第一章 自动化数据采集及清洗 1

主讲：方杰、李烜

福建江夏学院金融学院

本章目录

应用程序接口

应用程序接口（API，Application Programming Interface）是一个预定义的函数，这个函数中约定了数据资源提供方的通信规则。通过应用程序接口可以实现相互独立的系统之间的数据请求、获取和调用。

工作机制

资源提供方提供标准接口文档 $\longrightarrow$ 调用方按需选择接口并传入相关参数 $\longrightarrow$ 资源提供方服务器接收请求，进行业务处理，返回数据。

API 的优点

- 简单、易用
- 能设置报错机制，便于监控接口运行情，及时排除故障
- 分页和过滤功能节省服务器资源和带宽，以支持企业级的高并发和大容量
- 设置访问权限，提高信息的安全性
- 可平滑的移植和扩展，以保证系统的稳定性

API 的缺点

- 需要用不同的语言封装，增加了设计复杂度
- 封装后降低代码的可复用性，也可能会降低程序的执行效率

Python 模块

Python 模块 (Module)，本质上是一个 Python 程序，以.py 作为文件后缀，任何 py 文件都可以作为一个模块。模块能定义函数、类和变量，也能包含可执行的代码。

Python 模块一共有三种：

1. Python 内置模块（标准库）
2. 第三方模块
3. 应用程序/自定义的模块

简单的 Python 模块 module1.py

```
# 定义一个打印函数
def print_func(par):
    print('Hello:', par)
    return

# 定义一个求和函数
def sum(x,y):
    print('sum:', x+y)
    return x+y
```

Python 包

Python 按目录来组织模块，称为包（Package）。

包类似文件夹，用来管理和分类模块的。这个文件夹下必须存在`__init__.py`文件，用于标识当前文件夹是一个包。

包是一个分层次的文件目录结构，定义了一个由模块、子包、子包下的子包等组成的 Python 的应用环境。

演示：

Python 包 `Numpy` 中的文件及目录结构。

Python 包举例

在package_run目录下, 创建 runtest1.py、runtest2.py、__init__.py三个文件。

runtest1.py

```
def test1():
    print('I am in runtest1')
def sum1(x,y):
    print('sum1:',x+y)
    return x+y
```

runtest2.py

```
def test2():
    print('I am in runtest2')
def sum2(x,y):
    print('sum2:',x+y)
    return x+y
```

__init__.py

```
__all__=['runtest2'] #定义加载子模块的名称是runtest2
```

包导入与函数引用：import

对于已定义的包，可以使用 import 语句来导入包

```
import 包名称
```

该方法导入包中__init__.py文件所定义的模块

```
# 导入自定义Python 包package_run
import package_run                    #导入包
package_run.runtest2.test2()
                #调用runtest2.py中的函数test2
```

注意：

此时无法调用runtest1.py中的函数

包导入与函数引用：from ... import ...

使用 from ... import ...语句，可以从包中导入指定的模块

from 包名称 **import** 模块名称

导入包package_run 下的runtest1模块

```
from package_run import runtest1 #导入包中的模块
runtest1.test1()   #调用runtest1模块中的函数test1
```

包导入与函数引用：from ... import *

使用 from ... import * 语句导入包中__init__.py文件所定义的模块，具体语法为：

```
from 包名称 import *
模块名.函数名
```

导入package_run包中的runtest2模块，并调用其中的test2函数

```
from package_run import * #导入包中的所有模块
runtest2.test2()      #调用runtest2模块中的test2函数
```

举例:

输出 $\sqrt{2}$ 的结果

方法 1:

```
import math
print(math.sqrt(2))
```

方法 2:

```
from math import *
print(sqrt(2))
```

输出 π 的结果

```
from math import pi
print(pi)
```

API 接口调用的 Python 包

- `requests`
- `pandas`
- `numpy`
- `matplotlib`
- `OnePlusX`
- ……

requests

requests 专门用于发送 HTTP 请求，requests 包中最常用的是 get 和 post 请求函数。

get 函数

get 函数用于向服务器请求传送数据

```
requests.get(url)
```

post 函数

post 函数向服务器传送数据，并且可以携带请求参数

```
requests.post(url,data=None,json=None)
```

post 函数

```
requests.post(url, data=None, json=None)
```

- 参数url是拟获取页面的网址，不可省略
- 参数data是指定的获取条件，可省略
- 参数json表示请求是以json形式发送请求

json 是什么?

