Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 5.258

IJCSMC, Vol. 5, Issue. 4, April 2016, pg.444 – 448

Serialization and Deserialization of Python Objects using Pickle and cPickle Modules and their Performance Comparison

Rakshith C A[1], Varun Sharma N[2]
[1]Student, Department of CSE, RIT, Hassan- 573201, INDIA
[2]Student, Department of CSE, RIT, Hassan- 573201, INDIA
[1] rakshithca@gmail.com; [2] varun15sharma15@gmail.com

Abstract— Serialization is a process used to translate the state of an object or data structure into a format that can be stored in a file or buffered in memory or transmitted across a network and when de-serialized or reconstructed, creates an identical clone of object in the same or different computer environment. It converts python objects into sequence of binary data and vice versa and objects are said to be pickled or un-pickled accordingly. In this paper we have discussed how to serialize and de-serialize Python objects using Pickle module.

Keywords— Python objects, serialization, deserialization, Pickle, cPickle.

I. INTRODUCTION

Serialization as a concept of data persistence, allows the sequences of binary data to be stored in the form of binary files, thereby maintaining program's state data to be saved into disk and when read back again can carry on from where it left off when restarted. In the context of transmission across the network, in a multi core or distributed system, it can be used to send Python object sequenced data over a TCP connection. It can be used for storing Python objects in a database like MySQL or MongoDB.

In addition, as a part of memory caching, in order to obtain a dictionary key, any Python object can be converted into string data by using serialization. Python objects that can be converted include numeric, Boolean, string data types, collections like lists, tuples, dictionaries, functions, recursive objects in addition to system and user defined classes.

II. AIM OF STUDY

- To Know what is Serialization and de-serialization
- To Understand the Need of pickle module
- Knowledge on serialization of Python Objects using pickle and cPickle
- Performance comparison of pickle and cPickle modules

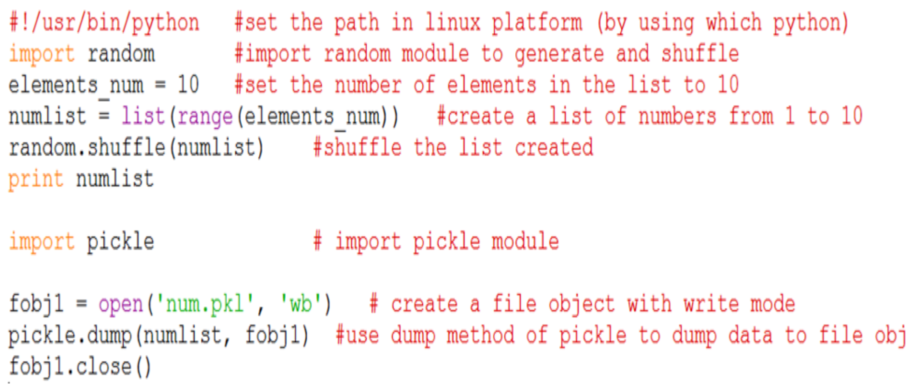
III.RELATED WORK

Serialization of Python object using pickle

In python, serialization of objects can be done by using a standard library module called pickle, which has an algorithm implemented to convert any Python object into a sequence of bytes. Installation of pickle: pickle comes as a default module in windows OS and Linux distributions. To check if pickle is available, on the python prompt, type import pickle command and there should not be any Traceback or errors. Use dir(pickle) command to check the huge number methods available in pickle module. The commonly used functions for serialization are dump for working with files and dumps for creating a string of an object. For de-serialization, load and loads are used accordingly to write into an open file or string back to object.

Dump and Load – working with files

Here is a sample code in Python to serialize a list object shown in Fig.1. Codes are tested on Python 2.7.6 on windows 7 and python 2.6.6 on centOS. This code creates a list of numbers with 10 elements and dump method of pickle is used to write pickled representation of number list into the num.pkl (.pkl extn is used for readability, but any file extension can be used) file opened for write mode. pickle.dump(obj, file, protocol=None): dump method accepts 3 arguments where protocol is None by default.



