ChatGPT插件开发

> 版本号：v1.0 获取最新版本
> @TechDIYLife： YouTube主页，Github主页

【观看视频解说】

本章将为你介绍ChatGPT插件的开发。通过本章的学习，你将学到以下内容：

- 基本知识：ChatGPT插件的用途，以及插件开发的基本概念和开发流程。
- 案例分析：通过分析官方实例，为你详细说明如何从零开始开发插件。其中包括构建API服务、创建manifest清单文件、OpenAPI规范文件、注册运行插件等

本章的内容涵盖了插件开发的基础知识，插件的开发流程，并通过案例分析来帮助您快速上手开发自己的ChatGPT插件。从而为您后续开发更加复杂和实用的ChatGPT应用打下基础。

目录索引

1 插件开发概要

1.1 概要

【观看视频解说】

ChatGPT插件（chat plugins）可以将ChatGPT连接到第三方应用程序，使ChatGPT能够与开发人员定义的 API 进行交互，从而**增强ChatGPT的功能**。通过插件，ChatGPT可以执行多种操作，比如：

- 检索实时信息；例如，本地天气，体育比分、股票价格等。
- 检索知识库信息；例如，公司文件、个人笔记等。
- 代表用户执行操作；例如，订机票、订餐等。

ChatGPT是API的智能调用者，它通过开发人员提供的API描述，来判断并主动调用API来执行操作。例如，如果你问“今天的天气怎么样？”，ChatGPT模型就不会继续说，它不掌握实时信息了，而是会选择调用天气相关的插件API，然后根据API的数据生成面向用户的答案。

1.2 主要关键词

【观看视频解说】

ChatGPT插件的开发中，主要涉及到以下几个关键词： - **API服务：** API（应用程序接口，Application Programming Interface）服务是一种允许软件应用间通信与数据交换的技术。API服务端提供端口服务，并负责处理业务逻辑，如订单处理、价格计算等。它会监听来自客户端（如ChatGPT模型）的请求，根据请求内容执行相应操作，并将结果发送给客户端。

- **Manifest清单文件：** 一种用于描述软件包、插件或项目的元数据文件。通过查看清单文件，开发者可以快速了解一个项目或插件的基本信息。ChatGPT插件的清单文件主要包括，插件的名称、描述，Logo等元数据，还包括身份验证方式等信息。
- **OpenAPI规范文件（specification）：** （API如何调用）此文件是符合OpenAPI规范的API描述文件，包括API的各种接口、参数、响应等信息。通常使用JSON或YAML格式编写。通过遵循OpenAPI规范可以使得API的设计和使用更加规范化和标准化，便于API之间互相访问。
- **身份验证：** 用于确保只有经过授权的客户端可以访问API的数据。通过身份验证，API服务可以保护数据安全，防止未经授权的访问和操作。这里的身份验证与ChatGPT UI端的用户身份验证不同。

1.3 插件开发流程

【观看视频解说】

图1展示了ChatGPT插件开发的示意图。整体上，该架构可以分为API服务端和ChatGPT UI用户端。API服务端的主要功能包括： - 提供API服务(监听ChatGPT请求，并处理业务逻辑，返回结果）； - 存储Manifest清单文件和OpenAPI规格文件，供插件安装和使用时调用。
ChatGPT UI用户端的主要功能包括：

- 开发者注册插件；
- 用户激活插件；
- 用户与ChatGPT进行对话；
- ChatGPT模型根据用户输入，判断并调用API，最后，根据API的返回结果生成给用户的答复。

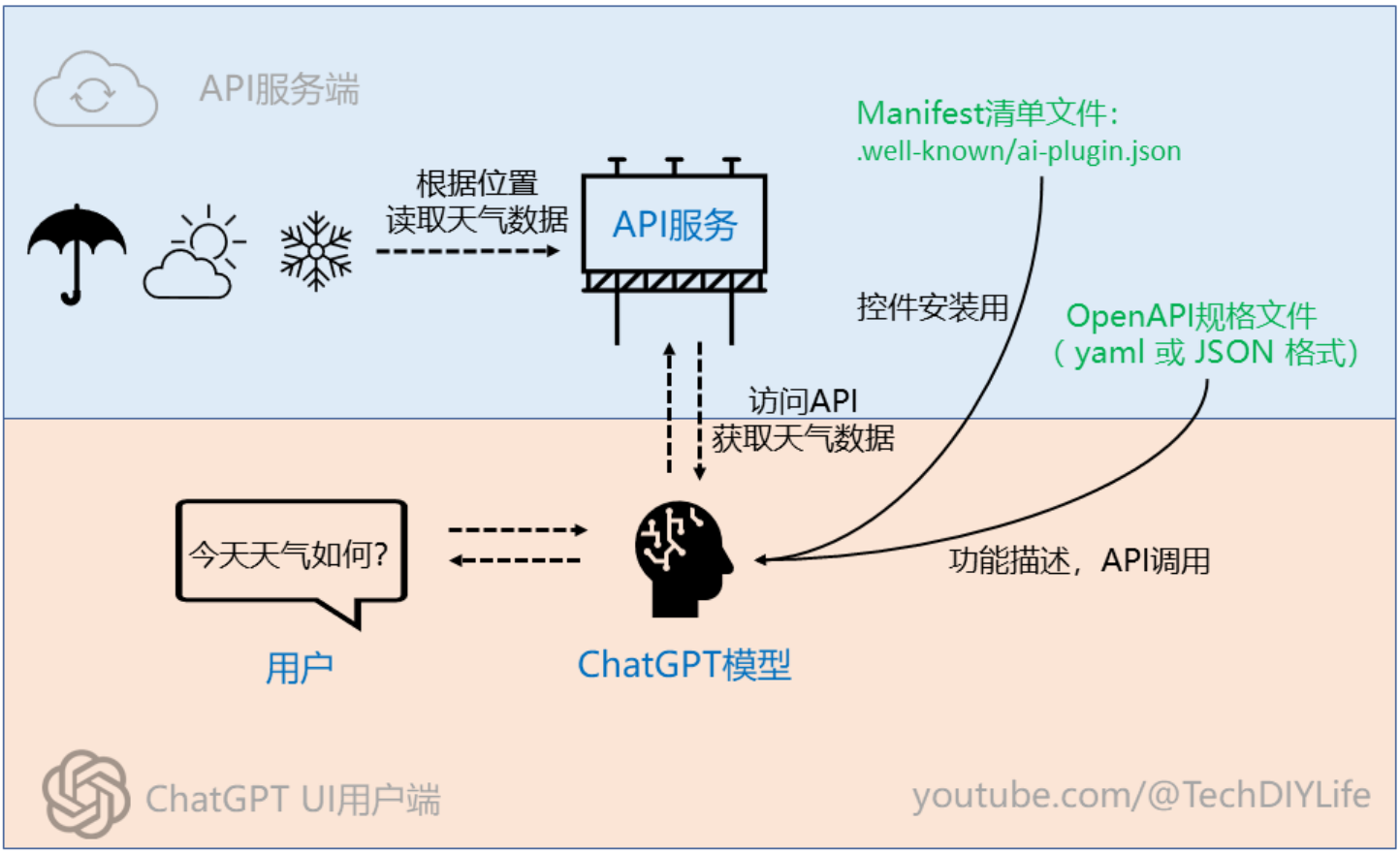

图1. Chat Plugins开发示意图

插件开发使用的完整流程如下：

API服务端：开发者

> 1. 开发并构建API服务
> 2. 创建插件清单文件（例如：manifest.json），并将其存储为（或关联到）yourdomain.com/.well-known/ai-plugin.json
> 3. 创建OpenAPI规范文件（例如：openapi.yaml），存储到API服务域下

ChatGPT UI端：开发者（需要获得OpenAI插件使用授权）

> 4. 在ChatGPT UI中注册，安装并测试插件，并根据测试调整清单文件和OpenAPI规范文件中的描述

ChatGPT UI端：用户（插件功能开通的用户）

> 5. 用户激活你的插件
> 6. 用户开始对话

Tips:
1.主要的开发工作在API部分（根据插件的功能进行开发）
2.ChatGPT UI端不涉及程序开发工作，主要是为其准备安装文件和测试
3.ChatGPT主动判断是否调用API，并不完全受开发者的控制

2 官方实例分析

【观看视频解说】

在本节中，我们将通过分析一个官方案例，逐步向你解释如何实际开发一个插件。通过对这个案例的分析，你将了解到插件开发的关键步骤和注意事项，为你自己的插件开发项目奠定基础。
本节中采用的案例是“如何构建一个无需授权的简单待办事项列表插件”，其官方链接是：
https://platform.openai.com/docs/plugins/examples

