# [Python Json序列化与反序列化](https://www.cnblogs.com/diaosicai/p/6419833.html)

　　在python中，**序列化可以理解为：把python的对象编码转换为json格式的字符串，反序列化可以理解为：把json格式字符串解码为python数据对象。**在python的标准库中，专门提供了json库与pickle库来处理这部分。

　　json的dumps方法和loads方法，可实现数据的序列化和反序列化。具体来说，dumps方法，可将json格式数据序列为[Python](http://lib.csdn.net/base/python)的相关的数据类型；loads方法则是相反，把python数据类型转换为json相应的数据类型格式要求。在序列化时，中文汉字总是被转换为unicode码，在dumps函数中添加参数ensure\_ascii=False即可解决。

 JSON可以理解为序列化的标准格式。JSON和Python内置的数据类型对应如下：

| **JSON类型** | **Python类型** |
| --- | --- |
| {} | dict |
| [] | list |
| “string” | str |
| 1234.56 | int/float |
| true/false | True/False |
| null | None |

　　下面是json的序列化与反序列化：

　　1、Json序列化如下：

1 import json

2 print (json.\_\_all\_\_) #查看json库的所有方法

3 ['dump', 'dumps', 'load', 'loads', 'JSONDecoder', 'JSONEncoder']

　　未在dumps函数中添加参数ensure\_ascii=False，结果如下：

[复制代码](javascript:void(0);)

 import **json**

 dict = {'name':'zhangsan', 'age':33, 'address':'红星路'}

**print**('未序列化前的数据类型为:', **type**(dict))

**print**('为序列化前的数据：', dict)

*#对dict进行序列化的处理*

 dict\_xu = **json**.**dumps**(dict)    *#直接进行序列化*

**print**('序列化后的数据类型为：', **type**(dict\_xu))

**print**('序列化后的数据为：', dict\_xu)

 ----------------------------------------------------------------

 未序列化前的数据类型为: <class 'dict'>

 为序列化前的数据： {'name': 'zhangsan', 'address': '红星路', 'age': 33}

 序列化后的数据类型为： <class 'str'>

 序列化后的数据为： {"name": "zhangsan", "address": "\u7ea2\u661f\u8def", "age": 33}

[复制代码](javascript:void(0);)

　　在dumps函数中添加参数ensure\_ascii=False，结果如下：



*#coding: utf-8*

import **json**

dict = {'name':'zhangsan', 'age':33, 'address':'红星路'}

**print**('未序列化前的数据类型为:', **type**(dict))

**print**('为序列化前的数据：', dict)

*#对dict进行序列化的处理*

dict\_xu = **json**.**dumps**(dict,ensure\_ascii=False)    *#添加ensure\_ascii=False进行序列化*

**print**('序列化后的数据类型为：', **type**(dict\_xu))

**print**('序列化后的数据为：', dict\_xu)

-------------------------------------------------------------------

未序列化前的数据类型为: <class 'dict'>

为序列化前的数据： {'address': '红星路', 'age': 33, 'name': 'zhangsan'}

序列化后的数据类型为： <class 'str'>

序列化后的数据为： {"address": "红星路", "age": 33, "name": "zhangsan"}

[复制代码](javascript:void(0);)

　　2、Json反序列化如下：

[复制代码](javascript:void(0);)

import **json**

dict = {'name':'zhangsan', 'age':33, 'address':'红星路'}

**print**('未序列化前的数据类型为:', **type**(dict))

**print**('为序列化前的数据：', dict)

*#对dict进行序列化的处理*

dict\_xu = **json**.**dumps**(dict,ensure\_ascii=False)    *#添加ensure\_ascii=False进行序列化*

**print**('序列化后的数据类型为：', **type**(dict\_xu))

**print**('序列化后的数据为：', dict\_xu)

*#对dict\_xu进行反序列化处理*

dict\_fan = **json**.**loads**(dict\_xu)

