**常用数据格式学习笔记**

CSV，JSON和XML是日常使用中常见的数据格式，而HDF5则非常适用于大数据的存储和传输，因此对以上几种数据类型进行学习。

**目 录**

[1. CSV格式 1](#_Toc87811522)

[1.1 文件写入 1](#_Toc87811523)

[1.2 文件读取 4](#_Toc87811524)

[1.3 CSV转JSON和XML格式 5](#_Toc87811525)

[2. JSON格式 6](#_Toc87811526)

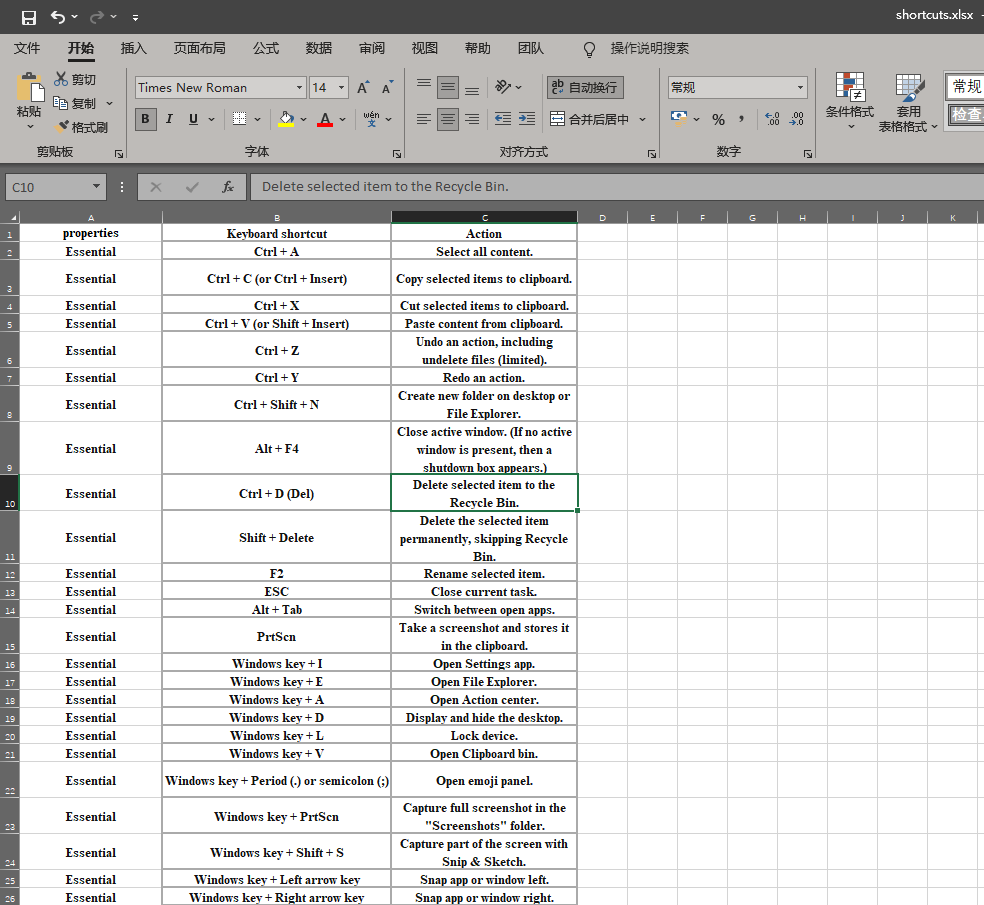
[3. XML格式 7](#_Toc87811527)

[4. HDF5格式 8](#_Toc87811528)

