

# python进程



黑马程序员  
[www.itheima.com](http://www.itheima.com)

传智教育旗下  
高端IT教育品牌



# 目录

## Contents

### ◆ 多任务的介绍

多任务概念、作用、并发和并行

### ◆ 进程的介绍

概念、作用、多进程基本工作方式

### ◆ 多进程完成多任务

创建多进程、传参

### ◆ 获取进程编号

获取进程编号、父进程编号

### ◆ 进程的注意点

不共享、主进程等待子进程结束



# 目录

Contents



## ◆ 多任务的介绍

多任务概念、作用、并发和并行

- ◆ 进程的介绍
- ◆ 多进程完成多任务
- ◆ 获取进程编号
- ◆ 进程的注意点



# 学习目标

Learning Objectives

1. 理解多任务的作用
2. 知道多任务是什么
3. 理解多任务的表现形式

思考

网盘下载资料时为什么要多个文件同时下载？

当前进度  已完成10% 13.79 M/S 已全部加载，共3个

<input type="checkbox"/> 文件名	大小	状态	操作
<input type="checkbox"/>  03.HOG特征.mp4	4.69 M/54.98 M	 3.74 MB/S - 剩余时间: 00:00:13	 
<input type="checkbox"/>  02.霍夫变换.mp4	6.75 M/37.77 M	 5.38 MB/S - 剩余时间: 00:00:05	 
<input type="checkbox"/>  01.canny边缘检测.mp4	5.92 M/66.92 M	 4.67 MB/S - 剩余时间: 00:00:13	 

## 多任务的优势

多个任务同时执行能够充分**利用CPU资源**，大大提高程序**执行效率**

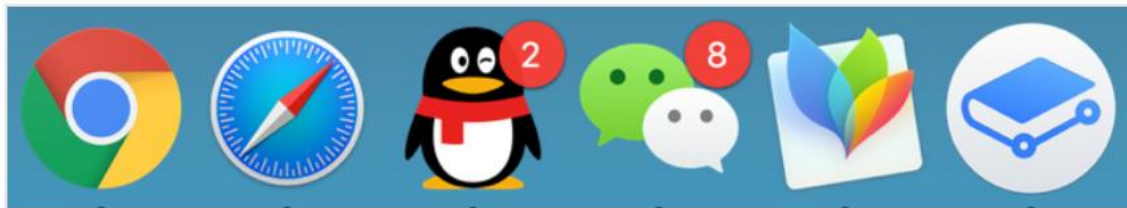
**思考一下：**利用现学知识能够让多个任务同时执行吗？

不能，因为之前所写的程序都是**单任务的**，也就是说一个函数或者方法执行完成，另外一个函数或者方法才能执行。要想实现多个任务同时执行就需要使用**多任务**。

## 多任务的概念

- 概念：多任务是指在**同一时间**内执行**多个任务**（给我们的感觉）。

例如：现在电脑安装的操作系统都是多任务操作系统，可以同时运行着多个软件。



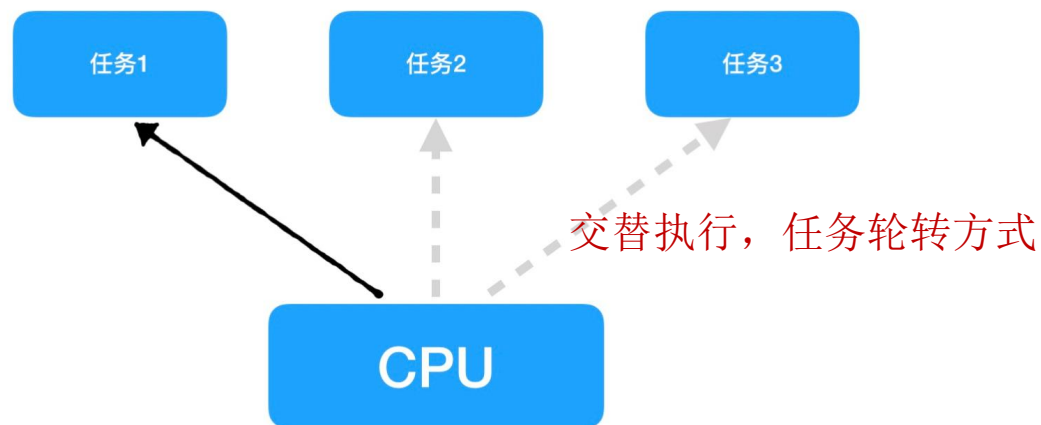
- 多任务的两种表现形式
  - 并发：在一段时间内，**交替**执行任务
  - 并行：在一段时间内，真正的**同时一起**执行多个任务

