Python 基础教程

version	0.5
author	枫无眠
msn	maple5218@163.com
create date	2009-08-01

Change List

version	date	author	content
0.1	2009-08-01	枫无眠	简介
0.2	2009-08-02	..	基础编程
0.3	2009-08-03	..	基础编程(文件操作)
0.4	2009-08-04	..	数据库编程,dbapi2 规范,cx_oralce
0.5	2009-08-05	..	cx_oralce 例子

目录

1 简介

Python 是一种脚本语言，已经有 20 多年的历史，比现在流行的 Java 和 C#要早很多年。不要一听说是脚本语言就认为他只能做一些简单的事情。其实凡是你能想到的 Java 和 C#能做的编程，Pyton 都能胜任。比如网络编程，游戏编程，web 编程等等，甚至在 smbian 的手机上都能使用 Python 来进行编程。Google 推出的 google engine 云计算环境，首先发布的就是 python 的平台（Python 语言的创始人都在为 Google 服务，直到一年以后才发布 java 的平台），对 web 编程感兴趣的同学可以去申请一个帐号来发布自己的作品。（以前是免费的，现在不知道政策改了没有，呵呵！）

Python 的语法简洁，功能强大，有大量的第三方开发包（模块），非常适合初学者上手。同时 Python 不像 java 一样对内存要求非常高，适合做一些经常性的任务方面的编程。

本教程也是一入门教程，没有介绍关于 python 面向对像化的编程。这样可以使初学者更容易上手，而不会感觉像 java 一样太庞大而无从下手。（其实 python 也很庞大，呵呵！）

下面我们就从安装 python 开始进入 python 的精彩世界。

1.1 安装 python

python 的版本很多，特别是 2008 年底推出 python3.0（又称 python3000）后，局面比较混乱。因为 python3.0 不向下兼容 2.x 版本，而 python3.0 刚出来时间不长，稳定还需要一段时间，可以再等等观察一下。

2.x 的最后一个版本是 2.6，但大量的第三包不停留在 2.5 阶段。所以可以选择 2.5 版本来进行学习。过一段时间再选择 2.6，或者直接跳过 2.6，升级到 3.0 上。

我们已 2.5.2 的安装为例。

在 windows 上的安装很简单，直接下载相应版本的安装包即可安装。（可以到老版本的页面上找一下，如果有问题可以给我发邮件 maple5218@163.com）

安装完成后，为了能直接在 dos 控制台中使用 python 来执行程序.需要在环境变量的 path 中加入 python 的安装路径。

1.2 安装 ide 环境__SPE

要编程嘛，一个好的开发环境是不可少的。python 的 ide 环境很多，有开源的，也有商业的。
推荐使用 SPE 来做为开发环境，因为 SPE 本身就是用 python 来写的，小巧又方便。

eclipse 中也有相关的开发插件，但需要繁琐的设置。同时 eclipse 是用 java 写的，要装 jdk 不说，运行起来比较占用内存，有点牛刀杀鸡的感觉。

SPE 的图形界面使用的是 wxPython,在使用 SPE 之前先安装 wxPython2.8 -win32-unicode-2.8.9.1-py25.exe，下一步下一步安装即可。

SPE 有绿色版本 SPE-0.8.4.c-wx2.6.1.0-no_setup.zip，直要解压就可以使用了，双击 SPE.py 或者在 dos 控制台执行

```
C:\Documents and Settings\Administrator> python 解压目录\SPE.py
```

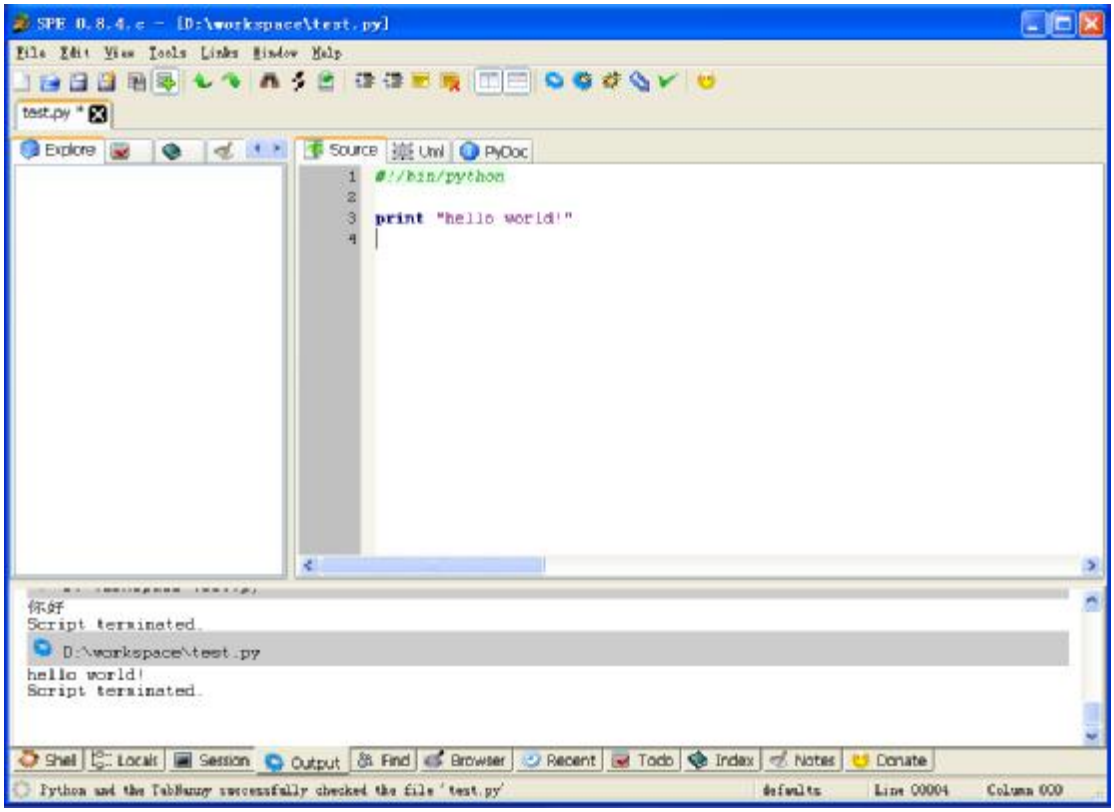
这时后面总会有一个黑色的 dos 窗口在后面，感觉很不舒服。
python 在 windows 上提供了一个没有 dos 窗口的程序叫 pythonw.exe，可以在桌

面上建一个快捷方式来方便启动。

快捷方式的目标是:

```
pythonw.exe 解压目录\SPE.py
```

启动后的界面如下：



万事具备我们要开始编程了...

