第一章 django学前准备

1.1 安装virtualenv

virtualenv 是用来创建虚拟环境的软件， 我们可以通过pip 或者pip3来安装

```
pip install virtualenv

pip3 install virtualenv
```

1.2 创建虚拟环境

创建虚拟环境非常简单，通过以下命令

```
virtualenv [虚拟环境的名字]
```

1.3 进入虚拟环境

虚拟环境创建好后， 那么可以进入这个虚拟环境中， 然后安装第三方包，

进入虚拟环境一般分两种， 第一种windows, 第二种linux

1. windows 进入虚拟环境：进入虚拟环境的scripts文件夹中
2. linux进入虚拟环境：source/path/to/virtualenv/bin/activate 一旦你进入这个虚拟环境中，
   你安装包， 卸载包都是在这个虚拟环境中， 不会影响到外部环境

1.4 退出虚拟环境

通过命令可以完成：deactivate

1.5 创建虚拟环境的时候指定python解释器

在电脑的环境变量中，一般是不会去更改一些环境变量的顺序的， 也就是说比如

你的python2/Scripts在python3/Scripts的前面， 那么你不会经常更改他们玩的位置， 但是、

这时候我确实想创建虚拟环境的时候用python3这个版本， 这时候可以通过-p参数来指定

具体的python解释器

```
virtualenv -p c:\Python36\python.exe
```

1.6 virtualenvwrapper

virtualenvwrapper 这个软件包让我们管理虚拟环境变得更加简单， 不再跑到某个目录下

通过virtualenv 来创建虚拟环境， 并且激活的时候也要跑到具体的目录下去激活

1 安装 virtualenvwrapper:

1. *nix: pip install virtualenvwrapper
2. windows: pip install virtualenvwrapper-win

2 virtualenvwrapper 基本使用：

1. 创建虚拟环境：

```
mkvirtualenv my_env
```

2. 切换到某个虚拟环境

```
workon my_env
```

3. 退出当前虚拟环境

```
deactivate
```

4. 删除某个虚拟环境

```
rmvirtualenv my_env
```

5. 列出所有虚拟环境

```
lsvirtualenv
```

6. 进入虚拟环境所在的目录

```
cdvirtualenv
```

3 修改 mkvirtualenv 的默认路径

在我的电脑-右键-属性-高级系统设置-环境变量-系统变量 中添加一个参数WORKON_HOME，将这个

参数的值设置为你需要的路径

4 创建虚拟环境的时候指定Python版本

在使用 mkvirtuakenv 的时候， 可以指定 --python 的参数来指定具体的python路径

```
mkvirtualenv --python==c:Python36\python.exe my_env
```

1.7 学前准备

1. 安装python3.6
2. 安装 virtualenvwrapper, 这个是用来创建虚拟环境的包， 使用虚拟环境可以让我们的包管理更加方便
3. 虚拟环境相关操作：

创建虚拟环境： mkvirtualenv --python=['python3.6文件所在路径'][虚拟环境名]

进入虚拟环境：workon [虚拟环境名]

退出虚拟环境：deactivate

4. 首先进入虚拟环境 workon django-env, 然后通过 pip install django==2.0
5. 安装pycharm profession 2017 (专业版)， community(社区版)不能用于网页开发
6. 安装最新 MySQL, windows版的MySQL下载地址：https://dev.mysql.com/downloads/windows/installer/5.7.html, 如果是其他操作系统， 来这个页面选择具体的MySQL进行下载：http://dev.mysql.com/downloads/mysql/
7. 安装 pymysl, 这个库是python 来操作数据库的， 没有它， django就不能操作数据库，pip install pymysql

```
django-admin startproject name .
pip install mysqlclient
如果报错在 则下载 mysqlclient-1.3.13-cp36-cp36m-win_amd64
 网址：'https://www.lfd.uci.edu/~gohlke/pythonlibs/' ，然后pip install 下
创建应用  python manage.py startapp user, 并把user这个包添加到setting.py文件的INSTALLED_APPS下面
创建model(创建数据库)
安装pymysql  命令; pip install pymysql
迁移命令
创建models  python manage.py makemigrations
执行迁移命令生成数据库表 python manage.py migrate


Django 后台管理
1, 创建管理员
管理员名称 admin
密码        poo14755
python manage.py createsuperuser
2, 本地化
setting.py中设置语言， 时区
语言名称可以查看django\contrib\admin\locale 目录
3, 启动服务器
python manage.py runserver
4, 登录后台页面
5，注册应用模块

路由
编写WSGI框架项目中， 路由功能就是实现url模式匹配和处理函数之间的映射
对于django也是如此， 路由配置要在项目中的urls.py中配置， 也可以多级配置
在每一个应用中， 建立一个urls.py文件配置路由映射
url函数
url(regex, view, kwargs=None, name=None), 进行模式匹配
regex: 正则表达式， 与之匹配的ull 会执行对应的第二个参数view
view: 用于执行与正则表达式匹配的url 请求
kwargs: 视图使用的字典类型参数
name: 用来反向获取url
urls.py 内容如下
```

1 变量
        语法：{{content}}
2 模板标签
        if/else标签
        基本格式如
        {% if condition %}
                ....display
        {% endif %}
条件也支持and, or, not
注意， 因为这些标签是断开的， 所以不能像python一样使用缩进就可以表示出来， 必须有个结束标签
例如: endif, endfor
  for 标签
  <url>
      {% for athlete in athlete_list %}
          {{ athlete.name}}
      {% endfor %}
  </url>
  {% for person in person_list %}
       {{ person.name }}
  {% endfor %}

```
变量                        说明
forloop.counter             当前循环从1开始计数
forloop.counter0            当前循环从0开始计数
forloop.recounter           从循环的末尾开始倒计数到1
forloop.recounter0          从循环的末尾开始倒计数到0
forloop.first               第一次进入循环
forloop.last                    最后一次进入循环
forloop.parentloop          循环嵌套时， 内层当前循环的外层循环

给标签增加一个reversed使得该列表反向迭代
{% for athlete in athlete_list reversed %}
...
{% empty %}
... 如果被迭代的列表是空的或者不存在， 执行empty
{% endfor %}

可以嵌套使用{% for %}标签
{% for athlete in athlete_list %}
        <h1>{{ athelete.name }} </h1>
        <ul>
        {% for sport in athlete.sports_played %}
                <li> {{ sport }} </li>
        {% endfor %}
        </ul>
{% endfor %}

ifequal/ifnoequal 标签
{% ifequal %} 标签比较两个值， 当它们相等， 显示在{% ifequal %} 和 {% endifequal %} 之中所有值
下面的例子比较两个模板变量user 和 currentuser
  {% ifequal user currentuser %}
      <h1> welcome </h1>
  {% endifequal %}
和 {% if %}类似， {% ifequal %} 支持可选的{% else %}标签
  {% ifequal section 'sitenews' %}
      <h1> Site News </h1>
   { else }
         <h1> No News Here </h1>
  {% endifequal %}
 其他标签
 csrf_token 用于跨站请求伪造保护， 防止跨站攻击
 {% csrf_token %}

3 注释标签
 单行注释 {# #}
 多行注释 {% comment %}...{% endcomment %}
```

4 过滤器

模板过滤器可以在变量被显示前修改它

语法{{变量|过滤器}}
过滤器使用管道字符|，例如{{name|lower}}，{{name}} 变量被过滤器lower处理后， 文档大写转换为小写

过滤管道可以被套接， 一个过滤管道的输出又可以作为下一个管道的输入， 例如{{my_list|first|upper}}，
将列表第一个元素装换为大写

过滤器参数
有些过滤器可以传递参数， 过滤器的参数跟随冒号之后并且总是以双引号包含。
例如：{{bio|truncatewords:"30"}}，截取显示变量的前30 个词。
{{my_list|join:","}}，将my_list的所有元素使用，逗号连接起来
其他过滤器

过滤器	说明	举例
first	取列表的第一个元素	
last	取列表最后一个元素	
yesno	变量可以是True，False，None， yesno:"yeah，no，maybe"}} yesno的参数给定逗号分隔的三个值， 返回3个值中的一个。 True多应第一个 False对应第二个 None对应第三个 如果参数只有2个， None等效False处理	{{value |
add	加法。参数是负数就是减法	数字加{{value | add:"100"}}
		列表合并 {{mylist | add: newlst}}
divisibleby	能否被整除	{{value | divisibleby:"3"}}， 能被3整除 返回True
addslashes	在反斜杠，单引号或者双引号前面加反斜杠	{{value| addslashes}}
length	返回变量的长度	{% if my_list | length>1 %}
default	变量等价False则使用缺省值	{{value | default:"nothing"}}
default_if_none	变量为None使用缺省值	{{value | default_if_none:"nothing"}}

date: 按指定的格式字符串参数格式化date或者datetime对象， 实例：

```
{{ pub_date|date:"n j, Y"}}
n 1~12月
j 1~31日
Y 2000年
```

练习，

```
<ul>
    {% for k, v in dct.items %}
        {% if forloop.counter0|divisibleby:"2" %}
        <li style="color:#FF0000">{{forloop.counter0}} {{k}} {{v}}</li>
        {% else %}
        <li style="color:#0000ff">{{forloop.counter0}} {{k}} {{v}}</li>
        {% endif %}
    {% endfor %}
</ul>

<ul>
    {% for k, v in dct.items %}
        <li style='color:{{forloop.counter0|divisibleby:"2"|yesno:"red,blue"}}'>
{{forloop.counter0}} {{k}} {{v}}</li>
    {% endfor %}
</ul>

 <ul>
 {% for k, v in dct.items %}
 <li class='{{forloop.counter0|divisibleby:"2"|yesno:"odd, even"}}'>{{forloop.counter0}} {{k}}
{{v}}</li>
 {% endfor %}
 </ul>
```

1.8 django用户接口设计之路由配置，视图函数

1 用户功能设计与实现

提供用户注册处理

提供用户登录处理

提供路由配置

2 用户注册接口设计

接受用户通过Post方法提交的注册信息， 提交的数据是json格式数据

检查email是否已存在于数据表中， 如果存爱返回错误的状态码，例如4xx, 如果不存在，

将用户提交的数据存入表中

整个过程都是采用AJAX异步过程， 用户提交json数据， 服务端获取数据后处理， 返回JSON

URL: /user/reg

METHOD: POST

3 路由配置

为了避免项目中的urls.py条目过多， 也为了让应用自己管理路由， 采用多级路由

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

include函数参数写应用.路由模块， 该函数就会动态导入指定的包的模块， 从模块里面读取urlpatterns

返回三元组，

url函数第二参数如果不是可调对象， 如果是元组或列表， 则会从路径中除去已匹配的部分， 将剩下部分与应用中的路由模块的urlpattern进行匹配

```
# 新建user/urls.py
from django.conf.urls import url
from .views import reg

urlpatterns = [
    url(r'^reg$', reg),
]
```

4 视图函数

在user/views.py中编写视图函数reg, 路由做相应的调整

```
import logging
FORMAT = "%(asctime)s %(threadName)s %(thread)d %(message)s"
logging.basicConfig(format=FORMAT, level=logging.INFO)

from django.http import JsonResponse, HttpRequest, HttpResponse, HttpResponseBadRequest
import json
import simplejson

def reg(request:HttpRequest):
    try:
        payload = json.loads(request.body)
        email = payload['email']
        # 数据库中看看email有没有
        return HttpResponse('welcome to django')
    except Exception as e:
        logging.info(e)
        raise HttpResponseBadRequest()
```

5 测试JOSN数据

使用POST方法， 提交的数据类型为application/json, json字符串要用双引号

这个数据是登录和注册的， 有客户端提交

```
{
  "password":"abc",
  "name":"tom",
  "email":"aas@wdwd.com"
}
```

6 JSON数据处理

simplejson 比标准库方便好用， 功能强大

```
pip install simplejson
```

浏览器端提交的数据放在了请求对象的body中， 需要使用simplejson解析， 解析的方式同json模块，但是simplejson更方便

7 错误处理

Django中有很多异常类， 定义在django.http下， 这些类都继承自HttpResponse.

```
# user/views.py中
from django.http import HttpRequest, HttpResponse, HttpResponseBadRequest, JsonResponse
import simplejson

def reg(request:HttpResponse):
        print(request.POST)
        print(request.body)
        payload = simple.loads(reqest.body)
        try:
                email = payload['email']
                name = payload['name']
                password = payload['password']
                print(email, name, password)
                return JsonResponse({}) # 如果正常， 返回json数据
        except Exception as e:       # 有任何异常， 都返回
                return HttpResponseBadRequest    # 这里返回实例， 这不是异常类
```

第二章 URL与视图

2.1 第一个django项目

创建项目
1. 通过命令行方式： 首先进入到安装的虚拟环境， 然后执行命令：
    django-admin startproject [项目名称]
2. 通过pycharm 的方式

运行项目：
1. python manage.py runserver， 这样可以在本地访问你得到网站， 默认端口号8000，
2. pycharm: 直接点击右上角的绿色三角箭头
注意：用pycharm 运行项目， 要避免一个项目运行多次
      在项目配置中， 把'只用单一实例' 那个选项勾选上

改变端口号：
1. 在终端运行时加上端口号： python manage.py runserver 9000, 这样可以通过9000端口来访问

让同局域网中的其他电脑访问本机的项目
1. 那么需要指定ip地址为 0.0.0.0
   示例为：python manage.py runserver 0.0.0.0:8000
   然后 修改 setting.py 文件 ALLOWED_HOSTS = ['192.168.43.117'] IP地址为本机ip地址
   注意： 关闭电脑防火墙

2.2 项目结构分析

1. manage.py
    以后和项目交互基本上都是基于这个文件， 一般都是在终端输入 python manage.py [子命令]
2. setting.py
    保存项目的所有配置信息
3. urls.py
    用来做url 与视图函数映射的， 以后来的一个请求， 就会从这个文件中找到视图函数
4. wsig.py
    专门用来做部署的， 不需要修改