## 并发

在一段时间内交替去执行多个任务

例子：

对于单核cpu处理多任务, 操作系统轮流让各个任务交替执行, 假如: 软件1执行0.01秒, 切换到软件2, 软件2执行0.01秒, 再切换到软件3, 执行0.01秒……这样反复执行下去, 实际上每个软件都是交替执行的。但是, 由于CPU的执行速度实在是太快了, 表面上我们感觉就像这些软件都在同时执行一样。这里需要注意单核cpu是并发的执行多任务的。



并发: 任务数量大于CPU的核心数

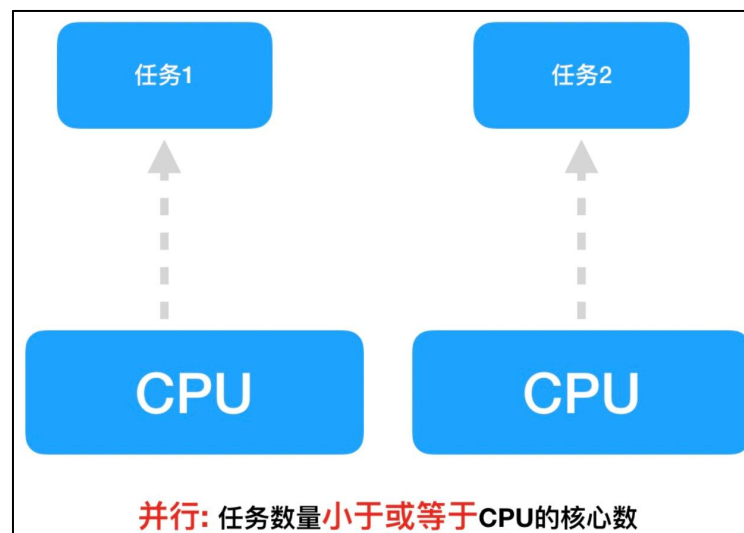


## 并行

在一段时间内真正的**同时**一起执行多个任务

例子：

对于多核cpu处理多任务，操作系统会给cpu的每个内核安排一个执行的任务，多个内核是真正的一起同时执行多个任务。这里需要注意多核cpu是并行的执行多任务，始终有多个任务一起执行。



