[](tencent://AddContact/?fromId=50&fromSubId=1&subcmd=all&uin=3440037784)[](https://jq.qq.com/?_wv=1027&k=50Xp8mW)

扫码获取更多课程资料，开通直播上课权限(点击下方课程直播即可免费观看）

学神IT教育微信公众号 学神IT官方 QQ群

关注学神公众号有机会获得Python官方书籍

[](https://jq.qq.com/?_wv=1027&k=50Xp8mW)

# 第二十二章 Python字符处理编程

本节所讲内容：

22.1 python json介绍

22.2 Python json数据封装

22.3 Python hashlib模块

22.4 Python base64

## 22.1 python json介绍

### 22.1.1 json介绍

什么是json？

Json：是一种轻量级的数据交换格式，易于人阅读和编写。就是相当于我们的python中类字典的。大白话就是复杂一点的字典。



一种数据格式如：

data = {  
 "name":"For",  
 "age":18  
}

上面这种就是一种比较典型的json格式。

注意：JSON定死了字符集必须是UTF-8，表示多语言就没有问题了。为了统一解析，JSON的字符串规定必须用双引号""，Object的键也必须用双引号""

JSON建构于两种结构：

1:“名称/值”对的集合（A collection of name/value pairs）。不同的语言中，它被理解为对象（object），纪录（record），结构（struct），字典（dictionary），哈希表（hash table），有键列表（keyed list），或者关联数组 （associative array）。   
 2: 值的有序列表（An ordered list of values）。在大部分语言中，它被理解为数组（array）。   
这些都是常见的数据结构。事实上大部分现代计算机语言都以某种形式支持它们。这使得一种数据格式在同样基于这些结构的编程语言之间交换成为可能。

## 22.2 json的数据封装

### 22.2.1 json的数据封装

Python中使用json需要导入json库，import json 来使用json

在json中有2个函数是我们常用的函数json.dumps()以及json.loads()这个函数，下面我们分别讲一下这2个函数的使用方法：

Json.dumps():将 Python 对象编码成 JSON 字符串

语法结构：

json.dumps(obj, skipkeys=False, check\_circular=True, allow\_nan=True, cls=None, indent=None, separators=None, encoding="utf-8", default=None, sort\_keys=False, \*\*kw)

Skipkeys：默认值是False，如果dict的keys内的数据不是python的基本类（[str](https://docs.python.org/2/library/functions.html#str),[unicode](https://docs.python.org/2/library/functions.html#unicode),[int](https://docs.python.org/2/library/functions.html#int) ,[float](https://docs.python.org/2/library/functions.html" \l "float" \o "float),[bool](https://docs.python.org/2/library/functions.html#bool),None)，设置为False时，就会报TypeError的错误。此时设置成True，则会跳过这类key

indent：应该是一个非负的整型，如果是0，或者为空，则一行显示数据，否则会换行且按照indent的数量显示前面的空白，这样打印出来的json数据也叫pretty-printed json

separators：分隔符，实际上是(item\_separator, dict\_separator)的一个元组，默认的就是(',',':')；这表示dictionary内keys之间用“,”隔开，而KEY和value之间用“：”隔开。

encoding：默认是UTF-8，设置json数据的编码方式。

sort\_keys：将数据根据keys的值进行排序。

示例：

import json  
data = {  
 "name":"For",  
 "age":18,  
 'data':[1,2,3]  
}  
*#利用json的dunmps方法把数据转换为一个json字符串*json\_data = json.dumps(data)  
print(type(json\_data))

结果：

{"age": 18, "name": "For", "data": [1, 2, 3]}

总结：就是把python的字段格式转换为json格式，使用dumps（）

python 原始类型向 json 类型的转化对照表：

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

那么还有另一个函数json.loads()这个函数用于解码 JSON 数据。该函数返回 Python 字段的数据类型。它正好是dumps的方法是相反，是把json数据转换为python可以使用的数据如下面的代码展示。

结果输出：{'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4}

如果你要处理是文件，那么你需要使用dump（）和load（）来处理数据如：

# 写入 JSON 数据

import json  
data = {"age": 18, 'name': "For", "data": [1, 2, 3]}  
with open('data.json','w',encoding='utf-8') as f:  
 json.dump(data,f)

