



中国海洋大学

调试及性能分析，元编程，大杂烩及
PyTorch 入门

姓名：乔宇恒

学号：23030021059

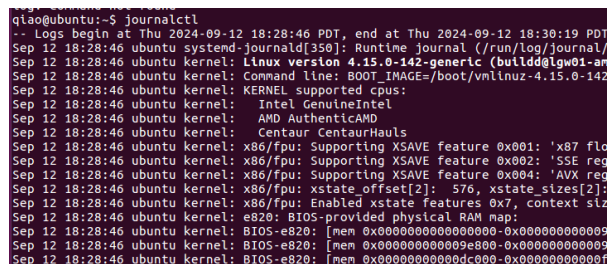
2024 年 9 月 13 日

目录

1	调试及性能分析	2
1.1	使用 journalctl 命令	2
1.2	打印两处代码之间的时间	2
1.3	使用 cProfile 进行性能分析	3
1.4	使用 pstats 模块来查看详细信息	5
2	元编程	6
2.1	C++元编程	6
3	大杂烩	6
3.1	Markdown	6
3.2	修改键位映射	7
3.3	备份	7
3.4	API (应用程序接口)	7
3.5	常见命令行标志参数及模式	7
3.5.1	‘-version’	7
3.5.2	‘-help’ 或 ‘-h’	8
3.5.3	‘-verbose’	8
3.5.4	‘-quiet’ 或 ‘-q’	8
3.5.5	‘-force’ 或 ‘-f’	8
3.5.6	‘-dry-run’	8
3.5.7	‘-output’ 或 ‘-o’	9
3.5.8	‘-input’ 或 ‘-i’	9
3.6	VPN	9
3.7	开机引导	9
4	PyTorch 入门	10
4.1	导入 PyTorch	10
4.2	创建张量	10
4.3	张量操作	10
4.4	定义神经网络	11
5	实验感悟	12
6	Github 网址	12

1 调试及性能分析

1.1 使用 journalctl 命令



```
qiao@ubuntu:~$ journalctl
-- Logs begin at Thu 2024-09-12 18:28:46 PDT, end at Thu 2024-09-12 18:30:19 PDT.
Sep 12 18:28:46 ubuntu systemd-journald[350]: Runtime journal (/run/log/journal/
Sep 12 18:28:46 ubuntu kernel: Linux version 4.15.0-142-generic (build@lgw01-am
Sep 12 18:28:46 ubuntu kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-4.15.0-142
Sep 12 18:28:46 ubuntu kernel: KERNEL supported cpus:
Sep 12 18:28:46 ubuntu kernel: Intel GenuineIntel
Sep 12 18:28:46 ubuntu kernel: AMD AuthenticAMD
Sep 12 18:28:46 ubuntu kernel: Centaur CentaurHauls
Sep 12 18:28:46 ubuntu kernel: x86/fpu: Supporting XSAVE feature 0x001: 'x87 flo
Sep 12 18:28:46 ubuntu kernel: x86/fpu: Supporting XSAVE feature 0x002: 'SSE reg
Sep 12 18:28:46 ubuntu kernel: x86/fpu: Supporting XSAVE feature 0x004: 'AVX reg
Sep 12 18:28:46 ubuntu kernel: x86/fpu: xstate_offset[2]: 576, xstate_sizes[2]:
Sep 12 18:28:46 ubuntu kernel: x86/fpu: Enabled xstate features 0x7, context siz
Sep 12 18:28:46 ubuntu kernel: e820: BIOS-provided physical RAM map:
Sep 12 18:28:46 ubuntu kernel: BIOS-e820: [mem 0x0000000000000000-0x000000000000
Sep 12 18:28:46 ubuntu kernel: BIOS-e820: [mem 0x0000000000000000-0x000000000000
Sep 12 18:28:46 ubuntu kernel: BIOS-e820: [mem 0x0000000000000000-0x000000000000
```

