类，然后通过对其设置类属性可以用来控制 model 的模型属性。如我们想实现表的排序，可以在 Meta 中增加一个 ordering = ['name'] 的属性即可。它表示按 name 进行排序。它可以有多个字段。如果在字段前加'-'表示倒序。修改完毕在浏览器中看一下效果就知道了。在 0.91 版 Meta 为 META 。同时我们在前一讲看到的 Admin 也是原本在 META 下使用的。但在 0.95 版发生了改变。

3 修改 templates/address/address_list.html 实现分页显示

```
<h1 id="title">通讯录</h1>
<hr>
<div>
<table border="0" width="500">
<tr align="right">
  <td>{% if has_previous %}
    <a href="/address?page={{ previous }}">上一页</a>
```

```
   {% endif %} {% if has_next %}
   <a href="/address?page={{ next }}">下一页</a>
   {% endif %}</td></tr>
</table>

<table border="1" width="500">
<tr>
 <th>姓名</th>
 <th>性别</th>
 <th>电话</th>
 <th>手机</th>
 <th>房间</th>
</tr>
{% for person in object_list %}
<tr>
 <td>{{ person.name }}</td>
 <td>{{ person.gender }}</td>
 <td>{{ person.telphone }}</td>
 <td>{{ person.mobile }}</td>
 <td>{{ person.room }}</td>
</tr>
{% endfor %}
</table>
</div>
<table border="0" width="500">
<tr>
<td>
<form enctype="multipart/form-data" method="POST"
action="/address/upload/">
文件导入：<input type="file" name="file"/><br/>
<input type="submit" value="上传文件"/>
</form>
</td>
<td><p><a href="/address/output/">导出为 csv 文件</a></p></td>
</tr>
</table>
```

这时我仍然使用的是 generic view 来处理。但对布局作了简单的调整，将导入和导出的内容移到下面去了。同时增加了对分页的支持:

```
{% if has_previous %}
<a href="/address?page={{ previous }}">上一页</a>
{% endif %} {% if has_next %}
<a href="/address?page={{ next }}">下一页</a>
{% endif %}
```

在使用 generic view 的 **object_list** 时，它会根据 URL Dispatch 中是否设置了 paginate_by 这个参数来决定是否使用分页机制。一会我们会看到在 urls.py 的这个参数。一旦设置了这个参数，则 object_list 会使用 Django 提供的一个分页处理器来实现分页。它会自动产生分页所用到的许多的变量，这里我们使用了 has_previous, previous, has_next, next 这四个变量，还有其它一些变量可以使用。具体的参见 Generic views 文档。这里是根据是否有前一页和下一页来分别生成相应的链接。对于分页的链接，需要在 url 中增加一个 Query 关键字 page 。因此我的模板中会使用 page={{ previous }} 和 page={{ next }} 分别指向前一页和下一页的页码。

4 修改 address/urls.py

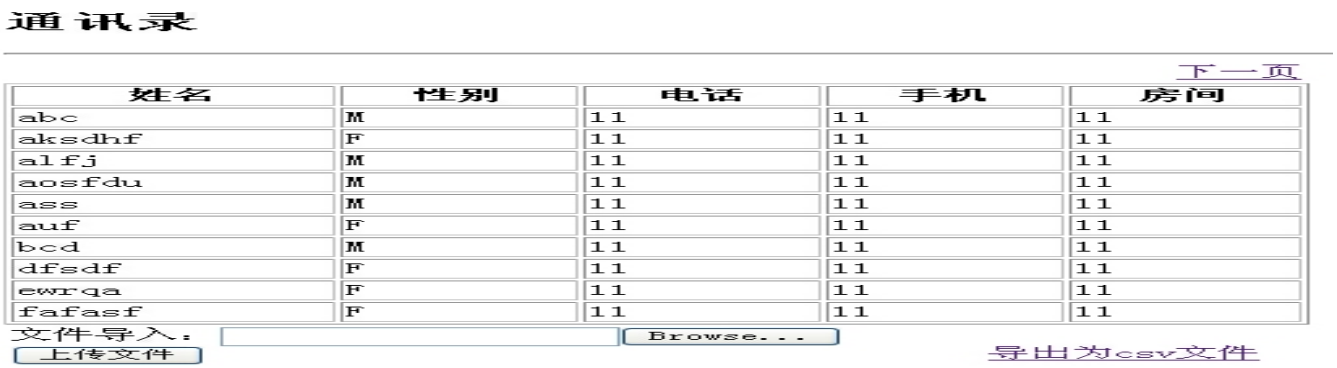
```
from django.conf.urls.defaults import *
from newtest.address.models import Address

info_dict = {
#    'model': Address,
   'queryset': Address.objects.all(),
}
urlpatterns = patterns('',
   (r'^/?$', 'django.views.generic.list_detail.object_list',
      dict(paginate_by=10, **info_dict)),
   (r'^upload/$', 'address.views.upload'),
   (r'^output/$', 'address.views.output'),
)
```

修改了原来传给的 object_list 的 info_dict 参数，这里设置每页的条数为 10 条:

```
dict(paginate_by=10, **info_dict)
```

这是将新的参数与原来的参数合成一个新的字典。

5 启动 server 测试:显示效果为



下面让我们为它添加一些 CSS 和图片，让它变得好看一些。首先要说明一下，我们一直处于开发和测试阶段，因此我们一直使用的都是 Django 自带的 server(其实我个人感觉这个 server 的速度也挺快的)，但最终我们的目的是把它部署到 Apache 上去。现在我们打算增加 CSS 和添加一些图片， Django 提供了这个能力，这就是对静态文件的支持，但是它只是建议在开发过程中使用。真正到了实际环境下，还是让专门的 web server 如 Apache 来做这些事情。只要改一下链接设置就好了。更详细的说明要参见 Serving static/media files 的文档。同时在 Django 中为了不让你依赖这个功能，特别在文档的开始有强烈的声明：使用这个方法是低效和不安全的。同时当 DEBUG 设置(在 settings.py 中有这个选项， True 表示处于调试期，会有一些特殊的功能)为 False 时，这个功能就自动无效了，除非你修改代码让它生效。

6 修改 urls.py

```
from django.conf.urls.defaults import *
from django.conf import settings
urlpatterns = patterns('',
    # Example:
```

```
    # (r'^testit/', include('newtest.apps.foo.urls.foo')),
    (r'^$', 'newtest.helloworld.index'),
    (r'^add/$', 'newtest.add.index'),
    (r'^list/$', 'newtest.list.index'),
    (r'^csv/(?P<filename>\w+)/$', 'newtest.csv_test.output'),
    (r'^login/$', 'newtest.login.login'),
    (r'^logout/$', 'newtest.login.logout'),
    (r'^wiki/$', 'newtest.wiki.views.index'),
    (r'^wiki/(?P<pagename>\w+)/$', 'newtest.wiki.views.index'),
    (r'^wiki/(?P<pagename>\w+)/edit/$', 'newtest.wiki.views.edit'),
    (r'^wiki/(?P<pagename>\w+)/save/$', 'newtest.wiki.views.save'),
    (r'^address/', include('newtest.address.urls')),
    (r'^site_media/(?P<path>.*)$', 'django.views.static.serve',
        {'document_root': settings.STATIC_PATH}),
    # Uncomment this for admin:
     (r'^admin/', include('django.contrib.admin.urls')),
)
```

你会看到 site_media 就是我将用来存放 CSS 和图片的地方。 django.views.static.serve 需要一个 document_root 的参数，这里我使用了一个 STATIC_PATH ，它从哪里来呢？它是我自已在 settings.py 中定义的。在前面有一个导入语句:

```
from django.conf import settings
```

从这里可以看到是如何使用 settings.py 的，我们完全可以自已定义新的东西，并让它在整个项目中生效。

7 修改 settings.py，在最后增加:

```
STATIC_PATH = './media'
```

那么我需要在 newtest 目录下创建一个 media 的目录。

8 创建 newtest/media 目录

这样根据上面 urls.py 的设置，我们以后将通过 /site_media/XXX 来使用某些静态文件。为了美化，我想需要一个 CSS 文件来定义一些样式，同时我还想提供一个 Django Powered 的图

片。在这里有官方提供的图标。于是我下了一个放在了 media 目录下。同时 CSS 怎么办，自已重头写，太麻烦，反正只是一个测试。于是我下载了 Django 站点用的 css 叫 base.css 也放在了 media 下面。下面就是对模板的改造。在 SVN 中我放了一个 css 和 gif 图片大家可以使用，不然可能看不出效果。为了通用化，我新增了一个 base.html 它是一个框架，而以前的 address_list.html 是它的一个子模板。这样我们就可以了解如何使用模板间的嵌套了。

