# 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

```
import multiprocessing
import time
def dance():
    for i in range(5):
        print("跳舞...")
        time.sleep(0.2)
def sing():
    for i in range(5):
        print("唱歌...")
         time.sleep(0.2)

if __name__ == '__main__':
    dance_process = multiprocessing.Process(target=dance,name="my_process_dance")
    sing_process = multiprocessing.Process(target=sing)
    dance_process.start()
    sing_process.start()
```

## 1-3 获取进程编号

### 1、获取当前进程编号

```
import multiprocessing
import time
import os

def dance():
    print("dance:", os.getpid())
```

```
    print("dance:", multiprocessing.current_process())
    for i in range(5):
        print("跳舞中...")
        time.sleep(0.2)
        os.kill(os.getpid(), 9)

def sing():
    print("sing:", os.getpid())
    print("sing:", multiprocessing.current_process())
    for i in range(5):
        print("唱歌中...")
        time.sleep(0.2)


if __name__ == '__main__':
    print("main:", os.getpid())
    print("main:", multiprocessing.current_process())
    dance_process = multiprocessing.Process(target=dance)
    sing_process = multiprocessing.Process(target=sing)

    dance_process.start()
    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

```
import multiprocessing
import time
import os

def dance():
    print("dance:", os.getpid())
    print("dance:", multiprocessing.current_process())
    print("dance的父进程编号:", os.getppid())
    for i in range(5):
        print("跳舞中...")
        time.sleep(0.2)
        if(i==1):
            os.kill(os.getpid(), 9)

def sing():
    print("sing:", os.getpid())
    print("sing:", multiprocessing.current_process())
```

```
    print("sing的父进程编号:", os.getppid())
    for i in range(5):
        print("唱歌中...")
        time.sleep(0.2)


if __name__ == '__main__':
    print("main:", os.getpid())
    print("main:", multiprocessing.current_process())
    dance_process = multiprocessing.Process(target=dance, name="myprocess1")
    sing_process = multiprocessing.Process(target=sing)

    dance_process.start()
    sing_process.start()

#result:
# main: 830
# main: <_MainProcess name='MainProcess' parent=None started>
# dance: 832
# dance: <Process name='myprocess1' parent=830 started>
# dance的父进程编号: 830
# 跳舞中...
# sing: 833
# sing: <Process name='Process-2' parent=830 started>
# sing的父进程编号: 830
# 唱歌中...
# 跳舞中...
# 唱歌中...
# 唱歌中...
# 唱歌中...
# 唱歌中...
```

# 1-4 进程执行带有参数的任务介绍

## 1.args参数使用

```
import multiprocessing
import time

def task(count):
    for i in range(count):
        print("process is going...")
        time.sleep(0.2)
    else:
        print('process is finished.')

if __name__ == '__main__':
    sub_process = multiprocessing.Process(target=task,args=(4,))
    sub_process.start()

# result:
#     process is going...
#     process is going...
#     process is going...
#     process is going...
```

```
#     process is finished.
```

### 2.kwargs参数使用

```
import multiprocessing
import time

def task(count):
    for i in range(count):
        print("process is going...")
        time.sleep(0.2)
    else:
        print('process is finished.')

if __name__ == '__main__':
    sub_process = multiprocessing.Process(target=task,kwargs={"count":3})
    sub_process.start()

# result:
#     process is going...
#     process is going...
#     process is going...
#     process is finished.
```

## 1-5 进程的注意点

### 1.进程之间不共享全局变量

```
import multiprocessing
import time

g_list = list()

def add_data():
    for i in range(5):
        g_list.append(i)
        print('add',i)
        time.sleep(0.2)
    print("add_data:",g_list)

def read_data():
    print("read_data:",g_list)

if __name__ == '__main__':
    add_data_process = multiprocessing.Process(target=add_data)
    read_data_process = multiprocessing.Process(target=read_data)
    add_data_process.start()
    add_data_process.join()
    read_data_process.start()
    print("main:",g_list)

#result:
    # add 0
    # add 1
    # add 2
    # add 3
```

```
    # add 4
    # add_data: [0, 1, 2, 3, 4]
    # main: []
    # 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

```python
import multiprocessing
import time


def child_process():
    for i in range(5):
        print("process is going...")
        time.sleep(0.2)


if __name__ == '__main__':
    child_sub_process = multiprocessing.Process(target=child_process)
    child_sub_process.start()
    time.sleep(0.5)
    print('parent process is end')
    exit()

#result:
    # process is going...
    # process is going...
    # process is going...
    # parent process is end
    # process is going...
    # 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

```python
import multiprocessing
import time


def child_process():
    for i in range(5):
        print("process is going...")
        time.sleep(0.2)


if __name__ == '__main__':
    child_sub_process = multiprocessing.Process(target=child_process)
    child_sub_process.start()
    time.sleep(0.5)
    print('parent process is end')
    child_sub_process.terminate()
    #terminate(): 不管任务是否完成，立即终止子进程
```

