# 库内容：

AsyncIO 是一个用于编写并发代码的 Python 标准库,它提供了一个高性能的事件循环和异步编程模型。AsyncIO 主要用于以下几个方面:

1. \*\*网络编程\*\*:AsyncIO 提供了一个高效的网络 I/O 模型,可用于编写高性能的网络服务器和客户端,如 Web 服务器、RPC 框架等。

2. \*\*异步 I/O\*\*:AsyncIO 可以帮助你编写高效的异步 I/O 代码,比如与数据库、文件系统、外部 API 等进行交互。

3. \*\*并发编程\*\*:AsyncIO 提供了一种基于协程的并发编程方式,可以帮助你编写出更加简洁、易维护的并发代码。

4. \*\*事件驱动编程\*\*:AsyncIO 的事件循环机制允许你编写事件驱动的应用程序,这在需要处理大量并发事件的场景下非常有用。

5. \*\*微服务和分布式系统\*\*:由于 AsyncIO 的高性能和可扩展性特点,它可以用于构建微服务架构和分布式系统。

总的来说,AsyncIO 是 Python 中一个非常强大和灵活的并发编程库,它可以帮助你编写出高性能、可扩展的 Python 应用程序。它广泛应用于网络编程、I/O 密集型应用、微服务架构等场景。

I/O 阻塞是指在执行某些 I/O 操作(如读写文件、网络通信等)时,程序会暂时停止执行,等待 I/O 操作完成后再继续执行。这种行为会影响程序的性能和响应速度。

具体来说,I/O 阻塞有以下特点:

1. \*\*同步 I/O\*\*:在传统的同步 I/O 编程中,当程序发起 I/O 操作时,它会一直等待 I/O 操作完成,期间无法处理其他任务,这就是 I/O 阻塞。

2. \*\*CPU 利用率低\*\*:由于程序在等待 I/O 操作完成时无法执行其他任务,因此 CPU 利用率会相对较低。

3. \*\*响应速度慢\*\*:I/O 阻塞会导致程序的响应速度变慢,尤其是在高并发、I/O 密集型的场景下。

4. \*\*线程模型局限性\*\*:在传统的基于线程的并发模型中,I/O 阻塞会导致线程被挂起,从而限制了并发处理的能力。

为了解决 I/O 阻塞的问题,异步 I/O 编程模型应运而生。在异步 I/O 中,程序在发起 I/O 操作时不会被阻塞,而是继续执行其他任务,当 I/O 操作完成时会收到通知。这样可以大大提高 CPU 利用率和程序的响应速度。

AsyncIO 就是 Python 中实现异步 I/O 编程的一个强大的库。它使用协程的方式来实现并发处理,从而避免了 I/O 阻塞的问题,提高了程序的性能和可扩展性。

`os` 库是 Python 中用于与操作系统交互的标准库之一,它提供了一系列与操作系统相关的功能和接口。`os` 库的主要用处包括以下几个方面:

1. \*\*文件和目录操作\*\*:

- 创建、删除、重命名文件和目录

- 获取文件和目录的属性信息

- 遍历目录结构

- 设置和获取当前工作目录

2. \*\*进程管理\*\*:

- 创建、终止和管理进程

- 获取进程 ID 和父进程 ID

- 等待进程终止并获取退出状态

3. \*\*环境变量管理\*\*:

- 获取和设置环境变量

- 操作系统相关的路径分隔符和换行符

4. \*\*系统相关操作\*\*:

- 获取操作系统名称和版本信息

- 执行系统命令并获取输出

- 访问和修改系统权限

5. \*\*跨平台支持\*\*:

- `os` 模块提供了许多跨平台的函数和常量,使得代码可以在不同操作系统上运行

总的来说,`os` 模块提供了一种直接与操作系统交互的方式,使得 Python 程序能够完成许多系统级别的任务,如文件管理、进程控制、环境配置等。这在系统编程、自动化脚本等场景下非常有用。同时,它还帮助 Python 程序实现了良好的跨平台性。

ClientSession 是 aiohttp 库中的一个核心组件,它用于在异步环境中发送 HTTP 请求。

ClientSession 的主要作用包括:

1. 建立 HTTP 会话

ClientSession 提供了一个持久的 HTTP 会话,可以在多个请求之间共享 cookies、headers 等信息。这样可以减少重复设置这些信息的开销。

1. 管理连接池

ClientSession 会管理底层的 TCP 连接,使用连接池来避免频繁的连接/断开开销。这对提高 HTTP 请求的性能很重要。

1. 支持异步

ClientSession 是基于 asyncio 实现的,可以与其他异步操作无缝集成,方便编写高性能的异步 HTTP 客户端程序。

1. 提供丰富的 API

ClientSession 提供了各种 HTTP 方法(GET、POST、PUT 等)的 API,以及对 cookies、headers、身份验证等的支持。

# def hero\_list():

进入英雄联盟官网的英雄链接https://lol.qq.com/data/info-heros.shtml，发现内容并不是储存在静态网页中，通过查看元素，找寻到了接口https://game.gtimg.cn/images/lol/act/img/js/heroList/hero\_list.js

# def run():

`asyncio.Semaphore(30)` 这段代码用于限制并发请求的数量,这在编写异步网络爬虫程序时非常有用。

在异步编程中,我们可以同时发起大量的网络请求,以提高整体的执行效率。但是如果直接发起大量请求,可能会给服务器造成负担,甚至导致网站访问被屏蔽。为了避免这种情况,我们可以使用 `asyncio.Semaphore` 来限制并发请求的数量。

`asyncio.Semaphore(30)` 的作用就是创建一个最多允许 30 个并发请求的信号量对象。

具体工作原理如下:

1. 当一个请求开始时,会先尝试获取信号量。

2. 如果当前并发请求数量未达到上限(30),则该请求可以立即获取信号量并继续执行。

3. 如果当前并发请求数量已达到上限,则该请求会被挂起,直到有其他请求释放了信号量。

4. 当请求执行完毕时,会释放占用的信号量,让其他挂起的请求有机会获取。

这样做的好处是:

1. 可以有效控制并发请求的数量,避免给服务器造成过大压力。

2. 对于需要长时间处理的请求,可以释放信号量供其他请求使用,提高整体的执行效率。

3. 可以根据服务器的承受能力,灵活调整并发请求的上限。

总之, `asyncio.Semaphore(30)` 是一种常见的限流策略,在编写高性能的异步网络爬虫程序时非常有用。

# def skins\_downloader()：

`.format()` 是 Python 中字符串格式化的一种方法,它可以用来将值插入到字符串中。

`.format()` 方法的主要作用如下:

1. \*\*替换占位符\*\*

- 您可以在字符串中使用花括号 `{}` 作为占位符,然后在 `.format()` 方法中传入相应的值,这些值将会替换掉这些占位符。

```python

name = "Alice"

age = 25

print("My name is {} and I'm {} years old.".format(name, age))

# Output: My name is Alice and I'm 25 years old.

```

2. \*\*控制格式\*\*

- 您可以在占位符中添加格式化选项,例如对齐方式、数字精度等,以控制值的显示方式。

```python

pi = 3.14159

print("The value of pi is {:.2f}".format(pi))

# Output: The value of pi is 3.14

```

3. \*\*按索引或关键字引用\*\*

- 占位符可以通过位置索引或关键字参数来引用对应的值。

```python

person = {"name": "Alice", "age": 25}

print("My name is {0[name]} and I'm {0[age]} years old.".format(person))

# Output: My name is Alice and I'm 25 years old.

```

4. \*\*格式化字符串文字(f-strings)\*\*

- 从 Python 3.6 开始,您还可以使用 f-strings 来执行更高级的字符串格式化,这种方式更加简洁和可读。

```python

name = "Alice"

age = 25

print(f"My name is {name} and I'm {age} years old.")

# Output: My name is Alice and I'm 25 years old.

```

总之,`.format()` 方法为您提供了一种灵活的方式来格式化字符串,使您能够将动态值插入到文本中,并控制它们的显示方式。这在需要构建动态文本输出时非常有用。