# CSV格式

## 文件写入

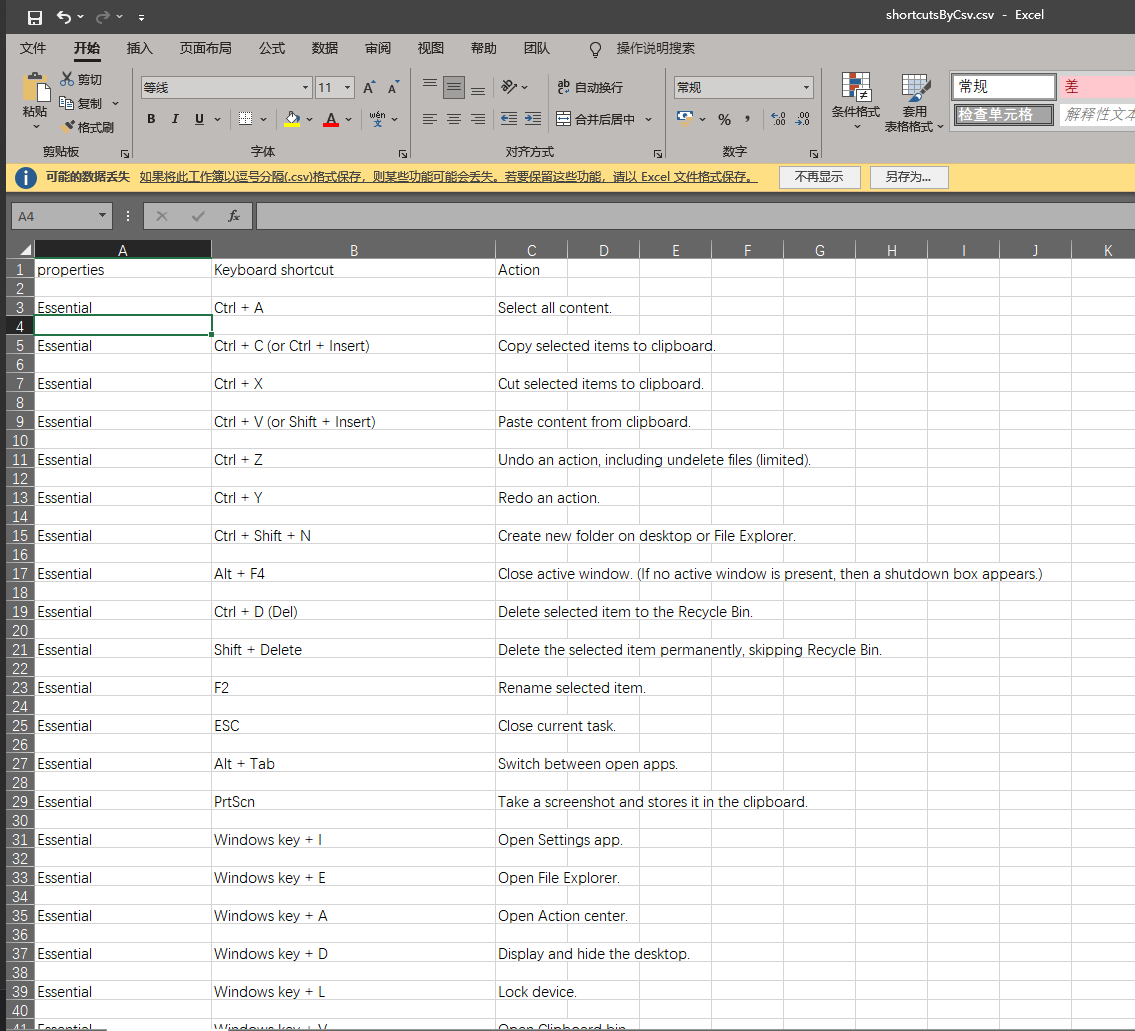
首先随便找了一些Windows常用快捷键，创立了一个Excel表，用于最初数据的读取：



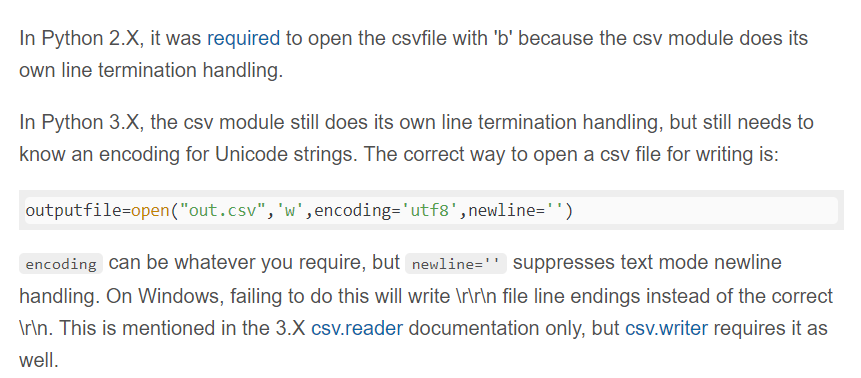
利用pandas库对Excel进行读取（参见<https://geek-docs.com/pandas/pandas-read-write/pandas-to-read-and-write-excel.html>），分别利用Python自带的csv库和pandas库进行CSV的写入（参见<https://geek-docs.com/pandas/pandas-read-write/csv-pandas-speaking-reading-and-writing.html>），代码如下：

