江 蘇 大 學

JIANGSU UNIVERSITY

《物联网系统实践 2》

云平台接入（Python）

课程设计报告

学院名称：计算机学院

专业班级：物联网工程 2303

学生姓名：邱佳亮

学生学号：3230611072

指导教师姓名：尹星

2025 年 1 月

目 录

基于 Python 的阿里云平台接入项目

1. 题目描述

本课程设计题目要求通过阿里云物联网平台实现数据传输功能。具体任务包括创建两个客户端，分别命名为 Sender 和 Receiver。Sender 负责将数据发送至阿里云物联网平台，平台接收到数据后，通过消息转发（云产品流转）功能将数据转发给 Receiver。Receiver 则负责接收并处理从平台转发过来的数据。

2. 设计目的

1.掌握物联网云平台的基本操作：学习如何在阿里云物联网平台上创建产品、设备，并获取设备的三元组信息。理解物联网云平台中设备管理、Topic 管理、消息转发等核心功能的使用方法。

2.理解 MQTT 协议及其在物联网中的应用：学习 MQTT 协议的基本原理，掌握发布/订阅模式的工作机制。通过编写 Python 客户端程序，实现设备与云平台之间的 MQTT 通信，理解如何通过 MQTT 协议发送和接收数据。

3.掌握设备与云平台之间的数据传输流程：实现设备数据的上传（Sender 发送数据到云平台）和数据的接收（Receiver 从云平台接收数据）。

4.学习云平台的消息转发（云产品流转）功能：掌握如何在阿里云物联网平台上配置消息转发规则，实现数据的自动流转。学习如何编写解析脚本，处理设备上报的数据，并将数据转发到指定的 Topic 或设备。

2. 功能要求和关键问题

2.1 系统功能描述

本系统基于阿里云物联网平台，旨在实现设备与云平台之间的数据传输，并通过云平台的消息转发功能将数据从发送端（Sender）传递到接收端（Receiver）。系统的主要功能模块包括设备管理、数据传输、消息转发和数据接收。

1. 设备管理功能：在阿里云物联网平台上创建两个设备，分别命名为 Sender 和 Receiver。为每个设备生成唯一的三元组信息，用于设备与云平台之间的身份认证和通信。

2. 数据传输功能：数据发送（Sender）：Sender 设备通过 MQTT 协议将数据发送到阿里云物联网平台。数据格式为 JSON，数据通过指定的 Topic 发布到云平台。

3. 消息转发功能：创建解析器，编写解析脚本，对 Sender 发送的数据进行解析，并将解析后的数据转发到 Receiver 的 Topic。

4. 数据接收与处理功能：Receiver 设备通过订阅指定的 Topic，实时接收从云平台转发过来的数

据。

2.2 复杂工程问题分析

1.数据传输与消息转发：数据需要通过 MQTT 协议从 Sender 设备发送到云平台，并通过云平台的消息转发功能将数据转发到 Receiver 设备。使用 JSON 格式作为数据传输的标准格式，确保数据的结构化和可解析性。在代码中实现数据的编码（Base64）和解码，确保二进制数据能够正确传输。

2.数据接收与展示：Receiver 设备需要实时接收从云平台转发过来的数据，并根据数据类型（如图片、音频、文本、视频）进行展示。使用 PyQt6 提供的控件（如 QLabel、QTextEdit）展示图片、文本等内容。对于音频和视频数据，使用 pyaudio 和 cv2 库进行播放和展示。

3.多线程与异步处理：系统需要同时处理多个任务，如设备连接、数据发送、数据接收、数据展示等，这些任务可能会相互阻塞，影响系统的响应速度。使用 Python 的 threading 模块实现多线程处理，确保每个任务能够独立运行，避免阻塞主线程。

4.用户界面与交互：系统需要提供友好的用户界面，允许用户进行设备配置、数据发送、数据接收等操作。使用 PyQt6 框架设计用户界面，提供设备信息输入、数据发送、数据接收等功能的操作界面。提供文件选择功能，允许用户选择要上传的文件（如图片、音频、视频等），并在界面中展示文件内容。

3. 系统整体框架设计

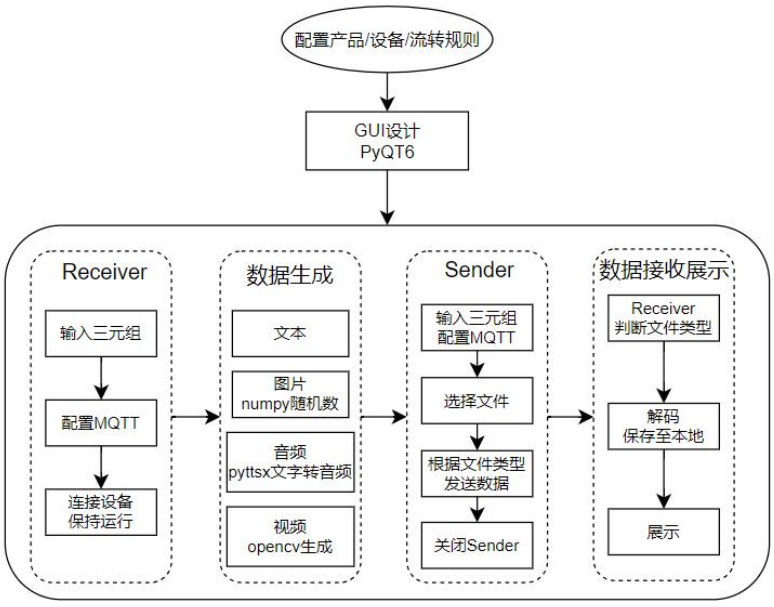

图 1　系统结构图

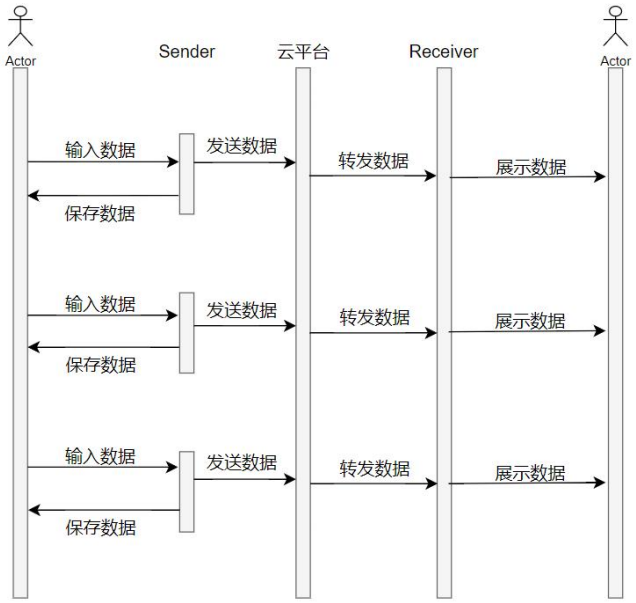

图 2　系统序列图

4. 模块的设计和实现

4.1 阿里云基础和设计

4.1.1 阿里云概述

阿里云物联网平台是阿里云提供的一个重要服务，旨在帮助用户快速构建和管理物联网设备，实现设备与云平台之间的高效通信和数据传输。通过阿里云物联网平台，用户可以轻松实现设备的接入、数据采集、设备管理、消息转发等功能，支持多种通信协议（如 MQTT、HTTP、CoAP 等），并提供丰富的 API 和 SDK，便于开发者进行二次开发和集成。阿里云物联网平台的核心功能包括设备管理、数据流转、规则引擎、设备影子等，能够满足从设备接入到数据分析的全流程需求。

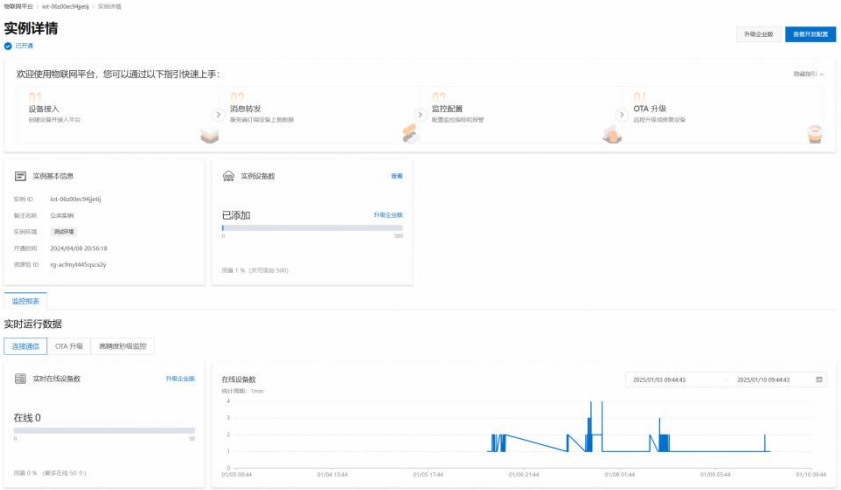

图 3　阿里物联网平台

4.1.2 阿里云设备创建

首先，用户需要在阿里云物联网平台上创建一个产品，产品是设备的抽象模型，定义了设备的类型、通信协议、数据格式等属性。创建产品时，用户需要填写产品名称、所属品类、联网方式、数据格式以及认证方式等信息。

