22. 休养生息-常用模块-02

本节主要内容

1. 什么是序列化
2. pickle(重点)
3. shelve
4. json(重点)
5. configparser模块

一. 什么是序列化

在我们存储数据或者网络传输数据的时候. 需要对我们的对象进行处理. 把对象处理成方便存储和传输的数据格式. 这个过程叫序列化. 不同的序列化, 结果也不同. 但是目的是一样的. 都是为了存储和传输.

在python中存在三种序列化的方案.

1. pickle. 可以将我们python中的任意数据类型转化成bytes并写入到文件中. 同样也可以把文件中写好的bytes转换回我们python的数据. 这个过程被称为反序列化
2. shelve. 简单另类的一种序列化的方案. 有点儿类似后面我们学到的redis. 可以作为一种小型的数据库来使用
3. json. 将python中常见的字典, 列表转化成字符串. 是目前前后端数据交互使用频率最高的一种数据格式.

二. pickle(重点)

pickle用起来很简单. 说白了. 就是把我们的python对象写入到文件中的一种解决方案. 但是写入到文件的是bytes. 所以这东西不是给人看的. 是给机器看的.

```
import pickle

class Cat:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def catchMouse(self):
        print(self.name, "抓老鼠")


c = Cat("jerry", 18)

bs = pickle.dumps(c) # 序列化一个对象.
print(bs) # 一堆二进制. 看不懂

cc = pickle.loads(bs) # 把二进制反序列化成我们的对象
cc.catchMouse() # 猫依然是猫. 还可以抓老鼠
```

pickle中的dumps可以序列化一个对象. loads可以反序列化一个对象. 我们使用dump还可以直接 把一个对象写入到文件中

```
# f = open("cat", mode="wb")
# pickle.dump(c, f)   # 写入到文件中
# f.close()

f = open("cat", mode="rb")
cc = pickle.load(f) # 从文件中读取对象
cc.catchMouse()
```

pickle还支持多个对象的写出.

```
lst = [Cat("jerry", 19), Cat("tommy", 20), Cat("alpha", 21)]

f = open("cat", mode="wb")
for el in lst:
    pickle.dump(el, f)   # 写入到文件中
f.close()

f = open("cat", mode="rb")
for i in range(len(lst)):
    cc = pickle.load(f) # 从文件中读取对象
    cc.catchMouse()
```

但是这样写并不够好. 因为读的时候. 并不能知道有多少对象要读. 这里记住, 不能一行一行的读. 那真的要写入或者读取多个内容怎么办? 很简单. 装list里. 然后读取和写入都用list

```
lst = [Cat("jerry", 19), Cat("tommy", 20), Cat("alpha", 21)]

f = open("cat", mode="wb")
pickle.dump(lst, f)

f = open("cat", mode="rb")
ll = pickle.load(f)
for el in ll:
    el.catchMouse()
```

记住一点, pickle序列化的内容是二进制的内容(bytes) 不是给人看的.

三. shelve

shelve提供python的持久化操作. 什么叫持久化操作呢? 说白话,就是把数据写到硬盘上. 在操作shelve的时候非常的像操作一个字典. 这个东西到后期. 就像redis差不多.

```
import shelve

shelf = shelve.open("sylar")
# shelf["jay"] = "周杰伦"
print(shelf['jay'])
shelf.close()
```

感觉到了么. 这个鬼东西和字典差不多. 只不过你的字典是一个文件. 接下来, 我们存储一些复杂的数据

```
s = shelve.open("sylar")
# s["jay"] = {"name":"周杰伦", "age":18, "hobby":"哄小孩"}
print(s['jay'])
s.close()
```

但是, 有坑

```
s = shelve.open("sylar")
s['jay']['name'] = "胡辣汤"     # 尝试改变字典中的数据
s.close()

s = shelve.open("sylar")
print(s['jay']) # 并没有改变
s.close()
```