## 1. 什么是多任务执行？

多任务是指在**同一时间**内执行**多个任务**（给我们的感觉）



## 总结

## 2. 多任务执行的优势是什么？

多任务执行能够**充分利用cpu资源**，**提高程序执行效率**

## 3. 多任务执行有哪两种表现形式？

**并发**: 在一段时间内**交替**去执行多个任务。

**并行**: 在一段时间内**真正的同时一起**执行多个任务



# 目录

Contents



◆ 多任务的介绍

◆ 进程的介绍

概念、作用、多进程工作方式

◆ 多进程完成多任务

◆ 获取进程编号

◆ 进程的注意点



# 学习目标

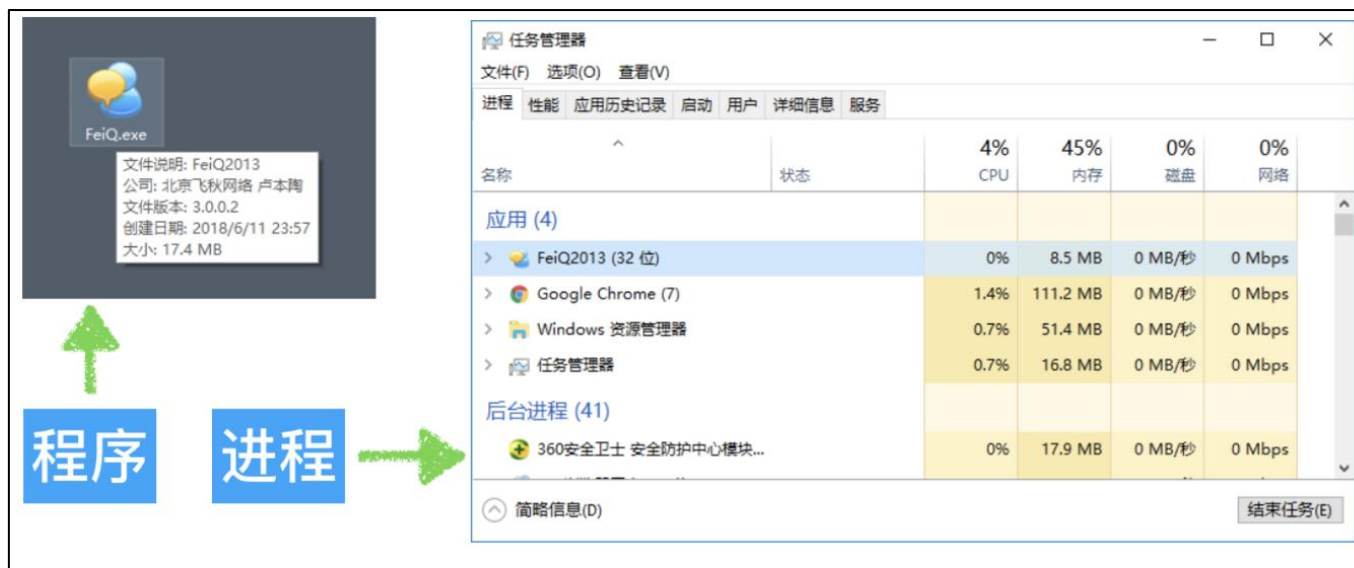
Learning Objectives

1. 理解进程的概念
2. 能够说出进程的作用

## 进程的概念

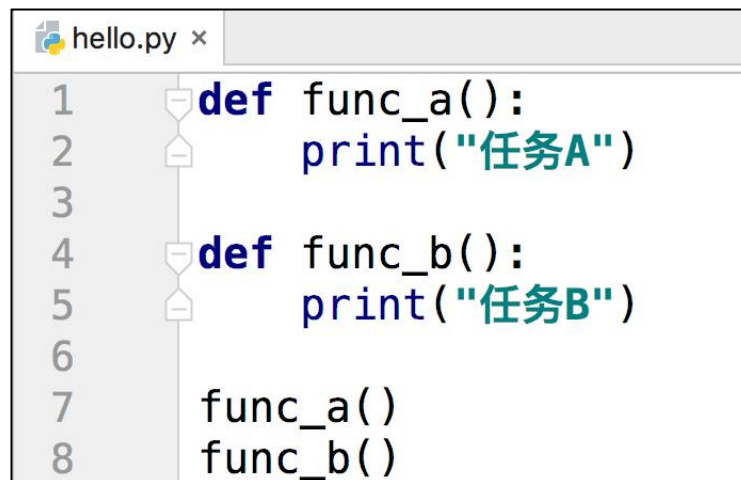
进程 (Process) 是CPU资源分配的最小单位，它是操作系统进行资源分配和调度运行的基本单位