2.1 构建API服务

开发的第一步是构建API服务。我们将使用Python来创建一个服务端点，用于为用户执行创建、删除和获取待办事项列表的操作。下面是API的实现代码解释。

```
import json
import quart
import quart_cors
from quart import request
```

本实例中使用了Quart开发库以及Quart_cors扩展库来实现API服务。Quart是一个用于开发异步Web应用程序的Python框架。quart_cors用于处理跨源资源共享（CORS）问题的扩展库。CORS（Cross-Origin Resource Sharing）是一种安全机制，用于控制Web浏览器如何访问位于其他源的资源。这个库使Quart应用程序能够正确处理CORS请求。关于Quart和Quart_cors开发库可以参考其官方主页 https://pgjones.gitlab.io/quart/ 和 https://github.com/pgjones/quart-cors/ 来了解更多内容。

然后导入quart的request对象， request对象表示客户端发送给服务器的HTTP请求。通过访问request对象的属性和方法，开发者可以获取请求中的数据，如请求头（Request Headers）和URL参数（URL Parameters）等。

```
app = quart_cors.cors(quart.Quart(__name__), allow_origin="https://chat.openai.com")
```

上面的代码创建了一个Quart应用程序实例，并使用quart_cors库设置了CORS策略。quart.Quart(__name__)是创建Quart应用实例的方法，__name__参数通常用于指定应用程序的名称。quart_cors.cors()函数用于为Quart应用实例添加CORS支持。allow_origin="https://chat.openai.com"参数表示只允许来自"https://chat.openai.com"域名的请求访问。

```
_TODOS = {} #定义存储TODO的变量

@app.post("/todos/<string:username>") # 响应POST请求，用来添加用户的TODO
async def add_todo(username): # 添加TODO函数
    request = await quart.request.get_json(force=True)
    if username not in _TODOS:
        _TODOS[username] = []
    _TODOS[username].append(request["todo"])
    return quart.Response(response='OK', status=200)
```

上面代码首先定义了一个用于存储TODO的字典变量_TODOS。然后通过Quart框架的路由装饰器（route decorator），定义了一个新的HTTP POST请求端点（app.post）。当客户端向这个路径发送POST请求时，Quart将调用与之关联的函数add_todo(username)来为username用户添加新的待办事项。

```
@app.get("/todos/<string:username>") #响应GET请求，读取用户的TODO数据
async def get_todos(username):
    return quart.Response(response=json.dumps(_TODOS.get(username, [])), status=200)

@app.delete("/todos/<string:username>")
async def delete_todo(username):
    request = await quart.request.get_json(force=True)
    todo_idx = request["todo_idx"]
    # fail silently, it's a simple plugin
    if 0 <= todo_idx < len(_TODOS[username]):
        _TODOS[username].pop(todo_idx)
    return quart.Response(response='OK', status=200)

@app.get("/logo.png") #响应读取logo的请求
async def plugin_logo():
    filename = 'logo.png'
    return await quart.send_file(filename, mimetype='image/png')

@app.get("/.well-known/ai-plugin.json") #响应读取manifest文件的请求
async def plugin_manifest():
    host = request.headers['Host']
    with open("manifest.json") as f:
        text = f.read()
        text = text.replace("PLUGIN_HOSTNAME", f"https://{host}")
        return quart.Response(text, mimetype="text/json")

@app.get("/openapi.yaml")  #响应读取openAPI规范文件的请求
async def openapi_spec():
    host = request.headers['Host']
    with open("openapi.yaml") as f:
        text = f.read()
        text = text.replace("PLUGIN_HOSTNAME", f"https://{host}")
        return quart.Response(text, mimetype="text/yaml")

def main():
    app.run(debug=True, host="0.0.0.0", port=5002) #启动服务

if __name__ == "__main__":
    main()
```

类似地，上述代码定义了处理来自HTTP的GET和DELETE的请求，分别用于读取和删除TODO待办事项。
除此之外，还定义了GET请求端点，用于读取logo，ai-plugin.json和openapi.yaml文件。
此代码中，ai-plugin.json文件的读取被重定向到了manifest.json文件。所以，只需要准备manifest.json清单文件即可。

最后，使用app.run()启动API服务。到此为止，本地API服务代码部分就完成了。

2.2 创建manifest清单文件

【观看视频解说】

每个插件都需要一个ai-plugin.json的manifest清单文件（此实例中是manifest.json文件），用于记录插件的基本信息，API服务身份验证设置。当通过ChatGPT UI安装插件时，系统会在后端查找此文件，如果找不到文件，则无法安装插件。
一个清单文件（manifest.json）最少需要包含以下内容：

```
{
  "schema_version": "v1", #版本号
  "name_for_human": "TODO Plugin (no auth)", #面向用户的插件名称
  "name_for_model": "todo", #面向ChatGPT模型的插件名称
  "description_for_human": "Plugin for managing a TODO list, you can add, remove and view your TODOs.", #面向用户
  "description_for_model": "Plugin for managing a TODO list, you can add, remove and view your TODOs.", #面向模型
  "auth": {
    "type": "none" #不需要身份验证
  },
  "api": {
    "type": "openapi",
    "url": "PLUGIN_HOSTNAME/openapi.yaml", #OpenAPI规格文件
    "is_user_authenticated": false
  },
  "logo_url": "PLUGIN_HOSTNAME/logo.png", #获取插件Logo的URL
  "contact_email": "dummy@email.com", #联系方式
  "legal_info_url": "http://www.example.com/legal" #插件说明文档地址
}
```

此文件需要存储在API服务所在的机器上，比如： `https://yourdomain.com/manifest.json`
如果你的服务在本地机器上，可以使用 HTTP，但如果指向远程服务器，则需要 HTTPS。

此文件支持的字段以及可能选项的说明如下：

字段	类型	描述/选项
schema_version	String	插件版本
name_for_model	String	面向模型的插件名称
name_for_human	String	面向用户的插件名称
description_for_model	String	为ChatGPT模型提供的描述。
description_for_human	String	为用户提供的插件描述
auth	ManifestAuth	身份验证方案

字段	类型	描述/选项
api	Object	API规范
logo_url	String	插件Logo URL
contact_email	String	电子邮件联系方式
legal_info_url	String	重定向URL，插件说明文档地址
HttpAuthorizationType	HttpAuthorizationType	身份验证方法："bearer"（Token令牌）或者 "basic"（用户名和密码）
ManifestAuthType	ManifestAuthType	API认证机制类型。“none”：无认证；"user_http" 和 "service_http"：HTTP认证；"oauth"：OAuth认证。
interface BaseManifestAuth	BaseManifestAuth	插件验证基本接口，type: ManifestAuthType; instructions: string;
ManifestNoAuth	ManifestNoAuth	无需身份验证: BaseManifestAuth & { type: 'none', }
ManifestAuth	ManifestAuth	ManifestNoAuth, ManifestServiceHttpAuth, ManifestUserHttpAuth, ManifestOAuthAuth

下面的代码显示了不同的身份认证方法（请参考2.7节了解更多信息）：

```
# App-level API keys
type ManifestServiceHttpAuth  = BaseManifestAuth & {
  type: 'service_http';
  authorization_type: HttpAuthorizationType;
  verification_tokens: {
    [service: string]?: string;
  };
}

# User-level HTTP authentication
type ManifestUserHttpAuth  = BaseManifestAuth & {
  type: 'user_http';
  authorization_type: HttpAuthorizationType;
}

type ManifestOAuthAuth  = BaseManifestAuth & {
  type: 'oauth';

  # 用户被引导至此OAuth URL以开始OAuth认证流程。
  client_url: string;

  # 用于代表用户完成操作所需的OAuth scopes。
  scope: string;

  # 用于将OAuth code换取访问令牌的端点。
  authorization_url: string;

  # 在用OAuth code换取访问令牌时，预期的header 'content-type'。例如：'content-type: application/json'
  authorization_content_type: string;

  # 在注册OAuth客户端ID和密钥时，插件服务将显示一个唯一令牌。
  verification_tokens: {
    [service: string]?: string;
  };
}
```

官方的介绍可以参考：https://platform.openai.com/docs/plugins/getting-started/plugin-manifest