2 基础编程

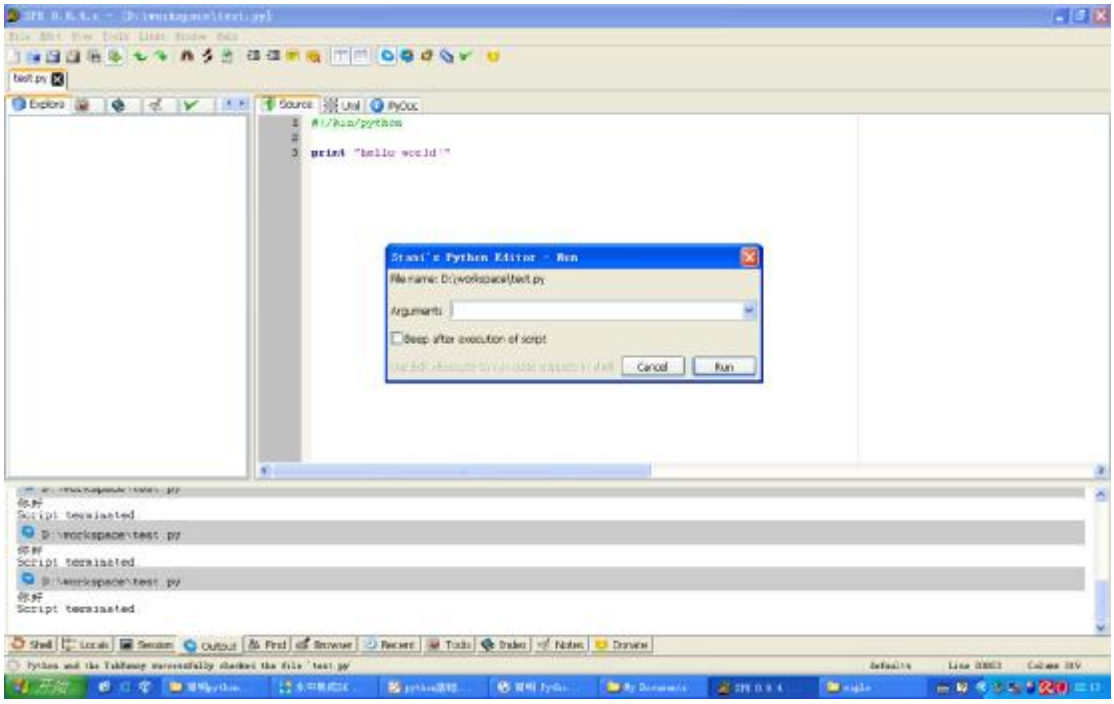
Python 的编程理念是那么的简单，我们先从第一个程序 hello world!开始，源文件以.py 做为扩展名。

test.py

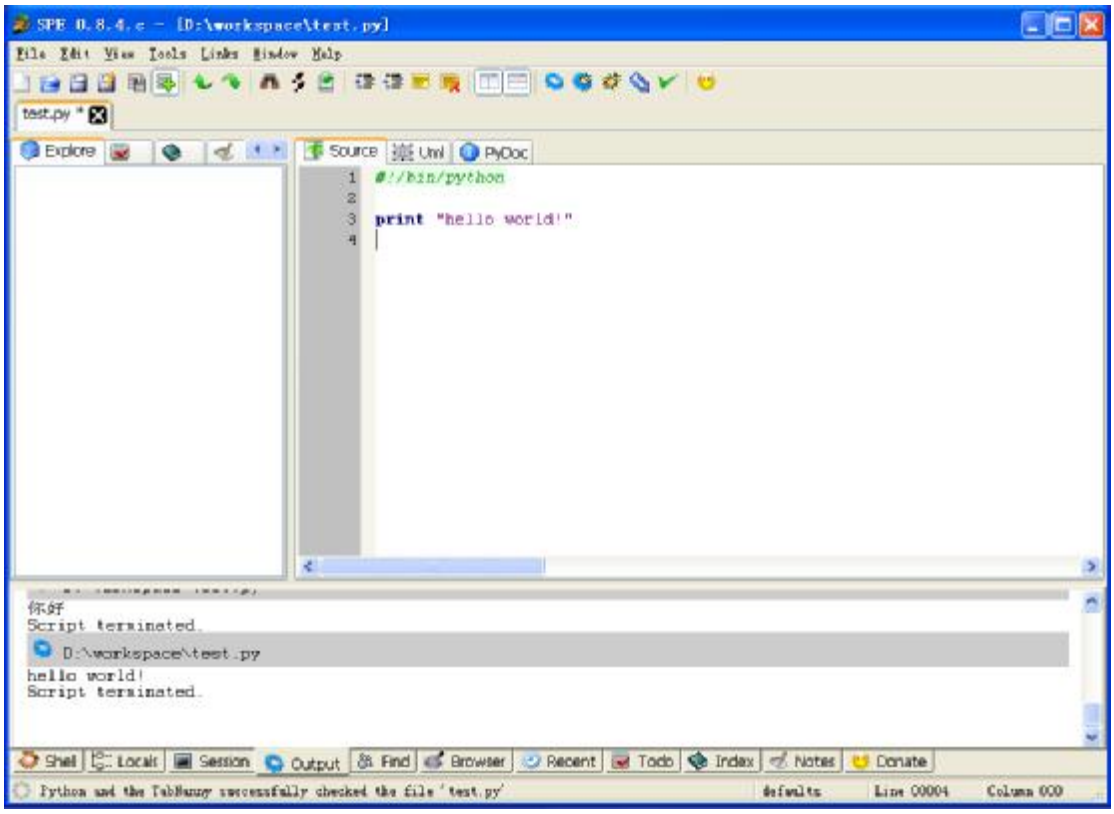
```
#!/bin/python
#coding=gb18030

print "hello word!"
```

编辑好后，用快捷键 F9 或点击小齿轮形状的图标，即可以弹出执行窗口，让你

输入入口参数来执行程序。



我们没有入口参数所以不写，直接点"Run"

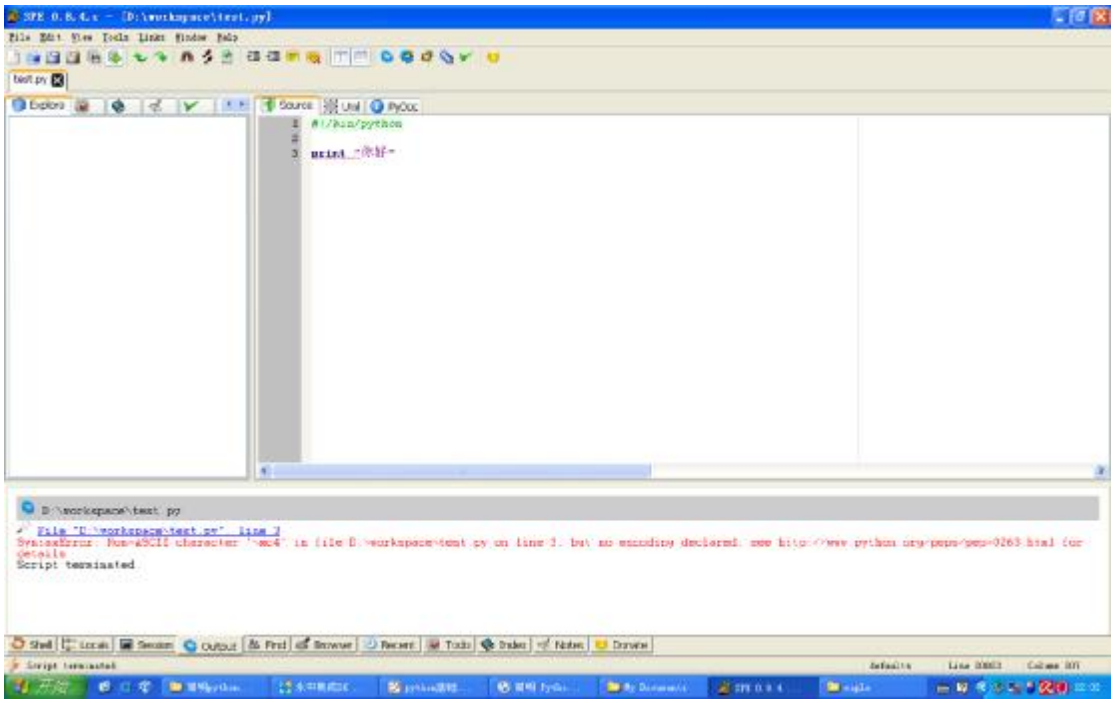


输出区得到输出结果 hello world!