# 读取数据

import json  
data = {"age": 18, 'name': "For", "data": [1, 2, 3]}  
with open('data.json', 'r',encoding='utf-8') as f:  
 text = json.load(f)

22.2.2 最新的json的模块：demjson

在我们使用json解析中可能会遇到不是标准的json格式，比如：{0:"000001\_Unit\_1. Hi,Birdie.mp3",1:"000005\_Unit\_2. Good morning,Miss Wang..mp3",2:"000008\_Unit\_3. What's your name\_.mp3"}细心的朋友会看出来这里“键”没有引号 只有“值”有引号。

那么怎么办呢？这时我们就可以使用 demjson来解析，那么demjson又是哪一个模块呢？，demjson是python的第三方模块，可用于编码和解码 JSON 数据，包含了 JSONLint 的格式化及校验功能。

- encode 将 Python 对象编码成 JSON 字符串   
- decode 将已编码的 JSON 字符串解码为 Python 对象

demjson模块提供用于编码或解码用语言中性JSON格式表示的数据的类和函数（这在ajax Web应用程序中通常被用作XML的简单替代品）。

此实现尽量尽可能遵从JSON规范（RFC 4627），同时仍然提供许多可选扩展，以允许较少限制的JavaScript语法。

它包括完整的Unicode支持，包括UTF-32，BOM，和代理对处理。它还可以支持JavaScript的南方和无穷数字类型以及它的“未定义”类型。它还包括一个像JSON语法验证器测试对于严格遵守标准的JSON文本。

安装：pip3 install demjson

示例：

import demjson  
s = '{a:"000001\_Unit\_1. Hi,Birdie.mp3",b:"000005\_Unit\_2. Good morning,Miss Wang..mp3",c:"000008\_Unit\_3. What\'s your name\_.mp3"}'  
data1 = demjson.decode(s)  
print(data1)  
print(type(data1))  
data2 = demjson.encode(data1)  
print(data2)  
print(type(data2))

运行结果：

{'c': "000008\_Unit\_3. What's your name\_.mp3", 'b': '000005\_Unit\_2. Good morning,Miss Wang..mp3', 'a': '000001\_Unit\_1. Hi,Birdie.mp3'}

<class 'dict'>

{"a":"000001\_Unit\_1. Hi,Birdie.mp3","b":"000005\_Unit\_2. Good morning,Miss Wang..mp3","c":"000008\_Unit\_3. What's your name\_.mp3"}

<class 'str'>

其实解析json的模块有很多，比如django中就有一个单独的json解析模块：simplejson这个我们可以在后面详细的讲解，最主要就是看我们如何的去使用它，让他可以成为我们编程路上的帮手，而不是一味见到一个模块就会去使用，我们要合理的利用各个模块.

那么我在使用一个示例如：a = {

‘data’: {

‘name’ : ‘xiaoming’,

‘age’ : ‘20’

}

}

这样的数据，我们应该怎么封装到json里面呢？

import json  
a = {  
 'data': {  
 'name' : 'xiaoming',  
 'age' : '20'  
 }  
}  
jsons = json.dumps(a)  
print(jsons)

## 22.3 Python hashlib模块

### 22.3.1 hashlib模块

介绍：Python的hashlib提供了常见的摘要算法，如MD5，SHA1等等。

什么是摘要算法？

摘要算法又称哈希算法、散列算法。它通过一个函数，把任意长度的数据转换为一个长度固定的数据串（通常用16进制的字符串表示）。

举个例子，你写了一篇文章，内容是一个字符串'how to use python hashlib - by Michael'，并附上这篇文章的摘要是'2d73d4f15c0db7f5ecb321b6a65e5d6d'。如果有人篡改了你的文章，并发表为'how to use python hashlib - by Bob'，你可以一下子指出Bob篡改了你的文章，因为根据'how to use python hashlib - by Bob'计算出的摘要不同于原始文章的摘要。

可见，摘要算法就是通过摘要函数f()对任意长度的数据data计算出固定长度的摘要digest，目的是为了发现原始数据是否被人篡改过。

摘要算法之所以能指出数据是否被篡改过，就是因为摘要函数是一个单向函数，计算f(data)很容易，但通过digest反推data却非常困难。而且，对原始数据做一个bit的修改，都会导致计算出的摘要完全不同。