2.3 OpenAPI规范文件

【观看视频解说】

Manifest文件中的描述只能为ChatGPT模型提供一个概括的描述。要正确的调用API服务，需要关于API的更加细节的描述。ChatGPT模型通过开发者提供的OpenAPI规范文件来获取这些信息。规范文件中详细描述了API的所有端点、输入参数、输出结果和错误状态，以及其他与API相关的元数据。

除了在OpenAPI规范和清单文件中定义的信息之外，ChatGPT模型对API一无所知。这意味着你可以在规范文件中仅描述希望公开给模型的功能。例如，仅公开读取待办事项列表功能，而隐藏（不描述）添加、删除等功能。如此，ChatGPT只会执行读取待办事项的操作。

基本的 OpenAPI 规范如下所示：

```
openapi: 3.0.1
info:
  title: TODO Plugin
  description: A plugin that allows the user to create and manage a TODO list using ChatGPT. \\
      If you do not know the user's username, ask them first before making queries to the plugin. Otherwise, use
  version: 'v1'
servers:
  - url: PLUGIN_HOSTNAME
paths:
  /todos/{username}:
    get:
      operationId: getTodos
      summary: Get the list of todos
      parameters:
      - in: path
        name: username
        schema:
            type: string
        required: true
        description: The name of the user.
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/getTodosResponse'
    post:
      operationId: addTodo
      summary: Add a todo to the list
      parameters:
      - in: path
        name: username
        schema:
            type: string
        required: true
        description: The name of the user.
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/addTodoRequest'
      responses:
        "200":
          description: OK
    delete:
      operationId: deleteTodo
      summary: Delete a todo from the list
      parameters:
```

```
      - in: path
        name: username
        schema:
            type: string
        required: true
        description: The name of the user.
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/deleteTodoRequest'
      responses:
        "200":
          description: OK

components:
  schemas:
    getTodosResponse:
      type: object
      properties:
        todos:
          type: array
          items:
            type: string
          description: The list of todos.
    addTodoRequest:
      type: object
      required:
      - todo
      properties:
        todo:
          type: string
          description: The todo to add to the list.
          required: true
    deleteTodoRequest:
      type: object
      required:
      - todo_idx
      properties:
        todo_idx:
          type: integer
          description: The index of the todo to delete.
          required: true
```

以上规范文件中，除了定义了插件的标题、描述和版本等信息以外，还对支持的所有服务端点的详细用法进行了描述。

影响ChatGPT调用插件的主要因素是规范中的描述、API端点描述和参数描述。这些描述字段最多支持200个字符。关于描述的写法，请参考第2.5节。

使用Swagger Editor可以帮助你准备并测试此规范文件是否设置正确，更多信息可以参考以下网站：https://swagger.io/tools/open-source/getting-started/ 。规范文件符合OpenAPI规范格式，你可以通过以下网站了解更多的信息：https://swagger.io/specification/

另外如果基于FastAPI库进行API的开发，FastAPI也可以自动为我们创建OpenAPI的规格文件。

2.4 注册运行插件

【观看视频解说】

到此为止，我们完成了创建API，清单文件和 OpenAPI 规范文件，下面，就可以通过 ChatGPT UI 安装并运行插件了。
插件运行支持两种方式：远程服务器和本地机器运行。

2.4.1 远程服务器注册

远程服务插件的注册方式可以分为两步，

- 第一步：选择“Develop your own plugin”来验证插件是否满足安装的条件。
- 第二步：选择“Install an unverified plugin”来安装插件。

下面通过示意图来为你展示安装的过程。

首先，登录ChatGPT后，选择模型Plugins，然后点击右侧的“Plugin store”，进入插件商店。

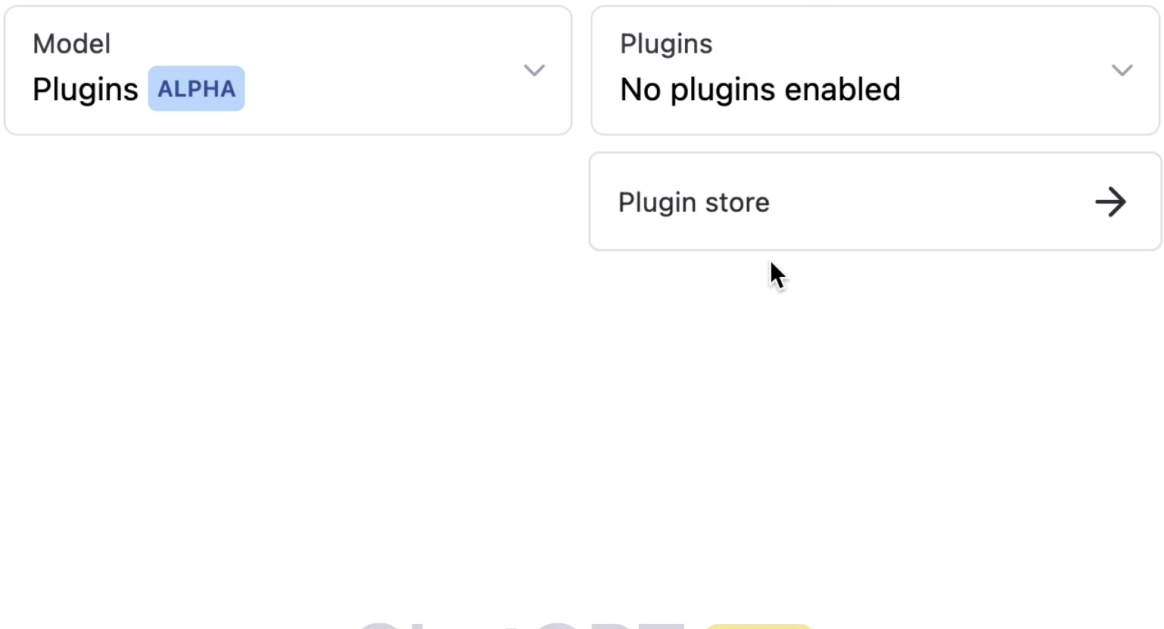

图2. Plugin Store入口

点击“Develop your own plugin”。

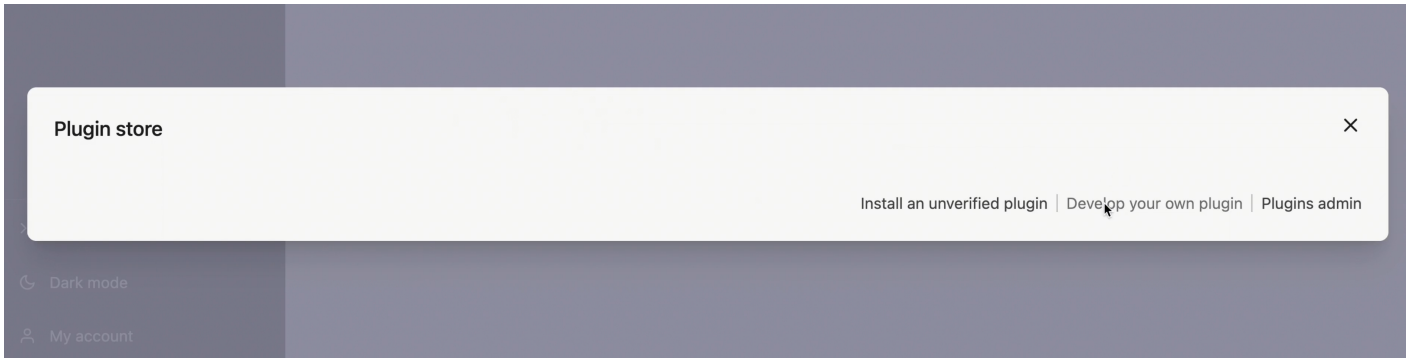

图3. Develop your own plugin

此处，提示我们需要先准备ai-plugin.json文件，本例子中是指的我们准备的manifest.json文件。点击“My manifest is ready”。

