手动开发MCP项目(CS架构)

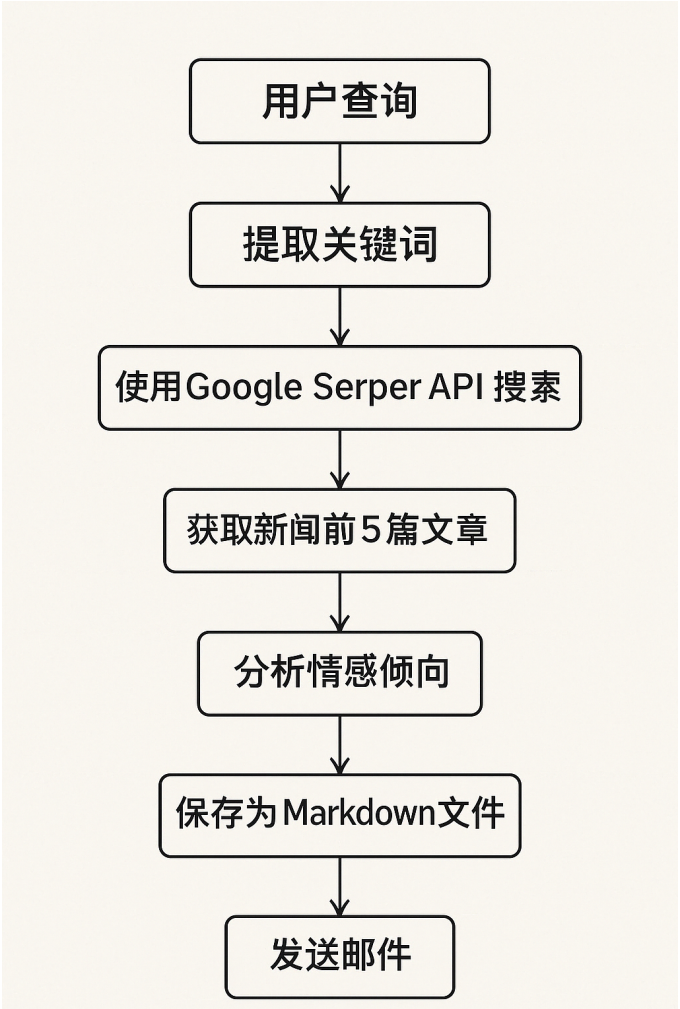
讲师：尚硅谷宋红康（江湖人称：康师傅）

官网： www.atguigu.com

douyin账号：是康师傅呀(尚硅谷)

1、项目需求分析

本项目旨在构建一个**本地智能舆情分析系统**，通过自然语言处理与多工具协作，实现用户查询意图的自动理解、新闻检索、情绪分析、结构化输出与邮件推送。具体如下：



系统整体采用 Client-Server **架构**，其中：

- 客户端（Client）作为用户的直接交互入口，负责接收输入、调用大语言模型进行语义解析与任务规划，并根据规划结果协调各类工具的执行流程；
- 而服务器端（Server）则作为工具能力提供者，内置多种独立功能模块，响应客户端的调用请求，完成实际的数据处理任务。

项目的执行流程：

1）在运行过程中，客户端会先加载本地模型配置，与服务器建立连接，并动态获取其可用工具列表。

2）用户输入查询后，客户端会自动调用大语言模型，将自然语言请求转化为结构化的“工具调用链”

3）客户端依次驱动服务器端工具完成如：关键词搜索、新闻采集、情绪倾向分析、报告生成与邮件发送等操作。

> 这一过程中，所有中间结果与最终输出都会自动保存，并反馈给用户。

项目特点：

整个系统运行于本地环境，通过标准输入输出通道进行进程间通信，无需依赖远程服务部署，确保了数据处理的私密性与可控性，适合用于敏感舆情监测、本地文本分析和低延迟的信息响应场景。

2、 MCP的环境准备

MCP的开发需要借助uv进行虚拟环境创建和依赖的管理。

2.1 安装uv

> 前面已经安装过了。没有安装的小伙伴，可以安装一下。

```
pip install uv
#或者
conda install uv  #针对于安装了Anoconda环境的用户
```

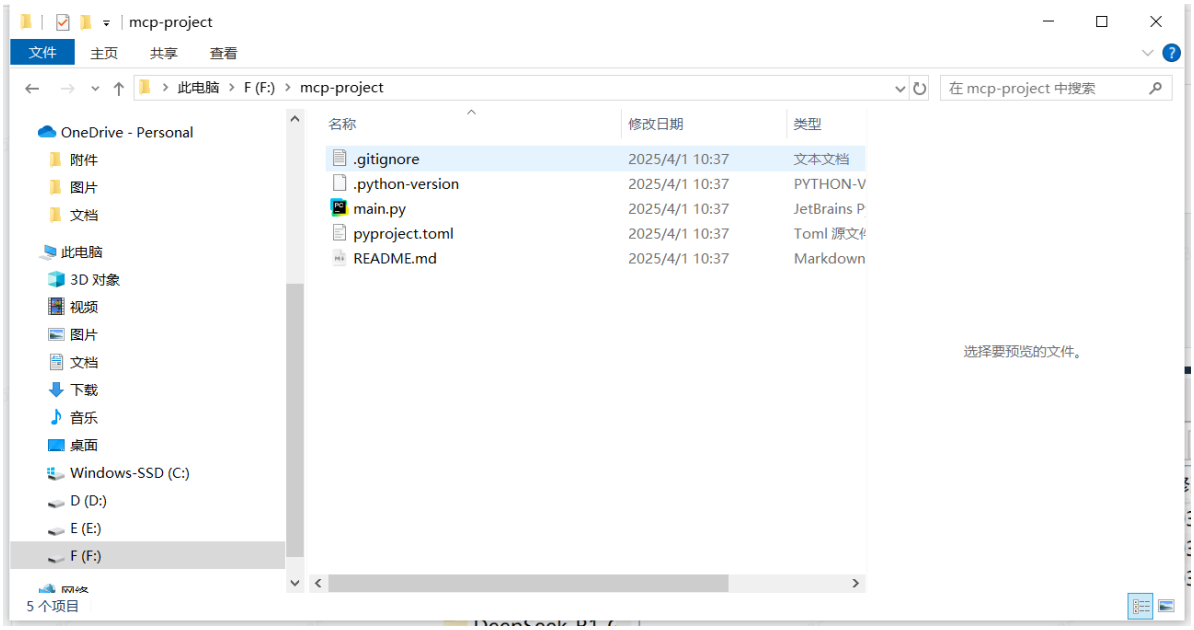
2.2 创建MCP项目

通过cd命令进入你要创建项目的空间，然后输入

```
uv init mcp-project
```

即可创建出一个空的MCP项目

```
(MCP) F:\>uv init mcp-project
Initialized project `mcp-project` at `F:\mcp-project`
```