2.3 视图函数

1. 视图函数的第一个参数必须是request, 这个参数绝对不能少
2. 视图函数的返回值必须是 'django.http.response.HttpResponseBase' 的子类的对象

2.4 url传参数

url 映射
1. 为什么会去urls.py 文件中寻找映射呢？
   是因为 setting.py 文件配置了 ROOT_URLCONF 为 urls.py. 所有django
   会去 urls.py 中寻找
2. 在 urls.py 中我们所有的映射， 都应该放在 urlpatterns 这个变量中
3. 所有的映射不是随便写的， 而是使用 path 函数或者是 re_path 函数进行包装的

url 传参数
1. 采用在url中使用变量的方式， 在path的第一个参数中， 使用 '<参数名>'的方式产地
   参数， 然后在视图函数中也要写一个参数，视图参数中的参数必须和 url 中的参数保持一致
   ， 不然就找不到这个参数， 另外， url 中可以传递多个参数
2. 采用查询字符串的方式： 在url中， 不需要单独的匹配查询字符串的部分， 只需要视图
   函数中使用'request.GET.get('参数名称')' 的方式来获取， 示例：
   ```
   def author_detail(request):
        author_id = request.GET.get('id')
        text = '作者的id是：%s' % author_id
        return HttpResponse(text)
   ```
   因为查询字符串使用的是 GET 请求， 所以我们通过request.GET 来获取参数，
   并且因为 GET 是一个类似字典的数据类型， 所有获取值跟字典的方式是一样的

2.5 url 命名

为什么需要url 命名？
  因为url是经常变化的， 如果在代码中写死可能会经常改代码，给url 取个名字
  以后使用url的时候就使用它的名字进行反转就可以了， 就不需要写死utl

如何给一个url指定名称
  在path 函数中， 传递一个name参数就可以指定。 示例代码如下：
  '''
  urlpatterns = [
    path('', views.index, name='index'),
    path('signin/', views.login, name='login'),
]
  '''

应用命名空间
  在多个app之间， 有可能产生同名url, 这个时候为了避免反转url的时候产生混淆， 就使用
  应用命名空间来区分， 只要在app的urls.py 中定义一个 app_name 的变量， 来指定这个应用的
  命名空间即可， 示例代码如下：
  '''
  #应用命名空间
  app_name = 'font'

   urlpatterns = [
       path('', views.index, name='index'),
       path('signin/', views.login, name='login'),
   ]
   '''
   以后在做反转的时候就可以使用 '应用命名空间: url名称' 的方式进程反转
   示例代码如下：
   '''
   login_url = reverse('font:login')
   '''

第三章 模板

3.1 模板介绍笔记

在之前的章节中，视图函数只是直接返回文本， 而实际生产环境中很少这样， 以为实际的页面大多是带有样式的HTML代码， 这可以让浏览器渲染出非常漂亮的页面。 目前市面上有很多的模板系统， 其中最知名最好用的就是DTL和J心急啊， DTL是Django Template Language 三个单词的缩写， 也是django自带的模板语言， 当然亦可以配置Django支持的Jinja2等其他模板引擎， 但是作为Django内置的模板语言， 和Django可以达到无缝衔接而不会产生一些不兼容的情况， 因此建议大家学好DTL

DTL与普通Html文件的区别：

DTL模板是带有特殊语法的HTML文件， 这个HTML文件可以被Django编译， 可以传递参数， 实现数据 动态化，在编译完成生成一个普通的HTML文件， 然后发送给客户端

3.2 渲染模板：

渲染模板有多中方式， 这里讲下两种常用的方式

1. render_to_string: 找到模板， 然后将模板编译后渲染成python的字符串格式， 最后再通过HttpResponse类包装成一个HttpRequest对象返回去， 示例代码：

```
from django.template.loader import render_to_string
from django.http import HttpResponse

# Create your views here.

def index(request):
    # html = render_to_string('index.html')
    # return HttpResponse(html)
```

2. 以上方式虽然已经很方便， 但是django还提供一个简单的方式， 直接将模板渲染成字符串和包装成HttpRequest对象一步到位

```
from django.shortcuts import render

# Create your views here.

def index(request):
    return render(request, 'index.html')
```

2.模板变量笔记：

```
1. 在模板中使用变量， 需要把变量放在 {{ 变量 }} 中
2. 如果要访问对象的属性， 通过 ‘对象.属性’ 的方式访问
3. 如果套访问一个字典key对应的value, 通过 ‘字典.key’来访问， 不能通过‘中括号[]’的形式来访问
4. 因为在访问字典的key 时候也是通过‘点’来访问， 因此不能在字典中定义字典本身有的属性名当做'key', 否则字典的属性将变成字典中的key了
5. 如果要访问列表或者元组， 那么也是通过‘点’的方式进行访问， 不能像python那样通过‘中括号[]’的形式来访问
context = {
        'username':'shuaibin',
        'person': p,
        'person_dic': {
            'username': '李四',
            'age': 19,
            'keys': 'ppww',
        },
        'persons': [
            '红楼梦',
            '西游记',
            '三国演义',
        ]
    }
```

3.常用的模板标签：

1. if标签： if标签相当于python中的if语句， 有elif和else 相对应， 但是所有的标签都需要标签符号 {% %} 来包裹， if标签中可以使用 ==, !=, <=, >=, not in, in, is, is not 等判断运算符， 示例代码如下：

```
{% if '张三' in person %}
        <p> 张三 </p>
{% else %}
        <p> 李四 </p>
{% endif %}
```

2. for .. in .. 标签： 类似于python的for .. in .. 可以遍历列表，元组， 字典等一切可遍历的对象， 示例代码如下：

```
{% for person in persons %}
        <p> person.name</p>
{% endfor %}
```

   如果要反向遍历， 在遍历的时候加上reversed, 示例代码如下：

```
{% for person in persons reversed %}
        <p> person.name</p>
{% endfor %}
```

   for .. in.. empty

```
    {% for comment in comments %}
        <li>{{ comment }}</li>
    {% empty %}
        <li>没有任何评论</li>
    {% endfor %}
```

3. url标签

url 标签： 在模板中使用类似反转的方式来实现，类似django中reverse一样， 示例代码如下：

```
<a href="{% url 'book:list' %}">图书列表页面</a>

如果url反转的时候需要传递参数， 那么可以在后面传递， 但是参数分位置参数， 和关键字参数，
位置参数和关键字参数不能同时使用， 示例代码如下：
#python部分
path(''detail/<book_id>/', views.detail, name='detail')

#url反转使用位置参数
<a href="{% url 'book:detail' 1 %}">图书详情页</a>

#url反转使用关键字参数
<a href="{% url 'book:detail' book_id=1 %}">图书详情页</a>

如果需要传递多个参数， 那么通过空格的方式进行分隔：
<a href="{% url 'book:detail' book_id=1 page=2 %}">图书详情页</a>
```

4.常用的模板过滤器

```
Django模板过滤器笔记

为什么需要过滤器
因为DTL中，不支持函数的调用形式'()'，因此不能给函数传递参数， 这将有很大的局限性，
而过滤器其实就是一个函数，可以接受一个参数（最多接受两个参数）

1.add过滤器
将传过来的参数添加到原来的值上面， 如果是整数就相加， 如果是字符串就拼接， 如果是列表就拼接成一个列表
{{ value1 | add: value2 }}

2.cut过滤器
类似于python里面的replace函数
{{ "hello world" | cut: " " }}

3.date过滤器
{{ today | date:"Y/m/d H:i:s" }}
时间格式见下表:
| 格式字符 | 描述 | 示例 |

| Y | 四位数的年份 | 2019 |

| m | 两位数的月份 | 01-12 |
| n | 月份 1-9前面没0 | 1-12 |

| d | 两位数的天 | 01-31 |
| j | 天， 1-9前面没0 | 1-31 |

| g | 小时， 12格式的， 1-9前面没0 | 1-12 |
| h | 小时， 12格式的， 1-9前面有0 | 01-12 |
| G | 小时， 24格式的， 1-9前面没0 | 1-23 |
| H | 小时， 24格式的， 1-9前面没0 | 0-23 |

| i | 分钟， 1-9前面没0 | 1-59 |

| s | 秒， 1-9前面没0 | 1-59 |
```

5.自定义过滤器

6.模板继承笔记

6.模板继承笔记
在前端页面开发中， 有些代码需要重复使用， 这种情况可以使用include标签实习， 也可以使用另外一个比较强大的范式来实现，
那就是模板继承， 类似于python总的类， 在父类中可以定义好一些变量和方法， 然后在子类中实现。
模板继承也可以在父模板中先定义好一些模板需要的代码， 然后在子模板中直接继承就可以了， 并且因为子模板有不同的代码，
因此可以在父模板中定义一个block接口， 然后在子模板中再去实现。代码如下：

```
{% extends 'base.html' %}

{% block content %}
    <p>{{ block.super }}</p>
    首页的代码
{% endblock %}
```

需要注意 extends 标签必须放在模板的第一行
子模板中的代码必须放在block中， 否则不会被渲染
如果在某个block中需要使用父模板的内容， 可以使用{{ block.super }} 来继承

在定义block的时候， 除了在block开始的地方定义这个block的名字， 还可以在block结束的时候定义名字，比如：

```
{% block title %} {% endblock title %}
```

这个在大型项目中显得尤其有用， 能够快速看到block包含在哪里

7.加载静态文件

加载静态文件

在一个网页中， 不仅仅只有一个html骨架， 还需要css样式文件， js执行文件以及一些图片等， 因此在DTL中加载静态文件是一个
必须要解决的问题， 在DTL中使用static标签加载静态文件， 要使用static标签， 首先需要 {% load static %}, 加载静态文件步骤如下：

1. 首先确保 django.contrib.staticfiles 已经添加到 settings.INSTALLED_APPS中

2. 确保settings.py 中设置了 STATIC_URL

3. 已经安装了的 app 下创建一个文件夹叫static, 然后再在这个static 文件夹下创建一个和当前app名字一样的文件夹， 在把静态文件放到这个文件夹下，

4. 如果一些静态文件是不和任何app挂钩， 那么可以在settings.py 中添加STATICFILES_DIRS， 以后DTL就会在这个列表的路径
   中查找文件， 比如可以设置为：
   ```
   STATICFILIES_DIRS = [os.path.join(BASE_DIR, 'static')]
   ```

5. 在模板中使用 load 标签加载 static 标签。 比如要加载static文件夹下的style.css文件， 示例代码：
   ```
   {% load static %}
   <link rel="stylesheet" href="{% static 'style.css' %}">
   ```

6. 如果不想每次在模板中加载静态文件都使用load加载static标签， 那么可以在setting.py中的TEMPLATES/OPTIONS 添加
   ```
   'builtins': ['django.templatetags.static']
   ```
   这样以后模板中就可以直接使用 static 标签， 而不用手动的load

7. 如果没有在 settings.INSTALLED_APPS 中添加 django.contrib.staticfiles。 那么我们就需要手动的将请求静态文件的url
   与静态文件的路径进行映射， 示例代码：
   ```
   from django.conf import settings
   from django.conf.urls.static import static

   urlpatterns = [
       # 其他url 映射
   ] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
   ```

第四章 django数据库

windows 下 MySQL下载与配置

1.1. 下载：

我下载的是64位系统的zip包：

下载地址：https://dev.mysql.com/downloads/mysql/

下载zip的包

1.2. 配置环境变量：

变量名：MYSQL_HOME

变量值：C:\Program Files\Java\mysql-8.0.13-winx64

path里添加：%MYSQL_HOME%\bin;

1.3. 生成data文件：

以管理员身份运行cmd

进入C:\Program Files\Java\mysql-8.0.13-winx64\bin 下

执行命令：mysqld --initialize-insecure --user=mysql

1.3.2. MySQl无法启动服务--Can't connect to MySQL server on localhost (10061)解决方法:
1.windows健+R 弹出运行对话框。输入regedit打开注册表编辑器
2.打开：k计算机名\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\
修改imagepath：为本机的路径 C:\Program Files\Java\mysql-8.0.13-winx64\bin\mysqld.exe

1.4. 启动服务：
执行命令：net start mysql 启动mysql服务，若提示：服务名无效...(后面有解决方法==步骤：1.5）；

1.5. 解决启动服务失败（报错）：
提示：服务名无效

解决方法：

执行命令：mysqld --install 即可（不需要my.ini配置文件 注意：网上写的很多需要my.ini配置文件，其实不需要my.ini配置文件也可以，我之前放置了my.ini文件，反而提示服务无法启动，把my.ini删除后启动成功了）

1.6. 登录mysql:
登录mysql:(因为之前没设置密码，所以密码为空，不用输入密码，直接回车即可）

C:\Program Files\Java\mysql-8.0.13-winx64\bin>mysql -u root -p

Enter password: ******

1.7. 查询用户密码:
查询用户密码命令：mysql> select host,user,authentication_string from mysql.user;

1.8. 设置（或修改）root用户密码：
设置（或修改）root用户密码：

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '123456';

#'123456',此处引号中的内容是密码，自己可以随便设置

Query OK, 1 row affected, 1 warning (0.00 sec)

Rows matched: 1  Changed: 1  Warnings: 1

mysql> flush privileges;  #作用：相当于保存，执行此命令后，设置才生效，若不执行，还是之前的密码不变

Query OK, 0 rows affected (0.01 sec)

1.9. 退出mysql:
mysql> quit

Bye
----------------------
作者：tao10180
来源：CSDN
原文：https://blog.csdn.net/tao10180/article/details/83781842
版权声明：本文为博主原创文章，转载请附上博文链接！
```

Django数据库笔记

4.1．MySQL驱动程序安装

常见的驱动程序介绍：

```
        1. MySQL-python: 是对c语言操作MySQL数据库的一个简单封装，目前只支持python2.
        2. mysqlclient: 是MySQL-python的另外一个分支， 支持python3
        3. pymysql: 是纯python实现的一个驱动， 执行效率没有MySQL-python 快
        4. MySQL connector/Python: MySQL官方推出的使用纯python链接MySQL的驱动， 因为纯Python开发的，效率不高
```

4.2. 操作数据库

Django配置链接数据库：