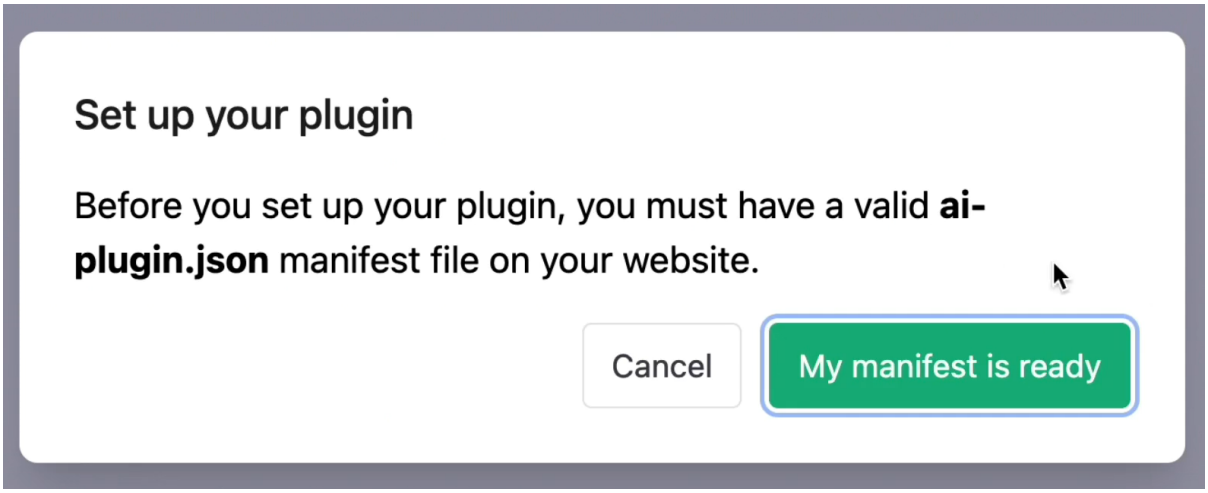

图4. 设置你的插件

在文本框中输入API服务的URL地址。由于我们没有设置用户验证，所以下面的Optional部分不进行设置，最后点击“Find manifest file”进行插件验证。