在创建了这个空的MCP项目之后，我们需要创建两个Python文件，分别是client.py和server.py。

- client.py是我们的客户端，用户与客户端进行交互。
- server.py是服务端，其中包含了多种工具函数，客户端会对其中的工具函数进行调用。

这样，我们的MCP项目的创建便完成了。

3、代码实现

3.1 确定大模型参数

创建.env文件，在.env中添加相关的环境变量，其分别代表了阿里百炼平台的URL、选择的模型名称、个人的百炼平台API。

```
BASE_URL="https://dashscope.aliyuncs.com/compatible-mode/v1"
MODEL=qwen2.5-vl-32b-instruct
DASHSCOPE_API_KEY="sk-bffd34fa1ff34aad9d7e4ee927d67273"
```

3.2 client.py的构建

3.2.1 功能分析

我们首先从客户端入手，进行client.py的构建。其总体架构如下

```
[配置初始化]
     ↓
[连接工具服务器（MCP Server）]
     ↓
[用户提问] ← chat_loop()
     ↓
[LLM 规划工具调用链]
     ↓
[顺序执行工具链]
     ↓
[保存分析结果 & 最终回答]
```

运行过程中有以下几个关键步骤：

1）客户端从本地配置文件中读取必要的信息，完成大模型参数的设定（见 3.1 **确定大模型参数**），并初始化所需的运行环境（见 3.2.2 **初始化客户端配置**）。

2）程序启动服务端脚本并与其建立通信，获取可用的工具信息（见 3.2.3 **启动 MCP 工具服务连接**）。

3）完成连接后，客户端将根据用户输入的请求，协调内部调度器对工具链任务进行统一管理（见 3.2.4 **工具链任务调度器**）。

4）在与用户交互的过程中，系统会持续监听用户输入（见 3.2.5 **用户交互循环**），并调用大模型对任务进行智能拆解，规划合适的工具链执行顺序（见 3.2.6 **智能规划工具链**）。

5）每次任务执行完毕后，客户端将自动释放相关资源，确保系统稳定运行与退出（见 3.2.7 **关闭资源**）。

6）整个流程由主函数串联驱动，形成完整的一条执行主线（见 3.2.8 **主流程函数**）。

3.2.2 初始化客户端配置

在client.py中创建一个 MCPClient 类，用于封装和管理与MCP协议相关的所有客户端逻辑，随后在里面编写各种相关函数。

```
class MCPClient:
```

在 client.py中，我们定义了 MCPClient类，用来负责客户端的主要功能。在初始化时，程序会先准备好资源管理工具，方便后续自动管理连接。然后从配置文件中读取大模型的相关信息（如 API 密钥、模型名称等），并创建一个 OpenAI 客户端，方便后续调用模型。最后，还预留了一个与服务器通信的会话对象，等真正建立连接后再使用。整个过程是为后续的任务执行做好准备工作。

```
 def __init__(self):
        # 创建 AsyncExitStack，用于托管所有异步资源释放，这是为了后续连接 MCP Server 时使用 `async with` 语法自动管理上下文。
        self.exit_stack = AsyncExitStack()

        # 从环境中读取配置项
        self.openai_api_key = os.getenv("DASHSCOPE_API_KEY")
        self.base_url = os.getenv("BASE_URL")
        self.model = os.getenv("MODEL")

        # 对 LLM 相关配置进行初始化
        if not self.openai_api_key:
            raise ValueError("❌ 未找到 OpenAI API Key，请在 .env 文件中设置 DASHSCOPE_API_KEY")

        # 初始化 OpenAI 客户端对象
        self.client = OpenAI(api_key=self.openai_api_key, base_url=self.base_url)

        # 初始化 MCP Session（用于延迟赋值），等待连接 MCP Server 后再初始化它
        self.session: Optional[ClientSession] = None
```

3.2.3 启动MCP工具服务连接

connect_to_server这个函数的作用是连接并启动本地的服务器脚本。它会先判断脚本类型（必须是 `.py` 或 `.js`），再根据类型选择对应的启动方式（Python 或 Node.js）。接着，它会通过 MCP 提供的方式启动服务端脚本，并建立起与服务端的通信通道。建立连接后，客户端会初始化会

话，并获取服务器上有哪些工具可以使用，方便后续根据任务调用这些工具。整个过程相当于“把工具服务开起来，并准备好对话”。

```
┌──────────────────────┐
│   server.py    | ←——— 传入的路径
└──────────┬───────────┘
           ↓
┌──────────────────────┐
│   判断类型      |
|  .py → python  |
|  .js → node    |
└──────────┬───────────┘
           ↓
┌──────────────────────────────────────────────┐
|  启动脚本并建立 stdio 连接    |
└──────────┬───────────────────────────────────┘
           ↓
┌──────────────────────────────────────────────────────┐
|  创建 MCP 会话对象（ClientSession）|
└──────────┬───────────────────────────────────────────┘
           ↓
┌──────────────────────────────────────┐
|   初始化会话 + 获取工具    |
└──────────────────────────────────────┘

```

```python
async def connect_to_server(self, server_script_path: str):
    # 对服务器脚本进行判断，只允许是 .py 或 .js
    is_python = server_script_path.endswith('.py')
    is_js = server_script_path.endswith('.js')
    if not (is_python or is_js):
        raise ValueError("服务器脚本必须是 .py 或 .js 文件")

    # 确定启动命令，.py 用 python，.js 用 node
    command = "python" if is_python else "node"

    # 构造 MCP 所需的服务器参数，包含启动命令、脚本路径参数、环境变量（为None 表示默认）
    server_params = StdioServerParameters(command=command, args=[server_script_path], env=None)

    # 启动 MCP 工具服务进程（并建立 stdio 通信）
    stdio_transport = await self.exit_stack.enter_async_context(stdio_client(server_params))

    # 拆包通信通道，读取服务端返回的数据，并向服务端发送请求
    self.stdio, self.write = stdio_transport

    # 创建 MCP 客户端会话对象
    self.session = await self.exit_stack.enter_async_context(ClientSession(self.stdio, self.write))

    # 初始化会话
    await self.session.initialize()
```

```
25
26          # 获取工具列表并打印
27          response = await self.session.list_tools()
28          tools = response.tools
29          print("\n已连接到服务器，支持以下工具:", [tool.name for tool in
    tools])
30
```

