**协程比较于线程的优点**：

1、协程极高的执行效率。因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显。

2、不需要多线程的锁机制，因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不加锁，只需要判断状态就好了，所以执行效率比多线程高很多。

小知识：利用多核CPU呢？最简单的方法是多进程+协程

**greelet**机制的主要思想是：生成器函数或者协程函数中的yield语句挂起函数的执行，直到稍后使用next()或send()操作进行恢复为止。可以使用一个调度器循环在一组生成器函数之间协作多个任务。

**greenlet**不是一种真正的并发机制，而是在同一线程内，在不同函数的执行代码块之间切换，实施“你运行一会、我运行一会”，并且在进行切换时必须指定何时切换以及切换到哪

实现代码：

**gevent**是一个基于协程（coroutine）的Python网络函数库。使用 greenlet 生成了一些协程，就可以在这些 协程里不断跳转执行，两个greenlet 之间的跳转被称为切换（switch）。通过切换，我们就可以实现对协程 的调度。还应该知道的是，每个 greenlet 都拥有一个父 greenlet ，这是在 greenlet 初始化时就确定的。 当一个 greenlet 执行完毕后，执行权会切换到其父 greenlet 中

**gevent特点**：

基于libev的快速事件循环，Linux上面的是epoll机制

基于greenlet的轻量级执行单元

API复用了Python标准库里的内容

支持SSL的协作式sockets

可通过线程池或c-ares实现DNS查询

通过monkey patching功能来使得第三方模块变成协作式

**#异步代码**

**import** requests  
**import** gevent  
**import** gevent.monkey  
  
*# 这里将socket变成异步*gevent.monkey.patch\_socket()  
  
url = **'http://httpbin.org/ip'  
  
def** run(i):  
 print(**"{}: {}"**.format(i, requests.get(url).text))  
  
tasks = [gevent.spawn(run, i) **for** i **in** range(50)]  
gevent.joinall(tasks)

运行大约1.7秒

**#同步请求**

**import** requests  
**import** datetime  
**def** run(url):  
 **for** i **in** range(50):  
 print(**"{}: {}"**.format(i, requests.get(url).text))  
  
url = **'http://httpbin.org/ip'**run(url)

运行大约34秒

二者比较有较大差距