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
def getManyPages(keyword,pages):
    # 获取下载图片的url路径
    params=[]
    for i in range(30, 30*pages+30, 30):
        params.append({
                       'tn': 'resultjson_com', 'ipn': 'rj',
                       'ct': 201326592, 'is': '', 'fp': 'result',
                       'queryWord': keyword,
                       'cl': 2, 'lm': -1, 'ie': 'utf-8', 'oe': 'utf-8',
                       'adpicid': '', 'st': -1, 'z': '', 'ic': 0,
                       'word': keyword, 's': '', 'se': '',
                       'tab': '', 'width': '', 'height': '', 'face': 0,
                       'istype': 2, 'qc': '',
                       'nc': 1, 'fr': '',
                       'pn': i,
                       'rn': 30, 'gsm': '1e', '1488942260214': ''
                   })
    url = 'https://image.baidu.com/search/acjson'
    urls = []
    for i in params:
        try:
            urls.extend(requests.get(url,params=i).json().get('data'))
        except:
            break

    return urls
```

1.1 base: 单线程

```python

def getImg(dataList, localPath):
    if not os.path.exists(localPath):  # 新建文件夹
        os.mkdir(localPath)

    x = 0
    for i in dataList:
        x += 1
        if i.get('thumbURL') != None:
            print('正在下载 {0}: {1}'.format(x, i.get('thumbURL')))
            ir = requests.get(i.get('thumbURL'))
            open(localPath + '%d.jpg' % (x), 'wb').write(ir.content)
        else:
            print('图片 {} 链接不存在'.format(x))


if __name__ == '__main__':
    dataList = getManyPages(u'火影', 30)  # 参数1:关键字, 参数2:要下载的页数
    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

```python
def getImg(url, localPath, timestamp):

    if url is not None:
        print('正在下载 : {}'.format(url))
        ir = requests.get(url)
        open(localPath + '%d.jpg' % (timestamp*1000000), 'wb').write(ir.co
        return Result(1)
    else:
        return Result(0)

def save_pic(dataList, localPath):
    if not os.path.exists(localPath):  # 新建文件夹
        os.mkdir(localPath)

    futures_set = set()
    with concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:
        for item in dataList:
            url = item.get('thumbURL', None)
            if url is not None:
                timestamp = time.time()
                futures_set.add(executor.submit(getImg, url, localPath, ti

        summary = wait_for(futures_set)
        if summary.canceled:
            executor.shutdown()
        summarize(summary, 4)
        return summary

if __name__ == '__main__':
    dataList = getManyPages('火影', 30)  # 参数1:关键字，参数2:要下载的页数
    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

```python
def getImg(url, localPath, timestamp):

    if url is not None:
        print('正在下载 : {}'.format(url))
        ir = requests.get(url)
        open(localPath + '%d.jpg' % (timestamp*1000000), 'wb').write(ir.co
        return Result(1)
    else:
        return Result(0)

def save_pic(dataList, localPath):
    if not os.path.exists(localPath):  # 新建文件夹
        os.mkdir(localPath)

    futures_set = set()
    with concurrent.futures.ProcessPoolExecutor(max_workers=4) as executor
        for item in dataList:
            url = item.get('thumbURL', None)
            if url is not None:
                timestamp = time.time()
                futures_set.add(executor.submit(getImg, url, localPath, ti

        summary = wait_for(futures_set)
        if summary.canceled:
            executor.shutdown()
        summarize(summary, 4)
        return summary