```
DATABASES = {
    'default': {
        # 数据库引擎（mysql 或者 oracle ）
        'ENGINE': 'django.db.backends.mysql',
        # 数据库名字
        'NAME': 'django_db1',
        # 连接mysql数据库的用户名
        'USER': 'root',
        # 连接mysql数据库的密码
        'PASSWORD': 'root',
        # mysql数据库的主机地址
        'HOST': '127.0.0.1',
        # mysql数据库的端口号
        'port': '3306',
    }
}
```

Django中操作数据库

有两种方式： 第一种就是使用原生sql语句， 第二种就是使用ORM模型操作

第一种：

```
from django.db import connection


cursor = connection.cursor()
# 插入数据
cursor.execute("insert into book(id, name, author) values(null, '%s', '%s')" % (name, author))
# 查询数据
cur.execute("select id, name, author from book")
books = cur.fetchall()
# 删除数据
cursor.execute("delete from book where id='%s'" % book_id)
# 更新数据
cursor.execute("update book set name='%s', author='%s' where id='%s'" % (name, author, book_id))
book = cursor.fetchone()

'''
flush privileges;  提交
'''
```

4.3. ORM模型介绍

随着项目越来越大， 采用原生SQL语句的方式代码中会出现大量的问题

1. SQL语句重复利用不高， 越复杂的SQL语句条件越多， 代码越长， 会出现很多相近的SQL语句.
    1. 很多SQL语句是业务逻辑拼出来的， 数据库需要修改，就要修改这些逻辑，容易漏掉对某些SQL语句的更改。
    2. 写SQL容易忽略web安全问题， 造成未知的隐患

ORM, 全称 Object Relational Mapping 叫做关系映射。通过把表映射成类， 把字段映射为属性， ORM执行队形操作的时候最终还是会把对应的操作转换为数据库原生语句。优点：

1. 易用性：较少重复的SQL语句概率
2. 性能损耗小：开发效率高， 代码可读性好
3. 设计灵活：可以轻松写出复杂的查询
4. 可移植性：很轻松的更换数据库

ORM模型的创建和映射：

创建ORM模型

```
    orm 模型一般都放在 ‘app'的’models.py'文件中
    需要在settings.py的'INSTALLED_APP' 中进行安装
```

```
from django.db import models

# Create your models here.
class Book(models.Model):
    id = models.AutoField(primary_key=True)
    title = models.CharField(max_length=100, null=False)
    author = models.CharField(max_length=100, null=False)
    price= models.FloatField(null=False, default=0)

# 1 makemigrations命令 生成迁移脚本文件
# python manage.py makemigrations
# 2 migrate命令 将生成的迁移脚本文件映射到数据库
# python manage.py migrate
```

ORM对数据库的基本操作

ORM 增删改查：

```
ORM对数据库的基本操作

添加数据
    book = Book(name='三国演义', author='罗贯中', price=199)
    book.save()
查找数据
    2.1. pk根据主键进行查找
    book = Book.objects.get(pk=1)
    print(book)
    2.2 根据其他条件进行查找
    books = Book.objects.filter(name='西游记').first()
    print(books)

删除数据
    book = Book.objects.get(pk=2)
    book.delete()

修改数据
    book = Book.objects.get(pk=3)
    book.price = 300
    book.save()
```

4 .4. ORM模型常用字段

模型常用属性

AutoField

```
# 整型
```

BigAutoField

```
# 长整型
```

BooleanField:

```
# 在定义字段的时候，如果没有指定null=True，那么默认情况下， null=False
# 就是不能为空
# 如果要使用可以为null的BooleanField，那么应该使用NullBooleanField (可以为空)
# 来代替BooleanField
```

CharField

```
# CharField，默认必须定义max_length属性， 如果超过254个字符， 就不建议使用了
# 就推荐使用TextField，
```

DateField

```
# 时间类型， 在数据库层是time类型， 在python中是datetime.time类型
```

EmailField

```
# 类似于CharField，在数据库层也是一个varchar类型， 最大长度是254个字符
```

FileField

```
# 用于存储文件的， 参考后面文件上传章节
```

ImageField

```
# 用于存储图片文件的， 参考后面图片上传章节
```

FloatField

```
# 浮点类型， 映射到数据库是float类型
```

IntegerField

```
# 整型， 值区间 -2147483648 -- 2147483647
```

BigIntegerField

```
# 大整型， 值区间 -9223372036854775808 -- 9223372036854775807
```

PositiveIntegerField

```
# 正整型， 值区间 0 -- 2147483647
```

SmallIntegerField

```
# 小整型， 值区间 -32768 -- 32767
```

PositiveSmallIntegerField

```
# 正小整型， 值区间 0 -- 32767
```

TextField

```
# 大量的文本类型， 映射到数据库是longtext 类型
```

UUIDField

```
# 只能存储uuid,格式的字符串， uuid是一个32位的全球唯一的字符串， 一般用来做主键
```

URLField

```
# 类似于CharField，只不过只能来存储url格式的字符串， 并且默认的max_length是200
```

Field的常用参数：

```
null = True 或者 False
```

4.5. 外键和表关系：

1 外键：

在MySQL中， 表有两种引擎， 一种是InnoDB，另外一种是myisam。 如果使用的是InnoDB引擎， 是支持外键约束的， 外键的存在使得ORM框架在处理表关系的时候异常的强大， 因此这里我们首先来介绍下外键在Django中的使用。

1. 类定义为 class ForeignKey(to, on_delete, **options)。 第一个参数值引用的是哪个模型， 第二个参数是在使用外键引用的模型数据被删除了， 这个字段该如何处理， 比如有一个CASCADE， SET_NULL等， 这里以一个实际案例来说明。 比如有个User 和一个Article 两个模型， 一个User可以发表多篇文章， 一个Article只能有一个Author， 并且通过外键进行引用， 那么相关的示例代码如下：

```
class User(models.Model):
        username = models.CharField(max_length=20)
        password = models.CharField(max_length=100)

class Article(models.Model):
        title = models.CharField(max_length=100)
        content = models.TextField()

        author = models.ForeignKey("User", on_delete=models.CASCADE)
```

以上使用 ForeignKey 来定义模型之间的关系， 即在article的实例中可以通过author属性来操作对应的User模型，这样使用起来非常方便， 实例代码如下：

```
article = Article(title='abc', content='123qwe')
author = User(username='张三', password='123456')
article.author = author
article.save()
```

1. 如果想引用另外一个app的模型， 那么应该在传递to参数的时候， 使用app.model_name进行指定， 以上示例为例， 如果User 和 Article不是在同一个app 中， 那么引用的时候示例代码如下：

```
class User(models.Model):
        username = models.CharField(max_length=20)
        password = models.CharField(max_length=100)

class Article(models.Model):
        title = models.CharField(max_length=100)
        content = models.TextField()

        author = models.ForeignKey("user.User", on_delete=models.CASCADE)
```

1. 如果模型的外键引用的是本身自己这个模型， 那么to参数可以为 'self', 或者是这个模型的名字， 在论坛开发中， 一般评论都可以进行二级评论， 即可以针对另外一个评论进行评论， 那么定义模型的时候就需要使用外

键来引用自身， 代码如下：

```
class Comment(models.Model):
        content = models.TextField()
        orign_comment = models.ForeignKey('self', on_delete=models.CASCADE, null=True)
        # 或者
        # orign_comment = models.ForeignKey('Comment', on_delete=models.CASCADE, null=True)
```

外键删除操作：

如果一个模型使用的外键， 那么在对方那个命被删除后， 该进行什么样的操作， 可以通过 on_delete 来指定， 可以指定的类型如下：

1. CASCADE: 级联操作， 如果外键对应的那条数据被删除了， 那么这条数据也会被删除
2. PROTECT: 受保护， 即只要这条数据引用了外键的那条数据， 那么不能删除外键的那条数据
3. SET_NULL: 设置为空， 如果外键的那条数据被删除了， 那么在本条数据上就将这个字段设置为空。 如果设置这个选项， 前提是要指定这个字段可为空
4. SET_DEFAULT: 设置默认值, 如果外键的那条数据被删除了， 那么本条数据将这个字段设置为默认值， 如果设置这个选型， 前提是要指定这个字段为默认值
5. SET(): 如果外键那条数据被删除了， 那么将会获取SET函数中的值作为这个外键的值。 SET函数可以接受一个调用的对象（比如函数或方法), 如果是可以调用的对象， 那么会将这个对象调用的结果作为值返回去
6. DO_NOTHING: 不采取任何行为， 一切全看数据库级别的约束

2 表关系：

表之间的关系是通过外键来进行关联的， 表之间的关系无非就三种关系： 一对一， 一对多， 多对多

3 一对多：

1. 应用场景： 比如文章和作者之间的关系。 文章只能有一个作者写， 但是一个作者可写多篇文章， 文章和作者的关系就是典型的一对多的关系
2. 实现方式： 一对多或者多对一， 都是ForeignKey来实现的， 还是以文章与作者的案例进行讲解：

```
class User(models.Model):
    username = models.CharField(max_length=20)
    password = models.CharField(max_length=100)

class Article(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()

    author = models.ForeignKey("User", on_delete=models.CASCADE)
```

1 添加文章

```
article = Article(title='闰土', content=r'1gwvwvwvw11111111wd')
author = User(username='admin', password='111')
author.save()
article.author = author
article.save()
```

并且以后如果想要获取某个用户的所有文章， 可以通过article_set来实现 示例代码如下：

```
user = User.objects.first()
articles = user.article_set.all()
for article in articles:
        print(article)
```

2 添加文章，

如果现想要将文章添加某个分类， 可以使用一下方式：

```
category = Category.objects.first()
article = Article('title'='111', content='qqqqqq')
article.author = FrontUser.objects.first()

category.articles_set.add(article, bulk=False)
```

使用bulk=False

那么Django会自动的保存article, 而不需要再添加category之前先保存article

或者是另外一种解决方式是， 在添加到category.article_set 之前， 先将article保存到数据库中， 但是如果article_category不能为空， 那么久产生一种死循环了， article没有category, 不能保存， 而将article添加到category_article_set中， 又需要article之前已经存储到数据库中的

如果是上面的需求， 建议使用bulk=False解决方案

4 多对多：

[Django数据库多对多数据关系]

本文主要描述Django数据中一对一的数据关系，创建app等配置不再赘述。

Django操作数据库，使用ORM

```
ORM：关系映射对象，把传统的SQL语句封装成了类和对象的形式，在操作表中的记录时，就像在操作类和对象一样。
```

使用默认数据库：sqlite

操作在models.py文件中完成。

多对多关系

举例：一个出版社发表多个文章，一篇文章也可以对多个出版社发表

ManyToManyFiled():关联两个表的函数，但是函数的设置位置，在模型中任选一个即可，不能在两个模型中都设置此函数。

```
class Publication(models.Model):
    id = models.AutoField(primary_key=True)
    title = models.CharField(max_length=30, null=True)  # 出版社名称

    class Meta:
        db_table = 'Publication'

    def __unicode__(self):
        return 'Publication-Name: %s' % self.title


class Article(models.Model):
    id = models.AutoField(primary_key=True)
    headline = models.CharField(max_length=100, null=True)

    # 让文章关联出版社
    publication = models.ManyToManyField(Publication)

    class Meta:
        db_table = 'Article'

    def __unicode__(self):
        return 'Article-Name: %s' % self.headline
```

创建完数据库模型之后，就需要创建数据库变更文件，应用数据库变更文件创建表的操作。
创建数据库变更文件：python manage.py makemigrations
应用数据库变更文件：python manage.py migrate
可以使用SQLite可视化工具查看已经建立成功的数据库表

由于未进行其他配置，所以只能在shell环境下，对数据库进行操作。
进入shell环境的命令：python manage.py shell

注意点：

1）注意ManyToManyField（）的设置位置；该实例中设置了Article模型中，在绑定两张表的时候，利用Article的对象a1进行绑定的，a1.publication.add()。但是不能使用Publication的对象p1进行绑定(p1.article.add())。

两张表之间的相互查询关系

查询a1这篇文章所属的出版社有哪些

print a1.publication.all()

查询某一出版社，所包含的所有文章有哪些

print p1.article_set.all()

4.6. 增删改查操作

6.1 数据更新:
Article.objects.filter(id=id).update(name=newname)

6.2 数据去重:

```
# 数据库去重，重复的数据保留一条
def delete_duplicate(self, start, end):
    # distinct() 方法去重
    novel = self.models.objects.values('articleId').distinct()
    for i in range(start, end):
        li = Novel.objects.filter(articleId=novel[i]['articleId'])
        if li.count() > 1:
            print(li)
            for item in li[1:]:
                print(item)
                item.delete()
```

4.7 查询操作

查找是数据库操作中一个非常重要的技术， 查询一般就是使用filter, exclude以及get 三个方法。 我们可以在调用这些方法的时候传递不同 的参数实现查询， 在orm层面， 这些查询条件都是使用field + ___ +condition, (即字段的名字+两个下划线+查询规则)的方式来使用的， 常用的查询条件如下:

exact:

精确的提供条件， 如果提供的是一个None， 那么在SQL层面就是解释为NULL。示例代码如下：

```
article = Article.ojects.get(id__exact=2)
article = Article.objects.get(id__exact=None)
```

print(article.query) 会打印出sql语句

以上的两个查找翻译成sql语句如下：

```
select .... from article where id=2
select .... from article where id IS NULL
```

iexact:

使用like进行查找示例代码如下：

```
article = Article.objects.filter(title__iexact='hello world')
```

以上的查找翻译成sql语句如下：

```
select .. from article where id like ..
```

contains:

```
contains  包含的意思  区分大小写
article = Article.objects.filter(title__contains='hello world')
```

icontains:

```
icontains  包含的意思  不区分大小写
```

4.8 聚合函数：

- 聚合函数都是放在 django.db.models 下面
- 聚合函数不能单独的执行， 需要放在一些可以执行聚合函数的方法下面去执行 比如：