3.2.4 工具链任务调度器

process_query这个函数是客户端处理用户提问的核心部分，它负责从接收问题，到规划任务、调用工具、生成回复，再到保存结果，完成整个闭环。

当用户输入一个问题后，程序会先去服务器获取当前支持的工具列表，比如“新闻搜索”“情感分析”“发送邮件”等，然后结合用户问题，从中提取关键词（比如“分析小米”中的“小米”），用它来生成一个统一的文件名，后续所有工具都会使用这个名字保存或读取文件，保证流程一致。这里请注意，要按照这句代码中设置好的格式标准提问

```
1   keyword_match = re.search(r'(关于|分析|查询|搜索|查看)([^的\s，。、？
    \n]+)', query)
```

会提取出 “关于|分析|查询|搜索|查看” 和“的”之间的内容作为关键词

接着，程序会把这个问题交给大语言模型，让它决定该如何使用这些工具（比如先查新闻，再分析情感，再发邮件），这一步叫做“工具链规划”。

拿到工具链后，程序就会按顺序一个个调用服务器上的工具，并在调用前动态地填入一些信息，比如刚才生成的文件名或路径。调用结果也会一一记录，作为下一步工具的输入或者最终输出的一部分。

等所有工具都执行完后，程序会再调用一次大模型，让它根据整个过程总结一个回答，这就是用户最终看到的内容。最后，这些对话记录（包括用户的提问和模型的回答）会被自动保存成一个 `.txt` 文件，方便后续查阅。

整个流程可以看作是一套自动化的信息处理流水线：用户只需要说一句话，系统就能从头到尾完成“理解 → 查询 → 分析 → 输出 → 保存”这一整套任务。

```python
async def process_query(self, query: str) -> str:
    # 准备初始消息和获取工具列表
    messages = [{"role": "user", "content": query}]
    response = await self.session.list_tools()

    available_tools = [
        {
            "type": "function",
            "function": {
                "name": tool.name,
                "description": tool.description,
                "input_schema": tool.inputSchema
            }
        } for tool in response.tools
    ]

    # 提取问题的关键词，对文件名进行生成。
    # 在接收到用户提问后就应该生成出最后输出的 md 文档的文件名，
    # 因为导出时若再生成文件名会导致部分组件无法识别该名称。
    keyword_match = re.search(r'(关于|分析|查询|搜索|查看)([^的\s，。、？\n]+)', query)
    keyword = keyword_match.group(2) if keyword_match else "分析对象"
    safe_keyword = re.sub(r'[\\/:*?"<>|]', '', keyword)[:20]
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    md_filename = f"sentiment_{safe_keyword}_{timestamp}.md"
    md_path = os.path.join("./sentiment_reports", md_filename)

    # 更新查询，将文件名添加到原始查询中，使大模型在调用工具链时可以识别到该信息
    # 然后调用 plan_tool_usage 获取工具调用计划
    query = query.strip() + f" [md_filename={md_filename}] [md_path={md_path}]"
    messages = [{"role": "user", "content": query}]

    tool_plan = await self.plan_tool_usage(query, available_tools)

    tool_outputs = {}
    messages = [{"role": "user", "content": query}]

    # 依次执行工具调用，并收集结果
    for step in tool_plan:
        tool_name = step["name"]
        tool_args = step["arguments"]

        for key, val in tool_args.items():
            if isinstance(val, str) and val.startswith("{{") and val.endswith("}}"):
                ref_key = val.strip("{} ")
                resolved_val = tool_outputs.get(ref_key, val)
                tool_args[key] = resolved_val

        # 注入统一的文件名或路径（用于分析和邮件）
        if tool_name == "analyze_sentiment" and "filename" not in tool_args:
            tool_args["filename"] = md_filename
```

```
51          if tool_name == "send_email_with_attachment" and
    "attachment_path" not in tool_args:
52              tool_args["attachment_path"] = md_path
53
54          result = await self.session.call_tool(tool_name,
    tool_args)
55
56          tool_outputs[tool_name] = result.content[0].text
57          messages.append({
58              "role": "tool",
59              "tool_call_id": tool_name,
60              "content": result.content[0].text
61          })
62
63      # 调用大模型生成回复信息，并输出保存结果
64      final_response = self.client.chat.completions.create(
65          model=self.model,
66          messages=messages
67      )
68      final_output = final_response.choices[0].message.content
69
70      # 对辅助函数进行定义，目的是把文本清理成合法的文件名
71      def clean_filename(text: str) -> str:
72          text = text.strip()
73          text = re.sub(r'[\\/:*?\"<>|]', '', text)
74          return text[:50]
75
76      # 使用清理函数处理用户查询，生成用于文件命名的前缀，并添加时间戳、设置
    输出目录
77      # 最后构建出完整的文件路径用于保存记录
78      safe_filename = clean_filename(query)
79      timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
80      filename = f"{safe_filename}_{timestamp}.txt"
81      output_dir = "./llm_outputs"
82      os.makedirs(output_dir, exist_ok=True)
83      file_path = os.path.join(output_dir, filename)
84
85      # 将对话内容写入 md 文档，其中包含用户的原始提问以及模型的最终回复结果
86      with open(file_path, "w", encoding="utf-8") as f:
87          f.write(f"🗣 用户提问：{query}\n\n")
88          f.write(f"🤖 模型回复：\n{final_output}\n")
89
90      print(f"📄 对话记录已保存为：{file_path}")
91
92      return final_output
93
```

3.2.5 用户交互循环

chat_loop这个函数就是客户端的“对话主入口”，也就是程序和用户真正开始交流的地方。

当程序运行到这里时，会先打印一句提示，告诉用户系统已经启动，可以开始提问了（输入 `quit` 就能退出）。然后它进入一个无限循环，不断等待用户输入问题。

每当用户输入一句话，程序就会把这个问题传给之前写好的 `process_query()` 函数，让它去自动规划任务、调用工具、生成回复。等处理完毕之后，再把结果打印出来。

