**实验 1：网页信息收集  
姓名：潘韵泽 学号：2023141530019  
完成日期：2025-09-24**

1. **Agent 代码结构**
2. **整体流程：**  
   agent.py → 创建 LLM → 绑定工具 → 构建 ReAct-Agent → 循环交互
3. **关键模块：**  
   **(1) LLM 初始化**

llm = ChatOpenAI(

        model='deepseek-chat',

        api\_key=OPENAI\_API\_KEY,

        base\_url=BASE\_URL,

    )

通过 OpenAI-compatible 接口调用 DeepSeek 提供的 Chat 模型；temperature 默认 0.7，实验未做额外调整以保证结果可复现。

1. **Prompt**

    prompt = """你是一个网页信息收集助手，可以帮助用户:

1. 搜索子域名 - 使用 search\_subdomains(domain)

2. 获取网页信息 - 使用 get\_webpage\_info(url)

3. 分析网页结构 - 使用 analyze\_webpage(url)

硬性规则：

- 任何涉及“获取网页信息/分析网页结构/总结网页内容/提取文本/解析HTML/元数据”的请求，必须调用相应工具完成；不得仅凭已有知识、猜测或模型能力直接生成结果。

- 若因权限或实现缺失无法调用工具，应明确回复“需要工具调用，当前无法完成”。

- 最终回答前请确认“已调用至少一个工具”。"""

明确列出 3 个工具及用途，设定“硬性规则”：凡涉及“获取/分析/提取网页”类请求必须调用工具，通过指令限定 Agent 在最终回答前自检“已调用至少一个工具”。

1. **运行时校验**

used\_tool = any(getattr(m, 'type', None) == 'tool' or getattr(m, 'role', None) == 'tool' for m in messages)

        print("☆" \* 20)

        if not used\_tool and any(kw in user\_input for kw in ["网页信息", "网页", "分析", "提取", "summary", "info", "analyze", "html", "HTML", "meta", "标题", "正文"]):

            print("未检测到工具调用。根据规则，此类请求必须通过工具完成。\n提示：请明确提供 URL，并重试。例如：‘获取网页信息 https://example.com’ 或 ‘分析网页结构 https://example.com’。")

        else:

            print(messages[-1].content if messages else "")

        print("☆" \* 20)

若关键字触发网页类请求且 used\_tool==False，则拒绝回答并提示“需要工具调用”。

1. **HTTP 与网页基本组成**
2. **HTTP 请求-响应报文**  
   请求：请求行（方法/URL/版本）+ 首部（Host、User-Agent…）+ 可选实体；  
   响应：状态行（版本/状态码/原因短语）+ 首部（Content-Type、Server…）+ 实体（HTML 字节流）。
3. **HTML 文档树**

HTML文档结构为：根元素 <html> → <head> + <body>，其中：

<head> 含 <title>、<meta>、<link> 等元数据；

<body> 含语义区段 <header>、<nav>、<main>、<article>、<footer> 等；

节点可嵌套，形成 DOM 树；浏览器通过 CSS 选择器与盒模型完成渲染。

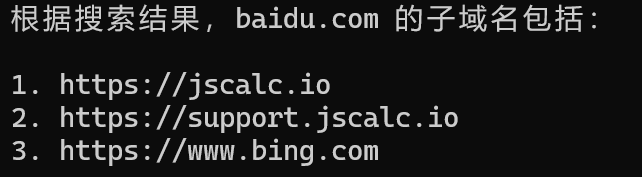
1. **工具原理对照**

**search\_subdomains：**把搜索结果里 <h2> 标签下的链接提取出来 → 只取域名部分，重复去掉，得到子域名清单

**get\_webpage\_info：**先用 requests 取状态码，再用 BeautifulSoup 把 <head> 里的标题、description、meta 标签、规范链接等一次性摘出来

**analyze\_webpage：**把整个 HTML 读进来，用 BeautifulSoup 遍历 DOM 树：统计标签总数、种类数、最大深度；优先找 <main> / <article> 当正文，没有就回退到 <body>；再数一下 h1/h2/h3/p 和链接的数量，剪 200 字摘要回来

