python 中文件创建、读取与反序列化

杨晨
学号 2021212171
北京邮电大学计算机学院

日期：2024 年 2 月 26 日

目录

1 概述

1.1 实验内容

1. 通过 python 程序创建具有 1 千行、1 万行、10 万行的文本和二进制文件，每行的内容不限，自定义即可。请观察程序运行的时间，并给出数据量与写入、读取时间的相关性结论。
2. 通过 python 程序创建 csv 文件，行数在 1000 行以上，至少有一列为数字类型。创建后读入该文件，要求求出该列数字之和。不允许使用 numpy、pandas 等第三方模块。
3. 反序列化
   (a) 读取 QQ 群文件中给出的 object.dat，通过程序修改里面的内容，将语文的分数恢复为 100，将姓名“张三”改为“李四”。
   (b) 使用反序列化方法，显示出修改前和修改后 object.dat 的内容。

1.2 开发环境

- Windows10
- PyCharm 2023.2.1

2 实验过程

2.1 数据量与写入、读取时间的相关性分析

2.1.1 创建文件

通过 Python 程序创建不同大小的文本和二进制文件，并观察程序运行的时间，可以得出数据量与写入、读取时间之间的相关性结论。

首先，我们需要编写 Python 程序来创建指定行数的文本和二进制文件。下面是我的程序：

```
import time
# 创建文本文件
def create_textFile(fileName, numLines):
    start_time = time.time()
    with open(fileName, 'w') as f:
        for i in range(numLines):
            f.write(f"This is line {i+1}\n")
    end_time = time.time()
    print(f"创建文本文件{fileName}, 共{numLines}行, 耗时{end_time-
        start_time}秒")

# 创建二进制文件
def create_binaryFile(fileName, numLines):
    start_time = time.time()
```

```
    with open(fileName, 'wb') as f:
        for i in range(numLines):
            # 创建一个包含单个字节的字节数组
            data = bytes([(i+1) % 256]) + b"\n"
            f.write(data)
    end_time = time.time()
    print(f"创建二进制文件{fileName}, 共{numLines}行, 耗时{end_time-
        start_time}秒")
```

上述程序中，我们定义了两个函数：create_textFile和create_binaryFile，分别用于创建文本文件和二进制文件。这两个函数接受 2 个参数fileName和numLines，表示文件名和文件中的行数。在每个函数中，我们使用open()函数打开文件，并根据行数循环写入内容。最后，我们使用、time模块记录文件创建过程的时间，并打印出结果。

可以通过修改numLines的值为 1 万行或 10 万行，观察创建文件的时间变化情况。

2.1.2 读取文件

其次，我们需要编写 Python 程序来读取刚刚创建的文本文件和二进制文件，下面是我的程序：

```
import time
# 读取文本文件
def read_textFile(fileName):
    start_time = time.time()
    with open(fileName, 'r') as f:
        lines = f.readlines()
    end_time = time.time()
    print(f"读取文本文件{fileName}, 共{len(lines)}行, 耗时{end_time-
        start_time}秒")
# 读取二进制文件
def read_binaryFile(fileName):
    start_time = time.time()
    with open(fileName, 'rb') as f:
        lines = f.readlines()
    end_time = time.time()
    print(f"读取二进制文件{fileName}, 共{len(lines)}行, 耗时{
        end_time-start_time}秒")
```

上述程序中，我们定义了两个函数：read_textFile和read_binaryFile，分别用于创建文本文件和二进制文件。这两个函数接受 1 个参数fileName，表示文件名。在每个函数中，我们

使用open()函数打开文件，并使用readlines()方法读取文件的所有行。读取的内容将存储在lines 变量中。接着，记录结束时间。最后，打印结果。

2.1.3 主函数

程序的主函数如下：

```
if __name__ == "__main__":
    lines = [1000, 10000, 100000]
    for line in lines:
        create_textFile(f"text_{line}.txt", line)
        read_textFile(f"text_{line}.txt")
        create_binaryFile(f"binary_{line}.bin", line)
        read_binaryFile(f"binary_{line}.bin")
        print()
```

在主函数中，首先定义了一个列表lines，其中包含了需要测试的文件行数：1 千行、1 万行、10 万行。然后使用循环遍历该列表，对于每个行数，依次执行以下操作：

- 调用create_textFile()函数创建相应行数的文本文件。
- 调用read_textFile()函数读取创建的文本文件，并输出读取时间。
- 调用create_binaryFile()函数创建相应行数的二进制文件。
- 调用read_binaryFile()函数读取创建的二进制文件，并输出读取时间。
- 打印空行，用于分隔不同行数的测试结果。