如果在运行过程中出现了什么错误，比如连接失败、参数出错等，程序也会把错误信息捕捉并打印出来，而不会直接崩掉。

简单来说，这段代码就是负责“一问一答”的交互流程，是整个系统运转起来之后，用户最直接接触的那一部分。

```
async def chat_loop(self):
    # 初始化提示信息
    print("\n🤖 MCP 客户端已启动！输入 'quit' 退出")

    # 进入主循环中等待用户输入
    while True:
        try:
            query = input("\n你: ").strip()
            if query.lower() == 'quit':
                break

            # 处理用户的提问，并返回结果
            response = await self.process_query(query)
            print(f"\n🤖 AI: {response}")

        except Exception as e:
            print(f"\n⚠️ 发生错误: {str(e)}")

```

3.2.6 智能规划工具链

plan_tool_usage这个函数的作用是让大模型根据用户的问题，自动规划出一组需要使用的工具和调用顺序。首先，程序会整理当前可用的工具列表，并将它们写入系统提示中，引导模型只能从这些工具中选择。提示中还明确要求模型使用固定的 JSON 格式输出结果，避免生成多余的自然语言。

然后，程序将提示内容和用户的问题一起发送给大模型，请求模型生成一个工具调用计划。计划的内容通常包括每一步要使用的工具名称，以及对应的输入参数。

接收到模型的回复后，程序会尝试从中提取出合法的 JSON 内容，并进行解析。如果解析成功，就把结果作为工具调用链返回；如果解析失败，则打印错误信息并返回一个空的计划。这个过程确保了用户的问题可以自动转化为结构化的工具执行步骤，方便后续依次调用处理。

```
async def plan_tool_usage(self, query: str, tools: List[dict]) -> List[dict]:
    # 构造系统提示词 system_prompt。
    # 将所有可用工具组织为文本列表插入提示中，并明确指出工具名，
    # 限定返回格式是 JSON，防止其输出错误格式的数据。
    print("\n📤 提交给大模型的工具定义:")
    print(json.dumps(tools, ensure_ascii=False, indent=2))
    tool_list_text = "\n".join([
        f"- {tool['function']['name']}: {tool['function']['description']}"
        for tool in tools
    ])
    system_prompt = {
        "role": "system",
        "content": (
            "你是一个智能任务规划助手，用户会给出一句自然语言请求。\n"
            "你只能从以下工具中选择（严格使用工具名称）：\n"
            f"{tool_list_text}\n"
            "如果多个工具需要串联，后续步骤中可以使用 {{上一步工具名}} 占位。\n"
            "返回格式：JSON 数组，每个对象包含 name 和 arguments 字段。\n"
            "不要返回自然语言，不要使用未列出的工具名。"
        )
    }

    # 构造对话上下文并调用模型。
    # 将系统提示和用户的自然语言一起作为消息输入，并选用当前的模型。
    planning_messages = [
        system_prompt,
        {"role": "user", "content": query}
    ]

    response = self.client.chat.completions.create(
        model=self.model,
        messages=planning_messages,
        tools=tools,
        tool_choice="none"
    )

    # 提取出模型返回的 JSON 内容
    content = response.choices[0].message.content.strip()
    match = re.search(r"```(?:json)?\\s*([\s\S]+?)\\s*```", content)
    if match:
        json_text = match.group(1)
    else:
        json_text = content

```

```
        # 在解析 JSON 之后返回调用计划
        try:
            plan = json.loads(json_text)
            return plan if isinstance(plan, list) else []
        except Exception as e:
            print(f"❌ 工具调用链规划失败：{e}\n原始返回：{content}")
            return []

```

3.2.7 关闭资源

这个函数用于在程序结束时**关闭并清理所有已打开的资源**。它调用的是之前创建的 `AsyncExitStack`，这个工具会自动管理在程序运行过程中建立的连接，比如与服务器的通信通道。通过调用 `aclose()`，可以确保所有资源都被优雅地释放，避免出现内存泄漏或卡住进程的问题。简单来说，就是让程序**收尾干净、退出彻底**。

```
async def cleanup(self):
    await self.exit_stack.aclose()
```

3.2.8 主流程函数

这是程序的主入口，控制整个客户端的运行流程。

程序一开始会创建一个 `MCPClient` 实例，也就是我们之前封装的客户端对象。然后指定服务端脚本的位置，并尝试连接服务器。一旦连接成功，就进入对话循环，开始等待用户输入并处理问题。

无论程序中途正常退出还是出错，最后都会执行 `cleanup()`，确保所有资源都被安全关闭。

而最底部的 `if __name__ == "__main__":` 表示：只有当这个文件被直接运行时，才会执行 `main()`，这是 Python 程序的标准写法。整段代码就像是在启动按钮按下后，按照顺序完成“连接 → 运行 → 清理”的全过程。

```python
async def main():
    server_script_path = "F:\\mcp-project\\server.py"
    client = MCPClient()
    try:
        await client.connect_to_server(server_script_path)
        await client.chat_loop()
    finally:
        await client.cleanup()


if __name__ == "__main__":
    asyncio.run(main())
```

3.3 server.py的构建

3.3.1 功能分析

服务器端主要负责提供新闻搜索、情感分析、邮件发送等基础工具能力，供客户端调用。分别对应着如下的三个工具：

- search_google_news 用于在Google上搜寻相关新闻。
- analyze_sentiment 用于对语句进行舆情分析。
- send_email_with_attachment 用于将本地的文件发送至目标邮箱

