Django 项目开发纪要 V1.0

(django 项目相关问题集)

Du Liang 250858386@qq.com

目录

本文档为两个 Django 项目开发过程中的错误整理，给出实际项目中错误和错误的解决方法。

该文档中涉及有 BootStrap 前端、SQL-Serrver2008 数据库、MySQL 数据库等。

url 跳转而页面不跳转问题处理：

1、url文件配置出错，注意url匹配时是与第一个相匹配的URL匹配。在本次开发过程中出现如下案例：

```
url(r'^Process_Review_UnEditing',Process_Depart_views.Process_Review_UnEditing,name='process_review_unedit'),
#工艺开发室未审核订单
url(r'^Process_Review_UnEditingItemDetails',Process_Depart_views.Process_Review_UnEditingItemDetails,name='Process_ReviewUneditItemDetails_url'),
#工艺开发室未审核订单详情
```

点击进入未审核订单详情页时，此时 URL 匹配会先与 Process_Review_UnEditing 匹配，而跳过第二个 URL，导致查看未审核订单详情页面时 URL 跳转而页面不跳转情况。

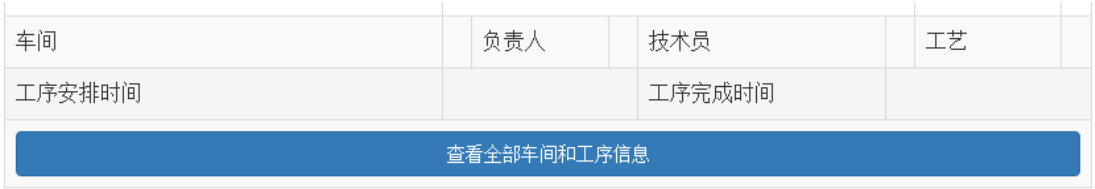
修改方法为：修改 URL，使其不具有相同字符

```
url(r'^Process_Review_UnEditing',Process_Depart_views.Process_Review_UnEditing,name='process_review_unedit'),
#工艺开发室未审核订单
url(r'^Process_Review_UnEditItemDetails',Process_Depart_views.Process_Review_UnEditingItemDetails,name='Process_ReviewUneditItemDetails_url')
```

---

button 填充整个 table 的 td 框

效果演示如下：



代码片段为：

```
<tr>
        <td colspan="8">
        <button class="btn btn-primary text-center" type="button" data-toggle="collapse" data-target="#collapseExample" aria-expanded="false" aria-controls="collapseExample" style="width:100%">
  查看全部车间和工序信息
</button></td>
    </tr>
```

处理页面请求时报 MultiValueDictKeyError 错误

引发这个问题是由于在 view.py 当中，获取前台传来的参数，但是前台并没有传递这个参数导致的。例如：
前台发送请求 getBooks
后台代码中:

```
node = request.GET['node']
```

获取 node 参数，但是前台没有传递过来，就会导致 MultiValueDictKeyError 错误.
解决办法：
1.通常的办法
首先判断一下在 request 当中有没有这个参数，然后再去取
例如：

```
if 'is_private' in request.POST:
    is_private = request.POST['is_private']
else:
    is_private = False
```

2.给出一个默认值

```
is_private = request.POST.get('is_private', False);
```

3.处理异常

```
try:
    is_private = request.POST['is_private']
except MultiValueDictKeyError:
    is_private = False
```

在本次项目开发过程中出现这一错误原因是在处理表单中<Select>属性（下拉菜单）时，表单提交错误导致的

```
<td><select class="form-control" name="Production_ComponentNo"
id="productionComponentNo">
                        {% for item in orderComponent %}
                        <option value="{{ item.0 }}">{{ item.0}}</option>
                        {% endfor %}
                        </select></td>
```

对于处理的表单，获取其传递参数时，POST 方法应该设置为：

```
productionComponentNo=request.POST['Production_ComponentNo']#获取零件编号
```

Django 中处理上传文件：

1）

```
def handle_uploaded_file(f,name):
    '''处理上传文件'''
    file_name = "uploads"+name
    try:
        destination = open(file_name,'wb+')
        for chunk in f.chunks():
```

```
                destination.write(chunk)
            destination.close()
        except Exception as e:
            return e
        return file_name
```

2）官方文档建议

上传文件表单可建立为：

```
from django import forms

class UploadFileForm(forms.Form):
    title = forms.CharField(max_length=50)
    file = forms.FileField()
```

视图函数为

```
from django.http import HttpResponseRedirect
from django.shortcuts import render_to_response
from .forms import UploadFileForm

# Imaginary function to handle an uploaded file.
from somewhere import handle_uploaded_file

def upload_file(request):
    if request.method == 'POST':
        form = UploadFileForm(request.POST, request.FILES)
        if form.is_valid():
            handle_uploaded_file(request.FILES['file'])
            return HttpResponseRedirect('/success/url/')
    else:
        form = UploadFileForm()
    return render_to_response('upload.html', {'form': form})
```

处理文件方法：

```
def handle_uploaded_file(f):
    with open('some/file/name.txt', 'wb+') as destination:
        for chunk in f.chunks():
            destination.write(chunk)
```

其他处理包括使用模型处理可见：https://docs.djangoproject.com/en/1.8/topics/http/file-uploads/

表单验证问题

一般采用 JS 验证，其中有 Jquery 等框架可以采用，主要思想和步骤一般是先获取表单属性（JS 获取输入值）随后采用正则规则进行输入值的验证。

- 不采用表单提交方式---这种情况下直接采用 js 获取<input>属性中输入值进行验证
  参见本项目中的一个例子

```
var StartTime = $("#StartTime").val();//获取工序所安排的开始时间
```

```
var ResponsePerson = $("#ResponsePerson").val();//获取负责人信息
var TechPerson = $("#TechPerson").val();//获取技术员信息
var PreparedInfo = $("#PreparedInfo").val();//获取工装信息
var Workshop = $("#Workshop").val();//获取车间信息
var Procedure = $("#Procedure").val();//获取工序信息
var EndTime = $("#EndTime").val();//获取结束时间
var workshopReviewCommoment = $("#workshopReviewCommoment").val();
/*
需要注意，当前的表单验证是直接验证的方式，由于没有采用表单验证方式导致JS/Jquery验证框架不能使用。
重构时需注意修改和完善，当前缺陷--if语句判断是线性判断，并且报错信息采用弹出框不合理
*/
if(StartTime==null||StartTime==""){
alert('安排时间不能为空')  ;
}
if(ResponsePerson==null||ResponsePerson==""){
alert('负责人不能为空')  ;
}
if(TechPerson==null||TechPerson==""){
alert('技术员不能为空')  ;
}
```

- 采用表单提交的表单验证----JS 表单验证/Jquery 框架验证 参见

1.
```
<form name="myForm" action="demo_form.asp" onsubmit="return validateForm()" method="post">
First name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

验证函数为：
```
function validateForm()
{
var x=document.forms["myForm"]["fname"].value;
if (x==null || x=="")
  {
  alert("First name must be filled out");
  return false;
  }
}
```

参见链接：http://www.runoob.com/js/js-form-validation.html

2. Jquery 框架验证：

思想和 JS 验证类似，参见 http://www.runoob.com/jquery/jquery-plugin-validate.html

建议采用这种方式进行表单验证

常用验证函数为：

