

Lab08-迭代器效率比较

Lab08-迭代器效率比较

- 1 基本信息
- 2 实验目的
- 3 实验内容
 - 3.1 编程题
 - 3.1.1 题目描述
 - 3.1.2 具体步骤
 - 3.1.3 实验结果
 - 3.1.4 结果分析
 - 一、内存
 - 二、时间
- 4 实验心得
- 5 源代码

1 基本信息

姓名	李懋
学号	2213189
实验题目	迭代器效率比较
完成时间	2024.11.6

2 实验目的

- 熟悉迭代器的理解和使用

3 实验内容

3.1 编程题

- [参考视频](#)

3.1.1 题目描述

- 设计一个迭代器效率比较实验，比较直接将文件内容读取到内存列表中与使用自定义迭代器逐行读取文件的性能，可以从内存占用和时间两个维度比较。
 - 内存占用：可以使用tracemalloc内置包
 - 时间：可以使用time内置包

3.1.2 具体步骤

1. 首先直接读取文件内容，用列表存储，补全如下代码，记录程序占用内存和用时信息：

- 使用 `open` 函数一次性读取整个文件内容到 `lines` 列表中。
- 使用 `tracemalloc` 记录内存占用，使用 `time` 记录用时。

```
##your code
filepath='./log.txt'
with open(filepath,'r') as f:
    lines=f.readlines()
for line in lines:
    process(line)    # 这里不考虑获取内容后的处理，process函数可以直接返回什么都不做
##your code
```

```
# 1. 直接读取文件内容到内存列表中
print("直接读取文件到内存列表中：")
tracemalloc.start()
start_time = time.time()

with open(filepath, 'r') as f:
    lines = f.readlines()

for line in lines:
    process(line)

end_time = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"内存占用：当前 {current / 10 ** 6} MB，峰值 {peak / 10 ** 6} MB")
print(f"用时：{end_time - start_time} 秒")
```

2. 自定义**迭代器**实现文件读取，一次迭代读取文件一行内容，补全如下代码，记录程序占用内存和用时信息

- 定义 `LineIterator` 类，实现 `__iter__` 和 `__next__` 方法，逐行读取文件内容。
- 使用 `tracemalloc` 记录内存占用，使用 `time` 记录用时。

```
# 自定义迭代器实现文件读取，一次迭代读取文件一行内容
##your code
class LineIterator:
    def __init__(self, filepath):
        pass
    def __iter__(self):
        return self
```

```
def __next__(self):
    """
    todo: 补全此处代码，实现一次迭代读取文件一行内容
    """
    pass

line_iter = LineIterator(filepath)
for line in line_iter:
    process(line)    # 这里不考虑获取内容后的处理，process函数可以直接返回什么都不做
##your code
```

```
31      # 2. 自定义迭代器实现文件读取
32      print("使用自定义迭代器逐行读取文件：")
33      tracemalloc.start()
34      start_time = time.time()
35
36      class LineIterator: 1 usage
37          def __init__(self, filepath):
38              self.filepath = filepath
39              self.file = open(filepath, 'r')
40
41          def __iter__(self):
42              return self
43
44          def __next__(self):
45              line = self.file.readline()
46              if not line:
47                  self.file.close()
48                  raise StopIteration
49              return line
```

```

51     line_iter = LineIterator(filepath)
52     for line in line_iter:
53         process(line)
54
55     end_time = time.time()
56     current, peak = tracemalloc.get_traced_memory()
57     tracemalloc.stop()
58
59     print(f"内存占用: 当前 {current / 10 ** 6} MB, 峰值 {peak / 10 ** 6} MB")
60     print(f"用时: {end_time - start_time} 秒")
61

```


3. 如果只需要日志操作信息是Create的日志信息，使用生成器实现迭代器，并补全如下代码：

- 定义 `line_generator` 生成器函数，逐行读取文件内容，并只处理包含 "Create" 的日志信息。
- 使用 `tracemalloc` 记录内存占用，使用 `time` 记录用时。

```

##your code
def line_generator(filepath):
    with open(filepath, 'r') as file:
        for line in file:
            ## your code
line_gen = line_generator(filepath)
for line in line_gen:
    print(line)