```
result = Book.objects.aggregate(Avg('price'))
```

- 聚合函数执行完成后， 给这个聚合函数的值取个名字， 取名字的规则，默认是"field+__+聚合函数名字", 如果不想使用默认名字， 那么可以在使用聚合函数时候传递关键字参数进去， 参数的名字就是聚合函数执行完成的名字， 示例代码如下：

```
result = Book.objects.aggregate(avg=Avg('price'))
```

以上传递了关键字参数 avg=Avg('price'), 以后Avg聚合函数的名字就叫做avg

- 'aggregate'： 这个方法不会返回一个"QuerySet'对象， 而是返回一个字典， 这个字典的key就是聚合函数的名字1

1. Avg: 求平均值， 比如想要获取所以图书的平均价格， 代码如下：

```
from django.db.models import Avg
result = Book.objects.aggregate(my_avg=Avg('price'))
print(result)
```

4.9. QuerySet API

返回新的QuerySet的方法：

filter/exclude/annotate: 过滤/排除满足条件的/给模型添加新的字段

在使用QuerySet进行查找的时候， 可以提供的各种操作， 比如过滤完后还有根据某个字段进行排序， 那么这一系列的操作我们可以通过一个非常流畅的 '链式调用' 的方法来进行， 比如要从-文章表中获取标题为123， 并且提取后要将结果根据发布的时间进行排序：

```
articles =Article.objects.filter(title='123').order_by('create_time')
```

可以看出order_by 方法是在filter执行后调用的， 这说明filter返回的对象是一个拥有order_by方法的对象， 而对象正是一个新的QuerySet对象， 因此可以使用order_by方法

下面介绍那些返回新的QuerySet对象的方法：

1. filter: 将满足条件的数据提取出来， 返回一个新的QuerySet对象
2. exclude: 排除满足条件的数据， 返回一个新的QuerySet， 示例代码：

```
articles = Article.objects.exclude(title__contains='hello')
```

以上代码的意思是提取那些标题不包含hello的图书

1. annotate: 给QuerySet中的每个对象添加一个使用的查询方式(聚合函数， F表达式， Q表达式， Func表达式等) 的新字段， 示例代码如下：

```
articles = Article.objects.annotate(author__name=F("author__name"))
```

以上代码将在每个对象中添加一个 author__name 字段， 来显示这个文章的作者的年龄

1. order_by: 指定查询的结果根据某个字段进行排序， 如果要倒叙排序， 那么可以在字段前加个符号。 示例代码如下：

```
根据创建的时间进行正序排序
articles = Article.objects.order_by('create_time')
根据创建的时间进行倒叙排序
articles = Artcle.objects.order_by('-create_time')
首先根据创建时间排序， 如果时间相同， 则根据作者的名字进行排序
articles = Article.objects.order_by('create_time', 'author_name')
根据图书销量从大到小进行排序
books = Book.objects.annotate(order_nums=Count('bookorder')).order_by('-order_nums')
```

一定要注意的一点：多个order_by, 会把前面排序的规则给打乱， 而使用后面的排序方式， 代码如下：

```
article = Article.objects.order_by('create_time').order_by('author__name')
```

它会根据作者的名字进行排序， 而不是根据创建时间

当然， 也可以在模型定义中 在'Meta' 类中定义'ordering', 来默认的排序方式：

```
class Meta:
    db_table = 'book_order'
    # 默认根据create_time 从大到小进行排序，如果create_time相等， 则根据price从大到小排序
    ordering = ['-create_time', '-price']
```

1. values:

4.10 f orm表单：

CharField:

参数：

- max_length: 这个字段的长度
- min_length: 这个字段的最小长度
- requried: 这个字段是否是必须的
- error_messages: 在某个条件验证失败的时候， 给出错误信息

EmailField:

用来接受邮件， 会自动验证邮件是否合法

错误信息key: requried, invalid

IntegerField:

用来接受整类型， 如果验证通过， 会将这个字段的值转换为整类型

参数：

- max_length: 最大值
- min_length: 最小值

错误信息的key: requried, invalid, max_value, min_value

FloatField:

用来接受浮点类型， 如果验证通过， 会将这个字段的值转换为浮点类型

参数：

- max_length: 最大值
- min_length: 最小值

错误信息的key: requried, invalid, max_value, min_value

URLField:

用来接受 url 格式的字符串

错误信息key: requried, invalid

4.10 常用的验证器：

需要引入一下： from django.core import validators

在验证某个字段的时候， 可以传递一个 validator 参数来指定验证器， 进一步对数据进行过滤， 验证器有很多，但是很多验证器我们其实可以通过这个Field或者一些参数就可以指定了， 比如EmailValidator, 我们可以通过EmailField来指定， 比如MaxValueValidator, 我们可以通过max_value参数来指定， 以下是常用的验证器：

1. MaxValueValidator: 验证最大值
2. MinValueValidator: 验证最小值
3. MinLengthValidator: 验证最小长度
4. MaxLengrhValidator: 验证最大长度
5. EmailValidator: 验证是否是邮箱格式
6. URLValidator: 验证是否是url格式
7. RegexValidator: 如果需要更加复杂的验证， 那么我们可以通过正则表达式的验证器： RegexValidator. 比如现在要验证手机号格式是否合格：

```
from django.core import validators
class MyForm(forms.Form):
        telephone = forms.Charfield(vallidators=[validators.RegexValidator("1[345678]\d{9}",
message='请输入正确格式的手机号！'])
```

4.11 自定义验证：

有时候对一个字段验证，不是一个长度， 一个正则表达式能够写清楚的， 还需要其他复杂的验证逻辑， 那么我们可以对某个字段，进行自定义验证， 比如注册的表单验证， 我们想验证手机号是否被注册过， 那么这时候需要在数据库中进行判断猜知道， 对某个字段进行验证的验证方式， 定义一个方法， 这个方法的名字定义规则是: clean_fieldname。 如果验证失败， 那么抛出一个验证错误， 示例：

```
class MyForm(forms.Form):
        telephone = forms.CharField(validators=[validators.RegexValidator('1[345678]\d{9}',
message='请输入正确格式的手机号！')])

        def clean_telphone(self):
                telphone = self.cleaned_data.get('telphone')
                exists = User.objects.filter(telphone=telphone).exists()
                if exists:
                        raise forms.ValidationError('手机号已经有存在！')
                return telphone
```

4.12. pycharm 连接数据库:

mysql驱动下载地址: https://dev.mysql.com/downloads/connector/j/

4.13 如何向数据库一次性插入多条数据

1 方法一：效率极低，不推荐使用

```
for i in range(1000):
    models.Book.objects.create(title=f'第{i}本书')
```

2 方法二

bulk_create

```
book_list = []
for i in range(100000):
    book_list.append(models.Book(title=f'第{i}本书'))
models.Book.objects.bulk_create(book_list)
```

models.Book.objects.bulk_create(book_list)

第五章 前端开发环境配置

5. 1 nvm安装：

nvm (Node Version Manager) 是一个用来管理node版本的工具， 我们之所以需要使用node, 是因为我们需要使用node中的npm(Node Package Manager), 使用npm的目的是为了能够方便管理一些前端开发的包， nvm的安装分成简单， 步骤如下：

1. nvm的安装包链接： https://github.com/coreybutler/nvm-windows/releases
2. 然后点击一顿下一步， 安装即可

3. 安装完成需要配置环境变量， 在我的电脑-》属性-》高级系统设置-》环境变量-》系统环境变量-》Path下新建一个， 把nvm所处的路径添加进去
4. 打开cmd, 然后输入nvm, 如果没有提示没有找到这个命令， 说明安装成功
5. Mac或者Linux安装nvm请看这里： https:github.com/creationix/nvm。 也要记得配置环境

nvm常用命令：

1. nvm install node： 安装最新版的node.js。 nvm i == nvm install
2. nvm insall (version): 安装指定版本的node.js
3. nvm use (version): 使用某个版本的node.js
4. nvm list: 列出当前安装的那些版本的node
5. nvm uninstall (version) : 卸载之指定版本的node
6. nvm node_mirror [url]： 设置nvm镜像
7. nvm npm_mirror [url]： 设置npm镜像

5.2 npm安装：

npm (Node Package Manager) 在安装node的时候会自动的安装。但是前提条件是你需要设置当前的node版本：nvm use 6.4.0, 然后可以使用npm

5.3 安装包：

安装包分为全局安装和本地安装， 全局安装是在安装在当前的环境中， 在可以cmd中当作命令使用。 而本地安装时安装在当前项目中， 只有在当前项目中能使用， 并且可以通过require引用， 安装参数只有 -g 参数的区别：

```
npm install express     # 本地安装
npm install express -g  # 全局安装
```

5.4 本地安装：

1. 将安装包放在 ./node_modules 下（运行npm命令时所在的目录）， 如果没有node_modules 目录， 会在当前执行npm命令的目录下生成node_modules
2. 可以通过require()来引入本地安装的包

5.5 全局安装：

1. 将安装包放在 /usr/local 下或者你的node的安装目录下
2. 可以直接在命令行中使用

5.6 卸载包：

```
npm uninstall [package]
```

5.7 更新包：

```
npm update [package]
```

5.8 搜索包：

```
npm search [package]
```

5.9 使用淘宝镜像：

npm install -g cnpm --registry = https://registry.npm.taobao.org

那么以后可以通过cnmp来安装包了！

5.10 前端项目搭建

前端我们使用gulp来自动化开发流程。 配置好gulp后可以自动给我们处理好一些工作。 比如写完css后， 要压缩成.min.css， 写完js后， 要混淆和压缩, 图片压缩等， 这些工作都可以让gulp帮我们完成。

5.11 安装gulp:

1. 创建本地宝管理环境：

   使用 **npm init** 命令在本地生成一个 **package.json** 文件， **package.json**是用来记录你当前这个项目依赖了那些包， 以后别人拿到你这个项目后， 不需要你的**node_modules**文件夹**(**因为**node_modules**中的包是在台庞大**)**。只需要执行**npm install** 命令， 即会自动化安装**package.json**下**devDependencies**

2. 安装gulp:

   gulp 的安装非常简单， 只要使用**npm**命令安装即可， 但是因为**gulp**需要作为命令的方式运行， 因此需要在安装在系统级别的目录中：

   ```
   npm install gulp -g
   ```

   因为在本地需要使用resquire 的方式gulp, 因此也需要在本地安装一份：

   ```
   npm install gulp --save-dev
   ```

   今天用学习用到了gulp安装的gulp 4.0版本 代码跟老师一样但是就是不行提示以下错误：
   gulp did you forget to signal async completion
   the following tasks did not complete:任务
   把版本重新安装3.9.1就可以快速解决
   npm install --save-dev gulp@3.9.1

```
3. ##### 创建gulp任务：

   ##### 使用gulp来流程化我们的开发工作。 首先需要在项目的根目录下创建一个gulpfile.js文件。 然后在gulpfile.js中填入一下代码：
```

```
var gulp = require("gulp")

gulp.task("greet", function(){

  console.log("hello world")

})
```

这里对代码的解释：

1. 通过require 语句引进已经安装的第三方依赖包。 这个require只能是引用当前项目的， 不能引用全局下的。require语句是node.js独有的， 只能在node.js环境下使用
2. gulp.task 是用来创建一个任务. gulp.task 的第一个参数是命令的名字， 第二个参数是一个函数， 就是执行这个命令的时候会做什么事情， 都是写在这个里面的
3. 写完以上代码后， 以后如果想要执行greet命令， 那么只需要进入到项目所在的目录的路径， 然后终端使用gulp greet 即可执行

4.创建处理css文件的任务：

gulp 只是提供一个框架给我们， 如果我们要实现更加复杂的功能， 比如css压缩， 那么我们还需要安装一下gulp-cssnano插件。 gulp 相关的插件安装也是通过npm命令安装的， 安装方式跟其他包是一模一样(gulp插件本身就是一个普通的包)。

对css文件处理， 需要做的事情就是压缩， 然后将压缩后的文件放到指定目录下（不要和原来的css文件重合）！这里我们使用gulp-cssnano来处理这个工作：

```
npm install gulp-cssnano  --save-dev
```

然后在gulpfile.js 中写入一下代码：

```
var gulp = require("gulp")
var cssnano = reuire("gulp-cssnano")

//定义一个处理css文件改动的任务
gulp.task("css", function(){

 gulp.src("./css/*.css")
 .pipe(cssnano())
 .pipe(gulp.dest("./css/dist/"))

})
```

以上代码进行详细解释：

1. gulp.task: 创建一个gulp处理任务
2. gulp.src: 找到当前css目录下所有的以.css结尾的css文件
3. pipe: 管道方法。将上一个方法返回结果传给另外一个处理器， 比如以上的cssnano.
4. gulp.dest: 将处理虎的文件， 放到指定的目录下， 不要放在和原文件相同的目录， 以免产生冲突， 也不方便管理

5.修改文件名：

像以上任务，压缩完css文件以后， 最好是给他添加一个.min.css的后缀，这样一眼就能知道这个是经过压缩后的文件。这时候我们需要使用gulp-rename来修改了， 当然首先需要安装npm install gulp-rename --save-dev。 示例代码如下：

```
var gulp = require("gulp")
var cssnano = require("gulp-cssnano")
var rename = require("gulp-rename")

gulp.task("css", function(){

 gulp.src("./css/*.css")
 .pipe(cssnano())
 .pipe(rename({"suffix": ".min"}))
 .pipe(gulp.dest("./dist/css/"))

})
```

在上述代码中， 我们增加了一行 `.pipe(rename({"suffix": ".min"}))`， 这样我们就是使用`rename`方法， 并且传递一个对象参数， 指定修改名字的规则为添加一个`.min`后缀名。这个`gulp-rename`还有其他的指定文件名 的方式， 比如可以在文件名前加个后缀， 更多的教程可以看这里： `https://www.npmjs.com/package/gulp-rename`

6.自动添加cs样式前缀：

一些`css3`的样式， 不同的浏览器处理的时候是不一样的， 比如`box-shadow`，是用来指定元素阴影的。 但是一些老版本的浏览器， 需要添加相应的前缀。比如：