```
/**
 * 检查输入的一串字符是否全部是数字
 * 输入:str  字符串
 * 返回:true 或 flase; true 表示为数字
 */
function checkNum(str){
    return str.match(/\D/) == null;
}


/**
 * 检查输入的一串字符是否为小数
 * 输入:str  字符串
 * 返回:true 或 flase; true 表示为小数
 */
function checkDecimal(str){
    if (str.match(/^-?\d+(\.\d+)?$/g) == null) {
        return false;
    }
    else {
        return true;
    }
}

/**
 * 检查输入的一串字符是否为整型数据
 * 输入:str  字符串
 * 返回:true 或 flase; true 表示为小数
 */
function checkInteger(str){
    if (str.match(/^[-+]?\d*$/) == null) {
        return false;
    }
    else {
        return true;
    }
}

/*************************************************************************
*****************/
/************************************    字    符    的    验    证
****************************************/
/*************************************************************************
*****************/
```

```
/**
 * 检查输入的一串字符是否是字符
 * 输入:str  字符串
 * 返回:true 或 flase; true 表示为全部为字符 不包含汉字
 */
function checkStr(str){
    if (/[^\x00-\xff]/g.test(str)) {
        return false;
    }
    else {
        return true;
    }
}


/**
 * 检查输入的一串字符是否包含汉字
 * 输入:str  字符串
 * 返回:true 或 flase; true 表示包含汉字
 */
function checkChinese(str){
    if (escape(str).indexOf("%u") != -1) {
        return true;
    }
    else {
        return false;
    }
}


/**
 * 检查输入的邮箱格式是否正确
 * 输入:str  字符串
 * 返回:true 或 flase; true 表示格式正确
 */
function checkEmail(str){
    if (str.match(/[A-Za-z0-9_-]+[@](\S*)(net|com|cn|org|cc|tv|[0-
9]{1,3})(\S*)/g) == null) {
        return false;
    }
    else {
        return true;
```

```
    }
}


/**
 * 检查输入的手机号码格式是否正确
 * 输入:str  字符串
 * 返回:true 或 flase; true 表示格式正确
 */
function checkMobilePhone(str){
    if (str.match(/^(?:13\d|15[89])-?\d{5}(\d{3}|\*{3})$/) == null) {
        return false;
    }
    else {
        return true;
    }
}


/**
 * 检查输入的固定电话号码是否正确
 * 输入:str  字符串
 * 返回:true 或 flase; true 表示格式正确
 */
function checkTelephone(str){
    if (str.match(/^(([0\+]\d{2,3}-)?(0\d{2,3})-)(\d{7,8})(-
(\d{3,}))?$/) == null) {
        return false;
    }
    else {
        return true;
    }
}

/**
 * 检查 QQ 的格式是否正确
 * 输入:str  字符串
 *  返回:true 或 flase; true 表示格式正确
 */
function checkQQ(str){
    if (str.match(/^\d{5,10}$/) == null) {
        return false;
    }
    else {
```

```
        return true;
    }
}

/**
 * 检查输入的身份证号是否正确
 * 输入:str  字符串
 *  返回:true 或 flase; true 表示格式正确
 */
function checkCard(str){
    //15 位数身份证正则表达式
    var arg1 = /^[1-9]\d{7}((0\d)|(1[0-2]))(([0|1|2]\d)|3[0-
1])\d{3}$/;
    //18 位数身份证正则表达式
    var arg2 = /^[1-9]\d{5}[1-9]\d{3}((0\d)|(1[0-
2]))(([0|1|2]\d)|3[0-1])((\d{4})|\d{3}[A-Z])$/;
    if (str.match(arg1) == null && str.match(arg2) == null) {
        return false;
    }
    else {
        return true;
    }
}

/**
 * 检查输入的 IP 地址是否正确
 * 输入:str  字符串
 *  返回:true 或 flase; true 表示格式正确
 */
function checkIP(str){
    var arg = /^(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-
4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-
4]\d|25[0-5])$/;
    if (str.match(arg) == null) {
        return false;
    }
    else {
        return true;
    }
}

/**
 * 检查输入的 URL 地址是否正确
 * 输入:str  字符串
```

```
 *  返回:true 或 flase; true 表示格式正确
 */
function checkURL(str){
    if (str.match(/(http[s]?|ftp):\/\/[^\/\.]+?\..+\w$/i) == null) {
        return false
    }
    else {
        return true;
    }
}

/**
 * 检查输入的字符是否具有特殊字符
 * 输入:str  字符串
 * 返回:true 或 flase; true 表示包含特殊字符
 * 主要用于注册信息的时候验证
 */
function checkQuote(str){
    var items = new Array("~", "`", "!", "@", "#", "$", "%", "^", "&"
, "*", "{", "}", "[", "]", "(", ")");
    items.push(":", ";", "'", "|", "\\", "<", ">", "?", "/", "<<", ">
>", "||", "//");
    items.push("admin", "administrators", "administrator", "管理员", "
系统管理员");
    items.push("select", "delete", "update", "insert", "create", "dro
p", "alter", "trancate");
    str = str.toLowerCase();
    for (var i = 0; i < items.length; i++) {
        if (str.indexOf(items[i]) >= 0) {
            return true;
        }
    }
    return false;
}


/*********************************************************************
*****************/
/************************************    时    间    的    验    证
****************************************/
/*********************************************************************
*****************/

/**
```

```
 * 检查日期格式是否正确
 * 输入:str  字符串
 * 返回:true 或 flase; true 表示格式正确
 * 注意：此处不能验证中文日期格式
 * 验证短日期（2007-06-05）
 */
function checkDate(str){
    //var value=str.match(/((^((1[8-9]\d{2})|([2-
9]\d{3}))(-)(10|12|0?[13578])(-)(3[01]|[12][0-9]|0?[1-9])$)|(^((1[8-
9]\d{2})|([2-9]\d{3}))(-)(11|0?[469])(-)(30|[12][0-9]|0?[1-
9])$)|(^((1[8-9]\d{2})|([2-9]\d{3}))(-)(0?2)(-)(2[0-8]|1[0-9]|0?[1-
9])$)|(^([2468][048]00)(-)(0?2)(-)(29)$)|(^([3579][26]00)(-)(0?2)(-)(
29)$)|(^([1][89][0][48])(-)(0?2)(-)(29)$)|(^([2-9][0-
9][0][48])(-)(0?2)(-)(29)$)|(^([1][89][2468][048])(-)(0?2)(-)(29)$)|(
^([2-9][0-
9][2468][048])(-)(0?2)(-)(29)$)|(^([1][89][13579][26])(-)(0?2)(-)(29)
$)|(^([2-9][0-9][13579][26])(-)(0?2)(-)(29)$))/);
    var value = str.match(/^(\d{1,4})(-|\/)(\d{1,2})\2(\d{1,2})$/);
    if (value == null) {
        return false;
    }
    else {
        var date = new Date(value[1], value[3] - 1, value[4]);
        return (date.getFullYear() == value[1] && (date.getMonth() +
1) == value[3] && date.getDate() == value[4]);
    }
}

