

1-多任务编程

1-1 多任务与进程

同一时间内执行多个任务。

执行方式：

- 1、并发：一段时间内交替执行任务。
- 2、并行：多个内核一起执行软件。

进程的概念：操作系统进行资源分配的基本单位。

1-2 多进程的使用

Process进程类的说明

Process([group [, target [, name [, args [, kwargs]]]])

- group：指定进程组，目前只能使用None
- target：执行的目标任务名
- name：进程名字
- args：以元组方式给执行任务传参
- kwargs：以字典方式给执行任务传参

Process创建的实例对象的常用方法：

- start()：启动子进程实例（创建子进程）
- join()：等待子进程执行结束
- terminate()：不管任务是否完成，立即终止子进程

Process创建的实例对象的常用属性：

name：当前进程的别名，默认为Process-N，N为从1开始递增的整数

```
1 import multiprocessing
2 import time
3 def dance():
4     for i in range(5):
5         print("跳舞...")
6         time.sleep(0.2)
7 def sing():
8     for i in range(5):
9         print("唱歌...")
10        time.sleep(0.2)
11
12 if __name__ == '__main__':
13     dance_process = multiprocessing.Process(target=dance,name="my_process_dance")
14     sing_process = multiprocessing.Process(target=sing)
15     dance_process.start()
16     sing_process.start()
```

1-3 获取进程编号

1、获取当前进程编号

```
1 import multiprocessing
2 import time
3 import os
4
5 def dance():
6     print("dance:", os.getpid())
```

```

7     print("dance:", multiprocessing.current_process())
8     for i in range(5):
9         print("跳舞中...")
10        time.sleep(0.2)
11        os.kill(os.getpid(), 9)
12
13 def sing():
14     print("sing:", os.getpid())
15     print("sing:", multiprocessing.current_process())
16     for i in range(5):
17         print("唱歌中...")
18         time.sleep(0.2)
19
20
21 if __name__ == '__main__':
22     print("main:", os.getpid())
23     print("main:", multiprocessing.current_process())
24     dance_process = multiprocessing.Process(target=dance)
25     sing_process = multiprocessing.Process(target=sing)
26
27     dance_process.start()
28     sing_process.start()

```

结果:

```

main: 6346
main: <_MainProcess name='MainProcess' parent=None started>
dance: 6348
dance: <Process name='Process-1' parent=6346 started>
跳舞中...
sing: 6349
sing: <Process name='Process-2' parent=6346 started>
唱歌中...
唱歌中...
唱歌中...
唱歌中...
唱歌中...

```

2、获取当前父进程编号

```

1 import multiprocessing
2 import time
3 import os
4
5 def dance():
6     print("dance:", os.getpid())
7     print("dance:", multiprocessing.current_process())
8     print("dance的父进程编号:", os.getppid())
9     for i in range(5):
10        print("跳舞中...")
11        time.sleep(0.2)
12        if(i==1):
13            os.kill(os.getpid(), 9)
14
15 def sing():
16     print("sing:", os.getpid())
17     print("sing:", multiprocessing.current_process())

```

```

18     print("sing的父进程编号:", os.getppid())
19     for i in range(5):
20         print("唱歌中...")
21         time.sleep(0.2)
22
23
24 if __name__ == '__main__':
25     print("main:", os.getpid())
26     print("main:", multiprocessing.current_process())
27     dance_process = multiprocessing.Process(target=dance, name="myprocess1")
28     sing_process = multiprocessing.Process(target=sing)
29
30     dance_process.start()
31     sing_process.start()
32
33 #result:
34 # main: 830
35 # main: <MainProcess name='MainProcess' parent=None started>
36 # dance: 832
37 # dance: <Process name='myprocess1' parent=830 started>
38 # dance的父进程编号: 830
39 # 跳舞中...
40 # sing: 833
41 # sing: <Process name='Process-2' parent=830 started>
42 # sing的父进程编号: 830
43 # 唱歌中...
44 # 跳舞中...
45 # 唱歌中...
46 # 唱歌中...
47 # 唱歌中...
48 # 唱歌中...

```

1-4 进程执行带有参数的任务介绍

1.args参数使用

```

1 import multiprocessing
2 import time
3
4 def task(count):
5     for i in range(count):
6         print("process is going...")
7         time.sleep(0.2)
8     else:
9         print('process is finished.')
10
11 if __name__ == '__main__':
12     sub_process = multiprocessing.Process(target=task,args=(4,))
13     sub_process.start()
14
15 # result:
16 #     process is going...
17 #     process is going...
18 #     process is going...
19 #     process is going...

