NSD Python2 DAY03

1 案例1：备份程序

1.1 问题

编写backup.py脚本，实现以下目标：

1. 需要支持完全和增量备份
2. 周一执行完全备份
3. 其他时间执行增量备份
4. 备份文件需要打包为tar文件并使用gzip格式压缩

1.2 方案

整体框架创建3个函数，分别实现完全备份、增量备份、文件加密3种功能：

1.首先导入time模块，利用if进行判断，如果当地时间是星期一，执行完全备份函数，否则执行增量备份函数，其中，通配符%a代表时间星期几缩写，上传参数分别为要备份的原目录、目标目录、md5字典存放目录

2.调用完全备份函数：

a)首先获取新文件名，将新文件名放入目标目录下，目的是定义备份文件的绝对路径，以写压缩方式打开目标目录下新文件，将原目录写入新文件中，完成完全备份，其中os.path.join作用是将目录名和文件的基名拼接成一个完整的路径

b)了解os.walk()目录遍历器输出文件结构，利用for循环将要备份原目录中文件遍历出来作为字典键值对键， md5加密结果作为字典键值对的值（此时将原目录中文件作为上传参数调用文件加密函数），存入空字典中，字典中每个文件对应一个md5值，最后将字典写入到md5字典存放目录中

3.调用文件加密函数：将原目录文件循环读取逐一加密，返回加密结果

4.调用增量备份函数：

a)增量备份函数代码与完全备份函数基本一致

b)区别在于，备份前要先以二进制读方式打开md5字典存放目录，读取旧数据，判断旧数据中键对应的加密值与新加密值是否相同，如果不相同，则将新增内容写入到目标文件中（即只备份新数据）

5.注意：md5主要用于原文件与新文件判断

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

Top

```
01. [root@localhost day06]# vim backup.py
```

```
02.  #! /usr/bin/env python3
03.
04.  import time
05.  import os
06.  import tarfile
07.  import hashlib
08.  import pickle
09.  #用于判断两个文件是否相同，提取每个文件中的前4字节的内容然后输出md5码进行比
10.  def check_md5(fname):
11.      m = hashlib.md5()
12.      with open(fname, 'rb') as fobj:
13.          while True:
14.              data = fobj.read(4096)
15.              if not data:
16.                  break
17.              m.update(data)
18.      return m.hexdigest()
19.
20.  def full_backup(src_dir, dst_dir, md5file):
21.      fname = os.path.basename(src_dir.rstrip('/'))
22.      fname = '%s_full_%s.tar.gz' %(fname, time.strftime('%Y%m%d'))
23.      fname = os.path.join(dst_dir, fname)
24.      md5dict = {}
25.
26.      tar = tarfile.open(fname, 'w:gz')
27.      tar.add(src_dir)
28.      tar.close()
29.
30.      for path, folders, files in os.walk(src_dir):
31.          for each_file in files:
32.              key = os.path.join(path, each_file)
33.              md5dict[key] = check_md5(key)
34.
35.      with open(md5file, 'wb') as fobj:
36.          pickle.dump(md5dict, fobj)
37.
38.
39.  def incr_backup(src_dir, dst_dir, md5file):
40.      fname = os.path.basename(src_dir.rstrip('/'))
41.      fname = '%s_incr_%s.tar.gz' %(fname, time.strftime('%Y%m%d'))
42.      fname = os.path.join(dst_dir, fname)
```

Top

```
    md5dict = {}

    with open(md5file, 'rb') as fobj:
        oldmd5 = pickle.load(fobj)

    for path, folders, files in os.walk(src_dir):
        for each_file in files:
            key = os.path.join(path, each_file)
            md5dict[key] = check_md5(key)

    with open(md5file, 'wb') as fobj:
        pickle.dump(md5dict, fobj)

    tar = tarfile.open(fname, 'w:gz')
    for key in md5dict:
        if oldmd5.get(key) != md5dict[key]:
            tar.add(key)
    tar.close()

if __name__ == '__main__':
    # mkdir /tmp/demo; cp -r /etc/security /tmp/demo
    src_dir = '/tmp/demo/security'
    dst_dir = '/var/tmp/backup'   # mkdir /var/tmp/backup
    md5file = '/var/tmp/backup/md5.data'
    if time.strftime('%a') == 'Mon':
        full_backup(src_dir, dst_dir, md5file)
    else:
        incr_backup(src_dir, dst_dir, md5file)
```

步骤二：测试脚本执行

```
[root@localhost day07]# python3 backup.py
[root@localhost day07]# cd /var/tmp/backup/
[root@localhost backup]# ls
md5.data  security_full_20180502.tar.gz  security_incr_20180502.tar.gz
```

2 案例2：编写酒店类

Top

2.1 问题

创建hotel.py脚本，要求如下：

1. 用于计算住宿开销
2. 酒店有会员卡可以打九折
3. 每天早餐15元
4. 根据住宿天数返加总费用

2.2 方案

创建一个酒店类，类中定义2种方法：

1.__init__方法：__init__方法用于初始化属性，创建对象后会自动调用__init__方法，属于构造器方法，此处初始化了房间、早餐以及折扣3个属性，并给出了默认参数，此处属性绑定在对象上，数据属性在每一个方法中都可以使用

2.定义开销方法：每天花费（花费房间*折扣+早餐费用）*天数，天数days作为参数绑定在函数中，属于局部变量，只能在函数中使用，通过传参上传不同的值，得到不同的结果

3.创建对象后，自动调用__init__方法，在调用开销方法计算花费

2.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

```
[root@localhost day07]# vim hotel.py

#! /usr/bin/env python3

class Hotel:
    def __init__(self, room=200, br=15, cf=1.0):
        self.room = room
        self.br = br
        self.cf = cf

    def cacl_all(self, days=1):
        return (self.room * self.cf + self.br) * days

if __name__ == '__main__':
    std_room = Hotel()
    print(std_room.cacl_all())
    print(std_room.cacl_all(2))
```

步骤二：测试脚本执行

Top

```
[root@localhost day07]# python3  hotel.py
```

```
02.     215.0
03.     430.0
```

3 案例3：出版商程序

3.1 问题

创建books.py文件，实现以下目标：

1. 为出版商编写一个Book类
2. Book类有书名、作者、页数等属性
3. 打印实例时，输出书名
4. 调用实例时，显示该书由哪个作者编写

3.2 方案

创建一个类，类中创建3种魔法方法：

1.__init__方法：__init__方法用于初始化实例属性，创建对象后会自动调用__init__方法，属于构造器方法，此处初始化了书名及作者两个属性

2.__str__方法：创建对象后，打印实例对象pybook，返回书名，打印出书名

3.__call__方法：创建对象后，可以像调用函数一样调用该方法，模拟函数的行为，打印出书名及作者

3.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

```
01.  [root@localhost day07]# vim books.py
02.  #!/usr/bin/env python3
03.
04.  class Book:
05.      def __init__(self, title, author):
06.          self.title = title
07.          self.author = author
08.
09.      def __str__(self):
10.          return '<Book: %s>' % self.title
11.
12.      def __call__(self):
13.          print('《%s》 is written by %s.' %(self.title, self.author))
14.
15.  if __name__ == '__main__':
16.      pybook = Book('Core Python', 'Weysley')
17.      print(pybook)  #调用__str__
```

Top

```
18.         pybook()    # 调用__call__
```

步骤二：测试脚本执行

```
01. [root@localhost day07]# python3 books.py
02. <Book: Core Python>
03. 《Core Python》 is written by Weysley.
```

Top