/**
 * 检查时间格式是否正确
 * 输入:str  字符串
 * 返回:true 或 flase; true 表示格式正确
 * 验证时间(10:57:10)
 */
function checkTime(str){
    var value = str.match(/^(\d{1,2})(:)?(\d{1,2})\2(\d{1,2})$/)
    if (value == null) {
        return false;
    }
    else {
        if (value[1] > 24 || value[3] > 60 || value[4] > 60) {
            return false
        }
        else {
```

```
            return true;
        }
    }
}

/**
 * 检查全日期时间格式是否正确
 * 输入:str  字符串
 * 返回:true 或 flase; true 表示格式正确
 * (2007-06-05 10:57:10)
 */
function checkFullTime(str){
    //var value = str.match(/^(\d{1,4})(-
|\/)(\d{1,2})\2(\d{1,2}) (\d{1,2}):(\d{1,2}):(\d{1,2})$/);
    var value = str.match(/^(?:19|20)[0-9][0-9]-(?:(?:0[1-9])|(?:1[0-
2]))-(?:(?:[0-2][1-9])|(?:[1-3][0-1])) (?:(?:[0-2][0-3])|(?:[0-1][0-
9])):[0-5][0-9]:[0-5][0-9]$/);
    if (value == null) {
        return false;
    }
    else {
        //var date = new Date(checkFullTime[1], checkFullTime[3] - 1,
 checkFullTime[4], checkFullTime[5], checkFullTime[6], checkFullTime[
7]);
        //return (date.getFullYear() == value[1] && (date.getMonth()
+ 1) == value[3] && date.getDate() == value[4] && date.getHours() ==
value[5] && date.getMinutes() == value[6] && date.getSeconds() == val
ue[7]);
        return true;
    }

}




/**********************************************************************
*****************/
/*********************************  身  份  证  号  码  的  验  证
***********************************/
/**********************************************************************
*****************/

/**
```

```
 * 身份证 15 位编码规则：dddddd yymmdd xx p
 * dddddd：地区码
 * yymmdd：出生年月日
 * xx：顺序类编码，无法确定
 * p：性别，奇数为男，偶数为女
 * <p />
 * 身份证 18 位编码规则：dddddd yyyymmdd xxx y
 * dddddd：地区码
 * yyyymmdd：出生年月日
 * xxx:顺序类编码，无法确定，奇数为男，偶数为女
 * y：校验码，该位数值可通过前 17 位计算获得
 * <p />
 * 18 位号码加权因子为（从右到左）Wi = [ 7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2,1 ]
 * 验证位 Y = [ 1, 0, 10, 9, 8, 7, 6, 5, 4, 3, 2 ]
 * 校验位计算公式：Y_P = mod( ∑(Ai×Wi),11 )
 * i 为身份证号码从右往左数的 2...18 位；Y_P 为脚丫校验码所在校验码数组位置
 *
 */
var Wi = [7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2, 1];// 加权因子
var ValideCode = [1, 0, 10, 9, 8, 7, 6, 5, 4, 3, 2];// 身份证验证位值.10 代表 X
function IdCardValidate(idCard){
    idCard = trim(idCard.replace(/ /g, ""));
    if (idCard.length == 15) {
        return isValidityBrithBy15IdCard(idCard);
    }
    else
        if (idCard.length == 18) {
            var a_idCard = idCard.split("");// 得到身份证数组
            if (isValidityBrithBy18IdCard(idCard) && isTrueValidateCodeBy18IdCard(a_idCard)) {
                return true;
            }
            else {
                return false;
            }
        }
        else {
            return false;
        }
}
```

```
/**
 * 判断身份证号码为 18 位时最后的验证位是否正确
 * @param a_idCard 身份证号码数组
 * @return
 */
function isTrueValidateCodeBy18IdCard(a_idCard){
    var sum = 0; // 声明加权求和变量
    if (a_idCard[17].toLowerCase() == 'x') {
        a_idCard[17] = 10;// 将最后位为 x 的验证码替换为 10 方便后续操作
    }
    for (var i = 0; i < 17; i++) {
        sum += Wi[i] * a_idCard[i];// 加权求和
    }
    valCodePosition = sum % 11;// 得到验证码所位置
    if (a_idCard[17] == ValideCode[valCodePosition]) {
        return true;
    }
    else {
        return false;
    }
}

/**
 * 通过身份证判断是男是女
 * @param idCard 15/18 位身份证号码
 * @return 'female'-女、'male'-男
 */
function maleOrFemalByIdCard(idCard){
    idCard = trim(idCard.replace(/ /g, ""));// 对身份证号码做处理。包括字
符间有空格。
    if (idCard.length == 15) {
        if (idCard.substring(14, 15) % 2 == 0) {
            return 'female';
        }
        else {
            return 'male';
        }
    }
    else
        if (idCard.length == 18) {
            if (idCard.substring(14, 17) % 2 == 0) {
                return 'female';
            }
            else {
```

```
                return 'male';
            }
        }
        else {
            return null;
        }
}

/**
 * 验证18位数身份证号码中的生日是否是有效生日
 * @param idCard 18位书身份证字符串
 * @return
 */
function isValidityBrithBy18IdCard(idCard18){
    var year = idCard18.substring(6, 10);
    var month = idCard18.substring(10, 12);
    var day = idCard18.substring(12, 14);
    var temp_date = new Date(year, parseFloat(month) - 1, parseFloat(
day));
    // 这里用getFullYear()获取年份，避免千年虫问题
    if (temp_date.getFullYear() != parseFloat(year) ||
    temp_date.getMonth() != parseFloat(month) - 1 ||
    temp_date.getDate() != parseFloat(day)) {
        return false;
    }
    else {
        return true;
    }
}

/**
 * 验证15位数身份证号码中的生日是否是有效生日
 * @param idCard15 15位书身份证字符串
 * @return
 */
function isValidityBrithBy15IdCard(idCard15){
    var year = idCard15.substring(6, 8);
    var month = idCard15.substring(8, 10);
    var day = idCard15.substring(10, 12);
    var temp_date = new Date(year, parseFloat(month) - 1, parseFloat(
day));
    // 对于老身份证中的你年龄则不需考虑千年虫问题而使用getYear()方法
    if (temp_date.getYear() != parseFloat(year) ||
    temp_date.getMonth() != parseFloat(month) - 1 ||
```

```
            temp_date.getDate() != parseFloat(day)) {
                return false;
            }
            else {
                return true;
            }
        }

        //去掉字符串头尾空格
        function trim(str){
            return str.replace(/(^\s*)|(\s*$)/g, "");
        }
```

代码 Pythonic 写法

输出方式：
Print('{key}'.format(key=value))
示例：print('{result} from {argith}'.format(result=s,argith='quicksort'))
示例 2（快速排序）：