核心功能剖析：

1）启动时，Server 会首先加载环境变量，配置必要的 API 密钥和服务信息。

2）注册一组功能模块，包括：调用 Serper API 搜索新闻内容、基于大模型分析文本情感、以及发送带有分析报告的邮件（对应各自的工具函数）。

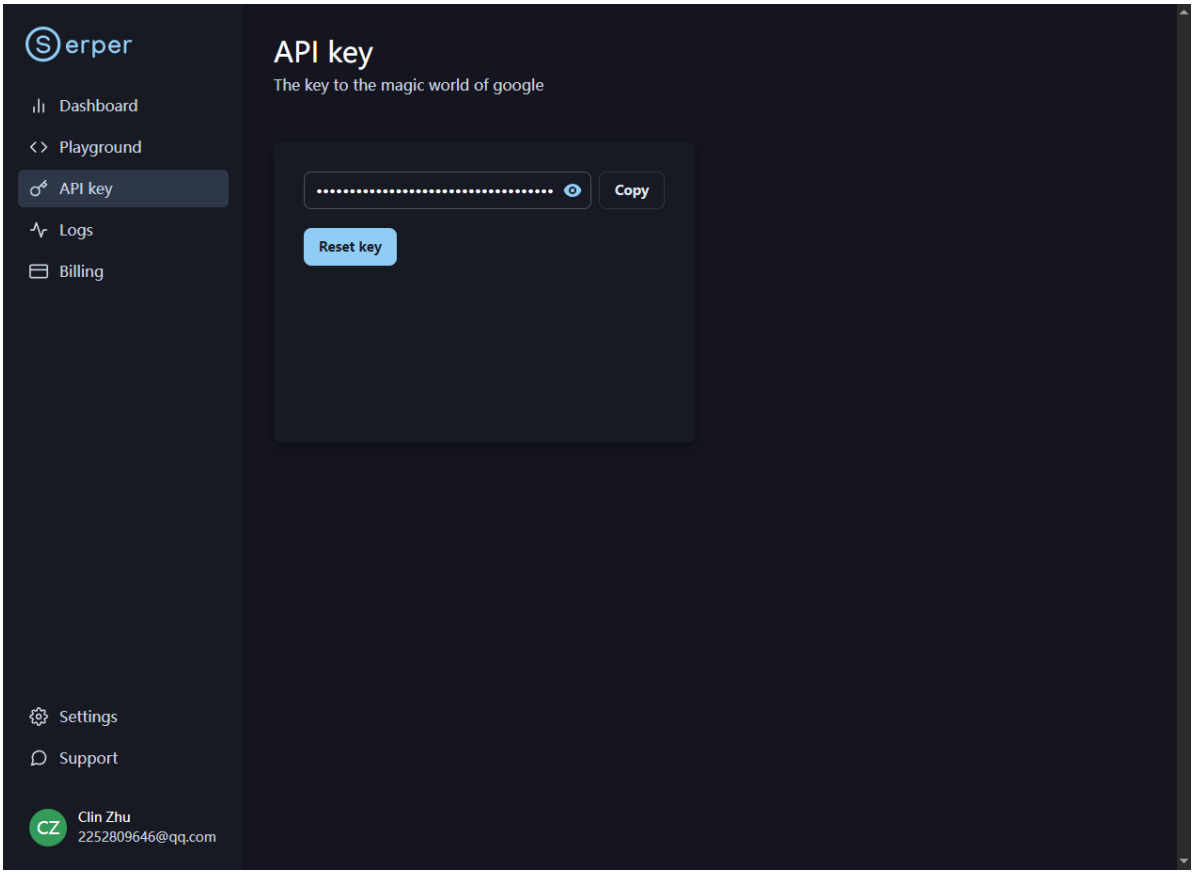
3）每个工具均以标准接口形式暴露，客户端可以根据任务需要按需调用。

4）程序以标准输入输出（stdio）模式运行，确保与客户端实现稳定、实时的交互。

3.3.1 search_google_news()

这个工具方法是通过Serper API 使用关键词从Google上去搜索获取新闻，返回前五条新闻并保存到本地文件中。

我们首先要申请Serper的API。来到网址 https://serper.dev/ ，简单注册后点击API key即可查看自己的API Key



在.env环境中配置Serper的API

```
SERPER_API_KEY="618b99091160938bb51b5968aad7312428bbba76"
```

search_google_news这个工具的作用是：**根据用户提供的关键词，调用 Serper API 搜索 Google 新闻，并返回前 5 条结果。**

它是通过 `@mcp.tool()` 装饰器注册为 MCP 工具，供客户端按需调用。

执行过程如下：

1. **读取 API 密钥**：程序从环境变量中获取用于访问 Serper API 的密钥，如果没配置好会直接报错。
2. **向新闻搜索接口发起请求**：将用户输入的关键词打包成请求体，发送给 Serper 提供的 Google News 接口。

3. **提取新闻信息**：从返回的数据中提取前 5 条新闻的标题、简介和链接，并整理成标准格式。
4. **保存为 JSON 文件**：将这些新闻内容保存成一个本地 `.json` 文件，文件名带有时间戳，方便归档。
5. **返回内容与保存路径**：最后，工具会将获取到的新闻数据、提示信息和保存路径一起返回，供客户端展示或传递给下一个工具使用。

```python
# @mcp.tool() 是 MCP 框架的装饰器，表明这是一个 MCP 工具。之后是对这个工具功能的描述
@mcp.tool()
async def search_google_news(keyword: str) -> str:
    """
    使用 Serper API（Google Search 封装）根据关键词搜索新闻内容，返回前5条标题、描述和链接。

    参数:
        keyword (str): 关键词，如 "小米汽车"

    返回:
        str: JSON 字符串，包含新闻标题、描述、链接
    """

    # 从环境中获取 API 密钥并进行检查
    api_key = os.getenv("SERPER_API_KEY")
    if not api_key:
        return "❌ 未配置 SERPER_API_KEY，请在 .env 文件中设置"

    # 设置请求参数并发送请求
    url = "https://google.serper.dev/news"
    headers = {
        "X-API-KEY": api_key,
        "Content-Type": "application/json"
    }
    payload = {"q": keyword}

    async with httpx.AsyncClient() as client:
        response = await client.post(url, headers=headers, json=payload)
        data = response.json()

    # 检查数据，并按照格式提取新闻，返回前五条新闻
    if "news" not in data:
        return "❌ 未获取到搜索结果"

    articles = [
        {
            "title": item.get("title"),
            "desc": item.get("snippet"),
            "url": item.get("link")
        } for item in data["news"][:5]
    ]

    # 将新闻结果以带有时间戳命名后的 JSON 格式文件的形式保存在本地指定的路径
    output_dir = "./google_news"
```

```
        os.makedirs(output_dir, exist_ok=True)
        filename = f"google_news_{datetime.now().strftime('%Y%m%d_%H%M%S')}.json"
        file_path = os.path.join(output_dir, filename)

        with open(file_path, "w", encoding="utf-8") as f:
            json.dump(articles, f, ensure_ascii=False, indent=2)

        return (
            f"✅ 已获取与 [{keyword}] 相关的前5条 Google 新闻：\n"
            f"{json.dumps(articles, ensure_ascii=False, indent=2)}\n"
            f"📄 已保存到：{file_path}"
        )