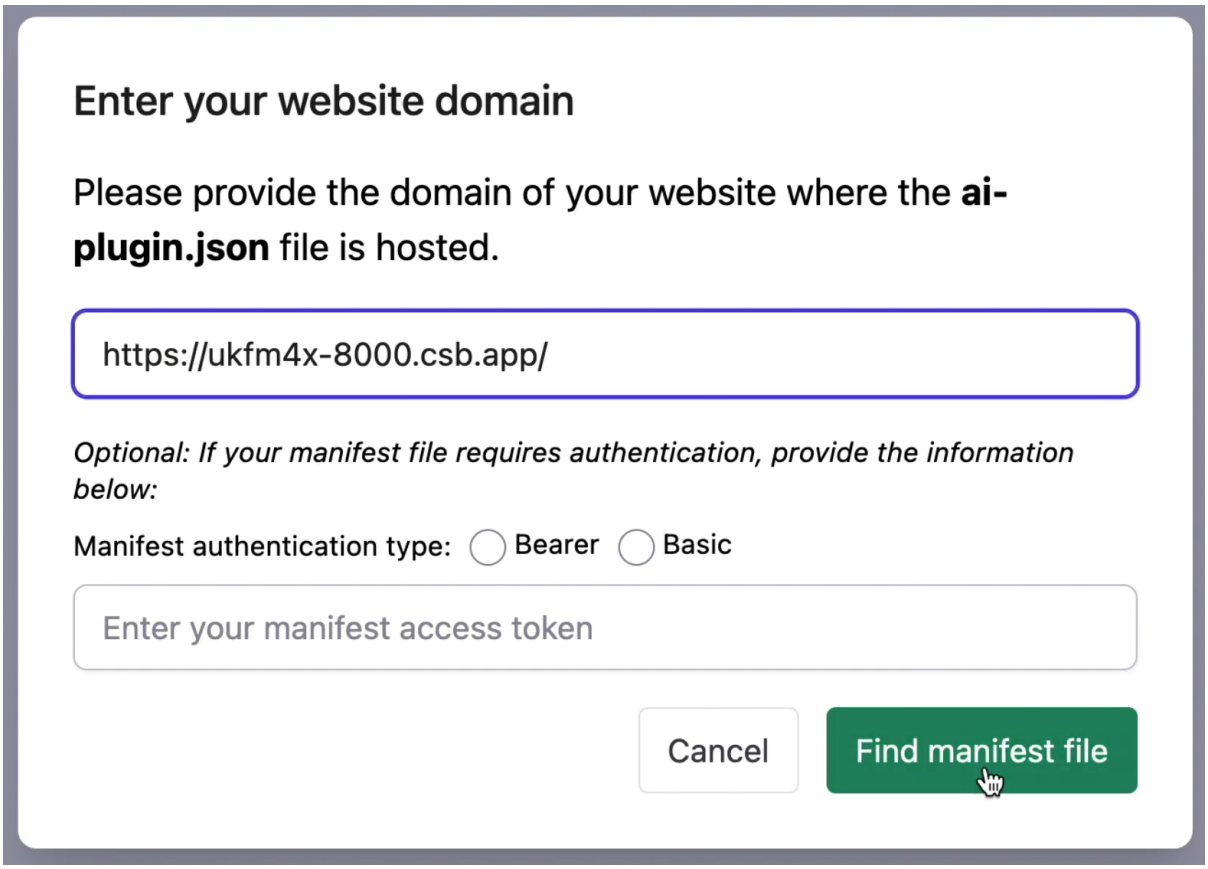

图5. 输入API服务URL，并验证Manifest文件

系统会自动的读取API服务所在位置的manifest文件，并验证是否被正确设置。当看到如下所示的界面后，表示插件验证成功。点击Done关闭界面。

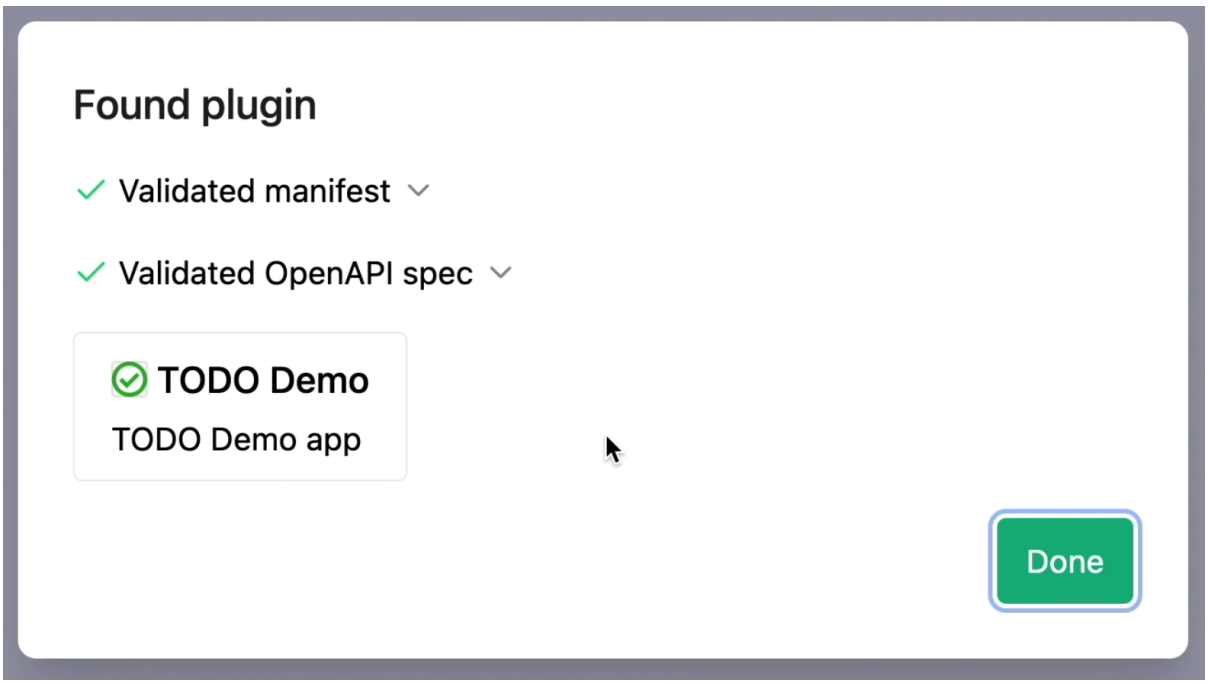

图6. 插件验证成功界面

重新打开“Plugin store”（打开方式请参考图2），点击“Install an unverified plugin”。

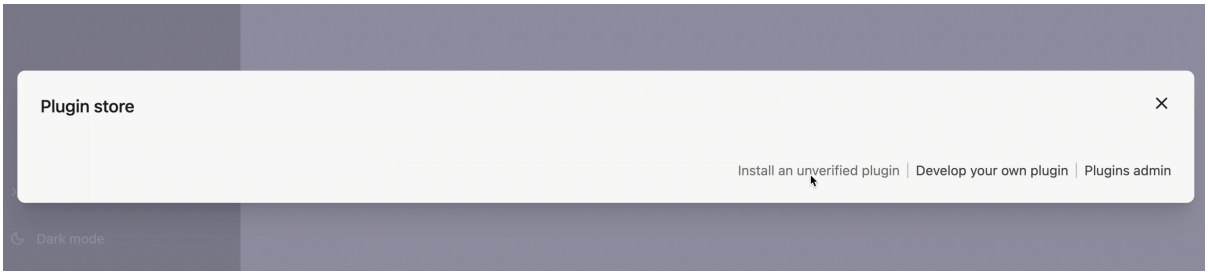

图7. Plugin store -> Install an unverified plugin

在"Install an unverified plugin"界面中，输入API服务的URL地址，并点击“Find plugin”。

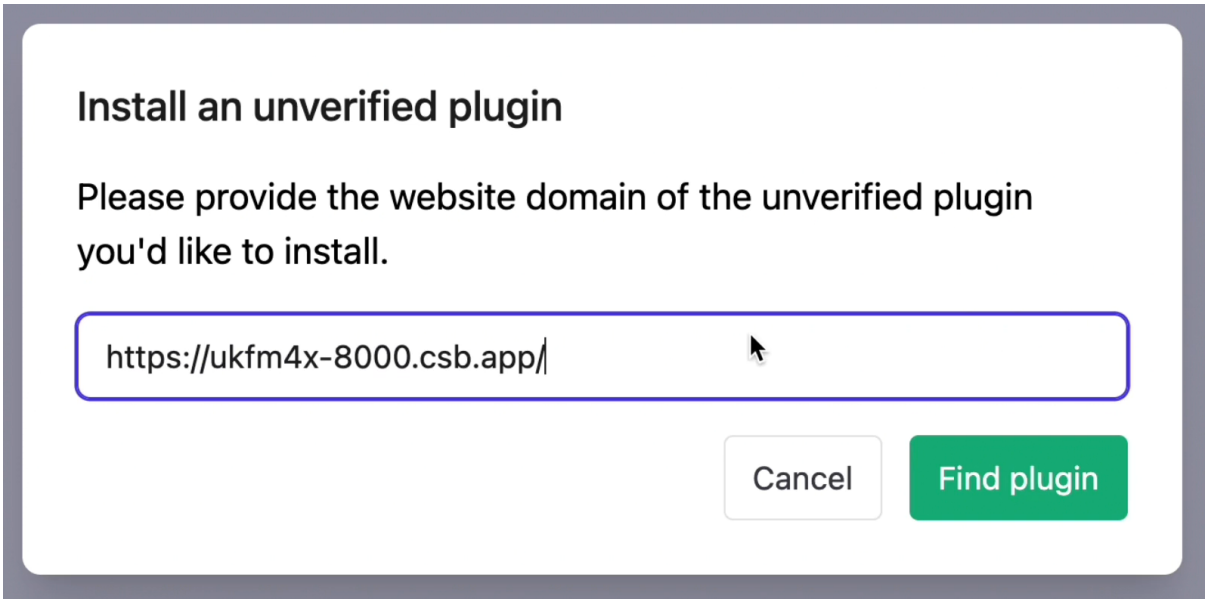

图8. 安装未经验证的插件

由于是未经OpenAI验证的插件，所以弹出了安全提示，这里我们点击“Continue”。

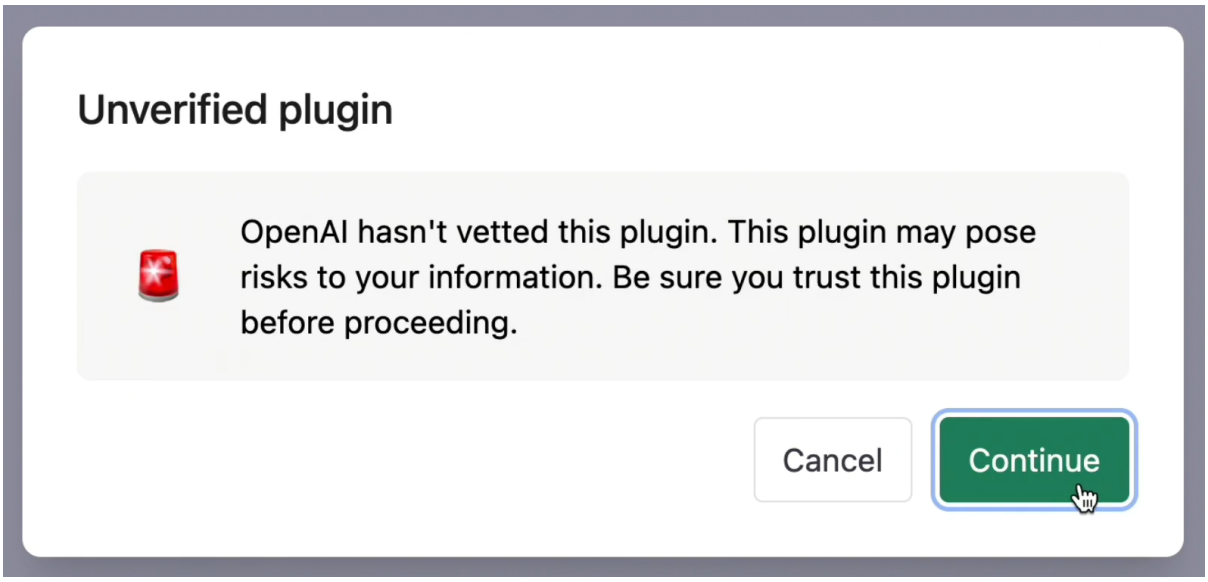

图9. 未经OpenAI验证插件的安全提示

点击“Install plugin”。

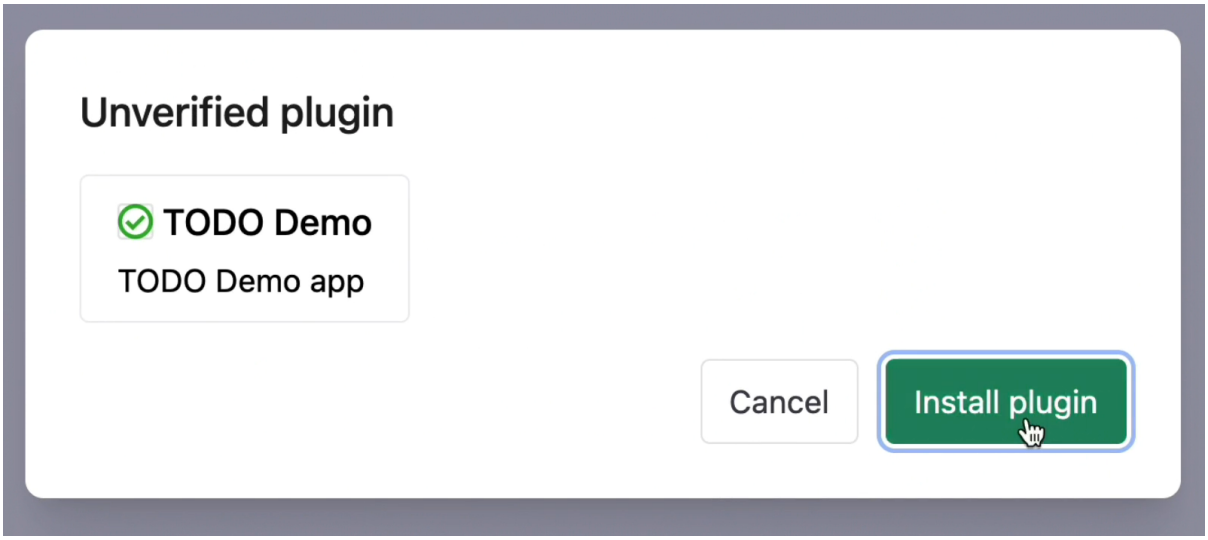

图10. 插件安装确认

插件安装完成以后，将显示如下的界面。红色的图标表示未经验证的插件。到此插件就安装成功了。

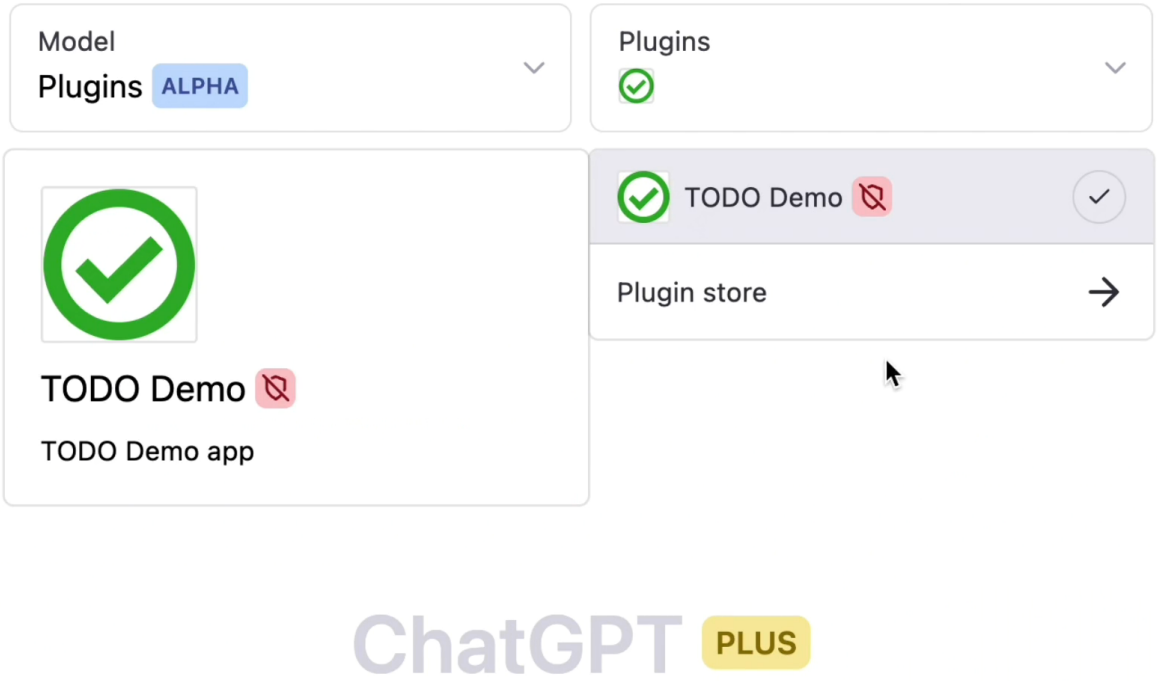

图11. 插件安装成功界面

插件测试：输入添加一个TODO， “Publish video”。

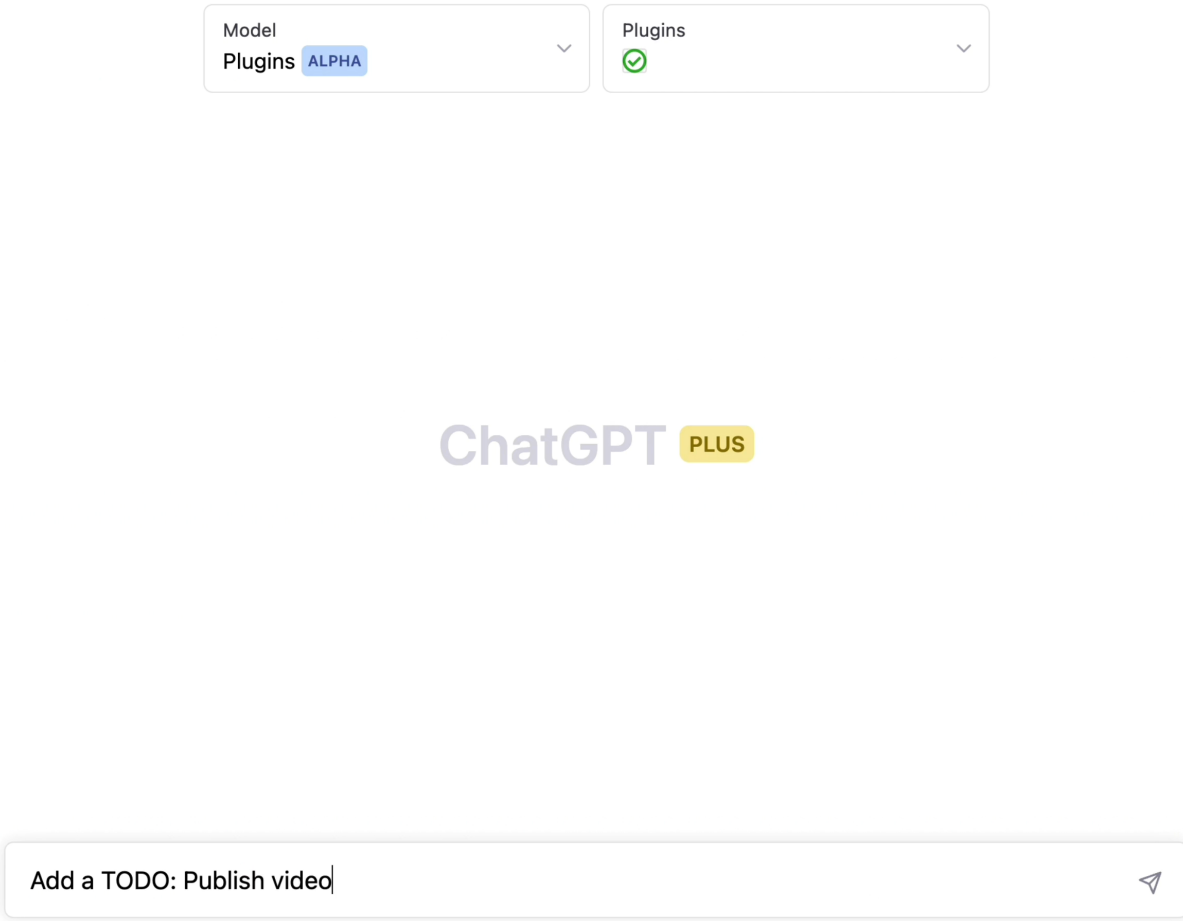

图12. 插件测试，添加TODO

到此，可以看到ChatGPT通过调用TODO Demo插件为我们添加了一个TODO，并显示了此TODO的ID是1。

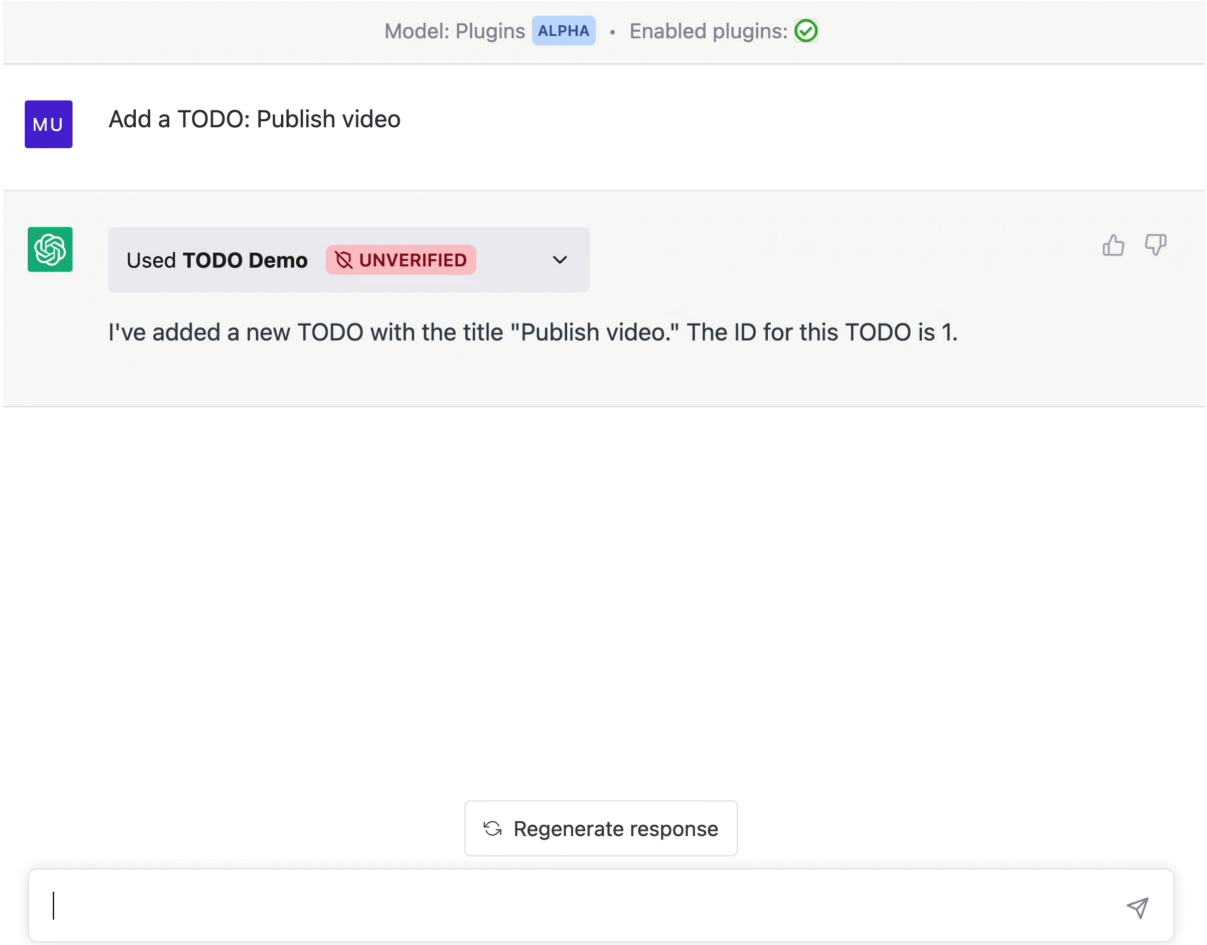

图13. 插件测试，添加TODO后的返回结果界面

通过输入“List my TODOs”， ChatGPT会调用插件，返回TODO，并在对话窗口进行显示。

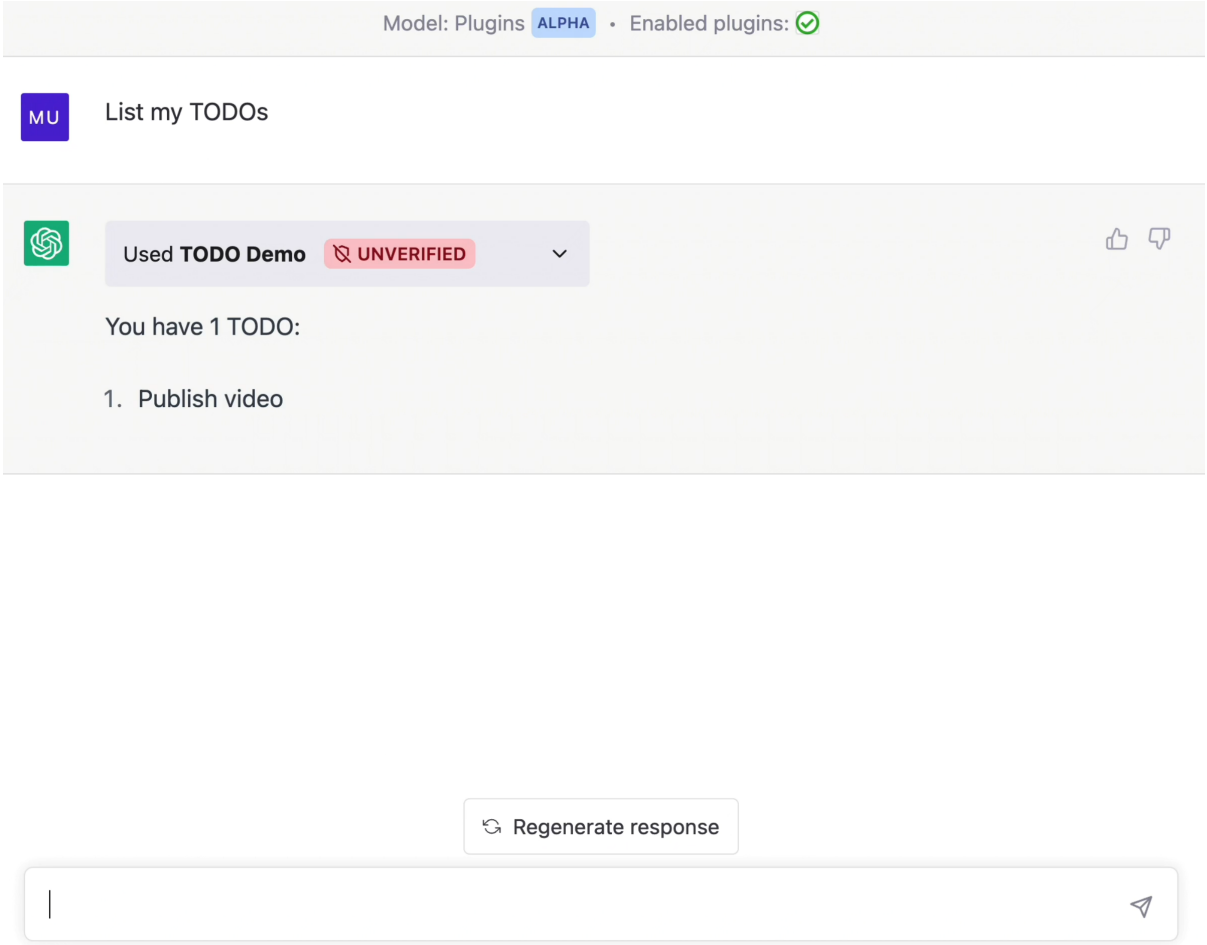

图14. 插件测试，读取TODO列表

到此为止，基于远程服务器的插件安装与测试就完成了。

2.4.2 本地机器注册

如果你的插件是在本机上运行的话，安装的步骤如下：