```
#快速排序
def quicksort(array):
    less=[]
    greater=[]
    if len(array)<=1:
        return array
    pivot=array.pop()

    for x in array:

        if x<=pivot:
            less.append(x)

        else:
            greater.append(x)
    print(pivot)
    return quicksort(less)+[pivot]+quicksort(greater)

s=quicksort([9,8,4,5,32,64,2,1,0,10,19,26])
print('{result} from {argith}'.format(result=s,argith='quicksort'))
```

ajax 传值 403Forbidden

在Ajax传值的时候，出现403Forbiden的解决方法

一般是csrf 防护原因，请参考
https://docs.djangoproject.com/en/1.8/ref/csrf/
方法1--------------

```
@csrf_exempt
def profile_delte(request):
del_file=request.POST.get("delete_file",'')
```

方法2
注释掉中间件: 'django.middleware.csrf.CsrfViewMiddleware',

方法3
若是表单提交，在表单下方添加

```
<form id="form" method="post">
{% csrf_token %}
</form>
```

方法4- 参考链接：http://dengzuoheng.github.io/Django-jQuery-ajax/

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

$.ajaxSetup({
    beforeSend: function(xhr, settings) {
        if (!csrfSafeMethod(settings.type) && !this.crossDomain) {
            xhr.setRequestHeader("X-CSRFToken", csrftoken);
        }
    }
});
$.ajax({
url:"http://127.0.0.1:8000/RequestAjax/",
data:{"data":some_data},
async:true,
dataType:"json",
type:"POST",
success:function(result){
    //do something

}
});
```

```
#views.py那里
from django.template import RequestContext
def some(request):
```

```
return render_to_response('some.html',context_instance=RequestContext(request))
```

js 获取传递的 JSON 值，并显示在模态框中的 table 中

```
<script type="text/javascript">
    $("#MName").change(function () {
        var SelectedMName = $("#MName option:selected").val();

        $.ajax({
            /*
Ajax传值和对返回值进行处理
*/
            url: '{% url 'Warehouse_Stroage_Details'%}',
            type: 'POST',
            data: {
                'MName':SelectedMName
            },
            success: function (data) {
                //alert('hello world');
                  //alert(data);
                  console.error(data);
                  console.error(data.length);
                  var MInfo;
                  for(item=0;item<data.length;item++){
                  /*
                  console.error(data[item][0]) ;  //材料数量
                  console.error(data[item][1]); //whcode
                  console.error(data[item][2]);   //名称
                  console.error(data[item][3]);     //规格 model
                  console.error(data[item][4]);   //单位


                  $("#MModel").text(data[item][3]);
                  $("#Whcode").text(data[item][1]);
                  $("#MNumber").text(data[item][0]);
                  $("#Munit").text(data[item][4]);
                  $("#MaterailName").text(data[item][2]);
        */

                    MInfo+="<tr><td>"+data[item][2]+"</td><td>"+
data[item][3]+"</td><td>"+data[item][1]+
```

```
"</td><td>"+data[item][0]+"</td><td>"+
          data[item][4]+ "</td></tr>";

                        //生成表格文件



                    }
                      console.error(MInfo);
                    $("#MaterialSelectedInfo").html(MInfo); //显示循环生成的表格
                    $("#myModal").modal('show');  //显示模态框
                     //alert(data.length);


                }
            }
                );



          });

      </script>
```

显示的模态框 –参考 BootStrap 文件

```
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-
labelledby="myModalLabel">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-
label="Close"><span aria-hidden="true">&times;</span></button>
        <h4 class="modal-title" id="myModalLabel">该原料的库存信息</h4>
      </div>

          <table class="table table-bordered">
        <thead>
           <tr >

            <td>材料名称</td>
            <td>材料规格</td>
           <td>whcode</td>
            <td>数量</td>
           <td>单位</td>
```

```
            </tr>
          </thead>
          <tbody id="MaterialSelectedInfo">

            </tbody>

      </table>


        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-default" data-dismiss="modal">关闭
</button>
          <button type="button" class="btn btn-primary">确定</button>
        </div>
      </div>
    </div>
```

SQL 判断某列中是否包含中文字符或者英文字符

```
select * from 表名 where 某列 like '%[吖-座]%'
      www.2cto.com
select * from 表名 where 某列 like '%[a-z]%'
```

JSON 可以作为传递信息的文件，但是后台在传递一个模型实例时，需要经过序列化操作

例如：

```
component_info=serializers.serialize("json",Oemcomponent.objects.filter(component_num=c
omponent_num))#对零件表模型的序列化操作
```

也可以采用以下的方法进行序列化操作：

```
def toJSON(self):
    fields = []
    for field in self._meta.fields:
        fields.append(field.name)
    d = {}
    for attr in fields:
        d[attr] = getattr(self, attr)

    return json.dumps(d)
```

然而第一种方法适应大多数模型，可是需要对象是 queryset，如果只是一个实例的话，是

无法序列化的，对于第二种可以针对一个实例，可是对于含有外键的属性则无能为力
注意：json 文件在网页脚本中得到的是一个字符串，需要进行解析才能得到 json 键值对

BootStrap 级联菜单 （http://www.cnblogs.com/xiexingen/）

```
<!DOCTYPE html>
<html lang="en">

    <head>
        <meta charset="UTF-8">
        <title>自定义表单元素</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet" href="../css/bootstrap.css">
        <!--自定义站点样式-->
        <link rel="stylesheet" href="../css/site.css">
        <script src="../lib/jquery.js"></script>
        <script src="../lib/bootstrap.js"></script>
        <!--工具方法-->
        <script src="../scripts/global.js"></script>
        <!--插件-->
        <script src="../scripts/jquery.smart-form.js"></script>
    </head>

    <body>
        <div class="panel panel-default">
            <div class="panel-heading">
                <label>级联的支持</label>
            </div>
            <div class="panel-body">
                <form action="#" id="formContainer" class="form form-horizontal"></form>
            </div>
        </div>
        <div class="panel panel-default">
            <div class="panel-heading">
                <label>介绍</label>
            </div>
            <div class="panel-body">
                <h3 class="lead">自定义级联</h3>
                <blockquote>
                    <p>配置表单的值时，需要将后台传递过来的数据格式修改成指定格式的数据源，可以自定义方法任意修改</p>
                    <p> 在 表 单 生 成 完 成 后 ， 需 要 调 用
```

<code>global.Fn.CascadeSelect({targets:[''],primaryKey:'',relativeKey:''})</code></p>
</blockquote>
<h4>参数说明：</h4>
<blockquote>
targets:字符串数组，顺序必须为依赖的先后顺序的 id，如：省、市、县
primaryKey:主键 id，默认为：data-id,
relativeKey:父级 id，默认为 data-parentId
</blockquote>

