



中国海洋大学

第四周实验报告

於佳杰

第四周实验报告

於佳杰

2024 年 9 月 14 日

目录

1 实验目的	1
2 实例展示	1
2.1 调试及性能分析	1
2.1.1 在 Linux上使用可获取最后一天的超级用户访问权限和命令。	1
2.1.2 安装 shellcheck 并尝试检查以下脚本。	2
2.1.3 查看占用进程的pid	3
2.1.4 运行并使用可视化 CPU 消耗。	4
2.1.5 执行 HTTP 请求并获取有关您的公有 IP 的信息。	5
2.1.6 使用 GDB 调试 C/C++ 程序	6
2.1.7 使用 strace 跟踪系统调用	7
2.1.8 检测内存泄漏	8
2.1.9 使用 perf 分析程序性能	10
2.2 元编程演示实验	11
2.2.1 make 重新构建	11
2.2.2 动态创建类和方法	12
2.2.3 动态属性添加	13
2.2.4 使用元类进行属性验证	14
2.2.5 使用装饰器进行方法计时	15
2.2.6 使用元类实现只读属性	16
2.2.7 使用装饰器进行方法日志记录	18
2.3 大杂烩	19
2.3.1 Markdown表格	19
2.3.2 Markdown嵌套列表	21
2.3.3 引用块	22
2.3.4 任务列表	23
2.3.5 自定义链接和图片	24
2.4 PyTorch编程	25
2.4.1 自定义自动微分	25

目录	2
2.4.2 动态计算图	26
2.4.3 自定义数据集	27
2.4.4 线性回归	28
2.4.5 神经网络分类器	30
2.4.6 卷积神经网络（CNN）进行图像分类	32
2.4.7 循环神经网络（RNN）进行序列预测	34
2.4.8 自定义数据集和数据加载器	38
3 困难与解决方案	40
3.1 调试及性能分析	40
3.2 元编程演示	41
3.3 PyTorch编程	41
4 心得体会	41
5 github网址	42

1 实验目的

掌握调试及性能分析，元编程演示实验，大杂烩，PyTorch编程。

2 实例展示

进行调试及性能分析，元编程演示实验，PyTorch编程实例展示。

2.1 调试及性能分析

调试及性能分析展示

2.1.1 在 Linux上使用可获取最后一天的超级用户访问权限和命令。

1. 在 Linux 上使用 journalctl来查看系统日志并检查超级用户执行的命令。

- 命令展示

```
sudo journalctl
```

```
sudo journalctl --since "yesterday"
```

```
sudo journalctl _UID=0 --since "yesterday"
```

```
sudo journalctl -u ssh --since "yesterday"
```

- 效果展示

- 效果展示

```

Sep 15:14:24:8 LAPTOP-7P56H910 kernel: Linux version 5.15.153.1-microsoft-standard-WSL2 (root@wslgit0176[0wsl1] gcc (GCC)
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: boot loader: systemd-boot, cmdline: linuxrdm.lmg_WSL_ROOT_INIT=1 panic=0 nr_cpus=20 hv_utils.c
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: MEMBLK supported config
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: Intel CemuIntel
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: AMD Authentic
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: Containr CentaurosX
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: BIOS-provided physical RAM map:
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: BIOS-e820: [mem 0x0000000000000000-0x0000000000000000] usable
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: BIOS-e820: [mem 0x0000000000000000-0x0000000000000000] reserved
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: BIOS-e820: [mem 0x0000000000000000-0x0000000000000000] ACPI data
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: BIOS-e820: [mem 0x0000000000000000-0x0000000000000000] usable
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: BIOS-e820: [mem 0x0000000000000000-0x0000000000000000] usable
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: NX (Execute Disable) protection: active
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: DMAR not present or invalid.
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: Hypervisor detected: Microsoft Hyper-V
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: Hypervisor: guestid: Windows 10, arch: amd64, high38030, hints 0x04020, misc 0x00000
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: Hypervisor-Vhost Built: 220211-10-04-A-4169
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: Hypervisor-Vhost Features: 0x300001
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: Hypervisor: LPTIC: 0x0000000000000000
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: Hypervisor-V: Using hypercall for remote I/O flush
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: Hypervisor: Using hypercall for remote I/O flush
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: tsc: Detected [mem 0x0000000000000000-0x0000000000000000] page: max_cycles: 0xffffffffffffffff max_cycles: 0
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: tsc: Update [mem 0x0000000000000000-0x0000000000000000]
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: e20: Detected [mem 0x0000000000000000-0x0000000000000000] usable => reserved
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: e20: reserved [mem 0x0000000000000000-0x0000000000000000]
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: last Phys: 0x00f400 max_arch_phys: 0x0000000000000000
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: vmlinux/PAR: Configuration (E) W: UC MS UC UC UC WB MB WP UC-WT
Sep 15:14:24:8 LAPTOP-7P56H910 kernel: 144 Phys: 0x00f400 max_arch_phys: 0x0000000000000000

```

(b) 效果

图 1: 效果展示

```
root@LAPTOP-7P6PHHQQ:~# sudo journalctl -u ssh --since "yesterday"
Sep 15 14:24:03 LAPTOP-7P6PHHQQ systemd[1]: Starting OpenBSD Secure Shell server...
Sep 15 14:24:04 LAPTOP-7P6PHHQQ sshd[280]: Server listening on 0.0.0.0 port 22.
Sep 15 14:24:04 LAPTOP-7P6PHHQQ sshd[280]: Server listening on :: port 22.
Sep 15 14:24:04 LAPTOP-7P6PHHQQ systemd[1]: Started OpenBSD Secure Shell server.
```

(b) 效果

图 2: 效果展示

1. 获在编辑器中安装 linter 插件，以便自动获取警告。

检查脚本

修复后的脚本

```
#!/bin/bash
```

```
## Example: a typical script with several problems
```

```
shopt -s nullglob # This handles the case of no .m3u files gracefully
```

```
for f in *.m3u; do
```

```
if grep -qi 'hq.*mp3' "$f"; then
```

```
echo "Playlist $f contains a HQ file in mp3 format"

fi

done
```

- 效果展示

```
root@LAPTOP-7P6PHHNQ:/net/e/作业/系统开发基础# sudo apt-get update
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:2 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1306 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [2823 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [1295 kB]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/main amd64 c-n-f Metadata [13.3 kB]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [2377 kB]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/restricted Translation-en [409 kB]
Get:11 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [982 kB]
Get:12 http://security.ubuntu.com/ubuntu jammy-security/universe Translation-en [176 kB]
Get:13 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 c-n-f Metadata [19.2 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [352 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [17.8 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [2437 kB]
Get:17 http://archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [419 kB]
Get:18 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1124 kB]
Get:19 http://archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [251 kB]
Get:20 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 c-n-f Metadata [26.1 kB]
Get:21 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [28.8 kB]
Get:22 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 c-n-f Metadata [672 B]
Fetched 13.1 MB in 4s (2912 kB/s)
Reading package lists... Done
root@LAPTOP-7P6PHHNQ:/net/e/作业/系统开发基础# sudo apt-get install shellcheck
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  shellcheck
0 upgraded, 1 newly installed, 0 to remove and 12 not upgraded.
Need to get 2359 kB of archives.
After this operation, 16.3 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/universe amd64 shellcheck amd64 0.8.0-2 [2359 kB]
Fetched 2359 kB in 4s (597 kB/s)
Selecting previously unselected package shellcheck.
(Reading database ... 27162 files and directories currently installed.)
Preparing to unpack .../shellcheck_0.8.0-2_amd64.deb ...
Unpacking shellcheck (0.8.0-2) ...
Setting up shellcheck (0.8.0-2) ...
```

(a) 效果

```
root@LAPTOP-7P6PHHNQ:/n x + v
line 37:
E092: Not an editor command: :PlugInstall
line 42:
E117: Unknown function: plug#begin
line 45:
E092: Not an editor command: Plug 'neovide/coc.nvim', {'branch': 'release'}
line 47:
E117: Unknown function: plug#end
line 88:
E117: Unknown function: vundle#begin
line 83:
E092: Not an editor command: Plugin 'Vundle/vim/Vundle.vim'
line 95:
E092: Not an editor command: Plugin 'ctrlpvim/ctrlp.vim'
line 92:
E117: Unknown function: vundle#end
Press ENTER or type command to continue
root@LAPTOP-7P6PHHNQ:/net/e/作业/系统开发基础# shellcheck check_playlist.sh
In check_playlist.sh line 3:
for f in $(ls *sh*)
~ SC2060 (error): Iterating over ls output is fragile. Use glob.
~ SC2058 (info): Use ./glob* or --glob* as names with dashes won't become options.

In check_playlist.sh line 5:
grep -q1 hq-mp3 $f \
~ SC2062 (warning): Quote the grep pattern so the shell won't interpret it.
~ SC2058 (info): Double quote to prevent globbing and word splitting.

Did you mean:
grep -q1 hq-mp3 "$f" \

In check_playlist.sh line 6:
do echo -e "Playlist $f contains a HQ file in mp3 format"
~ SC2057 (warning): In POSIX sh, echo flags are undefined.
~ SC2058 (info): Expressions don't expand in single quotes, use double quotes for that.

For more information:
https://www.shellcheck.net/wiki/SC2060 -- Iterating over ls output is fragi...
https://www.shellcheck.net/wiki/SC2062 -- Quote the grep pattern so the she...
https://www.shellcheck.net/wiki/SC2057 -- In POSIX sh, echo flags are undef...
root@LAPTOP-7P6PHHNQ:/net/e/作业/系统开发基础#
```