9 创建 templates/base.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
   <head>
      <title>Address</title>
      <link href="/site_media/base.css" rel="stylesheet" type="text/css" media="screen" />
   </head>
   <body>
      <div id="container">
         {% block content %}content{% endblock %}
      </div>
      <div id="footer">
         <div>
         <img src="/site_media/djangopowered.gif"/>
         </div>
         <p>&copy; 2005 Limodou. Django is a registered trademark of
Lawrence Journal-World.</p>
      </div>
   </body>
</html>
```

有些代码也是从 Django 的网页中拷贝来的。特别要注意的是:

```
{% block content %}content{% endblock %}
```

这样就是定了一个可以扩展的模块变量块，我们将在 address_list.html 中扩展它。同时对 CSS 和 Django-Powered 的图片引用的代码是:

```
<link href="/site_media/base.css" rel="stylesheet" type="text/css" media="screen" />
<img src="/site_media/djangopowered.gif"/>
```

前面都是从 site_media 开始的。这样就将使用我们前面在 urls.py 中的设置了。

10 修改 templates/address/address_list.html

```
{% extends "base.html" %}
{% block content %}
<style type="text/css">
h1#title {color:white;}
</style>
<div id="header">
<h1 id="title">通讯录</h1>
</div>
<hr>
<div id="content-main">
  <table border="0" width="500">
  <tr align="right">
    <td>{% if has_previous %}
      <a href="/address?page={{ previous }}">上一页</a>
      {% endif %} {% if has_next %}
      <a href="/address?page={{ next }}">下一页</a>
      {% endif %}</td></tr>
  </table>
  <table border="1" width="500">
  <tr>
    <th>姓名</th>
    <th>性别</th>
    <th>电话</th>
    <th>手机</th>
    <th>房间</th>
  </tr>
  {% for person in object_list %}
  <tr>
    <td>{{ person.name }}</td>
    <td>{{ person.gender }}</td>
    <td>{{ person.telphone }}</td>
    <td>{{ person.mobile }}</td>
    <td>{{ person.room }}</td>
```

```
    </tr>
    {% endfor %}
    </table>
    <table border="0" width="500">
    <tr>
    <td>
    <form enctype="multipart/form-data" method="POST"
action="/address/upload/">
    文件导入：<input type="file" name="file"/><br/>

    <input type="submit" value="上传文件"/>
    </form>
    </td>
    <td><p><a href="/address/output/">导出为 csv 文件</a></p></td>
    </tr>
    </table>
</div>
{% endblock %}
```

基本上没有太大的变化，主要是增加了一些 div 标签，同时最开始使用:

```
{% extends "base" %}
```

表示是对 base 的扩展，然后是相应的块的定义:

```
{% block content %}
...
{% endblock %}
```

所有扩展的东西一定要写在块语句的里面，一旦写到了外面，那样就不起作用了。Django 的模板可以不止一次的扩展，但这里没有演示。

11 启动 server 测试，现在你看到的页面是不是象我这样？

1 引言

首先让我们说一说 media 链接吧。在上一讲中我使用了 site_media 作为静态文件的起始目录。但你知道吗，原来我想使用的是 media ，但为什么又改了呢？原因就是： admin 给占了。如果我使用 media ， Django 会指向 admin 的 media 目录，这可不是你想要的。因此这一点要特别提醒。现在我们看一看所展示出来的页面，你满意吗？还有可以改进的地方。比如性别，它显示出来的直接是数据库的值，而不是对应的“男”，“女”，怎么办。还有表格显示也不是很好看。没说的，改！最初我想使用 CustomManipulator (Manipulator 是 Django 中用来自动生成元素对应的 HTML 代码的对象，你可以定制它)，但使用 Manipulator 的话，你不能再使用 generic view 了，需要自已去实现 generic view 的某些代码，当然可以 copy and paste，但我目前不想那样做。于是我想到可以扩展 Django 的模板，自定义一个 filter 来实现它。(具体扩展的文档参见 The Django template language: For Python programmers ，你不仅可以扩展 filter，还可以扩展 Tag ，还可以设置模板变量，还可以进行块处理等复杂的操作。)

2 创建 address/templatetags 目录：注意，这个目录要在某个应用的下面，同时它应与 models, views.py 在同一层目录下。

3 创建 address/templatetags/__init__.py 文件,文件为空即可。

4 创建自定义模板文件 change_gender.py

文件名为想要装入到模板中的名字。如文件名为 change_gender.py ，那么可以在模板中使用:

```
{% load change_gender %}
```

来导入。

5 编辑 change_gender.py

```
#coding=utf-8
from django import template
register = template.Library()
#@register.filter(name='change_gender')
def change_gender(value):
    if value == 'M':
        return '男'
    else:
        return '女'
register.filter('change_gender',change_gender)
```

先是导入 template 模块，然后生成一个 register 的对象，我将用来它注册我所定义的 filter 。我实现的 filter 将命名为 "change_gender" ，它没有参数(一个 filter 可以接受一个参数或没有参数)。当 value 为 M 时返回 男 ，当 value 为 F 时返回 女 。然后调用 register 的 filter 来注册它。这里有两种写法，一种是使用 Python 2.4 才支持的 decorator (此行注释掉了)，另一种是使用标准的写法。在使用 decorator 时，如果 filter 方法有多个参数的话，需要指明 name 参数，否则可以直接写为:

```
@register.filter
```

它自动将函数名认为是 filter 的名字。象 decorator(*@register.filter*) 这样的用法要在 Python 2.4 中才可以使用，因此如果你的代码也允许在 2.3 上运行的话，不要使用这样的用法。而改用传统的在函数定义之后重定义的方法。就象上面所做的一样。同时还要注意避免使用一些 2.4 的内置函数和语法，如 enumerate，generator 产生式之类的东西。因此上面我使用的是 2.3 的方式。

6 修改 templates/address/address_list.html

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

```
{% load change_gender %}
<style type="text/css">
h1#title {color:white;}
.mytr1 {background:#D9F9D0}
.mytr2 {background:#C1F8BA}
.myth {background:#003333}
.th_text {color:#ffffff}
</style>
<div id="header">

<h1 id="title">通讯录</h1>

</div>
<hr>
<div id="content-main">
   <table border="0" width="500">
   <tr align="right">
     <td>{% if has_previous %}

       <a href="/address?page={{ previous }}">上一页</a>

       {% endif %} {% if has_next %}

       <a href="/address?page={{ next }}">下一页</a>

       {% endif %}</td></tr>
   </table>
   <table border="0" width="500" cellspacing="2">
   <tr class="myth">

     <th><span class="th_text">姓名</span></th>

     <th><span class="th_text">性别</span></th>

     <th><span class="th_text">电话</span></th>

     <th><span class="th_text">手机</span></th>

     <th><span class="th_text">房间</span></th>

   </tr>
   {% for person in object_list %}
   <tr class="{% cycle mytr1,mytr2 %}">
     <td>{{ person.name }}</td>
     <td>{{ person.gender|change_gender }}</td>
     <td>{{ person.telphone }}</td>
     <td>{{ person.mobile }}</td>
     <td>{{ person.room }}</td>
   </tr>
   {% endfor %}
   </table>
```

```
<table border="0" width="500">
<tr>
<td>
<form enctype="multipart/form-data" method="POST"
action="/address/upload/">
文件导入：<input type="file" name="file"/><br/>

<input type="submit" value="上传文件"/>
</form>
</td>
<td><p><a href="/address/output/">导出为 csv 文件</a></p></td>
</tr>
</table>
</div>
{% endblock %}
```

改动了以下几个地方：

1. 增加了 {% load change_gender %} 来导入自定义的 filter 。
2. 增加了几个样式，象 mytr1, mytr2 等。
3. 显示结果的 table 改为:
4. <table border="0" width="500" cellspacing="2">
5. 表头改为:
6. <tr class="myth">
7. <th>姓名</th>
8. <th>性别</th>
9. <th>电话</th>
10. <th>手机</th>
11. <th>房间</th>
12. </tr>

增加了样式处理

13. 数据显示的 tr 标签改为:

```
14.  <tr class="{% cycle mytr1,mytr2 %}">
```

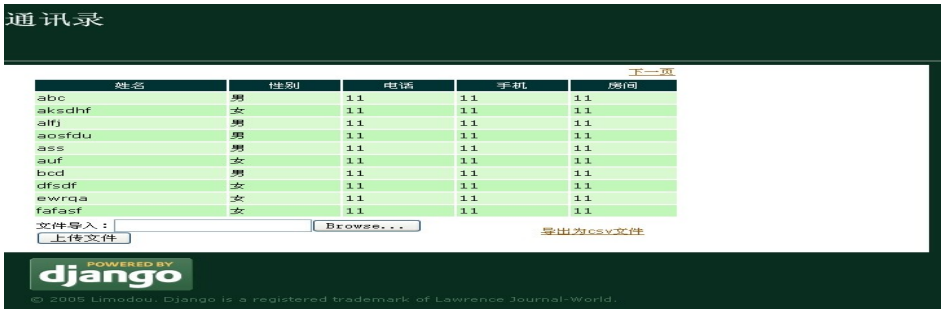
使用了 cycle Tag 来处理表格行的样式切换。注意： cycle 处理的是字符串。

15. 修改 {{ person.gender }} 为 {{ person.gender|change_gender }}

7 启动 server 进行测试

注意，一定要重启。象 templatetags 之类是在导入时处理的，因此如果 server 已经启动再添加的话是不起作用的。其它象增加 app, 修改 settings.py 都是要重启，而修改 urls.py，view, model 代码，模板什么的可以不用重启，在必要时 Django 的测试 web server 会自动重启。如果你使用 Apache 的话，估计绝大多数情况下要重启，可能只有修改模板不用吧。不过也仍然可以设置 Apache 以便让每次请求过来时重新装入 Python 模块。如果一切成功，你会看到 M, F 都改过来了。这里如果你感兴趣还可以改成小图标来表示，点缀一下。

效果画面为：



不过，在写完了这么多之后， Django 的 Model 还提供了一个方便的 db-api 专门来转换有 choices 的字段，比如你的字段叫 foo ，它有 choices 参数，那么可以使用 record (某个记录对象) 的 get_foo_display() 来得到对应的显示字符串。因此上，你可以不用写自已的 filter 就可以完成转换。象上面的模板中，只要将:{{ person.gender|change_gender }}

改为:{{ person.get_gender_display }}，同时将:{% load change_gender %}，去掉即可。这个方法很方便，你可以自已去试一试。不过，上面主要演示了自定义 filter 的实现。

第十一讲 用户管理和使用 authentication 来限制用户的行为

1 引言

我们再仔细看一下这个通讯录，我们知道，如果想增加新的记录，一种方法是通过 admin 界面，这个已经由 Django 自动为我们做好了。我们还可以批量导入，这个是我们实现的。但是这里有风险，为什么？如果什么人都可以导入这可是件不好的事： **加权限控制** 。Django 自带了一个权限控制系统，那么我们就用它。此先让我们简单地了解一下 Django 中的权限。同时我希望只有特殊权限的人才可以做这件事情，我们一直使用超级用户，但这并不是个好的习惯。因此让我们先创建个个人用户吧。

2 添加一个个人用户

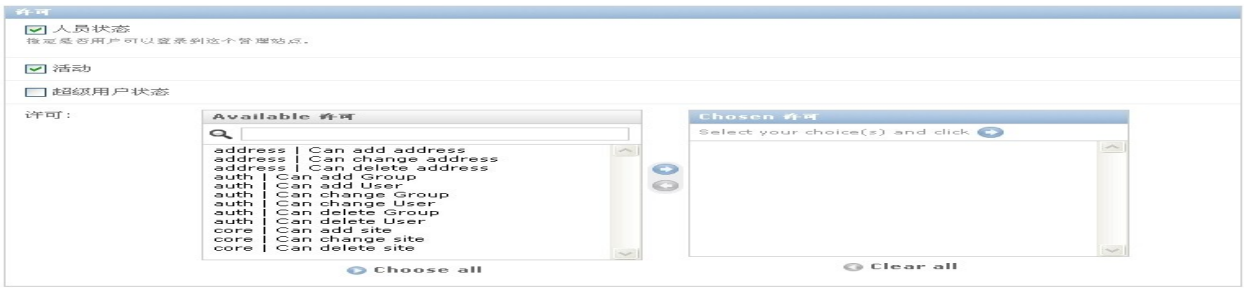
使用 admin 用户进入管理界面 http://localhost:8000/admin,在 Auth 下有用户一项，点击添加按钮进入添加界面，还挺复杂的。在这里提示是黑体的字段是必输项，其实只有两项是需要我们输的：用户名和口令。用户名好办，口令怎么还有格式呢：

格式为： '[algo]$[salt]$[hexdigest]'

这里 algo 是算法的名字，可以是 md5 或 sha1 算法。 salt 是一个随机数，它将用来参与密码信息的生成。 hexdigest 是将 salt +原始的密码 计算它的摘要算法得出来的东西。从 User authentication in Django 文档来看，并没有仔细地解释这个事情。我们需要这样做吗？但实际的情况要好，也要复杂的多。我们其实并不一定需要这样做， Django 在做口令检查时，一旦发现口令串不是组织成以 $ 分隔的形式，它会先认为是 md5 算出的结果，然后如果比较成功则自动使用 sha1 重新计算，然后保存到数据库中去。而这一过程是自动进行的。因此，最简单的就是按文档上那样，使用 Python 来生成一个 md5 的口令摘要码，如:

```
>>> import md5
>>> md5.new('test').hexdigest()
'098f6bcd4621d373cade4e832627b4f6'
```

上面就生成了一个口令为 test 的 md5 的摘要码。然后把它拷贝到输入口令的地方即可。然后在我们第一次成功验录后， Django 会自动替我们改成三段的格式，而我们不需要知道。不过，对于一般户其实不用担心，因为他们没有机会创建自已的用户，这一切都是管理员的工作，一般用户只是在管理员设定好口令之后，他们登录，然后可以修改自已的口令。所以真正麻烦的是管理员。我不知道 Django 为什么会这样，只是看到邮件列表中的确有人在讨论这个问题，以后再关注吧。知道了口令应该如何生成(md5 计算)，那么我们只要填入用户名，口令就行了。



人员状态检查框如果不打勾，则你的用户也无法使用，因为他不能登录。也许你担心，如果打勾了，那不是他就能做好多事了吗？其实不然。在 Django 中，创建一个 app 之后都有一些基本的权限会自动生成，而这些除了超级用户，它们是不会自动赋给某个用户的。因此如果管理员不给某个用户关于 app 的使用权限，那么这个用户根本没有办法操纵这些 app ，甚至连看都看不到(大家自已试一下就知道了)。这样他能够做的只是登录，但这也许就够了，有时我们需要的就是一个用户的合法身份，而不是一定要他能做些什么。request 对象提供一个 user 对象，你可以根据它来判断当前用户的身份，所属的组，所拥有的权限。我们可以在 view 代码中进行用户身份的检

查。现在我的想法是：限制特殊用户来做这件事。首先我可以在 settings.py 中设定这个用户名，然后在 view 中检查当前用户是否是 settings.py 中设定的用户。

3 修改 settings.py

在最后增加:

```
UPLOAD_USER = 'limodou'
```

这里请把 limodou 改成你想要的名字。要注意，在后面的测试中你需要按这里指定的名字创建一个用户。

4 修改 address/views.py

```
#...
from django.conf import settings

def upload(request):
    if request.user.username != settings.UPLOAD_USER:
        return render_to_response('address/error.html',
            {'message':'你需要使用 %s 来登录！' % settings.UPLOAD_USER})
#...
```

我们从 django.conf 导出了 settings ，然后在 upload() 中判断当前用户名是否是等于 settings.UPLOAD_USER 这个用户名，如果不是则提示出错信息。否则继续处理。好象一切都挺简单，但这里还有一个大问题：能不能自动导向一个用户注册的页面去呢？上面的处理是需要用户进入 admin 管理界面进行注册后，再进行操作。如果没有注册就上传文件，则只会报错。这里我希望实现：如果用户没有注册过，自动显示一个注册页面。如何做呢？文档中提出了一个方法:

```
from django.views.decorators.auth import login_required
@login_required
def my_view(request):
    # ...
```

这个方法我试过了，但失败了。主要的原因是：如果你还没有注册，它会自动导向 /accounts/login/ ，而这个 URL 目前是不存在的。在我分析了 login.py 代码之后，我认为它只是一个框架，并不存在 Django 已经提供好的模板可以直接使用，如果要使用它是不是需要我自已去建一个可以用的模板？没办法，我分析了 admin 的代码之后，最终找到了一种替代的方法:

```
from django.contrib.admin.views.decorators import staff_member_required
@staff_member_required
def upload(request):
```

admin 已经提供了这样的一个方法： staff_member_required 。它允许我使用 admin 的登录画面。注意 @staff_member_required 是 2.4 中的 decorator 的用法。如果希望在 2.3 上也可以运行，请改一下。一旦把上面的代码补充完整，代码是这样的:

```
#coding=utf-8
# Create your views here.
from newtest.address.models import Address
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import render_to_response
from django.template import loader, Context
from django.conf import settings
from django.contrib.admin.views.decorators import staff_member_required
@staff_member_required
def upload(request):
    if request.user.username != settings.UPLOAD_USER:
        return render_to_response('address/error.html',
            {'message':'你需要使用 %s 来登录！' % settings.UPLOAD_USER})
    file_obj = request.FILES.get('file', None)
    if file_obj:
        import csv
        import StringIO
        buf = StringIO.StringIO(file_obj['content'])
        try:
            reader = csv.reader(buf)
        except:
            return render_to_response('address/error.html',
                {'message':'你需要上传一个 csv 格式的文件！'})
        for row in reader:
#            objs = Address.objects.get_list(name__exact=row[0])
```