<script>
$(function () {
var eles = [
[
{ ele: { type: 'select', name: 'province', title: '省:', withNull: true, items: [{ text: '广东', value: 'GuangDong', extendAttr: { id: 1000 } }, { text: '湖南', value: 'HuNan', extendAttr: { id: 2000 } }] } },
{ ele: { type: 'select', name: 'city', title: '市:', withNull: true, items: [{ "text": "广州", "value": "GuangZhou", "extendAttr": { "id": "1000001", "parentId": "1000" } }, { "text": "花都", "value": "HuaDu", "extendAttr": { "id": "1000002", "parentId": "1000" } }, { "text": "邵阳", "value": "ShaoYang", "extendAttr": { "id": "2000001", "parentId": "2000" } }, { "text": "长沙", "value": "ChangSha", "extendAttr": { "id": "2000002", "parentId": "2000" } }] } },
{ ele: { type: 'select', name: 'region', title: '区:', withNull: true, items: [{ "text": "天河区", "value": "TH", "extendAttr": { "id": "1000001001", "parentId": "1000001" } }, { "text": "海珠区", "value": "HZ", "extendAttr": { "id": "1000001002", "parentId": "1000001" } }, { "text": "越秀区", "value": "YX", "extendAttr": { "id": "1000001003", "parentId": "1000001" } }, { "text": "白云区", "value": "BY", "extendAttr": { "id": "1000001004", "parentId": "1000001" } }, { "text": "花都区", "value": "HD", "extendAttr": { "id": "1000002001", "parentId": "1000002" } }, { "text": "aa 县", "value": "aa", "extendAttr": { "id": "2000001001", "parentId": "2000001" } }, { "text": "望城区", "value": "wc", "extendAttr": { "id": "2000002001", "parentId": "2000002" } }, { "text": "雨花区", "value": "yh", "extendAttr": { "id": "2000002002", "parentId": "2000002" } }] } },
]
];
var bsForm = new BSForm({ eles: eles,autoLayout: true }).Render('formContainer', function (sf) {
//编辑页面的绑定
sf.InitFormData({
province: 'GuangDong',
city: 'GuangZhou',
region:'TH'
});
//必须先赋值再生成插件
global.Fn.CascadeSelect({ targets: ['province', 'city', 'region'], primaryKey:

```
'data-id', relativeKey: 'data-parentId' });
                });
                });
        </script>
    </body>
```

Sql-server 分页查询

http://blog.csdn.net/qiaqia609/article/details/41445233
http://www.cnblogs.com/zyw-205520/archive/2012/10/23/2736223.html

qlserver 数据库分页查询一直是 Sqlserver 的短板，闲来无事，想出几种方法，假设有表 ARTICLE,字段 ID、YEAR...(其他省略)，数据 53210 条(客户真实数据，量不大)，分页查询每页 30 条，查询第 1500 页（即第 45001-45030 条数据），字段 ID 聚集索引，YEAR 无索引，Sqlserver 版本：2008R2

第一种方案、最简单、普通的方法：

代码如下:

```
SELECT TOP 30 * FROM ARTICLE WHERE ID NOT IN(SELECT TOP 45000 ID FROM ARTICLE ORDER BY YEAR DESC, ID DESC) ORDER BY YEAR DESC,ID DESC
```

平均查询 100 次所需时间：45s

第二种方案

```
SELECT * FROM (      SELECT TOP 30 * FROM (SELECT TOP 45030 * FROM ARTICLE ORDER BY YEAR DESC, ID DESC) f ORDER BY f.YEAR ASC, f.ID DESC) s ORDER BY s.YEAR DESC,s.ID DESC
```

平均查询 100 次所需时间：138S

第三种方案：

```
SELECT * FROM ARTICLE w1,
(
    SELECT TOP 30 ID FROM
    (
        SELECT TOP 50030 ID, YEAR FROM ARTICLE ORDER BY YEAR DESC, ID DESC
    ) w ORDER BY w.YEAR ASC, w.ID ASC
) w2 WHERE w1.ID = w2.ID ORDER BY w1.YEAR DESC, w1.ID DESC
```

第四种方案：

```
SELECT * FROM ARTICLE w1
    WHERE ID in
        (
            SELECT top 30 ID FROM
            (
                SELECT top 45030 ID, YEAR FROM ARTICLE ORDER BY YEAR DESC, ID DESC
            ) w ORDER BY w.YEAR ASC, w.ID ASC
```

)

ORDER BY w1.YEAR DESC, w1.ID DESC

第五种方案：

SELECT w2.n, w1.* FROM ARTICLE w1, (SELECT TOP 50030 row_number() OVER (ORDER BY YEAR DESC, ID DESC) n, ID FROM ARTICLE) w2 WHERE w1.ID = w2.ID AND w2.n > 50000 ORDER BY w2.n ASC

由此可见在查询页数靠前时，效率 3>4>5>2>1，页码靠后时 5>4>3>1>2，再根据用户习惯，一般用户的检索只看最前面几页，因此选择 3 4 5 方案均可，若综合考虑方案 5 是最好的选择，但是要注意 SQL2000 不支持 row_number()函数，由于时间和条件的限制没有做更深入、范围更广的测试，有兴趣的可以仔细研究下

Django 对 POST 参数查询后的结果进行分页

一般在采用 POST 传递查询参数后，需要对查询出来的结果进行分页。但是在进行分页时 Django 处理方法又为 POST 方法。在本次项目的开发过程中对于这个问题的解决思路是：首先将 POST 查询的结果进行保存（或者缓存），随后在 GET 方法分页时进行取出，然后进行分页显示。在本项目中，初步采用的是 session 保存 POST 查询结果，在 GET 方法的处理中取出结果

示例代码如下：

```
SearchedResultInfo=MaterialDataInfo.getStroageDataInfoByClass(selection=MName)
            #return HttpResponse('名称和规格均不为空')
        request.session['searedresultinfo']=SearchedResultInfo
        pageinator=Paginator(SearchedResultInfo,15)#分页操作，每页5条数据

        page=request.GET.get('page')
        try:

            SearchedResultInfo_List=pageinator.page(page)
        except PageNotAnInteger:
            SearchedResultInfo_List=pageinator.page(1)
        except EmptyPage:
            SearchedResultInfo_List=pageinator.page(pageinator.num_pages)


        return
render(request,'OEM_Manage/Warehouse_Depart/SearchedResultByNameAndModel.html',{'Search
edResultInfo':SearchedResultInfo,'SearchedResultInfo_List':SearchedResultInfo_List,'pag
einator':pageinator})
    elif request.method=='GET':
        SearchedResultInfo=request.session.get('searedresultinfo')
        pageinator=Paginator(SearchedResultInfo,15)#分页操作，每页5条数据

        page=request.GET.get('page')
```

```
        try:

            SearchedResultInfo_List=pageinator.page(page)
        except PageNotAnInteger:
            SearchedResultInfo_List=pageinator.page(1)
        except EmptyPage:
            SearchedResultInfo_List=pageinator.page(pageinator.num_pages)
            return
render(request,'OEM_Manage/Warehouse_Depart/SearchedResultByNameAndModel.html',{'SearchedResultInfo':SearchedResultInfo,'SearchedResultInfo_List':SearchedResultInfo_List,'pageinator':pageinator})
```

在对视图进行分页查询时，在处理无自增字段的视图时，可创建视图临时表，在临时表中添加自增字段随后进行查询。

创建临时表

```
/*SELECT  identity(int,1,1) as ID ,[MNumber]
      ,[WhCode]
      ,[MName]
      ,[Model]
      ,[PrimaryUnitName] into #tempTable

  FROM [erptest2].[dbo].[Kf_NowQuan]
  */
```