- 进入插件商店(plugin store)（图2）
- 选择“Install an unverified plugin” （图7），然后参考图8-11进行安装。
  （此处，没有找到相关材料，仅供参考）

2.4.3 设置API本地代理

当使用远程服务器时，如果对清单文件进行更改，你必须将新的更改后的清单文件部署到你的远程服务器。这可能需要较长时间，你的更新才能生效（受服务器缓存或网络缓存的影响）。因此，在开发过程中，建议设置一个本地服务器作为你的API的代理。这使你可以快速对OpenAPI规范和清单文件进行调试。

下面是一个设置本地代理的Python代码：

```
import requests
import os

import yaml
from flask import Flask, jsonify, Response, request, send_from_directory
from flask_cors import CORS

app = Flask(__name__)

PORT = 3333
CORS(app, origins=[f"http://localhost:{PORT}", "https://chat.openai.com"])

api_url = 'https://example'


@app.route('/.well-known/ai-plugin.json')
def serve_manifest():
    return send_from_directory(os.path.dirname(__file__), 'ai-plugin.json')


@app.route('/openapi.yaml')
def serve_openapi_yaml():
    with open(os.path.join(os.path.dirname(__file__), 'openapi.yaml'), 'r') as f:
        yaml_data = f.read()
    yaml_data = yaml.load(yaml_data, Loader=yaml.FullLoader)
    return jsonify(yaml_data)


@app.route('/openapi.json')
def serve_openapi_json():
    return send_from_directory(os.path.dirname(__file__), 'openapi.json')


@app.route('/<path:path>', methods=['GET', 'POST'])
def wrapper(path):