图 4　创建的产品

产品创建完成后，用户可以在该产品下添加具体的设备。每个设备都有唯一的设备名称和设备密钥，这些信息构成了设备的三元组，用于设备与云平台之间的身份认证和通信。

图 5　产品下的两个设备

4.1.3 阿里云设备数据流转

首先，用户需要定义数据源和数据目的。设备通过 MQTT 协议将数据发布到指定的 Topic，云平台会根据配置的规则监听这些 Topic 中的数据。

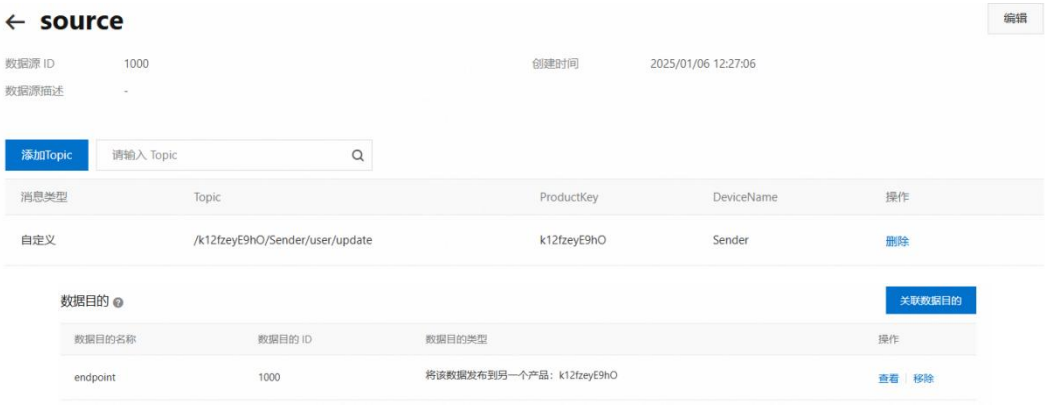

图 6　订阅的数据源和数据目的

用户可以通过编写解析脚本（JavaScript 脚本）对数据进行解析和转换。解析脚本可以从原始数据中提取有用的信息，并将其转换为目标格式。

编辑脚本 (当前展示为：线上运行脚本，您可以点击 编辑草稿)

```
// 草稿页为空时，进入草稿页会生成默认脚本
// 如果默认脚本自动保存过，继续绑定数据目的，默认脚本不会自动更新
// 此时清空脚本并保存之后，重新进入草稿页即可重新生成包含最新数据目的的默认脚本

// 设备上报数据内容，json格式
var data = payload('json');

// 流转到另一个Topic
writeIotTopic(1000, "/" + productKey() + "/Receiver/user/get", data);

```

图 7　解析脚本

最终完成流转解析器的配置并启动：

图 8　解析器

4.2 发布端程序设计

4.2.1 数据生成模块

在发布端（Sender）程序设计中，数据生成模块是核心功能之一，负责生成不同类型的数据并将其发送到阿里云物联网平台。

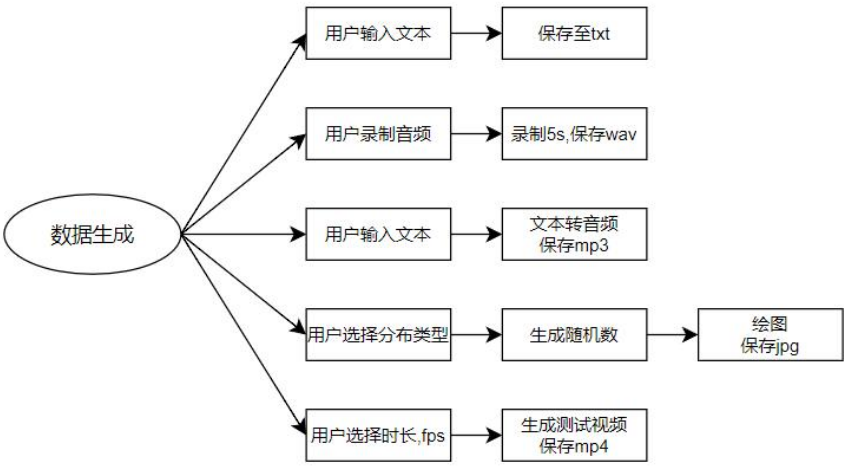

图 9　数据生成结构

4.2.1.1 文本生成

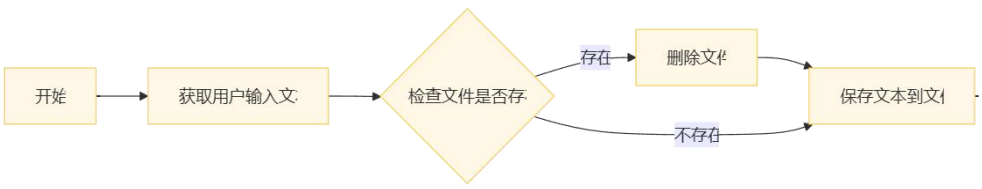

图 10　文本生成流程

用户可以通过界面输入文本内容，程序将用户输入的文本写入文本文件，保存在当前文件夹下，便于后续上传。

```
text=self.textinput.toPlainText() #从用户界面中的文本输入框（textinput）中获取用户输入的文本内容
if os.path.exists('word.txt'):
    os.remove('word.txt')
with open('word.txt','w',encoding='utf-8') as f: #指定编码为 UTF-8
    f.write(text)
```

4.2.1.2 音频生成

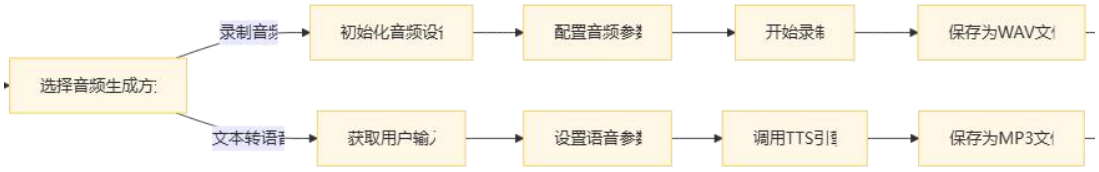

图 11　音频生成流程

音频生成模块的主要功能包括：通过麦克风录制音频，并将其保存为音频文件；通过文本转语音（TTS）技术生成音频文件。

音频录制功能通过 pyaudio 库实现。pyaudio 是一个跨平台的音频处理库，支持音频的录制和播放。通过 pyaudio 库初始化音频设备并配置参数（如采样率、通道数等），从麦克风录制音频数据并存储到缓冲区，最后将录制的音频数据保存为 WAV 文件，供后续处理或发送。

```
def recordF(self):
    audio = pyaudio.PyAudio()  # 初始化 PyAudio 对象
    FORMAT = pyaudio.paInt16  # 设置音频格式为 16 位深度
    CHANNELS = 1  # 设置音频通道数为 1（单声道）
    RATE = 44100  # 设置采样率为 44100Hz
    CHUNK = 1024  # 设置每个缓冲区的帧数为 1024
    RECORD_SECONDS = 5  # 设置录制时长为 5 秒
    # 打开音频流
    stream = audio.open(format=FORMAT,
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        frames_per_buffer=CHUNK)
    frames = []  # 用于存储录制的音频数据
    for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
        data = stream.read(CHUNK)  # 从音频流中读取数据
        frames.append(data)  # 将读取的数据添加到 frames 列表中
```

文本转语音（TTS）功能通过 pyttsx3 库实现。pyttsx3 是一个跨平台的 TTS 库。使用 pyttsx3 库将用户输入的文本转换为语音，设置语音引擎的属性（如语速、音量和语言），生成音频文件并保存为 MP3 格式，支持中文语音合成，确保生成的音频文件可直接用于发送或播放。

```
def music(self):
```

```
    # 设置语音引擎的属性
    self.engine.setProperty('rate', speed)  # 设置语速
    self.engine.setProperty('volume', vol)  # 设置音量
    # 设置语音引擎的语言为中文
    voices = self.engine.getProperty('voices')
    id = ''
    for voice in voices:
        if 'ZH-CN' in voice.id:  # 查找中文语音
            id = voice.id
            break
    self.engine.setProperty('voice', id)
    # 生成音频并保存为MP3文件
    self.engine.say(text)  # 将文本转换为语音
    self.engine.save_to_file(text, 'text.mp3')  # 将语音保存为MP3文件
    self.engine.runAndWait()  # 等待语音生成完成
    self.engine.stop()  # 停止语音引擎
```

4.2.1.3 图像生成

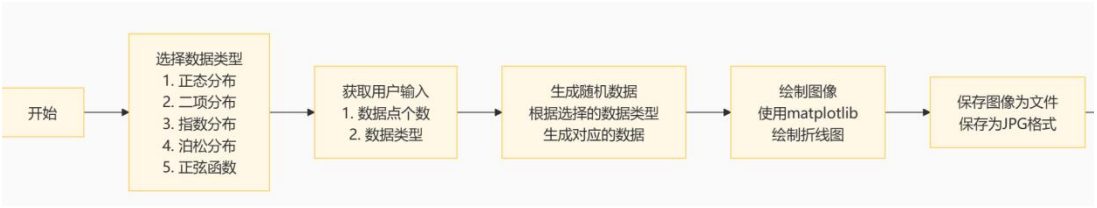

图 12 图像生成流程

通过 matplotlib 库生成随机数据的可视化图像。例如，生成正态分布、泊松分布等数据的折线图，并将其保存为图像文件。用户可以在下拉菜单中选择随机数据的类型。

```
def get_data(self):
    n = int(self.data_n.text())  # 获取用户设置的数据点个数
    # 根据用户选择的数据类型生成随机数据
    if mode == 0:
        data = np.random.normal(0, 1, n)  # 正态分布
    elif mode == 1:
        data = np.random.binomial(10, 0.5, size=n)  # 二项分布
    elif mode == 2:
        data = np.random.exponential(2, size=n)  # 指数分布
    elif mode == 3:
        lam = 10
        data = np.random.poisson(lam, n)  # 泊松分布
    elif mode == 4:
        t = np.linspace(1, n, n)
        data = np.sin(t)  # 正弦函数
    # 使用matplotlib生成图像
    fig, ax = plt.subplots(figsize=(10, 10))
    plt.tight_layout()
    plt.plot(data)  # 绘制折线图
```

4.2.1.4 视频生成

图 13　视频生成流程

通过 OpenCV 库生成简单的颜色变化视频。例如，生成一个颜色随时间变化的视频，并将其保存为 MP4 文件。

```
def make_vedio(self):
    # 创建视频写入对象
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # 设置视频编码格式
    out = cv2.VideoWriter(outpath, fourcc, fps, (width, height))  # 初始化视频写入对象
    # 生成视频帧
    for i in range(total_frames):
        frame = np.zeros((height, width, 3), dtype=np.uint8)  # 创建空白帧
        # 创建颜色变化
        r = (i * 255 // total_frames) % 256  # 红色通道
        g = ((i * 2) * 255 // total_frames) % 256  # 绿色通道
        b = ((i * 3) * 255 // total_frames) % 256  # 蓝色通道
        frame[:] = [b, g, r]  # 设置帧的颜色
        out.write(frame)  # 将帧写入视频文件
    out.release()  # 释放视频写入对象
```