解决方案

```
s = shelve.open("sylar", writeback=True)
s['jay']['name'] = "胡辣汤"     # 尝试改变字典中的数据
s.close()

s = shelve.open("sylar")
print(s['jay']) # 改变了.
s.close()
```

writeback=True可以动态的把我们修改的信息写入到文件中. 而且这个鬼东西还可以删除数据. 就像字典一样. 上一波操作

```
s = shelve.open("sylar", writeback=True)
del s['jay']
s.close()

s = shelve.open("sylar")
print(s['jay']) # 报错了, 没有了
s.close()


s = shelve.open("sylar", writeback=True)
s['jay'] = "周杰伦"
s['wlj'] = "王力宏"
s.close()

s = shelve.open("sylar")
for k in s: # 像字典一样遍历
    print(k)
print(s.keys()) # 拿到所有key的集合
for k in s.keys():
    print(k)
```

```
for k, v in s.items(): # 像字典一样操作
    print(k, v)
s.close()
```

综上shelve就当成字典来用就行了. 它比redis还简单.....

四. json(重点)

终于到json了. json是我们前后端交互的枢纽. 相当于编程界的普通话. 大家沟通都用json. 为什么这样呢? 因为json的语法格式可以完美的表示出一个对象. 那什么是json: json全称javascript object notation. 翻译过来叫js对象简谱. 很复杂是吧? 来上一段我们认识的代码:

```
wf = {
    "name":"汪峰",
    "age":18,
    "hobby":"上头条",
    "wife":{
        "name":'子怡',
        "age":19,
        "hobby":["唱歌", "跳舞", "演戏"]
    }
}
```

这个不是字典么? 对的. 在python里这玩意叫字典. 但是在javascript里这东西叫json. 一模一样的. 我们发现用这样的数据结构可以完美的表示出任何对象. 并且可以完整的把对象表示出来. 只要代码格式比较好. 那可读性也是很强的. 所以大家公认用这样一种数据结构作为数据交互的格式. 那在这个鬼东西之前是什么呢? XML.....来看一段代码

```
<?xml version="1.0" encoding="utf-8" ?>
<wf>
    <name>汪峰</name>
    <age>18</age>
    <hobby>上头条</hobby>
    <wife>
        <name>子怡</name>
        <age>18</age>
        <hobbies>
            <hobby>唱歌</hobby>
            <hobby>跳舞</hobby>
            <hobby>演戏</hobby>
        </hobbies>
    </wife>

</wf>
```

古人(老程序员)都是用这样的数据进行传输的. 先不管这个东西好不好看. 这玩意想要解析.. 那简直了. 想死的心都有. 所以老版本的xml在维护和处理上是非常复杂和繁琐的. 多说一嘴, 就是因为这个鬼东西太难解析. 以前的项目几乎没有用ajax的.

OK. 那json既然这么牛B好用. 怎么用呢? 注意. 这里又出来一个新问题. 我们的程序是在python里写的. 但是前端是在JS那边来解析json的. 所以. 我们需要把我们程序产生的字典转化成json格式的json串(字符串). 然后网络传输. 那边接收到了之后. 它爱怎么处理是它的事情. 那, 如何把字典转化成我们的json格式的字符串呢?很简单, 上代码.

```
import json
dic = {"a": "女王", "b": "萝莉", "c": "小清新"}
s = json.dumps(dic) # 把字典转化成json字符串
print(s)  # {"a": "\u5973\u738b", "b": "\u841d\u8389", "c":
"\u5c0f\u6e05\u65b0"}
```

结果很不友好啊. 那如何处理中文呢? 在dumps的时候给出另一个参数ensure_ascii=False就可以了

```
import json
dic = {"a": "女王", "b": "萝莉", "c": "小清新"}
s = json.dumps(dic, ensure_ascii=False) # 把字典转化成json字符串
print(s)  # {"a": "女王", "b": "萝莉", "c": "小清新"}
```

搞定了. 接下来. 前端给你传递信息了. 你要把前端传递过来的json字符串转化成字典.

```
import json


s = '{"a": "女王", "b": "萝莉", "c": "小清新"}'
dic = json.loads(s)
print(type(dic), dic)
```

搞定. 是不是很简单. 以上两个代码要求. 记住, 理解, 背会

json也可以像pickle一样把序列化的结果写入到文件中.

```
dic = {"a": "女王", "b": "萝莉", "c": "小清新"}
f = open("test.json", mode="w", encoding="utf-8")
json.dump(dic, f, ensure_ascii=False) # 把对象打散成json写入到文件中
f.close()
```