```
// Firefox4.0-
-moz-box-shadow: 10px 10px 5px #888888;
// Safari and  Google chrome10.0-
-webkit-box-shadow: 10px 10px 5px #888888;
// Forefox4.0+，Google chrome10.0+， Opera10.5+  and  IE9
box-shadow: 10px 10px 5px #888888;
```

7.gulp示例用法：

```
// "devDependencies": {
//   "browser-sync": "^2.26.5",
//   "gulp": "^3.9.1",
//   "gulp-autoprefixer": "^6.1.0",
//   "gulp-cache": "^1.1.1",
//   "gulp-concat": "^2.6.1",
//   "gulp-concat-folders": "^1.3.1",
//   "gulp-cssnano": "^2.1.3",
//   "gulp-imagemin": "^5.0.3",
//   "gulp-rename": "^1.4.0",
//   "gulp-uglify": "^3.0.2"
// }
```

```
var gulp = require("gulp");
var cssnano = require("gulp-cssnano");
var rename = require("gulp-rename");
var uglify = require("gulp-uglify");
var concat = require("gulp-concat")
var cache = require("gulp-cache")
```

```
var imagemin = require("gulp-imagemin")
var bs = require("browser-sync").create()

var path = {
    'html': './templates/**/',
    'css': './src/css/',
    'js': './src/js/',
    'images': './src/images/',
    'css_dist': './dist/css/',
    'js_dist': './dist/js/',
    'images_dist': './dist/images/',
}

// 处理html文件的任务
gulp.task('html', function () {
    gulp.src(path.html + '*.html')
        .pipe(bs.stream())
})

// 定义一个css任务
gulp.task("css", function () {
    gulp.src(path.css + "*.css")
        .pipe(cssnano())
        .pipe(rename({"suffix": ".min"}))
        .pipe(gulp.dest(path.css_dist))
        .pipe(bs.stream())
})

// 处理js文件的任务
gulp.task("js", function () {
    gulp.src(path.js + "*.js")
        .pipe(uglify())
        .pipe(gulp.dest(path.js_dist))
        .pipe(bs.stream())
})

// 定义处理图片文件任务
gulp.task("images", function () {
    gulp.src(path.images + "*.*")
        .pipe(cache(imagemin()))
        .pipe(gulp.dest(path.images_dist))
        .pipe(bs.stream())
```

})

// 定义监听文件修改的任务
gulp.task("watch", function () {

```
    gulp.watch(path.html + "*.html", ['html']);
    gulp.watch(path.css + "*.css", ['css']);
    gulp.watch(path.js + "*.js", ['js']);
    gulp.watch(path.images + "*.*", ['images']);
```

})

// 初始化 browser-sync的任务
gulp.task('bs', function () {

```
    bs.init({
        'server': {
            'baseDir': './'
        }
    })
```

})

// 创建一个默认的任务
gulp.task('default', ['bs', 'watch'])

```
##### iconfont图标库：https://www.iconfont.cn

##### 马克鳗：https://markman.com
```

第六章 AJAX提交数据

6.1 ajax登录示例

urls.py

```
from django.conf.urls import url
from django.contrib import admin
from app01 import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^login_ajax/$', views.login_ajax, name='login_ajax'),
    url(r'^index/$', views.index, name='index'),
]
```

views.py

```
from django.shortcuts import render, HttpResponse, redirect
import json


def index(request):
    return HttpResponse('this is index')


def login_ajax(request):
    if request.method == "POST":
        user = request.POST.get("user")
        pwd = request.POST.get("pwd")
        ret = {"status": 0, 'url': ''}
        if user == "alex" and pwd == "123":
            ret['status'] = 1
            ret['url'] = '/index/'
        return HttpResponse(json.dumps(ret))

    return render(request, "login_ajax.html")
```

login_ajax.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="x-ua-compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>登录</title>
    <link rel="stylesheet" href="/static/plugins/bootstrap-3.3.7-dist/css/bootstrap.css">
    <link rel="stylesheet" href="/static/css/signin.css">
</head>
<body>

<div class="container">

    <form class="form-signin" action="{% url 'login' %}" method="post">
        {% csrf_token %}
        <h2 class="form-signin-heading">请登录</h2>
        <label for="inputUser" class="sr-only">用户名</label>
        <input type="text" id="inputUser" class="form-control" placeholder="用户名" required=""
autofocus="" name="user">
        <label for="inputPassword" class="sr-only">密码</label>
        <input type="password" id="inputPassword" class="form-control" placeholder="密码"
required="" name="pwd">
        <div class="checkbox">
            <label>
                <input type="checkbox" value="remember-me"> 记住我
            </label>
        </div>
        <input class="btn btn-lg btn-primary btn-block" id="login" value="登陆">
    </form>

</div> <!-- /container -->


<script src="/static/jquery-3.3.1.min.js"></script>
<script>

    $('#login').click(function () {
        $.ajax({
            url: '/login_ajax/',
            type: 'post',
            data: {
                csrfmiddlewaretoken: $("[name='csrfmiddlewaretoken']").val(),
                user: $('[name="user"]').val(),
                pwd: $('[name="pwd"]').val()
            },
            success: function (data) {
                data = JSON.parse(data);
                if (data.status) {
                    window.location = data.url
                }
                else {
                    alert('登陆失败')
```

```
                }
            }
        })
    })
</script>

</body>
</html>
```

静态文件需要配置，使用了jQuery和Bootstrap。

6.2 CSRF跨站请求伪造

方式一

将

```
csrfmiddlewaretoken: $("[name='csrfmiddlewaretoken']").val()
```

放在POST的请求体中。

示例中就是使用的这种方式。

方式二

给ajax的请增加X-CSRFToken的请求头，对应的值只能是cookie中的csrftoken的值。

所以我们要从cookie中提取csrftoken的值，jQuery不能去cookie，我们使用jquery.cookie的插件。点击下载jquer.cookie插件。

HTML中导入jquery.cookie.js。

```
<script src="/static/jquery-3.3.1.min.js"></script>
<script src="/static/jquery.cookie.js"></script>
<script>

    $('#login').click(function () {
        $.ajax({
            url: '/login_ajax/',
            type: 'post',
            headers:{ "X-CSRFToken":$.cookie('csrftoken') },
            data: {
                user: $('[name="user"]').val(),
                pwd: $('[name="pwd"]').val()
            },
            success: function (data) {
                data = JSON.parse(data);
                if (data.status) {
                    window.location = data.url
                }
                else {
                    alert('登陆失败')
                }
            }
        })
    })
</script>
```

方式三

使用$.ajaxSetup()给全局的ajax添加默认参数。

可以按照方式一设置data，也可以按照方式二设置请求头。

```
$.ajaxSetup({
    data: {
        csrfmiddlewaretoken: $("[name='csrfmiddlewaretoken']").val(),
    }
});

$.ajaxSetup({
    headers: {"X-CSRFToken": $.cookie('csrftoken')},
});
```

方式四

官方推荐方法（用到jquery.cookie插件）：

```
function csrfSafeMethod(method) {
    // these HTTP methods do not require CSRF protection
    return (/^(GET|HEAD|OPTIONS|TRACE)$/.test(method));
}

$.ajaxSetup({
    beforeSend: function (xhr, settings) {
        if (!csrfSafeMethod(settings.type) && !this.crossDomain) {
            xhr.setRequestHeader("X-CSRFToken", $.cookie('csrftoken'));
        }
    }
});
```

第七章 django 验证码功能

7.1 生成验证码

```
from random import randint, choice
from PIL import Image, ImageDraw, ImageFont
#from cStringIO import StringIO  #python2的用法
from io import StringIO, BytesIO #python3 的用法

from string import printable

def create_captcha():
    font_path = 'static/font/FreeSans.ttf'
    font_color = (randint(150,200), randint(0,150), randint(0,150))
    line_color = (randint(0,150), randint(0,150), randint(150,200))
    point_color = (randint(0, 150), randint(50, 150), randint(150, 200))
    width, height = 100, 40
    image = Image.new('RGB', (width, height), (200, 200, 200))
    font = ImageFont.truetype(font_path, height-10)
    draw = ImageDraw.Draw(image)
    #生成验证码
    #print (printable)
    text =''.join([choice(printable[:62]) for i in range(4)])
    font_width, font_height = font.getsize(text)
    #把验证码写到画布上
    draw.text((10,10), text, font=font, fill=font_color)
    #绘制线条
    for i in range(0, 5):
        draw.line(((randint(0, width), randint(0, height)),
                   (randint(0, width), randint(0, height))),
                  fill=line_color, width=2)
    #绘制点
    for i in range(randint(100, 1000)):
        draw.point((randint(0, width), randint(0, height)), fill=point_color)
    #输出
    out = BytesIO()
    #验证码路径(自己临时设置的)
    code_path = '/home/pi/django-project/femdom/article/static/image/captcha.png'
    image.save(out, format='jpeg')
    image.save(code_path, format='jpeg')
    content = out.getvalue()
    #print(content)
    out.close()
    return text, content
```

7.2 views 返回验证码图片，

并把验证码值传给session， 并设置session 过期时间为60秒

```
def captcha(request):
    text, content = create_captcha()
    request.session['verCode'] = text.lower()
    request.session.set_expiry(60)
    return HttpResponse(content, content_type='image/png')
```

7.3 views 登录后，

将前端输入的验证码和后台session设置的验证码进行比较

```
def dologin(request):
    value=request.GET.get("codevalue")
    value2=request.session.get("verCode")


    if value==value2:

        return HttpResponse("success!!")

    else:
        return HttpResponse("error!!")
```

7.4 django 点击刷新验证码

与 验证码提交

方法一：

```
<div class="col-sm-4 has-success">
                    <img src="/blog/get_captcha/" title="点击更新验证码" id="captcha"
onclick="refresh_captcha(this)">
                </div>
```

```
function refresh_captcha(obj) {
    obj.src = '/blog/get_captcha/?temp='+Math.random()或(new Date()).gettime()
}
```

方法二：

```
<div class="input-group-bottom-right">
        <img  style="width: 100px;" id="Captcha_Img" onclick="Refresh_Captcha()"
                src="//hsajax.app/?t=captcha&amp;r=643131">
        <button id="btn" onclick="POST_Data(1);">提交反馈</button>
</div>

function Refresh_Captcha() {
        // $('#Captcha_Img').attr('src', CFG_Url_Ajax + '?t=captcha&r=' + Get_Random(999999));
        $('#Captcha_Img').attr('src', "{% url 'article:captcha' %}"+'?stamp='+Math.random());
        $('#Captcha').val('');
    }

function POST_Data(Type) {
        if (Type == 1) {
            if ($('#Captcha').val() == '') {
                Show_Prompt_Box(3, '请您输入右侧图片中的验证码.');
            } else if ($('#Content').val() == '') {
                Show_Prompt_Box(3, '请您输入需要反馈的内容,感谢您的支持.');
            } else if ($('#Contact').val() == '') {
                Show_Prompt_Box(3, '请输入您的联系方式,否则我们无法与您沟通.');
            } else {
                $.ajax({
                    url: 'http://192.168.43.249:8000/index/advice/',
                    type: 'post',
                    data: {
                        tel: $('[name="tel"]').val(),
                        content: $('[name="content"]').val(),
                        captcha: $('[name="captcha"]').val()
                    },
                    success: function (data) {
                        data = JSON.parse(data)
                        if (data.status) {
                            Show_Prompt_Box(2, '系统已经收到您的反馈,我们将会尽快查看并处理您的反
馈!');
                            window.location.href = data.url
                        } else {
                            Show_Prompt_Box(3, '您输入的验证码不正确，请从新输入.');
                        }
                    }
                })
            }
        }
    }
```

为什么要用Math.random()，每次生成的数字不一样，可以防止浏览器使用缓存！

第八章 auth认证系统

1 session

```
# 写入 session
rerquest.session['user'] = user

# 读取 session
request.session.get('user')

# 删除 session
request.session.flush()
```

1 写入session

```
rerquest.session['user'] = user
```

2 读取 session

```
request.session.get('user')
```

3 删除 session

```
request.session.flush()
```

2 auth

1 引入 auth

```
from django.contrib import auth
from django.contrib.auth.models import User
```

2 auth 操作

```
# 验证成功， 返回user对象，否则返回None
user = auth.authenticate(username=user_val, password=pwd_val)

# 类似session写操作 request.session['user'] = user
auth.login(request, user)

# 类似session 删除， request.session.flush()
auth.logout(request)

# 校验当前用户是否是登陆状态
request.user.is_authenticated

# 校验密码
old_pwd = '123'
new_pwd = 'qwe'
request.user.check_password(old_pwd)

# 修改密码
request.user.set_password(new_pwd)
```

3 urls

```
from django.urls import path
from . import views

app_name = 'app01'

urlpatterns = [
    path('index/', views.index, name='index'),
    path('login/', views.log_in, name='login'),
    path('logout/', views.log_out, name='logout'),
    path('reg/', views.reg, name='reg'),
]
```

4 forms

引入**forms**

```
from django import forms
from django.forms import widgets
from django.core import validators
from django.core.exceptions import ValidationError
from django.contrib.auth.models import User
```

登陆**form**

```
# 登陆表单
class LoginForm(forms.Form):
    user = forms.CharField(
        error_messages={
            'required': '不能为空'
        },
        widget=widgets.TextInput()
    )

    pwd = forms.CharField(
        error_messages={
            'required':'不能为空'
        },
        widget=widgets.PasswordInput(attrs={'class':'active'})
    )
```