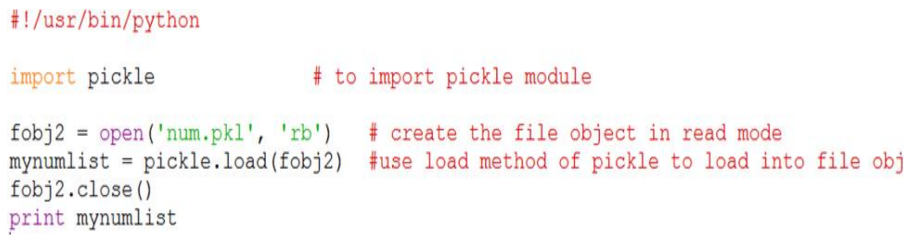
```
#!/usr/bin/python   #set the path in linux platform (by using which python)
import random       #import random module to generate and shuffle
elements_num = 10   #set the number of elements in the list to 10
numlist = list(range(elements_num))    #create a list of numbers from 1 to 10
random.shuffle(numlist)    #shuffle the list created
print numlist

import pickle              # import pickle module

fobj1 = open('num.pkl', 'wb')   # create a file object with write mode
pickle.dump(numlist, fobj1)  #use dump method of pickle to dump data to file obj
fobj1.close()
```

Fig.1 A sample code for Python list object serialization

Execution of this code will create a list of numbers with 10 elements.num.pkl file shown in Fig.1contains the byte stream equivalent of the list data.

De-serialization of python object using pickle

The following code in Fig.2 is used to create a file object for the same file where the list data was dumped earlier, but now it is opened for read mode and the load method is used to read the list object from the num.pkl and collected into mynumlist.



```
#!/usr/bin/python

import pickle              # to import pickle module

fobj2 = open('num.pkl', 'rb')   # create the file object in read mode
mynumlist = pickle.load(fobj2)  #use load method of pickle to load into file obj
fobj2.close()
print mynumlist
```

Fig.2 Sample code in Python list object deserialization using load method

Execution of this code will create same original list which was serialized will be de-serialized and collected in mynumlist.

Here is the same example in Fig.3 with dump method, but protocol parameter is -1 to indicate Highest_Protocol to be used. Possible versions of protocol are 0 to indicate original ASCII protocol, 1 to indicate old binary format and 2 new style classes and they differ in terms of backward compatibility. By default, 0 is used. Here is the serialization and deserialization with highest protocol being chosen by setting protocol parameter to -1.

```
#!/usr/bin/python
import random
NUMBER_OF_ELEMENTS = 10
numlist = list(range(NUMBER_OF_ELEMENTS))
random.shuffle(numlist)
print numlist
import pickle             #import pickle module

f = open('num.pkl', 'wb')   # 'wb' write mode for binary file
pickle.dump(numlist, f, -1)       #  indicates highest binary protocol
f.close()
```

Fig.3 Sample Python code to serialize with Highest Protocol field

Content of num.pkl file, will not be in the human readable format, as the Highest_Protocol is 2.

De-serialization of Python object using load method

Deserializations using pickle load method is shown below in Fig.4.

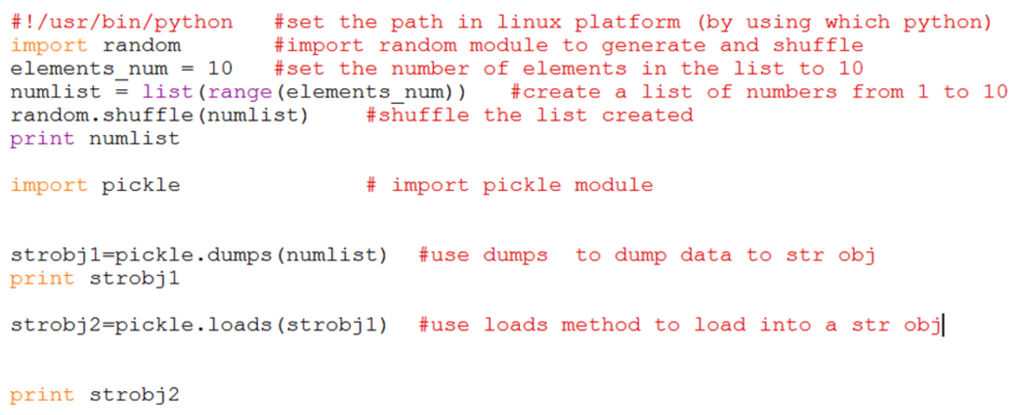
```
#!/usr/bin/python
import pickle

f = open('num.pkl', 'rb')   # 'rb' for reading binary file
mynumlist = pickle.load(f)
f.close()

print mynumlist
```