1. **import** csv
2. **import** pandas as pd
3. # reading data from excel
4. data = pd.read\_excel('shortcuts.xlsx')
6. # writing to csv file by pandas tool
7. data.to\_csv("shortcutsByPandas.csv", index=False, header=True)
9. # writing to csv file by csv tool
10. filename = "shortcutsByCsv.csv"
11. with open(filename, 'w+', newline='') as csvfile:
12. csvwriter = csv.writer(csvfile)
13. csvwriter.writerow(data.head())
14. csvwriter.writerows(data.values)

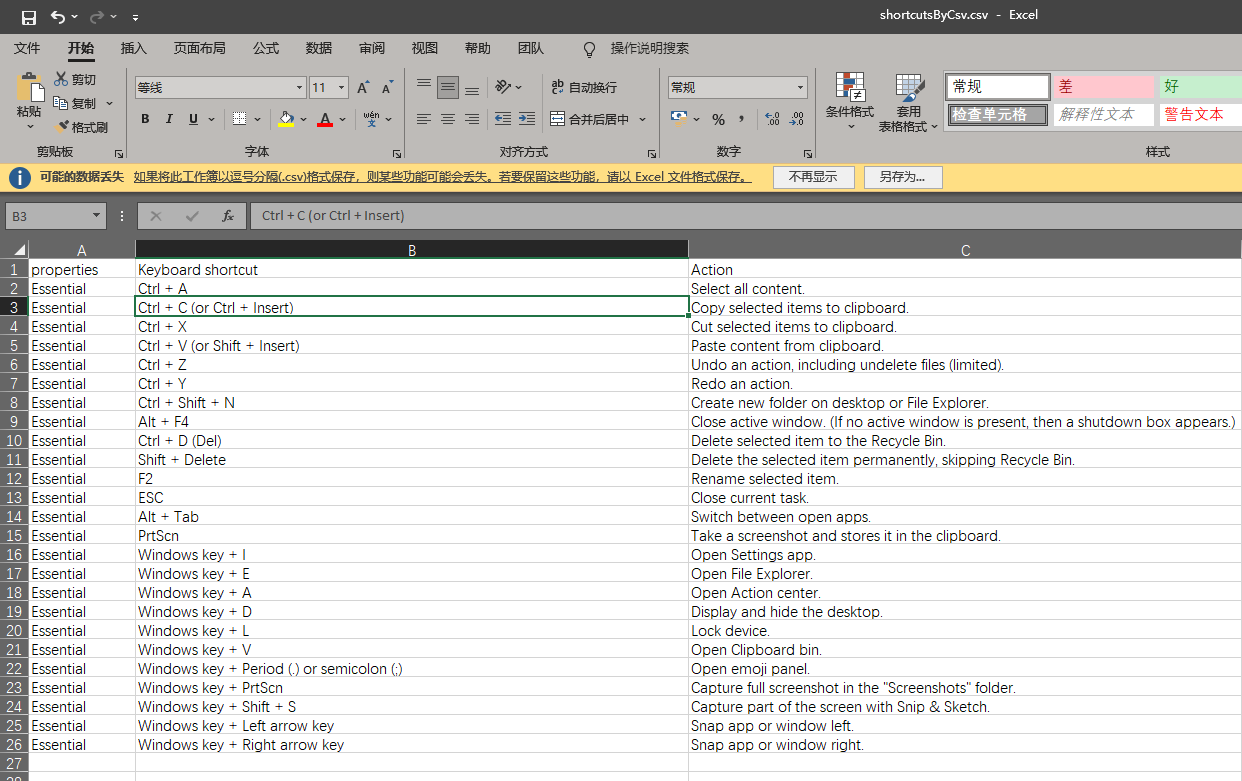
值得注意的是，在采用csv写入的时候，会**出现空行**：



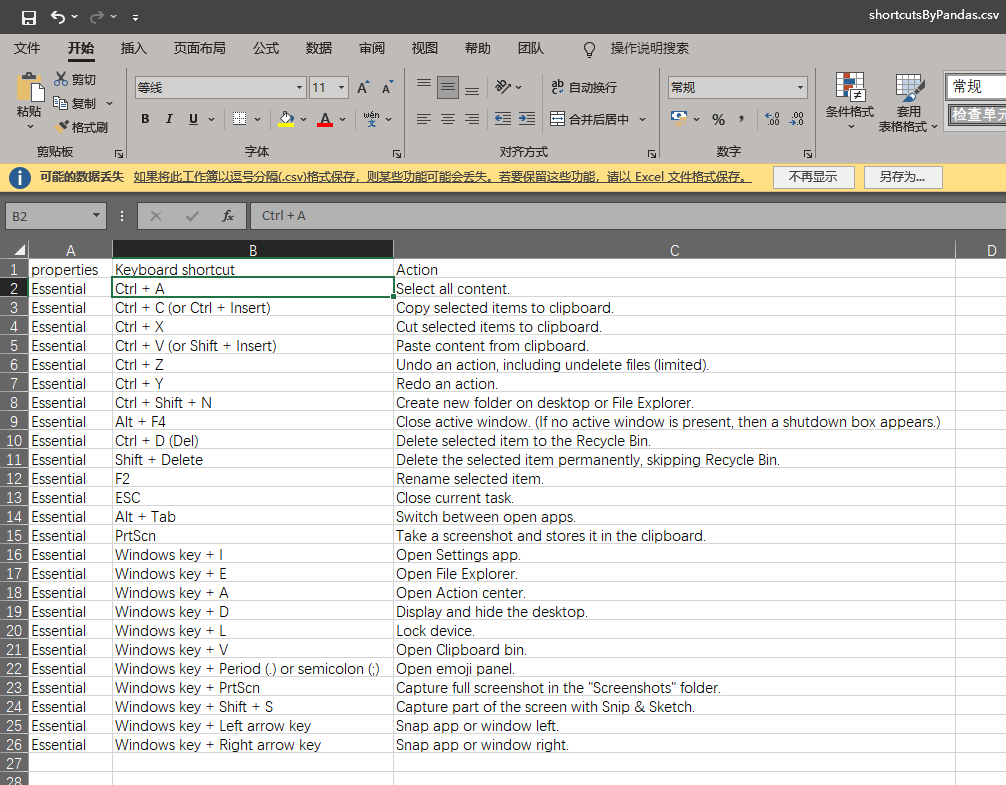
根据python参考手册里的解释，在windows这种使用\r\n的系统里，不用newline=‘’的话，会自动在行尾多添加个\r，导致多出一个空行，即行尾为\r\r\n（参见<https://blog.csdn.net/pfm685757/article/details/47806469>）：