通俗理解：一个正在运行的程序就是一个进程。例如:正在运行的qq，微信等他们都是一个进程



注意：一个程序运行后至少有一个进程

## 多进程的作用

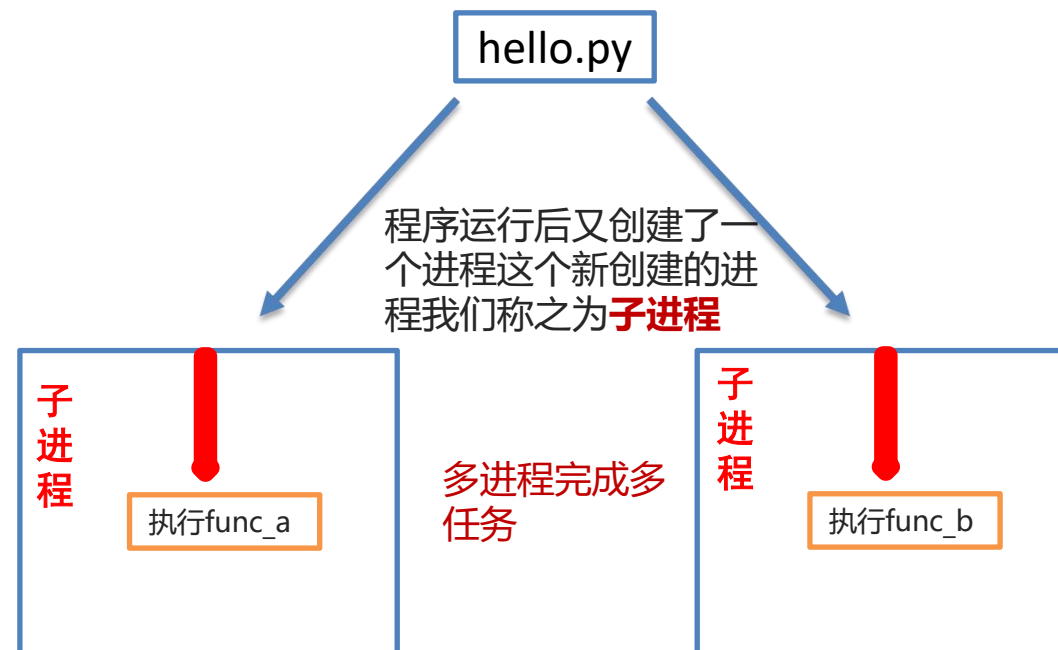
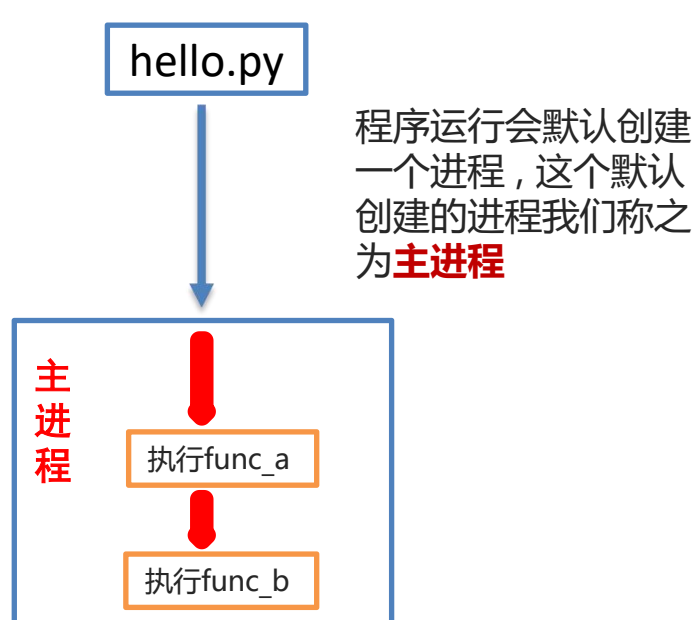


```
hello.py x
1  def func_a():
2      print("任务A")
3
4  def func_b():
5      print("任务B")
6
7  func_a()
8  func_b()
```

### 思考：

图中是一个非常简单的程序，一旦运行hello.py这个程序，按照代码的执行顺序，func\_a函数执行完完毕后才能执行func\_b函数。如果可以让func\_a和func\_b同时运行，显然执行hello.py这个程序的效率会大大提升。