```
    exit()

#result:
    # process is going...
    # process is going...
    # process is going...
    # 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

```python
import threading
import time

def sing():
    for i in range(3):
        print("正在唱歌...%d" % i)
        time.sleep(0.5)

def dance():
    for i in range(3):
        print("正在跳舞...%d" % i)
        time.sleep(1)


if __name__ == '__main__':
    sing_thread = threading.Thread(target=sing)
    dance_thread = threading.Thread(target=dance)
    sing_thread.start()
    dance_thread.start()

#result:
    # 正在唱歌...0正在跳舞...0
    #
    # 正在唱歌...1
    # 正在跳舞...1
    # 正在唱歌...2
    # 正在跳舞...2
```

## 3.带参数多线程

```python
import threading
import time

def sing(count):
```

```
    for i in range(count):
        print("正在唱歌...%d"%i)
        time.sleep(0.5)

if __name__ == '__main__':
    #sing_thread = threading.Thread(target=sing,args=(3,))
    sing_thread = threading.Thread(target=sing, kwargs={"count":3})
    sing_thread.start()

#result:
    # 正在唱歌...0
    # 正在唱歌...1
    # 正在唱歌...2
```

### 4.线程特性1:线程执行无序

```
import threading
import time
def task():
    time.sleep(0.5)
    print("当前线程:", threading.current_thread().name)
if __name__ == '__main__':
    for _ in range(5):
        sub_thread = threading.Thread(target=task)
        sub_thread.start()

#result:
    # 当前线程: Thread-1
    # 当前线程: Thread-3
    # 当前线程: 当前线程: Thread-2
    # Thread-5
    # 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:

```
import threading
import time

def show_info():
    for i in range(5):
        print("test:", i)
        time.sleep(0.5)


if __name__ == '__main__':
    sub_thread = threading.Thread(target=show_info, daemon=True)
    # sub_thread.setDaemon(True)
    sub_thread.start()
    time.sleep(1)
    print("over")

#result:
    # test: 0
    # test: 1
    # over
```

### 6.线程特性3:线程之间共享全局变量

```python
import threading
import time

my_list = list()

def write_data():
    for i in range(5):
        my_list.append(i)
        print("write_data:", my_list)
         time.sleep(0.1)


def read_data():
    for i in range(5):
        print("read_data:", my_list)
        time.sleep(0.1)


if __name__ == '__main__':
    write_thread = threading.Thread(target=write_data)
    read_thread = threading.Thread(target=read_data)
    write_thread.start()
    time.sleep(0.2)
    read_thread.start()

#result:
    # write_data: [0]
    # write_data: [0, 1]
    # read_data: [0, 1]
    # write_data: [0, 1, 2]
    # read_data: [0, 1, 2]
    # write_data: [0, 1, 2, 3]
    # read_data:write_data: [0, 1, 2, 3, 4]
    #  [0, 1, 2, 3, 4]
    # read_data: [0, 1, 2, 3, 4]
    # read_data: [0, 1, 2, 3, 4]
```

### 7.线程特性4:局部变量互斥

```python
import threading

g_num = 0

def sum_num1():
    for i in range(1000000):
        global g_num
        g_num += 1

    print("sum1:", g_num)

def sum_num2():
    for i in range(1000000):
        global g_num
        g_num += 1
```

```
    print("sum2:", g_num)

if __name__ == '__main__':
    first_thread = threading.Thread(target=sum_num1)
    second_thread = threading.Thread(target=sum_num2)
    first_thread.start()
    #进程等待
    first_thread.join()
    second_thread.start()
```

## 8.互斥锁

```
import threading
import time

g_num = 0

lock = threading.Lock()

def sum_num1():
    for i in range(1000000):
        lock.acquire()
        global g_num
        g_num += 1
        lock.release()

def sum_num2():
    for i in range(1000000):
        lock.acquire()
        global g_num
        g_num += 1
        lock.release()

if __name__ == '__main__':
    first_thread = threading.Thread(target=sum_num1)
    second_thread = threading.Thread(target=sum_num2)
    first_thread.start()
    second_thread.start()
    for i in range(35):
        print(g_num,end=' ')
        if (i%10==0)&(i!=0):
            print()
        time.sleep(0.01)

# result:
    # 20343 88397 153357 248728 302287 356780 437757 494705 544366 652081 705001
    # 755286 835664 943429 996975 1051972 1100072 1154027 1207119 1288185 1366398
    # 1430254 1498314 1547544 1599400 1653668 1735005 1817667 1885925 1965791 2000000
    # 2000000 2000000 2000000 2000000
```

## 9.死锁

```
import threading
import time

lock = threading.Lock()

```

```
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()
```