阅读笔记

Executor使用最多max\_workers 线程池的子类，以异步方式执行调用。

事件循环在一个线程中运行

标准睡眠，当前线程释放python解释器，如果存在，它可以与其他线程一起使用，但它是另一个主题。

asyncio通常非常适合IO绑定和高级 结构化网络代码。

asyncio提供了一组高级 API：

[同时运行Python协同程序](https://docs.python.org/3/library/asyncio-task.html#coroutine)并完全控制它们的执行;

执行[网络IO和IPC](https://docs.python.org/3/library/asyncio-stream.html#asyncio-streams) ;

控制[子过程](https://docs.python.org/3/library/asyncio-subprocess.html#asyncio-subprocess) ;

通过[队列](https://docs.python.org/3/library/asyncio-queue.html#asyncio-queues)分配任务;

[同步](https://docs.python.org/3/library/asyncio-sync.html#asyncio-sync)并发代码;

initializer是一个可选的callable，在每个工作线程的开头调用; initargs是传递给初始化程序的参数元组。如果初始化程序引发异常，则所有当前挂起的作业都将引发[BrokenThreadPool](https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.thread.BrokenThreadPool)，以及向池中提交更多作业的任何尝试。

类concurrent.futures.Future

封装可调用的异步执行。 [Future](https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.Future) 实例是由创建的[Executor.submit()](https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.Executor.submit)，不应该直接创建，除了测试。

cancel（）

尝试取消通话。如果当前正在执行呼叫且无法取消，则该方法将返回False，否则将取消呼叫并返回该方法True。

cancelled（）

True如果呼叫成功取消，则返回。

running（）

返回True如果当前正在执行的呼叫，无法取消。

done（）

返回True如果调用成功取消或结束运行。

concurrent.futures.wait（fs，timeout = None，return\_when = ALL\_COMPLETED ）

等待fs给出的[Future](https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.Future)实例（可能由不同[Executor](https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.Executor)实例创建 ）完成。返回一组命名的2元组。第一组命名包含在等待完成之前完成（已完成或已取消）的期货。第二组名称包含未完成的未来。donenot\_done

timeout可用于控制返回前等待的最大秒数。 timeout可以是int或float。如果未指定超时None，则等待时间没有限制。