## 多进程基本工作方式





# 总结

## 1. 进程是什么？

进程（Process）是操作系统**CPU资源分配**的最小单位

## 2. 多进程的作用是什么？

**多进程**是Python程序中**实现多任务**的一种方式，

使用**多进程**可以大大提高程序的执行效率。

## 3. Python中多进程的基本工作方式？

**程序运行起来**形成主进程；在**主进程上创建子进程**





# 目录

Contents



◆ 多任务的介绍

◆ 多进程的介绍

◆ 多进程完成多任务

创建多进程、传参

◆ 获取进程编号

◆ 进程的注意点



# 学习目标

Learning Objectives

1. 知道使用多进程完成多任务步骤
2. 能够编程实现多进程完成多任务
3. 知道进程任务的传参方式并实现

## 进程的创建步骤

1. 导入进程工具包

```
import multiprocessing
```

2. 通过进程类 实例化进程 对象

```
子进程对象 = multiprocessing.Process()
```

3. 启动进程执行任务

```
进程对象.start()
```

## 通过进程类实例化创建进程对象

子进程对象 =

```
multiprocessing.Process(group=None, target=None, name=None, args=(), kwargs={})
```

- group—参数未使用，值始终为None
- target—表示调用对象，即子进程要执行的任务（回调函数入口地址）
- args—表示以元组的形式向子任务函数传参，元组方式传参一定要和参数的顺序保持一致
- kwargs—表示以字典的方式给子任务函数传参，字典方式传参字典中的key要和参数名保持一致
- name—为子进程的名称

## 进程创建与启动的代码

### 案例分析

使用多进程来模拟一边编写代码，一边听音乐功能实现。

### 任务函数没有参数

# 1. 导入进程模块

```
import multiprocessing
```

```
import time
```

# 编写代码任务

```
def coding():
```

```
    for i in range(3):
```

```
        print("coding...")
```

```
        time.sleep(0.2)
```

# 听音乐任务

```
def music():
```

```
    for i in range(3):
```

```
        print("music...")
```

```
        time.sleep(0.2)
```

```
if __name__ == '__main__':
```

# 2. 通过进程类创建进程对象

```
coding_process = multiprocessing.Process(target=coding)
```

# 2. 通过进程类创建进程对象

```
music_process = multiprocessing.Process(target=music)
```

# 3. 启动进程

```
coding_process.start()
```

```
music_process.start()
```

## 进程带参数的任务

### 案例分析

使用多进程来模拟小明一边编写num行代码，一边听count首音乐功能实现。

### 任务函数有参数

# 1. 导入进程模块

```
import multiprocessing
```

```
import time
```

# 编写代码

```
def coding(num, name):
```

```
    for i in range(num):
```

```
        print(f"{name}在编写第{i}行代码...")
```

```
        time.sleep(0.2)
```

# 听音乐

```
def music(count, name):
```

```
    for i in range(count):
```

```
        print(f"{name}在听第{count}首音乐...")
```

```
        time.sleep(0.2)
```

```
if __name__ == '__main__':
```

# 2. 通过进程类创建进程对象

```
coding_process = multiprocessing.Process(target=coding, args=(3, "小明"))
```

# 2. 通过进程类创建进程对象

```
music_process = multiprocessing.Process(target=music, kwargs={"count":  
2, "name": "小明"})
```

# 3. 启动进程

```
coding_process.start()
```

```
music_process.start()
```

# 总结

## 1. 说出使用多进程完成多任务步骤？

### a. 导入进程包

```
import multiprocessing
```

### b. 创建子进程并指定执行的任务

```
sub_process = multiprocessing.Process (target=任务名)
```

### c. 启动进程执行任务

```
sub_process.start()
```

## 2. 进程传参的两种方式是什么？

a. 元组方式传参：元组方式传参一定要和任务函数的参数顺序保持一致。

b. 字典方式传参：字典方式传参字典中的key一定要和任务函数的参数保持一致



# 目录

Contents

- ◆ 多任务的介绍
- ◆ 多进程的介绍
- ◆ 多进程完成多任务



- ◆ 获取进程编号

获取进程编号、父进程编号

- ◆ 进程的注意点





# 学习目标

Learning Objectives

1. 知道进程编号的作用
2. 能够获取进程编号

## 进程编号的作用

进程编号**唯一标识一个进程**，方便管理进程。

在一个操作系统中，一个进程拥有的进程号**是唯一的**，进程号可以反复使用。

获取进程编号的目的是验证**主进程和子进程的关系**，可以得知**子进程是由那个主进程**创建出来的

获取进程编号的两种操作

- 获取当前进程编号
- 获取当前父进程编号

## 获取进程编号

- `os.getpid()` 的使用

```
import os
# 获取当前进程编号
pid = os.getpid()
print(pid)
或者
import multiprocessing
pid = multiprocessing.current_process().pid
print(pid)
```