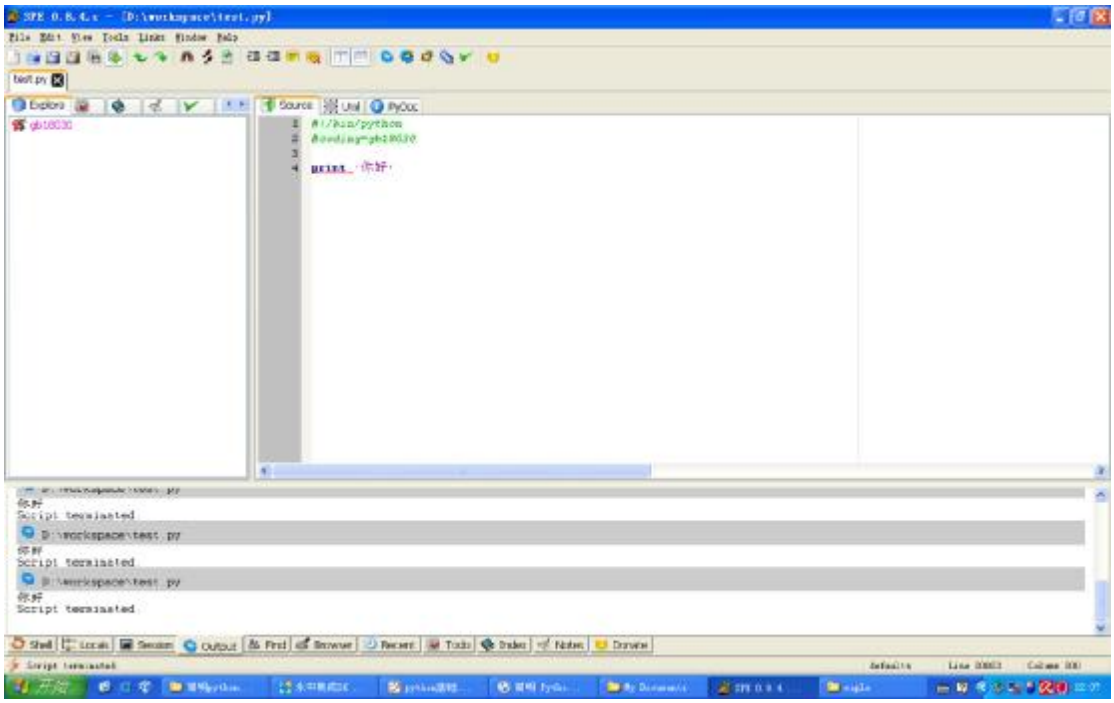
注解：

第 1 行的" #!/bin/pyton "是为了和 linux 的平台保持兼容。是用来告诉 shell 执行时，去用/bin/python 来解释执行。如我们在 linux 上写的 shell，都是以 #!/bin/sh 做为开头是一道理。这里我们为了兼容，因为我们的程序也有可台会跨平台执行，同时为了一开始编程就养成一个好的编程习惯。我们以后的 python 程序都装保持这个特色。
在有些教材中，也会将第一句写成 "#!/bin/env python",这样写的目的是因为有可能 python 没有装在/bin 目录下，使用环境变量来查找 python 的虚拟机在哪里。

第 2 行指明了编码方式，缺省是 utf-8 的。如果我们要输出中文或者采用中文的注释，就会出现下面的错误：



所以在处理中文时，我们都会加上编码，gb18030 或 gb2312.



注意： "#coding=gb18030",一定不能写成" #conding = gb18030",很多程序员会在=号前后加上空格，显得代码清晰，但在这里是行不能的。

第 3 行，很简单就是输出"hello world!",需要注意的是在 python2.x 版本中 print 是做为内部的关键词出现的，后面可以直接写要输出的东西。但在 python3000 中将 print 语句做成了函数要这样调用 print().

2.1 基本概念

2.1.1python 特色

1. 注释
python 的注释和其他语言如 java，c 都不太一样，而更像 shell 语言如 bash 的注释。

单行的注释是以#开始的，像我们刚开始讲的 test.py 都有用到。

```
#!/bin/python
#coding=gb18030

#相当于 main 函数
print "hello world! "   #output the hello world!
```

shell 中的 bash 是没有多行注释的，其实严格来讲 python 也没有多行注释，但 python 中有一个很好玩的东西，在字符串中会讲到，我们在这先提一下，哪就是三引号字符串。
三引号字符串在 python 中是用来表示多行的字符串的，在三引号字符串中你可以任意使用其他字符，包括表示字符串的单引号（'）和双引号（"）,所以三引号用来写多行注释，再合适不过了。

三引号可以是（'''）也可以是（"""）,他们完全等同。

```
#!/bin/python
#coding=gb18030

'''

name: test.py
email: maple5218@163.com
title: 这是我们的第一个程序，用来打印"hello world!"
这也是学习所以语言的第一个程序，
呵呵
'''
#相当于 main 函数
print "hello world! "   #output the hello world!
```

2. 缩进
python 最有特色的地方就是他的缩进，而且是强制缩进。大家虽然都知道缩进可以保持很好的代码风格，但这种强制缩进的风格还是上很多程序员受不了。但不用担心，习惯了以后你会爱上他的（我刚开始也不太习惯，但熟悉以后真的很喜欢，使代码很有条理，什么时候看都感觉像新写的代码一样熟悉）。

示例：

```
#!/bin/python
#coding=gb18030

'''
name: test.py
email: maple5218@163.com

title: 这是我们的第一个程序，用来打印"hello world!"
这也是学习所以语言的第一个程序，
呵呵
'''



def hello():
    """将 hello world 做到一个函数里面去 """
    print "hello world! "   #output the hello world!


if __name__ =='__main__':
    '''相当于 main 函数，以后介绍 '''
    hello()
```