```

```
20 # process is finished.
```

2.kwargs参数使用

```
1 import multiprocessing
2 import time
3
4 def task(count):
5     for i in range(count):
6         print("process is going...")
7         time.sleep(0.2)
8     else:
9         print('process is finished.')
10
11 if __name__ == '__main__':
12     sub_process = multiprocessing.Process(target=task, kwargs={"count":3})
13     sub_process.start()
14
15 # result:
16 # process is going...
17 # process is going...
18 # process is going...
19 # process is finished.
```

1-5 进程的注意点

1.进程之间不共享全局变量

```
1 import multiprocessing
2 import time
3
4 g_list = list()
5
6 def add_data():
7     for i in range(5):
8         g_list.append(i)
9         print('add', i)
10        time.sleep(0.2)
11    print("add_data:", g_list)
12
13 def read_data():
14    print("read_data:", g_list)
15
16 if __name__ == '__main__':
17     add_data_process = multiprocessing.Process(target=add_data)
18     read_data_process = multiprocessing.Process(target=read_data)
19     add_data_process.start()
20     add_data_process.join()
21     read_data_process.start()
22     print("main:", g_list)
23
24 #result:
25 # add 0
26 # add 1
27 # add 2
28 # add 3
```

```

29     # add 4
30     # add_data: [0, 1, 2, 3, 4]
31     # main: []
32     # read_data: []

```

解释:

进程操作的都是自己进程里的全局变量，但进程之间不共享全局变量。

创建子进程会对主进程资源进行拷贝，也就是说子进程是主进程的一个副本，好比是一对双胞胎，之所以进程之间不共享全局变量。

2.主进程会等待所有子进程执行结束再结束

```

1  import multiprocessing
2  import time
3
4
5  def child_process():
6      for i in range(5):
7          print("process is going...")
8          time.sleep(0.2)
9
10
11 if __name__ == '__main__':
12     child_sub_process = multiprocessing.Process(target=child_process)
13     child_sub_process.start()
14     time.sleep(0.5)
15     print('parent process is end')
16     exit()
17
18 #result:
19     # process is going...
20     # process is going...
21     # process is going...
22     # parent process is end
23     # process is going...
24     # process is going...

```

解决方案:

- 1、守护主进程：主进程退出后子进程销毁不再执行
- 2、子进程销毁：子进程执行结束

```

1  import multiprocessing
2  import time
3
4
5  def child_process():
6      for i in range(5):
7          print("process is going...")
8          time.sleep(0.2)
9
10
11 if __name__ == '__main__':
12     child_sub_process = multiprocessing.Process(target=child_process)
13     child_sub_process.start()
14     time.sleep(0.5)
15     print('parent process is end')
16     child_sub_process.terminate()
17     #terminate(): 不管任务是否完成，立即终止子进程

```

```
18     exit()
19
20 #result:
21     # process is going...
22     # process is going...
23     # process is going...
24     # parent process is end
```

1-6 线程

线程的概念：线程是进程中执行代码的一个分支，每个执行分支（线程）要想工作执行代码需要cpu进行调度，也就是说线程是cpu调度的基本单位，每个进程至少都有一个线程，而这个线程就是我们通常说的主线程。

1.线程类Thread参数说明

Thread([group [, target [, name [, args [, kwargs]]]])

- group: 线程组，目前只能使用None
- target: 执行的目标任务名
- args: 以元组的方式给执行任务传参
- kwargs: 以字典方式给执行任务传参
- name: 线程名，一般不用设置

2.多线程的使用

```
1  import threading
2  import time
3
4  def sing():
5      for i in range(3):
6          print("正在唱歌...%d" % i)
7          time.sleep(0.5)
8
9  def dance():
10     for i in range(3):
11         print("正在跳舞...%d" % i)
12         time.sleep(1)
13
14
15 if __name__ == '__main__':
16     sing_thread = threading.Thread(target=sing)
17     dance_thread = threading.Thread(target=dance)
18     sing_thread.start()
19     dance_thread.start()
20
21 #result:
22     # 正在唱歌...0正在跳舞...0
23     #
24     # 正在唱歌...1
25     # 正在跳舞...1
26     # 正在唱歌...2
27     # 正在跳舞...2
```

3.带参数多线程

```
1  import threading
2  import time
3
4  def sing(count):
```

```

5     for i in range(count):
6         print("正在唱歌...%d"%i)
7         time.sleep(0.5)
8
9     if __name__ == '__main__':
10        #sing_thread = threading.Thread(target=sing,args=(3,))
11        sing_thread = threading.Thread(target=sing, kwargs={"count":3})
12        sing_thread.start()
13
14    #result:
15        # 正在唱歌...0
16        # 正在唱歌...1
17        # 正在唱歌...2