4.2.2 数据发送模块

在发布端程序设计中，文件选择和数据发送模块是核心功能之一，负责选择文件（如图像、音频、视频、文本等）并将文件数据发送到阿里云物联网平台。

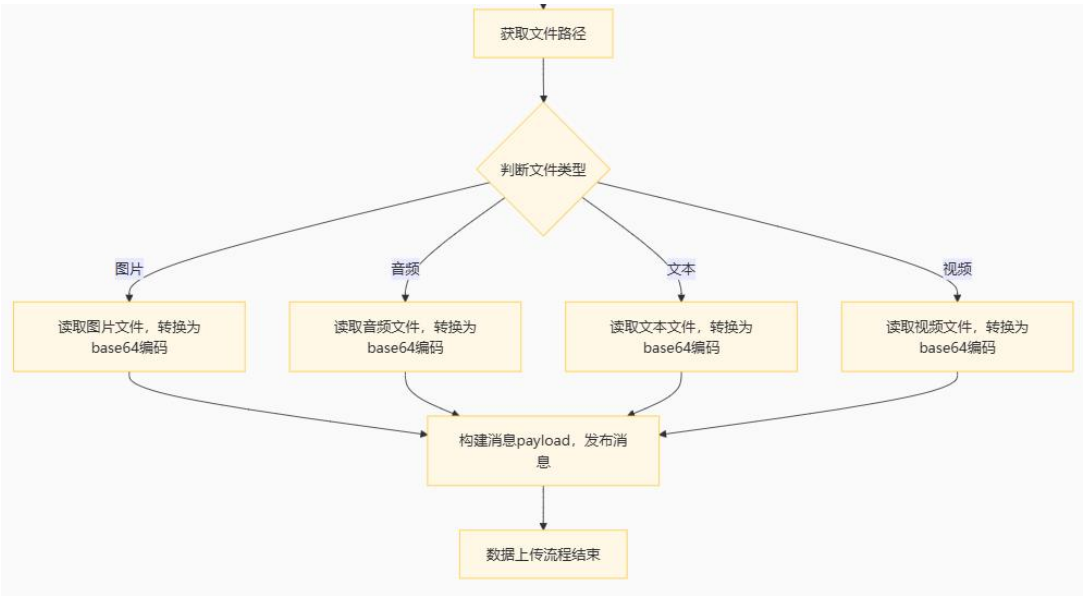

图 14　发送端模块流程

4.2.2.1 文件选择

文件选择模块的主要功能包括：通过用户界面选择本地文件（如图像、音频、视频、文本等）；根据文件扩展名识别文件类型（如 JPG、MP3、MP4、TXT 等）。

通过 QFileDialog 打开文件选择对话框，用户可以选择本地文件，程序将获取文件路径并显示在用户界面中。

```
def selectFile(self):
    file_dialog = QFileDialog()  # 创建文件选择对话框
    path, _ = file_dialog.getOpenFileName(None, "选择文件", "")  # 获取用户选择的文件路径
    self.path = path  # 保存文件路径
    self.filename.setText(path)  # 在界面中显示文件路径
```

4.2.2.2 数据发送

数据发送模块的主要功能包括：将文件内容进行 Base64 编码，以确保数据传输的完整性和安全性；通过 MQTT 协议将编码后的数据发送到阿里云物联网平台。

数据发送功能通过 linkkit 库实现，程序将文件内容进行 Base64 编码，并通过 MQTT 协议发送到阿里云物联网平台。根据文件类型读取文件内容，并将其进行 Base64 编码：

```
if type == "jpg" or type == "png" or type == "jpeg" or type == "gif":
    with open(file, "rb") as f:
        image_base64 = base64.b64encode(f.read()).decode("utf-8")  # 对图像文件进行Base64 编码
elif type == "mp3" or type == "wav":
    with open(file, 'rb') as f:
        audio_base64 = base64.b64encode(f.read()).decode("utf-8")  # 对音频文件进行Base64 编码
elif type == "txt":
    with open(file, 'rb') as f:
        word_base64 = base64.b64encode(f.read()).decode('utf-8')  # 对文本文件进行Base64 编码
elif type == 'mp4':
    with open(file, 'rb') as f:
        video_base64 = base64.b64encode(f.read()).decode('utf-8')  # 对视频文件进行Base64 编码
```

将编码后的数据封装为 JSON 格式的 MQTT 消息，并通过 MQTT 协议发送到阿里云物联网平台。

```
topic = f"/{sender_product_key}/{sender_device_name}/user/update"  # 设置MQTT Topic
if type == "jpg" or type == "png" or type == "jpeg" or type == "gif":
    payload = f'{{"type":"image","data":"{image_base64}"}}'  # 构造图像数据的MQTT 消息
elif type == "mp3" or type == "wav":
    payload = f'{{"type":"audio","data":"{audio_base64}"}}'  # 构造音频数据的MQTT 消息
elif type == "txt":
    payload = f'{{"type":"word","data":"{word_base64}"}}'  # 构造文本数据的MQTT 消息
elif type == 'mp4':
    payload = f'{{"type":"mp4","data":"{video_base64}"}}'  # 构造视频数据的MQTT 消息
lk_sender.publish_topic(topic, payload)  # 发送MQTT 消息
```

4.3 订阅端程序设计

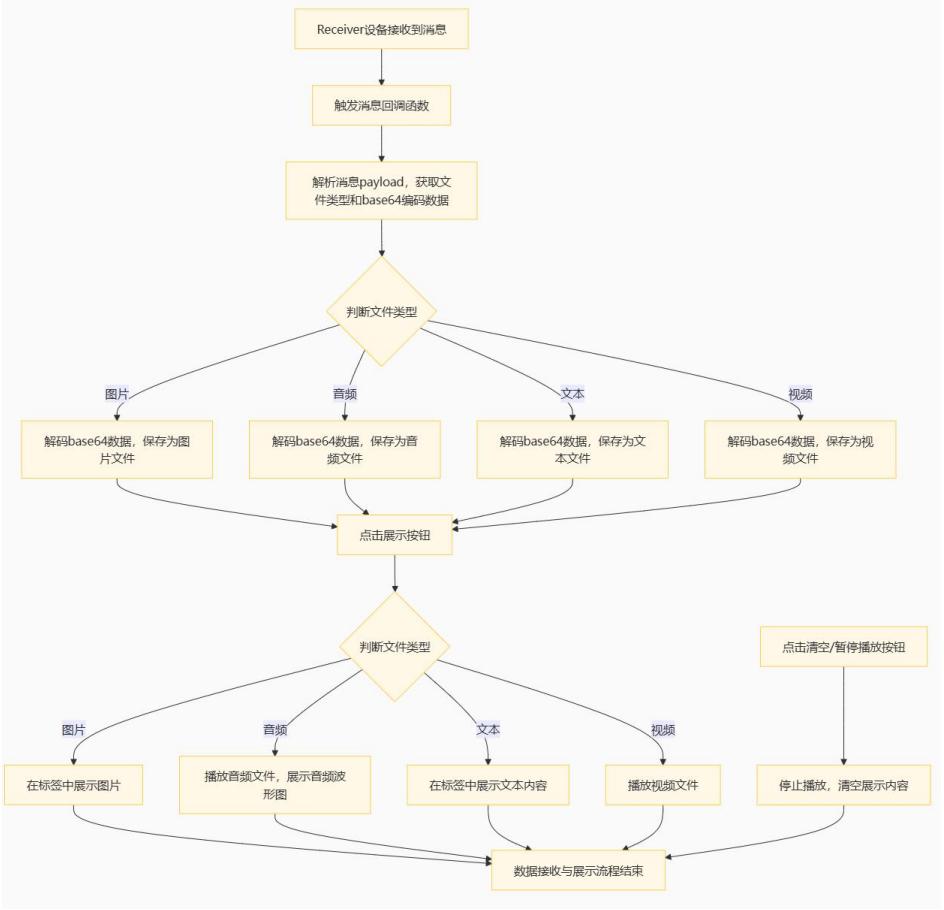

图 15　订阅端设计流程

4.3.1 数据接收模块

数据接收模块是订阅端程序的核心部分，负责从阿里云物联网平台接收数据，并根据数据类型进行解码、保存和展示。该模块的设计采用了多线程和异步连接机制，确保数据接收和处理的高效性和实时性。

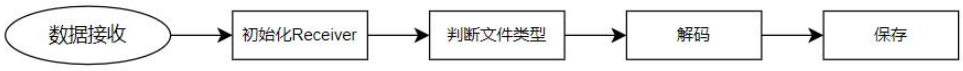

图 16　模块流程

数据接收模块首先需要初始化 Receiver 设备，配置 MQTT 连接，并订阅指定的 Topic 以接收数据。为了确保系统的响应性，Receiver 的初始化采用了多线程和异步连接机制。

Receiver 的初始化在一个独立的线程中运行，避免阻塞主线程。通过 threading.Thread 创建一个新的线程来执行 act_Receiver 方法：

```
def actF(self):
    self.receiver_thread = threading.Thread(target=self.act_Receiver)  # 创建线程
    self.receiver_thread.start()  # 启动线程
```

在 act_Receiver 方法中，程序通过 linkkit 库初始化 Receiver 设备，并配置 MQTT 连接。使用 connect_async()方法实现异步连接，确保连接过程不会阻塞主线程：

```
def act_Receiver(self):
    # Receiver 设备信息
    receiver_product_key = self.product_key_Edit_2.text()
    receiver_device_name = self.device_name_Edit_2.text()
    receiver_device_secret = self.device_secret_Edit_2.text()
    # 订阅Topic
    topic = f"/{receiver_product_key}/{receiver_device_name}/user/get"
    lk_receiver.subscribe_topic(topic, qos=1)
    lk_receiver.connect_async()  # 异步连接，避免阻塞主线程
    time.sleep(3)  # 等待连接完成
    self.is_running = True  # 设置运行状态为True
```