缩进用在函数定义，if ,for 等语句的子语句中。缩进结束，代表这个逻辑段结束，如函数结束，if 语句结束等等。

不要混合使用制表符(TAB)和空格(Space)来缩进。

我*建议*使用 *单个制表符来进行缩进*。

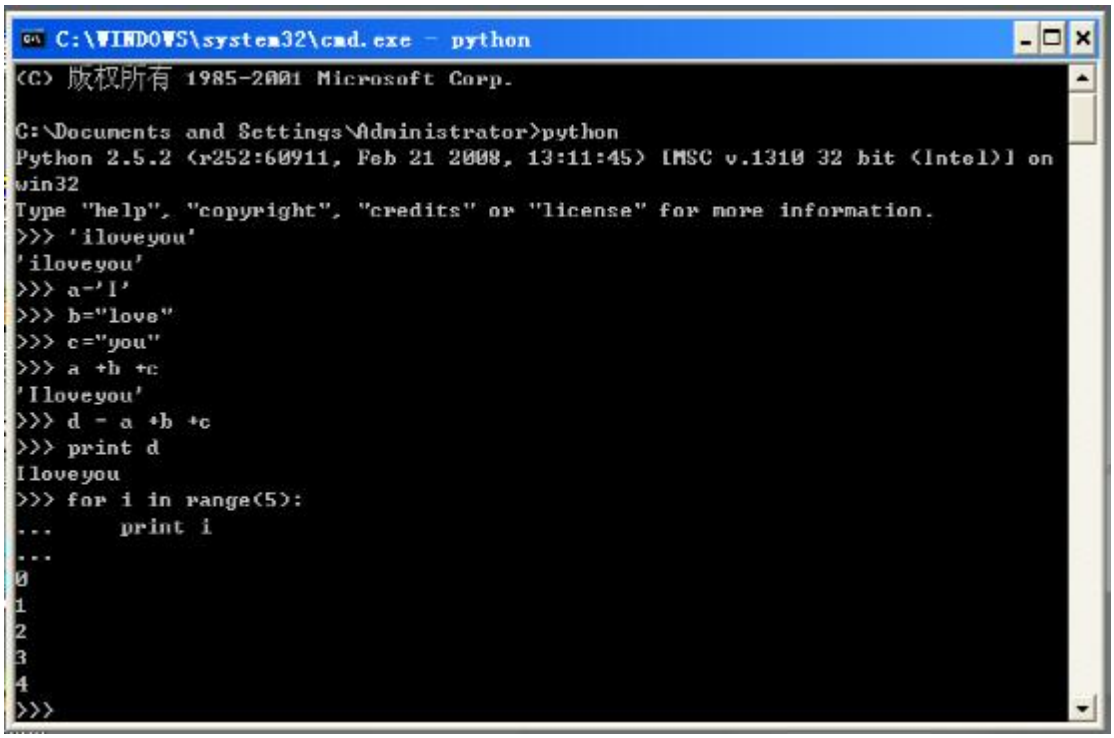
3. 字符串

python 中可以使用单引号（'）和双引号（"）来表示字符串，这两者是完全等同的。

```
a="xiaoming 's book"
b='I have a "stat war"'
print a
print b
```

4. 交互 shell

python 和 java,c#的不同之处，还有于 python 提供了一个高级的交互式 shell，你可以不写程序直接以里面进行计算和程序的验证。



2.1.2变量、运算符与表达式

1. 变量

python 中的变量不需要声明类型的，你直接使用=号赋值就可以了。

```
a = 1
b = 3.1415
c = "I love you"
```

变量可以是数字，字符串，布尔值（True,Flase,注意大小写），列表，字典等类型。

2. 运算符与它们的用法

运算符	名称	说明	例子
+	加	两个对象相加	3 + 5 得到 8。'a' + 'b' 得到'ab'。
-	减	得到负数或是一个数减去另一个数	-5.2 得到一个负数。50 - 24 得到 26。

*	乘	两个数相乘或是返回一个被重复若干次的字符串	2 * 3 得到6。'la' * 3 得到'lalala'。
**	幂	返回 x 的 y 次幂	3 ** 4 得到 81（即 3 * 3 * 3 * 3）
/	除	x 除以 y	4/3 得到 1（整数的除法得到整数结果）。4.0/3 或 4/3.0 得到 1.3333333333333333
//	取整除	返回商的整数部分	4 // 3.0 得到 1.0
%	取模	返回除法的余数	8%3 得到 2。-25.5%2.25 得到 1.5
<<	左移	把一个数的比特向左移一定数目（每个数在内存中都表示为比特或二进制数字，即 0 和 1）	2 << 2 得到 8。——2 按比特表示为 10
>>	右移	把一个数的比特向右移一定数目	11 >> 1 得到 5。——11 按比特表示为 1011，向右移动 1 比特后得到 101，即十进制的 5。
&	按位与	数的按位与	5 & 3 得到 1。
|	按位或	数的按位或	5 | 3 得到 7。
^	按位异或	数的按位异或	5 ^ 3 得到 6
~	按位翻转	x 的按位翻转是-(x+1)	~5 得到 6。
<	小于	返回 x 是否小于 y。所有比较运算符返回 1 表示真，返回 0 表示假。这分别与特殊的变量 True 和 False 等价。注意，这些变量名的大写。	5 < 3 返回 0（即 False）而 3 < 5 返回 1（即 True）。比较可以被任意连接:3 < 5 < 7 返回 True。
>	大于	返回 x 是否大于 y	5 > 3 返回 True。如果两个操作数都是数字，它们首先被转换为一个共同的类型。否则，它总是返回 False。
<=	小于等于	返回 x 是否小于等于 y	x = 3; y = 6; x <= y 返回 True。
>=	大于等于	返回 x 是否大于等于 y	x = 4; y = 3; x >= y 返回 True。

==	等于	比较对象是否相等	x = 2; y = 2; x == y 返回 True。x = 'str'; y = 'stR'; x == y 返回 False。x = 'str'; y = 'str'; x == y 返回 True。
!=	不等于	比较两个对象是否不相等	x = 2; y = 3; x != y 返回 True。
not	布尔“非”	如果 x 为 True，返回 False。如果 x 为 False，它返回 True。	x = True; not y 返回 False。
and	布尔“与”	如果 x 为 False，x and y 返回 False，否则它返回 y 的计算值。	x = False; y = True; x and y，由于 x 是 False，返回 False。在这里，Python 不会计算 y，因为它知道这个表达式的值肯定是 False（因为 x 是 False）。这个现象称为短路计算。
or	布尔“或”	如果 x 是 True，它返回 True，否则它返回 y 的计算值。	x = True; y = False; x or y 返回 True。短路计算在这里也适用。

优先级,由低到高。

运算符	描述
lambda	Lambda 表达式
or	布尔“或”
and	布尔“与”
not x	布尔“非”
in，not in	成员测试
is，is not	同一性测试
<，<=，>，>=，!=，==	比较

|	按位或
^	按位异或
&	按位与
<<, >>	移位
+, -	加法与减法
*, /, %	乘法、除法与取余
+x, -x	正负号
~x	按位翻转
**	指数
x.attribute	属性参考
x[index]	下标
x[index:index]	寻址段
f(arguments...)	函数调用
(experession,...)	绑定或元组显示
[expression,...]	列表显示
{key:datum,...}	字典显示
'expression,...'	字符串转换

计算顺序是从左到右的，()可以改变计算顺序。

3. 表达式
表达式就是将由变量，运算符等结合起来用来表达一种含义的式子。
如下：

```
(a+b)/2  #求平均值
a+b+c    #求和
print 'a+b=', a+b
```

2.2 流程控制

2.2.1顺序执行

python 的程序缺省是顺序执行的，而不像 c 和 java 一样需要一个 main()的入口

函数才能执行。这是脚本语言的特点。

但是遇到函数定义是不会执行的，只到函数调用时才执行。同时函数定义通常是放到调用前面的。注意这里的函数定义和 c 中的声明是不一样的，c 的函数是先一个声明，再有个实现。而 python 是放在一起的，所以我们直接叫他定义。在函数章节里我们再详述。

2.2.2条件执行 if...else....:

条件执行满足条件才执行，使用 if.... else....:语句。

```
love = True

if love== True:
   print "ilove you"
else:
   print "I not love you"
```