注册**form**

```
# 注册表单
class RegisterForm(forms.Form):
    user = forms.CharField(
        max_length=12,
        min_length=4,
        error_messages={
            'required': '不能为空',
            'min_length':'长度最小为4'
        },
        widget=widgets.TextInput()
    )

    pwd = forms.CharField(
        error_messages={
            'required':'不能为空',
        },
        widget=widgets.PasswordInput()
    )

    pwd2 = forms.CharField(
        error_messages={
            'required': '不能为空',
        },
        widget=widgets.PasswordInput()
    )

    def clean_user(self):
        user_val = self.cleaned_data.get('user')
        exists = User.objects.filter(username=user_val).exists()
        print('exists：', exists)
        if exists:
            raise ValidationError('该账户已经注册！')
        if not user_val.isdigit():
            return user_val
        else:
            raise ValidationError('不能全为数字')

    def clean_pwd(self):
        pwd_val = self.cleaned_data.get('pwd')
        if pwd_val.isdigit():
            raise ValidationError('不能为纯数字')
        else:
            return pwd_val

    def clean(self):
        pwd_val = self.cleaned_data.get('pwd')
        pwd_val2 = self.cleaned_data.get('pwd2')
        print('pwd: ', pwd_val)
        print('pwd2: ', pwd_val2)
        if pwd_val == pwd_val2:
            return self.cleaned_data
        else:
            raise  ValidationError('两次输入的密码不一致')
```

修改密码**fom**

```
# 修改密码表单
class ModifyForm(forms.Form):
    pwd_old = forms.CharField(
        error_messages={
            'required': '不能为空'
        },
        widget = widgets.PasswordInput()
    )
    pwd_new = forms.CharField(
        error_messages={
            'required': '不能为空'
        },
        widget=widgets.PasswordInput()
    )
```

5 views

```
from django.shortcuts import render, redirect, reverse, HttpResponse
from django.contrib import auth
from django.contrib.auth.models import User
from .froms import LoginForm
```

主页**view**

```
def index(request):
    print(request.user)
    print(type(request.user))
    return render(request, 'app01/index.html', locals())
```

登陆**view**

```
def log_in(request):
    form = LoginForm()
    if request.method == 'POST':
        form = LoginForm(request.POST)
        if form.is_valid():
            user_val = form.cleaned_data.get('user')
            pwd_val = form.cleaned_data.get('pwd')
            # 验证成功， 返回user对象，否则返回None
            user = auth.authenticate(username=user_val, password=pwd_val)
            if user:
                # 类似session写操作 session['user'] = user
                auth.login(request, user)
                return redirect(reverse('app01:index'))
        else:
            all_error = form.errors.get('__all__')
            return render(request, 'app01/login.html', locals())
    return render(request, 'app01/login.html', locals())
```

注销**view**

```
def log_out(request):
    auth.logout(request)
    return redirect(reverse('app01:login'))
```

注册**view**

```
def reg(request):
    User.objects.create_user(username='qwe123', password='1234')
    return redirect(reverse('app01:login'))
```

修改密码**view**

```
# 修改密码
def modify(request):
    form = ModifyForm()
    if request.method == 'POST':
        form = ModifyForm(request.POST)
        if form.is_valid():
            pwd_old = form.cleaned_data.get('pwd_old')
            # 密码校验， 返回一个布尔值（True/False）
            res = request.user.check_password(pwd_old)
            print('old_pwd', pwd_old)
            print(res)
            if res:
                pwd_new = form.cleaned_data.get('pwd_new')
                # 设置新密码
                request.user.set_password(pwd_new)
                # 保存到数据库
                request.user.save()
                # 校验， 并写入cookie,session
                user = auth.authenticate(username=request.user, password=pwd_new)
                auth.login(request, user)
                # 校验当前用户是否是登陆状态
                status = request.user.is_authenticated
                print('user status: ', status)
                return redirect(reverse('app01:index'))
            else:
                error = '密码错误'
                return render(request, 'app01/modify.html', locals())
        else:
            all_error = form.errors.get('__all__')
            render(request, 'app01/modify.html', locals())
    return render(request, 'app01/modify.html', locals())
```

6 templates

index.html

```
<p>Index页面</p>
<p> hello, {{ user }}</p>

<hr>
{% if request.user.is_authenticated %}
    <a href="{% url 'app01:logout' %}">注销</a>
    <a href="">修改密码</a>
    <p>helllo, {{ request.user.username }}</p>
{% else %}
    <a href="{% url 'app01:login' %}">登陆</a>
    <a href="{% url 'app01:reg' %}">注册</a>
{% endif %}
```

login.html

```
<style>
    span{
        color: red;
        font-size: small;
    }
</style>

<form action="" method="post" novalidate>
    {% csrf_token %}
    <table>
        <tr>
            <td>账户：</td>
            <td>{{ form.user }}</td>
            <td><span>{{ form.errors.user.0 }}</span></td>
        </tr>
        <tr>
            <td>密码：</td>
            <td>{{ form.pwd }}</td>
            <td><span>{{ from.errors.pwd.0 }}</span></td>
        </tr>
        <tr>
            <td></td>
            <td><input type="submit" value="提交"></td>
            <td><span>{{ all_error.0 }}</span></td>
        </tr>
    </table>
</form>
```

register.html

```
<style>
    span{
        color: red;
        font-size: small;
    }
</style>

<form action="" method="post" novalidate>
    {% csrf_token %}
    <table>
        <tr>
            <td>账户：</td>
            <td>{{ form.user }}</td>
            <td><span>{{ form.errors.user.0 }}</span></td>
        </tr>
        <tr>
            <td>密码：</td>
            <td>{{ form.pwd }}</td>
            <td><span>{{ form.errors.pwd.0 }}</span></td>
        </tr>
        <tr>
            <td>确认密码：</td>
            <td>{{ form.pwd2 }}</td>
            <td><span>{{ form.errors.pwd2.0 }}</span></td>
        </tr>
        <tr>
            <td></td>
            <td><input type="submit" value="提交"></td>
            <td><span>{{ all_error.0 }}</span></td>
        </tr>
    </table>
</form>
```

modify.html

```
<p>hello, {{ user }} </p>
<style>
    span {
        color: red;
        font-size: small;
    }
</style>
{% if request.user.is_authenticated %}
    <form action="" method="post" novalidate>
    {% csrf_token %}
    <table>
        <tr>
            <td>旧密码：</td>
            <td>{{ form.pwd_old }}</td>
            <td><span>{{ form.errors.pwd_old.0 }} {{ error }}</span></td>
        </tr>
        <tr>
            <td>新密码：</td>
            <td>{{ form.pwd_new }}</td>
            <td><span>{{ form.errors.pwd_new.0 }}</span></td>
        </tr>
        <tr>
            <td></td>
            <td><input type="submit" value="提交"></td>
            <td><span>{{ all_error.0 }}</span></td>
        </tr>
    </table>
</form>
{% else %}
{% endif %}
    <a href="{% url 'app01:login' %}">登陆</a>
    <a href="{% url 'app01:reg' %}">注册</a>
```

第九章 Django常见问题

Django 数据导入和导出（数据库的迁移方法）

9.1 数据导出

1 数据导出* python manage.py dumpdata*

不指定 appname 时默认为导出所有的app

python manage.py dumpdata [appname] > appname_data.json指定appnamde 导出 指定app 的数据（比如appname为cmdb）python manage.py dumpdata cmdb>cmdb.json2. 数据导入python manage.py loaddata

不需要指定 appnamepython manage.py loaddata blog_dump.json优点：可以兼容各种支持的数据库，也就是说，以前用的是 SQLite3，可以导出后，用这种方法导入到 MySQL, PostgreSQL 等数据库，反过来也可以。

缺点：数据量大的时候，速度相对较慢，表的关系比较复杂的时候可能导入不成功。

个人推荐导入数据做法：

1 将APP的migrations目录下，只保留**init**.py文件，其余文件全部清空；
重置文件
python manage.py migrate --fake cmdb zero # cmdb是app的名称
删除migrations的处init.py的其他文件

2 然后分别执行：python manage.py makemigrations 和 python3 manage.py migrate；

9.2 数据导入

3 最后导入数据：python manage.py loaddata blog_dump.json

以上做法，能够增加数据导入的成功率。

9.3 解决djang DEBUG = False和True

没有css和js

DEBUG = False的情况
1、在settings.py添加如下

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
2、运行下面命令把相关文件copy到static这个目录

python manage.py collectstatic

3、在项目下的总urls.py（不是app的urls.py）中，urlpatterns下面添加：

from django.contrib import admin
from django.urls import path,include,re_path
from . import views
from django.conf import settings
from django.conf.urls.static import serve,static

urlpatterns = [

```
path('admin/', admin.site.urls),
re_path(r'^static/(?P<path>.*)$', serve, {'document_root': settings.STATIC_ROOT}),
]
```

DEBUG = True的情况
配置

STATIC_URL = '/static/'
STATICFILES_DIRS = [

```
os.path.join(BASE_DIR, "static"),
```

]

第十章 虚拟环境配置

10.1 安装virtualenv

virtualenv 是用来创建虚拟环境的软件， 我们可以通过pip 或者pip3来安装

```
pip install virtualenv

pip3 install virtualenv
```

10.2 创建虚拟环境

创建虚拟环境非常简单，通过以下命令

```
virtualenv [虚拟环境的名字]
```

10.3 进入虚拟环境

虚拟环境创建好后， 那么可以进入这个虚拟环境中， 然后安装第三方包，

进入虚拟环境一般分两种， 第一种windows, 第二种linux

1. windows 进入虚拟环境：进入虚拟环境的scripts文件夹中
2. linux进入虚拟环境：source/path/to/virtualenv/bin/activate 一旦你进入这个虚拟环境中，
   你安装包， 卸载包都是在这个虚拟环境中， 不会影响到外部环境

10.4 退出虚拟环境

通过命令可以完成：deactivate

10.5 指定python解释器

在电脑的环境变量中，一般是不会去更改一些环境变量的顺序的， 也就是说比如

你的python2/Scripts在python3/Scripts的前面， 那么你不会经常更改他们玩的位置， 但是、

这时候我确实想创建虚拟环境的时候用python3这个版本， 这时候可以通过-p参数来指定

具体的python解释器

```
virtualenv -p c:\Python36\python.exe
```

10.6 virtualenvwrapper

virtualenvwrapper 这个软件包让我们管理虚拟环境变得更加简单， 不再跑到某个目录下

通过virtualenv 来创建虚拟环境， 并且激活的时候也要跑到具体的目录下去激活

1 安装 virtualenvwrapper:

1. *nix: pip install virtualenvwrapper
2. windows: pip install virtualenvwrapper-win

2 virtualenvwrapper 基本使用：

1 创建虚拟环境：

```
mkvirtualenv my_env
```

2 切换到某个虚拟环境

```
workon my_env
```

3 退出当前虚拟环境

```
deactivate
```

4 删除某个虚拟环境

```
rmvirtualenv my_env
```

5 列出所有虚拟环境

```
lsvirtualenv
```

6 进入虚拟环境所在的目录

```
cdvirtualenv
```

7 修改 **mkvirtualenv** 的默认路径

在我的电脑-右键-属性-高级系统设置-环境变量-系统变量 中添加一个参数WORKON_HOME，将这个

参数的值设置为你需要的路径

8 创建虚拟环境的时候指定**Python**版本

在使用 mkvirtuakenv 的时候， 可以指定 --python 的参数来指定具体的python路径

```
mkvirtualenv --python==c:Python36\python.exe my_env
```

10.7 学前准备

1. 安装python3.6
2. 安装 virtualenvwrapper, 这个是用来创建虚拟环境的包， 使用虚拟环境可以让我们的包管理更加方便
3. 虚拟环境相关操作：

   创建虚拟环境： mkvirtualenv --python=['python3.6文件所在路径'][虚拟环境名]

   进入虚拟环境：workon [虚拟环境名]

   退出虚拟环境：deactivate
4. 首先进入虚拟环境 workon django-env, 然后通过 pip install django==2.0

5. 安装pycharm profession 2017 (专业版)， community(社区版)不能用于网页开发
6. 安装最新 MySQL, windows版的MySQL下载地址：https://dev.mysql.com/downloads/windows/installer/5.7.html, 如果是其他操作系统， 来这个页面选择具体的MySQL进行下载：http://dev.mysql.com/downloads/mysql/
7. 安装 pymysl, 这个库是python 来操作数据库的， 没有它， django就不能操作数据库，pip install pymysql

10.8 virtualenv 设置

下载**python**的虚拟环境安装包

[budong@budong ~]$ sudo pip install virtualenv

创建虚拟环境

[budong@budong ~]$ virtualenv -p /usr/bin/python3 py3env
或者 virtualenv --no-site-packages --python=python3 env[虚拟环境名称]

进入虚拟环境

[budong@budong ~]$ source py3env/bin/activate
(py3env) [budong@budong ~]$ python

退出虚拟环境

(py3env) [budong@budong ~]$ deactivate

10.9 虚拟环境开机启动

把 source py3env/bin/activate 添加到 .bashrc 中，可以一登陆就进入啦python3的虚拟环境

如果要添加python2的虚拟环境只需把上面的python3改成python2即可，或者如下

[budong@budong ~]$ virtualenv env_py2

因为系统默认是python2，所以可以不用添加命令的路径

第十一章 中间件

django 翻页功能

django系统自带的翻页器 Paginator

在views.py 中引入

```
from django.core.paginator import Paginator, PageNotAnInteger, EmptyPage
```

1. 编写一个翻页函数，封装Paginator

   传入request对象， 以及要分页的数据对象列表 post_list

```
# 分页函数
def paginator(request, post_list):
    # p 为分页对象
    p = Paginator(post_list, 20)
    p.count  # 数据总数
    p.num_pages  # 总页数
    p.page_range  # [1,2,3，.] 得到页码，动态生成
    # 以get 方法从url中获取当前页码数
    page_num = request.GET.get('page')
    # 得到第几页
    try:
        # 获取指定页码数的数据
        videos = p.page(page_num)
    except PageNotAnInteger:
        # 如果输入页码错误，就显示第一页
        videos = p.page(1)
    except EmptyPage:
        # 如果超出页码范围就把最后一页显示出来
        videos = p.page(p.num_pages)
    return p, videos
```

1. 调用：

```
# 小说主页
def novel(request):
    novel_list_all = Novel.objects.order_by('-date')
    p, novels = paginator(request, novel_list_all)
    return render(request, 'article/novel_index.html', locals())
```

