pickle

Pythonic object serialization

Atul Varma

The Chicago Python Users Group

January 10, 2008

What is serialization?

It’s the process of saving an object onto a storage medium (such as a file) or to transmit it across a network in binary form.

A simple example...

```
>>> aList = [1]
>>> obj = (aList, aList)


>>> import pickle
>>> pickle.dumps( obj )
'((lp0\nI1\nag0\ntp1\n.'
>>> pickle.loads( '((lp0\nI1\nag0\ntp1\n.' )
([1], [1])
```

Works with class instances!

```
>>> class Player( object ):
...    def __init__( self, name ):
...       print “Player %s created.” % name
...       self.name = name
>>> player = Player( “Argon” )
Player Argon created.

>>> pickle.dumps( player )
‘ccopy_reg\n_reconstructor\np0\n(c__main__\nPlayer...’
>>> clonedPlayer = pickle.loads( _ )
>>> clonedPlayer.name
‘Argon’
```

pickle isn’t secure.

“no sufficient security analysis has been done to guarantee this and there isn't a use case that warrants the expense of such an analysis.”

from `pickletools.py`

pickle is cross-platform.

Protocol version 0 is text-only, versions 1 and 2 are platform-independent and binary.

Unanswered questions...

Unanswered questions...

- Why isn't the Player constructor called?

Unanswered questions...

- Why isn't the Player constructor called?
- What if I add a new attribute to the Player class, or change an instance method?

Unanswered questions...

- Why isn't the Player constructor called?
- What if I add a new attribute to the Player class, or change an instance method?
- What if I move the Player class to a different file?

Unanswered questions...

- Why isn't the Player constructor called?
- What if I add a new attribute to the Player class, or change an instance method?
- What if I move the Player class to a different file?
- What if there's things in a Player instance that I don't want serialized, like a socket?

What is a pickle, really?

"A *pickle* is a program for a virtual pickle machine (PM)... It's a sequence of opcodes, interpreted by the PM, building an arbitrarily complex Python object."

from `pickletools.py`

Pickle Machine properties

Pickle Machine properties

- Two data areas: *the stack* and *the memo*.

Pickle Machine properties

- Two data areas: *the stack* and *the memo*.
- No looping, testing, or conditional instructions.

Pickle Machine properties

- Two data areas: *the stack* and *the memo*.
- No looping, testing, or conditional instructions.
- No arithmetic instructions.

Pickle Machine properties

- Two data areas: *the stack* and *the memo*.
- No looping, testing, or conditional instructions.
- No arithmetic instructions.
- No function calls.

Pickle Machine properties

- Two data areas: *the stack* and *the memo*.
- No looping, testing, or conditional instructions.
- No arithmetic instructions.
- No function calls.
- Opcodes are executed from first to last until a `STOP` instruction is reached.

Disassembling pickles

```
>>> import pickletools
>>> # Let's disassemble (aList, aList), where aList = [1].
>>> pickletools.dis( '((lp0\nI1\nag0\ntp1\n.' )
    0: (    MARK
    1: (        MARK
    2: l            LIST       (MARK at 1)
    3: p        PUT        0
    6: I        INT        1
    9: a        APPEND
   10: g        GET        0
   13: t        TUPLE      (MARK at 0)
   14: p    PUT        1
   17: .    STOP
highest protocol among opcodes = 0
```

More disassembly...

```
>>> # Let's disassemble our Argon player.
>>> pickletools.dis( 'ccopy_reg\n_reconstructor...' )
    0: c    GLOBAL     'copy_reg _reconstructor'
   25: p    PUT        0
   28: (    MARK
   29: c        GLOBAL     '__main__ Player'
   46: p        PUT        1
   49: c        GLOBAL     '__builtin__ object'
   69: p        PUT        2
   72: N        NONE
   73: t        TUPLE      (MARK at 28)
   74: p    PUT        3
   77: R    REDUCE
   78: p    PUT        4
```

That was all shorthand for...

```
>>> copy_reg._reconstructor( __main__.Player,
...                          __builtin__.object,
...                          None )
<__main__.Player object at 0x41f170>
```

That was all shorthand for...

```
>>> copy_reg._reconstructor( __main__.Player,
...                          __builtin__.object,
...                          None )
<__main__.Player object at 0x41f170>
```

What's this reconstructor?

That was all shorthand for...

```
>>> copy_reg._reconstructor( __main__.Player,
...                          __builtin__.object,
...                          None )
<__main__.Player object at 0x41f170>
```

What's this reconstructor?

```
def _reconstructor(cls, base, state):
    if base is object:
        obj = object.__new__(cls)
    else:
        obj = base.__new__(cls, state)
        base.__init__(obj, state)
    return obj
```

Zomg, a half-born object!

Build our instance attributes!

```
81: (    MARK
82: d        DICT       (MARK at 81)
83: p    PUT        5
86: S    STRING     'name'
94: p    PUT        6
97: S    STRING     'Argon'
106: p   PUT        7
109: s   SETITEM
110: b   BUILD
111: .   STOP
```

That was shorthand for...

```
>>> obj.__dict__.update( {“name” : “Argon”} )
```

Pickling customization

Pickling customization

- `__getstate__()`, `__setstate__()`

Pickling customization

- `__getstate__()`, `__setstate__()`
- `__getnewargs__()`

Pickling customization

- `__getstate__()`, `__setstate__()`
- `__getnewargs__()`
- Subclass picklers and unpicklers

Pickling customization

- `__getstate__()`, `__setstate__()`
- `__getnewargs__()`
- Subclass picklers and unpicklers
- Override an unpickler's `find_class()` method

Where to go next...

Where to go next...

- Look at the Python documentation for `pickle` and `cPickle`.

Where to go next...

- Look at the Python documentation for `pickle` and `cPickle`.
- For more information on the Pickle Machine internals, see the source code for `pickletools.py`.

Where to go next...

- Look at the Python documentation for `pickle` and `cPickle`.
- For more information on the Pickle Machine internals, see the source code for `pickletools.py`.
- To learn more about the specifics of class instantiation and other Python internals, see David Beazley's *Python Essential Reference*.