因此，需要在**打开文件的时候增加newline=''**，结果如下：



采用pandas进行写入，默认索引和列名称连同数据一起写入，可以设置index和header选项的参数，按照自己理想模式进行写入，最终结果如下：

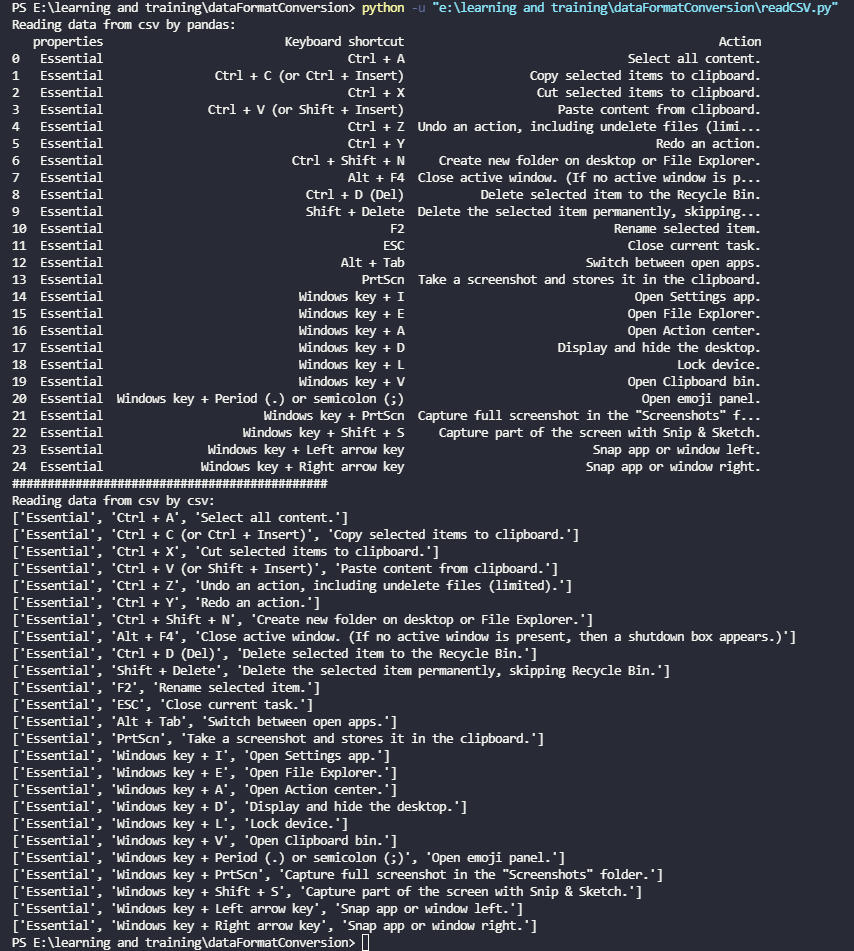


## 文件读取

分别采用csv和pandas库对CSV格式文件进行读取，代码如下：

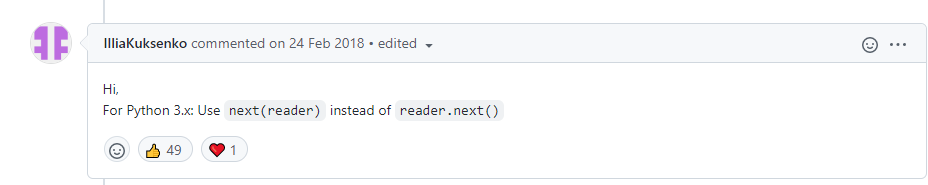
1. **import** csv
2. **import** pandas as pd
3. filename = "shortcuts.csv"
5. # reading data from csv by pandas
6. data = pd.read\_csv(filename)
7. **print**('Reading data from csv by pandas:')
8. **print**(data)
9. **print**('#############################################')
11. # reading data from csv by csv
12. fields = []
13. rows = []
14. with open(filename, 'r', newline='') as csvfile:
15. csvreader = csv.reader(csvfile)
16. fields = next(csvreader)
17. **for** row **in** csvreader:
18. rows.append(row)
19. **print**('Reading data from csv by csv:')
20. **for** row **in** rows:
21. **print**(row)

其结果如下：