中间件 的概念

是介于request, 与 response处理之间的一道处理过程， 相对比较轻量级， 并且在全局上改变

django的输入与输出， 因为改变的是全局， 所以徐亚谨慎使用， 不能==用不好会影响到性能

Django的中间件的定义

```
Middleware is a framework of hooks into Django's request ''/'' reponse processing.
It's a light, low lwve plugin system for globally altering Django's input or output
```

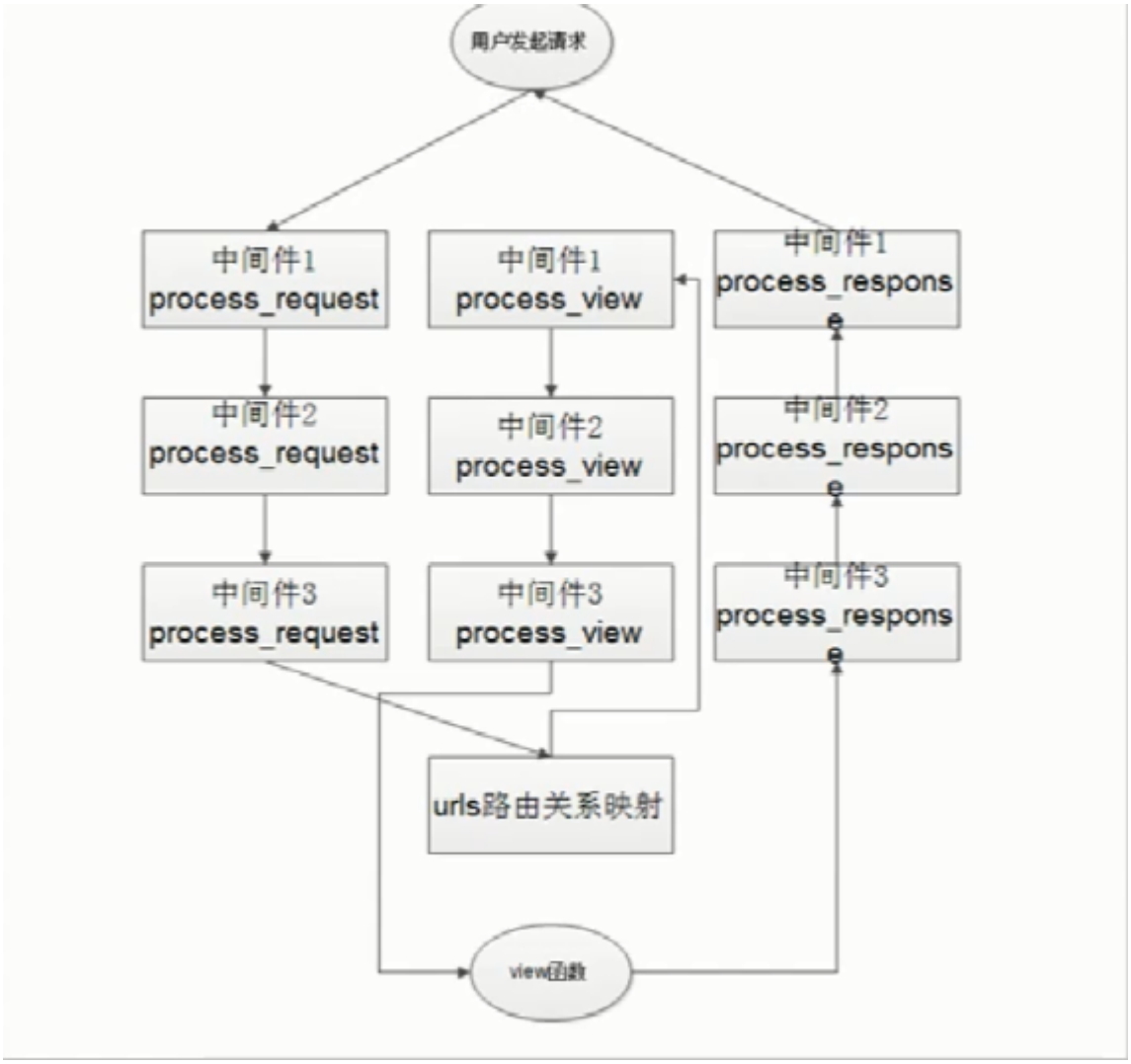
如果你想修改请求， 例如被传送到view中的HttpRequesr对象， 或者你想改变view返回的HttpResponse对象

这些都可以通过中间件来实现

可能你还想在view执行之前做一些操作，这种情况可以用middleware来实现

自定义中间件

流程图如下：



首先在settings.py 里添加自定义的中间件Md1, Md2

系统默认的有七个中间， 会按顺序执行

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'my_middleware.Md1',
    'my_middleware.Md2',
]
```

注意：中间件事逐一 的，按顺序执行的， 如果自定义中间件之间有相互调用的关系或者与系统自带的中间件有调用关系， 那么不要随意更改上面中间件的位置，

要不容易报错

中间件一共有四个方法：

```
process_request

process_view

process_exception

process_response
```

process_request

```
def process_request(self, request):
    print('Md1 request')
    # return HttpResponse('Md1中断')
```

process_response

```
def process_response(self, request, response):
    print('Md1 response')
    return response
```

结果如下：

```
Md1 request
Md2 request
Md1 process view
Md2 process view
views index...
Md2 response
Md1 response
```

process_view

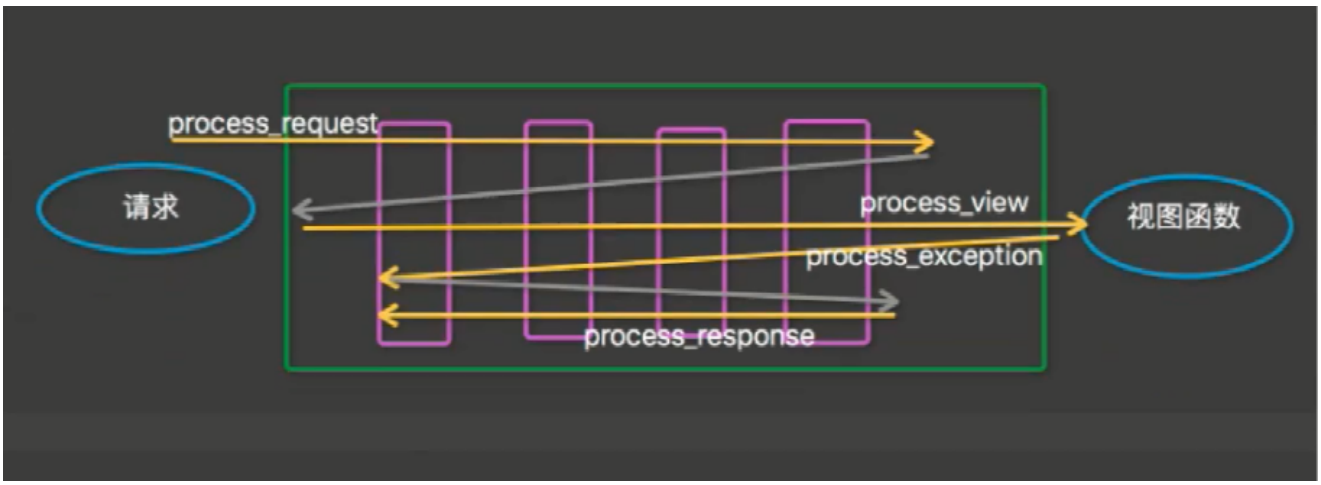
```
    def process_view(self, request, callback, callback_args, callback_kwargs):
        print('Md1 process view', callback, callback_args, callback_kwargs)
        # ret = callback(request)
        # return ret
```

注意：process_view 如果有返回值， 会越过其他的process_view 以及视图函数， 但是所有的process_response 都会执行

process_exception

```
    def process_exception(self, request, exception):
        print('md1 process_exception...')
        return HttpResponse(exception)
```

当views 出现错误时候



将Md1的process_exception 修改如下

```
    def process_exception(self, request, exception):
        print('md1 process_exception...')
        return HttpResponse(exception)
```

结果如下：