同样也可以从文件中读取一个json

```
f = open("test.json", mode="r", encoding="utf-8")
dic = json.load(f)
f.close()
print(dic)
```

注意. 我们可以向同一个文件中写入多个json串. 但是读不行.

```
import json

lst = [{"a": 1}, {"b": 2}, {"c": 3}]

f = open("test.json", mode="w", encoding="utf-8")
```

```
for el in lst:
    json.dump(el, f)
f.close()
```

注意，此时文件中的内容是一行内容.

```
{"a": 1}{"b": 2}{"c": 3}
```

这在读取的时候是无法正常读取的. 那如何解决呢? 两套方案. 方案一. 把所有的内容准备好统一进行写入和读取. 但这样处理，如果数据量小还好. 数据量大的话，就不够友好了. 方案二. 不用dump. 改用dumps和loads. 对每一行分别进行处理.

```
import json

lst = [{"a": 1}, {"b": 2}, {"c": 3}]

# 写入
f = open("test.json", mode="w", encoding="utf-8")
for el in lst:
    s = json.dumps(el, ensure_ascii=True) + "\n"
    f.write(s)
f.close()

# 读取
f = open("test.json", mode="r", encoding="utf-8")
for line in f:
    dic = json.loads(line.strip())
    print(dic)
f.close()
```

五. configparser模块

该模块适用于配置文件的格式与windows ini文件类似，可以包含一个或多个节(section)每个节可以有多个参数(键=值). 首先，我们先看一个xxx服务器的配置文件

```
[DEFAULT]
ServerAliveInterval = 45
Compression = yes
CompressionLevel = 9
ForwardX11 = yes

[bitbucket.org]
User = hg

[topsecret.server.com]
Port = 50022
ForwardX11 = no
```

我们用configparser就可以对这样的文件进行处理.首先，是初始化

```
import configparser

config = configparser.ConfigParser()
config['DEFAULT'] = {
```

```
    "sleep": 1000,
    "session-time-out": 30,
    "user-alive": 999999
}

config['TEST-DB'] = {
    "db_ip": "192.168.17.189",
    "port": "3306",
    "u_name": "root",
    "u_pwd": "123456"
}

config['168-DB'] = {
    "db_ip": "152.163.18.168",
    "port": "3306",
    "u_name": "root",
    "u_pwd": "123456"
}

config['173-DB'] = {
    "db_ip": "152.163.18.173",
    "port": "3306",
    "u_name": "root",
    "u_pwd": "123456"
}


f = open("db.ini", mode="w")
config.write(f) # 写入文件
f.flush()
f.close()
```

读取文件信息：

```
config = configparser.ConfigParser()

config.read("db.ini") # 读取文件
print(config.sections()) # 获取到section. 章节...DEFAULT是给每个章节都配备的信息
print(config.get("DEFAULT", "SESSION-TIME-OUT")) # 从xxx章节中读取到xxx信息
# 也可以像字典一样操作
print(config["TEST-DB"]['DB_IP'])
print(config["173-DB"]["db_ip"])

for k in config['168-DB']:
    print(k)

for k, v in config["168-DB"].items():
    print(k, v)


print(config.options('168-DB'))  # 同for循环,找到'168-DB'下所有键
print(config.items('168-DB'))    #找到'168-DB'下所有键值对
```

```
print(config.get('168-DB','db_ip')) # 152.163.18.168       get方法Section下的key对应的value
```

增删改操作：

```
# 先读取. 然后修改. 最后写回文件
config = configparser.ConfigParser()
config.read("db.ini") # 读取文件

# 添加一个章节
# config.add_section("189-DB")
# config["189-DB"] = {
#     "db_ip": "167.76.22.189",
#     "port": "3306",
#     "u_name": "root",
#     "u_pwd": "123456"
# }

# 修改信息
config.set("168-DB", "db_ip", "10.10.10.168")


# 删除章节
config.remove_section("173-DB")
# 删除元素信息
config.remove_option("168-DB", "u_name")

# 写回文件
config.write(open("db.ini", mode="w"))
```