可以看到，直接采用pandas读取的是以列表的形式展现的，而csv则是以list形式展示的。

需要注意的是，**教程中的csvreader.next()是Python2中的用法，在这里应该使用next(csvreader)来替代**（参见<https://github.com/GalvanizeDataScience/building-spark-applications-live-lessons/issues/2>）

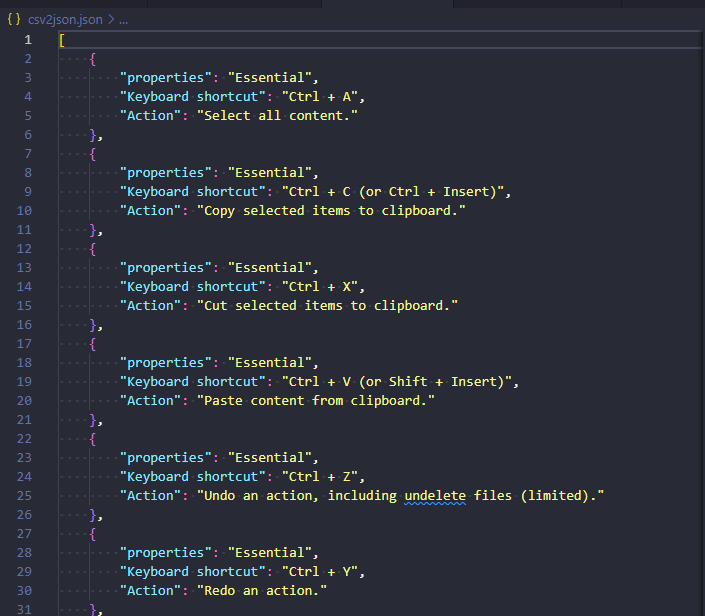


## CSV转JSON和XML格式

利用Pandas，我们可以很轻松地将csv转换为字典列表，从而利用dicttoxml库将其转换为XML格式，利用json库将其保存为JSON格式。

1. **import** pandas as pd
2. **from** dicttoxml **import** dicttoxml
3. **import** json
4. filename = "shortcuts.csv"
6. # reading data from csv by pandas
7. data = pd.read\_csv(filename)
9. # converting the dataframe to a dictionary
10. data\_dict = data.to\_dict(orient="records")
11. with open('csv2json.json', "w+") as json\_file:
12. json.dump(data\_dict, json\_file, indent=4)
14. # converting the dataframe to XML
15. xml\_data = dicttoxml(data\_dict).decode()
16. with open("csv2xml.xml", "w+") as xml\_file:
17. xml\_file.write(xml\_data)

