

Forum:[Hack Forums](#)

Thread: [Faster python networking?](#)

incognito04 - September 29, 2019, 9:46 am

I'm currently making a supreme bot and I want the networking portion of my code to be as fast as possible. I've used python's async and I'm wondering what else can be done in-order to make it faster.

Goodies - September 29, 2019, 11:19 pm

I mean there's really nothing wrong with this... it's good enough for what you need (around 190-200 URLs in parallel) as long as you're aware of the limitations of the `asyncio.as_completed()` utility (it's not exactly scalable to thousands of coroutines, especially on Windows). It seems like you like using async generators. It's totally fine honestly, just remember that each iteration of ``async for`` is sequential just like a normal for loop and cannot be ITERATED in parallel, so some of your functions don't really get any benefit from being async generators. It is not an important distinction here because the basis of your ``async for`` statements (the `SupremeAPI.all_items` function) will simply just yield each item from a cpu-bound parsing function.

You could have used:

```
Code async def all_items(self):
```

```
    soup = await self.soupify("https://www.supremenewyork.com/shop/all")
```

```
    return [elem.a.get("href") for elem in soup.find_all("div", class_="inner-article")]
```

And subsequently:

```
Code async def all_items_info(self):
```

```
    for soup in await asyncio.gather(*[self.soupify(f"https://www.supremenewyork.com/{item}")
```

```
        for item in await self.all_items()]):
```

```
        yield (
```

```
            soup.find("h1", itemprop="name").text,
```

```
            int(soup.find("span", itemprop="price").text.strip('£'))
```

)

Note: You can replace your current `asyncio.as_completed` statement with `asyncio.gather` just as I have done without changing anything else. The `asyncio.as_completed` utility is leftover from prior to 3.5 when async generators didn't exist. Personally, I believe ``async for`` should be used sparingly.

But if you just wanna show off your obviously above average knowledge of ``async for`` then that's cool too lol.

incognito04 - September 30, 2019, 2:20 pm

(Sep 29, 2019 11:19 PM)Goodies Wrote: [\[CITE\]](#) I mean there's really nothing wrong with this... it's good enough for what you need (around 190-200 URLs in parallel) as long as you're aware of the limitations of the `asyncio.as_completed()` utility (it's not exactly scalable to thousands of coroutines, especially on Windows). It seems like you like using async generators. It's totally fine honestly, just remember that each iteration of ``async for`` is sequential just like a normal for loop and cannot be ITERATED in parallel, so some of your functions don't really get any benefit from being async generators. It is not an important distinction here because the basis of your ``async for`` statements (the `SupremeAPI.all_items` function) will simply just yield each item from a cpu-bound parsing function.

You could have used:

```
Code  async def all_items(self):
        soup = await self.soupify("https://www.supremenewyork.com/shop/all")
        return [elem.a.get("href") for elem in soup.find_all("div", class_="inner-article")]
```

And subsequently:

```
Code  async def all_items_info(self):
        for soup in await asyncio.gather(*[self.soupify(f"https://www.supremenewyork.com/{item}")
                                           for item in await self.all_items()]):
```

```
            yield (
                soup.find("h1", itemprop="name").text,
                int(soup.find("span", itemprop="price").text.strip('£'))
```

)Note: You can replace your current `asyncio.as_completed` statement with `asyncio.gather` just as I have done without changing anything else. The `asyncio.as_completed` utility is leftover from prior to 3.5 when async generators didn't exist. Personally, I believe ``async for`` should be used sparingly.

But if you just wanna show off your obviously above average knowledge of ``async for`` then thats cool too lol.

Thank you. Making "all_items" return a list instead of a generator has greatly benefited my program.

l0lwtf - October 1, 2019, 3:23 pm

Please note your bottleneck right now is not `asyncio` but `beautifulsoup`. HTML parsing in pure Python is extremely slow. Seems like the information you want can be extracted via regex or a simple `.find()` instead. Yes, this is quite ugly (zomg parsing html with regex) but since you want speed it is your only option.

You should address this and afterwards you can look at `asyncio`.

Goodies - October 1, 2019, 10:21 pm

(Oct 1, 2019 03:23 PM)l0lwtf Wrote: [\[CITE\]](#) Please note your bottleneck right now is not `asyncio` but `beautifulsoup`. HTML parsing in pure Python is extremely slow. Seems like the information you want can be extracted via regex or a simple `.find()` instead. Yes, this is quite ugly (zomg parsing html with regex) but since you want speed it is your only option.

You should address this and afterwards you can look at `asyncio`.

Yeah, this is a good point. If you install it, you can use "lxml" instead of "html.parser" which is significantly faster as it's python bindings for the C library libxml2.