当 Receiver 接收到数据时，程序会根据数据中的 type 字段判断文件类型（如图像、音频、视频、文本等），以便进行后续的解码和保存操作。在消息回调函数 on_topic_message 中，程序解析接收到的 MQTT 消息，提取 type 字段和 data 字段：

```
def on_topic_message(topic, payload, qos, userdata):
    if isinstance(payload, bytes):
        data = json.loads(payload.decode("utf-8"))  # 解析JSON格式的payload
    self.type = data.get("type")  # 获取数据类型
    context = data.get("data")  # 获取数据内容
```

根据数据类型，程序调用 base64.b64decode()方法对数据进行解码（以图像为例）：

```
if self.type == "image":
    img_data = base64.b64decode(context)  # 解码图像数据
```

解码后的数据需要保存为本地文件，以便后续展示或处理。程序根据数据类型将数据保存为相应的文件格式（以图像为例）：

```
if self.type == "image":
    image = Image.open(BytesIO(img_data))  # 将二进制数据转换为图像
    image.save('temp.jpg')  # 保存图像文件
```

4.3.2 数据展示模块

数据展示模块是订阅端（Receiver）程序的重要组成部分，负责将接收到的数据（如图像、音频、视频、文本等）在用户界面中进行展示。该模块的设计需要根据不同的文件类型（如图像、音频、视频、文本）提供相应的展示方式，确保用户能够直观地查看和操作数据。

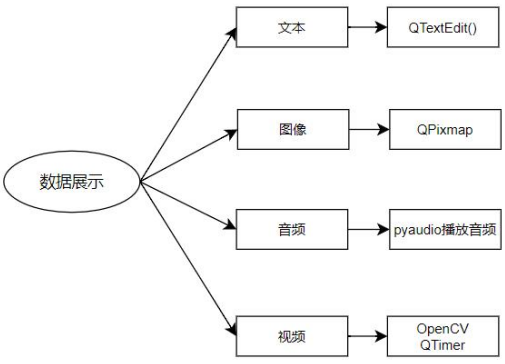

图 17　模块结构

4.3.2.1 图像展示

图 18　图像展示流程

图像展示功能通过 QPixmap 和 QLabel 实现，将接收到的图像文件显示在用户界面中。

```
if self.type == "image":
    pixmap = QPixmap(r"temp.jpg")  # 加载图像文件
    self.show_window.setPixmap(pixmap)  # 在界面中显示图像
    self.show_window.setScaledContents(True)  # 缩放图像以适应控件大小
```

4.3.2.2 文本展示

图 19　文本展示流程

文本展示功能通过 QTextEdit 实现，将接收到的文本文件内容显示在界面中：

```
if self.type == "word":
    with open('temp.txt', 'r', encoding='utf-8') as f:
        text = f.read()  # 读取文本文件内容
    self.show_window.setText(text)  # 在界面中显示文本
```

4.3.2.3 音频展示

图 20　音频展示流程

音频播放功能通过 pyaudio 库实现，播放过程在一个独立的线程中运行，以避免阻塞主线程。同时，程序提供了暂停播放的功能，用户可以通过界面按钮控制音频的播放状态。

```
def playAudio(self):
    self.stream = None
    self.p = None
    self.is_playing = False
    self.p = pyaudio.PyAudio()  # 初始化 PyAudio 对象
    wf = wave.open(file_path, 'rb')  # 打开音频文件
    # 打开音频流
    self.stream = self.p.open(
        format=self.p.get_format_from_width(wf.getsampwidth()),
        channels=wf.getnchannels(),
        rate=wf.getframerate(),
        output=True
    )
    self.is_playing = True  # 设置播放状态为 True
```

```
    # 启动新线程播放音频
    threading.Thread(target=self.play_audio, args=(wf,)).start()
def play_audio(self, wf):
    data = wf.readframes(1024)  # 读取音频数据
    while data and self.is_playing and self.stream:
        self.stream.write(data)  # 播放音频数据
        data = wf.readframes(1024)
```

通过设置 self.is_playing 变量控制音频播放状态。当用户点击暂停按钮时，程序将 self.is_playing 设置为 False，停止音频播放。

```
def pauseF(self):
    self.is_playing = False  # 设置播放状态为False，停止播放
    if self.stream:
        self.stream.stop_stream()  # 停止音频流
        self.stream.close()
        self.stream = None
    if self.p:
        self.p.terminate()  # 释放PyAudio 资源
        self.p = None
```

4.3.2.3 视频展示

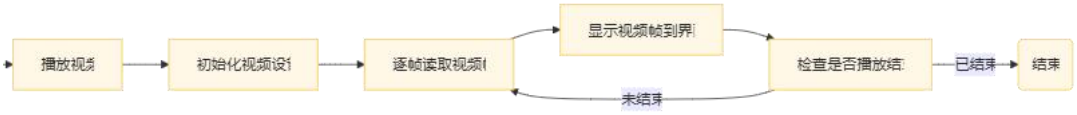

图 21 视频展示流程

视频播放功能通过 OpenCV 和 QTimer 实现，视频帧的更新在一个独立的线程中运行。视频播放过程通过 QTimer 定时更新视频帧，确保视频的流畅播放：

```
def play_vedio(self):
    path = "temp.mp4"  # 视频文件路径
    self.cap = cv2.VideoCapture(path)  # 打开视频文件
    self.timer = QTimer()  # 创建定时器
    self.timer.timeout.connect(lambda: self.update_frame(self.cap))  # 连接定时器信号
    self.timer.start(30)  # 设置定时器间隔为30 毫秒
def update_frame(self, cap):
    ret, frame = cap.read()  # 读取视频帧
    if not ret:
        self.timer.stop()  # 停止定时器
        cap.release()  # 释放视频资源
        return
    rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)  # 转换颜色空间
    h, w, ch = rgb_image.shape
    bytes_per_line = ch * w
    qt_image = QImage(rgb_image.data, w, h, bytes_per_line, QImage.Format.Format_RGB888)  #
转换为QImage
```

```
17.     pixmap = QPixmap.fromImage(qt_image)  # 转换为QPixmap
18.     pixmap = pixmap.scaled(350, 300)  # 缩放图像
19.     self.show_window.setPixmap(pixmap)  # 在界面中显示视频帧
20.     self.show_window.setScaledContents(True)
```

通过停止 QTimer 来暂停视频播放。当用户点击暂停按钮时，程序调用 self.timer.stop()停止定时器，从而停止视频帧的更新：

```
1.  def close_win(self):
2.      if self.timer.isActive():
3.          self.timer.stop()  # 停止定时器
4.      if self.cap:
5.          self.cap.release()  # 释放视频资源
6.      self.show_window.clear()  # 清空显示窗口
```

4.4 GUI 设计

该模块通过 PyQt6 框架实现，提供了丰富的控件和布局，确保用户界面的美观性和易用性。

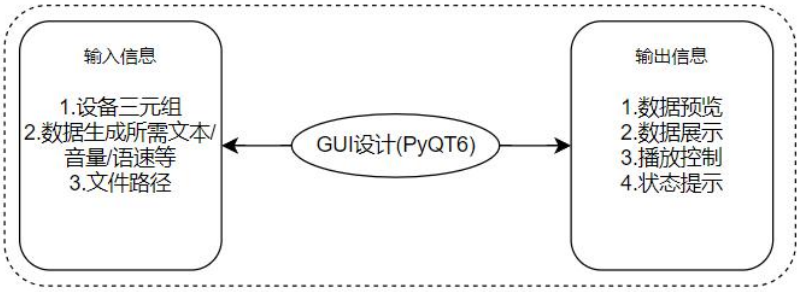

图 22　UI 模块结构

GUI 设计模块的主要功能包括：

1.设备信息输入：允许用户输入 Sender 和 Receiver 的设备信息（如 ProductKey、DeviceName、DeviceSecret）。

2.文件选择与上传：提供文件选择功能，用户可以选择本地文件并上传到云平台。

3.数据展示：根据数据类型（如图像、音频、视频、文本）在界面中展示接收到的数据。

4.播放控制：提供播放、暂停、清空等控制按钮，用户可以对音频和视频进行播放控制。

5. 调试和运行结果

5.1 运行结果(以图片传输为例)

点击激活按钮，初始化 Receiver，初始化完毕后，激活显示为红色：

图 23 激活成功

在阿里云物联网平台上也显示 Receiver 在线：

图 24 设备在线

生成正态分布的数据图像，并保存至本地：

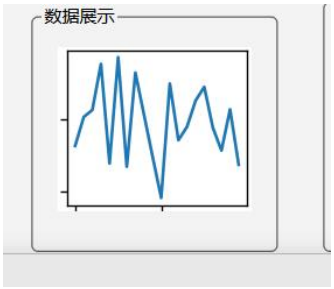

图 25 生成图像

选择上传该文件：

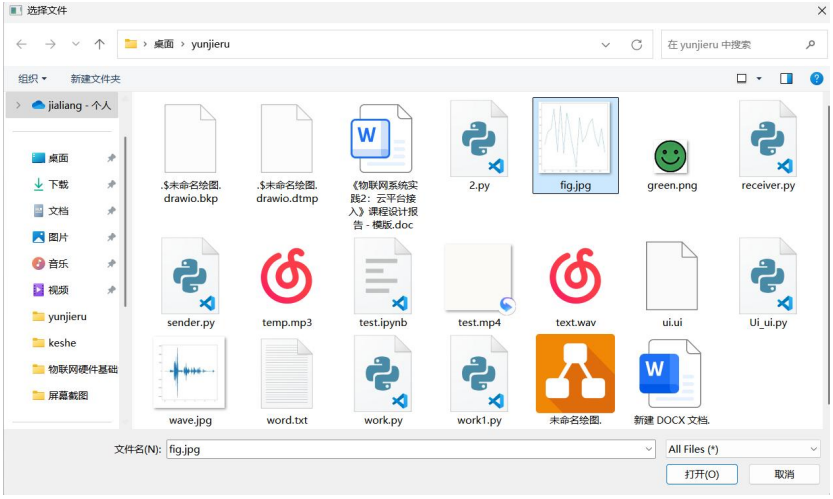

图 26 上传文件

在控制台提示上传成功，并打印了传输的 base64 数据：

```
Sender正在运行
上传成功
b'{"data":"/9j/4AAQSkZJRgABAQEAZABkAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UHRofHh0aHBwgJ
C4nICIsIxwcKDcpLDAxNDQ0Hyc5PTgyPC4zNDL/2wBDAQkJCQwLDBgNDRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIyM
jIyMjIyMjIyMjIyMjIyMjIyMjIyMjL/wAARCAPoA+gDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAA
AECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2Jyg
gkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZm
qKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBA
QEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBC
SMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaHi
ImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oAD
AMBAAIRAxEAPwD3+iiigAooooAKKKKACiigAooooAKKKKACiigAooooAKKKKACiigAooooAKKKKACiigAooooAKK
```