有两点要注意：
1 语句后面有:号
2 子句要缩进

还可以 elif 分支

```
if ......:
   print "ilove you"
elif .......:
   print "................................."
else:
   print "I not love you"
```

可以嵌套。

2.2.3循环执行 for... in...:

循环执行就是满足条件重复执行。使用最多的是 for .....in ...:

```
for i in range(5):
    print i
```

结果：

```
0
1
2
3
4
```

注意：
这里使用了一个内置函数 range().
range(起始值，终止值，步长)
起始值缺省是 0,可以不写
步长值缺省是 1，可以不写
终止值是采用小于，而不是小于等于，所以是满足条件的是不包括终止值。

```
>>>range(5)
[0,1,2,3,4]
>>> range(2,5)
[2,3,4]
>>>range(1,10,2)
[1,3,5,7,9]
>>>range(10,5,-1)
[10,9,8,7,6]
```

循环还有 while ...:语句，用得不多，不再详述。

2.3 函数

函数是面向过程编程的重要组成部份，要没有函数，代码的可读性和可重用性上都不会很好。
函数就是将一段逻辑相对独立，功能相对单一，又会在多处使用的代码，写成一个函数，再需要时进行调用，而不需要在每个地方都写一大堆重复代码。

2.3.1 自定义函数

格式：

```
def 函数名（参数，参数）:
    实现语句
    实现语句
```

    实现语句
    retrun ...

最后缩进结束，代表函数结束。

```
def sayhello(count):
    '''
    函数名：syahello
    参数： count
    '''
    for index in range(count):
        print "hello!"
# end fun sayhello

sayhello(5) #调用函数
```

注意：
1 参数，和 return 语句都不是必须的
2 缩进需要注意
3 局部变量
函数内部使用的变量，不影响外部的变量，即使变量同名。
4 全局变量
全局变量就是在所以函数外部定义的变量。函数内部可以使用全局变量，但要使用 global 声明。

```
id=1000

def addandprintId():
    '''
    函数名：addandprintId
    '''
    global id
    print id
    id = id +1



addandprintId() #调用函数 1000
addandprintId() #调用函数 1001
addandprintId() #调用函数 1002
```

2.3.2常用内置函数

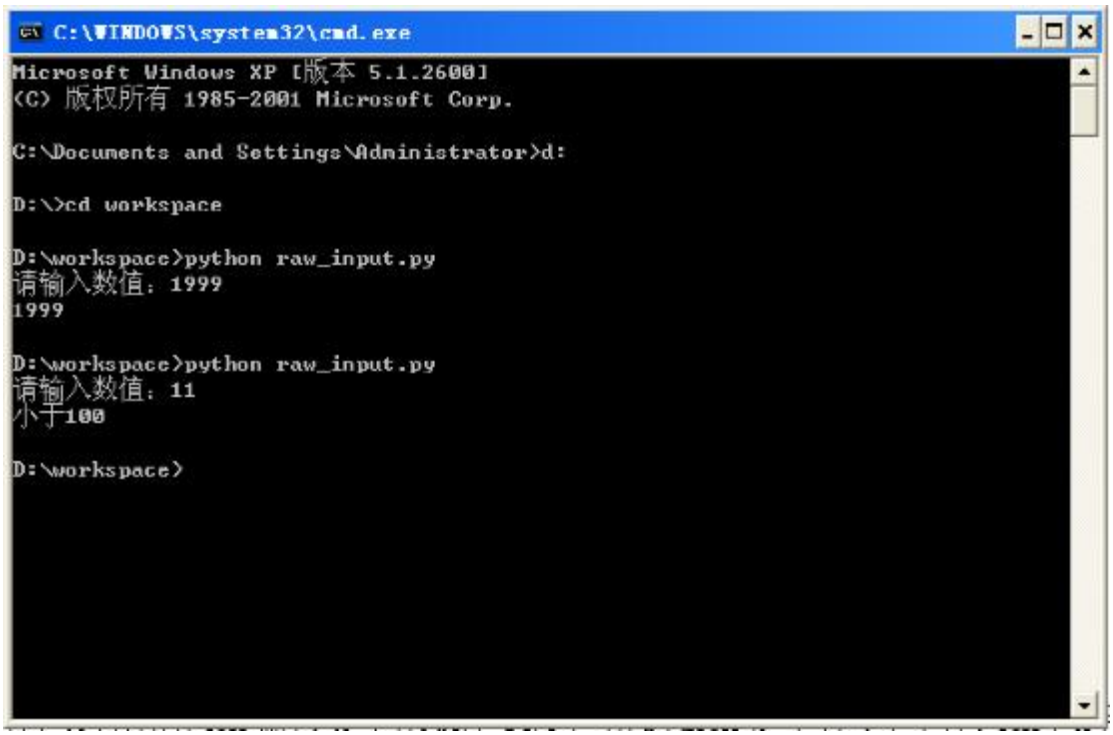
rawinput([prompt])

用来让用户输入数据。prompt 是提示，可以不要。
注意：SPE 的 ide 里没有提供输入的界面，所以要有这个功能要在 dos 控制台中执行

int()
用来返回整数，可以将 string 类型转化成 int 类型。

```
#!/bin/python
#coding=gb18030

a = raw_input("请输入数值：")

if int(a) >100:
    print a
else:
    print "小于 100"
```

执行结果：



len([string])
用来计算字符串，列表等对像的长度。

```
>>> len('iloveyou')
8
>>>len = 8
>>>len('iloveyou')
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'int' object is not callable
```

注意：很多程序员喜欢用 len 做为变量来表示长度，在 python 是不行了，因为 len 已经是一个内置的函数了。

range() 详见上文

str()
将对像转化成字符串

```
>>> str(1000)
'1000'
```

2.4 容器

容器就是用来存放一些对像的数据结构。根据使用方式的不同，python 提供了列表（List）,元组（trupe）,字典（dict）。

下面分别介绍：

2.4.1列表

用来动态的存放对像，可以插入，删除，更新。使用[ ]来给表示，看上去像 c 的数组，用起来像 java 的 list(不是很准确，更像 vector,不做过多解释).

定义： 变量 a = [ ] #空的列表
或者 变量 b =[1,3,4,5,'iloveyou'] #有值的列表，元素可以是不同的类型，这和 c，java 都不一样。

添加元素：a.append(元素) #添加元素到最后

插入元素：a.insert(位置，元素) #在固定位置，插入元素，原来后面的元素往后排，如果位置大于列表当然的长度，则后 append 的结果是一样的，添加到最后。列表是从 0 开始计数的，即第 1 个元素的索引是 0.

