一、先分清"两类接口"

OpenAI 风格 API 把"生成"分成两条完全不同的端点（endpoint）：

1. /chat/completions – 聊天补全
   入参必须带 messages 数组（对话历史），返回 choices[0].message.content。
   典型用途：多轮对话、Assistant / System / User 角色区分。

2. /completions – 纯文本补全
   入参必须带 prompt 字符串（没有角色概念），返回 choices[0].text。
   典型用途：续写、完形填空、代码补全。

二、再分清"两种调用方式"

每条端点又各自提供"同步"和"异步"两个 Python 方法，于是 2×2=4 个 public 方法：

Table Copy

方法名	端点	是否阻塞	依赖库	返回类型
create_chat_completion_sync	/chat/completions	阻塞	requests	Dict 或 Response
create_chat_completion_async	/chat/completions	非阻塞	aiohttp	Dict（必定已 json）
create_completion_sync	/completions	阻塞	requests	Dict 或 Response
create_completion_async	/completions	非阻塞	aiohttp	Dict（必定已 json）

三、代码级差异对照

1. 入参
   chat_* 系列：第一个参数必须是 messages: List[Dict[str,str]]
   completion_* 系列：第一个参数必须是 prompt: str

2. 请求体
   chat_* 会把 messages 放进 JSON 的 "messages" 字段。
   completion_* 会把 prompt 放进 JSON 的 "prompt" 字段。
   其余字段（model/temperature/max_tokens）完全相同。

3. 网络 I/O
   sync 版用 requests.post，直接阻塞直到收到响应；
   async 版用 aiohttp，async with session.post … await response.json()，需要运行在 asyncio 事件循环中。

4. 错误处理
   sync 版：http 非 200 时原样把 Response 对象抛给调用者；
   async 版：任何 http 状态都会 await response.json()，如果后端返回非 JSON 会抛异常。

5. 代理
   sync 版直接用 self.proxies；
   async 版通过 _get_url_proxy 把代理写成字符串传给 aiohttp 的 proxy 参数。

四、一句话记忆
“chat vs completion”决定你传对话历史还是纯文本；
“sync vs async”决定你是阻塞等待还是 await 异步返回。