图 27　上传成功

点击展示按钮，图像展示在 UI 中：

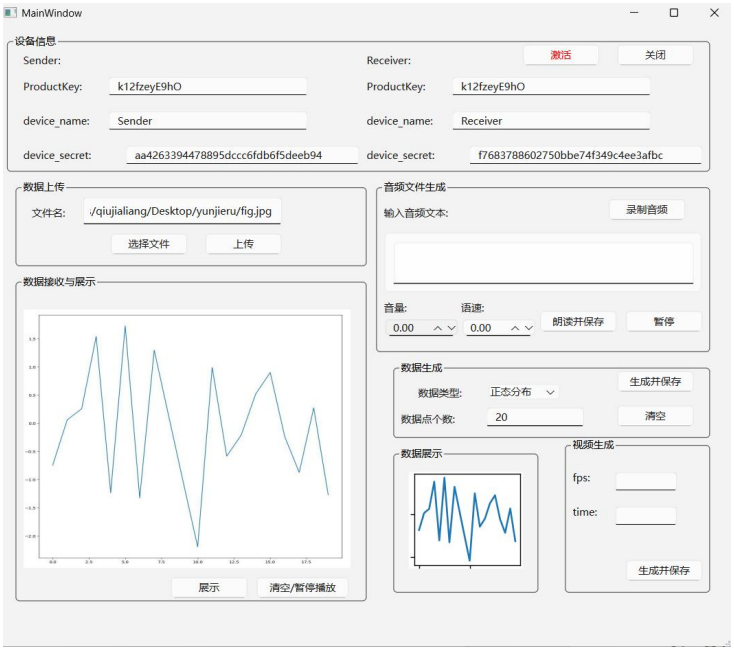

图 28　接收成功

类似的项目还可以传输文本、音频（显示音频波形图）和视频：

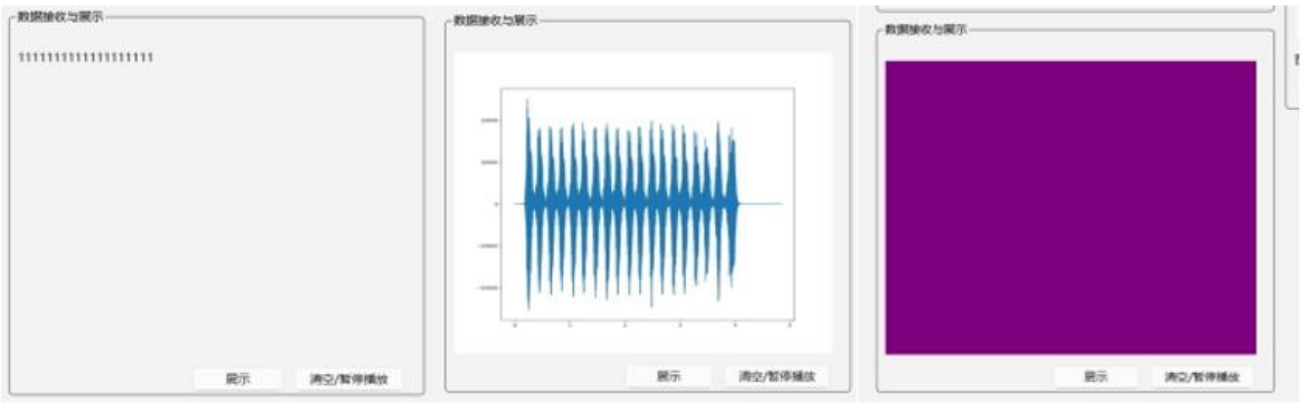

图 29　传输其他文件的结果

5.2 碰到的问题和解决方案

1. 在连接阿里云物联网平台时，MQTT 连接失败，设备无法与云平台建立通信。表现在文件传输不稳定，需要进行多次传输。

2. 在操作界面时，界面出现卡顿或无响应现象，影响用户体验。将数据处理、文件上传、播放等耗时操作放在独立的线程中运行，避免阻塞主线程。减少不必要的界面更新操作，例如仅在数据发生变化时更新界面。

6. 心得体会

通过本次课程设计，我深刻体会到了物联网系统开发的复杂性和挑战性。在实现阿里云物联网平

台的接入过程中，我不仅掌握了 MQTT 协议的基本原理和应用，还深入了解了物联网云平台的核心功能，如设备管理、Topic 管理、消息转发等。

在项目初期，我首先学习了如何在阿里云物联网平台上创建产品和设备，并获取设备的三元组信息（ProductKey、DeviceName、DeviceSecret）。这一过程让我理解了设备与云平台之间的身份认证机制，以及如何通过 MQTT 协议实现设备与云平台之间的通信。通过编写 Python 客户端程序，我进一步提升了编程能力，尤其是在处理 JSON 数据、多线程编程和异步处理方面有了更深的理解。JSON 作为数据传输的标准格式，确保了数据的结构化和可解析性，而多线程和异步处理则帮助我解决了系统响应速度慢的问题，确保了数据发送、接收和展示的流畅性。

在设计用户界面时，我使用了 PyQt6 框架，增强了界面的交互性和用户体验。通过合理的布局和控件设计，我实现了设备信息输入、文件选择与上传、数据展示等功能。用户界面的设计不仅要求美观，还需要考虑用户操作的便捷性。例如，在文件选择模块中，我通过 QFileDialog 实现了文件的选择和路径显示；在数据展示模块中，我根据数据类型（如图像、音频、视频、文本）提供了不同的展示方式，确保用户能够直观地查看和操作数据。此外，我还为音频和视频播放功能提供了播放、暂停、清空等控制按钮，进一步提升了用户的操作体验。

在调试过程中，我遇到了许多实际问题，如 MQTT 连接不稳定、音频视频播放卡顿、界面响应延迟等。在音频和视频播放卡顿的问题上，我通过将播放过程放在独立的线程中运行，避免了主线程的阻塞，从而解决了卡顿现象。此外，界面响应延迟的问题也通过减少不必要的界面更新操作得到了改善。

此外，本次实践让我更加理解了团队协作的重要性。在与同学的交流中，我学到了很多新的思路和方法。例如，在解决音频播放问题时，一位同学建议我使用 pyaudio 库的异步播放功能，这一建议极大地简化了我的代码逻辑，并提升了播放的流畅性。通过与同学的讨论，我还了解到了一些优化界面响应速度的技巧，如减少界面控件的频繁更新、使用缓存机制等。这些经验让我认识到，技术开发不仅是个人的努力，还需要借助团队的力量。

总的来说，这次课程设计让我受益匪浅，不仅提升了我的技术能力，也让我对物联网系统的整体架构和开发流程有了更清晰的认识。通过这次实践，我学会了如何将理论知识应用到实际项目中，如何解决开发过程中遇到的各种问题，以及如何通过团队协作提升开发效率。

