项目源代码文档

main.py

```
#!/usr/bin/python3
from flask import Flask, render_template_string, request
import threading
import time
from display_manager import DisplayManager
import signal
import sys
import json
import os

app = Flask(__name__)
display_manager = DisplayManager()
THRESHOLD_FILE = "/tmp/energy_threshold.json"   # 共享存储文件

# 网页控制界面 HTML 模板
HTML_TEMPLATE = """
<!DOCTYPE html>
<html>
<head>
    <title>树莓派 OLED 控制</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .container { max-width: 600px; margin: 0 auto; }
        button { padding: 10px 15px; margin: 5px; }
    </style>
</head>
<body>
    <div class="container">
        <h1>OLED 显示屏控制</h1>
        <p>当前状态: {{ status }}</p>
        <p>最后更新: {{ last_update }}</p>

        <h2>控制命令</h2>
        <button onclick="sendCommand('start')">启动显示</button>
        <button onclick="sendCommand('stop')">停止显示</button>
        <button onclick="sendCommand('refresh')">刷新电量</button>

        <h2>电量警告阈值</h2>
```

```
            <input type="number" id="threshold_input" value="100" placeholder="阈值(kWh)">
            <button onclick="setThreshold()">设置</button>

        </div>

        <script>
            function sendCommand(cmd) {
                fetch('/command?cmd=' + cmd)
                    .then(response => response.text())
                    .then(data => alert(data))
                    .catch(error => alert('Error: ' + error));
            }

            function setThreshold() {
                const threshold = document.getElementById('threshold_input').value;
                fetch('/set_threshold', {
                    method: 'POST',
                    headers: {'Content-Type': 'application/x-www-form-urlencoded'},
                    body: 'threshold=' + threshold
                }).then(response => response.text())
                  .then(data => alert(data));
            }
        </script>
</body>
</html>
"""

@app.route('/')
def control_panel():
    status = "运行中" if display_manager.process else "已停止"
    return render_template_string(HTML_TEMPLATE,
                                  status=status,
                                  last_update=time.strftime("%Y-%m-%d %H:%M:%S"))

@app.route('/command')
def handle_command():
    cmd = request.args.get('cmd', '')
    if cmd == 'start':
        display_manager.start()
        return "显示已启动"
    elif cmd == 'stop':
        display_manager.cleanup()
        return "显示已停止"
    elif cmd == 'refresh':
            # 这里需要实现电量刷新逻辑
            return "电量已刷新"
        return "无效命令"

if not os.path.exists(THRESHOLD_FILE):
    with open(THRESHOLD_FILE, 'w') as f:
        json.dump({"threshold": 100}, f)  # 默认阈值 100

@app.route('/set_threshold', methods=['POST'])
def set_threshold():
    try:
        threshold = float(request.form.get('threshold'))
        # 保存到共享文件
        with open(THRESHOLD_FILE, 'w') as f:
            json.dump({"threshold": threshold}, f)
        return "阈值已更新为 {}kWh".format(threshold)
    except Exception as e:
        return "错误: " + str(e)

def run_flask():
    app.run(host='0.0.0.0', port=5000, threaded=True)

def signal_handler(sig, frame):
    print("\n 正在关闭服务器...")
    display_manager.cleanup()
    sys.exit(0)

if __name__ == "__main__":
    # 注册信号处理
    signal.signal(signal.SIGINT, signal_handler)

    # 启动显示
    display_manager.start()

    # 在单独线程中启动 Flask
    flask_thread = threading.Thread(target=run_flask)
    flask_thread.daemon = True
    flask_thread.start()

    print("服务器已启动，请访问 http://树莓派 IP:5000")
    print("按 CTRL+C 退出")

    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        signal_handler(None, None)
```

```