对临时表进行查询

```
  SELECT [MNumber]
      ,[WhCode]
      ,[MName]
      ,[Model]
      ,[PrimaryUnitName] FROM #tempTable WHERE ID=660
```

django 中文乱码问题解决

1、在所有涉及到中文显示的页面头加上
#coding:utf-8
2、修改 settings.py 文件，添加
FILE_CHARSET = 'utf-8'
DEFAULT_CHARSET = 'utf-8'
然后修改 LANGUAGE_CODE = 'zh-cn'
3、用记事本打开，另存为 utf-8 编码的。

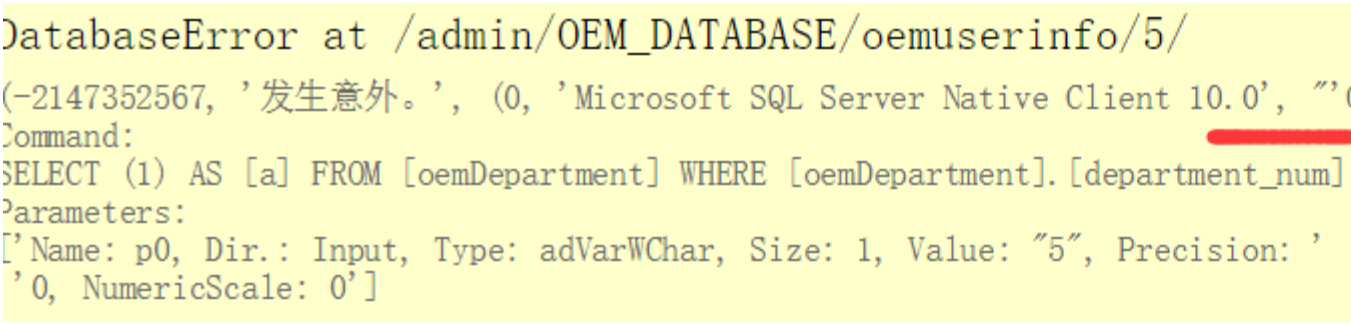
Django 序列化（Decimal 序列化问题）

参考链接：http://www.tuicool.com/articles/mmaeqiY
https://github.com/bluedazzle/django-simple-serializer

```
class DecimalEncoder(json.JSONEncoder):
    def _iterencode(self, o, markers=None):
        if isinstance(o, decimal.Decimal):
            # wanted a simple yield str(o) in the next line,
            # but that would mean a yield on the line with super(...),
            # which wouldn't work (see my comment below), so...
            return (str(o) for o in [o])
        return super(DecimalEncoder, self)._iterencode(o, markers)
```

Then use it like so:
json.dumps({'x': decimal.Decimal('5.5')}, cls=DecimalEncoder)

sql-server2008 中对数据库命令运行出错解决



修改 sql-server_ado 中 Compiler.py 文件中
'''
#sql += ' OFFSET %d ROWS' % (self.query.low_mark or 0)
if self.query.high_mark is not None:
#sql += ' FETCH NEXT %d ROWS ONLY' % (self.query.high_mark - self.query.low_mark)
#在 SQL-Server2008 中需要注释掉以上语句，因为在 SQL-Server2008 中不支持 offset
'''

1 Mysql 去除 MySQL 去除字段中的换行和回车符

UPDATE tablename SET field = REPLACE(REPLACE(field, CHAR(10), ”), CHAR(13), ”);

char(10): 换行符
char(13): 回车符

问题产生原因：

2 种方法生成 excel 模式的报表：

1）手动生成

将表中的数据导出，生成 CSV 文件。

用 mysqldump 导出数据

#mysqldump -u xxx -p --tab=/tmp/ --fields-terminated-by="#" DBName TBName

将会在 tmp 目录下生成 TBName.txt 文件。

在 EXCEL 中导入生成的 txt 文件

2）直接生成 csv 格式文件

mysqldump -u samu -p -T --fields-terminated-by="," --fields-enclosed-by=""
--lines-terminated-by="\n" --fields-escaped-by="" test Customer

或者：

mysqldump -u samu -p --tab=/tmp/ --fields-terminated-by="," --fields-enclosed-by=""
--lines-terminated-by="\n" --fields-escaped-by="" test Customer

但是，无论上面哪一种方法，如果表的某个列里包含回车符或者换行符，
那么生成的 CSV 文件或者进行 excel 导入，都会将原本的 1 行数据，拆分成 2 行。
因为 CSV 或者 excel 导入，是按数据的行来认定数据条数。

所以，必须在此之前，将字段中的回车符或者换行符，进行替换。

参考网页错误集

语用错误

让我们从基础开始，从那些刚学习编程的人钻研语法之前碰到的事情开始。如果你已经编过一些程了，那么以下这些可能看起来十分的简

单；如果你曾经尝试过教新手们怎么编程，它们可能就不这么简单了。

在交互提示符中输入 Python 代码

在>>>交互提示符中你只能输入 Python 代码，而不是系统命令。时常有人在这个提示符下输入 emacs，ls，或者 edit 之类的命令，这些可不是 Python 代码。在 Python 代码中确实有办法来调用系统命令（例如 os.system 和 os.popen），但可不是像直接输入命令这么直接。如果你想要在交互提示符中启动一个 Python 文件，请用 import file，而不是系统命令 python file.py。

Print 语句（仅仅）是在文件中需要

因为交互解释器会自动的讲表达式的结果输出，所以你不需要交互的键入完整的 print 语句。这是个很棒的功能，但是记住在代码文件里，通常你只有用 print 语句才能看得到输出。

小心 Windows 里的自动扩展名

如果你在 Windows 里使用记事本来编辑代码文件的话，当你保持的时候小心选择“所有文件”（All Files）这个类型，并且明确的给你的文件加一个.py 的后缀。不然的话记事本会给你的文件加一个.txt 的扩展名，使得在某些启动方法中没法跑这个程序。更糟糕的是，像 Word 或者是写字板一类的文字处理软件还会默认的加上一些格式字符，而

这些字符 Python 语法是不认的。所以记得，在 Windows 下总是选“所有文件”（All Files），并保存为纯文本，或者使用更加“编程友好”的文本编辑工具，比如 IDLE。在 IDLE 中，记得在保存时手动加上.py 的扩展名。

在 Windows 下点击图标的问题

在 Windows 下，你能靠点击 Python 文件来启动一个 Python 程序，但这有时会有问题。首先，程序的输出窗口在程序结束的瞬间也就消失了，要让它不消失，你可以在文件最后加一条 raw_input()的调用。另外，记住如果有错的话，输出窗口也就立即消失了。要看到你的错误信息的话，用别的方法来调用你的程序：比如从系统命令行启动，通过提示符下用 import 语句，或者 IDLE 菜单里的选项，等等。

Import 只在第一次有效

你可以在交互提示符中通过 import 一个文件来运行它，但是这只会在一个会话中起一次作用；接下来的 import 仅仅是返回这个已经加载的模块。要想强制 Python 重新加载一个文件的代码，请调用函数 reload(module)来达到这个目的。注意对 reload 请使用括号，而 import 不要使用括号。