7. 附录

7.1 分工

邱佳亮：数据生成、GUI 设计、数据展示　　陈默霖：Sender 和 Receiver 之间的数据传输。

7.2 源代码

```
import json
from PyQt6.QtGui import QPixmap,QImage
from PyQt6 import QtCore, QtGui, QtWidgets
from PyQt6.QtCore import QTimer,Qt
from PyQt6.QtWidgets import QVBoxLayout,QFileDialog
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_qtagg import FigureCanvasQTAgg as FigureCanvas
import pyttsx3
import threading
import time
import logging
import os
import time
import base64
from linkkit import linkkit
from PIL import Image
from io import BytesIO
import wave
import pyaudio
from scipy.io import wavfile
import numpy as np
from io import StringIO,BytesIO
import cv2
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(859, 723)
        self.centralwidget = QtWidgets.QWidget(parent=MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.data_creat = QtWidgets.QGroupBox(parent=self.centralwidget)
        self.data_creat.setGeometry(QtCore.QRect(460, 390, 371, 91))
        self.data_creat.setObjectName("data_creat")
        self.creat = QtWidgets.QPushButton(parent=self.data_creat)
        self.creat.setGeometry(QtCore.QRect(260, 10, 93, 28))
        self.creat.setObjectName("creat")
        self.data_feature = QtWidgets.QComboBox(parent=self.data_creat)
        self.data_feature.setGeometry(QtCore.QRect(110, 25, 87, 22))
        self.data_feature.setObjectName("data_feature")
        self.data_feature.addItem("")
        self.data_feature.addItem("")
        self.data_feature.addItem("")
        self.data_feature.addItem("")
        self.data_feature.addItem("")
        self.label = QtWidgets.QLabel(parent=self.data_creat)
        self.label.setGeometry(QtCore.QRect(30, 30, 72, 15))
        self.label.setObjectName("label")
```

```
48.         self.label_2 = QtWidgets.QLabel(parent=self.data_creat)
49.         self.label_2.setGeometry(QtCore.QRect(10, 60, 91, 16))
50.         self.label_2.setObjectName("label_2")
51.         self.data_n = QtWidgets.QLineEdit(parent=self.data_creat)
52.         self.data_n.setGeometry(QtCore.QRect(110, 55, 113, 21))
53.         self.data_n.setObjectName("data_n")
54.         self.clear = QtWidgets.QPushButton(parent=self.data_creat)
55.         self.clear.setGeometry(QtCore.QRect(260, 50, 93, 28))
56.         self.clear.setObjectName("clear")
57.         self.groupBox_2 = QtWidgets.QGroupBox(parent=self.centralwidget)
58.         self.groupBox_2.setGeometry(QtCore.QRect(460, 490, 171, 171))
59.         self.groupBox_2.setObjectName("groupBox_2")
60.         self.data_p = QtWidgets.QLabel(parent=self.groupBox_2)
61.         self.data_p.setGeometry(QtCore.QRect(10, 20, 151, 131))
62.         self.data_p.setText("")
63.         self.data_p.setObjectName("data_p")
64.         self.groupBox = QtWidgets.QGroupBox(parent=self.centralwidget)
65.         self.groupBox.setGeometry(QtCore.QRect(440, 180, 391, 201))
66.         self.groupBox.setObjectName("groupBox")
67.         self.label_3 = QtWidgets.QLabel(parent=self.groupBox)
68.         self.label_3.setGeometry(QtCore.QRect(10, 30, 101, 16))
69.         self.label_3.setObjectName("label_3")
70.         self.scrollArea = QtWidgets.QScrollArea(parent=self.groupBox)
71.         self.scrollArea.setGeometry(QtCore.QRect(10, 60, 371, 71))
72.         self.scrollArea.setWidgetResizable(True)
73.         self.scrollArea.setObjectName("scrollArea")
74.         self.scrollAreaWidgetContents = QtWidgets.QWidget()
75.         self.scrollAreaWidgetContents.setGeometry(QtCore.QRect(0, 0, 369, 69))
76.         self.scrollAreaWidgetContents.setObjectName("scrollAreaWidgetContents")
77.         self.textinput = QtWidgets.QTextEdit(parent=self.scrollAreaWidgetContents)
78.         self.textinput.setGeometry(QtCore.QRect(10, 10, 351, 51))
79.         self.textinput.setObjectName("textinput")
80.         self.scrollArea.setWidget(self.scrollAreaWidgetContents)
81.         self.gen = QtWidgets.QPushButton(parent=self.groupBox)
82.         self.gen.setGeometry(QtCore.QRect(190, 150, 93, 28))
83.         self.gen.setObjectName("gen")
84.         self.pause = QtWidgets.QPushButton(parent=self.groupBox)
85.         self.pause.setGeometry(QtCore.QRect(290, 150, 93, 28))
86.         self.pause.setObjectName("pause")
87.         self.label_4 = QtWidgets.QLabel(parent=self.groupBox)
88.         self.label_4.setGeometry(QtCore.QRect(10, 140, 41, 16))
89.         self.label_4.setObjectName("label_4")
90.         self.label_5 = QtWidgets.QLabel(parent=self.groupBox)
91.         self.label_5.setGeometry(QtCore.QRect(100, 140, 41, 16))
92.         self.label_5.setObjectName("label_5")
93.         self.volumn = QtWidgets.QDoubleSpinBox(parent=self.groupBox)
94.         self.volumn.setGeometry(QtCore.QRect(10, 160, 81, 22))
```

```
95.         self.volumn.setObjectName("volumn")
96.         self.speed = QtWidgets.QDoubleSpinBox(parent=self.groupBox)
97.         self.speed.setGeometry(QtCore.QRect(100, 160, 81, 22))
98.         self.speed.setObjectName("speed")
99.         self.record = QtWidgets.QPushButton(parent=self.groupBox)
100.         self.record.setGeometry(QtCore.QRect(270, 20, 93, 28))
101.         self.record.setObjectName("record")
102.         self.groupBox_3 = QtWidgets.QGroupBox(parent=self.centralwidget)
103.         self.groupBox_3.setGeometry(QtCore.QRect(10, 10, 821, 161))
104.         self.groupBox_3.setObjectName("groupBox_3")
105.         self.label_6 = QtWidgets.QLabel(parent=self.groupBox_3)
106.         self.label_6.setGeometry(QtCore.QRect(20, 20, 61, 21))
107.         self.label_6.setObjectName("label_6")
108.         self.label_7 = QtWidgets.QLabel(parent=self.groupBox_3)
109.         self.label_7.setGeometry(QtCore.QRect(420, 20, 81, 21))
110.         self.label_7.setObjectName("label_7")
111.         self.label_8 = QtWidgets.QLabel(parent=self.groupBox_3)
112.         self.label_8.setGeometry(QtCore.QRect(20, 50, 91, 21))
113.         self.label_8.setObjectName("label_8")
114.         self.label_9 = QtWidgets.QLabel(parent=self.groupBox_3)
115.         self.label_9.setGeometry(QtCore.QRect(20, 90, 101, 21))
116.         self.label_9.setObjectName("label_9")
117.         self.label_10 = QtWidgets.QLabel(parent=self.groupBox_3)
118.         self.label_10.setGeometry(QtCore.QRect(20, 130, 121, 21))
119.         self.label_10.setObjectName("label_10")
120.         self.product_key_Edit = QtWidgets.QLineEdit(parent=self.groupBox_3)
121.         self.product_key_Edit.setGeometry(QtCore.QRect(120, 50, 231, 21))
122.         self.product_key_Edit.setObjectName("product_key_Edit")
123.         self.device_name_Edit = QtWidgets.QLineEdit(parent=self.groupBox_3)
124.         self.device_name_Edit.setGeometry(QtCore.QRect(120, 90, 231, 21))
125.         self.device_name_Edit.setObjectName("device_name_Edit")
126.         self.device_secret_Edit = QtWidgets.QLineEdit(parent=self.groupBox_3)
127.         self.device_secret_Edit.setGeometry(QtCore.QRect(140, 130, 271, 21))
128.         self.device_secret_Edit.setObjectName("device_secret_Edit")
129.         self.product_key_Edit_2 = QtWidgets.QLineEdit(parent=self.groupBox_3)
130.         self.product_key_Edit_2.setGeometry(QtCore.QRect(520, 50, 231, 21))
131.         self.product_key_Edit_2.setObjectName("product_key_Edit_2")
132.         self.device_secret_Edit_2 = QtWidgets.QLineEdit(parent=self.groupBox_3)
133.         self.device_secret_Edit_2.setGeometry(QtCore.QRect(540, 130, 271, 21))
134.         self.device_secret_Edit_2.setObjectName("device_secret_Edit_2")
135.         self.label_11 = QtWidgets.QLabel(parent=self.groupBox_3)
136.         self.label_11.setGeometry(QtCore.QRect(420, 50, 91, 21))
137.         self.label_11.setObjectName("label_11")
138.         self.label_12 = QtWidgets.QLabel(parent=self.groupBox_3)
139.         self.label_12.setGeometry(QtCore.QRect(420, 130, 121, 21))
140.         self.label_12.setObjectName("label_12")
141.         self.label_13 = QtWidgets.QLabel(parent=self.groupBox_3)
```

```
142.         self.label_13.setGeometry(QtCore.QRect(420, 90, 101, 21))
143.         self.label_13.setObjectName("label_13")
144.         self.device_name_Edit_2 = QtWidgets.QLineEdit(parent=self.groupBox_3)
145.         self.device_name_Edit_2.setGeometry(QtCore.QRect(520, 90, 231, 21))
146.         self.device_name_Edit_2.setObjectName("device_name_Edit_2")
147.         self.act = QtWidgets.QPushButton(parent=self.groupBox_3)
148.         self.act.setGeometry(QtCore.QRect(600, 10, 93, 28))
149.         self.act.setObjectName("act")
150.         self.close = QtWidgets.QPushButton(parent=self.groupBox_3)
151.         self.close.setGeometry(QtCore.QRect(710, 10, 93, 28))
152.         self.close.setObjectName("close")
153.         self.groupBox_4 = QtWidgets.QGroupBox(parent=self.centralwidget)
154.         self.groupBox_4.setGeometry(QtCore.QRect(20, 180, 411, 101))
155.         self.groupBox_4.setObjectName("groupBox_4")
156.         self.label_14 = QtWidgets.QLabel(parent=self.groupBox_4)
157.         self.label_14.setGeometry(QtCore.QRect(20, 30, 51, 16))
158.         self.label_14.setObjectName("label_14")
159.         self.filename = QtWidgets.QLineEdit(parent=self.groupBox_4)
160.         self.filename.setGeometry(QtCore.QRect(80, 20, 231, 31))
161.         self.filename.setObjectName("filename")
162.         self.upload = QtWidgets.QPushButton(parent=self.groupBox_4)
163.         self.upload.setGeometry(QtCore.QRect(220, 60, 93, 28))
164.         self.upload.setObjectName("upload")
165.         self.select = QtWidgets.QPushButton(parent=self.groupBox_4)
166.         self.select.setGeometry(QtCore.QRect(110, 60, 93, 28))
167.         self.select.setObjectName("select")
168.         self.face = QtWidgets.QLabel(parent=self.groupBox_4)
169.         self.face.setGeometry(QtCore.QRect(330, 20, 71, 71))
170.         self.face.setText("")
171.         self.face.setObjectName("face")
172.         self.groupBox_5 = QtWidgets.QGroupBox(parent=self.centralwidget)
173.         self.groupBox_5.setGeometry(QtCore.QRect(20, 290, 411, 381))
174.         self.groupBox_5.setObjectName("groupBox_5")
175.         self.clr_win = QtWidgets.QPushButton(parent=self.groupBox_5)
176.         self.clr_win.setGeometry(QtCore.QRect(280, 350, 111, 28))
177.         self.clr_win.setObjectName("clr_win")
178.         self.show_window = QtWidgets.QLabel(parent=self.groupBox_5)
179.         self.show_window.setGeometry(QtCore.QRect(10, 40, 381, 301))
180.         self.show_window.setText("")
181.         self.show_window.setAlignment(QtCore.Qt.AlignmentFlag.AlignLeading|QtCore.Qt.Alignment
