Python 程序设计基础

Python Programming

对象序列化

- 序列化是指将一个对象转换为能够存储在文件中或在网络上进行传输的字节流的过程。反序列化则相反，指的是从字节流中提取出对象的过程。
  - pickle 模块用于 Python 对象的序列化和反序列化。
  - Python 中的数据都是对象。使用 pickle 模块的 dump 和 load 函数读写任何数

```
# 对象输入输出
import pickle
with open("pickle.dat", "wb") as fo:
    pickle.dump(45, fo)
    pickle.dump(56.6, fo)
    pickle.dump("编程非常有意思", fo)
    pickle.dump([111, 222, 333, 444], fo)

with open("pickle.dat", "rb") as fo:
    print(pickle.load(fo))
    print(pickle.load(fo))
    print(pickle.load(fo))
    print(pickle.load(fo))
```

```
45
56.6
编程非常有意思
[111, 222, 333, 444]
```

对象序列化

- 使用 load 方法反复读取二进制文件中的数据，当到达文件末尾，会抛出 EOFError 异常。当抛出这个异常时，捕获并处理它可以结束文件读取过程。
- 若不知道文件中有多少数据，可以利用异常处理机制读取文件中的所有数据。

```
# 对象序列化，读取批量数据
import pickle
with open("pickle.dat", "rb") as fo:
    end_of_file = False
    while not end_of_file:
        try:
            print(pickle.load(fo), end=' ')
        except EOFError:
            end_of_file = True
print("\n所有数据都读取了")
```

```
45 56.6 编程非常有意思 [111, 222, 333, 444]
所有数据都读取了
```

对象序列化

- pickle 模块的对象序列化和反序列化只能用于 Python 。要在不同的编程语言之间传递对象，就必须把对象序列化为标准数据交换格式。
- 常用的标准数据交换格式：
  - 可扩展标记语言 XML （ Extensible Markup Language ）
  - JavaScript 对象标记 JSON （ JavaScript Object Notation ）。

对象序列化

- JSON 是一种轻量级的数据交换格式，采用完全独立于编程语言的文本格式来存储和表示数据，易于阅读和理解，可以被所有编程语言读取，方便存储在文件中或在网络上传输。比 XML 更简单、更快。
- JSON 格式如下：
  - 对象是一个无序的键 / 值对集合。一个对象以左花括号 { 开始，右花括号 } 结束。每个键后跟一个冒号，键 / 值对之间使用逗号分隔。
  - 数组是值的有序集合。一个数组以左方括号 [开始，右方括号] 结束。值之间使用逗号分隔。
  - 字符串必须用双引号，键也必须用双引号，字符集必须是 UTF-8 。

```
{
    "students": [
        {
            "name": "小明",
            "gender": true,
            "skills": ["C++", "Java"]
        },
        {
            "name": "小慧",
            "gender": false,
            "skills": ["JavaScript", "Python"]
        }
    ]
}
```

对象序列化

- Python 的 json 模块用于 Python 对象和 JSON 格式之间的序列化和反序列化。
- json 模块的 dumps 方法将 Python 对象序列化（编码）为 JSON 格式字符串。 loads 方法将 JSON 格式字符串反序列化（解码）为 Python 对象。
- 编码 / 解码过程中 的数据类型会相互转换。

Python 数据类型	JSON 数据类型
dict	object
list、tuple	array
str	string
int、float	number
True	true
False	false
None	null

对象序列化

```
# 对象序列化，JSON
import json
d1 = {"name": "小明", "gender": True, "skills": ["C++", "Java"]}
print(d1)
json_string = json.dumps(d1, ensure_ascii=False, indent=4)
print(json_string)
d2 = json.loads(json_string)
print(d2)
print(d1 == d2)
```

```
{'name': '小明', 'gender': True, 'skills': ['C++', 'Java']}
{
    "name": "小明",
    "gender": true,
    "skills": [
        "C++",
        "Java"
    ]
}
{'name': '小明', 'gender': True, 'skills': ['C++', 'Java']}
True
```