JSON（JavaScript Object Notation）是一种轻量级的数据交换格式。易于阅读和编写，可以在多种语言之间进行数据交换。

例：获取某平台上张三的信息

```
import requests              # 导入包
targetUrl = 'http://www.iyyyf.com' # 服务器地址
data = {'name':'张三', 'age':20}
# 向请求地址并传递data参数
response = requests.post(targetUrl, data = data)
```

pandas 包

pandas 包可将接口调取的数据输出到 excel 文件中

```
df.to_excel(文件位置及名称, sheet_name=0, columns=None,
            header=True, usecols=None, dtype=None)
```

例：将获取到的 price 数据存储为 excel 文档

```
price.to_excel('D:\price_data.xlsx')
```

其他类似的命令

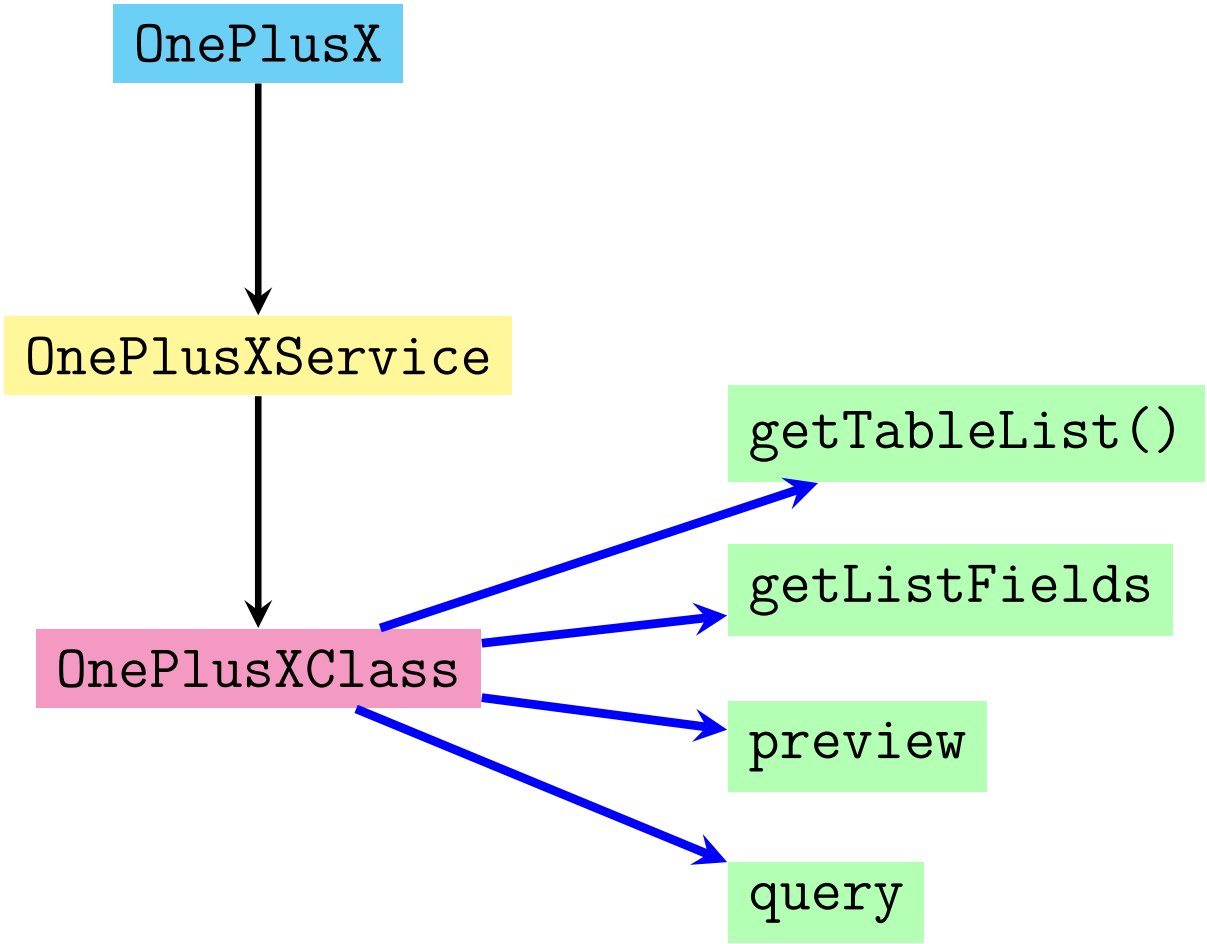
```
df.to_csv(), df.to_sql(), df.to_hdf()
```