Flag.AlignLeft|QtCore.Qt.AlignmentFlag.AlignTop)
182.         self.show_window.setObjectName("show_window")
183.         self.play = QtWidgets.QPushButton(parent=self.groupBox_5)
184.         self.play.setGeometry(QtCore.QRect(180, 350, 93, 28))
185.         self.play.setObjectName("play")
186.         self.groupBox_6 = QtWidgets.QGroupBox(parent=self.centralwidget)
187.         self.groupBox_6.setGeometry(QtCore.QRect(660, 480, 171, 181))
```

```
        self.groupBox_6.setObjectName("groupBox_6")
        self.fps = QtWidgets.QLineEdit(parent=self.groupBox_6)
        self.fps.setGeometry(QtCore.QRect(60, 40, 71, 21))
        self.fps.setObjectName("fps")
        self.label_15 = QtWidgets.QLabel(parent=self.groupBox_6)
        self.label_15.setGeometry(QtCore.QRect(10, 30, 41, 31))
        self.label_15.setObjectName("label_15")
        self.time = QtWidgets.QLineEdit(parent=self.groupBox_6)
        self.time.setGeometry(QtCore.QRect(60, 80, 71, 21))
        self.time.setText("")
        self.time.setObjectName("time")
        self.label_16 = QtWidgets.QLabel(parent=self.groupBox_6)
        self.label_16.setGeometry(QtCore.QRect(10, 70, 41, 31))
        self.label_16.setObjectName("label_16")
        self.creat_vedio = QtWidgets.QPushButton(parent=self.groupBox_6)
        self.creat_vedio.setGeometry(QtCore.QRect(70, 140, 93, 28))
        self.creat_vedio.setObjectName("creat_vedio")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(parent=MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 859, 26))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(parent=MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)
    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.data_creat.setTitle(_translate("MainWindow", "数据生成"))
        self.creat.setText(_translate("MainWindow", "生成并保存"))
        self.data_feature.setItemText(0, _translate("MainWindow", "正态分布"))
        self.data_feature.setItemText(1, _translate("MainWindow", "二项分布"))
        self.data_feature.setItemText(2, _translate("MainWindow", "指数分布"))
        self.data_feature.setItemText(3, _translate("MainWindow", "泊松分布"))
        self.data_feature.setItemText(4, _translate("MainWindow", "正弦函数"))
        self.label.setText(_translate("MainWindow", "数据类型:"))
        self.label_2.setText(_translate("MainWindow", "数据点个数:"))
        self.clear.setText(_translate("MainWindow", "清空"))
        self.groupBox_2.setTitle(_translate("MainWindow", "数据展示"))
        self.groupBox.setTitle(_translate("MainWindow", "音频文件生成"))
        self.label_3.setText(_translate("MainWindow", "输入音频文本:"))
        self.gen.setText(_translate("MainWindow", "朗读并保存"))
        self.pause.setText(_translate("MainWindow", "暂停"))
        self.label_4.setText(_translate("MainWindow", "音量:"))
        self.label_5.setText(_translate("MainWindow", "语速:"))
```

```
235.         self.record.setText(_translate("MainWindow", "录制音频"))
236.         self.groupBox_3.setTitle(_translate("MainWindow", "设备信息"))
237.         self.label_6.setText(_translate("MainWindow", "Sender:"))
238.         self.label_7.setText(_translate("MainWindow", "Receiver:"))
239.         self.label_8.setText(_translate("MainWindow", "ProductKey:"))
240.         self.label_9.setText(_translate("MainWindow", "device_name:"))
241.         self.label_10.setText(_translate("MainWindow", "device_secret:"))
242.         self.product_key_Edit.setText(_translate("MainWindow", "k12fzeyE9hO"))
243.         self.device_name_Edit.setText(_translate("MainWindow", "Sender"))
244.         self.device_secret_Edit.setText(_translate("MainWindow", "aa4263394478895dccc6fdb6f5de
eb94"))
245.         self.product_key_Edit_2.setText(_translate("MainWindow", "k12fzeyE9hO"))
246.         self.device_secret_Edit_2.setText(_translate("MainWindow", "f7683788602750bbe74f349c4e
e3afbc"))
247.         self.label_11.setText(_translate("MainWindow", "ProductKey:"))
248.         self.label_12.setText(_translate("MainWindow", "device_secret:"))
249.         self.label_13.setText(_translate("MainWindow", "device_name:"))
250.         self.device_name_Edit_2.setText(_translate("MainWindow", "Receiver"))
251.         self.act.setText(_translate("MainWindow", "激活"))
252.         self.close.setText(_translate("MainWindow", "关闭"))
253.         self.groupBox_4.setTitle(_translate("MainWindow", "数据上传"))
254.         self.label_14.setText(_translate("MainWindow", "文件名:"))
255.         self.upload.setText(_translate("MainWindow", "上传"))
256.         self.select.setText(_translate("MainWindow", "选择文件"))
257.         self.groupBox_5.setTitle(_translate("MainWindow", "数据接收与展示"))
258.         self.clr_win.setText(_translate("MainWindow", "清空/暂停播放"))
259.         self.play.setText(_translate("MainWindow", "展示"))
260.         self.groupBox_6.setTitle(_translate("MainWindow", "视频生成"))
261.         self.label_15.setText(_translate("MainWindow", "fps:"))
262.         self.label_16.setText(_translate("MainWindow", "time:"))
263.         self.creat_vedio.setText(_translate("MainWindow", "生成并保存"))
264.         self.speed.setMaximum(1000)
265.         self.speed.setMinimum(0)
266.         self.volumn.setMaximum(10)
267.         self.volumn.setMinimum(0)
268.         self.engine = pyttsx3.init()  # 初始化语音引擎
269.         self.data_n.textChanged.connect(self.showText)
270.         self.filename.textChanged.connect(self.showText)
271.         self.time.textChanged.connect(self.showText)
272.         self.fps.textChanged.connect(self.showText)
273.         self.creat.clicked.connect(self.get_data)
274.         self.clear.clicked.connect(self.clearF)
275.         #self.clear_vedio.clicked.connect(self.clearF)
276.         self.gen.clicked.connect(self.music)
277.         self.pause.clicked.connect(self.pauseF)
278.         self.upload.clicked.connect(self.uploadF)
279.         self.act.clicked.connect(self.actF)
```

```
280.         self.close.clicked.connect(self.closeF)
281.         self.is_running=None
282.         self.mode=0
283.         self.clr_win.clicked.connect(self.close_win)
284.         self.play.clicked.connect(self.playF)
285.         self.select.clicked.connect(self.selectFile)
286.         self.creat_vedio.clicked.connect(self.make_vedio)
287.         self.record.clicked.connect(self.recordF)
288.     def recordF(self):
289.         audio = pyaudio.PyAudio()
290.         FORMAT = pyaudio.paInt16  # 16 位深度
291.         CHANNELS = 1  # 单声道
292.         RATE = 44100  # 采样率
293.         CHUNK = 1024  # 每个缓冲区的帧数
294.         RECORD_SECONDS = 5  # 录制时长
295.         BITRATE = "16k"
296.         # 打开音频流
297.         stream = audio.open(format=pyaudio.paInt16,
298.                             channels=1,
299.                             rate=44100,
300.                             input=True,
301.                             frames_per_buffer=1024)
302.         print("开始录制...")
303.         frames = []
304.         for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
305.             data = stream.read(CHUNK)
306.             frames.append(data)
307.         print("录制结束")
308.         stream.stop_stream()
309.         stream.close()
310.         audio.terminate()
311.         wf = wave.open('text.wav', 'wb')
312.         wf.setnchannels(CHANNELS)
313.         wf.setsampwidth(audio.get_sample_size(FORMAT))
314.         wf.setframerate(RATE)
315.         wf.writeframes(b''.join(frames))
316.         wf.close()
317.     def make_vedio(self):
318.         duration=int(self.time.text())
319.         fps=int(self.fps.text())
320.         width,height=100,100
321.         total_frames=fps*duration
322.         outpath='test.mp4'
323.         fourcc=cv2.VideoWriter_fourcc(*'mp4v')
324.         out=cv2.VideoWriter(outpath,fourcc,fps,(width,height))
325.         for i in range(total_frames):
326.             frame=np.zeros((height,width,3),dtype=np.uint8)
```

```
            # 创建颜色变化
            r = (i * 255 // total_frames) % 256
            g = ((i * 2) * 255 // total_frames) % 256
            b = ((i * 3) * 255 // total_frames) % 256
            frame[:]=[b,g,r]
            out.write(frame)
        out.release()
    def selectFile(self):
        file_dialog = QFileDialog()
        path, _ = file_dialog.getOpenFileName(None, "选择文件", "")
        self.path=path
        self.filename.setText(path)
    def play_audio(self,wf):
            data = wf.readframes(1024)
            while data and self.is_playing and self.stream:
                self.stream.write(data)
                data = wf.readframes(1024)
            if self.stream:
                self.stream.stop_stream()
                self.stream.close()
                self.stream=None
            if self.p:
                self.p.terminate()
                self.p=None
            self.is_playing = False
    def playAudio(self):
        self.stream=None
        self.p=None
        self.is_playing=False
        file_path = "temp.mp3"
        self.p=pyaudio.PyAudio()
        wf=wave.open(file_path,'rb')
        self.stream = self.p.open(format=self.p.get_format_from_width(wf.getsampwidth()),
                        channels=wf.getnchannels(),
                        rate=wf.getframerate(),
                        output=True)
        self.is_playing=True
        threading.Thread(target=self.play_audio,args=(wf,)).start()

        sample_rate, audio_data = wavfile.read(file_path)
        if len(audio_data.shape) > 1:
            audio_data=audio_data[:, 0]
        fig,ax=plt.subplots(figsize=(10,10))
        ax.plot(np.arange(len(audio_data))/sample_rate,audio_data)
        fig.savefig("wave.jpg")
        pixmap=QPixmap(r"wave.jpg")
        self.show_window.setPixmap(pixmap)
```

```
374.         self.show_window.setScaledContents(True)
375.     def update_frame(self,cap):
376.         ret,frame=cap.read()
377.         if not ret:
378.             self.timer.stop()
379.             cap.release()
380.             return
381.         rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
382.         h, w, ch = rgb_image.shape
383.         bytes_per_line = ch * w
384.         qt_image = QImage(rgb_image.data, w, h, bytes_per_line, QImage.Format.Format_RGB888)
385.
386.         # 将 QImage 转换为 QPixmap 并显示
387.         pixmap = QPixmap.fromImage(qt_image)
388.         pixmap = pixmap.scaled(350, 300)  # 缩放图像
389.         self.show_window.setPixmap(pixmap)
390.         self.show_window.setScaledContents(True)
391.     def play_vedio(self):
392.         path="temp.mp4"
393.         self.cap=cv2.VideoCapture(path)
394.         self.timer=QTimer()
395.         self.timer.timeout.connect(lambda:self.update_frame(self.cap))