对象序列化

- dumps 方法的 ensure_ascii 参数指定如何处理非 ASCII 字符（如中文字符），默认值为 True ，非 ASCII 字符转换为 UTF-8 编码的字节码，修改为 False ，就可以正常显示非 ASCII 字符了；
- indent 参数指定缩进空格数，使得生成的 JSON 格式字符串更具有可读性；
- sort_keys 参数默认值为 False ，修改为 True ，将数据按键进行排序；
- separators 参数的作用是去掉逗号和冒号后面的空格。

对象序列化

```
# 对象序列化，JSON，压缩
import json
d = {'b':456, 'c':789, 'a':123}
json_string1 = json.dumps(d, sort_keys=True)
json_string2 = json.dumps(d, sort_keys=True, indent=4)
json_string3 = json.dumps(d, sort_keys=True, separators=(',', ':'))
print(json_string1, len(json_string1))
print(json_string2, len(json_string2))
print(json_string3, len(json_string3))
```

```
{"a": 123, "b": 456, "c": 789} 30
{
    "a": 123,
    "b": 456,
    "c": 789
} 44
{"a":123,"b":456,"c":789} 25
```

比较输出的 JSON 格式字符串的长度，可以发现通过移除多余的空格，达到了压缩数据的目的。

对象序列化

- json 模块的 dump 方法将 Python 对象序列化（编码）为 JSON 格式字符串，并存放在文件中。 load 方法将文件中的 JSON 格式字符串反序列化（解码）为 Python 对象。

```
# 对象序列化，JSON，反序列化
import json
d1 = {"name": "小明", "gender": True, "skills": ("C++", "Java")}
print(d1)
with open("json.txt", 'w') as fo:
    json.dump(d1, fo, ensure_ascii=False)
with open("json.txt", 'r') as fo:
    d2 = json.load(fo)
print(d2)
```

```
{'name': '小明', 'gender': True, 'skills': ('C++', 'Java')}
{'name': '小明', 'gender': True, 'skills': ['C++', 'Java']}
```

通过输出结果可以看出，解码后有些数据类型改变了，例如元组转换为了列表。

对象序列化

- json 模块可以直接序列化 Python 的内置数据类型。对于用户自定义类对象，则无法直接序列化，需要进行定制，否则会抛出 TypeError 异常。

```
import json
class Student:
    def __init__(self, name, gender, skills):
        self.name = name
        self.gender = gender
        self.skills = skills
s = Student("小明", True, ["C++", "Java"])
print(json.dumps(s))
```

```
TypeError: Object of type 'Student' is not JSON serializable
```

对象序列化

- Python 的内置数据类型与 JSON 的数据类型相互转换， object 类型是与 dict 类型相关联的。
- 把用户自定义类对象转换为 dict 类型对象，然后再进行处理。
  - 自定义一个转换函数 object_to_dict ，编码时利用 dumps 方法的 default 参数指定调用该函数，就可以序列化自定义类对象。
  - 解码时也需要自定义一个转换函数 dict_to_object ，利用 loads 方法的 object_hook 参数指定调用该函数，就可以反序列化自定义类对象。

对象序列化

```
# 自定义对象序列化，JSON
import json
class Student:
    def __init__(self, name, gender, skills):
        self.name = name
        self.gender = gender
        self.skills = skills
def object_to_dict(obj):
    return {"name":obj.name, "gender":obj.gender, "skills":obj.skills}
def dict_to_object(obj):
    return Student(obj["name"], obj["gender"], obj["skills"])
s = Student("小明", True, ["C++", "Java"])
json_string = json.dumps(s, ensure_ascii=False, default=object_to_dict)
print(json_string)
d = json.loads(json_string,object_hook=dict_to_object)
print(d.name, d.gender, d.skills)
```

```
{"name": "小明", "gender": true, "skills": ["C++", "Java"]}
小明 True ['C++', 'Java']
```