图 1: 使用 journalctl 命令

1.2 打印两处代码之间的时间

```
1 import time, random
2 n = random.randint(1, 10) * 100
3
4 # 获取当前时间
5 start = time.time()
6
7 # 执行一些操作
8 print("Sleeping for {} ms".format(n))
9 time.sleep(n/1000)
10
11 # 比较当前时间和起始时间
12 print(time.time() - start)
```

1.3 使用 cProfile 进行性能分析

```
Profiling quick sort:
Fri Sep 13 10:55:53 2024    insertion_sort_stats

    5 function calls in 0.013 seconds

Ordered by: cumulative time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    1    0.000    0.000    0.013    0.013 {built-in method builtins.exec}
    1    0.000    0.000    0.013    0.013 <string>:1(<module>)
    1    0.013    0.013    0.013    0.013 E:\PyCharm 2023.2.3\Project1\main.py:6(insertion_sort)
    1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
    1    0.000    0.000    0.000    0.000 {built-in method builtins.len}

Fri Sep 13 10:55:53 2024    quick_sort_stats

    2001 function calls (671 primitive calls) in 0.001 seconds

Ordered by: cumulative time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    1    0.000    0.000    0.001    0.001 {built-in method builtins.exec}
    1    0.000    0.000    0.001    0.001 <string>:1(<module>)
```

图 2: 使用 cProfile 进行性能分析

```
1 import cProfile
2
3 def profile_sorting_algorithms():
4     arr1 = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
5     arr2 = arr1.copy()
6
7     print("Profiling insertion sort:")
8     cProfile.run('insertion_sort(arr1)', 'insertion_sort_stats')
9
10    print("\nProfiling quick sort:")
11    cProfile.run('quick_sort(arr2)', 'quick_sort_stats')
12
13 profile_sorting_algorithms()
14 import cProfile
15 import pstats
16 import random
17
18 # 定义插入排序算法
19 def insertion_sort(arr):
20     for i in range(1, len(arr)):
21         key = arr[i]
22         j = i - 1
23         while j >= 0 and key < arr[j]:
24             arr[j + 1] = arr[j]
25             j -= 1
```

```

26         arr[j + 1] = key
27
28 # 定义快速排序算法
29 def quick_sort(arr):
30     def partition(low, high):
31         pivot = arr[high]
32         i = low - 1
33         for j in range(low, high):
34             if arr[j] < pivot:
35                 i += 1
36                 arr[i], arr[j] = arr[j], arr[i]
37         arr[i + 1], arr[high] = arr[high], arr[i + 1]
38         return i + 1
39
40     def _quick_sort(low, high):
41         if low < high:
42             pi = partition(low, high)
43             _quick_sort(low, pi - 1)
44             _quick_sort(pi + 1, high)
45
46     _quick_sort(0, len(arr) - 1)
47
48 # 定义性能分析函数
49 def profile_sorting_algorithms():
50     # 生成随机数据
51     array_size = 1000
52     data_insertion = [random.randint(0, 10000) for _ in range(array_size
53 )]
54     data_quick = data_insertion.copy()
55
56     # 性能分析插入排序
57     print("Profiling insertion sort:")
58     cProfile.runctx('insertion_sort(data_insertion)', globals(), locals
59 ), 'insertion_sort_stats')
60
61     # 性能分析快速排序
62     print("\nProfiling quick sort:")

```

```
61     cProfile.runctx('quick_sort(data_quick)', globals(), locals(), '  
    quick_sort_stats')  
62  
63 # 运行性能分析  
64 profile_sorting_algorithms()  
65  
66 # 打印性能分析结果  
67 def print_profile_stats(filename):  
68     p = pstats.Stats(filename)  
69     p.sort_stats('cumulative').print_stats(10)  
70  
71 print_profile_stats('insertion_sort_stats')  
72 print_profile_stats('quick_sort_stats')
```