(b) 效果

图 3: 效果展示

2.1.3 查看占用进程的pid

1. 侦听的端口已被另一个进程占用，找到该进程 pid 并通过运行终止它。

- 命令展示

```
Linux

python3 -m http.server 8000

lsof -i :8000 | grep LISTEN

kill 12345
```

```
Windows

python -m http.server 8000

netstat -aon | findstr :8000

taskkill /PID 12345 /F
```

- 效果展示

```
C:\Users\lenovo>netstat -asn | findstr 8000
TCP    0.0.0.0:8000          0.0.0.0:0          LISTENING        3352
TCP    127.0.0.1:8000      0.0.0.0:0          LISTENING        10772
TCP    [::]:8000          [::]:0             LISTENING        3352
TCP    [240c:ca82:2152:d48:fd7b:cd19:579c:1669]:53409 [2686:58c0:8000:1153]:443 ESTABLISHED      25912

C:\Users\lenovo>taskkill /PID 3352 /F
成功: 已终止 PID 为 3352 的进程。

C:\Users\lenovo>taskkill /PID 10772 /F
成功: 已终止 PID 为 10772 的进程。

C:\Users\lenovo>
```

(a) 效果

```
Windows PowerShell  X  root@LAPTOP-7P6I  X  root@LAPTOP-7P6  X  +  v  -
root@LAPTOP-7P6PHNQ:~# lsof -i :8000 | grep LISTEN
python3 42397 root    3u  IPv4 708113      0t0  TCP *:8000 (LISTEN)
root@LAPTOP-7P6PHNQ:~# kill 42397
root@LAPTOP-7P6PHNQ:~# lsof -i :8000 | grep LISTEN
```

(b) 效果

图 4: 效果展示

2.1.4 运行并使用可视化 CPU 消耗。

1. 运行并使用可视化 CPU 消耗

- 命令展示

安装 stress 工具

```
sudo apt-get update
```

```
sudo apt-get install stress
```

```
stress -c 3
```

```
taskset --cpu-list 0,2 stress -c 3
```

查看 CPU 使用情况

```
sudo apt-get install htop
```

```
htop
```

创建和使用 cgroups

```
sudo apt-get install cgroup-tools
```

```
sudo cgcreate -g cpu,memory:/mygroup
```

```
echo 50000 | sudo tee /sys/fs/cgroup/cpu/mygroup/cpu.cfs_quota_us
```

```
echo 100000 | sudo tee /sys/fs/cgroup/cpu/mygroup/cpu.cfs_period_us
```

```
echo 100M | sudo tee /sys/fs/cgroup/memory/mygroup/memory.limit_in_bytes
```

```
sudo cgexec -g cpu,memory:/mygroup stress -c 3 -m 1
```

使用 cgroups 查看和调整资源限制

```
cat /sys/fs/cgroup/cpu/mygroup/cpu.cfs_quota_us
```

```
cat /sys/fs/cgroup/memory/mygroup/memory.limit_in_bytes
```

- 效果展示

• 效果展示

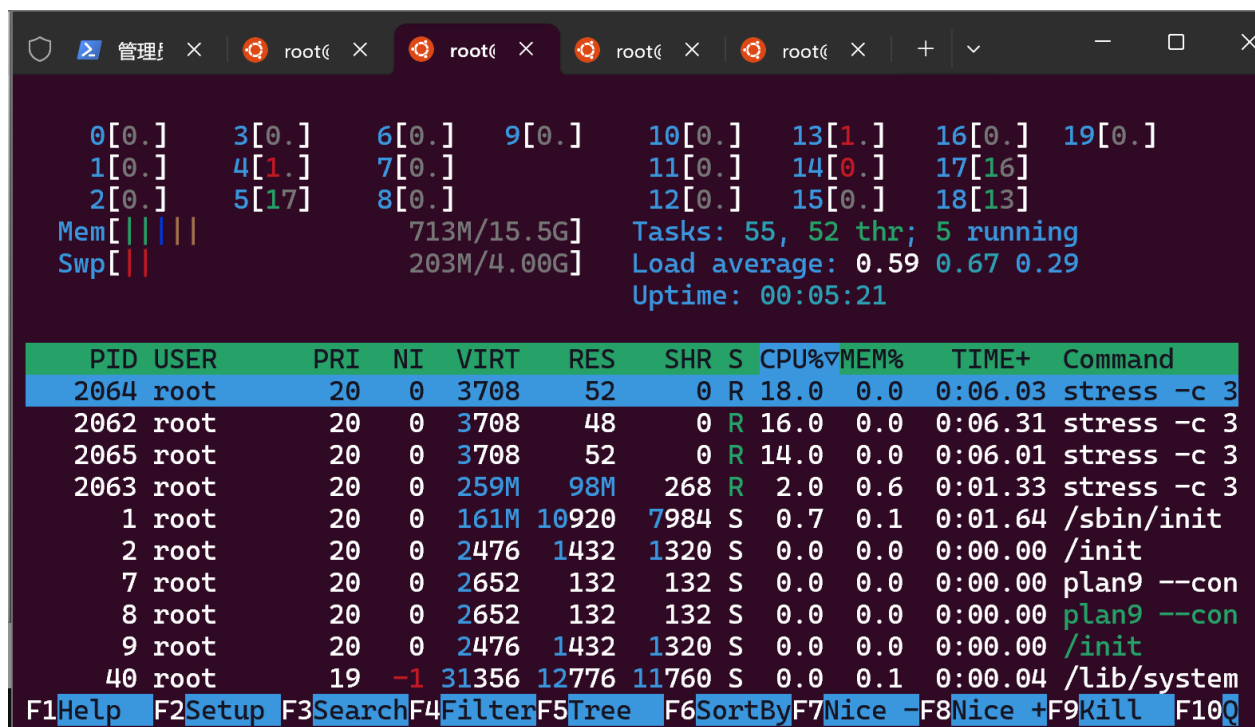


图 5: 列表效果展示

2.1.5 执行 HTTP 请求并获取有关您的公有 IP 的信息。

1. 打开 Wireshark 并尝试嗅探发送和接收的请求和回复数据包。

• 命令展示

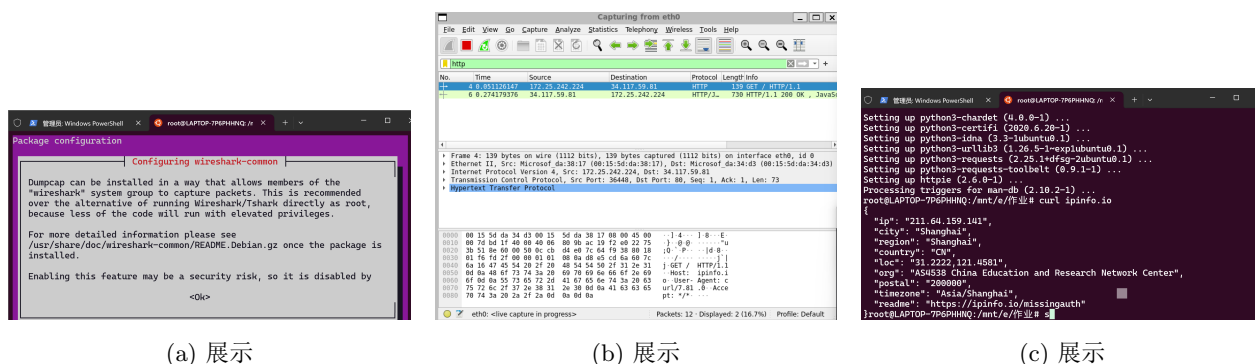
```
sudo apt update
sudo apt install wireshark
sudo dnf install wireshark
sudo usermod -aG wireshark $USER
```

```
curl --version
```

```
sudo apt update
sudo apt install httpie
```

```
http
curl ipinfo.io
```


- 效果展示



(a) 展示

(b) 展示

(c) 展示

图 6: 效果展示

2.1.6 使用 GDB 调试 C/C++ 程序

- 使用一个简单的程序来计算数组的平均值，但程序中存在一个错误。使用 GDB 来调试并找出问题所在。

- 命令展示

c程序

```
#include <stdio.h>
```

```
float calculate_average(int *arr, int size) {
    int sum = 0;
    for (int i = 0; i <= size; i++) { // 注意：这里有一个错误，应该是 i < size
        sum += arr[i];
    }
    return (float)sum / size;
}
```

```
int main() {
    int numbers[] = {1, 2, 3, 4, 5};
    int size = sizeof(numbers) / sizeof(numbers[0]);
    float average = calculate_average(numbers, size);
    printf("Average: %f\n", average);
    return 0;
}
```