- `os.getppid()` 的使用

```
# 获取父进程的编号
ppid = os.getppid()
print(ppid)
```



# 总结

## 1. 进程编号的作用？

区分不同的进程，便于进行进程管理

## 2. 获取当前进程编号？

`os.getpid()` 或 `multiprocessing.current_process().pid`

## 3. 获取当前父进程编号？

`os.getppid()`



# 目录

## Contents

- ◆ 多任务的介绍
- ◆ 多进程的介绍
- ◆ 多进程完成多任务
- ◆ 获取进程编号
- ➔ ◆ 进程的注意点

不共享数据、主进程等待子进程结束



# 学习目标

Learning Objectives

1. 知道并能够说出进程的注意点

## 进程的注意点介绍

1. 进程之间不共享全局变量
2. 主进程会等待所有的子进程执行结束再结束

## 进程间不共享全局变量

例如，在不同进程中修改列表`my_list[]`并新增元素，试着在各个进程中观察列表的最终结果。

```
import multiprocessing
import time
# 全局变量
my_list = []
# 写入数据
def write_data():
    for i in range(3):
        my_list.append(i)
        print("add:", i)
        print("write_data", my_list)
# 读取数据
def read_data():
    print("read_data", my_list)
```

```
if __name__ == '__main__':
    # 创建写入数据进程
    write_process = multiprocessing.Process(target=write_data)
    # 创建读取数据进程
    read_process = multiprocessing.Process(target=read_data)

    # 启动进程执行相应任务
    write_process.start()
    time.sleep(1)
    read_process.start()
```



## 进程间不共享全局变量



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, 是因为操作的不是同一个进程里面的全局变量, 只不过不同进程里面的全局变量名字相同而已。

## 默认主进程会等待所有的子进程执行结束再结束

假如我们现在创建一个子进程，子进程执行完大概需要2秒钟，现在让主进程执行1秒钟就退出程序：

```
import multiprocessing
import time

# 工作函数
def work():
    for i in range(10):
        print("工作中...")
        time.sleep(0.2)
```

```
if __name__ == '__main__':
    # 创建子进程
    work_process = multiprocessing.Process(target=work)
    # 启动子进程
    work_process.start()
    # 延时1秒
    time.sleep(1)
    print("主进程执行完毕")
```

通过上面代码的执行结果，我们可以得知：主进程会等待所有的子进程执行结束再结束。

## 不让主进程等待子进程，方法1：子进程设置守候进程

子进程设置守护主进程的目的：是主进程退出子进程销毁，**不让主进程再等待子进程去执行。**

```
import multiprocessing
import time

# 工作函数
def work():
    for i in range(10):
        print("工作中...")
        time.sleep(0.2)
```

```
if __name__ == '__main__':
    # 创建子进程
    work_process = multiprocessing.Process(target=work)
    ## 设置守护主进程
    work_process.daemon = True
    # 启动子进程
    work_process.start()

    # 延时1秒
    time.sleep(1)
    print("主进程执行完毕")
```

不让主进程等待子进程，方法2：子进程自己主动的终止子进程

守护进程或子进程提前结束

```
import multiprocessing
import time

# 工作函数
def work():
    for i in range(10):
        print("工作中...")
        time.sleep(0.2)
```

```
if __name__ == '__main__':
    # 创建子进程
    work_process = multiprocessing.Process(target=work)

    # 启动子进程
    work_process.start()

    # 延时1秒
    time.sleep(1)

    # 手动结束子进程
    work_process.terminate() #这种不建议使用，僵尸进程，不会清理资源
    print("主进程执行完毕")
```



# 总结

1. 为了保证子进程能够正常的运行，主进程会等所有的子进程执行完成以后再销毁，设置守护主进程的目的是主进程退出子进程销毁，不让主进程再等待子进程去执行。
2. 设置守护主进程方式：子进程对象.daemon = True
3. 销毁子进程方式：子进程对象.terminate()



传智教育旗下高端IT教育品牌