Fig. 4 Sample code for deserialization using load method

Object will be de-serialized and contained in mynumlist .Same code can be used for serialization and de-serialization of any arbitrary Python object.

Dumps and loads- working with string objects

Python sample code for string object serialization and de-serialization is shown below in Fig.5 with dumps and loads method



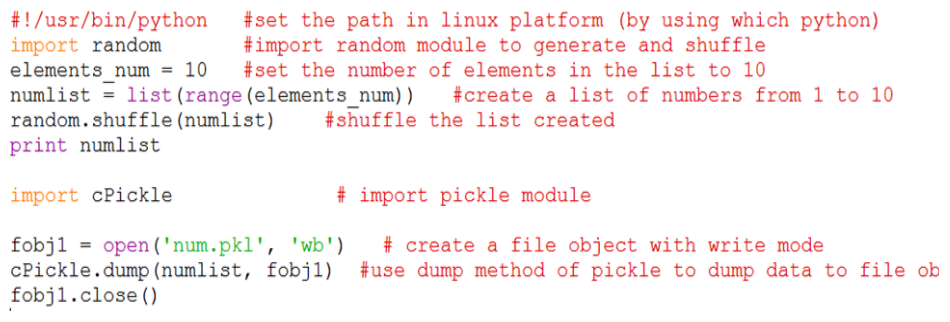
```
#!/usr/bin/python   #set the path in linux platform (by using which python)
import random       #import random module to generate and shuffle
elements_num = 10   #set the number of elements in the list to 10
numlist = list(range(elements_num))   #create a list of numbers from 1 to 10
random.shuffle(numlist)    #shuffle the list created
print numlist

import pickle              # import pickle module


strobj1=pickle.dumps(numlist)  #use dumps  to dump data to str obj
print strobj1

strobj2=pickle.loads(strobj1)  #use loads method to load into a str obj


print strobj2
```

Fig. 5 Sample code for serialization and deserialization of Python string object using dumps and loads method

Serialization and de-serialization python object with cPickle

cPickle module provides almost identical functionalities as that of pickle module, but is very much faster, as the implementation is done in C. It is also available as a standard library module.

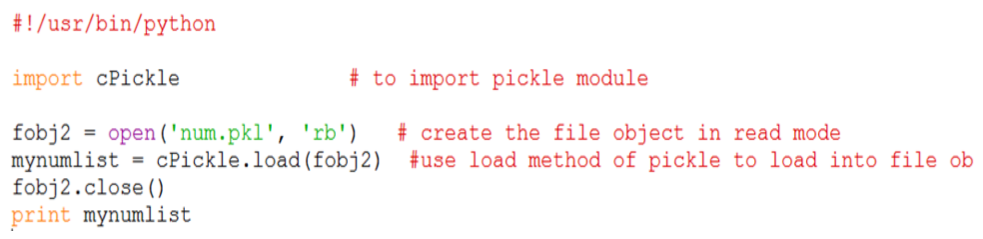
Python code with cPickle for serialization using dump method is shown in Fig.6 for list objects.



```
#!/usr/bin/python   #set the path in linux platform (by using which python)
import random       #import random module to generate and shuffle
elements_num = 10   #set the number of elements in the list to 10
numlist = list(range(elements_num))   #create a list of numbers from 1 to 10
random.shuffle(numlist)    #shuffle the list created
print numlist

import cPickle              # import pickle module

fobj1 = open('num.pkl', 'wb')   # create a file object with write mode
cPickle.dump(numlist, fobj1)  #use dump method of pickle to dump data to file ob
fobj1.close()
```

Fig.6 Sample code of Python list object serialization using dump method

Sample Python code with cPickle for De-serialization is shown in fig.7.

