lambdaJSON Documentation

Release 0.2.16

Pooya Eghbali

September 17, 2013

CONTENTS

Serialize python standard types (function, tuple, class, memoryview, set, frozenset, exceptions, complex, range, bytes, bytearray, dict with number keys, byte keys or tuple keys, and etc) with json. Contents:

CHAPTER

ONE

INSTALLATION

Source package is available on PyPi. It is pure python and you do not need a c/c++ compiler.

You can get the package from PyPi: https://pypi.python.org/pypi/lambdaJSON

To install, run:

```
python setup.py install
```

You can also use pip to install the package:

```
pip install lambdaJSON
```

to upgrade using pip:

```
pip install lambdaJSON --upgrade
```

CHAPTER

TWO

USAGE

Serialize python standard types (function, tuple, class, memoryview, set, frozenset, exceptions, complex, range, bytes, bytearray, dict with number keys, byte keys or tuple keys, and etc) with json. lambdaJSON lets you serialize python standard library objects with json.

2.1 Typical usage

I'll show you some basic usage of the lambdaJSON lib. for more advanced examples please visit the examples section.

2.1.1 Serialize Complex dict

You can serialize any dicts with lambdaJSON supported keys (and also hashable). this includes dictionaries with byte keys, tuple keys and etc:

```
#!/usr/bin/env python

>>> import lambdaJSON
>>> myComplexData = {True: (3-5j), (3+5j): b'json', (1, 2, 3): {b'lambda': [1, 2, 3, (3, 4, 5)]}}
>>> mySerializedData = lambdaJSON.serialize(myComplexData)
>>> myComplexData  == lambdaJSON.deserialize(mySerializedData)
True

>>>
```

2.1.2 Passing values to json functions

To pass args and kwargs to the encoder/decoder simply pass them to the serialize/deserialize function, example for json:

```
>>> mySerializedData = lambdaJSON.serialize(myComplexData, sort_keys = True)
>>> myComplexData  == lambdaJSON.deserialize(mySerializedData, object_hook = my_hook)
```

It can be done for ujson too.

2.1.3 Serializing python functions

You can also serialize python functions:

```
>>> import lambdaJSON
>>> def f(): print('lambdaJSON Rocks!')

>>> mySerializedFunction = lambdaJSON.serialize(f)
>>> myNewFunction  = lambdaJSON.deserialize(mySerializedFunction)
>>> myNewFunction()
'lambdaJSON Rocks!'
>>>
```

Changed int 0.2.4, for function deserialization you must pass a function which returns the list of globals for the function:

```
>>> import lambdaJSON
>>> y = 10
>>> def f(x): return x*y

>>> mySerializedFunction = lambdaJSON.serialize(f)
>>> myNewFunction  = lambdaJSON.deserialize(mySerializedFunction, globs = (lambda: globals()))
>>> myNewFunction(5)
50
>>> y = 3
>>> myNewFunction(5)
15
>>>
```

If no globs passed to function, the globs will be just the __builtins__ module. Note that passing globals will pass the lambdaJSONs globals and it will not work, if you want to include all the globals from where the deserialization function is called, just use globs = (lambda: globals()), else implement your own function. You can do some nice hacks too:

```
>>> z = 10
>>> def g():
        global z
        z += 1
        return {'z':z}

>>> def f(x,y): return x*y+z

>>> mySerializedFunction = lambdaJSON.serialize(f)
>>> myNewFunction  = lambdaJSON.deserialize(mySerializedFunction, globs = g)
>>> myNewFunction(2,3)
17
>>> myNewFunction(2,3)
18
>>>
```

2.1.4 Serializing builtin exceptions

You can serialize Builtin Exceptions like this:

```
>>> a = lambdaJSON.serialize(OSError('FILE NOT FOUND'))
>>> b = lambdaJSON.deserialize(a)
>>> raise b
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    raise b
OSError: FILE NOT FOUND
>>>
```

2.1.5 Serializing python classes

introduced in version 0.2.15, you can now serialize basic classes and types. The support is basic, but I'm planning to develop the class serialization support in the next subversion. to deserialize a class, you must pass the globals function too, if you do not pass the globals, only __builtins__ will be passed to the class functions. this is an example to do it:

```
>>> class test(object):
        def __init__(self):
            self.var = 'lambdaJSON'

>>> serializedClass = lambdaJSON.serialize(test)
>>> newClass = lambdaJSON.deserialize(serializedClass, globs = lambda: globals())
>>> newClass().var
'lambdaJSON'
>>>
```

2.2 Find version

To check version, simply use lambdaJSON.__version__, or if you want to know which json lib is in use, try lambdaJSON.__json__

2.3 Json lib in use

LambdaJSON first tries to import ujson, if it fails it will import simplejson, and if that fails too, the json lib will be imported. you can check which json lib is in use with lambdaJSON.__json__ variable.

2.4 Currently Supported Types

This types are covered in this version:

1. Functions
2. Bytes and Bytearrays
3. Classes (basic support)
4. Builtin Exceptions
5. Tuples
6. Complex
7. Range
8. Set and Frozenset
9. Memoryview
10. Dicts (With Number, Tuple, String, Bool and Byte keys)
11. other json supported types

CHAPTER

THREE

EXAMPLES

Here you will find some more advanced examples of using the lambdaJSON lib.

3.1 Serialize an inherited class

It is easy to serialize a class if it is just inherited from builtin types, but serializing classes that are inherited from third party classes is a little tricky. see the examples to find out how to serialize these classes with lambdaJSON.