OnePlusX 包

OnePlusX 包是数据资源方 CSMAR 数据公司所定义的数据接口服务包，包中封装了 OnePlusXService 类。其中的 OnePlusXClass 模块中包含了许多接口调用函数

- 查看数据表 getTableList()
- 查看表格字段 getListFields
- 预览数据 preview
- 条件查询 query

OnePlusX 包的层次关系



任务实施（见 P21）

两种方法：

1. 调用 API，基于 `OnePlusX` 包进行数据的调用及处理
2. 使用 `requests` 包，通过 Web Service API 进行调用

三个步骤：

1. 查询数据表
2. 查看数据表的字段（field）列表
3. 根据条件查询数据

结构化数据

结构化数据，是指可以使用二维表结构表示和存储的数据，具有易于输入、存储、查询和分析的特点。

使用 Python 对表结构的数据进行处理，需要掌握Pandas库的 DataFrame 数据结构。

DataFrame（数据帧）

使用 DataFrame 首先需要导入Pandas包，Pandas（Panel data analysis）是 Python 的一个数据分析包, 内含大量库和一些标准的数据模型。

DataFrame 是Pandas中一种表格型数据结构，是结构化数据在 Python 语言中的一种表现形式。

构建 DataFrame 对象的方法：

1. 从字典构造
2. 从表格文件（excel, csv,...）构造

什么是字典（dict)?

字典是 Python 内存放具有映射关系的表结构数据的常用方式。字典中的内容放在 {...} 里，其中保存了一一对应的若干组数据，以“键: 值”的方式存储。

```
{key1 : value1, key2 : value2, key3 : value3, ....}
```

键（key）是关键字，相当于数据库中的字段（field)，键是不能重复的；值（value）是键所对应的值，相当于数据库中的记录（record)。

注意：

json 文件具有类似于字典的数据结构。通常可以利用 json 文件的文本，生成字典。

字典方式构建 DataFrame 对象

构建 DataFrame 对象时，先定义数据字典，再将字典转化为 DataFrame 对象。

构造方式

```
import pandas as pd
data = { 列名称A : [数据A1, 数据A2, ...],
         列名称B : [数据B1, 数据B2, ...], ...}
df = pd.DataFrame(data)
```

注意:

此时 DataFrame 的行索引是从 0 开始的整数。我们可以通过index选项对行索引进行修改。columns选项则可以修改列索引的名称。

从表格文件构建 DataFrame 对象

对于已存在的表格文件，可以使用Pandas将其中的数据转换为 DataFrame 对象。

```
import pandas as pd
df1 = pd.read_excel(r'D:\abc\test.xlsx',
                    sheet_name=Sheet2)
df2 = pd.read_csv(r'D:\abc\test.csv')
```

提示:

若 csv 文件中有非西文字符（如汉字），务必确保文件的编码是UTF-8，否则表格的读取会报错

DataFrame 中数据的提取

常用的方式：

1. `df[columns][index]`：获取 df 中行索引值为 index，列索引值为 columns 的数据（先列后行）
2. `df.loc[index, columns]`：结果同上（先行后列）
3. `df.iloc[val1,val2]`：获取 df 中行索引位置为 val1，列索引位置为 val2 的数据，只接受从 0 开始的整数（integer）

说明：

- `loc`: location
- `iloc`: integer location

获取 DataFrame 的某一列数据

获取 DataFrame 对象df中所有基金简称列的数据

```
print(df['基金简称'])
print(df.loc[:, '基金简称'])
print(df.iloc[:, 0])
```

说明:

“基金简称”这一列在df中位于第一列，因此列索引位置的序号为 0

获取 DataFrame 的多列数据

获取 DataFrame 对象df中所有基金简称和基金代码数据

```
print(df[['基金简称', '基金代码']])
print(df.loc[:, ['基金简称','基金代码']])
print(df.iloc[:, [0,2]])
```

说明:

多个列名需要存入列表（list）[ ]来表示。“基金简称”和“基金代码”在df中位于第 1 列和第 3 列，因此列索引位置的序号分别为 0 和 2

获取 DataFrame 的某一行数据

获取上文 DataFrame 对象df中第 2 行的数据

```
print(df.loc['二']) # 第2行的行索引值为“二”
print(df.iloc[1])
```

说明:

Python 中位置序号从 0 开始，因此第 2 行在序列位置中记为 1

获取 DataFrame 的多行数据

获取 DataFrame 对象df中第 2、3 行的数据

```
print(df.loc[['二', '三']])
print(df.iloc[[1, 2]])
```

注意：

多个行名时需要存入列表（list）[ ]来表示；第 2、3 行的位置分别记为 1、2。

数据的读写

可以通过 Pandas 提供的多种读写函数将表格型数据读取为 DataFrame 对象。

- `pd.read_excel`
- `pd.to_excel`
- `pd.read_csv`
- `pd.to_csv`
- `pd.read_json`
- `pd.to_json`
- `read_table`

从 Excel 文件读取数据

- 确定文件在系统中所存放的路径。本任务的操作全部基于 Windows 操作系统，在电脑中找到文件，鼠标右键查看文件属性，即可得到文件路径。
- 将位置和文件名进行组合，得到文件的完整路径。其中位置与文件名之间要用目录分隔符隔开（用/或者\\）。
- 使用 Pandas 中的`pd.read_excel()`函数，将文件路径作为参数，将 Excel 文件中的数据读入并转化为 DataFrame 对象形式。

pd.read_excel() 函数的使用方法

```
pd.read_excel(io, sheet_name=0, header=0, names=None,
              index_col=None, usecols=None, dtype=None)
```

- io 读取文件的路径，URL 地址等
- sheet_name 需要读取的 Excel 文件中 sheet 页的名字
- header 指定哪一行做为列索引
- names 设置列索引，默认不指定
- index_col 指定哪一列做为行索引，默认不指定，自动从 0 开始生成索引号
- usecols 指定读取的列，默认全部读取
- dtype 指定读取列数据的数据类型（data type）

将数据写入 Excel 文件

当数据经过预处理之后，如需将清洗后的数据存写入 Excel 文件中，可使用to_excel()函数

```
df.to_excel(excel_writer, sheet_name=0, columns=None,
            header=True, index=True)
```

- excel_writer 写入的路径对象。
- sheet_name 将数据写入的 Excel 文件某个指定的 sheet 中。
- columns 需要写入文件的列索引，默认所有列都写入文件。
- header 是否将列索引写入文件，默认写入列名。
- index 是否将行索引写入文件，默认写入行索引。

从 CSV 文件读取数据

CSV（Comma-Separated Value, 逗号分隔值）文件以纯文本形式存储表格数据（数字和文本），每条记录由字段组成，字段间通常以逗号或制表符（Tab, \t）隔开，每条记录间以换行符分隔。

Pandas 中以`pd.read_csv()`函数将 CSV 文件数据读入

pd.read_csv() 函数的使用方法

```
pd.read_csv(filepath, sep=',', header='infer', names=None,
        index_col=None, usecols=None, skip_blank_lines=True)
```

- filepath 读取文件的路径，URL 链接等
- sep 指定 CSV 文件中的分隔符，默认用逗号分隔，可以指定\n（换行符）、\r（回车符）、\t（制表符）等
- header 指定某行作为列索引
- names 设置列索引，默认不指定
- index_col 指定哪一列作为 Dataframe 的行索引，默认没有列索引，自动添加 0、1、2....
- usecols 指定读取 CSV 文件的某些列
- skip_blank_lines 是否跳过空白行，默认不读空白行

将数据写入 CSV 文件

当需要把新的数据写入 CSV 文件进行保存时使用to_csv()函数

```
df.to_csv(path_or_buf, sep=',', na_rep='',
          columns=None, header=True, index=True)
```

- path_or_buf 输出文件路径或者文件对象
- sep 同一行记录中，各字段间的分隔符，默认为逗号
- na_rep 空值的替代字符，默认为空字符串
- columns 要写入文件中的df中的列，默认为全部列
- header 是否将列索引写入文件，默认写入列索引
- index 是否将行索引写入文件，默认写入行索引