常用的属性和方法：

algorithms：列出所有加密算法（'md5','sha1','sha224','sha256','sha384','sha512'）  
digesti\_size：产生的散列的字节大小  
md5()/sha1()：创建一个md5或者sha1加密模式的hash对象  
update(arg)：用字符串参数来更新hash对象，如果同一个hash对象重复调用该方法，如下：m.update(a); m.update(b)，则等于m.update(a+b)  
digest()：返回摘要，作为二进制数据字符串值  
hexdigest()：返回摘要，作为十六进制数据字符串值  
copy()：复制

我们以常见的摘要算法MD5为例，计算出一个字符串的MD5值：

import hashlib  
md5 = hashlib.md5()  
md5.update('how to use md5 in python hashlib?'.encode())  
print(md5.hexdigest())

计算结果如下：

d26a53750bc40b38b65a520292f69306

如果数据量很大，可以分块多次调用update()，最后计算的结果是一样的：

import hashlib  
md5 = hashlib.md5()  
md5.update('how to use md5 in '.encode())  
md5.update('python hashlib?'.encode())  
print (md5.hexdigest())

如果数据量很大，可以分块多次调用update()，最后计算的结果是一样的：

import hashlib  
md5 = hashlib.md5()  
md5.update('how to use md5 in '.encode())  
md5.update('python hashlib?'.encode())  
print (md5.hexdigest())

另一种常见的摘要算法是SHA1，调用SHA1和调用MD5完全类似：

import hashlib  
sha1 = hashlib.sha1()  
sha1.update('how to use sha1 in '.encode())  
sha1.update('python hashlib?'.encode())  
print (sha1.hexdigest())

SHA1的结果是160 bit字节，通常用一个40位的16进制字符串表示。

比SHA1更安全的算法是SHA256和SHA512，不过越安全的算法越慢，而且摘要长度更长。

有没有可能两个不同的数据通过某个摘要算法得到了相同的摘要？完全有可能，因为任何摘要算法都是把无限多的数据集合映射到一个有限的集合中。这种情况称为碰撞，比如Bob试图根据你的摘要反推出一篇文章'how to learn hashlib in python - by Bob'，并且这篇文章的摘要恰好和你的文章完全一致，这种情况也并非不可能出现，但是非常非常困难。

### 22.3.2 摘要算法应用

摘要算法能应用到什么地方？举个常用例子：

任何允许用户登录的网站都会存储用户登录的用户名和口令。如何存储用户名和口令呢？方法是存到数据库表中：

name | password

michael | 123456

bob | abc999

alice | alice2008

如果以明文保存用户口令，如果数据库泄露，所有用户的口令就落入黑客的手里。此外，网站运维人员是可以访问数据库的，也就是能获取到所有用户的口令。

正确的保存口令的方式是不存储用户的明文口令，而是存储用户口令的摘要，比如MD5：

username | password*---------+---------------------------------*

michael | e10adc3949ba59abbe56e057f20f883e

bob | 878ef96e86145580c38c87f0410ad153

alice | 99b1c2188db85afee403b1536010c2c9

当用户登录时，首先计算用户输入的明文口令的MD5，然后和数据库存储的MD5对比，如果一致，说明口令输入正确，如果不一致，口令肯定错误。

设计一个用户的登录验证：

接受用户的登录信息：

import hashlib  
#模拟数据库  
user = {'name':'bob','password': 'e10adc3949ba59abbe56e057f20f883e'}  
class User\_login(object):  
 def \_\_init\_\_(self,*name*,*password*):  
 self.name = *name* self.password = *password* def login(self):  
 md5= hashlib.md5()  
 md5.update(self.password.encode('utf-8'))  
 password = md5.hexdigest()  
 if self.name == user['name'] and user['password'] == password:  
 print('密码正确')  
 return ''  
 else:  
 print('账号密码不正确')  
 return ''

if \_\_name\_\_ == ‘\_\_main\_\_’:  
name = input('请输入姓名')  
password = input('请输入密码')  
user\_login = User\_login(name,password)  
user\_login.login()

