

【Python】python讀取json

在 Python 中，JSON 是一種常用的資料格式，用來儲存和交換資料。Python 提供內建的 `json` 模組來讀取與處理 JSON 資料，支援從檔案或字串中解析 JSON，以及將資料轉換成 JSON 格式輸出。

以下是 Python 中讀取 JSON 的相關用法：

1. Python 的 `json` 模組介紹

Python 的 `json` 模組提供以下幾個主要功能：

- `json.load()`：從檔案中讀取 JSON 資料。
- `json.loads()`：從字串中解析 JSON 資料。
- `json.dump()`：將 Python 資料寫入檔案，並格式化為 JSON。
- `json.dumps()`：將 Python 資料轉換為 JSON 格式的字串。

2. 讀取 JSON 的方式

2.1 從檔案讀取 JSON

假設有一個 `data.json` 檔案，內容如下：

```
{
  "name": "Alice",
  "age": 25,
  "skills": ["Python", "JavaScript"]
}
```

用法：

```
import json

# 打開 JSON 檔案並讀取
with open('data.json', 'r', encoding='utf-8') as file:
    data = json.load(file)

# 輸出讀取的資料
print(data)          # {'name': 'Alice', 'age': 25, 'skills': ['Python', 'JavaScript']}
print(data['name'])  # Alice
```

2.2 從字串解析 JSON

如果 JSON 資料是以字串的形式存在，可以使用 `json.loads()` 來解析：

```
import json

# JSON 字串
json_string = '{"name": "Bob", "age": 30, "skills": ["Java", "C++"]}'

# 解析 JSON 字串
data = json.loads(json_string)

# 輸出讀取的資料
print(data)          # {'name': 'Bob', 'age': 30, 'skills': ['Java', 'C++']}
print(data['skills']) # ['Java', 'C++']
```

3. JSON 資料寫入檔案

3.1 寫入 JSON 到檔案

使用 `json.dump()` 將 Python 資料寫入 JSON 檔案：

```
import json

# Python 資料
data = {
    "name": "Charlie",
    "age": 35,
    "skills": ["Go", "Rust"]
}

# 寫入 JSON 檔案
with open('output.json', 'w', encoding='utf-8') as file:
    json.dump(data, file, ensure_ascii=False, indent=4)

print("JSON 資料已成功寫入檔案")
```

- **`ensure_ascii=False`**：避免非 ASCII 字元（例如中文）被編碼為 Unicode（如 `\u4e2d\u6587`）。
- **`indent=4`**：設定縮排，讓輸出的 JSON 更易於閱讀。

輸出的檔案內容：

```
{
  "name": "Charlie",
  "age": 35,
  "skills": [
    "Go",
    "Rust"
  ]
}
```

3.2 將 Python 資料轉換為 JSON 字串

使用 `json.dumps()` 將資料轉換為 JSON 格式字串：

```
import json

# Python 資料
data = {
    "name": "Daisy",
    "age": 28,
    "skills": ["HTML", "CSS"]
}

# 轉換為 JSON 字串
json_string = json.dumps(data, ensure_ascii=False, indent=2)
print(json_string)
```

輸出：

```
{
  "name": "Daisy",
  "age": 28,
  "skills": [
    "HTML",
    "CSS"
  ]
}
```

4. 常見的 JSON 資料類型對應

Python 與 JSON 的資料類型有一對一的對應關係：

JSON 類型	Python 類型
Object	Dictionary (dict)
Array	List (list)
String	String (str)
Number (int/float)	Integer (int)/Float (float)
Boolean	Boolean (bool)
null	None

範例：

```
json_string = '''
{
    "name": "Eve",
    "age": 40,
    "is_active": true,
    "children": null
}
'''

data = json.loads(json_string)
print(type(data))          # <class 'dict'>
print(data['is_active'])    # True (Python 的布林值)
print(data['children'])     # None (Python 的 None)
```

5. 捕捉 JSON 解析錯誤

處理 JSON 時，可能會遇到格式不正確的情況，可以透過 try-except 捕捉例外：

```
import json

json_string = '{"name": "Frank", "age": 45,' # 少了結尾的 }

try:
    data = json.loads(json_string)
except json.JSONDecodeError as e:
    print(f"JSON 解析錯誤: {e}")
```

輸出：

```
JSON 解析錯誤: Expecting property name enclosed in double quotes: line 1 column 30 (char 29)
```

6. 自訂 JSON 編碼與解碼

6.1 處理非內建類型

如果你需要將自定義類型的資料轉換為 JSON，可以透過繼承 json.JSONEncoder 來實現：

範例：

```
import json

class Person:
```

```
def __init__(self, name, age):
    self.name = name
    self.age = age

class PersonEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, Person):
            return {"name": obj.name, "age": obj.age}
        return super().default(obj)

person = Person("Grace", 50)
json_string = json.dumps(person, cls=PersonEncoder, indent=4)
print(json_string)
```

輸出：

```
{
  "name": "Grace",
  "age": 50
}
```

6.2 自訂解碼器

你也可以自訂解碼器將 JSON 資料轉換為自定義的 Python 類別：

```
import json

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

def decode_person(dct):
    if "name" in dct and "age" in dct:
        return Person(dct['name'], dct['age'])
    return dct

json_string = '{"name": "Henry", "age": 60}'
person = json.loads(json_string, object_hook=decode_person)
print(type(person)) # <class '__main__.Person'>
print(person.name) # Henry
```

結論

- 使用 `json.load()` 和 `json.loads()` 來解析 JSON 檔案或字串。
- 使用 `json.dump()` 和 `json.dumps()` 將 Python 資料輸出為 JSON 格式。
- 可以透過 `ensure_ascii` 和 `indent` 參數來控制 JSON 的輸出格式。
- 若需處理自定義類型，可使用自訂的編碼器和解碼器。
- 遇到格式錯誤時，使用 `try-except` 捕捉 `json.JSONDecodeError`。

這些功能幾乎涵蓋了日常開發中處理 JSON 資料的需求，讓你能更高效地處理結構化數據！

json to distionary

```
import json

# json 的資料形式字串
x = '{ "name":"jim", "age":25, "city":"Taiwan"}'

# 轉換json
person = json.loads(x)
```

```
print(type(person)) #<class 'dict'>
print(person) {'name': 'jim', 'age': 25, 'city': 'Taiwan'}
print(person['age']) #25
```

dictionary to json

```
import json

person = {'name': 'jim', 'age': 25, 'city': 'Taiwan'}

data = json.dumps(person)

print(type(data)) #<class 'str'>
print(data) #{"name": "jim", "age": 25, "city": "Taiwan"}
```

🕒 修訂版本 #3

★ 由 treeman 建立於 24 🕒🕒@🕒🕒 2021 00:08:44

✎ 由 treeman 更新於 23 🕒@🕒🕒 2025 11:50:50