```

screen.py

```
#!/usr/bin/python3
from luma.core.interface.serial import i2c
from luma.oled.device import sh1106
from luma.core.render import canvas
from PIL import ImageFont
import time
import getElectricityBalance
import signal
import sys
import json
import os

class DisplayController:
    def __init__(self):
        # 初始化设备
        self.threshold_file = "/tmp/energy_threshold.json"
        self.current_threshold = 100  # 默认值
        self.serial = i2c(port=1, address=0x3C)
        self.device = sh1106(self.serial)
        self.running = True
        self.last_balance = None
        self.first_load = True  # 首次加载标志

        # 加载字体
        try:
            self.font = ImageFont.truetype('wqy-microhei.ttc', 14)
        except:
            self.font = None

        # 注册信号处理
        signal.signal(signal.SIGINT, self.signal_handler)
        signal.signal(signal.SIGTERM, self.signal_handler)

    def signal_handler(self, signum, frame):
        self.running = False
        self.device.clear()
        sys.exit(0)
```

```python

    def read_threshold(self):
        """从共享文件读取最新阈值"""
        try:
            with open(self.threshold_file, 'r') as f:
                data = json.load(f)
                self.current_threshold = float(data["threshold"])
        except:
            pass  # 保持当前阈值不变

    def show_loading(self):
        """只在首次显示加载状态"""
        if self.first_load:
            with canvas(self.device) as draw:
                draw.rectangle(self.device.bounding_box, outline="white", fill="black")
                draw.text((10, 30), "Loading...", fill="white", font=self.font)
            time.sleep(0.5)  # 确保用户能看到加载状态

    def show_data(self, balance):
        self.read_threshold()  # 每次显示前读取最新阈值
        """显示时间和电量数据"""
        with canvas(self.device) as draw:
            draw.rectangle(self.device.bounding_box, outline="white", fill="black")
            draw.text((5, 10), time.strftime("%H:%M:%S"), fill="white", font=self.font)
            draw.text((10, 25), "Current Power:", fill="white", font=self.font)
            draw.text((40, 40), f"{balance:.2f} kWh", fill="white", font=self.font)
            # 动态阈值判断
            if balance < self.current_threshold:
                warning_text = "!!!!!"
                draw.text((97, 10), warning_text, fill="white", font=self.font)

    def run(self):
        while self.running:
            try:
                # 首次加载显示状态
                self.show_loading()

                # 只在首次或需要刷新时获取电量
                if self.first_load or self.should_refresh_balance():
                    balance = getElectricityBalance.get_electricity_balance()
                    self.last_balance = balance
                    self.first_load = False

                # 显示数据（时间会自动更新）
                self.show_data(self.last_balance)

                time.sleep(1)  # 更新时间间隔

            except Exception as e:
                print(f"显示错误: {e}")
                self.first_load = True  # 出错后下次重新加载
                time.sleep(5)

    def should_refresh_balance(self):
        """自定义电量刷新条件（例如每 10 分钟）"""
        # 示例：每 600 秒刷新一次（10 分钟）
        return hasattr(self, 'last_refresh_time') and \
               (time.time() - self.last_refresh_time) > 600

if __name__ == "__main__":
    controller = DisplayController()
    controller.run()

```

getElectricityBalance.py

```
# -*- coding: utf-8 -*-
import time
import subprocess
import pytesseract
from PIL import Image
import mss
import re

# 配置 Tesseract 路径
pytesseract.pytesseract.tesseract_cmd = '/usr/bin/tesseract'

def click_at(x, y):
    """使用 xdotool 模拟点击"""
    subprocess.run(f"xdotool mousemove {x} {y} click 1", shell=True)

def get_electricity_balance():
    try:
        # 点击微信第一个窗口“南林智慧校园”
        click_at(661, 306)
        time.sleep(2)
        # 点击“一卡通”
        click_at(904, 862)
        time.sleep(2)
        # 点击“一卡通自助服务”
```

```python
        click_at(892, 710)
        time.sleep(2)
        # 点击“缴电费”
        click_at(712, 572)
        time.sleep(2)
        # 选择“本部玄武公寓用电”
        click_at(354, 472)
        time.sleep(2)
        # 楼栋选择 20 栋
        # 点击“楼栋”选项
        click_at(1801, 741)
        time.sleep(2)
        # 键盘输入“2”，选择 20 栋
        subprocess.run(['xdotool', 'key', '2'])
        time.sleep(1)
        subprocess.run(['xdotool', 'key', 'Return'])
        time.sleep(2)
        click_at(1803, 862)
        subprocess.run("xdotool type '201202'", shell=True)
        time.sleep(1)
        # 点击并输入房间号“201202”
        click_at(1493, 798)
        time.sleep(2)
        power_bbox = (1621, 892, 1854, 935)
        # 处理电量数据
        text = extract_text_from_coords(power_bbox)
        match = re.search(r'(\d+\.\d+)', text)
        if match:
            balance = float(match.group(1))
        else:
            raise ValueError("No balance found in text")

        return balance

    except Exception as e:
        print(f"获取电量失败: {str(e)}")
        return 0.0  # 返回默认值

if __name__ == "__main__":
    try:
        balance = get_electricity_balance()
        print(f"当前电量: {balance} 度")
    except Exception as e:
        print(f"操作失败: {str(e)}")