**print**('反序列化后的数据类型为：', **type**(dict\_fan))

**print**('反序列化后的数据为: ', dict\_fan)

结果：

15 ----------------------------------------------------------------------

16 未序列化前的数据类型为: <class 'dict'>

17 为序列化前的数据： {'name': 'zhangsan', 'age': 33, 'address': '红星路'}

18 序列化后的数据类型为： <class 'str'>

19 序列化后的数据为： {"name": "zhangsan", "age": 33, "address": "红星路"}

20 反序列化后的数据类型为： <class 'dict'>

21 反序列化后的数据为: {'name': 'zhangsan', 'age': 33, 'address': '红星路'}

[复制代码](javascript:void(0);)

 　　在实际的工作中，序列化或者反序列化的可能是一个文件的形式，不可能像如上写的那样简单的，下来就来实现这部分，把文件内容进行序列化和反序列化，先来看序列化的代码，两步操作：1、先序列化 列表对象 ；2、步把序列化成的字符串写入文件：



*#coding: utf-8*

import **json**

list = ['Apple','Huawei','selenium','java','python']

*#把list先序列化，写入到一个文件中*

*# 两步操作 1步先序列化 列表对象  2步把序列化成的字符串写入文件*

**json**.**dump**(list, **open**('e:/test.txt','w'))

r1=**open**('e:/test.txt','r')

**print**(r1.**read**())

10 -------------------------------------------------------------------

11 ["Apple", "Huawei", "selenium", "java", "python"]

[复制代码](javascript:void(0);)

反序列化，两步操作：1、先读取文件的字符串对象；2、然后反序列化成列表对象：



*#coding: utf-8*

import **json**

list = ['Apple','Huawei','selenium','java','python']

*#把list先序列化，写入到一个文件中*

*# 两步操作 1步先序列化 列表对象  2步把序列化成的字符串写入文件*

**json**.**dump**(list, **open**('e:/test.txt','w'))

r1=**open**('e:/test.txt','r')

**print**(r1.**read**())

*#------------------------------------------------------------*

*#两步操作：1、先读取文件的字符串对象；2、然后反序列化成列表对象*

res=json.load(**open**('e:/test.txt','r'))

**print** (res)

**print**('数据类型:',**type**(res))

结果：

15 -------------------------------------------------------------

16 ["Apple", "Huawei", "selenium", "java", "python"]

17 ['Apple', 'Huawei', 'selenium', 'java', 'python']

18 数据类型: <class 'list'>

[复制代码](javascript:void(0);)

**在class对象中使用JSON：**

通常情况下，我们都习惯使用**class**表示对象，若我们需要将Student类进行序列化 ，我们可以做如下尝试：

import json

class Student(object):

def \_\_init\_\_(self, name, age, score):

self.name = name

self.age = age

self.score = score

s = Student('WeiYu', 23, 98)

print(json.dumps(s))

**看似很合理，但是会得到一个错误：**

Traceback(most recent call last):

...

TypeError: <\_\_main\_\_.Student object at 0x0000023A4C643128> is not JSON serializable

原因是Student这个对象并不能序列化为JSON。  
但是[dumps()方法的参数列表](https://docs.python.org/3/library/json.html#json.dumps)中还有很多可选参数，这些款参数就是提供给我们来定制JSON序列化的。  
其中default参数就是把任意一个对象编程一个可序列的JSON的对象，但是在这之前需要为Student写一个转换函数，然后再将函数传进去：json只能够序列化字典类型的数据

import json

class Student(object):

def \_\_init\_\_(self, name, age, score):

self.name = name

self.age = age

self.score = score

*# 编写一个转换函数如下：*

def student2dict(std):

return {

'name': std.name,

'age': std.age,

'score': std.score

}

s = Student('WeiYu', 23, 98)

print(json.dumps(s, default=student2dict)) *# 将转换函数传入参数default*

>>> *# 输出结果*

{"score": 98, "age": 23, "name": "WeiYu"}

这样Student实例先被student2dict转换成dict，然后被序列化为JSON，但是如果下一次遇到另外一个实例Teacher，又需要编写一个转换函数，这样可能就太麻烦了。  
我们可以利用**\_\_dict\_\_**属性，将任意class实例变为dict:

>>>print(json.dumps(s, default=lambda obj: obj.\_\_dict\_\_))

