## 1.为什么需要序列化和反序列化？

1.**便于存储**。**序列化过程将文本信息转变为二进制数据流**。这样就信息就容易存储在硬盘之中，当需要读取文件的时候，从硬盘中读取数据，然后再将其反序列化便可以得到原始的数据。在Python程序运行中得到了一些字符串、列表、字典等数据，想要长久的保存下来，方便以后使用，而不是简单的放入内存中关机断电就丢失数据。python模块大全中的Pickle模块就派上用场了，它可以将对象转换为一种可以传输或存储的格式。

**2.便于传输**。当两个进程在进行远程通信时，彼此可以发送各种类型的数据。无论是何种类型的数据，都会以二进制序列的形式在网络上传送。发送方需要把這个对象转换为字节序列，在能在网络上传输；接收方则需要把字节序列在恢复为对象。

pickle模块有两类主要的接口，即序列化和反序列化。  
其中**序列化操**作包括：

pickle.dump()

Pickler(file, protocol).dump(obj)  
**反序列化操作**包括：

pickle.load()

Unpickler(file).load()

## 2.序列化操作

### 2.1序列化方法pickle.dump()

dump(obj, file, protocol=None),该方法实现的是将序列化的对象obj以二进制的形式写入文件file中，它的功能等同于 Pickler(file, protocol).dump(obj)。  
关于参数file，有一点需要注意，必须是以二进制的形式进行操作（写入）。

#!/usr/bin/env python

# -\*- coding:utf-8 -\*-

import pickle

F = open(*'datafile.pkl'*, *'wb'*)

D = dict.fromkeys([*'a'*, *'b'*], 10)

E = [1, 2, 3, 4]

pickle.dump(D, F) # 序列化第一个数据

pickle.dump(E, F) # 序列化第二个数据

F.close()

F = open(*'datafile.pkl'*, *'rb'*)

D1 = pickle.load(F) # 反序列化第一个数据

E1 = pickle.load(F) # 反序列化第二个数据，如果继续load(F),没有数据抛出EOFError

F.close()

print D1, E1 # {'a': 10, 'b': 10} [1, 2, 3, 4]

关于参数protocol，一共有5中不同的类型，即（0,1,2,3,4）。（0,1,2）对应的是python早期的版本，（3,4）则是在python3之后的版本。  
此外，参数可选 pickle.HIGHEST\_PROTOCOL和pickle.DEFAULT\_PROTOCOL。当前，python3.5版本中，pickle.HIGHEST\_PROTOCOL的值为4，pickle.DEFAULT\_PROTOCOL的值为3。当protocol参数为负数时，表示选择的参数是pickle.HIGHEST\_PROTOCOL。

### 2.2 序列化方法pickle.dumps()

dumps(obj, protocol=None),pickle.dumps()方法不需要写入文件中，它是直接返回一个序列化的bytes对象。

D = dict.fromkeys([*'a'*, *'b'*], 10)

E = [1, 2, 3, 4]

D\_S = pickle.dumps(D)

E\_S = pickle.dumps(E)

print *"D serializing:%s"* % D\_S

print *"E serializing:%s"* % E\_S

结果：

D serializing:(dp0

S'a'

p1

I10

sS'b'

p2

I10

s.

E serializing:(lp0

I1

aI2

aI3

aI4

a.

### 2.3 序列化方法Pickler(file, protocol).dump(obj)

pickle模块提供了序列化的面向对象的类方法，即 class pickle.Pickler(file, protocol=None,\*,fix\_imports=True),Pickler类有dump()方法。  
Pickler(file, protocol).dump(obj) 实现的功能跟 pickle.dump() 是一样的。

### 2.4 反序列化操作

load(file)，该方法实现的是将序列化的对象从文件file中读取出来。它的功能等同于 Unpickler(file).load()。  
关于参数file，有一点需要注意，必须是以二进制的形式进行操作（读取）。

### 2.5 反序列化方法pickle.loads()

loads(str)，pickle.loads()方法是直接从bytes对象中读取序列化的信息，而非从文件中读取。

D1 = pickle.loads(D\_S)

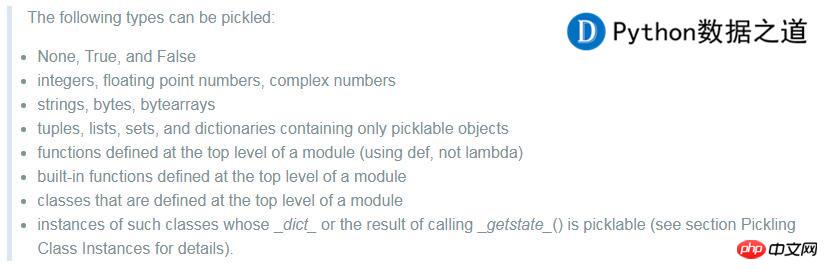
E1 = pickle.loads(E\_S)

print D1, E1 # {'a': 10, 'b': 10} [1, 2, 3, 4]

### 2.6 反序列化方法Unpickler(file).load()

pickle模块提供了反序列化的面向对象的类方法，即 class pickle.Unpickler(file, \*,fix\_imports=True, encoding="ASCII". errors="strict"),Pickler类有load()方法。  
Unpickler(file).load() 实现的功能跟 pickle.load() 是一样的。

## 3.那些类型可进行序列化



## 4.对象持久化

shelve模块提供一个额外的层结构，允许按照键来存储pickle处理后的对象。shelve使用pickle把一个对象转换为其pickle化的字符串，并将其存储在一个dbm文件中的一个键之下；随后载入的时候，shelve通过键获取pickle化的字符串，并用pickle在内存中重新创建最初的对象。

class **Person**:

def **\_\_init\_\_**(*self*, name, job=None, pay=0):

*self*.name = name

*self*.job = job

*self*.pay = pay

def **last\_name**(*self*):

return *self*.name.split()[-1]

def **give\_raise**(*self*, percent):

*self*.pay = int(*self*.pay \* (1+percent))

def **\_\_str\_\_**(*self*):

return *'[Person: %s %s]'* % (*self*.name, *self*.pay)

if \_\_name\_\_ == *"\_\_main\_\_"*:

bob = Person(*'Bob Smith'*)

sue = Person(*'Sue Jones'*, job=*'dev'*, pay=10000)

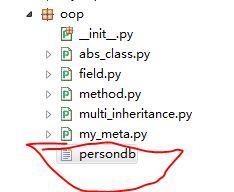
import shelve

db = shelve.open(*'persondb'*)

for obj in (bob, sue):

db[obj.name] = obj # 使用obj.name作为键

db.close()



### 4.1持久化方法shelve.open()

FUNCTIONS

open(filename, flag='c', protocol=None, writeback=False)

Open a persistent dictionary for reading and writing.

The filename parameter is the base filename for the underlying

database. As a side-effect, an extension may be added to the

filename and more than one file may be created. The optional flag

parameter has the same interpretation as the flag parameter of

anydbm.open(). The optional protocol parameter specifies the

version of the pickle protocol (0, 1, or 2).

See the module's \_\_doc\_\_ string for an overview of the interface.

参数：

打开一个持久化字典用于读写。

filename：文件名，作为底层数据库

protocol：可选的协议参数，（0,1,2）

### 4.2 持久化数据交互

读取数据：

db = shelve.open(*'persondb'*)

print len(db) # 2

print list(db.keys()) # ['Bob Smith', 'Sue Jones']

for key in db:

print key, *'=>'*, db[key]

db.close()

更新数据：

sue = db[*'Sue Jones']*

sue.give\_raise(.2)

db[*'Sue Jones'] = sue*

*db.close()*