```

auto_cutoff_if_empty.py

```python
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import time
from datetime import datetime

# 硬件配置
SENSOR_PIN = 27        # 红外传感器 GPIO(BCM27)
SERVO_PIN = 18         # 舵机 GPIO(BCM18)
PWM_FREQ = 50          # SG90 频率(Hz)
UP_ANGLE = 180         # 有人角度
DOWN_ANGLE = 0         # 无人角度
DEBOUNCE_TIME = 0.5  # 防抖时间(秒)


class ServoControl:
    def __init__(self):
        # 初始化设置
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(SENSOR_PIN, GPIO.IN)
        GPIO.setup(SERVO_PIN, GPIO.OUT)

        # PWM 初始化
        self.pwm = GPIO.PWM(SERVO_PIN, PWM_FREQ)
        self.pwm.start(0)
        self.current_angle = None  # 初始角度未设置
        self.last_state = None     # 初始状态未设置

        # 传感器预热
        print("传感器预热中(30 秒)...")
        time.sleep(30)
        print("系统就绪，开始检测")

    def set_angle(self, angle):
        """设置舵机角度并保持"""
        if angle == self.current_angle:
            return  # 已在目标角度

        # 计算占空比 (0°=2.5%, 180°=12.5%)
        duty = angle / 18 + 2.5
        self.pwm.ChangeDutyCycle(duty)
        time.sleep(5)  # 确保转动完成
        self.pwm.ChangeDutyCycle(0)  # 停止 PWM 防止抖动
        self.current_angle = angle
        print(f"角度设置: {angle}°")

    def run(self):
        try:
            # 获取初始状态并设置初始角度
            initial_state = GPIO.input(SENSOR_PIN)
            if initial_state == 1:  # 初始有人
                self.set_angle(UP_ANGLE)
            else:  # 初始无人
                self.set_angle(DOWN_ANGLE)
            self.last_state = initial_state

            while True:
                current_state = GPIO.input(SENSOR_PIN)
                print( current_state)
                # 只有状态变化时才动作
                if current_state != self.last_state:
                    if current_state == 1:  # 从无人变有人
                        self.set_angle(UP_ANGLE)
                    else:  # 从有人变无人
                        self.set_angle(DOWN_ANGLE)
                    self.last_state = current_state

                time.sleep(DEBOUNCE_TIME)  # 检测间隔

        except KeyboardInterrupt:
            print("\n 正在停止...")
        finally:
            self.pwm.stop()
            GPIO.cleanup()
            print("系统关闭")

if __name__ == "__main__":
    controller = ServoControl()
    controller.run()

```

night_power_saving_control.py

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import time
from datetime import datetime

# 硬件配置
```

```
SENSOR_PIN = 27        #  红外传感器 GPIO(BCM27)
SERVO_PIN = 18         #  舵机 GPIO(BCM18)
PWM_FREQ = 50          # SG90 频率(Hz)
UP_ANGLE = 180         #  有人角度
DOWN_ANGLE = 0         #  无人角度
DEBOUNCE_TIME = 0.5   #  防抖时间(秒)

def is_night_time():
    now = datetime.now().time()
    return now.hour >= 23 or now.hour < 6

class ServoControl:
    def __init__(self):
        #  初始化设置
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(SENSOR_PIN, GPIO.IN)
        GPIO.setup(SERVO_PIN, GPIO.OUT)

        # PWM 初始化
        self.pwm = GPIO.PWM(SERVO_PIN, PWM_FREQ)
        self.pwm.start(0)
        self.current_angle = None   #  初始角度未设置
        self.last_state = None      #  初始状态未设置

        #  传感器预热
        print("传感器预热中(30 秒)...")
        time.sleep(30)
        print("系统就绪，开始检测")

    def set_angle(self, angle):
        """设置舵机角度并保持"""
        if angle == self.current_angle:
            return   #  已在目标角度

        #  计算占空比  (0°=2.5%, 180°=12.5%)
        duty = angle / 18 + 2.5
        self.pwm.ChangeDutyCycle(duty)
        time.sleep(5)   #  确保转动完成
        self.pwm.ChangeDutyCycle(0)   #  停止 PWM 防止抖动
        self.current_angle = angle
        print(f"角度设置: {angle}°")

    def run(self):
        try:
            #  获取初始状态并设置初始角度
            initial_state = GPIO.input(SENSOR_PIN)
            if initial_state == 1:  # 初始有人
                self.set_angle(UP_ANGLE)
            else:  # 初始无人
                self.set_angle(DOWN_ANGLE)
            self.last_state = initial_state

            while True:
                current_state = GPIO.input(SENSOR_PIN)
                print( current_state)
                # 只有状态变化时才动作
                if current_state != self.last_state:
                    if current_state == 1:  # 从无人变有人
                        self.set_angle(UP_ANGLE)
                    else:  # 从有人变无人
                        self.set_angle(DOWN_ANGLE)
                    self.last_state = current_state

                time.sleep(DEBOUNCE_TIME)  # 检测间隔

        except KeyboardInterrupt:
            print("\n 正在停止...")
        finally:
            self.pwm.stop()
            GPIO.cleanup()
            print("系统关闭")

if __name__ == "__main__":
    while True:
        if is_night_time():
            controller = ServoControl()
            controller.run()


```