删除元素：a.remove(元素) #从 0 开始索引，直到匹配到相等的第一个元素，删掉该元素。只会删除一个。

索引元素：a.index(元素，开始，结束) #返回匹配到的第一个元素的索引。

引用元素：a[索引]

遍历列表：

```
>>>a=[1,2,3,4,5]
>>>for v in a:
...    print v
...
1
2
3
4
5
```

例子如下：

```
>>> a=[]
>>> a.append(10)
>>>a.insert(0,9)
>>>a
[9,10]
>>>a.append(10)
>>>a.append(8)
>>>a.index(10)
1
>>>a.remove(10)
>>>a
[9,10,8]
```

2.4.2元组

元组和列表很相似，唯一的区别是元组定义好之后，就不能再改动了。所以元组简单很多。

定义：
变量 a = (元素 1，元素 2，元素 3...)

```
>>>a=(1,2,'iloveyou')
>>>a
(1,2,'iloveyou')
>>>a[1]
2
>>>a[2]
'iloveou'
```

元组与字符串

```
>>>'li li li %s,%d'%('my',12)
'li li li my,12'
```

2.4.3字典

字典就像是 java 中的 map,根据一个键来对应一个对像。
定义：
变量 a={}
变量 a={key1:value1,key2:value2}

```
#!/usr/bin/python

# 'ab' is short for 'a'ddress'b'ook

ab = {       'Swaroop'   : 'swaroopch@byteofpython.info',
             'Larry'     : 'larry@wall.org',
             'Matsumoto' : 'matz@ruby-lang.org',
             'Spammer'   : 'spammer@hotmail.com'
     }

print "Swaroop's address is %s" % ab['Swaroop']

# Adding a key/value pair
ab['Guido'] = 'guido@python.org'

# Deleting a key/value pair
del ab['Spammer']

print '\nThere are %d contacts in the address-book\n' %
len(ab)
for name, address in ab.items():
    print 'Contact %s at %s' % (name, address)

if 'Guido' in ab: # OR ab.has_key('Guido')
    print "\nGuido's address is %s" % ab['Guido']
```

2.4.4序列

列表、元组和字符串都是序列，但是序列是什么，它们为什么如此特别呢？序列的两个主要特点是**索引**操作符和**切片**操作符。索引操作符让我们可以从序列中抓取一个特定项目。切片操作符让我们能够获取序列的一个切片，即一部分序列。

```
>>>a=’ Iloveyou’
>>>a[0]
’I’
>>>a[0:1] #后面的哪个不包括本身
’I’
>>>a[0:2]
’Il’
>>>a[5:]
’you’
>>>a[:-1]
’Iloveyo’
>>>
```

2.5 模块

2.5.1概念

模块是现代程序组织中所不可少的，每一个.py 文件就是一个模块。每个一模块实现相对独立功能，当需要使用这个模块时只要通过 import 指令导入即可，而不需要都去造一个相同"轮子"。

import 模块名
使用模块提供的对像时需要加上模块名
（python 所有东西都是对象，如变量，函数等等）

```
#!/usr/bin/python

import sys

print 'The command line arguments are:'
for i in sys.argv:
  print i

print '\n\nThe PYTHONPATH is', sys.path, '\n'
```

输出结果：

```
The command line arguments are:
argv.py


The PYTHONPATH is ['D:\\workspace',
'C:\\WINDOWS\\system32\\python25.zip', 'C:\\
Python25\\DLLs', 'C:\\Python25\\lib',
'C:\\Python25\\lib\\plat-win', 'C:\\Python
25\\lib\\lib-tk', 'C:\\Python25',
'C:\\Python25\\lib\\site-packages', 'C:\\Pytho
n25\\lib\\site-packages\\wx-2.8-msw-unicode']
```

sys.path 就是 python 寻找加载模块的路径。

注意：自己写的模块(.py 文件)一定不要与系统自带的模块重名，除非你想替代系统模块。因为你的运行路径放在了第一的位置。

如上图的 D:\\worspace

from 模块名 import *或 from 模块名 import 对象
可以直接使用模块提供的对象。

使用 dir(模块名)可以查看，模块提供的对象。

2.5.2 常用的标准模块

sys

- sys.argv 程序的入口参数，是一个列表
- sys.path 装载模块的搜索路径，是一个列表
-
- sys.version python 的版本，是一个 string
- sys.exit(status) 退出程序 ,是一个函数

注意：自己写的模块(.py 文件)一定不要与系统自带的模块重名，除非你想替代系统模块。因为你的运行路径放在了第一的位置。

time

- time.sleep(n) 休息 n 秒，可以是小数
- time.time() 返回一个浮点数，从 1970-1-1，0：0：0 到当前绝对时间的秒数，还有 8 位的小数
- time.localtime(second) 返回一个元组,如果没有 second,就使用 time.time()返回的秒，

  (2009, 8, 2, 20, 40, 3, 6, 214, 0)

Index	Attribute	Values
0	tm_year	(for example, 1993)
1	tm_mon	range [1,12]
2	tm_mday	range [1,31]
3	tm_hour	range [0,23]
4	tm_min	range [0,59]
5	tm_sec	range [0,61]; see (1)in strftime()description
6	tm_wday	range [0,6], Monday is 0
7	tm_yday	range [1,366]
8	tm_isdst	0, 1 or -1; see below

- time.strftime(format) 格式：

  time.strftime('%Y-%m-%d %H:%M:%d')

  '2009-08-02 20:50:02'

os

- os.name 字符串指示你正在使用的平台。比如对于 Windows，它是'nt'，而对于 Linux/Unix 用户，它是'posix'。
- os.getcwd()函数得到当前工作目录，即当前 Python 脚本工作的目录路径。
- os.getenv()和 os.putenv()函数分别用来读取和设置环境变量。
- os.listdir()返回指定目录下的所有文件和目录名。
- os.remove()函数用来删除一个文件。
- os.system()函数用来运行 shell 命令。

md5

- md5.new(arg) arg 要 md5 的内容，返回一个 md5 对象
- digest(),摘要，返回 16 个字节
- hexdigest(),16 进制摘要，返回 32 个字节

```
#!/bin/python
#coding=gb18030
#name: md5_test.py

import md5

m = md5.new("iloveou")
print m.digest()
print m.hexdigest()
```

结果：

```
D:\workspace>python md5_test.py
 Mb0*醱 i`-?蘋
fb10a7ae624f2ae15e69601edb10cc71
```

更多的模块，可以参考 python 自带的 doc\python.chm 的章节 library reference。

学习模块的最好的方法就是进入到 python 的交互 shell 中，然后尝试，下面看看 random 这个模块

```
>>>import random
>>> random.random()        # Random float x, 0.0 <= x < 1.0
0.37444887175646646
>>> random.uniform(1, 10)  # Random float x, 1.0 <= x < 10.0
1.1800146073117523
>>> random.randint(1, 10)  # Integer from 1 to 10, endpoints
included
7
>>> random.randrange(0, 101, 2)  # Even integer from 0 to 100
26
>>> random.choice('abcdefghij')  # Choose a random element
'c'

>>> items = [1, 2, 3, 4, 5, 6, 7]
>>> random.shuffle(items)
>>> items
[7, 3, 2, 5, 6, 4, 1]

>>> random.sample([1, 2, 3, 4, 5],  3)  # Choose 3 elements
[4, 1, 5]
```

2.6 文件操作

基础编程讲完了,我们以一个实战的项目来做个结尾,来做一个训练.

文件操作可能是我们遇到最多,最经常的操作.我们下面介绍一下文件操作的对像.

首页我们来介绍一下对像.编程世界的对像与现实世界比较相似.比如说现实世界的车就是一个对像.车有宽,高,长等属性,车可以前进,倒车等操作或功能.对应到编程世界,对像就是具有一组属性和一组方法(操作或功能)的一个集合体.

属性可以理解为变量,方法可以理解为函数.