1.4 使用 pstats 模块来查看详细信息

```
1 import pstats  
2  
3 def print_profile_stats(filename):  
4     p = pstats.Stats(filename)  
5     p.sort_stats('cumulative').print_stats(10)  
6  
7 print_profile_stats('insertion_sort_stats')  
8 print_profile_stats('quick_sort_stats')
```

2 元编程

2.1 C++元编程

```
1 #include <iostream>
2
3 // 模板元编程计算阶乘
4 template<int N>
5 struct Factorial {
6     static const int value = N * Factorial<N - 1>::value;
7 };
8
9 // 基础情况, 0! = 1
10 template<>
11 struct Factorial<0> {
12     static const int value = 1;
13 };
14
15 int main() {
16
17 }
```

3 大杂烩

3.1 Markdown

```
1 # 项目文档
2
3 ## 介绍
4
5 这是一个示例 Markdown 文档，展示了如何使用 Markdown 进行文档编写。Markdown
6 是一种轻量级的标记语言，广泛用于编写文档和 README 文件。
7
8 ## 目录
9
10 1. [介绍](#介绍)
11 2. [安装指南](#安装指南)
```

```
11 3. [使用示例](#使用示例)
12 4. [表格示例](#表格示例)
13 5. [代码示例](#代码示例)
14 6. [图片示例](#图片示例)
15
16 ## 安装指南
17
18 ### 安装步骤
19
20
21
22 ## 使用示例
```

3.2 修改键位映射

打开 PowerToys，进入 Keyboard Manager 选项。

点击 Remap a key，选择你要重新映射的按键和目标按键。

保存更改后，映射立即生效，无需重启。

3.3 备份

摘抄：有效备份方案的几个核心特性是：版本控制，删除重复数据，以及安全性。对备份的数据实施版本控制保证了用户可以从任何记录过的历史版本中恢复数据。在备份中检测并删除重复数据，使其仅备份增量变化可以减少存储开销。在安全性方面，作为用户，你应该考虑别人需要什么信息或者工具才可以访问或者完全删除你的数据及备份。

3.4 API（应用程序接口）

摘抄：这些 API 大多具有类似的格式。它们的结构化 URL 通常使用 api.service.com 作为根路径，用户可以访问不同的子路径来访问需要调用的操作，以及添加查询参数使 API 返回符合查询参数条件的结果。

3.5 常见命令行标志参数及模式

3.5.1 ‘-version’

显示当前工具或程序的版本信息。


```
1 # 示例:
2 git --version
3 python --version
```

3.5.2 ‘-help’ 或 ‘-h’

显示帮助信息或命令的用法，包括可用的选项、标志及其描述。

```
1 # 示例:
2 python --help
```

3.5.3 ‘-verbose’

显示详细的操作过程信息，用于调试或获取更多输出。

```
1 # 示例:
2 tar --verbose -czf archive.tar.gz directory/
```

3.5.4 ‘-quiet’ 或 ‘-q’

抑制大部分输出，只显示必要的信息或错误消息。

```
1 # 示例:
2 make --quiet
```

3.5.5 ‘-force’ 或 ‘-f’

强制执行某项操作，即使这可能会覆盖文件或产生不必要的风险。

```
1 # 示例:
2 rm --force file.txt
```

3.5.6 ‘-dry-run’

模拟执行命令，但不实际执行任何操作，用于测试命令的影响。

```
1 # 示例:
2 rsync --dry-run -av source/ destination/
```

3.5.7 ‘-output’ 或 ‘-o’

指定输出文件或目录。

```
1 # 示例:  
2 gzip --output=file.gz file.txt
```

3.5.8 ‘-input’ 或 ‘-i’

指定输入文件或目录。

```
1 # 示例:  
2 grep --input=file.txt pattern
```

3.6 VPN

摘抄：使用了 VPN 的你对于互联网而言，最好的情况下也就是换了一个网络供应商（ISP）。所有你发出的流量看上去来源于 VPN 供应商的网络而不是你的“真实”地址，而你实际接入的网络只能看到加密的流量。