```

3.3.2 analyze_sentiment()

这个工具用于对一段新闻文本或任意内容进行**情绪倾向分析**，并将分析结果保存为 Markdown 格式的报告文件。

它通过 @mcp.tool() 注册为 MCP 工具，支持被客户端自动调用。

具体功能流程如下：

1. **读取大模型配置**：从环境变量中加载大模型的 API 密钥、模型名称和服务地址，用于后续调用语言模型。
2. **构造分析指令**：将用户传入的文本包装为一个“请分析这段内容情感倾向”的提示，发送给大模型处理。
3. **获取模型回复**：调用模型接口，获取分析结果文本，例如情绪是正面、中立或负面，以及理由说明。
4. **生成 Markdown 报告**：将原始文本与分析结果整理成结构清晰的 Markdown 报告，包含时间戳、原文、分析结果等部分。
5. **保存到本地文件**：将生成的报告保存到本地的 `./sentiment_reports` 文件夹中，文件名由用户指定，或默认自动生成带时间戳的名称。
6. **返回报告路径**：最终返回生成的报告文件路径，方便后续工具（如邮件发送）使用。

```
# @mcp.tool() 是 MCP 框架的装饰器，标记该函数为一个可调用的工具
@mcp.tool()
async def analyze_sentiment(text: str, filename: str) -> str:
    """
    对传入的一段文本内容进行情感分析，并保存为指定名称的 Markdown 文件。

    参数:
        text (str): 新闻描述或文本内容
        filename (str): 保存的 Markdown 文件名（不含路径）

    返回:
        str: 完整文件路径（用于邮件发送）
```

```
    """

    # 这里的情感分析功能需要去调用 LLM，所以从环境中获取 LLM 的一些相应配置
    openai_key = os.getenv("DASHSCOPE_API_KEY")
    model = os.getenv("MODEL")
    client = OpenAI(api_key=openai_key, base_url=os.getenv("BASE_URL"))

    # 构造情感分析的提示词
    prompt = f"请对以下新闻内容进行情绪倾向分析，并说明原因：\n\n{text}"

    # 向模型发送请求，并处理返回的结果
    response = client.chat.completions.create(
        model=model,
        messages=[{"role": "user", "content": prompt}]
    )
    result = response.choices[0].message.content.strip()

    # 生成 Markdown 格式的舆情分析报告，并存放进设置好的输出目录
    markdown = f"""# 舆情分析报告

**分析时间：** {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}

---

## 📥 原始文本

{text}

---

## 📊 分析结果

{result}
"""

    output_dir = "./sentiment_reports"
    os.makedirs(output_dir, exist_ok=True)

    if not filename:
        filename = f"sentiment_{datetime.now().strftime('%Y%m%d_%H%M%S')}.md"

    file_path = os.path.join(output_dir, filename)
    with open(file_path, "w", encoding="utf-8") as f:
        f.write(markdown)

    return file_path

```

3.2.3 send_email_with_attachment()

这个工具类是通过获取本地路径下的文件，然后将其发送给指定的邮箱

首先在环境中添加发件邮箱的SMTP配置信息。这里使用的是网易163邮箱。点击此处进行SMTP配置

开启POP3/SMTP服务。获取手机验证码之后即可查看自己的授权密码

将以下信息填入.env 注意，将邮箱和授权码换成你自己的

```
SMTP_SERVER=smtp.163.com
SMTP_PORT=465
EMAIL_USER=code14pudding@163.com
EMAIL_PASS=AZeDNekeCx6Ht3Vr
```

send_email_with_attachment这个工具用于**将生成好的Markdown 报告通过邮件发送给指定收件人**，并附带分析报告作为附件。

它通过 `@mcp.tool()` 装饰器注册为 MCP 工具，支持客户端自动调用发送邮件任务。

执行流程如下：

1. **读取发件邮箱配置**：从环境变量中读取 SMTP 服务器地址、端口、发件人邮箱和授权码。这些信息是发送邮件的基础（如 `smtp.163.com` 和授权密码）。
2. **拼接附件路径并检查是否存在**：程序会在默认的 `./sentiment_reports` 文件夹中查找附件，如果找不到文件，就会提示失败。
3. **构造邮件内容**：创建邮件对象，设置主题、正文、收件人等基本信息。
4. **添加附件**：将 Markdown 报告文件读取为二进制，并以附件形式加入邮件中。
5. **连接 SMTP 服务器并发送邮件**：通过 SSL 安全连接登录邮箱服务器，并发送邮件。如果发送成功会返回确认信息，如果失败则返回错误说明。

这个工具的作用非常关键，它完成了整个舆情分析流程的“最后一步”：**将分析结果自动发给用户**，实现真正的自动化闭环。对于实际应用来说，非常适合定时汇报、报告推送等场景。