```
            objs = Address.objects.filter(name=row[0])
            if not objs:
                obj = Address(name=row[0], gender=row[1],
                    telphone=row[2], mobile=row[3], room=row[4])
            else:
                obj = objs[0]
                obj.gender = row[1]
                obj.telphone = row[2]
                obj.mobile = row[3]
                obj.room = row[4]
            obj.save()
        return HttpResponseRedirect('/address/')
    else:
        return render_to_response('address/error.html',
            {'message':'你需要上传一个文件！'})

def output(request):
    response = HttpResponse(mimetype='text/csv')
    response['Content-Disposition'] = 'attachment; filename=%s' %
'address.csv'
    t = loader.get_template('csv.html')
#    objs = Address.objects.get_list()
    objs = Address.objects.all()
    d = []
    for o in objs:
        d.append((o.name, o.gender, o.telphone, o.mobile, o.room))
    c = Context({
        'data': d,
    })
    response.write(t.render(c))
    return response
```

基本没有变化，主要是开始的一些地方增加了用户权限的处理。

5 启动 server 测试

在点击上传之后，如果没有注册会进入登录画面。如果已经注册，但用户名不对，则提示一个出错信息。不过，一旦注册出错，没有提供自动重新登录的功能，因此你需要进入 admin 管理地址，然后注销当前用户，再重新上传或先用正确的用户登录。因为是个简单的 app ，没必要做得那么完善。同时还存在的一个问题是，如果你没有注册过，那么点击上传按钮后，将进入登录画面，

但如果成功，你上传的文件将失效，需要重新再上传。那么解决这个问题的一个好方法就是：不要直接显示上传的东西，而是先提供一个链接或按钮，认证通过后，再提供上传的页面，这样可能更好一些。在 User authentication in Django 文档中还有许多的内容，如权限，在模板中如何使用与认证相关的变量，用户消息等内容。

第十二讲搜索功能的实现和 Apache 上的部署

1 引言

如果通讯录中的记录很多，我希望有一种搜索的方法，下面就让我们加一个搜索功能吧。当然，这个搜索功能是很简单的。在 Django 邮件列表中看到 WorldOnline(好象是它)有一个搜索的框架，可以定义哪些模块的哪些字段要参加搜索。这样在处理时会自动将相应的信息加入到搜索数据库中进行预处理。现在这个框架并没有开放源码，而且它底层使用的搜索的东西并不是 Django 本身的。这里我只是对姓名字段进行查找。

2 修改 templates/address/address_list.html

```
[...]
<hr>
<div id="content-main">
   <table border="0" width="500">
     <tr align="right"><td>
   <form method="GET" action="/address/search/">
   搜索姓名：<input name="search" type="text"
value="{{ searchvalue }}"/>
   <input type="submit" value="提交"/>
     </form>
   </td></tr>
   </table>
   <table border="0" width="500">
   <tr align="right">
     <td>{% if has_previous %}
[...]
```

在显示分页的代码上面增加了搜索的处理。从上面可以看到，条件输入处我增加了一个 searchvalue 的变量，希望在提交一个搜索后，显示页面的同时显示当前显示时使用的条件。这里存在一个困难：如何把搜索条件，搜索字符串与通用 view 相关联呢？只要我们生成正确的 queryset(结果集) 即可。但这个结果集需要查询姓名为指定名称的记录，如何实现呢？在以前 object_list 可以传入 extra_lookup_kwargs 参数，但后来由于使用了 queryset ，则这个参数不再需要了。通过 urls.py 我想是不行的，因为它只从 url 解析，而且对于 QUERY_STRING 是不进行解析的(QUERY_STRING 是指： http://example.com/add/?name=test 中？后面的东西，也就是 name=test)。对于搜索条件，我会使用一个 form 来处理， method 会设为 GET ，因此生成的 url 中，查询条件正如这个例子，如： http://localhost:8000/address/search/?search=limodou 。这样无法变成上面所要用到的参数。因此我决定自定义一个新的 view 方法。

3 修改 address/views.py

```
from django.views.generic.list_detail import object_list
def search(request):
    name = request.REQUEST['search']
    if name:
        extra_lookup_kwargs = {'name__icontains':name}
        extra_context = {'searchvalue':name}
#       return object_list(request, Address,
#           paginate_by=10, extra_context=extra_context,
#           extra_lookup_kwargs=extra_lookup_kwargs)
        return object_list(request,
Address.objects.filter(name__icontains=name),
           paginate_by=10, extra_context=extra_context)
    else:
        return HttpResponseRedirect('/address/')
```

上述代码加到最后去。这里并没有完全重写，而是在 object_list 外面封装了一层，主要是生成要用在 object_list 的中参数。extra_context 是可以传入到模板中的上下文字典 。name_icontains 是 Django 中过滤条件的写法。这里是说只要包含指定的字符的即可，而且不区分大小写。详细地要看 Django 的 DB-API 文档。request.REQUEST ['search'] 或者从 GET

或者从 POST 中得到数据，是一个方便的用法。它将得到提交的查询姓名条件，如果存在，则生成 extra_lookup_kwargs 和 extra_context 参数，然后按 object_list 的要求传入。如果没有提交，则回到 address 的起始页面。

4 修改 address/urls.py，增加了一个 search 的 url 链接映射。

```
from django.conf.urls.defaults import *
from newtest.address.models import Address
info_dict = {
#    'model': Address,
   'queryset': Address.objects.all(),
}
urlpatterns = patterns('',
   (r'^/?$', 'django.views.generic.list_detail.object_list',
      dict(paginate_by=10, **info_dict)),
   (r'^upload/$', 'address.views.upload'),
   (r'^output/$', 'address.views.output'),
   (r'^search/$', 'address.views.search'),
)
```

5 启动 server 测试

感觉这个通讯录也差不多了，现在让我们将其部署到 Apache 上去跑一跑吧。但部署到 apache 时才知道，问题很多啊。主要问题如下：

a. 模块名不全：比如许多例子我都是从当前目录(newtest)下开始计算，因为在 Windows 下，Python_ 会自动将当前目录加入到 sys.path 中，因此直接使用 address.* 之类的不会出错，但在 Apache 下需要使用 newtest.address.* 这样的方式。必须按教程的方式处理主要修改 urls.py 文件。

b. 相对路径的问题:许多使用相对路径的地方都不对了。必须使用绝对路径。不过这一点对于部署来说的确有些麻烦，好在要改动的地方不多，主要在 settings.py 中。如数据库名字(sqlite3)，模板的位置。其它的就是要注意的地方了。

6 部署到 Apache 上的体验

只能说是体验了，因为我不是 Apache 的专家，也不是 mod_python 的专家，因此下面的内容只能算是我个人的配置记录，希望对大家有所帮助。

6.1 安装 mod_python 模块

Django 对于 Apache 使用 2.X ，对于 mod_python 使用 3.X。安装 mod_python(在 windows 下)倒是不麻烦。但在 Django 的邮件列表中却有人对于 mod_python 和 Apache 有所讨论，主要的问题是这些改动相对较大，比如说复载，安装需要 root 权限，要重启 Apache 等。这的确是一个要注意的问题，因此有人建议使用 FastCGI 或 SCGI 来处理。

6.2 修改 httpd.conf 文件

```
Listen 127.0.0.1:8888
<VirtualHost 127.0.0.1:8888>
   <Location "/">
      SetHandler python-program
      PythonPath "['E:/python'] + sys.path"
      PythonHandler django.core.handlers.modpython
      SetEnv DJANGO_SETTINGS_MODULE mysite.settings_apache
       PythonInterpreter mysite
      PythonAutoReload Off
      PythonDebug On
   </Location>
     Alias /site_media E:/python/mysite/media
     # Alias /media C:/Python25/Lib/site-
packages/django/contrib/admin/media
     Alias /media E:/python/mysite/media
     C:/Python25/Lib/site-packages/django/contrib/admin/media
   <Location "/site_media">
      SetHandler None
   </Location>
   <Location "/media">
      SetHandler None
   </Location>
</VirtualHost>
```

这里我使用了虚拟主机(参考文档) 来设置。即使用一台机器，不同的端口来对应不同的服务。主要原因是我希望 Django 的服务可以从 / 开始，但我还有其它的一些东西要处理，因此不希望对其它的东西有所影响。我没有两个域名，或两个 IP ，因此采用了两个不同的端口。这只是我的一种方式。因为我在本机处理，因此 IP 是 127.0.0.1 。实际中你应该进行修改。上面 PythonPath 主要是将 newtest 的目录加入到 sys.path ，以便 Django 可以找到。需要使用绝对路径 。SetEvn 中设置的 DJANGO_SETTINGS_MODULE 就对应于你的项目名.配置文件。因此为了能导入项目名.配置文件，就需要前面的 PythonPath 的设置。PythonDebug 和 PythonAutoReload 建议在生产时设为 Off 。这里我还设了两个别名，用来指向 site_media 和 media 目录。

Alias /site_media :是用来将 newtest 的静态文件设置一个 URL 访问的别名。

Alias /media :是将 Django Admin 的静态文件设置一个 URL 的访问别名。

在 site_media 和 media 的 Location 中设置不进行脚本的解析。上面的 media 路径是指向 Django Admin 所在的目录。你完全可以将其拷贝出来，这样可能要方便得多。另外在 linux 下使用 ln 也相当的方便。同时可以注意到 settings 我改为了 settings_apache 了。一方面将要把其中的内容有关相对路径的东西改为绝对路径，另一方面我还想保持现在的 settings.py 。

添加：

LoadModule python_module modules/mod_python.so

#==========

<Directory "C:/Python25/Lib/site-packages/django/contrib/admin/media">

Order Deny,Allow

Allow from all

</Directory>

<Directory "E:/python/mysite/media">

Order Deny,Allow

Allow from all

</Directory>

#==========

6.3 复制 settings.py 到 settings_apache.py

6.4 修改 settings_apache.py

将相对路径改为绝对路径。主要有：

- DATABASE_NAME
- MEDIA_ROOT
- TEMPLATE_DIRS
- STATIC_PATH

将 DEBUG 和 TEMPLATE_DEBUG 改为 False 。这样静态文件 serverview 就无效了。这就是为什么上面的 Apache 的配置中要配置 site_media 的原因。

6.5 测试:http://localhost:8888/address

更详细的内容请参见 mod_python 文档。关于 admin 的 media 和 template 好象并不需要配置，大家有什么结果可以告诉我。同时如果你不想每次重启 Apache 来进行测试，可以将:

```
MaxRequestsPerChild 0
```

改为:

```
MaxRequestsPerChild 1
```

7 后话

上面的步骤是直接把开发的东西发布到了 Apache 中去，但实际中开发与运行可能环境根本不一样，最主要可能就是数据库方面的变化，如果 model 变化，则有可能要编写数据切换程序。许多实际的问题都需要仔细地考虑。

第十三讲 简单的 Ajax 的实现(一)，MochiKit 的一些使用

1 引言

Ajax 是什么？它是一种技术的总称，包括了 Html, CSS, XML, Javascript 等与 web 相关技术的合集，在我以前的 Blog 也有一些涉及，但那时关注的焦点不在 web 上。在 Django 的 community 的 blog 上，有人发表了一篇关于使用 dojo (一个 Ajax 的库)来实现在搜索栏中实时输入信息时，可以动态显示与输入信息相匹配的 blog 列表的一个例子。他利用 dojo 实现了一个自定义的 widget ，但我感到这种技术对于我这种对于 dojo 框架不熟悉的人非常有困难。从 blog 上看，实现的过程还是有些复杂。我喜欢先从简单的东西入手。 MochiKit 在 Django 的 Ajax 的讨论中是另一个为大家关注的东西，最大的好处是它的文档最齐全，而且从本人的理解来说，它更简单。而 dojo 则更是提供了很多的 web UI 的控件， MochiKit 基本上没有。不过，在目前情况下我也只是希望体验一下 Ajax 技术，并且做一些简单的应用，而在简单的情况下，我认为 MochiKit 做为入门，作为简单的应用也足够了。下面就让我以 MochiKit 为基础来向大家介绍一下如何在 Django 中使用它，使用一些简单的 Ajax 技术。

首先让我们关心一下 Ajax 与 Django 的关系。其实 Ajax 本身包含许多的内容，它有浏览器端的显示技术，有与后台通讯的处理，因此与 Django 有关系的其实只有与后台交互那块东西。这样，更多的关于前端显示的技术，如：显示特效，这些都属于 CSS, Javascript 的内容，而这些与 Python 本身的关系也不大，因此你还需要掌握这些东西才可以做得更好。也许有机会会有

专题和学习和介绍这些方面的东西。下面的试验主要关注的是前端与后端的交互，也就是如何实现浏览器与 Django 交互，体验不进行页面的刷新(这是 Ajax 最大的好处，一切都好象在本地进行一样)。

就目前来说， Ajax 与后台交互都是通过浏览器提供的 XMLHttpRequest 对象来实现的。这个对象支持同步和异步的调用，但由于 Javascript 本身没有多线程这个东西，因此为了不阻塞浏览器，一般都采用异步方式来调用，而这也是一般的 Ajax 框架提供了默认方式。就目前来说，交互数据也有多种格式，比如：XML, Json，纯文本/Html。XML 不用说了，但一般采用 http 协议的 web server 是无法直接支持，因此需要进行转换。同时在浏览器你要同时进行 XML 的解析，不是非常方便。 Json 是一种数据表示格式，它非常象 Python 的数据类型。而且它只有数据，没有任何的格式，因此数据传输量非常小。再加上处理起来也很方便，在传输上可以直接转换为文本，然后再转换成不同语言的数据结构即可。对于 Python 是非常方便。再有就是文本/Html 方式，一种是自定义格式，通过转化为文本进行处理，另一种就是直接使用 html 标记。前一种需要自行做扩展，后一种则是最方便。 下面我们将先使用 html 方式，然后再使用 Json 来进行试验。

我设计了一个非常简单的例子：提供一个输入框，用户输入文本，然后点提交，直接在下面显示后台返回的结果。因为我不是 Javascript，CSS 的专家，可能有不对的地方。

2 创建 Ajax 应用

```
manage.py startapp ajax
```

3 修改 ajax/views.py

```
# Create your views here.
from django.http import HttpResponse
def input(request):
    input = request.REQUEST["input"]
    return HttpResponse('<p>You input is "%s"</p>' % input)
```

从这里可以看出，我需要一个 input 字段，然后返回一个 HTML 的片段。

4 创建 templates/ajax 目录

5 创建 templates/ajax/ajax.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Ajax Test</title>
    <script type="text/javascript" src="/site_media/MochiKit.js"></script>
    <script type="text/javascript" src="/site_media/ajax_test.js"></script>
  </head>
  <body>
    <h1>

      Ajax 演示

    </h1>
    <div>
    <form id="form">

    输入：<input type="text" name="input"/>


    <input id="submit" type="button" value="提交" />

    </form>
    </div>
    <div id="output"></div>
  </body>
</html>
```

这个模板将作为初始页面，它用来处理向后台发起请求。在这里它没有需要特殊处理的模板变量，只需要显示即可。但在这里的确有许多要说明的东西。这是一个标准的 html 的页面，在 head 标签中，它将引入两个 js 文件： MochiKit.js 和 ajax_test.js 。从 url 上可以看出，我将会把它们放在 site_media 下，这个地址就是 media 目录。 MochiKit.js 你需要从 MochiKit 网站下载(最新版本为 1.2)。MochiKit 下载后有两种格式，一种是单个文件，另一种是分散的文件。我这里使用的是单个文件。在 html 文件中有一个 form ，它的 id 是 form ，我将用它来查找

form 对象。它有一个文本输入框，还有一个按钮，但这个按钮并不是 submit 按钮。这里有许多与标准的 form 不一样的地方，没有 action, 没有 method ，而且没有 submit 按钮。为什么要这样，为了简单，而且我发现这是 MochiKit 的开发方式。以前写 HTML，CSS, Javascript 和事件之类的处理，我们一般可能会写在一起，但这样的确很乱。在学习了一段 MochiKit 之后，我发现它的代码分离做得非常棒，而这也是目前可能流行的做法。它会在独立的 Javascript 中编写代码，在装载页面时动态地查找相应的元素，然后设置元素的一些属性，如 style ，事件代码等。而在 Html 文档中，你看到的元素中一般就只有 id，class 等内容。这样的好处可以使得处理为以后重用及优化带来方便，同时可以通过编程的方式实现批量的处理，而且也使得 Html 页面更简单和清晰。因为我要使用 Ajax 去动态提交信息，不需要真正的 form 的提交机制，我只是需要用到 form 元素中的数据而已，因此象 action, method 等内容都没有用。 id 是必须的，我需要根据它找到我想要处理的元素对象。不过分离的作法是你的文件将增多，也可能不如放在一个文件中便于部署吧。这是一个仁者见仁，智者见智的作法。 它是用来显示结果的层。

整个处理过程就是：在装载 html 页面时，会对按钮进行初始化处理，即增加一个 onclick 的事件处理，它将完成 Ajax 的请求及结果返回后的处理。然后用户在页面显示出来后，可以输入文本，点击按钮后，将调用 onclick 方法，然后提交信息到 Django ，由 Django 返回信息，再由 Ajax 的 deferred 对象(后面会介绍)调用显示处理。

6 创建 media/ajax_test.js

```
function submit(){
    var form = $("form");
    var d = doSimpleXMLHttpRequest('/ajax/input/', form);
    d.addCallbacks(onSuccess, onFail);
}
onSuccess = function (data){
    var output = $("output");
    output.innerHTML = data.responseText;
```

```
    showElement(output);
}
onFail = function (data){
    alert(data);
}
function init() {
    var btn = $("submit");
    btn.onclick = submit;
    var output = $("output");
    hideElement(output);
}
addLoadEvent(init);
```

