

并发实践

并发并行

这一部分主要以实验对比形式展现。

一、IO任务任务实验：

方面方式：base, 多线程, 多进程, 进程+queue, 协程 百度图片下载

Python

```
1 def getManyPages(keyword, pages):
2     # 获取下载图片的url路径
3     params=[]
4     for i in range(30, 30*pages+30, 30):
5         params.append({
6             'tn': 'resultjson_com', 'ipn': 'rj',
7             'ct': 201326592, 'is': '', 'fp': 'result',
8             'queryWord': keyword,
9             'cl': 2, 'lm': -1, 'ie': 'utf-8', 'oe': 'utf-8',
10            'adpicid': '', 'st': -1, 'z': '', 'ic': 0,
11            'word': keyword, 's': '', 'se': '',
12            'tab': '', 'width': '', 'height': '', 'face': 0,
13            'istype': 2, 'qc': '',
14            'nc': 1, 'fr': '',
15            'pn': i,
16            'rn': 30, 'gsm': '1e', '1488942260214': ''
17        })
18     url = 'https://image.baidu.com/search/acjson'
19     urls = []
20     for i in params:
21         try:
22             urls.extend(requests.get(url, params=i).json().get('data'))
23         except:
24             break
25
26     return urls
```

1.1 base: 单线程

```
1
2 def getImg(dataList, localPath):
3     if not os.path.exists(localPath): # 新建文件夹
4         os.mkdir(localPath)
5
6     x = 0
7     for i in dataList:
8         x += 1
9         if i.get('thumbURL') != None:
10            print('正在下载 {0}: {1}'.format(x, i.get('thumbURL')))
11            ir = requests.get(i.get('thumbURL'))
12            open(localPath + '%d.jpg' % (x), 'wb').write(ir.content)
13        else:
14            print('图片 {} 链接不存在'.format(x))
15
16
17 if __name__ == '__main__':
18     dataList = getManyPages(u'火影', 30) # 参数1:关键字, 参数2:要下载的页数
19     getImg(dataList, 'data/') # 参数2:指定保存的路径
```

930次图片下载请求, 耗时 133.3475s

1.2 多线程

```
1 def getImg(url, localPath, timestamp):
2
3     if url is not None:
4         print('正在下载 : {}'.format(url))
5         ir = requests.get(url)
6         open(localPath + '%d.jpg' % (timestamp*1000000), 'wb').write(ir.co
7         return Result(1)
8     else:
9         return Result(0)
10
11 def save_pic(dataList, localPath):
12     if not os.path.exists(localPath): # 新建文件夹
13         os.mkdir(localPath)
14
15     futures_set = set()
16     with concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:
17         for item in dataList:
18             url = item.get('thumbURL', None)
19             if url is not None:
20                 timestamp = time.time()
21                 futures_set.add(executor.submit(getImg, url, localPath, ti
22
23     summary = wait_for(futures_set)
24     if summary.canceled:
25         executor.shutdown()
26     summarize(summary, 4)
27     return summary
28
29 if __name__ == '__main__':
30     dataList = getManyPages('火影', 30) # 参数1:关键字, 参数2:要下载的页数
31     summary = save_pic(dataList, 'data/') # 参数2:指定保存的路径
```

930次图片下载请求, 耗时83.56204199790955

1.3 多进程

```

1  def getImg(url, localPath, timestamp):
2
3      if url is not None:
4          print('正在下载 : {}'.format(url))
5          ir = requests.get(url)
6          open(localPath + '%d.jpg' % (timestamp*1000000), 'wb').write(ir.co
7          return Result(1)
8      else:
9          return Result(0)
10
11 def save_pic(dataList, localPath):
12     if not os.path.exists(localPath): # 新建文件夹
13         os.mkdir(localPath)
14
15     futures_set = set()
16     with concurrent.futures.ProcessPoolExecutor(max_workers=4) as executor
17         for item in dataList:
18             url = item.get('thumbURL', None)
19             if url is not None:
20                 timestamp = time.time()
21                 futures_set.add(executor.submit(getImg, url, localPath, ti
22
23     summary = wait_for(futures_set)
24     if summary.canceled:
25         executor.shutdown()
26     summarize(summary, 4)
27     return summary
28
29 if __name__ == '__main__':
30     dataList = getManyPages('火影', 30) # 参数1:关键字, 参数2:要下载的页数
31     summary = save_pic(dataList, 'data/') # 参数2:指定保存的路径