转换后的JSON格式数据如下：



转换后的XML内容如下：



# JSON格式

分别利用pandas和json库进行JSON格式文件的读写（参见<https://geek-docs.com/pandas/pandas-read-write/pandas-reading-and-writing-json.html>），代码如下：

1. **import** json
2. **import** pandas as pd
4. # read and write json by pandas
5. data = pd.read\_json('csv2json.json', orient='records')
6. export = data.to\_json('shortcutsByPandas.json', orient='records')
8. # read and write json by json
9. with open('csv2json.json') as file:
10. data = json.load(file)
11. with open('shortcutsByJson.json', 'w+') as json\_file:
12. json.dump(data, json\_file, indent=4, sort\_keys=True)

其中，在利用json库中的dump函数进行写入时，可以通过自定义参数，修改最终的格式（参见<https://www.jianshu.com/p/cfbcd9f8691c>），例如此处用到的indent参数代表了缩进的空格式，而sort\_keys是告诉编码器按照字典key排序(a到z)输出。

通过pandas库所生成的JSON文件内容如下：



而通过json库生成的文件内容如下：



可以看到，即使读取的是同一个文件，不同的库再次写入的格式大不相同，json中的dump更为美观。

# XML格式

利用Python的内置xml模块和子模块ElementTree，并依靠xmltodict和dicttoxml，可以实现XML数据格式的读写，以及将其转换成JSON等格式（参见<https://mp.weixin.qq.com/s/A7HOW5JXZwtRrZtic7Gdhw>），具体代码如下：

1. **import** xml.etree.ElementTree as ET
2. **import** xmltodict
3. # from dicttoxml import dicttoxml
4. **import** dicttoxml
5. **import** json
7. tree = ET.parse('csv2xml.xml')
8. xml\_data = tree.getroot()
9. xmlstr = ET.tostring(xml\_data, encoding='utf8', method='xml')
10. data\_dict = dict(xmltodict.parse(xmlstr))
12. # write to xml
13. # xml\_data = dicttoxml(data\_dict).decode()
14. xml\_data = dicttoxml.dicttoxml(data\_dict).decode()
15. with open("shortcutsByXml.xml", "w+") as xml\_file:
16. xml\_file.write(xml\_data)
18. # xml to json
19. with open('xml2Json.json', 'w+') as json\_file:
20. json.dump(data\_dict, json\_file, indent=4, sort\_keys=True)

在第一次跑程序的过程中，遇到了一个报错：



通过查阅资料，发现代码里我直接import dicttoxml，底下也直接调用了它。应该写成from dicttoxml import dicttoxml再直接调用dicttoxml函数，或是底下写成dicttoxml.dicttoxml的形式。这里涉及到import和from import的区别（参见<https://cloud.tencent.com/developer/article/1579422>）。简而言之，from import引入的是包或者说是类，而直接import引入的是函数或者说是方法，二者是包含关系。那么为什么不都只使用import呢？一来是方便起见，如果命名没有冲突且用到的包内函数较少的时候，from import更加方便；其次直接import可能会出现无法直接使用包内的子包和模块（参见<https://www.pythonf.cn/read/106215>），以后还是要多关注一下这些细节问题。

最终生成的XML文件如下：



可以看到，经过xml库调整过的数据比之前转换完直接dump进去的美观很多。

# HDF5格式

HDF5是一种层次化的格式（hierarchial format），经常用于存储复杂的科学数据。与其他方式对比，其速度快、压缩效率高，适合大数据的存储。

HDF5文件结构中有两个关键对象，一个是Groups，另一个是Datasets。Groups就类似于文件夹，每个HDF5文件其实就是根目录；而Datasets类似于NumPy中的数组array。每个Dataset可以分成两部分：原始数据(raw) data values和元数据（参见<https://zhuanlan.zhihu.com/p/104145585>）。

简单生成一个H5文件，并读取打印其key和value，代码如下：

1. **import** h5py
2. **import** numpy as np
4. # writing to h5
5. imgData = np.zeros((30, 3, 128, 256))
6. f = h5py.File('testH5.h5', 'w')
7. f['data'] = imgData
8. f['labels'] = range(100)
9. f.close()
11. # reading from h5
12. f = h5py.File('testH5.h5', 'r')
13. **print**('key:')
14. **print**(f.keys())
15. **print**('#######################################')
16. a = f['data'][:]
17. **print**('value:')
18. **print**(a)
19. f.close()

其输出如下：