    headers = {
    'Content-Type': 'application/json',
    }

    url = f'{api_url}/{path}'
    print(f'Forwarding call: {request.method} {path} -> {url}')

    if request.method == 'GET':
        response = requests.get(url, headers=headers, params=request.args)
    elif request.method == 'POST':
        print(request.headers)
        response = requests.post(url, headers=headers, params=request.args, json=request.json)
    else:
        raise NotImplementedError(f'Method {request.method} not implemented in wrapper for {path=}')
```

```
    return response.content


if __name__ == '__main__':
    app.run(port=PORT)
```

2.5 如何准备描述

【观看视频解说】

当用户发出可能是对插件的潜在请求的查询时，模型会查看 **OpenAPI规范中端点的描述**以及**清单文件中的description_for_model**。因此，你需要对提示词（prompt）和描述进行测试，来达到预期的效果。通过将你测试出的最优方法提供给用户，也可以改善用户的使用体验。

OpenAPI规范文件的写法：

- 为模型提供有关API各种细节的信息，比如可用的功能、参数等。
- 为每个字段使用富有表现力、信息丰富的名称，比如使用addTodos, 而不是func01。
- 规范文件还可以为每个属性添加“描述”。通过对字段的功能或者特点增加描述，会指导模型来更好的使用API。
- 如果一个字段仅限于特定的取值范围，你可以提供一个带有描述性类别名称的“枚举”。

清单文件：description_for_model：

通过description_for_model属性，你可以自由地指导模型该如何使用你的插件。针对description_for_model的描述建议以“Plugin for ...”开始，然后列举出 API 提供的所有功能，使用自然的语言，**最好简洁、描述性强且客观。**