if __name__ == '__main__':
    dataList = getManyPages('火影', 30)  # 参数1:关键字, 参数2:要下载的页数
    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

```python
def add_jobs(dataList, jobs, localPath):
    for index, item in enumerate(dataList, start=1):
        jobs.put((item.get('thumbURL', None), localPath))

    return index


def save_pic(datalist, localPath, concurrency):
    if not os.path.exists(localPath):  # 新建文件夹
```

```
        os.mkdir(localPath)

    canceled = False
    jobs = multiprocessing.JoinableQueue()  # task queue
    # jobs = multiprocessing.Queue()  # task queue
    results = multiprocessing.Queue()  # 结果队列
    create_process(jobs, results, concurrency)
    todo = add_jobs(datalist, jobs, localPath)  # 将待办任务加入任务队列

    try:
        jobs.join()
    except KeyboardInterrupt:
        canceled = True
    success = 0
    while not results.empty():
        result = results.get_nowait()
        success += result.success

    return Summary(todo, success, canceled)


def create_process(jobs, results, concurrency):
    for _ in range(concurrency):
        process = multiprocessing.Process(target=worker, args=(jobs, resul
        process.daemon = True
        process.start()


def worker(jobs, results):
    while True:
        try:
            url, localpath = jobs.get()
            try:
                result = getImg(url, localPath=localpath)
                results.put(result)
            except Exception as e:
                print(e)
        finally:
            jobs.task_done()


if __name__ == '__main__':
    dataList = getManyPages('火影', 30)  # 参数1:关键字，参数2:要下载的页数
    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
async def getImg(i, localPath):

    if not os.path.exists(localPath):  # 新建文件夹
        os.mkdir(localPath)

    down_pic(i, localPath)

def down_pic(i, localPath):
    timestamp = time.time() * 1000000
    if i.get('thumbURL') != None:
        print('正在下载 {0}'.format(i.get('thumbURL')))
        ir = requests.get(i.get('thumbURL'))
        open(localPath + '%d.jpg' % (timestamp), 'wb').write(ir.content)
    else:
        print('图片 {} 链接不存在'.format(i.get('thumbURL')))


if __name__ == '__main__':
    dataList = getManyPages(u'火影', 30)  # 参数1:关键字, 参数2:要下载的页数

    loop = asyncio.get_event_loop()
    tasks = [getImg(i, 'data/') for i in dataList]
    loop.run_until_complete(asyncio.wait(tasks))
    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 + math.sqrt(x) + math.sin(x) + math.cos(x)最小值

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

公式实现

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

1. base

```python
def getSum(num):
    for i in range(1, num):
        total = 0.0
        for x in range(1, i):
            total += 1/x + math.sqrt(x) + math.sin(x) + math.cos(x)
        print(total)

    return total

print(getSum(20000))
```

20000次计算，平均耗时131.50s

2. 多线程

```python

def save_pic(num):

    futures_set = set()
    with concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:
        for i in range(1, num):
            if i is not None:
                futures_set.add(executor.submit(getSum, i))

        summary = wait_for(futures_set)
        if summary.canceled:
            executor.shutdown()
        # summarize(summary, 4)
        return summary

if __name__ == '__main__':
    summary = save_pic(20000)  #
```

20000次计算，平均耗时124.02s

3. multiprocess

```python
def save_pic(num, concurrency):

    futures_set = set()
    with concurrent.futures.ProcessPoolExecutor(max_workers=concurrency) a
        # for url in get_job(num):
        for i in range(1, num):
            if i is not None:
                futures_set.add(executor.submit(getSum, i))

        summary = wait_for(futures_set)
        if summary.canceled:
            executor.shutdown()
        # summarize(summary, 4)
        return summary
```

20000次计算，平均耗时71.51s

4. queue + process

```python
def add_jobs(num, jobs):
    for index in range(1, num):
        jobs.put(index)

    return index

def save_pic(num, concurrency):
    # num 计算次数, concurrency 进程个数
    canceled = False
    jobs = multiprocessing.JoinableQueue()  # task queue
    # jobs = multiprocessing.Queue()  # task queue
    results = multiprocessing.Queue()  # 结果队列
    create_process(jobs, results, concurrency)
    todo = add_jobs(num, jobs)  # 将待办任务加入任务队列

    try:
        jobs.join()
    except KeyboardInterrupt:
        canceled = True
    success = 0
    while not results.empty():
        result = results.get_nowait()
        success += result.success

    return Summary(todo, success, canceled)

def create_process(jobs, results, concurrency):
    for _ in range(concurrency):
        process = multiprocessing.Process(target=worker, args=(jobs, resul
        process.daemon = True
        process.start()


def worker(jobs, results):
    while True:
        try:
            sum_result = jobs.get()
            try:
                result = getSum(sum_result)
                results.put(result)
            except Exception as e:
                print(e)
        finally:
            jobs.task_done()
```

20000次计算，平均耗时77.31s

5. 协程

```python
async def getSum(num):
    print('计算{}'.format(num))
    total = 0.0
    for x in range(1, num):
        total += 1 / x + math.sqrt(x) + math.sin(x) + math.cos(x)
    return total


if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    tasks = [getSum(i) for i in range(1, 20000)]
    loop.run_until_complete(asyncio.wait(tasks))
    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