6.2 读写JSON数据¶

问题¶

你想读写JSON(JavaScript Object Notation)编码格式的数据。

解决方案¶

`json` 模块提供了一种很简单的方式来编码和解码JSON数据。 其中两个主要的函数是 `json.dumps()` 和 `json.loads()` ， 要比其他序列化函数库如pickle的接口少得多。 下面演示如何将一个Python数据结构转换为JSON：

```
import json

data = {
    'name' : 'ACME',
    'shares' : 100,
    'price' : 542.23
}

json_str = json.dumps(data)
```

下面演示如何将一个JSON编码的字符串转换回一个Python数据结构：

```
data = json.loads(json_str)
```

如果你要处理的是文件而不是字符串，你可以使用 `json.dump()` 和 `json.load()` 来编码和解码JSON数据。例如：

```
# Writing JSON data
with open('data.json', 'w') as f:
    json.dump(data, f)

# Reading data back
with open('data.json', 'r') as f:
    data = json.load(f)
```

讨论¶

JSON编码支持的基本数据类型为 `None` ， `bool` ， `int` ， `float` 和 `str` ， 以及包含这些类型数据的lists，tuples和dictionaries。 对于dictionaries，keys需要是字符串类型(字典中任何非字符串类型的key在编码时会先转换为字符串)。 为了遵循JSON规范，你应该只编码Python的lists和dictionaries。 而且，在web应用程序中，顶层对象被编码为一个字典是一个标准做法。

JSON编码的格式对于Python语法而已几乎是完全一样的，除了一些小的差异之外。 比如，True会被映射为true，False被映射为false，而None会被映射为null。 下面是一个例子，演示了编码后的字符串效果：

```
>>> json.dumps(False)
'false'
>>> d = {'a': True,
...      'b': 'Hello',
...      'c': None}
>>> json.dumps(d)
'{"b": "Hello", "c": null, "a": true}'
>>>
```

如果你试着去检查JSON解码后的数据，你通常很难通过简单的打印来确定它的结构， 特别是当数据的嵌套结构层次很深或者包含大量的字段时。 为了解决这个问题，可以考虑使用pprint模块的 `pprint()` 函数来代替普通的 `print()` 函数。 它会按照key的字母顺序并以一种更加美观的方式输出。 下面是一个演示如何漂亮的打印输出Twitter上搜索结果的例子：

```
>>> from urllib.request import urlopen
>>> import json
>>> u = urlopen('http://search.twitter.com/search.json?q=python&rpp=5')
```

```
>>> resp = json.loads(u.read().decode('utf-8'))
>>> from pprint import pprint
>>> pprint(resp)
{'completed_in': 0.074,
'max_id': 264043230692245504,
'max_id_str': '264043230692245504',
'next_page': '?page=2&max_id=264043230692245504&q=python&rpp=5',
'page': 1,
'query': 'python',
'refresh_url': '?since_id=264043230692245504&q=python',
'results': [{'created_at': 'Thu, 01 Nov 2012 16:36:26 +0000',
            'from_user': ...
            },
            {'created_at': 'Thu, 01 Nov 2012 16:36:14 +0000',
            'from_user': ...
            },
            {'created_at': 'Thu, 01 Nov 2012 16:36:13 +0000',
            'from_user': ...
            },
            {'created_at': 'Thu, 01 Nov 2012 16:36:07 +0000',
            'from_user': ...
            }
            {'created_at': 'Thu, 01 Nov 2012 16:36:04 +0000',
            'from_user': ...
            }],
'results_per_page': 5,
'since_id': 0,
'since_id_str': '0'}
>>>
```

一般来讲，JSON解码会根据提供的数据创建dicts或lists。 如果你想要创建其他类型的对象，可以给 `json.loads()` 传递object_pairs_hook或object_hook参数。 例如，下面是演示如何解码JSON数据并在一个OrderedDict中保留其顺序的例子：

```
>>> s = '{"name": "ACME", "shares": 50, "price": 490.1}'
>>> from collections import OrderedDict
>>> data = json.loads(s, object_pairs_hook=OrderedDict)
>>> data
OrderedDict([('name', 'ACME'), ('shares', 50), ('price', 490.1)])
>>>
```

下面是如何将一个JSON字典转换为一个Python对象例子：

```
>>> class JSONObject:
...     def __init__(self, d):
...         self.__dict__ = d
...
>>>
>>> data = json.loads(s, object_hook=JSONObject)
>>> data.name
'ACME'
>>> data.shares
50
>>> data.price
490.1
>>>
```

最后一个例子中，JSON解码后的字典作为一个单个参数传递给 `__init__()` 。 然后，你就可以随心所欲的使用它了，比如作为一个实例字典来直接使用它。

在编码JSON的时候，还有一些选项很有用。 如果你想获得漂亮的格式化字符串后输出，可以使用 `json.dumps()` 的indent参数。 它会使得输出和pprint()函数效果类似。比如：

```
>>> print(json.dumps(data))
{"price": 542.23, "name": "ACME", "shares": 100}
>>> print(json.dumps(data, indent=4))
{
    "price": 542.23,
    "name": "ACME",
    "shares": 100
```

```
}
>>>
```

对象实例通常并不是JSON可序列化的。例如：

```
>>> class Point:
...     def __init__(self, x, y):
...         self.x = x
...         self.y = y
...
>>> p = Point(2, 3)
>>> json.dumps(p)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "/usr/local/lib/python3.3/json/__init__.py", line 226, in dumps
        return _default_encoder.encode(obj)
    File "/usr/local/lib/python3.3/json/encoder.py", line 187, in encode
        chunks = self.iterencode(o, _one_shot=True)
    File "/usr/local/lib/python3.3/json/encoder.py", line 245, in iterencode
        return _iterencode(o, 0)
    File "/usr/local/lib/python3.3/json/encoder.py", line 169, in default
        raise TypeError(repr(o) + " is not JSON serializable")
TypeError: <__main__.Point object at 0x1006f2650> is not JSON serializable
>>>
```

如果你想序列化对象实例，你可以提供一个函数，它的输入是一个实例，返回一个可序列化的字典。例如：

```
def serialize_instance(obj):
    d = { '__classname__' : type(obj).__name__ }
    d.update(vars(obj))
    return d
```

如果你想反过来获取这个实例，可以这样做：

```
# Dictionary mapping names to known classes
classes = {
    'Point' : Point
}

def unserialize_object(d):
    clsname = d.pop('__classname__', None)
    if clsname:
        cls = classes[clsname]
        obj = cls.__new__(cls) # Make instance without calling __init__
        for key, value in d.items():
            setattr(obj, key, value)
        return obj
    else:
        return d
```

下面是如何使用这些函数的例子：

```
>>> p = Point(2,3)
>>> s = json.dumps(p, default=serialize_instance)
>>> s
'{"__classname__": "Point", "y": 3, "x": 2}'
>>> a = json.loads(s, object_hook=unserialize_object)
>>> a
<__main__.Point object at 0x1017577d0>
>>> a.x
2
>>> a.y
3
>>>
```

`json` 模块还有很多其他选项来控制更低级别的数字、特殊值如NaN等的解析。 可以参考官方文档获取更多细节。