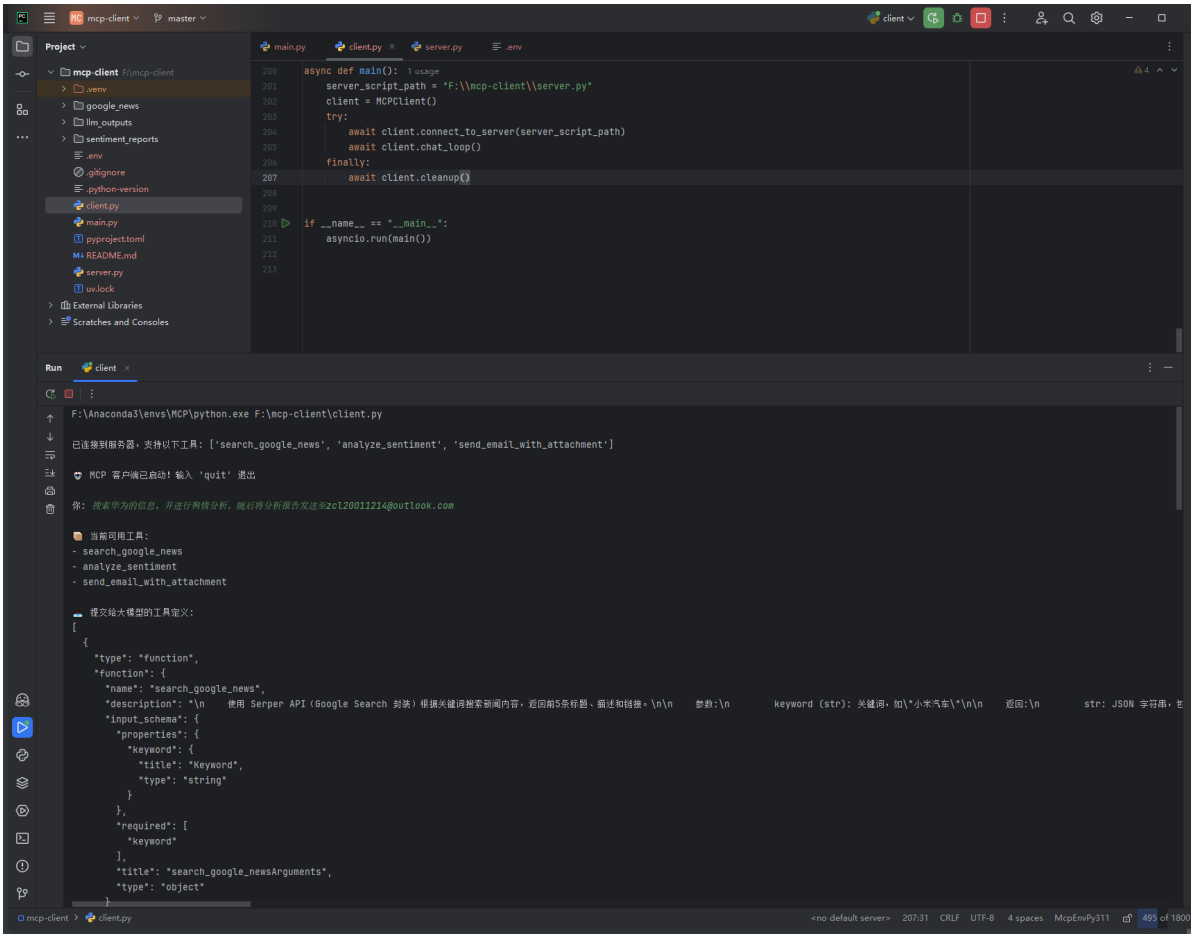
```python
@mcp.tool()
async def send_email_with_attachment(to: str, subject: str, body: str, filename: str) -> str:
    """
    发送带附件的邮件。

    参数:
        to: 收件人邮箱地址
        subject: 邮件标题
        body: 邮件正文
        filename (str): 保存的 Markdown 文件名（不含路径）

    返回:
        邮件发送状态说明
    """

    # 获取并配置 SMTP 相关信息
```

```
        smtp_server = os.getenv("SMTP_SERVER")  # 例如 smtp.qq.com
        smtp_port = int(os.getenv("SMTP_PORT", 465))
        sender_email = os.getenv("EMAIL_USER")
        sender_pass = os.getenv("EMAIL_PASS")

        # 获取附件文件的路径，并进行检查是否存在
        full_path =
    os.path.abspath(os.path.join("./sentiment_reports", filename))
        if not os.path.exists(full_path):
            return f"❌ 附件路径无效，未找到文件：{full_path}"

        # 创建邮件并设置内容
        msg = EmailMessage()
        msg["Subject"] = subject
        msg["From"] = sender_email
        msg["To"] = to
        msg.set_content(body)

        # 添加附件并发送邮件
        try:
            with open(full_path, "rb") as f:
                file_data = f.read()
                file_name = os.path.basename(full_path)
                msg.add_attachment(file_data, maintype="application",
    subtype="octet-stream", filename=file_name)
        except Exception as e:
            return f"❌ 附件读取失败：{str(e)}"

        try:
            with smtplib.SMTP_SSL(smtp_server, smtp_port) as server:
                server.login(sender_email, sender_pass)
                server.send_message(msg)
            return f"☑ 邮件已成功发送给 {to}，附件路径：{full_path}"
        except Exception as e:
            return f"❌ 邮件发送失败：{str(e)}"