文件操作可以使用内置的函数 open 来进行。

open(文件名[,mode]) 返回一个文件对像。

mode:

"r",读模式
"w", 写模式
"r+", 读写模式

文件对像的方法:
file.read(size) ：读取文件的 size 个字节.返回一个 string 对象.如果没有设置 size,则读取整个文件。

file.readline():读取一行，返回一个 string 对象，如果返回的内容为空，则说明文件结束 Eof。

file.readlines(): 读取所有的行，返回一个 list

file.write(buffer)：写 buffer 的内容到文件.
file.flush() 强制写缓冲区的内容到文件

file.seek(offset[,flag]): 定位文件读写光标。
    offset 偏移量
    flag 标志： os.SEEK_SET or 0 从文件头算起，缺省的方式。
                os.SEEK_CUR or 1 从当前位置算起
                os.SEEK_END or 2 从结束处算起

file.close() 关闭文件

文件对象的属性:
file.name:文件名
file.mode:文件打开的模式

作业：
1 输出一个文本文件内容的程序。（有多种方法，我给出一种）

```
#!/bin/python
#coding=gb18030


file = open('c:\\http.txt')  #相应的文件

for line in file.readlines():
    print line

file.close()
```

SPE 因为编码的原因可能会执行报错,可以在 dos 控制台中执行.

2 建立 1000 个文件，文件名是 1.txt,2.txt 以此类推。

3 建立 100 个文件，内容是随机的，以 md5 的值，来为文件起名。

3 数据库编程

3.1 DB-API 2.0 规范

python 的 DB-API 和 java 的 jdbc 类似,都是一个规范.定义了 python 中如何操作数据库的数据,然后由不同的数据库去实现不同的模块(jdbc 里叫驱动).

参考规范 http://www.python.org/dev/peps/pep-0249/对 DB_API2.0 做个介绍.

3.1.1模块接口 connect()方法.

所有符合 DB-API 2.0 规范的模块都提供了 connect()方法,用来连接数据库,返回一个数据库连接对象 Connection.

connect(参数 1,参数 2,..)

3.1.2Connection 对象

- .close()
  关闭与数据库的连接,如果有数据没有提交,这部分数据就撤销掉.

- .commit()
  提交该连接的所有没有提交的事务.
  如果数据库支持自动提交,哪么最初应该是关掉这个功能的.
  数据库模块可以提供方法来打开自动提交功能.

- .rollback()
  撤销该连接的事务.

- .cursor()
  返回一个数据库游标,用来执行具体的数据库操作.

3.1.3Cursor 对象

- .description
  只读属性,用来描述游标.

  name,
  type_code,
  display_size,
  internal_size,
  precision,
  scale,
  null_ok

- .close()
  关闭游标

- .execute(sql[,参数])
  预编译或执行一个 sql(查询,或 dml,ddl).如果是预编译,需要提供一个序列(列表,元组)或字典去绑定 sql 中的变量

execue(' insert into a (id,name) values(:1,:2)' ,(12,' lili))

如果想一条语句执行多条记录,则需要使用下面的 executemany(),即传说中的数组绑定.

- .executemany(sql,obj)

预编译一个 sql(dml),并且执行它.它关联的所有参数序列或字典在序列 obj 中.有点拗口.举个例子:

execue(' insert into a (id,name) values(:1,:2)' ,[(12,' lili),(13,' lixiaomei)])

在 oracle 中效率非常高,简单数据可以达到 5 万条/s

- .fetchone()
  取下一行记录.如是为 Nnne 则表示记录集取完了.

- .fetchall()

取所有的记录.

- .arraysize
  可读写属性.
  用来设置 fecthmary()取的记录行数,缺省为 1

- .fetchmany(size)
  读取 size 条的记录,如果没有 size,就以.arraysize 的设置为准.如果有 size,.arraysize 就等于 size.

- .setinputsizes(sizes)
  设置输入大小 ,用来处理 blog 等大字段.用在 execute 或 executemary 前面.

- .setoutputsize(size[,column])
  设置输出大小 ,用来处理 blog 等大字段.用在 execute 或 executemary 前面.

3.2 oracle(cx_Oracle)

3.2.1 安装

cx_Oralce 是 pyton 的 oracle 模块,使用前需要安装 oracle 的 instantclient.

安装:
instantclient 推荐使用 10.2,可以使用字符串来连接数据库,不需要建立 dsn.
在 windows 上直接解压,然后将目录加入到环境变量的 path 中即可.

cx_Oracle 最新版本(4.4.1).
windows 上安装非常方便,只要下一步下一步即可.安装完成后可以使用 ptyhon 的 shell 来验证一下.

```
>>>import cx_Oracle
>>>
```

没有报错,就说明安装没有问题.如果提示找不到 ocixxx.dll,instantclient 的环境变量的 path,没有设置正确.

linux 上一般采用源码安装,编译时需要 oci 的开发包(就是头文件),可以从 www.oracle.com 下载.

3.2.2连接数据库

连接 10g 或 9i:
可以使用连接字符串

用户名/密码@主机 ip:port/实例名

```
>>>import cx_Oracle
>>>conn = cx_Oracle.connect('gjw/gjw@172.168.112.50:1521/ORCL')
>>>
```

注意:如果 python 的机器上刚好安装了 9i 客户端或 9i 的服务器,这时候 10g 的 instantclient 就起作用了.需要建立 dsn,用 dsn 来访问.

用户名/密码@dsn

```
>>>import cx_Oracle
>>>conn = cx_Oracle.connect('gjw/gjw@ORCL_172.168.112.50' )
>>>
```

3.2.3直接 sql

直接 sql,就是采用 sql 语句组装的方法,不采用预编译和变量绑定.

建表示例:

```
connOra = cx_Oracle.connect(g_connStr)
    try:
        cursorOra = connOra.cursor()
        cursorOra.execute('''
        CREATE TABLE TEST
           (
           ID NUMBER(10, 0),
           NAME VARCHAR2(50),
           TIME TIMESTAMP,
           VALUE NUMBER(10, 3),
      primary key (ID)
           )
        ''')
    except cx_Oracle.DatabaseError,e:
        error, = e.args
        print error.message
    finally:
        cursorOra.close()
        connOra.close()
```

插入示例:

```
    currentStr = time.strftime("%Y-%m-%d %H:%M:%S")
    connOra = cx_Oracle.connect(g_connStr)
    begin = time.time()

    cursorOra = connOra.cursor()
    for i in range(1, total+1):
        sqlStr = "INSERT INTO TEST (ID, NAME, TIME,VALUE) VALUES
(%d, '%s', to_date('%s','yyyy-MM-dd HH24:mi:ss'), %s)"%(genId(),
genString(50), currentStr, genDouble())
        cursorOra.execute(sqlStr)
    connOra.commit()

    end = time.time()
    print "sql Count:%s Time:%s"%(total, end - begin)
    cursorOra.close()
    connOra.close()
```

3.2.4预编译

预编译就是采用变量绑定的方式来处理数据,主要是可以让数据库缓存 sql 来提高性能.