采用MD5存储口令是否就一定安全呢？也不一定。假设你是一个黑客，已经拿到了存储MD5口令的数据库，如何通过MD5反推用户的明文口令呢？暴力破解费事费力，真正的黑客不会这么干。

考虑这么个情况，很多用户喜欢用123456，888888，password这些简单的口令，于是，黑客可以事先计算出这些常用口令的MD5值，得到一个反推表：

'e10adc3949ba59abbe56e057f20f883e': '123456'

'21218cca77804d2ba1922c33e0151105': '888888'

'5f4dcc3b5aa765d61d8327deb882cf99': 'password'

这样，无需破解，只需要对比数据库的MD5，黑客就获得了使用常用口令的用户账号。

对于用户来讲，当然不要使用过于简单的口令。但是，我们能否在程序设计上对简单口令加强保护呢？

由于常用口令的MD5值很容易被计算出来，所以，要确保存储的用户口令不是那些已经被计算出来的常用口令的MD5，这一方法通过对原始口令加一个复杂字符串来实现，俗称“加盐”：

def salt\_md5(password):

return get\_md5(password + 'the-Salt')

import hashlib  
*#模拟数据库*user = {'name':'bob','password': '207acd61a3c1bd506d7e9a4535359f8a'}  
class User\_login(object):  
 def \_\_init\_\_(self,name,password):  
 self.name = name  
 self.password = password  
 def salt\_md5(self,password):  
 return password+'salt'  
 def login(self):  
 md5= hashlib.md5()  
 md5.update(self.salt\_md5(self.password).encode())  
 password = md5.hexdigest()  
 if self.name == user['name'] and user['password'] == password:  
 print('密码正确')  
 return ''  
 else:  
 print('账号密码不正确')  
 return ''  
if \_\_name\_\_ == '\_\_main\_\_':  
 name = input('请输入姓名:')  
 password = input('请输入密码:')  
 user\_login = User\_login(name,password)  
 user\_login.login()

经过Salt处理的MD5口令，只要Salt不被黑客知道，即使用户输入简单口令，也很难通过MD5反推明文口令。

但是如果有两个用户都使用了相同的简单口令比如123456，在数据库中，将存储两条相同的MD5值，这说明这两个用户的口令是一样的。有没有办法让使用相同口令的用户存储不同的MD5呢？

如果假定用户无法修改登录名，就可以通过把登录名作为Salt的一部分来计算MD5，从而实现相同口令的用户也存储不同的MD5。

### 22.3.3 小结

1、什么叫hash:hash是一种算法（不同的hash算法只是复杂度不一样）（3.x里代替了md5模块和sha模块，主要提供 SHA1, SHA224, SHA256, SHA384, SHA512 ，MD5 算法），该算法接受传入的内容，经过运算得到一串hash值

2、hash值的特点是(hash值/产品有三大特性：)：

2.1 只要传入的内容一样，得到的hash值必然一样=====>要用明文传输密码文件完整性校验

2.2 不能由hash值返解成内容=======》把密码做成hash值，不应该在网络传输明文密码（只能有内容返回hash值）

2.3 只要使用的hash算法不变，无论校验的内容有多大，得到的hash值长度是固定的(如从网上下载文件要进行hash校验，保证网络传输没有丢包)

基于2.1和2.3可以做文件下载一致性的校验

基于2.1和2.2可以对用户密码进行加密

hash算法就像一座工厂，工厂接收你送来的原材料（可以用m.update()为工厂运送原材料），经过加工返回的产品就是hash值。

## 22.4 python base64

### 22.4.1 什么是base64

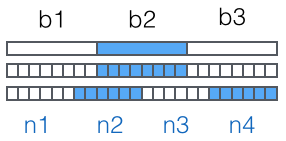
Base64是一种用64个字符来表示任意二进制数据的方法。

用记事本打开exe、jpg、pdf这些文件时，我们都会看到一大堆乱码，因为二进制文件包含很多无法显示和打印的字符，所以，如果要让记事本这样的文本处理软件能处理二进制数据，就需要一个二进制到字符串的转换方法。Base64是一种最常见的二进制编码方法。