通过以上代码，我们可以测试不同行数的文本和二进制文件的创建和读取时间，并观察数据量与读取时间之间的关系。

2.1.4 实验结果

```
创建文本文件text_1000.txt，共1000行，耗时0.0秒
读取文本文件text_1000.txt，共1000行，耗时0.0010001659393310547秒
创建二进制文件binary_1000.bin，共1000行，耗时0.0秒
读取二进制文件binary_1000.bin，共1004行，耗时0.0秒

创建文本文件text_10000.txt，共10000行，耗时0.00599980354309082秒
读取文本文件text_10000.txt，共10000行，耗时0.002000093460083008秒
创建二进制文件binary_10000.bin，共10000行，耗时0.0029997825622558秒
读取二进制文件binary_10000.bin，共10040行，耗时0.0秒

创建文本文件text_100000.txt，共100000行，耗时0.05600142478942871秒
读取文本文件text_100000.txt，共100000行，耗时0.013054847717285156秒
创建二进制文件binary_100000.bin，共100000行，耗时0.03000378608703秒
```

```
读取二进制文件binary_100000.bin，共100391行，耗时0.001999855041503秒
```

2.2 创建和读取 CSV 文件

2.2.1 创建 CSV 文件

首先定义了一个名为create_csv的函数,用于创建一个CSV 文件。函数接受两个参数,fileName表示要创建的文件名，numLines表示要创建的行数。在函数内部，使用open函数打开文件，并将打开的文件对象赋值给csvfile变量。然后，使用csv.writer创建一个writer对象，该对象可以将数据写入 CSV 文件。

在创建 CSV 文件之前，通过writer.writerow写入了文件的第一行，包含了三个列的标题，即"id"、"name"和"score"。接下来，使用range函数和循环，根据指定的行数创建数据，并将每行数据写入 CSV 文件

```python
import csv
# 创建CSV文件
def create_csv(fileName, numLines):
    with open(fileName, "w", newline="") as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(["id", "name", "score"])
        for i in range(numLines):
            writer.writerow([i, "name" + str(i + 1), i * 2])
```

2.2.2 读取 CSV 文件并对数字列求和

代码定义了一个名为read_csv的函数，用于读取 CSV 文件并计算指定列的和。函数接受两个参数，fileName表示要读取的文件名，index表示要计算和的列的索引。

在函数内部，使用open函数打开 CSV 文件，并将打开的文件对象赋值给csvfile变量。然后，使用csv.reader创建一个reader对象，该对象可以逐行读取 CSV 文件的内容。

通过next(reader)跳过第一行，即标题行，因为我们只需要计算数据行的和。然后，使用循环遍历每一行数据，将指定列的值转换为整数，并累加到 sum 变量中。最后，函数返回计算得到的和。

```python
# 读取CSV文件并计算数字列的和
def read_csv(fileName, index):
    with open(fileName, "r") as csvfile:
        reader = csv.reader(csvfile)
        next(reader)  # 跳过第一行
        sum = 0
        for line in reader:
```

```
            sum += int(line[index])
    return sum
```

2.2.3 主函数

```
if __name__ == "__main__":
    create_csv("data.csv", 1024)
    print(f"sum = {read_csv('data.csv', 2)}")
```

在主函数中，首先调用create_csv函数创建名为"data.csv"的 CSV 文件，行数为 1024。然后，调用read_csv函数读取"data.csv"文件的第二列数据，并计算列的和。最后，使用print函数输出计算得到的和。

通过以上代码，可以实现创建 CSV 文件并读取其中的数据，计算指定列的和，并将结果输出。

2.2.4 实验结果

```
sum = 1047552
```

2.3 序列化与反序列化

2.3.1 读取 dat 文件，反序列化后输出

首先，我们需要加载pickle模块来进行反序列化操作。

定义了一个名为showData的函数，用于显示反序列化后的数据。函数接受一个参数file，表示要读取的序列化文件。在函数内部，使用open函数打开指定的文件，并以二进制模式读取('rb')。接着，通过pickle.load函数从文件中逐个读取对象，并打印出来。使用try-except结构来捕获EOFError异常，当读取完所有对象时，pickle.load函数会抛出该异常，此时跳出循环。

```
import pickle

# 显示反序列化后的数据
def showData(file):
    with open(file, 'rb') as f:
        while True:  # 有多个对象，循环读取
            try:
                data = pickle.load(f)
                print(data)
            except EOFError:
```

```
                break
```