```

930次图片下载请求, 耗时82.85834980010986

1.4 process+queue

```

1  def add_jobs(dataList, jobs, localPath):
2      for index, item in enumerate(dataList, start=1):
3          jobs.put((item.get('thumbURL', None), localPath))
4
5      return index
6
7
8  def save_pic(dataList, localPath, concurrency):
9      if not os.path.exists(localPath): # 新建文件夹

```

```

10         os.mkdir(localPath)
11
12         canceled = False
13         jobs = multiprocessing.JoinableQueue() # task queue
14         # jobs = multiprocessing.Queue() # task queue
15         results = multiprocessing.Queue() # 结果队列
16         create_process(jobs, results, concurrency)
17         todo = add_jobs(datalist, jobs, localPath) # 将待办任务加入任务队列
18
19         try:
20             jobs.join()
21         except KeyboardInterrupt:
22             canceled = True
23         success = 0
24         while not results.empty():
25             result = results.get_nowait()
26             success += result.success
27
28         return Summary(todo, success, canceled)
29
30
31 def create_process(jobs, results, concurrency):
32     for _ in range(concurrency):
33         process = multiprocessing.Process(target=worker, args=(jobs, results))
34         process.daemon = True
35         process.start()
36
37
38 def worker(jobs, results):
39     while True:
40         try:
41             url, localpath = jobs.get()
42             try:
43                 result = get_img(url, localPath=localpath)
44                 results.put(result)
45             except Exception as e:
46                 print(e)
47         finally:
48             jobs.task_done()
49
50
51 if __name__ == '__main__':
52     dataList = get_many_pages('火影', 30) # 参数1:关键字, 参数2:要下载的页数
53     summary = save_pic(dataList, 'data/') # 参数2:指定保存的路径

```

930次图片下载请求, 耗时 84.98532915115356

1.5 协程

Python

```
1  async def getImg(i, localPath):
2
3      if not os.path.exists(localPath): # 新建文件夹
4          os.mkdir(localPath)
5
6      down_pic(i, localPath)
7
8  def down_pic(i, localPath):
9      timestamp = time.time() * 1000000
10     if i.get('thumbURL') != None:
11         print('正在下载 {}'.format(i.get('thumbURL')))
12         ir = requests.get(i.get('thumbURL'))
13         open(localPath + '%d.jpg' % (timestamp), 'wb').write(ir.content)
14     else:
15         print('图片 {} 链接不存在'.format(i.get('thumbURL')))
16
17
18 if __name__ == '__main__':
19     dataList = getManyPages(u'火影', 30) # 参数1:关键字, 参数2:要下载的页数
20
21     loop = asyncio.get_event_loop()
22     tasks = [getImg(i, 'data/') for i in dataList]
23     loop.run_until_complete(asyncio.wait(tasks))
24     loop.close()
```

930次图片下载请求， 耗时65.67994403839111

上述方式比较

项目	描述	耗时	加速
base	单线程, 930下载	133.35s	1
thread	4线程 930下载	83.56s	1.60
multiprocess	4进程,930下载	82.86s	1.61
process + queue	4进程+队列,930下载	84.98s	1.57
coroutine	协程, 930下载	65.78s	2.03

总结：对于IO密集型任务，协程表现好

二、CPU任务：

方面方式：base, 多线程, 多进程, 进程+queue, 协程

数值计算：x (a, b为正整数, $a < x < b$),

计算式子 $1/x + \text{math.sqrt}(x) + \text{math.sin}(x) + \text{math.cos}(x)$ 最小值

$$f(x) = \sum_{n=1}^x \left(\cos n + \sin n + \frac{1}{n} + \sqrt{n} \right) \leftarrow$$