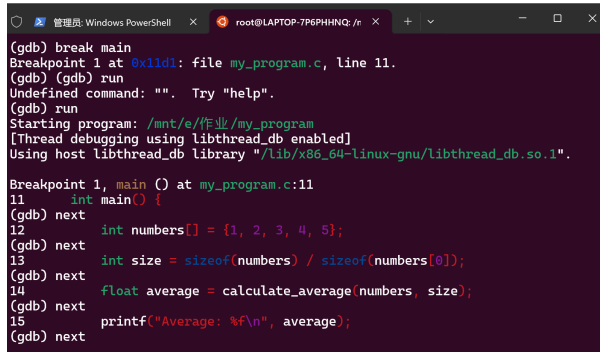
```
gcc -g -o my_program my_program.c
```

```
gdb ./my_program
```

```
(gdb) break main
```

```
(gdb) run
(gdb) print my_variable
```

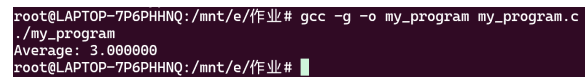
- 效果展示



```
(gdb) break main
Breakpoint 1 at 0x11d1: file my_program.c, line 11.
(gdb) (gdb) run
Undefined command: "". Try "help".
(gdb) run
Starting program: /mnt/e/作业/my_program
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at my_program.c:11
11   int main() {
(gdb) next
12   int numbers[] = {1, 2, 3, 4, 5};
(gdb) next
13   int size = sizeof(numbers) / sizeof(numbers[0]);
(gdb) next
14   float average = calculate_average(numbers, size);
(gdb) next
15   printf("Average: %f\n", average);
(gdb) next
```

(a) 效果



```
root@LAPTOP-7P6PHHNQ:/mnt/e/作业# gcc -g -o my_program my_program.c
./my_program
Average: 3.000000
root@LAPTOP-7P6PHHNQ:/mnt/e/作业#
```

(b) 修改之后

图 7: 效果展示

2.1.7 使用 strace 跟踪系统调用

1. 使用 strace 跟踪和分析一个简单 C 程序的系统调用。

- 命令展示

c程序

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main() {
    FILE *file = fopen("example.txt", "r");
    if (file == NULL) {
        perror("Failed to open file");
        return EXIT_FAILURE;
    }

    char buffer[256];
    while (fgets(buffer, sizeof(buffer), file)) {
        printf("%s", buffer);
    }

    fclose(file);
    return EXIT_SUCCESS;
}
```

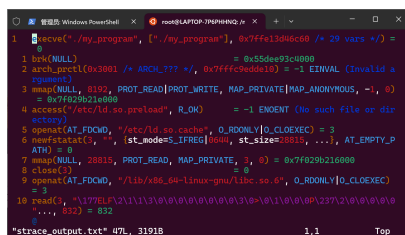
```
}

echo "This is a test file."

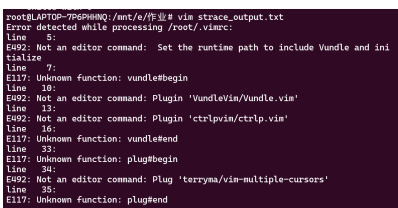
gcc -o my_program my_program.c

strace -o strace_output.txt ./my_program
strace -e trace=open,close,read,write ./my_program
```

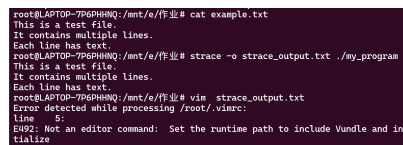
- 效果展示



(a) 展示



(b) 展示



(c) 展示

图 8: 效果展示

2.1.8 检测内存泄漏

1. 使用 Valgrind 执行检测 C/C++ 程序内存泄漏和其他内存问题

- 命令展示

```
sudo apt-get install valgrind

valgrind --leak-check=full ./my_program
```

c 程序

```
#include <stdio.h>
#include <stdlib.h>

void create_memory_leak() {
    int *array = malloc(10 * sizeof(int)); // 分配内存但没有释放
    if (!array) {
        fprintf(stderr, "Memory allocation failed\n");
    }
}
```

```
        return;
    }
    for (int i = 0; i < 10; i++) {
        array[i] = i * i;
    }
    // 忘记调用 free(array);
}

int main() {
    create_memory_leak();
    return 0;
}

gcc -g -o my_program my_program.c
修复内存泄漏
#include <stdio.h>
#include <stdlib.h>

void create_memory_leak() {
    int *array = malloc(10 * sizeof(int));
    if (!array) {
        fprintf(stderr, "Memory allocation failed\n");
        return;
    }
    for (int i = 0; i < 10; i++) {
        array[i] = i * i;
    }
    free(array); // 释放内存
}

int main() {
    create_memory_leak();
    return 0;
}
```

- 效果展示

```
Press ENTER or type command to continue
root@LAPTOP-7P6PHHNQ:/mnt/e/作业# gcc -g -o my_program my_program.c
root@LAPTOP-7P6PHHNQ:/mnt/e/作业# valgrind --leak-check=full ./my_program
==59408== Memcheck, a memory error detector
==59408== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==59408== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==59408== Command: ./my_program
==59408==
==59408== HEAP SUMMARY:
==59408==   in use at exit: 40 bytes in 1 blocks
==59408== total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==59408==
==59408== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==59408==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memche
ck-amd64-linux.so)
==59408==    by 0x10917E: create_memory_leak (my_program.c:5)
==59408==    by 0x1091F1: main (my_program.c:17)
==59408==
==59408== LEAK SUMMARY:
==59408==   definitely lost: 40 bytes in 1 blocks
==59408==   indirectly lost: 0 bytes in 0 blocks
==59408==   possibly lost: 0 bytes in 0 blocks
==59408==   still reachable: 0 bytes in 0 blocks
==59408==   suppressed: 0 bytes in 0 blocks
==59408==
==59408== For lists of detected and suppressed errors, rerun with: -s
==59408== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

(a) 效果

```
==59408== HEAP SUMMARY:
==59408==   in use at exit: 40 bytes in 1 blocks
==59408== total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==59408==
==59408== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==59408==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memche
ck-amd64-linux.so)
==59408==    by 0x10917E: create_memory_leak (my_program.c:5)
==59408==    by 0x1091F1: main (my_program.c:17)
==59408==
==59408== LEAK SUMMARY:
==59408==   definitely lost: 40 bytes in 1 blocks
==59408==   indirectly lost: 0 bytes in 0 blocks
==59408==   possibly lost: 0 bytes in 0 blocks
==59408==   still reachable: 0 bytes in 0 blocks
==59408==   suppressed: 0 bytes in 0 blocks
==59408==
==59408== For lists of detected and suppressed errors, rerun with: -s
==59408== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

(b) 效果

图 9: 效果展示

2.1.9 使用 perf 分析程序性能

1. 使用 perf 分析程序性能

- 命令展示

c程序

```
#include <stdio.h>
#include <stdlib.h>
```

```
void compute_squares(int *array, int size) {
    for (int i = 0; i < size; i++) {
        array[i] = i * i;
    }
}

int main() {
    int size = 1000000; // 1 million elements
    int *array = malloc(size * sizeof(int));
    if (!array) {
        fprintf(stderr, "Memory allocation failed\n");
        return 1;
    }

    compute_squares(array, size);

    printf("Computation completed\n");
    free(array);
    return 0;
}
```

```
sudo apt-get update
sudo apt-get install linux-tools-common linux-tools-generic

gcc -O2 -o my_program my_program.c
perf record -g ./my_program
perf report
```

● 效果展示

```
root@LAPTOP-7P6PHHNQ:~# sudo journalctl
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: Linux version 5.15.153.1-microsoft-standard-WSL2 (root@941d701f84f1) (gcc (GCC)>
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: Command line: initrd=\initrd.img WSL_ROOT_INIT=1 panic=-1 nr_cpus=20 hv_utils.t>
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: KERNEL supported cpus:
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: Intel GenuineIntel
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: AMD AuthenticAMD
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: Centaur CentaurHauls
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: BIOS-provided physical RAM map:
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: BIOS-e820: [mem 0x0000000000000000-0x0000000000009ffff] usable
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: BIOS-e820: [mem 0x000000000000e000-0x000000000000e0fff] reserved
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: BIOS-e820: [mem 0x0000000000010000-0x000000000001ffff] ACPI data
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: BIOS-e820: [mem 0x0000000000020000-0x000000000002ffff] usable
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: BIOS-e820: [mem 0x0000000000030000-0x000000000003ffff] usable
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: NX (Execute Disable) protection: active
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: DMI not present or invalid.
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: Hypervisor detected: Microsoft Hyper-V
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: Hyper-V: privilege flags low 0xae7f, high 0x3b8030, hints 0xa4e24, misc 0xe0bed>
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: Hyper-V Host Build:22621-10.0-4-0.4036
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: Hyper-V: Nested features: 0x3e0101
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: Hyper-V: LAPIC Timer Frequency: 0x1e8480
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: Hyper-V: Using hypercall for remote TLB flush
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: clocksource: hyperv_clocksource_tsc_page: mask: 0xffffffffffffffff max_cycles: >
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: tsc: Detected 2995.200 MHz processor
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: e820: update [mem 0x00000000-0x00000fff] usable ==> reserved
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: e820: remove [mem 0x000a0000-0x000fffff] usable
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: last_pfn = 0x3fff400 max_arch_pfn = 0x400000000
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: x86/PAT: Configuration [0-7]: WB WC UC- UC WB WP UC- WT
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: last_pfn = 0xf8000 max_arch_pfn = 0x400000000
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: Using GB pages for direct mapping
Aug 30 14:25:25 LAPTOP-7P6PHHNQ kernel: Incomplete global flushes, disabling PCID
```

图 10: 列表效果展示

2.2 元编程演示实验

元编程演示实验展示

2.2.1 make 重新构建

1. 在 Makefile 中实现一个 clean 目标，删除构建过程中生成的所有文件

● 命令展示