1. **Web 工具实现细节**
2. **search\_subdomains**



1. **get\_webpage\_info**

def get\_webpage\_info(url: str) -> str:

    """获取网页基本信息

    Args:

        url (str): 网页URL

    Returns:

        str: JSON 字符串，作为“必须调用工具”的执行证明，占位等待实现。

    """

    #抓取并解析 HTML，返回标题、描述、主要 meta、规范化 URL、HTTP 状态

    # 发送请求并获取 HTML 内容

    html = urlopen(url).read().decode('utf-8')

    soup = BeautifulSoup(html, features="html.parser")

    # 规范化URL

    parsed\_url = urlparse(url)

    normalized\_url = f"{parsed\_url.scheme}://{parsed\_url.netloc}{parsed\_url.path}"

    # 获取HTTP状态码

    response = requests.get(url)

    http\_status = response.status\_code

    # 找到标题

    title = soup.find\_all('title')

    title = title[0].get\_text() if title else "N/A"

    # 找到描述

    description = soup.find('meta', attrs={'name': 'description'})

    description = description['content'] if description else "N/A"

    # 主要meta

    meta\_tags = soup.find\_all('meta')

    meta\_info = {}

    # 收集常规 meta

    for tag in meta\_tags:

        key = tag.get('name') or tag.get('property')

        if key:

            meta\_info[key] = tag.get('content', '')

    # 收集 charset

    charset = None

    for tag in meta\_tags:

        if tag.get('charset'):

            charset = tag['charset']

            break

    if charset:

        meta\_info['charset'] = charset

    # 收集 http-equiv

    for tag in meta\_tags:

        http\_equiv = tag.get('http-equiv')

        if http\_equiv:

            meta\_info[f"http-equiv:{http\_equiv}"] = tag.get('content', '')

    # 收集 canonical

    canonical = soup.find('link', rel=lambda v: v and 'canonical' in v.lower())

    if canonical and canonical.get('href'):

        meta\_info['canonical'] = canonical['href']

    # 例如优先取 description

    description = meta\_info.get('description') or meta\_info.get('og:description') or meta\_info.get('twitter:description', '')

    proof = {

        "status": "success",

        "tool": "get\_webpage\_info",

        "url": url,

        "timestamp": datetime.utcnow().isoformat() + "Z",

        "proof\_id": str(uuid4()),

        "note": "stub-only; students should implement real logic",

        "data": {

            "normalized\_url": normalized\_url,

            "http\_status": http\_status,

            "title": title,

            "description": description,

            "meta": meta\_info

        }

    }

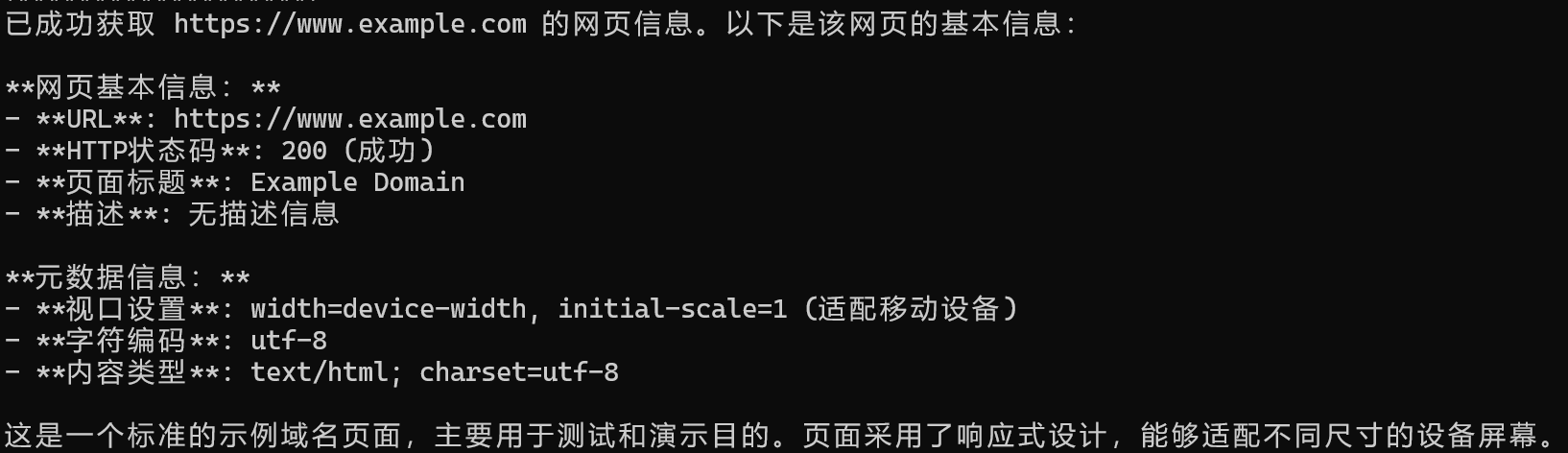
    return json.dumps(proof, ensure\_ascii=False)

