**一、关于JSON**

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，易于人阅读和编写.是一种文件规范,绝大多数的编程语言均可以轻松读写.当然python也不例外.  
本文要讲的主要是python对JSON数据的序列化和反序列化.所谓序列化就是写入到文件,反序列化就是从文件中写读到到程序中成为对应语言的数据类型.  
python数据类型与JSON格式的对照表:

| **Python** | **JSON** |
| --- | --- |
| dict | object |
| list,tuple | array |
| str,unicode | string |
| int,long,float | number |
| True | true |
| False | false |
| None | null |

**二、python中json库与函数**

python通过导入json库:import json 来解析操作与序列化JSON格式文件.  
主要通过两个函数  
json.dumps是将dict转化成str格式，json.loads是将str转化成dict格式。这一对是用来做编码或叫解析与反解析JSON数据.注意这里没有写入文件的功能,只是在python内部处理.因此这里我习惯叫做解析与反解析.  
代码:

import json

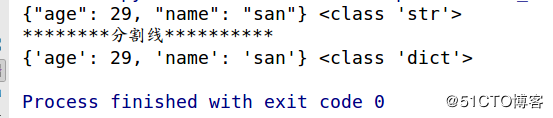
a = {'name': 'san', 'age': 29}

b = json.dumps(a)

print(b, type(b))

print("\*\*\*\*\*\*\*\*分割线\*\*\*\*\*\*\*\*\*\*")

print(json.loads(b),type(json.loads(b)))

输出结果:  
  
json.dump和json.load也是类似的功能，但是这两个函数需要文件对象结合操作。即序列化( Serialize)与反序列化(Deserialize)  
通过查看这对函数的帮助说明文档如下:

\*\*Help on function dump **in** module json:\*\*

dump(obj, fp, skipkeys=False, ensure\_ascii=True, check\_circular=True, allow\_nan=True, cls=None, indent=None, separators=None, encoding='utf-8', **default**=None, sort\_keys=False, \*\*kw)

\*\* Serialize\*\* ``obj`` **as** a JSON formatted stream to ``fp`` (a

``.write()``-supporting file-like object).

\*\*Help on function load **in** module json:\*\*

load(fp, encoding=None, cls=None, object\_hook=None, parse\_float=None, parse\_int=None, parse\_constant=None, object\_pairs\_hook=None, \*\*kw)

\*\* Deserialize \*\*``fp`` (a ``.read()``-supporting file-like object containing

a JSON document) to a Python object.

整明白了概念和常用的函数,接下来我们来看一个实例.

**三、python对JSON序列化与反序列化实战**

以下是JSON格式的文件(游戏配置文件)  
内容大致如下:

[

{

"phyId": 0,

"zoneId": 0,

"loadStatus": 0,

"isPublic": false,

"nameId": "user"

},

{

"phyId": 0,

"zoneId": 0,

"loadStatus": 3,

"isPublic": false,

"nameId": "misc"

},

{

"chName": "乱世枭雄",

"tag": "hunfu,ios",

"phyId": 1,

"zoneId": 1,

"localIP": "192.168.100.168",

"loadStatus": 3,

"url": [

"http://10.247.12.92/zhmjServer/"

],

"isPublic": true,

"nameId": "10001"

},

{

"chName": "倾国倾城",

"tag": "hunfu,ios",

"phyId": 1,

"zoneId": 2,

"localIP": "192.168.100.168",

"loadStatus": 3,

"url": [

"http://10.247.12.92/zhmjServer/"

],

"isPublic": true,

"nameId": "10012"

}

]

反序列化,从以下game.json中读取配置信息  
代码:

*#coding:utf-8*

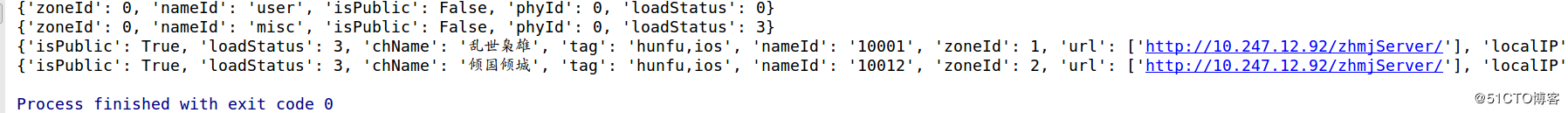
**import** json

**with** open("game.json") **as** game\_info:

games = json.load(game\_info, encoding="utf-8")

**for** i **in** games:

print(i)

运行结果:  


序列化:  
向game.json中更新写入JSON格式流或叫更新JSON文件  
例如向game.json中新增加一条游戏服配置:  
所谓更新思路就是把原有的数据反解析反序列化读出,再添加新的数据,解析序列化写回文件.

**def** **bakSrv**(NameId="",GameName="",zoneId=""):

localIP='192.168.100.168'

tag = 'hunfu,ios'

url = ['http://10.247.12.92/zhmjServer/']

GameSrv = {}

**with** open("game.json") **as** game\_info:

old\_games = json.load(game\_info,encoding="utf-8")

GameSrv["localIP"] = u"%s" % localIP

GameSrv["tag"] = u"%s" % tag

GameSrv["url"] = url

GameSrv["phyId"] = 1

GameSrv["nameId"] = str(NameId)

GameSrv["chName"] = u"%s" % GameName

GameSrv["isPublic"] = **False**

GameSrv["zoneId"] = int(zoneId)

GameSrv["loadStatus"] = 0

old\_games.append(GameSrv)

repr(old\_games)

f = open("game.json",'w+') *#创建写入文件对象*

json.dump(old\_games, f,ensure\_ascii=**False**,indent=4,separators=(',',': ')) *#*

f.close()

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*生成新的JSON配置文件成功!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

bakSrv(NameId="10003",GameName="新测试服",zoneId=4)

说明:  
json.dump的参数:  
注意,此处是dump函数的参数,indent是json文件生成后的缩进,默认None则输出的格式是在一行,很不美观,separators默认可不指定,可以修改成类似separators=(',\n',': \t') 如此就是加了:后的tab, ennsure\_ascii默认是True 如果不修改成False你的中文是unicode的,所以这里必须指定.

此时查看game.json文件时会发现已经新增加了一条,如图:  


**四、总结**

通过需要从一个JSON格式文件中load读出反序列化并通过dump生成或修改重新写入序列化到一个JSON格式的文件中.而loads与dumps主要是python内部进行解析.最终要写入到文件需要dump.或从文件中读取用load.  
用错了地方则会出错.