这里有许多是 MochiKit 的方法。首先让我们看 addLoadEvent(init); 它表示将 init() 函数加到 onload 的响应事件对列中。浏览器在装载完一个页面后，会自动调用 onload 事件处理。因此在这里是进行初始化的最好的地方。init() 方法一方面完成对 id 名为 submit 的按钮 onclick 处理函数的绑定工作，另一个是将 id 为 output 的元素隐藏。其实不隐藏也无所谓，因为它本来就是空的，因此你也看不到东西。不过如果有其它的东西这样的处理却也不错。$() 是 MochiKit 提供的一个 getElement() 函数别名，它将根据元素的 id 来得到某个对象。hideElement() 是隐藏某个元素。想要显示某个元素可以使用 showElement() 。最重要的工作都在 submit() 这个函数中。它首先得到 id 为 form 的对象，然后调用 MochiKit 提供的 doSimpleXMLHttpRequest() 函数提交一个 Ajax 请求到后台。第一个参数是请求的 url ，第二个如果有的话，应该是 Query String ，即一个 url 的？后面的东西。这里我只是将 form 传给它， doSimpleXMLHttpRequest() 会自动调用 queryString() (也是 MochiKit 的一个方法)来取得 form 中的字段信息。比如你输入了 aaa ，那么最终在 Django 你会看到的是:

```
/ajax/input/?input=aaa
```

doSimpleXMLHttpRequest() 会返回一个 deferred 对象，它是一个延迟执行对象，在执行了 doSimpleXMLHttpRequest() 之后，结果可能当时并没有返回回来，因为这是一个异步调用。因此为了在结果回来之后做后续的处理，我还需要挂接两个异步函数，一个用来处理成功的情

况，一个是用来处理失败的情况。 d.addCallbacks(onSuccess, onFail); 就是做这件事的 。onSuccess() 在 deferred 正确返回后会被调用。data 是 XMLHttpRequest 对象本身，它有一个 responseText 属性可以使用。这里因为 Django 返回的是 Html 片段，因此我只是简单地将 output 对象(用于显示的 div 层)的内容进行了设置。然后调用 showElement() 来将层显示出来 onFail() 则只是调用 alert() 显示出错而已。这里有许多 Javascript 和 MochiKit 的东西，如果大家不了解则需要补补课了。其中 MochiKit 的内容在它自带的例子和文档中可以查阅，特别是 MochiKit 自带了一个象 Python shell 一样的命令行解释环境可以进行测试，非常的方便。具体的看 MochiKit 网站上的 ScreenCast 可以了解。

7 修改 urls.py,增加两行:

```
(r'^ajax/$', 'django.views.generic.simple.direct_to_template',{'template': 'ajax/ajax.html'}),
(r'^ajax/input/$', 'newtest.ajax.views.input'),
```

前一个使用了 generic view 所提供的 direct_to_template() 方法可以直接显示一个模板。后一个则指向了 views.index() 方法，它用于在前一个页面点击按钮后与后台交互的处理。

8 安装 ajax 应用,修改 settings.py

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'newtest.wiki',
    'newtest.address',
    'newtest.ajax',
    'django.contrib.admin',
)
```

9 启动 server 测试

这样你在文本框中输入内容，点击提交后就会立即在文本框的下面看到结果，而页面没有刷新，这就是 Ajax 就直接的应用。

第十四讲简单的 Ajax 的实现(二)，使用 SimpleJson 来交换数据

1 引言

Ajax 因为大量地使用了 Javascript ，而调试 Javascript 的确不是件容易的事，在这方面只有不停地测试，还要靠耐心。而且 Ajax 本身可能还有一些安全方面的东西需要考虑，但这些话题需要你自已去学习了。在试验了简单的 Html 返回片段之后，让我们再体验一下 Json 的应用吧。为了使用 Json ，我下载了 simplejson 模块。我下载的是 1.1 版本。还可以使用 easy_install 来安装。如何使用 simplejson 在它自带的文档有示例很简单，下面我们就用它来试验 Json 的例子。我将在上一例的基础之上，增加一个按钮，这个按钮点击后，会发送一个请求(不带 Json 信息)，然后 Django 会返回一个 Json 格式的表格数据，分为头和体两部分。然后前端动态生成一个表格显示在 output 层中。

2 修改 ajax/views.py

```
#coding=utf-8
# Create your views here.
from django.http import HttpResponse

def input(request):
    input = request.REQUEST["input"]
    return HttpResponse('<p>You input is "%s"</p>' % input)
def json(request):
    a = {'head':('Name', 'Telphone'), 'body':[(u'张三', '1111'), (u'李四',
'2222')]}
    import simplejson
    return HttpResponse(simplejson.dumps(a))
```

json() 是新加的方法。 a 是一个字典，它会被封装为 Json 的格式。这里还使用了汉字，但使用了 unicode 的表示。我发现 simplejson 在处理非 ascii 码时会自动转为 unicode ，但不正确因此我直接使用了 unicode。因此我希望浏览器可以根据这个数据生成表格。

3 修改 templates/ajax/ajax.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
   <head>
      <title>Ajax Test</title>
      <script type="text/javascript" src="/site_media/MochiKit.js"></script>
      <script type="text/javascript"
src="/site_media/ajax_test.js"></script>
   </head>
   <body>
      <h1>

         Ajax 演示

      </h1>
      <div>
      <form id="form">

      输入：<input type="text" name="input"/>

      <input id="submit" type="button" value="提交" />

      <input id="json" type="button" value="JSON 演示" />

      </form>
      </div>
      <div id="output"></div>
   </body>
</html>
```

这里只是增加了一个按钮， id 是 json 。它将用来触发 Ajax 请求。

4 修改 media/ajax_test.js

```
function callJson(){
   var d = loadJSONDoc('/ajax/json/');
   d.addCallbacks(onSuccessJson, onFail);
```

```
}
row_display = function (row) {
    return TR(null, map(partial(TD, null), row));
}
onSuccessJson = function (data){
    var output = $("output");
    table = TABLE({border:"1"}, THEAD(null, row_display(data.head)),
        TBODY(null, map(row_display, data.body)));
    replaceChildNodes(output, table);
    showElement(output);
}
function init() {
    var btn = $("submit");
    btn.onclick = submit;
    var output = $("output");
    hideElement(output);
    var btn = $("json");
    btn.onclick = callJson;
}
```

在最后一行 addLoadEvent(init); 前加入上面的内容。对于 id 为 json 的按钮的事件绑定方式与上一例相同，都是在 init() 中进行的。在 callJson() 中进行实际的 Json 调用，这次使用了 MochiKit 提供的 loadJSONDoc() 函数，它将执行一个 url 请求，同时将返回结果自动转化为 Json 对象。一旦成功，将调用 onSuccessJson() 函数。在这里将动态生成一个表格，并显示出来。表格的显示使用了 MochiKit 的 DOM 中的示例的方法。 row_display() 是用来生成一行的。TBODY 中使用 map 来处理数组数据。在 MochiKit 中有许多象 Python 内置方法的函数，因为它的许多概念就是学的 Python 。 replaceChildNodes() 是用来将生成的结果替换掉 output 元素的内容。

5 修改 urls.py

```
(r'^ajax/json/$', 'newtest.ajax.views.json'),
```

增加上面一行。这样就增加了一个 Json 的 url 映射。

6 启动 server 进行测试

这里两个演示共用了 output 层作为显示的对象，你可以同时试一试两个例子的效果。不过这里有一个问题：只有返回时使用了 Json 。的确是，这样是最简单处理的情况。因为 Json 可以包装为字符串，这样不用在底层进行特殊处理。如果请求也是 Json 的，需要设计一种调用规则，同时很有可能要实现 MiddleWare 来支持。在 Django 中的确有人已经做过类似的工作。不过我目前没有研究得那么深，因此只要可以处理返回为 Json 的情况已经足够了。而且 Django 也正在进行 Ajax 的支持工作，不过可能是以 dojo 为基础的，让我们拭目以待吧。

第十五讲 i18n 的一个简单实现

1 引言

在 Ajax 的试验中，你会看到有一些是用英文写的。下面就让我们学习如何将应用改为支持 i18n 处理的吧。在本讲中我会讲述我实现的过程，同时对一些问题进行讨论。Django 中 i18n 的实现过程:

1.1 在程序和模板中定义翻译字符串

在程序中就是使用 _() 将要翻译的字符串包括起来。这里有几种做法，一种是什么都不导入，这样就使用缺省的方式，另一种是导入 Django 提供的翻译函数。特别是 Django 提供了 Lazy 翻译函数，特别可以用在动态语言的切换。在模板中分几种情况：

- 可以使用 {% trans %} 标签。它用来翻译一句话，但不能在它中间使用模板变量。
- 如果是大段的文本，或要处理模板变量，可以使用 {% blocktrans %}{% endblocktrans %} 来处理。