空白行（仅仅）在交互提示符中有作用

在模块文件中空白行和注释统统会被忽略掉，但是在交互提示符中键入代码时，空白行表示一个复合语句的结束。换句话说，空白行告诉交互提示符你完成了一个复合语句；在你真正完成之前不要键入回车。事实上当你要开始一个新的语句时，你需要键入一个空行来结束当前的语句——交互提示符一次只运行一条语句。

代码错误

一旦你开始认真写 Python 代码了，接下来了一堆陷阱就更加危险了——这些都是一些跨语言特性的基本代码错误，并常常困扰不细心的程序员。

别忘了冒号

这是新手程序员最容易犯的一个错误：别忘了在复合语句的起始语句（if，while， for 等语句的第一行）结束的地方加上一个冒号“：”。也许你刚开始会忘掉这个，但是到了很快这就会成为一个下意识的习惯。课堂里 75%的学生当天就可以记住这个。

初始化变量

在 Python 里，一个表达式中的名字在它被赋值之前是没法使用的。这是有意而为的：这样能避免一些输入失误，同时也能避免默认究竟应

该是什么类型的问题（0，None，””，[]，?）。记住把计数器初始化为0，列表初始化为[]，以此类推。

从第一列开始

确保把顶层的，未嵌套的代码放在最左边第一列开始。这包括在模块文件中未嵌套的代码，以及在交互提示符中未嵌套的代码。Python 使用缩进的办法来区分嵌套的代码段，因此在你代码左边的空格意味着嵌套的代码块。除了缩进以外，空格通常是被忽略掉的。

缩进一致

在同一个代码块中避免讲 tab 和空格混用来缩进，除非你知道运行你的代码的系统是怎么处理 tab 的。否则的话，在你的编辑器里看起来是 tab 的缩进也许 Python 看起来就会被视作是一些空格。保险起见，在每个代码块中全都是用 tab 或者全都是用空格来缩进；用多少由你决定。

在函数调用时使用括号

无论一个函数是否需要参数，你必须要加一对括号来调用它。即，使用 function()，而不是 function。Python 的函数简单来说是具有特殊功能（调用）的对象，而调用是用括号来触发的。像所有的对象一样，他们也可以被赋值给变量，并且间接的使用他们：x=function:x()。
在 Python 的培训中，这样的错误常常在文件的操作中出现。通常会看

到新手用 file.close 来关闭一个问题，而不是用 file.close()。因为在 Python 中引用一个函数而不调用它是合法的，因此不使用括号的操作（file.close）无声的成功了，但是并没有关闭这个文件！

在 Import 时不要使用表达式或者路径

在系统的命令行里使用文件夹路径或者文件的扩展名，但不要在 import 语句中使用。即，使用 import mod，而不是 import mod.py，或者 import dir/mod.py。在实际情况中，这大概是初学者常犯的第二大错误了。因为模块会有除了.py 以为的其他的后缀（例如，.pyc），强制写上某个后缀不仅是不合语法的，也没有什么意义。

和系统有关的目录路径的格式是从你的模块搜索路径的设置里来的，而不是 import 语句。你可以在文件名里使用点来指向包的子目录（例如，import dir1.dir2.mod），但是最左边的目录必须得通过模块搜索路径能够找到，并且没有在 import 中没有其他路径格式。不正确的语句 import mod.py 被 Python 认为是要记在一个包，它先加载一个模块 mod，然后试图通过在一个叫做 mod 的目录里去找到叫做 py 的模块，最后可能什么也找不到而报出一系列费解的错误信息。

不要在 Python 中写 C 代码

以下是给不熟悉 Python 的 C 程序员的一些备忘贴士：

- 在 if 和 while 中条件测试时，不用输入括号（例如，if (X==1):）。如果你喜欢的话，加上括号也无妨，只是在这里是完全多余的。
- 不要用分号来结束你的语句。从技术上讲这在 Python 里是合法的，但是这毫无用处，除非你要把很多语句放在同一行里（例如，x=1; y=2; z=3）。
- 不要在 while 循环的条件测试中嵌入赋值语句（例如，`while ((x=next() != NULL))`。在 Python 中，需要表达式的地方不能出现语句，并且赋值语句不是一个表达式。

编程错误

下面终于要讲到当你用到更多的 Python 的功能（数据类型，函数，模块，类等等）时可能碰到的问题了。由于篇幅有限，这里尽量精简，尤其是对一些高级的概念。要想了解更多的细节，敬请阅读 Learning Python, 2nd Edition 的“小贴士”以及“Gotchas”章节。

打开文件的调用不使用模块搜索路径

当你在 Python 中调用 open()来访问一个外部的文件时，Python 不会使用模块搜索路径来定位这个目标文件。它会使用你提供的绝对路径，或者假定这个文件是在当前工作目录中。模块搜索路径仅仅为模块加载服务的。

不同的类型对应的方法也不同

列表的方法是不能用在字符串上的，反之亦然。通常情况下，方法的调用是和数据类型有关的，但是内部函数通常在很多类型上都可以使用。举个例子来说，列表的 reverse 方法仅仅对列表有用，但是 len 函数对任何具有长度的对象都适用

不能直接改变不可变数据类型

记住你没法直接的改变一个不可变的对象（例如，元组，字符串）：

```
1  T = (1, 2, 3)
2  T[2] = 4      # 错误
```

用切片，联接等构建一个新的对象，并根据需求将原来变量的值赋给它。因为 Python 会自动回收没有用的内存，因此这没有看起来那么浪费：

```
1  T = T[:2] + (4,)   # 没问题了: T 变成了 (1, 2, 4)
```

使用简单的 for 循环而不是 while 或者 range

当你要从左到右遍历一个有序的对象的所有元素时，用简单的 for 循环（例如，`for x in seq:`）相比于基于 while-或者 range-的计数循环而言会更容易写，通常运行起来也更快。除非你一定需要，尽量避免在一个 for 循环里使用 range：让 Python 来替你解决标号的问题。在下面的例子中三个循环结构都没有问题，但是第一个通常来说更好；在 Python 里，简单至上。

```
S = "lumberjack"

for c in S: print c                                    # 最简单

for i in range(len(S)): print S[i]          # 太多了

i = 0                    # 太多了
while i < len(S): print S[i]; i += 1
```

不要试图从那些会改变对象的函数得到结果

诸如像方法 list.append()和 list.sort()一类的直接改变操作会改变一个对象，但不会将它们改变的对象返回出来（它们会返回 None）；正确的做法是直接调用它们而不要将结果赋值。经常会看见初学者会写诸如此类的代码：

```
mylist = mylist.append(X)
```

目的是要得到 append 的结果，但是事实上这样做会将 None 赋值给 mylist，而不是改变后的列表。更加特别的一个例子是想通过用排序后的键值来遍历一个字典里的各个元素，请看下面的例子：

```
D = {...}
for k in D.keys().sort(): print D[k]
```

差一点儿就成功了——keys 方法会创建一个 keys 的列表，然后用 sort 方法来将这个列表排序——但是因为 sort 方法会返回 None，这个循环会失败，因为它实际上是要遍历 None（这可不是一个序列）。要改正这段代码，将方法的调用分离出来，放在不同的语句中，如下：

```
Ks = D.keys()
Ks.sort()
for k in Ks: print D[k]
```