曾经发生过 VPN 提供商错误使用弱加密或者直接禁用加密的先例。另外，有些恶意的或者带有投机心态的供应商会记录和你有关的所有流量，并很可能会将这些信息卖给第三方。找错一家 VPN 经常比一开始就不用 VPN 更危险。

3.7 开机引导

电源自检（POST）：

当计算机开机时，电源会初始化所有硬件组件。计算机的 BIOS（或 UEFI）会执行自检程序（POST），检查硬件设备是否正常工作。BIOS/UEFI 引导：

BIOS: 基本输入输出系统（BIOS）是旧式的固件，负责启动过程的初始阶段。

UEFI: 统一可扩展固件接口（UEFI）是现代替代 BIOS 的固件，提供更先进的功能。

BIOS/UEFI 查找引导设备（如硬盘、SSD、USB 驱动器）并加载引导程序。

加载引导程序：

BIOS/UEFI 从引导设备加载引导程序（如引导扇区中的引导加载程序）。

引导程序（Bootloader）负责将操作系统内核加载到内存中。常见的引导程序有 GRUB（用于 Linux）、Windows Boot Manager（用于 Windows）。

加载操作系统内核：

引导程序将操作系统内核加载到内存中并将控制权移交给它。

操作系统内核初始化系统的各种组件，如驱动程序、系统服务等。

启动系统服务：

内核加载并启动系统服务和后台进程，使系统能够提供用户界面和应用程序功能。

用户登录:

操作系统完成启动后, 呈现登录界面, 用户可以输入用户名和密码登录系统。

4 PyTorch 入门

4.1 导入 PyTorch

```
1 import torch
2 import torchvision
```

4.2 创建张量

```
1 # 从列表创建张量
2 a = torch.tensor([1, 2, 3, 4])
3 print(a) # 输出: tensor([1, 2, 3, 4])
4
5 # 创建指定形状的零张量
6 b = torch.zeros(2, 3)
7 print(b) # 输出: tensor([[0., 0., 0.],
8                     [0., 0., 0.]])
9
10 # 创建指定形状的随机张量
11 c = torch.rand(2, 3)
12 print(c) # 输出: tensor([[0.8122, 0.2394, 0.7886],
13                     [0.4924, 0.8756, 0.3412]])
```

4.3 张量操作

```
1 # 从列表创建张量
2 a = torch.tensor([1, 2, 3, 4])
3 print(a) # 输出: tensor([1, 2, 3, 4])
4
5 # 创建指定形状的零张量
6 b = torch.zeros(2, 3)
7 print(b) # 输出: tensor([[0., 0., 0.],
8                     [0., 0., 0.]])
```

```

9
10 # 创建指定形状的随机张量
11 c = torch.rand(2, 3)
12 print(c) # 输出: tensor([[0.8122, 0.2394, 0.7886],
13                      [0.4924, 0.8756, 0.3412]])

```

4.4 定义神经网络

```

1 import torch.nn as nn
2 import torch.optim as optim
3
4 class SimpleNet(nn.Module):
5     def __init__(self):
6         super(SimpleNet, self).__init__()
7         self.fc1 = nn.Linear(10, 5) # 全连接层, 从10个输入到5个输出
8         self.fc2 = nn.Linear(5, 1)  # 全连接层, 从5个输入到1个输出
9
10    def forward(self, x):
11        x = torch.relu(self.fc1(x)) # 激活函数 ReLU
12        x = self.fc2(x)
13        return x
14
15 # 实例化模型
16 model = SimpleNet()
17 print(model)

```

5 实验感悟

学习了调试及性能分析，了解到 C++ 中模板类可叫做元编程，了解了修改键位映射、备份的重要性、Markdown 文本编辑器、API 的用处、常见命令行操作，VPN 的安全性、开机引导，以及 PyTorch 的入门，如何构建神经网络，训练模型。

6 Github 网址

<https://github.com/Eternity-JOE/Git>