Django 还支持简单的 Javascript 的 i18n 的处理，但有兴趣自已去看吧。

1.2 生成 po 文件

定义好翻译串之后使用 bin/make-messages.py 来生成 po 文件。Django 支持多层次的处理。比如在整个 Django 的源码项目，在某一个工程，在某一个应用。在不同层次去实现 i18n 时，需要在不同的层次的根目录去执行 make-messages.py 。那么可以将 make-messages.py 拷贝到相应的目录去执行，特别是在你的工程或应用中。在执行 make-messasges.py 时，需要你预先创建 conf/locale 或 locale 目录，而 make-messasges.py 是不会自动为你创建的。那么 conf/locale 多用在源码中，象 Django 的源码就是放在 conf/locale 中的。 **但在运行时，对于自已的项目和应用却是从 ``locale`` 中来找的** 。因此还是建议你创建 locale 来存放 po 文件。第一次执行时:

```
make-messages.py -l zh_CN
```

这时会生成 locale/zh_CN/LC_MESSAGES/django.po 和 django.pot 两个文件。在 0.91 版需要使用 po 工具将 .pot 合并到 .po 文件中。但 0.95 版则已经自动做好合并了。如果有 .pot 文件，可以：用 poEdit 打开 django.po 后，选择类目->从 POT 文件更新(我使用的是 poEdit 中文版)。这样内容就更新到 po 中去了。然后你就可以开始翻译了。翻译完成之后，首先要执行类目->设置，将缺省的参数修改一下。主要是：项目名称及版本，团队，团队专用电子邮件，字符集(一般为 utf-8)。这些如果不改， poEdit 在保存时会报错。使用 poEdit 的一个好处是，在保存时它会自动将 po 编译成 mo 文件。以后再更新时:

```
make-messasges.py -a
```

如果已经有多个语言文件，那么执行时会同时更新这些 po 文件。

1.3 配置

Django 有一系列的策略来实现 i18n 的功能。基本上分为静态和动态。**静态**是指在 settings.py 中设置 LANGUAGE_CODE 为你想要的语言。那么这里要注意，中文的语言编码是 zh-cn ，但 locale 目录下却是 zh_CN 。这是为什么：其实一个是 language(zh-cn) ，一个是 locale(zh_CN) ，在 Django 的 utils.translation.py 中有专门的方法可以进行转换。因此在 Django 的程序中使用的是 language 的形式，在目录中却是使用 locale 的形式。一旦设为静态则它表示是全局性质的，在所有其它的策略失效后将使用这种策略。而**动态是**指在运行中对于不同的用户，不同的浏览器的支持的语言可以有不同的语言翻译文件被使用。这种方式需要在 settings.py 中安装 django.middleware.locale.LocaleMiddleware 到 MIDDLEWARE_CLASSES 中去。同时如果你想在实现应用中的翻译文件被使用，也要采用这种方式。在一个请求发送到 Django 之后，如果安装了 LocaleMiddleware ，它会采用下面的策略

- 在当前用户的 session 中查找 django_language 键字。
- 如果没有找到则在 cookie 中查找叫 django_language 的值。
- 如果没有找到，则查看 Accept-Language HTTP 头。这个头是由浏览器发送给服务器的。
- 如果没有找到，则使用全局的 LANGUAGE_CODE 设置。

如果你使用 FireFox 可以在 Tools->Options->Advanced->Eidt Languages 设置你所接受的语言，并且将 zh-cn 放在最前面。上面讲述得还是有些粗，建议你好好阅读 i18n 的文档。国际化处理的文档请参阅： Internationalization 文档.下面开始我们的试验。

2 修改 ajax/views.py

```
#coding=utf-8
# Create your views here.
from django.http import HttpResponse
def input(request):
    input = request.REQUEST["input"]
    return HttpResponse(_('<p>You input is "%s"</p>') % input)
def json(request):
```

```
a = {'head':(unicode(_('Name'), 'utf-8'), unicode(_('Telphone'), 'utf-8')),
    'body':[(u'张三', '1111'), (u'李四', '2222')]}
import simplejson
return HttpResponse(simplejson.dumps(a))
```

这里对所有英文都使用 _() 进行了封装。但对于 Json 方法，这里我使用 unicode(_('Name'), 'utf-8') 进行了转换。目前来说， Django 内部使用 utf-8 编码。因此从 _() 返回的并不象我以前认为的是 unicode ，而是 utf-8 编码。那么关于缺省编码在 django.conf.global_settings.py 中有一个 DEFAULT_CHARSET 的值。缺省情况下它是 utf-8 。但现在的问题是 simplejson 需要 unicode 来正确处理汉字。那么我只有将 _() 返回的值转化为 unicode 。在 Django 的邮件列表中已经开始讨论 Django 的核心是否全部采用 unicode 的问题。

3 修改 settings.py,增加 LocaleMiddleware

```
MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.doc.XViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
)
```

这里在文档中对于 LocaleMiddleware 的顺序有要求，要求排在 SessionMiddleware 之后，但在其它的 Middleware 之前。话虽如此，但我感觉目前顺序影响不大，也许只是个人感觉吧。

4 创建 ajax/locale 目录

5 拷贝 make-messasges.py 到 ajax 目录下

6 执行 make-messasges.py

```
cd ajax
make-message.py -l zh_CN
```

7 使用 poEdit 翻译 django.po 文件

按上面说的先更新 pot 文件，然后修改缺省的参数，再保存。如果你没有 poEdit，或不在 Windows 平台下，那么只好自已去想办法了。同时这里 make-message.py 还需要 Windows 下的 xgettext 工具。可以在 http://code.djangoproject.com/wiki/Localization 找到说明。这里我没有演示模板的处理。因为 Ajax 所用到的模板没有放在 ajax 目录下，而是放在 templates 目录下。因此，如果想支持 i18n 的话，目录的布置是一个问题。所以不再试验了。

8 启动 server 测试：是中文了呢？如不是，看一看是否浏览器没有设置成接受 zh-cn 。

第十六讲 自定义 Calendar Tag

1 引言

Django 中的模板系统可以被自由扩展，如自定义 filter, 自定义 Tag 等。其中 filter 用于对变量的处理。而 Tag 则功能强大，几乎可以做任何事情。我认为 Tag 的好处有非常多，比如：

- 可以简单化代码的生成。一个 Tag 相当于一个代码片段，把重复的东西做成 Tag 可以避免许多重复的工作。
- 可以用来组合不同的应用。将一个应用的展示处理成 Tag 的方式，这样就可以在一个模板中组合不同的应用展示 Tag，而且修改模板也相对容易。

如果要自定义 Tag ，那么要了解 Tag 的处理过程。在 Django 中，Tag 的处理分为两步。

1. 编译。即把 Tag 编译为一系列的 django.template.Node 结点。
2. 渲染(Render)。即对每个 Node 调用它们的 render() 方法，然后将输出结果拼接起来。

因此自定义一个 Tag ，你需要针对这两步处理来做工作。在 The Django template language: For Python programmers 文档中讲解了一些例子。大家可以看一下。那么下面，我将实现一个显示日历的自定义 Tag 。

2 下载 HTMLCalendar 模块并安装

不想全部自已做，因此找了一个现成的模块。去 HTMLCalender 的主页下载这个模块。

然后解压到一个目录下，执行安装:

```
python setup.py install
```

3 下载 HTMLTemplate 模块并安装

然后解压到一个目录下，执行安装:

```
python setup.py install
```

因为上面的 HTMLCalender 需要它才可以运行。去 HTMLTemplate 主页下载这个模块。

4 创建 my_alendar 应用

```
manage.py startapp my_calendar
```

这里起名为 my_calendar 。因为如果起名为 calendar 会与系统的 calendar 模块重名。

5 创建 my_calendar/templatetags 目录

6 创建 my_calendar/templatetags/__init__.py 文件，空文件即可。

7 创建 my_calendar/templatetags/my_calendar.py 文件

```
from django import template
register = template.Library()
```

```
class CalendarNode(template.Node):
    def __init__(self):
        pass
    def render(self, context):
        return "Calendar"
def do_calendar(parser, token):
    return CalendarNode()
register.tag('calendar', do_calendar)
```

上面的代码只是一个空架子。不过让我们仔细地解释一下：

- register 与自定义 filter 一样，它将用来注册一个 Tag 的名字到系统中去。
- CalendarNode 它是 template.Node 的一个子类。每个 Tag 都需要从 Node 派生。这个类可以只有 render() 方法，用来返回处理后的文本。 __init__() 可能是有用的，先预留。
- render() 方法接受一个 context 参数。这个参数就是在执行模板的渲染时由 View 传入的。不过更复杂的例子是你可以修改 context ，这样达到注入新变量的目的。不过本例没有演示。
- do_calendar() 是一个由模板处理引擎在发现一个 Tag 的名字之后，将进行调用的方法。那么我们的 Tag 可能在模板中写为 {% calendar %} 。这个方法将在下面通过注册过程与一个名字相对应，这里我们想使用 calendar 。