```

4.线程特性1:线程执行无序

```

1  import threading
2  import time
3  def task():
4      time.sleep(0.5)
5      print("当前线程:", threading.current_thread().name)
6  if __name__ == '__main__':
7      for _ in range(5):
8          sub_thread = threading.Thread(target=task)
9          sub_thread.start()
10
11    #result:
12        # 当前线程: Thread-1
13        # 当前线程: Thread-3
14        # 当前线程: 当前线程: Thread-2
15        # Thread-5
16        # 当前线程: Thread-4

```

5.线性特征2:主线程等待所有子线程结束后再结束

守护主线程方式:

```

1  import threading
2  import time
3
4  def show_info():
5      for i in range(5):
6          print("test:", i)
7          time.sleep(0.5)
8
9
10 if __name__ == '__main__':
11     sub_thread = threading.Thread(target=show_info, daemon=True)
12     # sub_thread.setDaemon(True)
13     sub_thread.start()
14     time.sleep(1)
15     print("over")
16
17    #result:
18        # test: 0
19        # test: 1
20        # over

```

6.线程特性3:线程之间共享全局变量

```
1 import threading
2 import time
3
4 my_list = list()
5
6 def write_data():
7     for i in range(5):
8         my_list.append(i)
9         print("write_data:", my_list)
10        time.sleep(0.1)
11
12
13 def read_data():
14     for i in range(5):
15         print("read_data:", my_list)
16         time.sleep(0.1)
17
18
19 if __name__ == '__main__':
20     write_thread = threading.Thread(target=write_data)
21     read_thread = threading.Thread(target=read_data)
22     write_thread.start()
23     time.sleep(0.2)
24     read_thread.start()
25
26 #result:
27     # write_data: [0]
28     # write_data: [0, 1]
29     # read_data: [0, 1]
30     # write_data: [0, 1, 2]
31     # read_data: [0, 1, 2]
32     # write_data: [0, 1, 2, 3]
33     # read_data:write_data: [0, 1, 2, 3, 4]
34     # [0, 1, 2, 3, 4]
35     # read_data: [0, 1, 2, 3, 4]
36     # read_data: [0, 1, 2, 3, 4]
```

7.线程特性4:局部变量互斥

```
1 import threading
2
3 g_num = 0
4
5 def sum_num1():
6     for i in range(1000000):
7         global g_num
8         g_num += 1
9
10    print("sum1:", g_num)
11
12 def sum_num2():
13     for i in range(1000000):
14         global g_num
15         g_num += 1
```



```

16     print("sum2:", g_num)
17
18 if __name__ == '__main__':
19     first_thread = threading.Thread(target=sum_num1)
20     second_thread = threading.Thread(target=sum_num2)
21     first_thread.start()
22     #进程等待
23     first_thread.join()
24     second_thread.start()

```

8.互斥锁

```

1  import threading
2  import time
3
4  g_num = 0
5
6  lock = threading.Lock()
7
8  def sum_num1():
9      for i in range(1000000):
10         lock.acquire()
11         global g_num
12         g_num += 1
13         lock.release()
14
15  def sum_num2():
16      for i in range(1000000):
17         lock.acquire()
18         global g_num
19         g_num += 1
20         lock.release()
21
22  if __name__ == '__main__':
23     first_thread = threading.Thread(target=sum_num1)
24     second_thread = threading.Thread(target=sum_num2)
25     first_thread.start()
26     second_thread.start()
27     for i in range(35):
28         print(g_num,end=' ')
29         if (i%10==0)&(i!=0):
30             print()
31         time.sleep(0.01)
32
33  # result:
34     # 20343 88397 153357 248728 302287 356780 437757 494705 544366 652081 705001
35     # 755286 835664 943429 996975 1051972 1100072 1154027 1207119 1288185 1366398
36     # 1430254 1498314 1547544 1599400 1653668 1735005 1817667 1885925 1965791 2000000
37     # 2000000 2000000 2000000 2000000

```

9.死锁

```

1  import threading
2  import time
3
4  lock = threading.Lock()
5

```

```
6 def get_value(index):
7
8     lock.acquire()
9     print(threading.current_thread())
10    my_list = [3,6,8,1]
11    if index >= len(my_list):
12        print("下标越界:", index)
13        #在合适的地方释放锁
14        lock.release()
15        return
16    value = my_list[index]
17    print(value)
18    time.sleep(0.2)
19    lock.release()
20
21
22 if __name__ == '__main__':
23     for i in range(30):
24         sub_thread = threading.Thread(target=get_value, args=(i,))
25         sub_thread.start()
```