Base64的原理很简单，首先，准备一个包含64个字符的数组：

['A', 'B', 'C', ... 'a', 'b', 'c', ... '0', '1', ... '+', '/']

然后，对二进制数据进行处理，每3个字节一组，一共是3x8=24bit，划为4组，每组正好6个bit：



这样我们得到4个数字作为索引，然后查表，获得相应的4个字符，就是编码后的字符串。

所以，Base64编码会把3字节的二进制数据编码为4字节的文本数据，长度增加33%，好处是编码后的文本数据可以在邮件正文、网页等直接显示。

如果要编码的二进制数据不是3的倍数，最后会剩下1个或2个字节怎么办？Base64用\x00字节在末尾补足后，再在编码的末尾加上1个或2个=号，表示补了多少字节，解码的时候，会自动去掉。

### 22.4.2 python内置的base64模块的作用

Python内置的base64可以直接进行base64的编解码：

import base64  
data = base64.b64encode('binary\x00string'.encode()) *#'YmluYXJ5AHN0cmluZw=='*print(data.decode())  
  
de\_data = base64.b64decode('YmluYXJ5AHN0cmluZw==') *#'binary\x00string'*print(de\_data.decode())

运行结果如下：

YmluYXJ5AHN0cmluZw==

binary string

base64模块提供了6个函数用于Base64的编码和解码，可以将他们分为三组。

base64.b64encode(s, altchars=None)   
base64.b64decode(s, altchars=None, validate=False)

参数s代表需要编码/解码的数据。其中b64encode的参数s的类型必须是字节包（bytes）。b64decode的参数s可以是字节包（bytes），也可以是字符串（str）。

由于Base64编码后的数据中可能会含有’+’或者’/’两个符号，如果编码后的数据用于url或者文件系统的路径中，就可能会导致Bug。所以base64模块提供了将编码后的数据中’+’和’/’进行替换的方法。

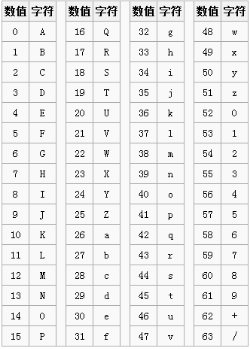
参数altchars必须是长度为2的字节包，这两个符号会用于替换编码后数据中的’+’和’/’。这个参数默认是None。

参数validate默认为False。如果它为True时，base64模块在进行解码前会先检查s中是否有非base64字母表中的字符，如果有的话则抛出错误bin.Error: Non-base64 digit found。

如果数据的长度不正确则会抛出错误bin.Error: Incorrect padding。

Base64字符串的原理：众所周知，传统的一个字节占8位，base64是将原先的二进制字节流以6位为分割，然后在经过base64字符编码对照表，编码为可打印的字符串。

base64字符编码对照表：



### 22.4.3 自定义base64字符串

可以自己定义64个字符的排列顺序，这样就可以自定义Base64编码，不过，通常情况下完全没有必要。

Base64是一种通过查表的编码方法，不能用于加密，即使使用自定义的编码表也不行。

Base64适用于小段内容的编码，比如数字证书签名、Cookie的内容等。

由于=字符也可能出现在Base64编码中，但=用在URL、Cookie里面会造成歧义，所以，很多Base64编码后会把=去掉：

# 标准Base64:'abcd' -> 'YWJjZA=='# 自动去掉=:'abcd' -> 'YWJjZA'

去掉=后怎么解码呢？因为Base64是把3个字节变为4个字节，所以，Base64编码的长度永远是4的倍数，因此，需要加上=把Base64字符串的长度变为4的倍数，就可以正常解码了。

### 22.4.4 base64编码解码的应用

Base64编码是一种“防君子不防小人”的编码方式。广泛应用于MIME协议，作为电子邮件的传输编码，生成的编码可逆，后一两位可能有“=”，生成的编码都是ascii字符。

它不适用与一些关键的信息的场合，比如用户的账号密码等信息

在实际应用中多数用来储存url，cookice和网页中传输少量的二进制数据，如存储图片的信息。

总结：

22.1 python json介绍

22.2 Python json数据封装

22.3 Python hashlib模块

22.4 Python base64