只有在数字类型中才存在类型转换

在 Python 中，一个诸如 123+3.145 的表达式是可以工作的——它会自动将整数型转换为浮点型，然后用浮点运算。但是下面的代码就会出错了：

```
S = "42"
I = 1
X = S + I                    # 类型错误
```

这同样也是有意而为的，因为这是不明确的：究竟是将字符串转换为数字（进行相加）呢，还是将数字转换为字符串（进行联接）呢？在 Python 中，我们认为“明确比含糊好”（即，EIBTI（Explicit is better than implicit）），因此你得手动转换类型：

```
X = int(S) + I       # 做加法: 43
X = S + str(I)       # 字符串联接: "421"
```

循环的数据结构会导致循环

尽管这在实际情况中很少见，但是如果一个对象的集合包含了到它自己的引用，这被称为**循环对象**（cyclic object）。如果在一个对象中发现一个循环，Python 会输出一个[...]，以避免在无限循环中卡住：

```
>>> L = ['grail']   # 在 L 中又引用 L 自身会
>>> L.append(L)         # 在对象中创造一个循环
>>> L
['grail', [...]]
```

除了知道这三个点在对象中表示循环以外，这个例子也是很值得借鉴的。因为你可能无意间在你的代码中出现这样的循环的结构而导致你

的代码出错。如果有必要的话，维护一个列表或者字典来表示已经访问过的对象，然后通过检查它来确认你是否碰到了循环。

赋值语句不会创建对象的副本，仅仅创建引用

这是 Python 的一个核心理念，有时候当行为不对时会带来错误。在下面的例子中，一个列表对象被赋给了名为 L 的变量，然后 L 又在列表 M 中被引用。内部改变 L 的话，同时也会改变 M 所引用的对象，因为它们俩都指向同一个对象。

```
>>> L =[1, 2, 3]                    # 共用的列表对象
>>> M =['X', L, 'Y']          # 嵌入一个到 L 的引用
>>> M
['X', [1, 2, 3], 'Y']

>>> L[1] =0       # 也改变了 M
>>> M
['X', [1, 0, 3], 'Y']
```

通常情况下只有在稍大一点的程序里这就显得很重要了，而且这些共用的引用通常确实是你需要的。如果不是的话，你可以明确的给他们创建一个副本来避免共用的引用；对于列表来说，你可以通过使用一个空列表的切片来创建一个顶层的副本：

```
>>> L =[1, 2, 3]
>>> M =['X', L[:], 'Y']      # 嵌入一个 L 的副本

>>> L[1] =0        # 仅仅改变了 L，但是不影响 M
>>> L
[1, 0, 3]
>>> M
['X', [1, 2, 3], 'Y']
```

切片的范围起始从默认的 0 到被切片的序列的最大长度。如果两者都省略掉了，那么切片会抽取该序列中的所有元素，并创造一个顶层的副本（一个新的，不被公用的对象）。对于字典来说，使用字典的 dict.copy()方法。

静态识别本地域的变量名

Python 默认将一个函数中赋值的变量名视作是本地域的，它们存在于该函数的作用域中并且仅仅在函数运行的时候才存在。从技术上讲，Python 是在编译 def 代码时，去静态的识别本地变量，而不是在运行时碰到赋值的时候才识别到的。如果不理解这点的话，会引起人们的误解。比如，看看下面的例子，当你在一个引用之后给一个变量赋值会怎么样：

```
1  >>> X = 99
2  >>> def func():
3  ...         print X           # 这个时候还不存在
4  ...         X = 88    # 在整个 def 中将 X 视作本地变量
5  ...
6  >>> func( )                    # 出错了！
```

你会得到一个“未定义变量名”的错误，但是其原因是很微妙的。当编译这则代码时，Python 碰到给 X 赋值的语句时认为在这个函数中的任何地方 X 会被视作一个本地变量名。但是之后当真正运行这个函数时，执行 print 语句的时候，赋值语句还没有发生，这样 Python 便会报告一个“未定义变量名”的错误。

事实上，之前的这个例子想要做的事情是很模糊的：你是想要先输出那个全局的 X，然后创建一个本地的 X 呢，还是说这是个程序的错误？如果你真的是想要输出这个全局的 X，你需要将它在一个全局语句中声明它，或者通过包络模块的名字来引用它。

默认参数和可变对象

在执行 def 语句时，默认参数的值只被解析并保存一次，而不是每次在调用函数的时候。这通常是你想要的那样，但是因为默认值需要在每次调用时都保持同样对象，你在试图改变可变的默认值（mutable defaults）的时候可要小心了。例如，下面的函数中使用一个空的列表作为默认值，然后在之后每一次函数调用的时候改变它的值：

```
>>> def saver(x=[]):      # 保存一个列表对象
...         x.append(1)        # 并每次调用的时候
...         print x          # 改变它的值
...
>>> saver([2])                  # 未使用默认值
[2, 1]
>>> saver()                         # 使用默认值
[1]
>>> saver()                         # 每次调用都会增加!
[1, 1]
>>> saver()
[1, 1, 1]
```

有的人将这个视作 Python 的一个特点——因为可变的默认参数在每次函数调用时保持了它们的状态，它们能提供像 C 语言中静态本地函数变量的类似的一些功能。但是，当你第一次碰到它时会觉得这很奇

怪，并且在 Python 中有更加简单的办法来在不同的调用之间保存状态（比如说类）。

要摆脱这样的行为，在函数开始的地方用切片或者方法来创建默认参数的副本，或者将默认值的表达式移到函数里面；只要每次函数调用时这些值在函数里，就会每次都得到一个新的对象：

```
>>> def saver(x=None):
...         if x is None: x = []      # 没有传入参数?
...         x.append(1)                          # 改变新的列表
...         print x
...
>>> saver([2])                                   # 没有使用默认值
[2, 1]
>>> saver()                                        # 这次不会变了
[1]
>>> saver()
[1]
```

其他常见的编程陷阱

下面列举了其他的一些在这里没法详述的陷阱：

- 在顶层文件中语句的顺序是有讲究的：因为运行或者加载一个文件会从上到下运行它的语句，所以请确保将你未嵌套的函数调用或者类的调用放在函数或者类的定义之后。
- reload 不影响用 from 加载的名字：reload 最好和 import 语句一起使用。如果你使用 from 语句，记得在 reload 之后重新运行一遍 from，否则你仍然使用之前老的名字。

- 在多重继承中混合的顺序是有讲究的：这是因为对 superclass 的搜索是从左到右的，在类定义的头部，在多重 superclass 中如果出现重复的名字，则以最左边的类名为准。
- 在 try 语句中空的 except 子句可能会比你预想的捕捉到更多的错误。在 try 语句中空的 except 子句表示捕捉所有的错误，即便是真正的程序错误，和 sys.exit()调用，也会被捕捉到。
- 兔子可能会比他们看起来更加危险。（原句 Bunnies can be more dangerous than they seem. 意思是一些看起来比较细微的问题实际上可能更危险。——译者注