先以 requests.head() 获取状态码，再 requests.get() 拉取 HTML，防止大文件直接下载

用 soup.find('title') 及 soup.find('meta', attrs={'name':'description'}) 提取基础元数据；

遍历全部 <meta>，按 name/property/http-equiv/charset 分键存储；

规范化 URL：拼接 scheme+netloc+path 并去除片段；



1. **analyze\_webpage**

"""分析网页结构

    Args:

        url (str): 网页URL

    Returns:

        str: JSON 字符串，作为“必须调用工具”的执行证明，占位等待实现。

    """

    # 输出 DOM 结构摘要、主内容区定位思路、重要节点与链接统计

    # DOM结构分析

    html = urlopen(url).read().decode('utf-8')

    soup = BeautifulSoup(html, features="lmxl")

    # 摘要

    dom\_summary = {

        "total\_tags": len(soup.find\_all()),

        "unique\_tags": len(set([tag.name for tag in soup.find\_all()])),

        "max\_depth": max([len(list(tag.parents)) for tag in soup.find\_all()]),

    }

    # 主内容区定位

    main\_content = soup.find('main') or soup.find('article') or soup.find('body')

    main\_content\_strategy = "Look for <main> or <article> tags, fallback to <body>."

    # 主要内容区文本

    main\_text = main\_content.get\_text(separator=' ', strip=True) if main\_content else ""

    # 统计重要节点与链接

    important\_nodes = soup.find\_all(['h1', 'h2', 'h3', 'p'])

    links = soup.find\_all('a', href=True)

    important\_node\_count = len(important\_nodes)

    link\_count = len(links)

    proof = {

        "status": "success",

        "tool": "analyze\_webpage",

        "url": url,

        "timestamp": datetime.utcnow().isoformat() + "Z",

        "proof\_id": str(uuid4()),

        "note": "stub-only; students should implement real logic",

        "data": {

            "dom\_summary": dom\_summary,

            "main\_content\_strategy": main\_content\_strategy,

            "important\_node\_count": important\_node\_count,

            "link\_count": link\_count,

            "main\_text\_snippet": main\_text[:200]  # 前200字符

        }

    }

    return json.dumps(proof, ensure\_ascii=False)

计算网站中的主要指标：

total\_tags = len(soup.find\_all(True))  
– unique\_tags = len({t.name for t in soup.find\_all(True)})  
– max\_depth = max(len(list(t.parents)) for t in soup.find\_all(True))

主内容区策略：优先 <main> → 次选 <article> → 回退 <body>,并统计：  
– important\_nodes = soup.find\_all(['h1','h2','h3','p'])  
– links = soup.find\_all('a', href=True

截取主内容前 200 字符作为 main\_text\_snippet，防止 JSON 过大；