```

如上便完成了主要内容的编写

4、测试

在运行的时候只需要运行client.py就可以运行整个项目了。

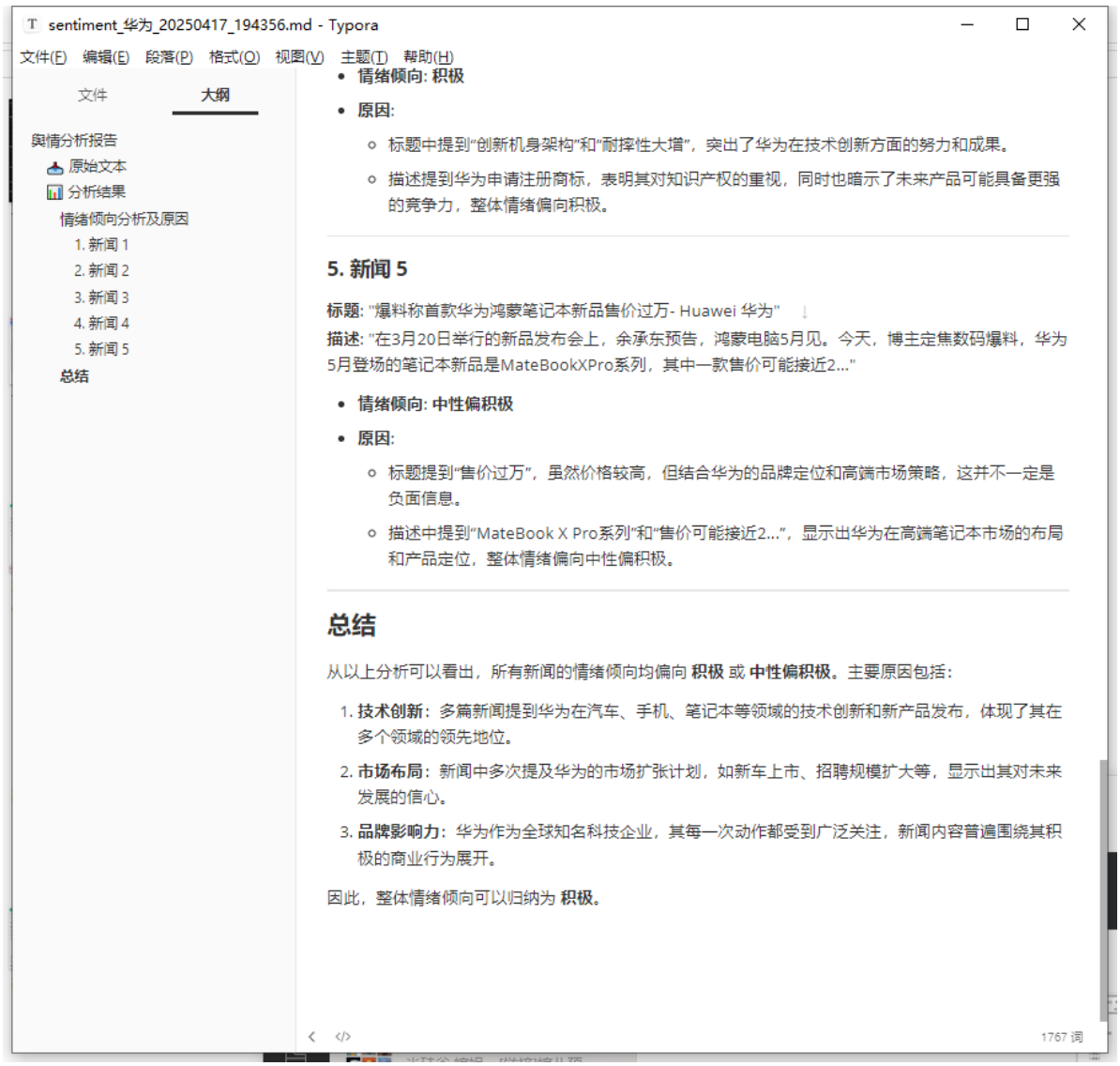


结果显示已发送，来到邮箱查看

在邮箱中查看，得知报告已经发送成功

查看一下报告内容，经验证，是一份完整的报告


sentiment_华为_20250417_194356.md - Typora
文件(F) 编辑(E) 段落(P) 格式(O) 视图(V) 主题(T) 帮助(H)
文件
大纲
舆情分析报告
原始文本
分析结果
情绪倾向分析及原因
1. 新闻 1
2. 新闻 2
3. 新闻 3
4. 新闻 4
5. 新闻 5
总结
舆情分析报告
分析时间： 2025-04-17 19:44:20
原始文本
已获取与 [华为] 相关的前5条 Google 新闻:
[
{
"title": "享界S9增程版上市：华为黑科技赋能，重塑豪华轿车新标杆",
"desc": "享界S9 增程版凭借十亿投入实现六大维度全面升级，涵盖配置、设计、科技、空间、安全、驾乘等方面。",
"url": "http://tech.huanqiu.com/article/4MIsNomqhHP"
},
{
"title": "华为鸿蒙智行冲销量 “第五界”首款车型计划秋季上市",
"desc": "【财新网】华为终端与上汽集团（600104.SH）联合打造的“尚界”品牌正式亮相，首款车型将在2025年秋季上市。4月16日，华为终端董事长余承东和上汽集团总裁...",
"url": "https://companies.caixin.com/2025-04-17/102310405.html"
},
{
"title": "华为2025年拟招聘应届生一万余人，较2024年预计实现两位数增幅",
"desc": "凤凰网科技讯4月17日，据羊城派报道，华为公司2025年“勇敢新世界”校园招聘计划将面向应届毕业生开放超60类技术岗位，招聘规模达1万余人，较2024年实现两位数增长。",
"url": "https://i.ifeng.com/c/8idDEKMl6Dv"
},
{
"title": "华为申请注册华为玄甲商标：创新机身架构 耐摔性大增",
"desc": "快科技4月16日消息，近日华为技术有限公司申请注册“华为玄甲”商标，国际分类为科学仪器，当前商标状态为等待实质审查。公开信息显示，华为玄甲是华为手机...",
"url": "https://news.mydrivers.com/1/1042/1042191.htm"
},
{
"title": "爆料称首款华为鸿蒙笔记本新品售价过万- Huawei 华为",
1767 词


sentiment_华为_20250417_194356.md - Typora
文件(F) 编辑(E) 段落(P) 格式(O) 视图(V) 主题(T) 帮助(H)
文件
大纲
舆情分析报告
原始文本
分析结果
情绪倾向分析及原因
1. 新闻 1
2. 新闻 2
3. 新闻 3
4. 新闻 4
5. 新闻 5
总结
情绪倾向: 积极
原因:
标题中提到"创新机身架构"和"耐摔性大增"，突出了华为在技术创新方面的努力和成果。
描述提到华为申请注册商标，表明其对知识产权的重视，同时也暗示了未来产品可能具备更强的竞争力，整体情绪偏向积极。
5. 新闻 5
标题: "爆料称首款华为鸿蒙笔记本新品售价过万- Huawei 华为"
描述: "在3月20日举行的新品发布会上，余承东预告，鸿蒙电脑5月见。今天，博主定焦数码爆料，华为5月登场的笔记本新品是MateBookXPro系列，其中一款售价可能接近2..."
情绪倾向: 中性偏积极
原因:
标题提到"售价过万"，虽然价格较高，但结合华为的品牌定位和高端市场策略，这并不一定是负面信息。
描述中提到"MateBook X Pro系列"和"售价可能接近2..."，显示出华为在高端笔记本市场的布局和产品定位，整体情绪偏向中性偏积极。
总结
从以上分析可以看出，所有新闻的情绪倾向均偏向 积极 或 中性偏积极。主要原因包括：
1. 技术创新：多篇新闻提到华为在汽车、手机、笔记本等领域的技术创新和新产品发布，体现了其在多个领域的领先地位。
2. 市场布局：新闻中多次提及华为的市场扩张计划，如新车上市、招聘规模扩大等，显示出其对未来发展的信心。
3. 品牌影响力：华为作为全球知名科技企业，其每一次动作都受到广泛关注，新闻内容普遍围绕其积极的商业行为展开。
因此，整体情绪倾向可以归纳为 积极。
1767 词