同时每次发送的数据里不用在包含 sql 语句,只有数据就可以了,但数据库还是会对每个数据做一个响应的.这个与下面的数据绑定又不尽相同.

```
    cst = time.localtime()
    current = datetime.datetime(cst[0], cst[1], cst[2], cst[3],
cst[4], cst[5])
    connOra = cx_Oracle.connect(g_connStr)
    cursorOra = connOra.cursor()
    #cursorOra.prepare()
    for i in range(1, total+1):
        cursorOra.execute("INSERT INTO TEST (ID, NAME, TIME,
VALUE) VALUES (:1, :2, :3, :4)", (genId(), genString(0), current,
genDouble()))
    connOra.commit()

    cursorOra.close()
    connOra.close()
```

或者

```
    cst = time.localtime()
    current = datetime.datetime(cst[0], cst[1], cst[2], cst[3],
cst[4], cst[5])
    connOra = cx_Oracle.connect(g_connStr)

    cursorOra = connOra.cursor()
    #cursorOra.prepare("INSERT INTO TEST (ID, NAME, TIME, VALUE)
VALUES (:1, :2, :3, :4)")
    for i in range(1, total+1):
        cursorOra.execute(None, (genId(), genString(0), current,
genDouble()))
    connOra.commit()

    cursorOra.close()
    connOra.close()
```

3.2.5数组绑定

数据绑定也是基于变量绑定(预编译),但是数据绑定的特色将准备好的多条记录一起打包发给数据库,数据库只要对数据做一次响应即可.这样大大提高了入库的速度.

每秒钟可以达到数万条.

```
cst = time.localtime()
    current = datetime.datetime(cst[0], cst[1], cst[2], cst[3],
cst[4], cst[5])

    connOra = cx_Oracle.connect(g_connStr)
    begin = time.time()

    cursorOra = connOra.cursor()
    cursorOra.prepare("INSERT INTO TEST (ID, NAME, TIME, VALUE)
VALUES (:1, :2, :3, :4)")
    for i in range(1, 10000):  #一次 10000 条记录
        tempitems.append((genId(), genString(6), current,
genDouble()))
    cursorOra.executemany(None, tempitems)
    connOra.commit()

    end = time.time()
    print "array Count:%s Time:%s"%(total, end - begin)
    cursorOra.close()
    connOra.close()
```

3.2.6blob

插入 blog 前,需要设定.setinputsizes(),指明变量和类型.

'BFILE', 'BINARY', 'BLOB', 'Binary', 'CLOB','LOB', 'LONG_BINARY', 'LONG_STRING', 'NCLOB',

```
    cst = time.localtime()
    current = datetime.datetime(cst[0], cst[1], cst[2], cst[3],
cst[4], cst[5])

    connOra = cx_Oracle.connect(g_connStr)
    begin = time.time()

    try:
        cursorOra = connOra.cursor()
        #insert one
        logging.info("insert 6k table")
        bindvar= {}
        cursorOra.setinputsizes(value4 = cx_Oracle.LONG_BINARY)
        bindvar['value4']= g_6k
        cursorOra.prepare("INSERT INTO
blob_6k_"+str(businessNO)+" (ID,\
         value1,value2,value4,value3) VALUES (:id, :
value1,:value2, :value4, :value3)")
        for i in range(total+1):
            bindvar['id'] = genId()
            bindvar['value1'] = genASCII(5)
            bindvar['value2'] = current
            bindvar['value3'] = genDouble()
            cursorOra.execute(None, bindvar)

        #commit
        logging.info("commit")
        connOra.commit()
    except cx_Oracle.DatabaseError,e:
        error, = e.args
        #print e.args
        logging.info("insert failed " + error.message)
    finally:
        cursorOra.close()
        connOra.close()

    end = time.time()
    print "Count:%s Time:%s"%(total, end - begin)
```

3.2.7查询

```
    connOra = cx_Oracle.connect(g_connStr)
    begin = time.time()

    cursorOra = connOra.cursor()
    cursorOra.prepare("select count(id) from test")
    cursorOra.execute(None)
    total = cursorOra.fetchone()

    end = time.time()
    print "select Total Count:%s Time:%s"%(total, end - begin)
    print "-------------------------------------------------"
    cursorOra.close()
    connOra.close()
```

3.2.8例子

```
#!/bin/env python
#coding=gb18030


'''
test the date type in oralce database.

include date,timestamp type.



'''


import cx_Oracle
import time
import datetime
import os

os.environ["NLS_LANG"] = "SIMPLIFIED CHINESE_CHINA.ZHS16GBK"

conn_string="unimas/unimas@192.168.1.158/orcl"

def createTable():
```

```
    sql = "create table testdate (id number(10,0),test1 date,test2
timestamp)"
    conn = cx_Oracle.connect(conn_string)

    try:
        cursor = conn.cursor()

        cursor.execute(sql)

    except cx_Oracle.DatabaseError,e:
        error, = e.args
        print error.message
    finally:
        cursor.close();
        conn.close()

#end fun of createTable()

def insertRecordWithSqL(id):
    currentStr = time.strftime("%Y-%m-%d %H:%M:%S")

    sql ='''
        insert into testdate(id,test1,test2) values
            (
            %d,to_date('%s','yyyy-mm-dd'),
             to_timestamp('%s', 'yyyy-mm-dd hh24:mi:ss.ff')
             )
        '''%(id,currentStr[:10],currentStr)
    print sql
    conn = cx_Oracle.connect(conn_string)

    try:
        cursor = conn.cursor()

        cursor.execute(sql)
        conn.commit()
    except cx_Oracle.DatabaseError,e:
        error, = e.args
        print error.message
    finally:
        cursor.close();
        conn.close()

#end fun of insertRecordWithSqL()
```

```
def insertRecordWithPre(id):
    today = cx_Oracle.DATETIME.today() #datetime.datetime.today()
    timestamp = cx_Oracle.Timestamp.today()
    sql ='''
        insert into testdate(id,test1,test2) values(:1,:2,:3)
        '''
    print sql
    conn = cx_Oracle.connect(conn_string)

    try:
        cursor = conn.cursor()

        cursor.execute(sql,(id,today,timestamp))
        conn.commit()
    except cx_Oracle.DatabaseError,e:
        error, = e.args
        print error.message
    finally:
        cursor.close();
        conn.close()
#end fun of insertRecordWithPre()

def query():
    sql ='''
        select * from testdate order by test2 desc
        '''
    print sql
    conn = cx_Oracle.connect(conn_string)

    try:
        cursor = conn.cursor()

        cursor.execute(sql)
        result = cursor.fetchall()
        for row in result:
            print row
    except cx_Oracle.DatabaseError,e:
        error, = e.args
        print error.message
    finally:
        cursor.close();
        conn.close()
```

```
#end fun of query();

createTable()
insertRecordWithSqL(1)
insertRecordWithPre(2)
query()
```

结果:

```
        insert into testdate(id,test1,test2) values
            (
            1,to_date('2009-08-05','yyyy-mm-dd'),
             to_timestamp('2009-08-05 16:54:37', 'yyyy-mm-dd
hh24:mi:ss.ff')
             )


        insert into testdate(id,test1,test2) values(:1,:2,:3)


        select * from testdate order by test2 desc

(2, datetime.datetime(2009, 8, 5, 16, 54, 37),
datetime.datetime(2009, 8, 5, 16, 54, 37))
(1, datetime.datetime(2009, 8, 5, 0, 0), datetime.datetime(2009,
8, 5, 16, 54, 37))
```

分析:

1. oracle 的 date 类型可以存储到秒,timestamp 可以存储到 ms(6 位缺省),可以增加到 9 位.
2. python 中的 datetime.datetime 可以到 ms(7 位),但在入库过程中做了截断,所以只能看到秒.
3. to_date('2009-08-05','yyyy-mm-dd') 转化成日期
4. to_timestamp('2009-08-05 17:16:43.584015', 'yyyy-mm-dd hh24:mi:ss.ff') 转化成时间戳.

3.3 Mssql Server 编程

3.4 Mysql 编程