```
list = [1, 2, 3, 'frist', 9.0]
ok = [99, 'I am ']
ok.extend(list)
print(list)
print(list[0])
```

```
print(list[1:3])
print(list[2:])
print(ok * 2)
print(list + ok)
print(ok)
print(ok.index('frist'))
print(ok.count('frist'))
print(ok.pop(2))
print(ok)
ok.reverse()
print(ok)
```

- 效果展示



```
root@LAPTOP-7P6PHHNQ:/mnt/e/作业# make
./plot.py -i data.dat -o plot-data.png
pdflatex paper.tex
```

2.2.2 动态创建类和方法

1. 用元类（metaclass）来动态创建一个具有加法、减法和乘法功能的数学类

- 命令展示

```
# 定义一个元类
class MathMeta(type):
    def __new__(cls, name, bases, attrs):
        # 动态添加方法
        def add(self, a, b):
            return a + b

        def subtract(self, a, b):
            return a - b

        def multiply(self, a, b):
            return a * b

        # 把新方法添加到类属性中
```

```

        attrs['add'] = add
        attrs['subtract'] = subtract
        attrs['multiply'] = multiply

    # 创建类
    return super().__new__(cls, name, bases, attrs)

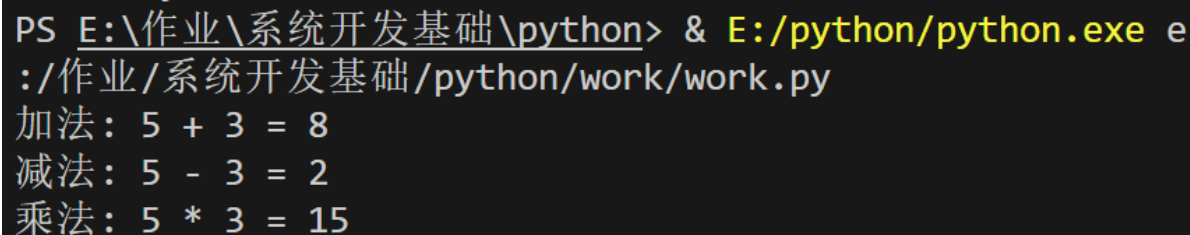
# 使用元类创建一个新的数学类
class MathOperations(metaclass=MathMeta):
    pass

# 使用动态创建的类
math_op = MathOperations()

# 测试动态添加的方法
print("加法: 5 + 3 =", math_op.add(5, 3))          # 输出: 加法: 5 + 3 = 8
print("减法: 5 - 3 =", math_op.subtract(5, 3))      # 输出: 减法: 5 - 3 = 2
print("乘法: 5 * 3 =", math_op.multiply(5, 3))      # 输出: 乘法: 5 * 3 = 15

```

- 效果展示



```

PS E:\作业\系统开发基础\python> & E:/python/python.exe e
:/作业/系统开发基础/python/work/work.py
加法: 5 + 3 = 8
减法: 5 - 3 = 2
乘法: 5 * 3 = 15

```

图 11: 效果展示

2.2.3 动态属性添加

1. 创建一个类，它可以在运行时动态添加属性。

- 命令展示

```

class DynamicAttributes:
    def __init__(self):
        self.existing_attribute = "I already exist!"

    def __setattr__(self, name, value):
        print(f"Setting attribute '{name}' to '{value}'")

```

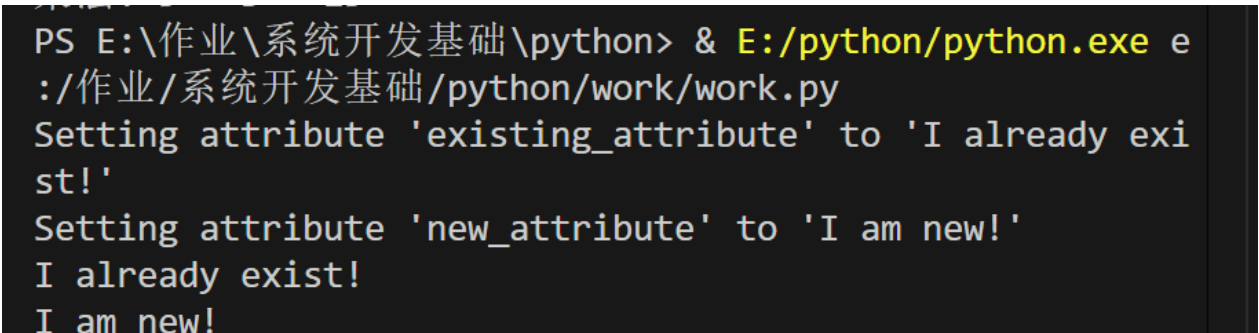


```
super().__setattr__(name, value)

# 创建实例
obj = DynamicAttributes()

# 添加动态属性
obj.new_attribute = "I am new!"
print(obj.existing_attribute)
print(obj.new_attribute)
```

- 效果展示



```
PS E:\作业\系统开发基础\python> & E:/python/python.exe e
:/作业/系统开发基础/python/work/work.py
Setting attribute 'existing_attribute' to 'I already exist!'
Setting attribute 'new_attribute' to 'I am new!'
I already exist!
I am new!
```

图 12: 效果展示

2.2.4 使用元类进行属性验证

1. 定义一个元类，用于在创建类时验证某些属性的类型。

- 命令展示

```
class TypedMeta(type):
    def __new__(cls, name, bases, attrs):
        # 检查属性类型
        for attr_name, attr_value in attrs.items():
            if isinstance(attr_value, type):
                attrs[attr_name] = attr_value()

        return super().__new__(cls, name, bases, attrs)
```

```
class Person(metaclass=TypedMeta):
    name = str
    age = int

    def __init__(self):
        self.name = "John Doe"
        self.age = 30

# 使用示例
person = Person()
print(f"Name: {person.name}, Age: {person.age}")
# 输出: Name: John Doe, Age: 30
```

- 效果展示

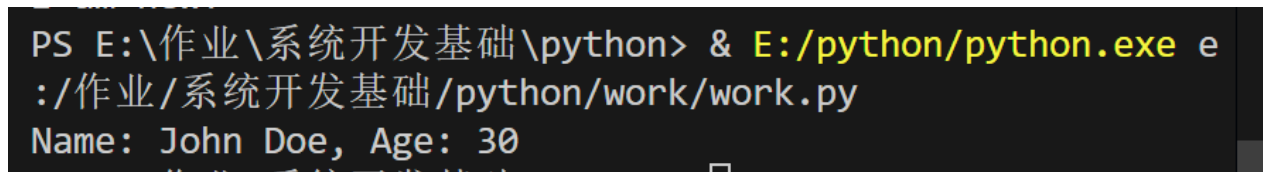


图 13: 效果展示

2.2.5 使用装饰器进行方法计时

1. 使用装饰器来记录方法调用的时间。

- 命令展示

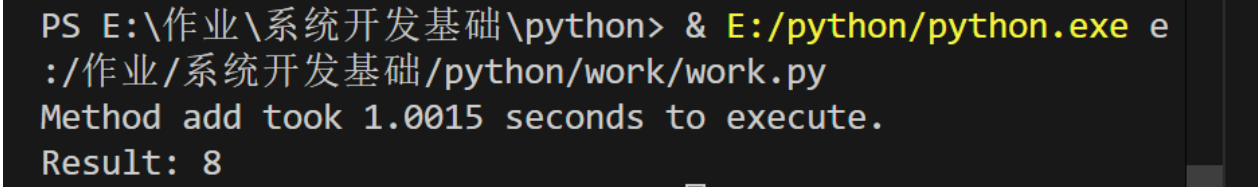
```
import time

def timing_decorator(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"Method {func.__name__} took {end_time - start_time:.4f} seconds to execute")
        return result
    return wrapper
```

```
class MathOperations:
    @timing_decorator
    def add(self, a, b):
        time.sleep(1) # 模拟耗时操作
        return a + b

# 使用示例
math_op = MathOperations()
result = math_op.add(5, 3)
print("Result:", result) # 输出: Result: 8
```

- 效果展示



```
PS E:\作业\系统开发基础\python> & E:/python/python.exe e
:/作业/系统开发基础/python/work/work.py
Method add took 1.0015 seconds to execute.
Result: 8
```

图 14: 效果展示

2.2.6 使用元类实现只读属性

1. 创建一个元类，该元类会将指定的属性设置为只读。即这些属性只能在对象创建时赋值，之后不能再次修改。

- 命令展示