3.1.1 Serialize QWidget inheritance

When you are deserializing a class, lambdaJSON first searches for bases of the class inside the globals, if it did not find the base class it will try builtins, if not found an exception will be raised.

if you serialize a class that is inherited by a QtGui.QWidget, you will find out that the base class for this class is <class 'PySide.QtGui.QWidget'>, so in order to deserialize this class, you need to have 'PySide.QtGui.QWidget' in your globals dict (or just the globals you pass to deserialization function).

Simply importing QtGui from PySide will not work, and i will not create a function to search for the base class inside the globals because of security issues (idk if i change my mind or not). here is how you can serialize these classes:

```
>>> import lambdaJSON
>>> from PySide import QtGui
>>> class myWidget(QtGui.QWidget):
        pass

>>> serializedClass = lambdaJSON.serialize(myWidget)
>>> newClass = lambdaJSON.deserialize(serializedClass, globs = lambda: {'PySide.QtGui.QWidget': QtGui
```

3.1.2 Count Execution of deserialized function

I could develop lambdaJSON in a way that you just could pass a dictionary with references as globals, but i decided to accept functions instead of a dict, *to add more controling power*. with this, you can do some nice hacks, and one of them is this. The example source is:

```
>>> import lambdaJSON
>>> times = 0
>>> def g():
        global times
        times += 1
        print('function executed %s time%s'%(times, '' if times == 1 else 's'))
```

```
        return globals()

>>> def f(x): return x

>>> mySerializedFunction = lambdaJSON.serialize(f)
>>> myNewFunction  = lambdaJSON.deserialize(mySerializedFunction, globs = g)
>>> myNewFunction(2)
function executed 1 time
2
>>> myNewFunction(2)
function executed 2 times
2
>>>
```

CHAPTER

FOUR

QUICK REFERENCE

This section will provide a brief explaination to all parts and functions included in lambdaJSON lib.

4.1 __init__.py

Everything inside __init__.py!

4.1.1 __author__

if used, this value returns name of the author of the lambdaJSON lib, 'Pooya Eghbali'.

4.1.2 __version__

if used, this value returns the version of the lambdaJSON lib, eg: 2.0.16.

4.1.3 __json__

if used, this value returns the name of the json lib in use, eg: 'ujson'. You can use this to find the json lib if you want to pass for example ujson specific arguments to the deserialization function.

4.1.4 __builtins__

this is imported from __main__. the __builtins__ included in the top level module.

4.1.5 eval

actually this is the same as ast.literal_eval, used instead of builtins.eval to avoid security issues.

4.1.6 flatten

lambdaJSON uses this function to flatten objects and convert them to a format that json understands. if you just want to flattend an object, use this function.

4.1.7 restore

this is the reverse if the flatten function. restores object from the flattened one.

4.1.8 freezef

this function is used to flatten function objects. (imported from functions.py)

4.1.9 defreezef

reverse function for freezef. (imported from functions.py)

4.1.10 functions

this is same as functions.py.

4.1.11 ntypes

a list of available numerical types and bool. this is different in versions of python (there is no long type in py3k)

4.1.12 json

returns the json lib in use.

4.1.13 serialize

this is the main serialization function.

4.1.14 deserialize

and the main deserialize function!

4.2 functions.py

4.2.1 freezef

this function is used to flatten function objects.

4.2.2 defreezef

reverse function for freezef.

CHAPTER

FIVE

NOTES

Here are some notes and some warnings.

5.1 Serializing class and functions

1. Keep in mind, deserialized functions and classes are not equal to the original ones.
2. Deserialized class and functions are not serializable!
3. if you want to serialize a class, define class vars inside __init__. just class functions are included in serialization.

CHAPTER

SIX

CHANGES

v0.2.16, September 16 2013 – Functions default arg values are now serialized.

v0.2.15, September 15 2013 – Added basic support for classes.

v0.2.14, September 14 2013 – Added support for Builtin Exceptions.

v0.2.13, September 12 2013 – Added support for simplejson.

v0.2.12, September 11 2013 – Added support for memoryview.

v0.2.11, September 6 2013 – Added json lib and version identifiers. – Fixed the __builtins__ issue.

v0.2.10, September 3 2013 – Better code for function freezing.

v0.2.9, September 3 2013 – Fixed some compatibility issues.

v0.2.8, September 3 2013 – Added support for bytearray.

v0.2.7, September 2 2013 – Fixed a problem where objects in tuples didn't truly serialized.

v0.2.6, September 2 2013 – Fixed a compatibility issue.

v0.2.5, September 1 2013 – Added support for set and frozenset.

v0.2.4, September 1 2013 – globs for function deserialization now must be a function.

v0.2.3, September 1 2013 – Added Range support.

v0.2.2, August 31 2013 – Ability to pass globals to deserialized Functions. – Fixed a problem with globs exception on lists and dicts.

v0.2.1, August 31 2013 – Added __builtins__ to deserialized Functions.

v0.2.0, August 31 2013 – Added ability to serialize functions.

v0.1.10, August 30 2013 – Fixed a problem with *args and **kwargs

v0.1.9, August 30 2013 – Moved 'long' type existence determination outside of function to increase speed.

v0.1.8, August 30 2013 – Ability to pass args and kwargs to the json encoder/decoder.

v0.1.7, August 30 2013 – Fixed a problem came from vars(__builtins__)

v0.1.6, August 30 2013 – Added support for a faster json lib: ujson.

v0.1.5, August 29 2013 – Security fix. Using ast.literal_eval as eval.

v0.1.4, August 29 2013 – Support for py2 long with no hacks!

v0.1.3, August 29 2013 – Added support for Complex numbers.

v0.1.2, August 29 2013 – Added support for bool as dict key.

v0.1.1, August 28 2013 – Added support for python 2 long type.

v0.1, August 28 2013 – Initial release.