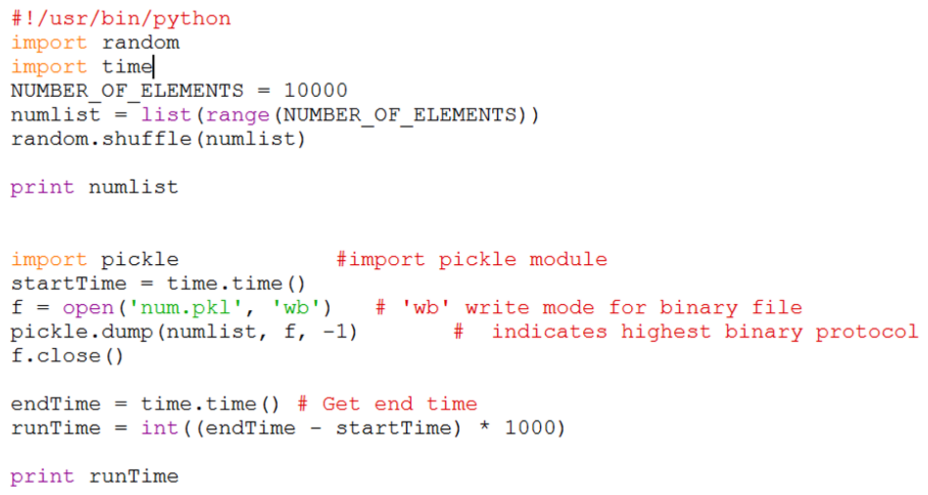


```
#!/usr/bin/python

import cPickle              # to import pickle module

fobj2 = open('num.pkl', 'rb')   # create the file object in read mode
mynumlist = cPickle.load(fobj2)  #use load method of pickle to load into file ob
fobj2.close()
print mynumlist
```

Fig. 7 Sample code of Python list object deserialization using load method

Performance comparison of pickle and cpickle

To compare the performance between the 2 modules, in terms of serialization the following code shown in Fig. 8 is used to create a huge list where the number of elements are 10000 and time module is used to calculate the run time, as the difference between start and end times required each for serialization and de-serialization. It can be noticed that the time required for cPickle to serialize or de-serialize is much lesser and hence it is faster compared to pickle module.



```
#!/usr/bin/python
import random
import time
NUMBER_OF_ELEMENTS = 10000
numlist = list(range(NUMBER_OF_ELEMENTS))
random.shuffle(numlist)

print numlist


import pickle             #import pickle module
startTime = time.time()
f = open('num.pkl', 'wb')    # 'wb' write mode for binary file
pickle.dump(numlist, f, -1)       #  indicates highest binary protocol
f.close()

endTime = time.time() # Get end time
runTime = int((endTime - startTime) * 1000)

print runTime
```

Fig. 8 Sample code to calculate the time for serialization of Python list object

Serialization with Pickle run time to create list object serialization is 122 msec, whereas the time with cPickle is obtained as 4 msec. Deserialization time using pickle and cPickle module module is obtained as 41 and 7 msec, thus proving cPickle is better than pickle module for Pythonobject serialization and deserialization.

IV. CONCLUSIONS

In summary, serialization/de-serialization concepts, their practical requirement scenarios are discussed. In addition, the pickle and cPickle modules, their important methods to work on file and string objects are discussed along with the codes and their responses. Performance comparison between pickle and cPickle modules are also analyzed through the code and cPickle is faster compared to pickle module. Further study can be done on serialization of other Python objects using Camel, Yaml modules.

REFERENCES

[1] https://docs.python.org/2/tutorial/index.html
[2] https://docs.python.org/2/library/pickle.html?highlight=pickle#module-pickle
[3] https://pymotw.com/2/pickle/