```
class ReadOnlyMeta(type):
    def __new__(cls, name, bases, attrs):
        # 找到以 _readonly_ 开头的属性并将其设置为只读
        for attr_name, attr_value in attrs.items():
            if isinstance(attr_value, str) and attr_name.startswith('_readonly_'):
                # 创建新的属性，只能在初始化时赋值
                original_value = attr_value

        def getter(self, name=attr_name):
            return getattr(self, "_" + name)
```

```
# 创建一个设置器，抛出异常
def setter(self, value, name=attr_name):
    raise AttributeError(f"{name} 是只读属性，不能被修改。")

# 将属性设置为只读
attrs[attr_name] = property(getter, setter)
attrs["_" + attr_name] = original_value

return super().__new__(cls, name, bases, attrs)

class Person(metaclass=ReadOnlyMeta):
    _readonly_name = "John Doe"
    _readonly_age = 30

# 使用示例
person = Person()
print(person._readonly_name) # 输出: John Doe
print(person._readonly_age) # 输出: 30

# 尝试修改只读属性
try:
    person._readonly_name = "Jane Doe"
except AttributeError as e:
    print(e) # 输出: _readonly_name 是只读属性，不能被修改。
```

- 效果展示

```
Result: 0
PS E:\作业\系统开发基础\python> & E:/python/python.exe e
:/作业/系统开发基础/python/work/work.py
Traceback (most recent call last):
  File "e:\作业\系统开发基础\python\work\work.py", line
22, in <module>
    class Person(metaclass=ReadOnlyMeta):
  File "e:\作业\系统开发基础\python\work\work.py", line
4, in __new__
    for attr_name, attr_value in attrs.items():
RuntimeError: dictionary changed size during iteration
```

图 15: 效果展示

2.2.7 使用装饰器进行方法日志记录

1. 创建一个装饰器，用于记录每次方法调用的日志。

- 命令展示

```
def log_method_call(func):
    def wrapper(*args, **kwargs):
        print(f"Calling method: {func.__name__} with arguments: {args}, {kwargs}")
        result = func(*args, **kwargs)
        print(f"Method {func.__name__} returned: {result}")
        return result
    return wrapper

class Calculator:
    @log_method_call
    def add(self, a, b):
        return a + b

    @log_method_call
    def multiply(self, a, b):
        return a * b

# 使用示例
calc = Calculator()
result_add = calc.add(5, 3)          # 记录调用
result_multiply = calc.multiply(4, 2) # 记录调用
```

- 效果展示

```
RuntimeError: dictionary changed size during iteration
PS E:\作业\系统开发基础\python> & E:/python/python.exe e
:/作业/系统开发基础/python/work/work.py
Calling method: add with arguments: (<__main__.Calculator
object at 0x0000029D92ADBE00>, 5, 3), {}
Method add returned: 8
Calling method: multiply with arguments: (<__main__.Calc
ulator object at 0x0000029D92ADBE00>, 4, 2), {}
Method multiply returned: 8
```

图 16: 效果展示

2.3 大杂烩

大杂烩展示

2.3.1 Markdown表格

1. Markdown支持使用管道符号（—）创建表格。表格可以非常方便地用于展示数据。

- 命令展示

姓名	年龄	职业
Alice	30	软件工程师
Bob	25	数据分析师
Charlie	35	产品经理

- 效果展示



姓名	年龄	职业
Alice	30	软件工程师
Bob	25	数据分析师
Charlie	35	产品经理

图 17: 效果展示

2.3.2 Markdown嵌套列表

1. 子项目之前添加空格或制表符来实现。

- 命令展示

1. 第一项

- 子项 1

- 子项 2

2. 第二项

1. 子子项 1

2. 子子项 2

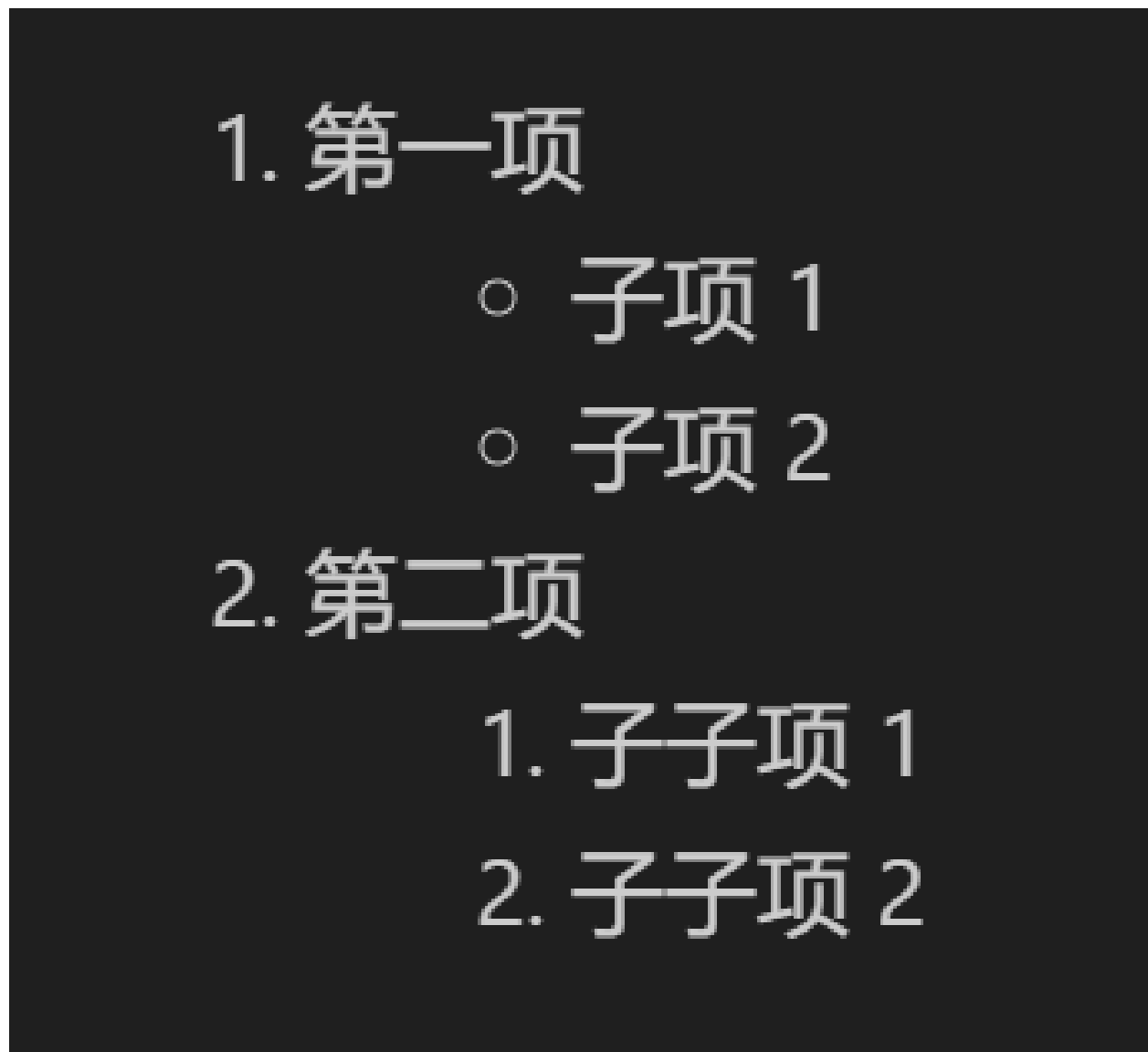


图 18: 效果展示

2.3.3 引用块

1. 引用块可以用于引用其他人的话或文献，也可以嵌套使用

- 命令展示

- > 这是一个引用块。

- >

- > > 这是一个嵌套引用块。

- 效果展示

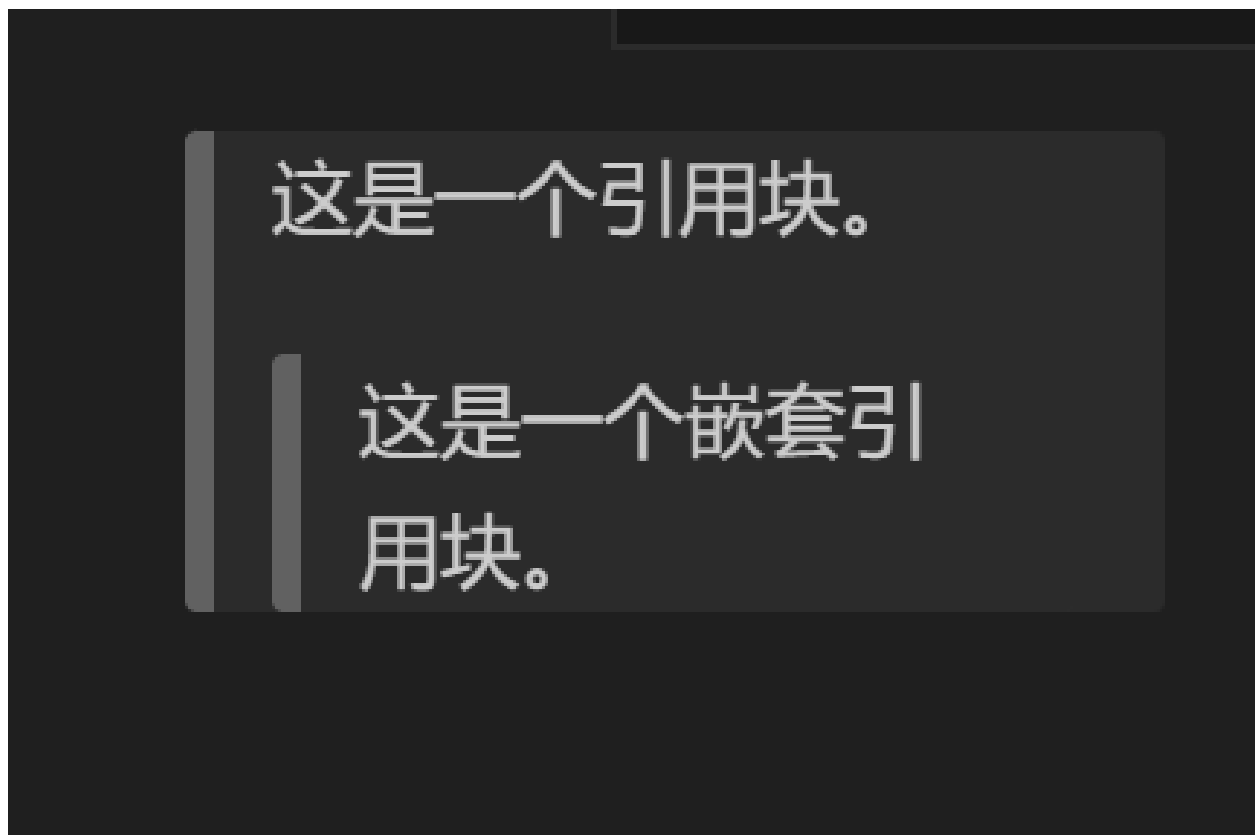


图 19: 效果展示

2.3.4 任务列表

1. 使用 - [] 和 - [x] 来表示未完成和已完成的任务。

- 命令展示

- [] 任务 1
- [x] 任务 2
- [] 任务 3

- 效果展示



图 20: 效果展示

2.3.5 自定义链接和图片

1. 可以使用方括号和圆括号来实现。

- 命令展示

[GitHub](<https://github.com>) 是一个流行的代码托管平台。

![示例图片](<https://via.placeholder.com/150> "示例图片")

- 效果展示

GitHub 是一个流行的代码
托管平台。

150 x 150

2.4 PyTorch编程

PyTorch编程展示

2.4.1 自定义自动微分

1. 自动微分功能允许定义自定义的前向和反向传播逻辑。

- 命令展示