  它接受两个参数：

  - parser 这是模板处理引擎对象，我们没有用到。
  - token 表示 Tag 的原始文本。如果在模板中我们定义 Tag 为 {% calendar 2006 1 %}, 那么 token 就为 calendar 2006 1 。因此你需要对它进一步地处理。

  它将返回一个 Node 的实例，在本例中就是 CalendarNode 实例。

- register.tag('calendar', do_calendar) 用来注册 Tag 名字和对应的处理方法。

尽管我们没有对 calendar 所带的参数进行处理，但它仍然可以显示。要知道我们还没有使用 HTMLCalender 模块呢。

8 创建 mysite/templates/my_calendar 目录

9 创建 mysite/templates/my_calendar/calendar.html 文件

```
{% load my_calendar %}
{% calendar 2006 1 %}
```

10 修改 usls.py

增加下面的 url 配置:

```
(r'^calendar/$', 'django.views.generic.simple.direct_to_template',
    {'template': 'my_calendar/calendar .html'}),
```

11 修改 settings.py 安装 my_calendar 应用

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'newtest.wiki',
    'newtest.address',
    'newtest.ajax',
    'newtest.my_calendar',
    'django.contrib.admin',
)
```

12 启动 server 测试

页面上应该显示出 Calendar 的文本。我们在模板中定义的参数没有被用到。因为我们没有真正调用 HTMLCalender 输出，因此上面只是说明框架是可用的。下面让我们加入参数的处理。

13 修改 my_calendar/templatetags/my_calendar.py

```
from django import template
import HTMLCalendar
register = template.Library()
class CalendarNode(template.Node):
    def __init__(self, year, mon):
        self.year = int(year)
        self.mon = int(mon)
    def render(self, context):
        return HTMLCalendar.MonthCal().render(self.year, self.mon)
def do_calendar(parser, token):
    try:
        tag_name, arg = token.contents.split(None, 1)
    except ValueError:
        #if no args then using current date
        import datetime
        today = datetime.date.today()
        year, mon = today.year, today.mon
    else:
        try:
            year, mon = arg.split(None, 1)
        except ValueError:
            raise template.TemplateSyntaxError, "%r tag requires year and mon
arguments" % tag_name
    return CalendarNode(year, mon)
register.tag('calendar', do_calendar)
```

主要改动如下：

1. 增加了 import HTMLCalendar 的导入。
2. 修改了 CalendarNode 的 __init__() 方法，增加了两个参数。
3. 修改了 CalendarNode 的 render() 方法。改成输出一个 Calendar 的表格。
4. 修改了 do_calendar() 函数，增加了参数的处理。如果没有输入参数则使用当前的年、月值。否则使用指定的年、月参数。如果解析有误，则引发异常。

在调试的过程中，的确有一些错误。象开始时我命名为 calendar 目录，结果造成与系统的 calendar 模块重名。然后不得已进行了改名。为什么发现要导入 HTMLTemplate 呢？因为在处理时 HTMLCalender 抛出了异常。但成功后我已经把这些调试语句去掉了。而且发现这些错误

Django 报告得有些简单，你可能不清楚倒底是什么错。因此最好的方法：一是在命令行下导入试一下，看一看有没有导入的错误。另外就是使用 try..except 然后使用 traceback 模块打印异常信息。

14 启动 server 测试，你会看到：

也许感到不好看，没关系，可以通过 CSS 进行美化。当然，这样可能还是不让人满意，比如：不是 i18n 方式的，因此看不到中文。不过这已经不是我们的重点了。掌握了自定义 Tag 的方法就可以自行进行改造了。同时 HTMLCalender 模块本身可以传入一些链接，这样就可以在日历上点击了。这里不再试验了。有兴趣的可以自已做一下。

第十七讲 View, Template, Tag 之间的关系

1 引言

经过前面许多讲之后，我想大家应该对 Django 的基本开发概念和过程已经有所了解。那么是时候讲一些关于设计方面的东西。首先要声明，目前 Django 基本上还没有什么设计的教程，而我也只能写一些个人体会。那么这篇教程的体会就是：View, Template and Templatetag

2 View, Temaplte 和 Tag 之间的关系

View 在 Django 中是用来处理请求的，一个 url 请求上来后经过 Django 的处理首先找到这个 url pattern 对应的 View 模块的某个方法。因此 View 是处理请求的起点，同时，在 View

中的方法需要返回，因此它还是一个请求的终点。因此象 Template 和 Tag 只不过是处理中的某些环节。 View 可处理的范围远大于 Template 而 Tag 则只能用在 Template 中。因此从使用范围上说：View > Template > Tag。Template 是用来输出内容的，目前在 Django 中你可以用它输出文本之类的东西。但象图片之类的非文本的东西，则只能通过 View 来实现，再有如果想在输出时加入一些特殊的 HttpHeader 的控制也只能在 View 中实现。当然，在大多数情况下我们只处理动态的文本生成，其它许多东西都是静态的。象图片之类的可以通过链接来引用。Tag 是在 Template 中被使用的。它的作用很多，如控制模板逻辑，还可以输出内容并做转换等。Tag 可以自定义，因此你可以在 Tag 中做几乎你想做的有关内容输出的任何事，如从数据库中取出内容，然后加工，输出，许多事情。在 Django 中提供了一种方便的方法，可以直接将 url 与模板相对应起来。但并不是说你不需要 View 的参与，而是这个 View 的功能是预先写好的，它的作用很简单，就是在 View 方法中直接渲染一个模板并输出。因此说，看上去好象是直接对应，但实际上还是有 View 的处理。比如:

```
(r'^ajax/$', 'django.views.generic.simple.direct_to_template',
    {'template': 'ajax/ajax.html'}),
```

这是在讲 Ajax 的一个 url 的配置，其中使用了 django.views.generic.simple.direct_to_template 这个做好的 View 方法。

3 如何设计

从上面的分析我们可以看出， View, Template, Tag 功能不尽相同，但的确有部分功能的重叠，特别是在文本信息的输出。如何比较好的选择使用什么来输出呢？可以从几下以方面考虑：

1. 输出内容

HTML 或文本内容，可以考虑使用 View + Template + Tag ，其它的考虑使用 View

2. 输出范围

如果是多数据源，比如一个首页，可能包含许多不同的内容，如个人信息统计，Blog 展示，日历，相关的链接，分类等，这些信息类型不同，如何比较好的处理呢？可以以 View 为主，即数据在 View 中提供，在模板中考虑输出和布局。但有一个问题，重用不方便。因此采用 Tag 可能更适合。因此对于单一或简单数据源可以只采用 View 和 Template 来实现，而对于多数据源可以采用使用 Temaplate 控制布局，Tag 用来输出单数据源信息。

同时对于多数据源的信息还可以考虑使用 Ajax 技术动态的将信息结合在一起。但使用 Ajax 则需要动态与后台交互，将单数据源的信息组织在一起，这样每个来源都是一个 View 的处理。不过这个有些复杂，这里我们不去考虑它。因此当你设计结构时，首先考虑实现的内容，是文本的，则可以考虑使用 View, Template 和 Tag。然后再看是否有重用的需要，有的话，将可重用的部分使用 Tag 来实现，而 View 和 Template 作布局和控制。

4 结论

这里我想到一个问题：我一直想使用 Admin 作为我的数据管理的界面。但经过上面的分析，Admin 目前大多数情况下只处理单一数据表，有些包含关系的，比如一对一，多对一，多对多的可以在一个编辑页面中同时处理多个表的记录，但它还是有可能无法满足复杂的多数据源的表现和编辑问题。因此 Admin 应该可以认为是一个缺省的数据库管理界面，而不完全是一个用户管理界面。因此大多数情况下，你仍然需要自定义管理界面，而不能完全依靠 Admin 。除非你的应用简单，同时对于管理界面的要求不高。

解决了这个问题，于是我们不必太留恋 Admin 的功能，我相信会有一些好的解决方案来满足我们的要求，或者就是我们自已来创建这样的项目。

Wiki 一词来源于夏威夷语的“wee kee”，原本是“快点”的意思。在这里 Wiki 指的是一种网上共同协作的超文本系统,可由多人共同对网站内容进行维护和更新。我们可以通过网页浏览器对 Wiki 文本进行浏览、创建、更改，而且创建、更改、发布的代价远比 HTML 文本为小,您并不需要懂得 HTLM 代码，只要简单了解少量的 Wiki 的语法的约定，您就可以在系统中发布您的页面！与其它超文本系统相比，Wiki 有使用方便及开放的特点，所以 Wiki 系统可以帮助我们在一个社群内共同收集、创作某领域的知识，发布大家都关心和感兴趣的话题。

Wiki 系统创造者的 Ward Cunningham，共同为 Wiki 下了定义：一群相互连接并可自由扩展的网页、一套用来储存与修改信息的超文字系统，所有的网页储存在一套数据库中，任何人透过具有表单功能的浏览器用户程序，皆可轻易加以编辑。

想看具体的 WIKI 的教程,请到:http://www.searchweb.cn/news/2005-10/20051031161155.htm