396.         self.timer.start(30)
397.     def playF(self):
398.         self.show_window.clear()
399.         if self.type=="image":
400.             pixmap=QPixmap(r"temp.jpg")
401.             self.show_window.setPixmap(pixmap)
402.             self.show_window.setScaledContents(True)
403.         elif self.type=="audio":
404.             self.playAudio()
405.         elif self.type=="word":
406.             with open('temp.txt','r',encoding='utf-8') as f:
407.                     text=f.read()
408.             self.show_window.setText(text)
409.         elif self.type=="mp4":
410.             self.play_vedio()
411.     def close_win(self):
412.         self.is_playing=False
413.         if self.timer.isActive():
414.             self.timer.stop()
415.         if self.cap:
416.             self.cap.release()
417.         self.show_window.clear()
418.     def closeF(self):
419.         self.is_running=False
420.         if os.path.exists('temp.mp4'):
```

```
421.             os.remove('temp.mp4')
422.         if os.path.exists('temp.txt'):
423.             os.remove('temp.txt')
424.         if os.path.exists('temp.mp3'):
425.             os.remove('temp.mp3')
426.         if os.path.exists('temp.wav'):
427.             os.remove('temp.wav')
428.         if os.path.exists('temp.jpg'):
429.             os.remove('temp.jpg')
430.     def uploadF(self):
431.         target=self.product_key_Edit_2.text()
432.         __log_format = "%(asctime)s-%(process)d-%(thread)d - %(name)s:%(module)s:%(funcName)s
- %(levelname)s - %(message)s"
433.         logging.basicConfig(format=__log_format)
434.         sender_product_key = self.product_key_Edit.text()
435.         sender_device_name = self.device_name_Edit.text()
436.         sender_device_secret = self.device_secret_Edit.text()
437.         #self.check()
438.         lk_sender = linkkit.LinkKit(
439.             host_name="cn-shanghai",
440.             product_key=sender_product_key,
441.             device_name=sender_device_name,
442.             device_secret=sender_device_secret,
443.         )
444.         lk_sender.config_mqtt(
445.             port=8883,
446.             protocol="MQTTv311",
447.             transport="TCP",
448.             secure="TLS",
449.             keep_alive=60,
450.             clean_session=True,
451.             max_inflight_message=20,
452.             max_queued_message=0,
453.             auto_reconnect_min_sec=1,
454.             auto_reconnect_max_sec=60,
455.             cadata=None,
456.         )
457.         lk_sender.connect_async()
458.         file=self.filename.text()
459.         if os.path.exists(file):
460.             # pixmap=QPixmap("green.png")
461.             # self.label_face.setPixmap(pixmap)
462.             # self.label_face.setScaledContents(True)
463.             name,type=file.split(".")
464.             time.sleep(1)
465.             print("Sender 正在运行")
466.             if type=="jpg" or type =="png" or type=="jpeg" or type =="gif":
```

```
                self.mode=1
                with open(file, "rb") as f:
                    image_base64 = base64.b64encode(f.read()).decode("utf-8")
                topic = f"/{sender_product_key}/{sender_device_name}/user/update"
                payload = f'{{"type":"image","data":"{image_base64}"}}'
                lk_sender.publish_topic(topic, payload)
            elif type=="mp3" or type=="wav":
                self.mode=2
                with open(file,'rb') as f:
                    audio_base64 = base64.b64encode(f.read()).decode("utf-8")
                topic = f"/{sender_product_key}/{sender_device_name}/user/update"
                payload = f'{{"type":"audio","data":"{audio_base64}"}}'
                lk_sender.publish_topic(topic, payload)
            elif type=="txt":
                self.mode=3
                with open(file,'rb') as f:
                    word_base64=base64.b64encode(f.read()).decode('utf-8')
                topic = f"/{sender_product_key}/{sender_device_name}/user/update"
                payload = f'{{"type":"word","data":"{word_base64}"}}'
                lk_sender.publish_topic(topic, payload)
            elif type=='mp4':
                self.mode=4
                with open(file,'rb') as f:
                    video_base64=base64.b64encode(f.read()).decode('utf-8')
                topic = f"/{target}/{sender_device_name}/user/update"
                payload = f'{{"type":"mp4","data":"{video_base64}"}}'
                lk_sender.publish_topic(topic, payload)
            print("上传成功")
            time.sleep(2)
            lk_sender.disconnect()
        else:
            self.filename.setText("文件名不正确")
    def act_Receiver(self):
        logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(levelname)s - %(messag
e)s")
        logger = logging.getLogger(__name__)
        # Receiver 设备信息
        receiver_product_key = self.product_key_Edit_2.text()
        receiver_device_name = self.device_name_Edit_2.text()
        receiver_device_secret = self.device_secret_Edit_2.text()
        lk_receiver = linkkit.LinkKit(
            host_name="cn-shanghai",
            product_key=receiver_product_key,
            device_name=receiver_device_name,
            device_secret=receiver_device_secret,
        )
        lk_receiver.config_mqtt(
```

```
            endpoint=f"{receiver_product_key}.iot-as-mqtt.cn-shanghai.aliyuncs.com",
            port=8883,
            protocol="MQTTv311",
            transport="TCP",
            secure="TLS",
            keep_alive=60,
            clean_session=True,
            max_inflight_message=20,
            max_queued_message=0,
            auto_reconnect_min_sec=1,
            auto_reconnect_max_sec=60,
            cadata=None,
        )
        topic = f"/{receiver_product_key}/{receiver_device_name}/user/get"
        lk_receiver.subscribe_topic(topic, qos=1)
        logger.info(f"Receiver: Subscribed to topic: {topic}")
        def on_topic_message(topic, payload, qos, userdata): # 当接收到数据时，会调用此函数
            print(payload)
            pixmap=QPixmap('green.png')
            self.face.setScaledContents(True)
            self.face.setPixmap(pixmap)
            if isinstance(payload,bytes):
                data = json.loads(payload.decode("utf-8"))
            self.type=data.get("type")
            context=data.get("data")
            if self.type=="image":
                img_data=base64.b64decode(context)
                image=Image.open(BytesIO(img_data))
                if os.path.exists('temp.jpg'):
                    os.remove('temp.jpg')
                image.save('temp.jpg')
                pass
            elif self.type=="audio":
                audio_data=base64.b64decode(context)
                audio_buffer=BytesIO(audio_data)
                if os.path.exists('temp.mp3'):
                    os.remove('temp.mp3')
                with open('temp.mp3','wb',buffering=1024 * 1024) as f:
                    f.write(audio_buffer.getvalue())
                pass
            elif self.type=="word":

                word_data=base64.b64decode(context).decode('utf-8')

                text_buffer=StringIO(word_data)
                if os.path.exists('temp.txt'):
                    os.remove('temp.txt')
```

```
                with open('temp.txt','w',encoding='utf-8',buffering=1024 * 1024) as f:
                    f.write(text_buffer.getvalue())
                pass
            elif self.type=="mp4":
                video_data=base64.b64decode(context)
                with open('temp.mp4','wb') as f:
                    f.write(video_data)
                pass
            self.face.clear()
        lk_receiver.on_topic_message = on_topic_message
        lk_receiver.connect_async()
        time.sleep(3)  # 等待连接完成
        self.is_running=True
        self.act.setStyleSheet("color:red;")
        while self.is_running:
            time.sleep(5)
        lk_receiver.disconnect()
        self.act.setStyleSheet("color:black;")
        print("已关闭")
    def actF(self):
        self.receiver_thread = threading.Thread(target=self.act_Receiver)
        #sender_thread.start()
        self.receiver_thread.start()
    def showText(self):
        self.data_n.setText(self.data_n.text())
        self.filename.setText(self.filename.text())
        self.time.setText(self.time.text())
        self.fps.setText(self.fps.text())
    def pauseF(self):
        self.engine.stop()
    def music(self):
        vol=self.volumn.value()
        speed=self.speed.value()
        text=self.textinput.toPlainText()
        self.engine.setProperty('rate', speed)  # 设置语速
        self.engine.setProperty('volume', vol)  # 设置音量
        voices = self.engine.getProperty('voices')
        id=''
        for voice in voices:
            if 'ZH-CN' in voice.id:
                id=voice.id
                break
        self.engine.setProperty('voice',id)
        self.engine.say(text)
        self.engine.save_to_file(text,'text.mp3')
        if os.path.exists('word.txt'):
            os.remove('word.txt')
```

```
        with open('word.txt','w',encoding='utf-8') as f:
            f.write(text)
        self.engine.runAndWait()
        self.engine.stop()
    def clearF(self):
        def del_layout(self,layout):
            if layout is not None:
                for i in reversed(range(layout.count())):
                    widget = layout.itemAt(i).widget()
                    if widget is not None:
                        widget.deleteLater()  # 删除控件
                layout.deleteLater()
        del_layout(self,self.data_p.layout())
    def get_data(self):
        n=int(self.data_n.text())
        mode=self.data_feature.currentIndex()
        if mode==0:
            data=np.random.normal(0,1,n)
        elif mode==1:
            data=np.random.binomial(10,0.5,size=n)
        elif mode==2:
            data=np.random.exponential(2,size=n)
        elif mode==3:
            lam=10
            data=np.random.poisson(lam,n)
        elif mode==4:
            t=np.linspace(1,n,n)
            data=np.sin(t)
        fig,ax=plt.subplots(figsize=(10,10))
        plt.tight_layout()
        plt.plot(data)
        canvas=FigureCanvas(fig)
        self.fig=fig
        layout=QtWidgets.QVBoxLayout(self.data_p)
        layout.addWidget(canvas)
        self.data_p.setLayout(layout)
        canvas.draw()
        fig.savefig('fig.jpg')
if __name__=="__main__":
    import sys
    app=QtWidgets.QApplication(sys.argv)
    Server=QtWidgets.QMainWindow()
    ui=Ui_MainWindow()
    ui.setupUi(Server)
    Server.show()
    sys.exit(app.exec())
```