>>>

{"name": "WeiYu", "age": 23, "score": 98}

得到同样的结果，因为通常的class的实例都有一个–dict–(前后是两个下滑线)属性，它自身就是一个dict，用来存储实例变量。  
同理，需要反序列化JSON时，loads()方法先转换出一个dict对象，然后通过传入的object\_hook函数将dict转换为Student实例，[查看loads()参数列表](https://docs.python.org/3/library/json.html#json.loads)。定义转换函数：

def dict2student(d):

return Student(d['name'], d['age'], d['score'])

*# 测试输出*

>>> json\_str = '{"age": 20, "score": 88, "name": "Bob"}'

>>> print(json.loads(json\_str, object\_hook=dict2student))

<\_\_main\_\_.Student object at 0x000001F7FDE1E940>

# Python Pickle序列化

Python中的Pickle模块主要的用途在于，方便一些复杂的数据类型的保存与加载，如果仅仅保存到txt文件中，只能保存为str类型的数据，这在反序列化的时候是非常不方便的, 需要加载str类型数据后再通过格式转换得到你想要的数据，这里对pickle的功能进行简单的学习，最佳排版请看: [Python Pickle](https://link.zhihu.com/?target=https%3A//rentainhe.github.io/2022/02/06/python-pickle/%23more)

**Pickle常见API**

* pickle.dump(obj, file): 将python的数据转换并保存到pickle格式的文件内
* pickle.dumps(obj): 将python数据转换为pickle格式的bytes字串
* pickle.load(file): 从pickle格式的文件中读取数据并转换为python类型
* pickle.loads(bytes\_object): 从pickle格式的bytes字串转换为python类型

可以看到pickle.dump对应pickle.load, pickle.dumps对应pickle.loads

**Pickle Basic Usage**

* Pickle最基本用法就是使用上述的4个接口进行序列化和反序列化操作

**import** pickle

**import** pickle

*# 定义了一个dict*

my\_dict **=** {"name": "ren", "age": 28}

*# 利用pickle.dump方法序列化这个dict并保存到一个.pickle后缀的文件中*

**with** open("my\_dict.pickle", "wb") **as** f:

pickle**.**dump(my\_dict, f) *# pickle保存下来的.pickle文件是不具有任何可读性的编码*

*# 再利用pickle.load方法载入并赋给一个新的变量*

**with** open("my\_dict.pickle", "rb") **as** f:

my\_dict\_unpickled **=** pickle**.**load(f)

**print**(my\_dict\_unpickled) *# {'name': 'ren', 'age': 28}，也是dict类型的数据*

利用pickle模块保存下来的.pickle后缀的文件是不具有任何可读性的

* 可以同时将许多不同类型数据保存在同一份.pickle文件中, 再按照相同的顺序读出

**import** pickle

a1 **=** "apple"

b1 **=** {1: 'One', 2: 'Two'}

c1 **=** ['test', 'name', 1, 2, 3]

f1 **=** open("temp.pkl", "wb")

*# 可以保存到同一个文件中*

pickle**.**dump(a1, f1)

pickle**.**dump(b1, f1)

pickle**.**dump(c1, f1)

f1**.**close()

*# 再按照相同的顺序读出*

f2 **=** open("temp.pkl", "rb")

a2 **=** pickle**.**load(f2)

b2 **=** pickle**.**load(f2)

c2 **=** pickle**.**load(f2)

**print**(a2) *# apple*

**print**(b2) *# {1: 'One', 2: 'Two'}*

**print**(c2) *# ['test', 'name', 1, 2, 3]*

f2**.**close()

**Pickle需要注意的细节**

* pickle不仅可以保存python中的基本类型，也可以保存自己定义的class，但是在反序列化的时候，这个class的定义必须存在

**import** pickle

**class** **Person**:

**def** \_\_init\_\_(self, name, age):

self**.**name **=** name

self**.**age **=** age

**def** **show**(self):

**print**(self**.**name, self**.**age)

person1 **=** Person("ren", 22)

person1**.**show() *# ren 22*

*# 可以保存一个自定义的数据类型*

**with** open("temp.pkl", "wb") **as** f:

pickle**.**dump(person1, f)

*# 如果在加载前将自定义数据类型删除的话, 则会加载失败*

*# del Person*

**with** open("temp.pkl", "rb") **as** f:

person\_unpickled **=** pickle**.**load(f)

person\_unpickled**.**show() *# ren 22*