```
import torch
from torch.autograd import Function

class MyReLU(Function):
    @staticmethod
    def forward(ctx, input):
        ctx.save_for_backward(input)
        return input.clamp(min=0)

    @staticmethod
    def backward(ctx, grad_output):
        input, = ctx.saved_tensors
        grad_input = grad_output.clone()
        grad_input[input < 0] = 0
        return grad_input

# 使用自定义ReLU
x = torch.tensor([-1.0, 2.0, 3.0], requires_grad=True)
relu = MyReLU.apply
y = relu(x)
y.sum().backward()
print(x.grad) # Output: tensor([0., 1., 1.] )
```

- 效果展示

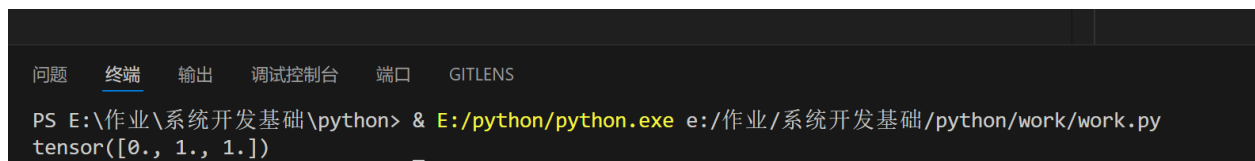


图 21: 效果展示

2.4.2 动态计算图

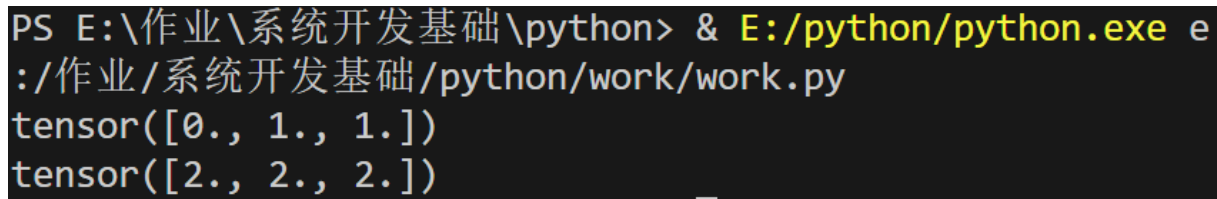
1. PyTorch 的计算图是动态的, 允许在每次迭代中更改模型的计算逻辑

- 命令展示

```
x = torch.tensor([1.0, 2.0, 3.0], requires_grad=True)
```

```
y = x * 2 if x.sum() > 0 else x / 2
y.sum().backward()
print(x.grad) # Output: tensor([2., 2., 2.]
```

- 效果展示
- 效果展示



```
PS E:\作业\系统开发基础\python> & E:/python/python.exe e
:/作业/系统开发基础/python/work/work.py
tensor([0., 1., 1.]
tensor([2., 2., 2.]
```

图 22: 效果展示

2.4.3 自定义数据集

1. 使用 PyTorch 的 Dataset 和 DataLoader 来处理自定义数据

- 命令展示

```
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):
    def __init__(self, data, labels):
        self.data = data
        self.labels = labels

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return self.data[idx], self.labels[idx]

data = torch.randn(100, 10)
labels = torch.randint(0, 2, (100,))
dataset = MyDataset(data, labels)
dataloader = DataLoader(dataset, batch_size=5, shuffle=True)
```

- 效果展示

```
Epoch [10/100], Loss: 1.7161
Epoch [20/100], Loss: 0.1478
Epoch [30/100], Loss: 0.1012
Epoch [40/100], Loss: 0.0943
Epoch [50/100], Loss: 0.0888
Epoch [60/100], Loss: 0.0836
Epoch [70/100], Loss: 0.0787
Epoch [80/100], Loss: 0.0742
Epoch [90/100], Loss: 0.0698
Epoch [100/100], Loss: 0.0658
Predicted values: [[2.4128728]
 [4.200065 ]
 [5.9872575]
 [7.77445  ]]
```

图 23: 自定义数据集的示例

2.4.4 线性回归

1. 线性回归分析

- 命令展示

```
import torch
import torch.nn as nn
import torch.optim as optim

\# 定义简单的线性模型
class LinearRegressionModel(nn.Module):
    def __init__(self):
        super(LinearRegressionModel, self).__init__()
        self.linear = nn.Linear(1, 1) \# 输入维度为1, 输出维度为1

    def forward(self, x):
```

```
        return self.linear(x)

\# 准备数据
x_train = torch.tensor([[1.0], [2.0], [3.0], [4.0]], dtype=torch.float32)
y_train = torch.tensor([[2.0], [4.0], [6.0], [8.0]], dtype=torch.float32)

\# 实例化模型，定义损失函数和优化器
model = LinearRegressionModel()
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

\# 训练模型
num_epochs = 100
for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(x_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

\# 测试模型
model.eval()
with torch.no_grad():
    predicted = model(x_train).numpy()
    print("Predicted values:", predicted)
```

- 效果展示


```
:/作业/系统开发基础/python/work/work.py
Epoch [10/100], Loss: 1.8482
Epoch [20/100], Loss: 0.0546
Epoch [30/100], Loss: 0.0078
Epoch [40/100], Loss: 0.0062
Epoch [50/100], Loss: 0.0058
Epoch [60/100], Loss: 0.0055
Epoch [70/100], Loss: 0.0052
Epoch [80/100], Loss: 0.0049
Epoch [90/100], Loss: 0.0046
Epoch [100/100], Loss: 0.0043
Predicted values: [[1.8940283]
 [3.94865 ]
 [6.003271 ]
 [8.057893 ]]
```

图 24: 线性回归模型的预测结果

2.4.5 神经网络分类器

1. 神经网络分类器

- 命令展示

```
import torch
import torch.nn as nn
import torch.optim as optim

# 生成随机数据
def generate_data(num_samples=1000, num_features=20, num_classes=2):
    X = torch.randn(num_samples, num_features) # 随机生成特征数据
    y = torch.randint(0, num_classes, (num_samples,)) # 随机生成0或1作为标签
    return X, y

# 定义简单的神经网络模型
class SimpleNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
```

```
        super(SimpleNN, self).__init__()
        self.layer1 = nn.Linear(input_size, hidden_size)
        self.layer2 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu(self.layer1(x))
        x = self.layer2(x)
        return x