公式实现

```
1 def getSum(num):
2     print('计算{}'.format(num))
3     total = 0.0
4     for x in range(1, num):
5         total += 1/x + math.sqrt(x) + math.sin(x) + math.cos(x)
6     return Result(num, total, 1)
```

Python

1. base

```
1 def getSum(num):
2     for i in range(1, num):
3         total = 0.0
4         for x in range(1, i):
5             total += 1/x + math.sqrt(x) + math.sin(x) + math.cos(x)
6         print(total)
7
8     return total
9
10 print(getSum(20000))
```

Python

20000次计算, 平均耗时131.50s

2. 多线程

```

1  def save_pic(num):
2
3
4      futures_set = set()
5      with concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:
6          for i in range(1, num):
7              if i is not None:
8                  futures_set.add(executor.submit(getSum, i))
9
10         summary = wait_for(futures_set)
11         if summary.canceled:
12             executor.shutdown()
13         # summarize(summary, 4)
14         return summary
15
16 if __name__ == '__main__':
17     summary = save_pic(20000) #

```

20000次计算，平均耗时124.02s

3. multiprocessing

```

1  def save_pic(num, concurrency):
2
3
4      futures_set = set()
5      with concurrent.futures.ProcessPoolExecutor(max_workers=concurrency) as executor:
6          # for url in get_job(num):
7          for i in range(1, num):
8              if i is not None:
9                  futures_set.add(executor.submit(getSum, i))
10
11         summary = wait_for(futures_set)
12         if summary.canceled:
13             executor.shutdown()
14         # summarize(summary, 4)
15         return summary

```

20000次计算，平均耗时71.51s

4. queue + process


```
1 def add_jobs(num, jobs):
2     for index in range(1, num):
3         jobs.put(index)
4
5     return index
6
7 def save_pic(num, concurrency):
8     # num 计算次数, concurrency 进程个数
9     canceled = False
10    jobs = multiprocessing.JoinableQueue() # task queue
11    # jobs = multiprocessing.Queue() # task queue
12    results = multiprocessing.Queue() # 结果队列
13    create_process(jobs, results, concurrency)
14    todo = add_jobs(num, jobs) # 将待办任务加入任务队列
15
16    try:
17        jobs.join()
18    except KeyboardInterrupt:
19        canceled = True
20    success = 0
21    while not results.empty():
22        result = results.get_nowait()
23        success += result.success
24
25    return Summary(todo, success, canceled)
26
27 def create_process(jobs, results, concurrency):
28     for _ in range(concurrency):
29         process = multiprocessing.Process(target=worker, args=(jobs, results))
30         process.daemon = True
31         process.start()
32
33
34 def worker(jobs, results):
35     while True:
36         try:
37             sum_result = jobs.get()
38             try:
39                 result = getSum(sum_result)
40                 results.put(result)
41             except Exception as e:
42                 print(e)
43         finally:
44             jobs.task_done()
```

20000次计算, 平均耗时77.31s

5. 协程

Python

```
1  async def getSum(num):
2      print('计算{}'.format(num))
3      total = 0.0
4      for x in range(1, num):
5          total += 1 / x + math.sqrt(x) + math.sin(x) + math.cos(x)
6      return total
7
8
9  if __name__ == '__main__':
10     loop = asyncio.get_event_loop()
11     tasks = [getSum(i) for i in range(1, 20000)]
12     loop.run_until_complete(asyncio.wait(tasks))
13     loop.close()
```

20000次计算，平均耗时124.85s

上述方式比较:

项目	描述	耗时	加速
base	单线程, 20000次计算	131.50s	1
thread	4线程 20000次计算	124.02s	1.06
multiprocess	4进程,20000次计算	71.51s	1.84
process + queue	4进程+队列,20000次计算	77.31s	1.70
coroutine	协程, 20000次计算	124.85s	1.05

总结：面对CPU任务，多进程表现好