Python Programming

by Narendra Allam

Chapter 9

Serialization

Topics Covering

- Pickle Module
- pickling built-in data structures
    byte strings
    binary
- xml construction and parsing
- json construction and parsing

Serialization: Serialization is the process of transforming data from one container to to another. An employee info is stored in a databse table as a record, the same is stored in a program as a tuple, structure variable or a class object. Same data can be transformed into some text representation like an xml file or json file. Data changes its container but not the structure. This process of transformation happens at various stages. The process of transforming from format A to format B is called **Serialization** and B to A is called **Deserialization**. Different technologies have different names for this process.

Encoding - Decoding
Marshalling - Unmarshalling
Pickling - Unpickling

all refer to the same process. But at present, most of the technical contexts serialization is being referred when conversion happens to and from XML and JSON formats.

In this chapter we mainly discuss 3 formats.

1. Pickling - python exclusive format
2. xml
3. json

Pickle

The pickle module implements a fundamental, but powerful algorithm for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream is converted back into an object hierarchy.

converting a python data structure into pickle format

dumps() function converts python data structure into a binary string format, later we can transfer this to a file or network.

In[]
```
import pickle
d = {}
d['id '] = 12345
d['name'] = 'Obama'
d['salary'] = 9000000.0

print d
```

```
{'salary': 9000000.0, 'name': 'Obama', 'id ': 12345}
```

In[]
```
bs = pickle.dumps(d)
```

In[]
```
print bs
```

```
(dp0
S'salary'
p1
F9000000.0
sS'name'
p2
S'Obama'
p3
sS'id '
p4
I12345
s.
```

Writing into a file in binary format('wb' mode)

In[]
```
f = open('employ.pickle', 'wb')
f.write(bs)
f.close()
```

Checking the file content

```
In[]  !cat employ.pickle
```

```
(dp0
S'salary'
p1
F9000000.0
sS'name'
p2
S'Obama'
p3
sS'id '
p4
I12345
s.
```

Reading content from a pickle file

loads() funcion is used to load data back from a pickle file

```
In [ ]: f = open('employ.pickle', 'rb')
        bs = f.read()
        d1 = pickle.loads(bs)
        print d1
        f.close()
```

Creating an intermediatory binary string is not rquired all the time. We may want to directly write into a file. Python provides a straight way to do this. load() and dump() functions.

```
In[]  d = {}
      d['id '] = 12345
      d['name'] = 'Obama'
      d['salary'] = 9000000.0

      f = open('employee.pickle', 'wb')
      pickle.dump(d, f, pickle.HIGHEST_PROTOCOL)
      f.close()
```

pickle.HIGHEST_PROTOCOL ensures the highest protocol to be used in pickling protocol. We can see the pure binary form of pickling.

```
In[]  !cat employee.pickle
```

```
��}q(UsalaryqGAa*�UnameqUObamaqUid qM90u.
```

Using load to unpickle back to original data structure:

```
In[] d = {}
     with open('employee.pickle', 'rb') as f:
         d = pickle.load(f)
     print d
```

```
{'salary': 9000000.0, 'name': 'Obama', 'id ': 12345}
```

Xml

Xml is another popular format used mainly in web based data exchange scenorios. XML is an hierarchical data format, and the most natural way to represent it is with atree.

1. `xml.etree.ElementTree`
2. `ET.Element()`