# 生成数据
X_train, y_train = generate_data()
X_test, y_test = generate_data()

# 转换为PyTorch张量
y_train = y_train.float().view(-1, 1)
y_test = y_test.float().view(-1, 1)

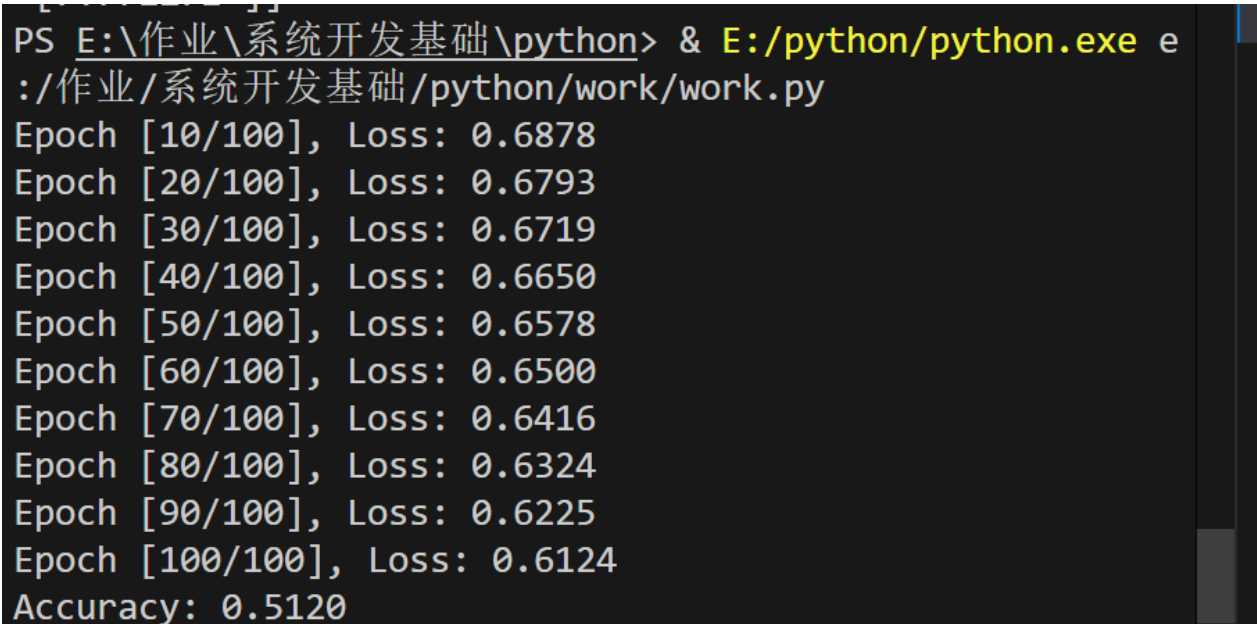
# 实例化模型，定义损失函数和优化器
input_size = X_train.shape[1]
hidden_size = 64
num_classes = 1 # 二分类任务
model = SimpleNN(input_size, hidden_size, num_classes)
criterion = nn.BCEWithLogitsLoss() # 使用带有sigmoid激活的二元交叉熵损失
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 训练模型
num_epochs = 100
for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

```
# 测试模型
model.eval()
with torch.no_grad():
    outputs = model(X_test)
    predicted = (outputs > 0).float()
    accuracy = (predicted.eq(y_test).sum() / y_test.shape[0]).item()
    print(f'Accuracy: {accuracy:.4f}')
```

- 效果展示



A terminal window showing the execution of a Python script. The command is `PS E:\作业\系统开发基础\python> & E:/python/python.exe e:/作业/系统开发基础/python/work/work.py`. The output displays the loss for each of the 100 epochs, showing a steady decrease from 0.6878 to 0.6124. At the end, the accuracy is reported as 0.5120.

```
Epoch [10/100], Loss: 0.6878
Epoch [20/100], Loss: 0.6793
Epoch [30/100], Loss: 0.6719
Epoch [40/100], Loss: 0.6650
Epoch [50/100], Loss: 0.6578
Epoch [60/100], Loss: 0.6500
Epoch [70/100], Loss: 0.6416
Epoch [80/100], Loss: 0.6324
Epoch [90/100], Loss: 0.6225
Epoch [100/100], Loss: 0.6124
Accuracy: 0.5120
```

图 25: 效果展示

2.4.6 卷积神经网络（CNN）进行图像分类

1. 用 PyTorch 中的 CNN 来在随机生成的图像数据上进行分类。

- 命令展示

```
import torch
import torch.nn as nn
import torch.optim as optim

# 定义简单的卷积神经网络
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
```

```
self.layer1 = nn.Conv2d(1, 16, kernel_size=3, stride=1, padding=1)
self.layer2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
self.fc1 = nn.Linear(32*8*8, 128)
self.fc2 = nn.Linear(128, 10)

def forward(self, x):
    x = torch.relu(self.layer1(x))
    x = torch.max_pool2d(x, 2)
    x = torch.relu(self.layer2(x))
    x = torch.max_pool2d(x, 2)
    x = x.view(x.size(0), -1)
    x = torch.relu(self.fc1(x))
    x = self.fc2(x)
    return x

# 生成随机数据
def generate_image_data(num_samples=1000, image_size=(1, 32, 32), num_classes=10):
    X = torch.randn(num_samples, *image_size)
    y = torch.randint(0, num_classes, (num_samples,))
    return X, y

# 生成数据
X_train, y_train = generate_image_data()
X_test, y_test = generate_image_data()

# 实例化模型，定义损失函数和优化器
model = SimpleCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 训练模型
num_epochs = 10
batch_size = 64
for epoch in range(num_epochs):
    model.train()
    for i in range(0, len(X_train), batch_size):
        inputs = X_train[i:i + batch_size]
        targets = y_train[i:i + batch_size]

        optimizer.zero_grad()
```

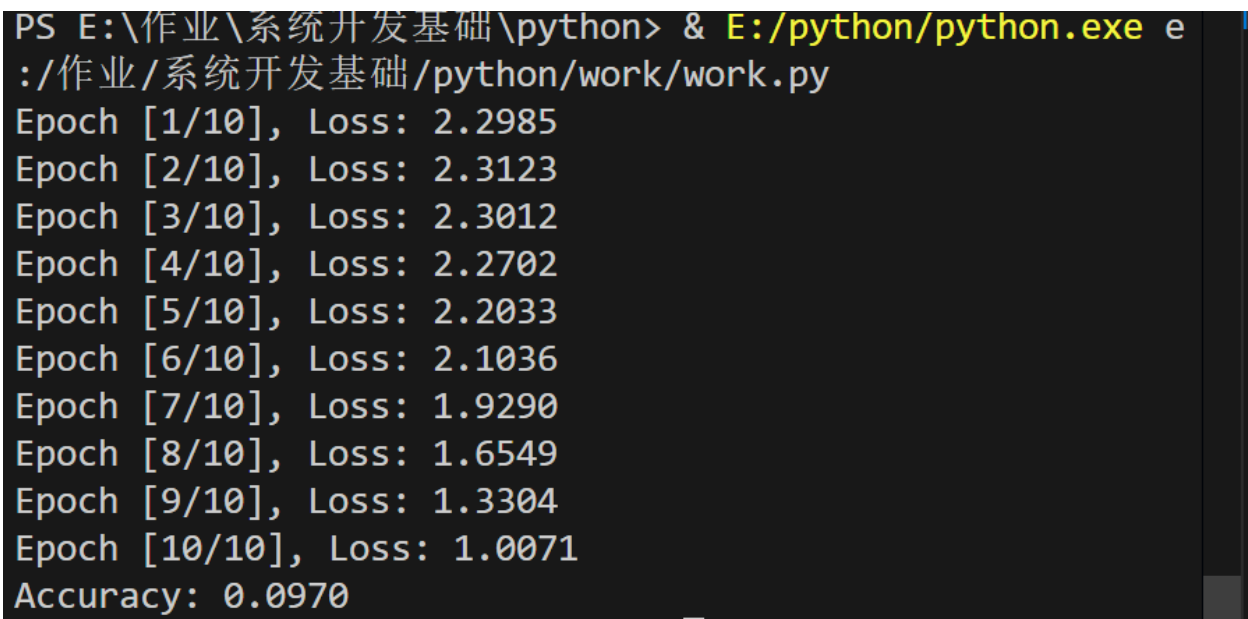
```
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

测试模型

```
model.eval()
with torch.no_grad():
    outputs = model(X_test)
    _, predicted = torch.max(outputs, 1)
    accuracy = (predicted == y_test).sum().item() / y_test.size(0)
    print(f'Accuracy: {accuracy:.4f}')
```

- 效果展示



```
PS E:\作业\系统开发基础\python> & E:/python/python.exe e
:/作业/系统开发基础/python/work/work.py
Epoch [1/10], Loss: 2.2985
Epoch [2/10], Loss: 2.3123
Epoch [3/10], Loss: 2.3012
Epoch [4/10], Loss: 2.2702
Epoch [5/10], Loss: 2.2033
Epoch [6/10], Loss: 2.1036
Epoch [7/10], Loss: 1.9290
Epoch [8/10], Loss: 1.6549
Epoch [9/10], Loss: 1.3304
Epoch [10/10], Loss: 1.0071
Accuracy: 0.0970
```

图 26: 效果展示

2.4.7 循环神经网络（RNN）进行序列预测

1. 使用 RNN 来模拟序列预测任务

- 命令展示