```

```
66 # 3. 使用生成器实现迭代器
67 print("使用生成器实现迭代器: ")
68 tracemalloc.start()
69 start_time = time.time()
70
71
72  def line_generator(filepath): 1 usage
73      with open(filepath, 'r') as file:
74          for line in file:
75             if "Create" in line: # 只处理包含 "Create" 的日志信息
76                 yield line
77
78
79 line_gen = line_generator(filepath)
80 for line in line_gen:
81     process(line)
82
83 end_time = time.time()
84 current, peak = tracemalloc.get_traced_memory()
85 tracemalloc.stop()
86
87 print(f"内存占用: 当前 {current / 10 ** 6} MB, 峰值 {peak / 10 ** 6} MB")
88 print(f"用时: {end_time - start_time} 秒")
```

3.1.3 实验结果

直接读取文件到内存列表中：

内存占用：当前 100.881546 MB，峰值 100.898375 MB

用时：0.686185359954834 秒

=====

使用自定义迭代器逐行读取文件：

内存占用：当前 0.005074 MB，峰值 0.046121 MB

用时：0.5830466747283936 秒

=====

|

使用生成器实现迭代器：

内存占用：当前 0.00078 MB，峰值 0.034661 MB

用时：0.5993452072143555 秒

Process finished with exit code 0

3.1.4 结果分析

一、内存

1. 直接读取文件到内存列表中的方法占用了大量的内存，因为它一次性将整个文件内容加载到内存中。这对于大文件来说是非常不利的，可能会导致内存不足的问题。
2. 使用自定义迭代器和生成器的方法内存占用非常低，因为它们是逐行读取文件内容，不会一次性加载整个文件到内存中。
3. 生成器的内存占用最低，因为它在生成数据时不会保留之前的数据，进一步减少了内存占用。

二、时间

1. 直接读取文件到内存列表中的方法用时最长，因为它需要一次性读取整个文件内容，并且需要处理整个列表。
2. 使用自定义迭代器和生成器的方法用时相近，且都比直接读取文件到内存列表中的方法快。这是因为它们逐行读取文件内容，减少了初始读取文件的时间开销。

4 实验心得

1. 通过本次实验熟悉了迭代器，生成器的相关知识。
2. 对于大文件的处理，推荐使用生成器或自定义迭代器的方法，以减少内存占用并提高性能。

5 源代码

```
import tracemalloc
import time

def process(line):
    pass # 这里不考虑获取内容后的处理，process函数可以直接返回什么都不做

filepath = 'log.txt'

# 1. 直接读取文件内容到内存列表中
print("直接读取文件到内存列表中：")
tracemalloc.start()
start_time = time.time()

with open(filepath, 'r') as f:
    lines = f.readlines()

for line in lines:
    process(line)

end_time = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"内存占用：当前 {current / 10 ** 6} MB，峰值 {peak / 10 ** 6} MB")
print(f"用时：{end_time - start_time} 秒")

# 分割线
print("\n" + "=" * 40 + "\n")

# 2. 自定义迭代器实现文件读取
print("使用自定义迭代器逐行读取文件：")
tracemalloc.start()
start_time = time.time()

class LineIterator:
    def __init__(self, filepath):
```

```

        self.filepath = filepath
        self.file = open(filepath, 'r')

    def __iter__(self):
        return self

    def __next__(self):
        line = self.file.readline()
        if not line:
            self.file.close()
            raise StopIteration
        return line

line_iter = LineIterator(filepath)
for line in line_iter:
    process(line)

end_time = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"内存占用: 当前 {current / 10 ** 6} MB, 峰值 {peak / 10 ** 6} MB")
print(f"用时: {end_time - start_time} 秒")

# 分割线
print("\n" + "=" * 40 + "\n")

# 3. 使用生成器实现迭代器
print("使用生成器实现迭代器: ")
tracemalloc.start()
start_time = time.time()

def line_generator(filepath):
    with open(filepath, 'r') as file:
        for line in file:
            if "Create" in line: # 只处理包含 "Create" 的日志信息
                yield line

line_gen = line_generator(filepath)
for line in line_gen:
    process(line)

end_time = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"内存占用: 当前 {current / 10 ** 6} MB, 峰值 {peak / 10 ** 6} MB")
print(f"用时: {end_time - start_time} 秒")

```