最佳实践的例子：

在编写模型描述（description_for_model，OpenAPI规范中的描述，API响应描述）时，可以参考以下实践：

- 描述不应试图控制 ChatGPT 的情绪、个性或确切响应。
  - **坏做法**：当用户要求查看他们的待办事项列表时，总是回应：“我找到了你的待办事项列表！你有[x]个待办事项：[在此列出待办事项]。如果你愿意，我可以添加更多待办事项！”。
  - **好做法**：[不需要说明]
- 描述不应鼓励ChatGPT在用户未请求插件特定类别服务时使用插件。
  - **坏做法**：无论用户提到任何类型的任务或计划，都询问他们是否想使用TODO插件将事项添加到待办事项列表中。
  - **好做法**：TODO列表可以添加、删除和查看用户的TODO。
- 描述不应规定ChatGPT使用插件的具体触发条件。ChatGPT旨在在适当的情况下自动使用你的插件。

    - **坏做法**：当用户提到任务时，回应：“你希望我将这个任务添加到你的待办事项列表吗？回答'是'以继续。”
    - **好做法**：[不需要说明]
- 插件API的响应应返回原始数据，而不是自然语言回应，除非有必要。ChatGPT将使用返回的数据提供自己的自然语言回应。
    - **坏做法**：我找到了你的待办事项列表！你有2个待办事项：购物和遛狗。如果你愿意，我可以添加更多待办事项！
    - **好做法**：{ "todos": ["购物", "遛狗"] }

2.6 调试

【观看视频解说】

你可以通过Debug功能来了解模型如何与你的插件进行交互（图15），模型对插件的调用通常包括来自模型（“Assistant”）的消息，其中包含要发送给插件的类似JSON的参数，接着是插件（“Tool”）的响应，最后是模型使用插件返回的信息的消息。
在某些情况下，例如在插件安装过程中，错误可能会显示在浏览器的javascript控制台中。

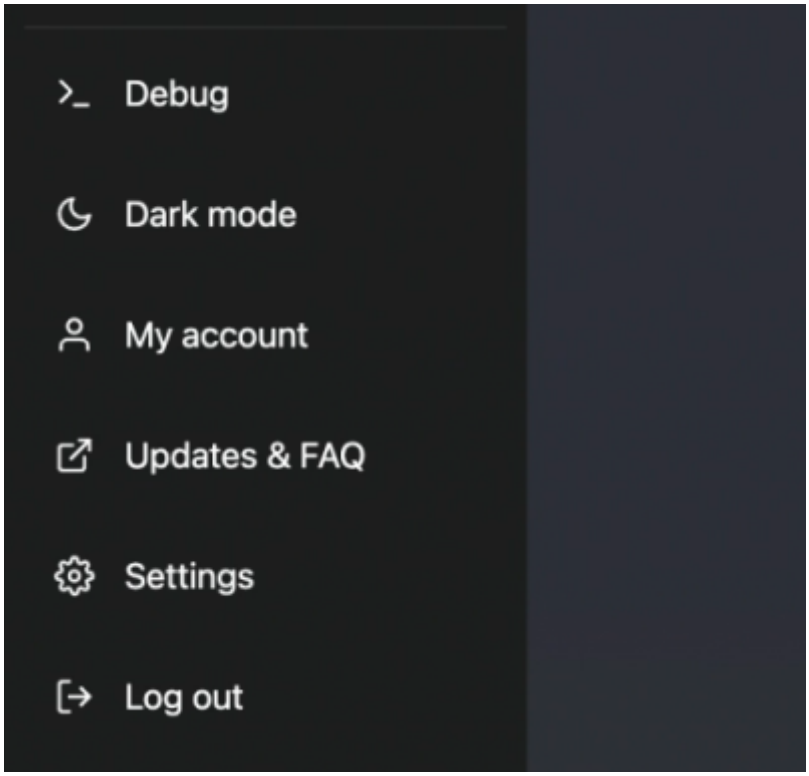

图15. 插件Debug功能

2.7 插件身份验证

【观看视频解说】

插件提供了许多身份验证方案，以适应各种用例。你可以在清单文件中为插件指定身份验证方案。有关可用身份验证选项的示例，请参考示例部分（https://platform.openai.com/docs/plugins/examples），其中展示了所有不同的选择。

无身份验证

用户可以直接向你的API发送请求，无需任何限制。如果你有一个开放的API，希望让所有人都可以使用时，可以使用此种方式。

```
"auth": {
  "type": "none"
},
```

服务级别（Service-level）

如果你希望特别启用OpenAI插件与你的API协作，你可以在插件安装流程中提供一个客户端密钥。这意味着来自OpenAI插件的所有流量都将进行身份验证，但不是在用户级别上。这种流程使最终用户体验简单，但从API角度来看，控制较少。

首先，开发者在清单文件中添加访问令牌（全局密钥），在插件安装时，ChatGPT会存储令牌的加密版本，用户在安装插件时无需执行任何操作。在ChatGPT向插件发出请求时，会将访问令牌按照（“Authorization”：“[Bearer/Basic][user's token]”）格式传递到API服务端口。

```
"auth": {
  "type": "service_http",
  "authorization_type": "bearer",
  "verification_tokens": {
    "openai": "cb7cdfb8a57e45bc8ad7dea5bc2f8324"
  }
},
```

用户级别

当你的服务需要对用户进行认证时，可以设置用户级别的身份验证。通过在插件安装过程中让终端用户将其密钥API密钥复制粘贴到ChatGPT UI中。尽管ChatGPT在数据库中存储密钥时对其进行了加密，但鉴于用户体验不佳，所以不建议采用这种方法，推荐采用OAuth方式。

```
"auth": {
  "type": "user_http",
  "authorization_type": "bearer",
},
```

OAuth

OAuth（开放授权）是一个开放的授权标准，它允许用户在不共享用户名和密码的情况下，将他们的资

源（如数据、功能等）授权给第三方应用。此方式，让用户能够控制自己的数据和隐私，同时允许第三方应用在用户同意的情况下访问用户的资源。

插件协议与OAuth兼容。使用OAuth流程的简单示例如下：

1. 开发者粘贴他们的OAuth client_id和 client_secret
2. 将verification tokens添加到清单文件中
3. 安装时存储客户端密钥的加密版本
4. 用户在安装插件时通过插件的网站登录
5. ChatGPT端获得用户的OAuth访问令牌（以及可选的刷新令牌），并加密存储
6. 在向插件发出请求时，将该用户的令牌传递到授权头中（“Authorization”： “[Bearer/Basic] [user's token]”）

配置方式如下：

```
"auth": {
  "type": "oauth",
  "client_url": "https://my_server.com/authorize",
  "scope": "",
  "authorization_url": "https://my_server.com/token",
  "authorization_content_type": "application/json",
  "verification_tokens": {
    "openai": "abc123456"
  }
},
```

为了更好地理解OAuth的URL结构，以下是关于字段的简短说明：

- 当你使用ChatGPT设置插件时，将要求你提供OAuth client_id和client_secret
- 当用户登录插件时，ChatGPT会将用户的浏览器引导至

```
[client_url]?response_type=code&client_id=[client_id]
&scope=[scope]&redirect_uri=https%3A%2F%2Fchat.openai.com%2Faip%2F[plugin_id]%2Foauth%2Fcallback
```

- 在你的插件重定向回给定的redirect_uri之后，ChatGPT将通过向authorization_url发送POST请求以完成OAuth流程，内容类型为authorization_content_type，参数为

```
{“grant_type”：“authorization_code”，“client_id”：[client_id]，“client_secret”：[client_secret]，
“code”：[重定向返回的代码]，“redirect_uri”：[与之前相同的重定向URI] }
```

3 常见问题

如果你有任何问题，欢迎在视频下方留言提问，我会尽可能为你解答相关疑惑。
我会不定期追加相关问题以及解答。

参考文献

- 官方文档： https://platform.openai.com/docs/plugins/introduction
- Quart开发库： https://pgjones.gitlab.io/quart/
- Quart_cors开发库：https://github.com/pgjones/quart-cors/
- OpenAPI规范格式：https://swagger.io/tools/open-source/getting-started/

致谢与免责声明

感谢OpenAI提供了如此优秀的ChatGPT技术，让我们享受到AI带来的便利。
更要感谢你花时间阅读本内容，也非常期待你的意见与反馈。

由于能力有限以及相关技术更新速度的影响，部分内容不可避免地存在错误或已过时的问题。建议你及时通过页面顶部的链接查看是否有最新版本。