```
import torch
```

```
import torch.nn as nn
import torch.optim as optim

# 生成随机数据
def generate_data(num_samples=1000, num_features=20, num_classes=2):
    X = torch.randn(num_samples, num_features) # 随机生成特征数据
    y = torch.randint(0, num_classes, (num_samples,)) # 随机生成0或1作为标签
    return X, y

# 定义简单的神经网络模型
class SimpleNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(SimpleNN, self).__init__()
        self.layer1 = nn.Linear(input_size, hidden_size)
        self.layer2 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu(self.layer1(x))
        x = self.layer2(x)
        return x

# 生成数据
X_train, y_train = generate_data()
X_test, y_test = generate_data()

# 转换为PyTorch张量
y_train = y_train.float().view(-1, 1)
y_test = y_test.float().view(-1, 1)

# 实例化模型，定义损失函数和优化器
input_size = X_train.shape[1]
hidden_size = 64
num_classes = 1 # 二分类任务
model = SimpleNN(input_size, hidden_size, num_classes)
criterion = nn.BCEWithLogitsLoss() # 使用带有sigmoid激活的二元交叉熵损失
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 训练模型
```

```
num_epochs = 100
for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()

import torch
import torch.nn as nn
import torch.optim as optim

# 定义简单的RNN模型
class SimpleRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleRNN, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), hidden_size)
        out, _ = self.rnn(x, h0)
        out = self.fc(out[:, -1, :])
        return out

# 生成随机序列数据
def generate_sequence_data(num_samples=1000, seq_length=10, input_size=1, num_classes=2):
    X = torch.randn(num_samples, seq_length, input_size)
    y = torch.randint(0, num_classes, (num_samples,))
    return X, y

# 生成数据
input_size = 1
hidden_size = 16
output_size = 2
X_train, y_train = generate_sequence_data(num_samples=2000)
X_test, y_test = generate_sequence_data(num_samples=100)

# 实例化模型，定义损失函数和优化器
model = SimpleRNN(input_size, hidden_size, output_size)
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 训练模型
num_epochs = 10
batch_size = 64
for epoch in range(num_epochs):
    model.train()
    for i in range(0, len(X_train), batch_size):
        inputs = X_train[i:i + batch_size]
        targets = y_train[i:i + batch_size]

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

```
# 测试模型
model.eval()
with torch.no_grad():
    outputs = model(X_test)
    _, predicted = torch.max(outputs, 1)
    accuracy = (predicted == y_test).sum().item() / y_test.size(0)
    print(f'Accuracy: {accuracy:.4f}')
```

- 效果展示


```
PS E:\作业\系统开发基础\python> & E:/python/python.exe e
:/作业/系统开发基础/python/work/work.py
Epoch [10/100], Loss: 0.6878
Epoch [20/100], Loss: 0.6793
Epoch [30/100], Loss: 0.6719
Epoch [40/100], Loss: 0.6650
Epoch [50/100], Loss: 0.6578
Epoch [60/100], Loss: 0.6500
Epoch [70/100], Loss: 0.6416
Epoch [80/100], Loss: 0.6324
Epoch [90/100], Loss: 0.6225
Epoch [100/100], Loss: 0.6124
Accuracy: 0.5120
```

图 27: 效果展示

2.4.8 自定义数据集和数据加载器

1. 使用 PyTorch 的 Dataset 和 DataLoader 来处理自定义数据。

- 命令展示

```
import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.optim as optim

# 自定义数据集
class MyDataset(Dataset):
    def __init__(self, num_samples=1000, num_features=10):
        self.data = torch.randn(num_samples, num_features)
        self.labels = (self.data.sum(axis=1) > 0).float()

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return self.data[idx], self.labels[idx]

# 定义简单的神经网络模型
```

```
class SimpleNN(nn.Module):
    def __init__(self, input_size):
        super(SimpleNN, self).__init__()
        self.layer1 = nn.Linear(input_size, 64)
        self.layer2 = nn.Linear(64, 1)

    def forward(self, x):
        x = torch.relu(self.layer1(x))
        x = torch.sigmoid(self.layer2(x))
        return x

# 实例化数据集和数据加载器
dataset = MyDataset()
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)

# 实例化模型，定义损失函数和优化器
input_size = dataset.data.shape[1]
model = SimpleNN(input_size)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 训练模型
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    for inputs, targets in dataloader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets.unsqueeze(1))
        loss.backward()
        optimizer.step()

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# 在训练数据上测试模型
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for inputs, targets in dataloader:
```

```
outputs = model(inputs)
predicted = (outputs > 0.5).float()
total += targets.size(0)
correct += (predicted.squeeze(1) == targets).sum().item()
accuracy = correct / total
print(f'Accuracy: {accuracy:.4f}')
```

- 效果展示



```
PS E:\作业\系统开发基础\python> & E:/python/python.exe e
:/作业/系统开发基础/python/work/work.py
Epoch [1/10], Loss: 0.5111
Epoch [2/10], Loss: 0.5262
Epoch [3/10], Loss: 0.3767
Epoch [4/10], Loss: 0.4405
Epoch [5/10], Loss: 0.3248
Epoch [6/10], Loss: 0.2073
Epoch [7/10], Loss: 0.1340
Epoch [8/10], Loss: 0.1469
Epoch [9/10], Loss: 0.1401
Epoch [10/10], Loss: 0.2374
Accuracy: 0.9850
```

图 28: 效果展示

3 困难与解决方案

3.1 调试及性能分析

- 问题: rr 的使用不成功, 报告一个致命错误, 指出它无法识别 Intel CPU 的微架构 (Intel CPU type 0xb06a0 unknown)
 - 解决方案: rr 不支持当前机器的 CPU 类型, 或者 rr 版本过旧, 未包括对新 CPU 的支持。从GitHub上下载最新的rr, 来更新 rr。
- 问题: 编译 rr 时遇到了与 32 位交叉编译相关的错误。
 - 解决方案: 安装 32 位库
- 问题: 下载pycallgraph库报错

- 解决方案: pycallgraph 似乎不再维护, 并且与较新的 Python 版本不兼容。由于该库依赖于较旧的工具和配置选项 (例如 use2to3), 这些选项在当前的 Python 和 setuptools 环境中不再支持, 因此安装失败。所以换成其他的。
4.
 - 问题: 在使用journalctl命令时, 未能成功获取到超级用户的日志信息。
 - 解决方案: 使用sudo来提升权限, 并正确指定用户ID (UID) 过滤参数。
 5.
 - 问题: 在使用htop命令时, 未能成功安装或启动。
 - 解决方案: 确保系统已更新, 并且正确安装了htop包, 使用sudo apt-get install htop进行安装。

3.2 元编程演示

1.
 - 问题: make无法找到生成paper.pdf所需的paper.tex文件
 - 解决方案: 确保所有在Makefile中列出的依赖文件都存在于当前目录中, 同时检查和创建必要文件。
2.
 - 问题: 在使用装饰器进行方法计时时, 遇到了计时不准确的问题。
 - 解决方案: 检查并确保在装饰器中正确地计算了时间差, 并且没有其他代码干扰了计时。
3.
 - 问题: 在使用元类创建只读属性时, 未能正确限制属性的修改。
 - 解决方案: 确保在元类中正确地定义了属性的getter和setter, 并且setter抛出了适当的异常。

3.3 PyTorch编程

1.
 - 问题: 在自定义数据集时, 遇到了数据加载不正确的问题。
 - 解决方案: 检查数据集类的getitem方法, 确保它正确地返回了数据和标签。
2.
 - 问题: 在训练神经网络分类器时, 模型的准确率未达到预期。
 - 解决方案: 调整模型结构和参数, 使用交叉验证来优化模型性能, 并确保数据预处理正确。
3.
 - 问题: 在实现卷积神经网络 (CNN) 时, 遇到了模型不收敛的问题。
 - 解决方案: 调整学习率和优化器参数, 增加数据增强, 以及使用合适的初始化方法。

4 心得体会

- 通过本次的学习, 我对调试工具和性能分析有了更深入的认识, 学会了在Linux上使用journalctl来获取系统日志, 使用stress和htop工具来模拟和监控系统负载。此外, 我也掌握了如何使用linter插件来提高代码质量, 以及如何通过netstat和lsof命令来查找并处理端口占用问题。
- 在元编程方面, 我学习了如何使用装饰器和元类来增强代码的灵活性和可维护性, 这些技术让我能够编写更加动态和强大的Python程序。

- 通过PyTorch编程的学习，我对深度学习有了更实际的理解，特别是在图像处理和分类任务中。我学会了如何构建和训练神经网络，以及如何使用PyTorch提供的各种工具和类来处理数据和构建模型。
- Markdown是非常有用的工具，通过本次学习我也学会了用Markdown绘制表格等功能，深刻感受到了Markdown的魅力。
- 我期待将这些新学到的知识应用到未来的项目中，以解决更复杂的问题

5 github网址

GitHub仓库 <https://github.com/KeepingMoving/work1.git>