```
Md1 request
Md2 request
Md1 process view <function index at 0xb54d0030> () {}
Md2 process view <function index at 0xb54d0030>
views index...
md2 process_exception...
md1 process_exception...
Md2 response
Md1 response
```

附录

HTTP状态码

状态码	含义
100	客户端应当继续发送请求。这个临时响应是用来通知客户端它的部分请求已经被服务器接收，且仍未被拒绝。客户端应当继续发送请求的剩余部分，或者如果请求已经完成，忽略这个响应。服务器必须在请求完成后向客户端发送一个最终响应。
101	服务器已经理解了客户端的请求，并将通过Upgrade 消息头通知客户端采用不同的协议来完成这个请求。在发送完这个响应最后的空行后，服务器将会切换到在Upgrade 消息头中定义的那些协议。 只有在切换新的协议更有好处的时候才应该采取类似措施。例如，切换到新的HTTP 版本比旧版本更有优势，或者切换到一个实时且同步的协议以传送利用此类特性的资源。
102	由WebDAV（RFC 2518）扩展的状态码，代表处理将被继续执行。
200	请求已成功，请求所希望的响应头或数据体将随此响应返回。
201	请求已经被实现，而且有一个新的资源已经依据请求的需要而建立，且其 URI 已经随Location 头信息返回。假如需要的资源无法及时建立的话，应当返回 '202 Accepted'。
202	服务器已接受请求，但尚未处理。正如它可能被拒绝一样，最终该请求可能会也可能不会被执行。在异步操作的场合下，没有比发送这个状态码更方便的做法了。　　返回202状态码的响应的目的是允许服务器接受其他过程的请求（例如某个每天只执行一次的基于批处理的操作），而不必让客户端一直保持与服务器的连接直到批处理操作全部完成。在接受请求处理并返回202状态码的响应应当在返回的实体中包含一些指示处理当前状态的信息，以及指向处理状态监视器或状态预测的指针，以便用户能够估计操作是否已经完成。
203	服务器已成功处理了请求，但返回的实体头部元信息不是在原始服务器上有效的确定集合，而是来自本地或者第三方的拷贝。当前的信息可能是原始版本的子集或者超集。例如，包含资源的元数据可能导致原始服务器知道元信息的超级。使用此状态码不是必须的，而且只有在响应不使用此状态码便会返回200 OK的情况下才是合适的。
204	服务器成功处理了请求，但不需要返回任何实体内容，并且希望返回更新了的元信息。响应可能通过实体头部的形式，返回新的或更新后的元信息。如果存在这些头部信息，则应当与所请求的变量相呼应。　　如果客户端是浏览器的话，那么用户浏览器应保留发送了该请求的页面，而不产生任何文档视图上的变化，即使按照规范新的或更新后的元信息应当被应用到用户浏览器活动视图中的文档。 　　由于204响应被禁止包含任何消息体，因此它始终以消息头后的第一个空行结尾。
205	服务器成功处理了请求，且没有返回任何内容。但是与204响应不同，返回此状态码的响应要求请求者重置文档视图。该响应主要是被用于接受用户输入后，立即重置表单，以便用户能够轻松地开始另一次输入。　　与204响应一样，该响应也被禁止包含任何消息体，且以消息头后的第一个空行结束。

状态码	含义
206	服务器已经成功处理了部分 GET 请求。类似于 FlashGet 或者迅雷这类的 HTTP 下载工具都是使用此类响应实现断点续传或者将一个大文档分解为多个下载段同时下载。 该请求必须包含 Range 头信息来指示客户端希望得到的内容范围，并且可能包含 If-Range 来作为请求条件。 响应必须包含如下的头部域： Content-Range 用以指示本次响应中返回的内容的范围；如果是 Content-Type 为 multipart/byteranges 的多段下载，则每一 multipart 段中都应包含 Content-Range 域用以指示本段的内容范围。假如响应中包含 Content-Length，那么它的数值必须匹配它返回的内容范围的真实字节数。 Date ETag 和/或 Content-Location，假如同样的请求本应该返回200响应。 Expires, Cache-Control，和/或 Vary，假如其值可能与之前相同变量的其他响应对应的值不同的话。 假如本响应请求使用了 If-Range 强缓存验证，那么本次响应不应该包含其他实体头；假如本响应的请求使用了 If-Range 弱缓存验证，那么本次响应禁止包含其他实体头；这避免了缓存的实体内容和更新了的实体头信息之间的不一致。否则，本响应就应当包含所有本应该返回200响应中应当返回的所有实体头部域。 假如 ETag 或 Last-Modified 头部不能精确匹配的话，则客户端缓存应禁止将206响应返回的内容与之前任何缓存过的内容组合在一起。 任何不支持 Range 以及 Content-Range 头的缓存都禁止缓存206响应返回的内容。
207	由WebDAV(RFC 2518)扩展的状态码，代表之后的消息体将是一个XML消息，并且可能依照之前子请求数量的不同，包含一系列独立的响应代码。
300	被请求的资源有一系列可供选择的回馈信息，每个都有自己特定的地址和浏览器驱动的商议信息。用户或浏览器能够自行选择一个首选的地址进行重定向。 除非这是一个 HEAD 请求，否则该响应应当包括一个资源特性及地址的列表的实体，以便用户或浏览器从中选择最合适的重定向地址。这个实体的格式由 Content-Type 定义的格式所决定。浏览器可能根据响应的格式以及浏览器自身能力，自动作出最合适的选择。当然，RFC 2616规范并没有规定这样的自动选择该如何进行。 如果服务器本身已经有了首选的回馈选择，那么在 Location 中应当指明这个回馈的 URI；浏览器可能会将这个 Location 值作为自动重定向的地址。此外，除非额外指定，否则这个响应也是可缓存的。
301	被请求的资源已永久移动到新位置，并且将来任何对此资源的引用都应该使用本响应返回的若干个 URI 之一。如果可能，拥有链接编辑功能的客户端应当自动把请求的地址修改为从服务器反馈回来的地址。除非额外指定，否则这个响应也是可缓存的。 新的永久性的 URI 应当在响应的 Location 域中返回。除非这是一个 HEAD 请求，否则响应的实体中应当包含指向新的 URI 的超链接及简短说明。 如果这不是一个 GET 或者 HEAD 请求，因此浏览器禁止自动进行重定向，除非得到用户的确认，因为请求的条件可能因此发生变化。 注意：对于某些使用 HTTP/1.0 协议的浏览器，当它们发送的 POST 请求得到了一个301响应的话，接下来的重定向请求将会变成 GET 方式。
302	请求的资源现在临时从不同的 URI 响应请求。由于这样的重定向是临时的，客户端应当继续向原有地址发送以后的请求。只有在Cache-Control或Expires中进行了指定的情况下，这个响应才是可缓存的。 新的临时性的 URI 应当在响应的 Location 域中返回。除非这是一个 HEAD 请求，否则响应的实体中应当包含指向新的 URI 的超链接及简短说明。 如果这不是一个 GET 或者 HEAD 请求，那么浏览器禁止自动进行重定向，除非得到用户的确认，因为请求的条件可能因此发生变化。 注意：虽然RFC 1945和RFC 2068规范不允许客户端在重定向时改变请求的方法，但是很多现存的浏览器将302响应视作为303响应，并且使用 GET 方式访问在 Location 中规定的 URI，而无视原先请求的方法。状态码303和307被添加了进来，用以明确服务器期待客户端进行何种反应。

状态码	含义
303	对应当前请求的响应可以在另一个 URI 上被找到，而且客户端应当采用 GET 的方式访问那个资源。这个方法的存在主要是为了允许由脚本激活的POST请求输出重定向到一个新的资源。这个新的 URI 不是原始资源的替代引用。同时，303响应禁止被缓存。当然，第二个请求（重定向）可能被缓存。 新的 URI 应当在响应的 Location 域中返回。除非这是一个 HEAD 请求，否则响应的实体中应当包含指向新的 URI 的超链接及简短说明。 注意：许多 HTTP/1.1 版以前的 浏览器不能正确理解303状态。如果需要考虑与这些浏览器之间的互动，302状态码应该可以胜任，因为大多数的浏览器处理302响应时的方式恰恰就是上述规范要求客户端处理303响应时应当做的。
304	如果客户端发送了一个带条件的 GET 请求且该请求已被允许，而文档的内容（自上次访问以来或者根据请求的条件）并没有改变，则服务器应当返回这个状态码。304响应禁止包含消息体，因此始终以消息头后的第一个空行结尾。 该响应必须包含以下的头信息： Date，除非这个服务器没有时钟。假如没有时钟的服务器也遵守这些规则，那么代理服务器以及客户端可以自行将 Date 字段添加到接收到的响应头中去（正如RFC 2068中规定的一样），缓存机制将会正常工作。 ETag 和/或 Content-Location，假如同样的请求本应返回200响应。 Expires, Cache-Control，和/或Vary，假如其值可能与之前相同变量的其他响应对应的值不同的话。 假如本响应请求使用了强缓存验证，那么本次响应不应该包含其他实体头；否则（例如，某个带条件的 GET 请求使用了弱缓存验证），本次响应禁止包含其他实体头；这避免了缓存了的实体内容和更新了的实体头信息之间的不一致。 假如某个304响应指明了当前某个实体没有缓存，那么缓存系统必须忽视这个响应，并且重复发送不包含限制条件的请求。 假如接收到一个要求更新某个缓存条目的304响应，那么缓存系统必须更新整个条目以反映所有在响应中被更新的字段的值。
305	被请求的资源必须通过指定的代理才能被访问。Location 域中将给出指定的代理所在的 URI 信息，接收者需要重复发送一个单独的请求，通过这个代理才能访问相应资源。只有原始服务器才能建立305响应。 注意：RFC 2068中没有明确305响应是为了重定向一个单独的请求，而且只能被原始服务器建立。忽视这些限制可能导致严重的安全后果。
306	在最新版的规范中，306状态码已经不再被使用。
307	请求的资源现在临时从不同的URI 响应请求。由于这样的重定向是临时的，客户端应当继续向原有地址发送以后的请求。只有在Cache-Control或Expires中进行了指定的情况下，这个响应才是可缓存的。 新的临时性的URI 应当在响应的 Location 域中返回。除非这是一个HEAD 请求，否则响应的实体中应当包含指向新的URI 的超链接及简短说明。因为部分浏览器不能识别307响应，因此需要添加上述必要信息以便用户能够理解并向新的 URI 发出访问请求。 如果这不是一个GET 或者 HEAD 请求，那么浏览器禁止自动进行重定向，除非得到用户的确认，因为请求的条件可能因此发生变化。
400	1、语义有误，当前请求无法被服务器理解。除非进行修改，否则客户端不应该重复提交这个请求。 2、请求参数有误。
401	当前请求需要用户验证。该响应必须包含一个适用于被请求资源的 WWW-Authenticate 信息头用以询问用户信息。客户端可以重复提交一个包含恰当的 Authorization 头信息的请求。如果当前请求已经包含了 Authorization 证书，那么401响应代表着服务器验证已经拒绝了那些证书。如果401响应包含了与前一个响应相同的身份验证询问，且浏览器已经至少尝试了一次验证，那么浏览器应当向用户展示响应中包含的实体信息，因为这个实体信息中可能包含了相关诊断信息。参见RFC 2617。
402	该状态码是为了将来可能的需求而预留的。

状态码	含义
403	服务器已经理解请求，但是拒绝执行它。与401响应不同的是，身份验证并不能提供任何帮助，而且这个请求也不应该被重复提交。如果这不是一个 HEAD 请求，而且服务器希望能够讲清楚为何请求不能被执行，那么就应该在实体内描述拒绝的原因。当然服务器也可以返回一个404响应，假如它不希望让客户端获得任何信息。
404	请求失败，请求所希望得到的资源未被在服务器上发现。没有信息能够告诉用户这个状况到底是暂时的还是永久的。假如服务器知道情况的话，应当使用410状态码来告知旧资源因为某些内部的配置机制问题，已经永久的不可用，而且没有任何可以跳转的地址。404这个状态码被广泛应用于当服务器不想揭示到底为何请求被拒绝或者没有其他适合的响应可用的情况下。
405	请求行中指定的请求方法不能被用于请求相应的资源。该响应必须返回一个Allow 头信息用以表示出当前资源能够接受的请求方法的列表。 鉴于 PUT，DELETE 方法会对服务器上的资源进行写操作，因而绝大部分的网页服务器都不支持或者在默认配置下不允许上述请求方法，对于此类请求均会返回405错误。
406	请求的资源的内容特性无法满足请求头中的条件，因而无法生成响应实体。 除非这是一个 HEAD 请求，否则该响应就应当返回一个包含可以让用户或者浏览器从中选择最合适的实体特性以及地址列表的实体。实体的格式由 Content-Type 头中定义的媒体类型决定。浏览器可以根据格式及自身能力自行作出最佳选择。但是，规范中并没有定义任何作出此类自动选择的标准。
407	与401响应类似，只不过客户端必须在代理服务器上进行身份验证。代理服务器必须返回一个 Proxy-Authenticate 用以进行身份询问。客户端可以返回一个 Proxy-Authorization 信息头用以验证。参见 RFC 2617。
408	请求超时。客户端没有在服务器预备等待的时间内完成一个请求的发送。客户端可以随时再次提交这一请求而无需进行任何更改。
409	由于和被请求的资源的当前状态之间存在冲突，请求无法完成。这个代码只允许用在这样的情况下才能被使用：用户被认为能够解决冲突，并且会重新提交新的请求。该响应应当包含足够的信息以便用户发现冲突的源头。 冲突通常发生于对 PUT 请求的处理中。例如，在采用版本检查的环境下，某次 PUT 提交的对特定资源的修改请求所附带的版本信息与之前的某个（第三方）请求向冲突，那么此时服务器就应该返回一个409错误，告知用户请求无法完成。此时，响应实体中很可能会包含两个冲突版本之间的差异比较，以便用户重新提交归并以后的新版本。
410	被请求的资源在服务器上已经不再可用，而且没有任何已知的转发地址。这样的状况应当被认为是永久性的。如果可能，拥有链接编辑功能的客户端应当在获得用户许可后删除所有指向这个地址的引用。如果服务器不知道或者无法确定这个状况是否是永久的，那么就应该使用404状态码。除非额外说明，否则这个响应是可缓存的。 410响应的目的主要是帮助网站管理员维护网站，通知用户该资源已经不再可用，并且服务器拥有者希望所有指向这个资源的远端连接也被删除。这类事件在限时、增值服务中很普遍。同样，410响应也被用于通知客户端在当前服务器站点上，原本属于某个个人的资源已经不再可用。当然，是否需要把所有永久不可用的资源标记为'410 Gone'，以及是否需要保持此标记多长时间，完全取决于服务器拥有者。
411	服务器拒绝在没有定义 Content-Length 头的情况下接受请求。在添加了表明请求消息体长度的有效 Content-Length 头之后，客户端可以再次提交该请求。

状态码	含义
412	服务器在验证在请求的头字段中给出先决条件时，没能满足其中的一个或多个。这个状态码允许客户端在获取资源时在请求的元信息（请求头字段数据）中设置先决条件，以此避免该请求方法被应用到其希望的内容以外的资源上。
413	服务器拒绝处理当前请求，因为该请求提交的实体数据大小超过了服务器愿意或者能够处理的范围。此种情况下，服务器可以关闭连接以免客户端继续发送此请求。 如果这个状况是临时的，服务器应当返回一个 Retry-After 的响应头，以告知客户端可以在多少时间以后重新尝试。
414	请求的URI 长度超过了服务器能够解释的长度，因此服务器拒绝对该请求提供服务。这比较少见，通常的情况包括： 本应使用POST方法的表单提交变成了GET方法，导致查询字符串（Query String）过长。 重定向URI “黑洞”，例如每次重定向把旧的 URI 作为新的 URI 的一部分，导致在若干次重定向后 URI 超长。 客户端正在尝试利用某些服务器中存在的安全漏洞攻击服务器。这类服务器使用固定长度的缓冲读取或操作请求的 URI，当 GET 后的参数超过某个数值后，可能会产生缓冲区溢出，导致任意代码被执行[1]。没有此类漏洞的服务器，应当返回414状态码。
415	对于当前请求的方法和所请求的资源，请求中提交的实体并不是服务器中所支持的格式，因此请求被拒绝。
416	如果请求中包含了 Range 请求头，并且 Range 中指定的任何数据范围都与当前资源的可用范围不重合，同时请求中又没有定义 If-Range 请求头，那么服务器就应当返回416状态码。 假如 Range 使用的是字节范围，那么这种情况就是指请求指定的所有数据范围的首字节位置都超过了当前资源的长度。服务器也应当在返回416状态码的同时，包含一个 Content-Range 实体头，用以指明当前资源的长度。这个响应也被禁止使用 multipart/byteranges 作为其 Content-Type。
417	在请求头 Expect 中指定的预期内容无法被服务器满足，或者这个服务器是一个代理服务器，它有明显的证据证明在当前路由的下一个节点上，Expect 的内容无法被满足。
421	从当前客户端所在的IP地址到服务器的连接数超过了服务器许可的最大范围。通常，这里的IP地址指的是从服务器上看到的客户端地址（比如用户的网关或者代理服务器地址）。在这种情况下，连接数的计算可能涉及到不止一个终端用户。
422	从当前客户端所在的IP地址到服务器的连接数超过了服务器许可的最大范围。通常，这里的IP地址指的是从服务器上看到的客户端地址（比如用户的网关或者代理服务器地址）。在这种情况下，连接数的计算可能涉及到不止一个终端用户。
422	请求格式正确，但是由于含有语义错误，无法响应。（RFC 4918 WebDAV）423 Locked 当前资源被锁定。（RFC 4918 WebDAV）
424	由于之前的某个请求发生的错误，导致当前请求失败，例如 PROPPATCH。（RFC 4918 WebDAV）
425	在WebDav Advanced Collections 草案中定义，但是未出现在《WebDAV 顺序集协议》（RFC 3658）中。
426	客户端应当切换到TLS/1.0。（RFC 2817）
449	由微软扩展，代表请求应当在执行完适当的操作后进行重试。

状态码	含义
500	服务器遇到了一个未曾预料的状况，导致了它无法完成对请求的处理。一般来说，这个问题都会在服务器的程序码出错时出现。
501	服务器不支持当前请求所需要的某个功能。当服务器无法识别请求的方法，并且无法支持其对任何资源的请求。
502	作为网关或者代理工作的服务器尝试执行请求时，从上游服务器接收到无效的响应。
503	由于临时的服务器维护或者过载，服务器当前无法处理请求。这个状况是临时的，并且将在一段时间以后恢复。如果能够预计延迟时间，那么响应中可以包含一个 Retry-After 头用以标明这个延迟时间。如果没有给出这个 Retry-After 信息，那么客户端应当以处理500响应的方式处理它。　　注意：503状态码的存在并不意味着服务器在过载的时候必须使用它。某些服务器只不过是希望拒绝客户端的连接。
504	作为网关或者代理工作的服务器尝试执行请求时，未能及时从上游服务器（URI标识出的服务器，例如HTTP、FTP、LDAP）或者辅助服务器（例如DNS）收到响应。　　注意：某些代理服务器在DNS查询超时时会返回400或者500错误
505	服务器不支持，或者拒绝支持在请求中使用的 HTTP 版本。这暗示着服务器不能或不愿使用与客户端相同的版本。响应中应当包含一个描述了为何版本不被支持以及服务器支持哪些协议的实体。
506	由《透明内容协商协议》（RFC 2295）扩展，代表服务器存在内部配置错误：被请求的协商变元资源被配置为在透明内容协商中使用自己，因此在一个协商处理中不是一个合适的重点。
507	服务器无法存储完成请求所必须的内容。这个状况被认为是临时的。WebDAV (RFC 4918)
509	服务器达到带宽限制。这不是一个官方的状态码，但是仍被广泛使用。
510	获取资源所需要的策略并没有没满足。（RFC 2774）