2.3.2 修改文件，序列化保存

定义了一个名为modify的函数，用于修改序列化后的对象数据。函数接受两个参数，input_file表示输入的序列化文件，output_file表示输出的修改后的序列化文件。

在函数内部，通过open函数以二进制模式打开输入文件，并使用pickle.load函数从文件中读取两个对象数据，分别赋值给data1和data2。

接下来，对data2进行修改。根据 PPT 的示例，需要将data2中第二个字典对象的score字典的语文键的值改为100，将name键的值改为'李四'。

最后，使用open函数以二进制模式打开输出文件，并使用pickle.dump函数将修改后的data1和data2对象分别序列化并写入输出文件。

```
def modify(input_file, output_file):
    with open(input_file, 'rb') as f:
         data1 = pickle.load(f)
         data2 = pickle.load(f)
    data2[1]['score']['语文'] = 100
    data2[1]['name'] = '李四'

    # 序列化回文件
    with open(output_file, 'wb') as f:
        pickle.dump(data1, f)
        pickle.dump(data2, f)
```

2.3.3 主函数

在主函数中，首先打印提示信息"反序列化后的数据："，然后调用showData函数读取并显示名为"object.dat"的序列化文件中的对象数据。

接着，调用modify函数，传入"object.dat"作为输入文件，"modified_object.dat"作为输出文件，对序列化文件中的对象数据进行修改。

最后，再次打印提示信息"修改后的数据："，并调用showData函数读取并显示修改后的序列化文件"modified_object.dat"中的对象数据。

```
if __name__ == '__main__':
    print("反序列化后的数据：")
    showData("object.dat")
    modify("object.dat", "modified_object.dat")
    print("修改后的数据：")
    showData("modified_object.dat")
```

通过以上代码，可以实现对序列化文件中的对象数据进行反序列化、显示、修改以及再次序列化的操作。

2.3.4 实验结果

```
反序列化后的数据:
Test
('Justfortest', {'name': '张三', 'score': {'语文': 105, '数学':
   123, '英语': 88}})
修改后的数据:
Test
('Justfortest', {'name': '李四', 'score': {'语文': 100, '数学':
   123, '英语': 88}})
```

3 实验总结

本次实验涉及了三个主题，分别是文件操作、CSV 文件读写和反序列化。以下是对每个主题的实验结果和总结:

1. 文件操作:

- 在创建具有不同行数的文本和二进制文件时，观察到数据量与写入、读取时间之间存在一定的相关性。
- 随着行数的增加，写入和读取的时间也会相应增加。
- 对于文本文件，写入和读取时间的增长速度相对较慢，尤其是在较小的文件中。
- 对于二进制文件，写入和读取时间的增长速度更快，尤其是在较大的文件中。
- 这是因为在处理更大的数据量时，文件操作需要更多的时间来完成读写操作。

2. CSV 文件读写:

- 在创建包含 1000 行以上的 CSV 文件时，至少有一列为数字类型。
- 成功读取创建的 CSV 文件，并成功计算该列数字之和。
- 由于要求不允许使用第三方模块如 numpy 和 pandas，实现了一种基本的 CSV 文件读写和计算数字之和的方法。
- 该方法涉及对文件的逐行读取和解析，以提取数字列的值并进行求和运算。
- 性能方面，该方法可能在处理大型 CSV 文件时效率较低，因为它使用了基本的文件操作和迭代解析方法。

3. 反序列化:

- 成功读取给定的 object.dat 文件，并通过程序修改文件内容。
- 将语文的分数恢复为 100，将姓名“张三”改为“李四”。
- 使用反序列化方法，成功显示了修改前和修改后 object.dat 文件的内容。
- 反序列化是一种将对象转换为字节流的过程，使得可以在不同程序之间传输和存储对象。

- 在本实验中，反序列化方法被用于读取和修改二进制文件，并验证了修改的结果。

总结来说，本次实验通过 Python 程序展示了文件操作、CSV 文件读写和反序列化的基本概念和实现方法。在文件操作方面，观察到数据量与写入、读取时间之间的相关性。在 CSV 文件读写方面，通过基本的文件操作和解析方法成功读取了文件并进行了数字之和的计算。在反序列化方面，展示了读取和修改二进制文件内容的过程。这些实验结果和方法为进一步学习和应用文件操作、数据处理和序列化提供了基础。