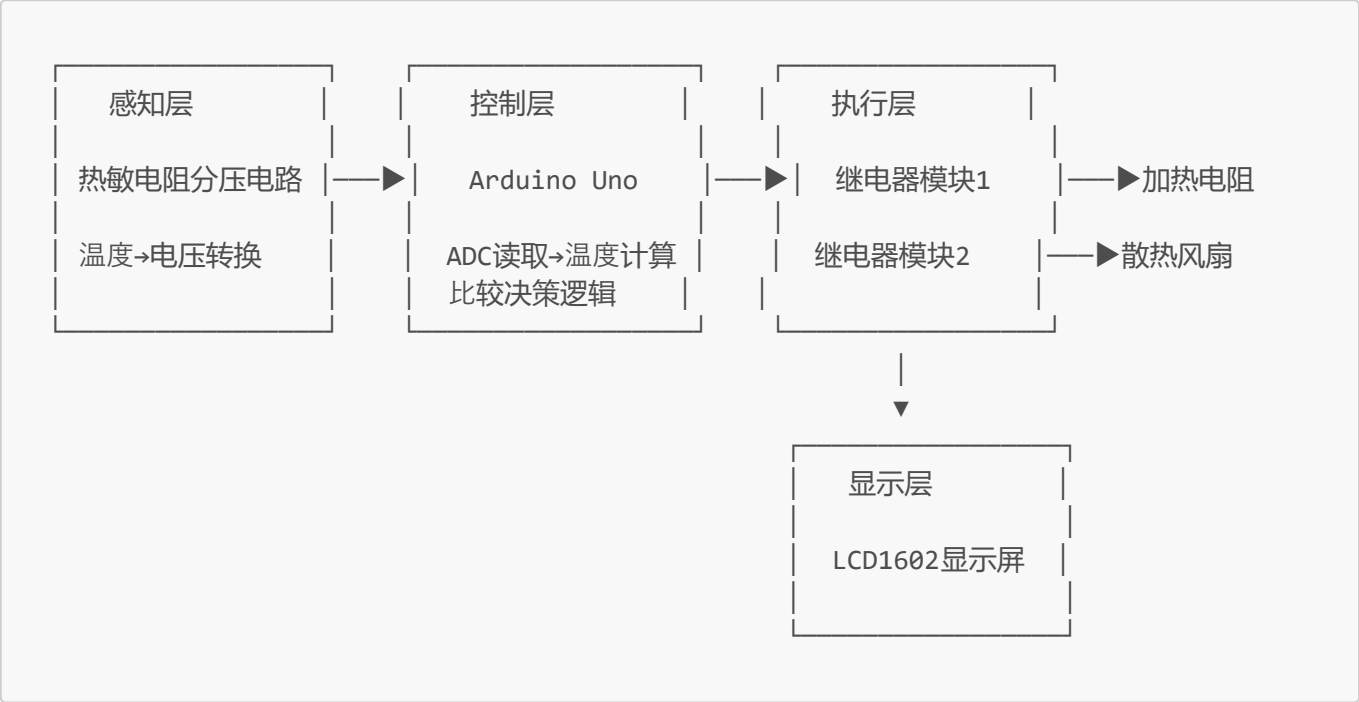
迷你温室自动温控系统（自制热敏电阻传感器版）详细设计方案

项目概述 这是一个完整的DIY温控系统，使用自制的热敏电阻温度传感器，通过Arduino控制加热和制冷设备，将密闭瓶内温度精确控制在设定值。

第一步：详细系统架构设计

1.1 系统框图



1.2 信号流程详解

1. 温度采集：热敏电阻阻值随温度变化 → 分压电路产生电压变化
2. 信号转换：Arduino ADC将0-5V模拟电压转换为0-1023数字值
3. 温度计算：使用Steinhart-Hart方程将电阻值转换为温度值
4. 决策控制：当前温度与设定温度比较，决定执行器状态
5. 执行输出：通过继电器控制加热/制冷设备
6. 状态显示：LCD实时显示温度和系统状态

第二步：详细元器件清单与规格

2.1 核心元器件清单

类别	元件	规格参数	数量	备注说明
控制核心	Arduino Uno R3	ATmega328P, 5V, 16MHz	1	主控制器，14个数字IO，6个模拟输入
温度传感	NTC热敏电阻	10KΩ, B值3950, MF52型	1	负温度系数，电阻随温度升高而降低

类别	元件	规格参数	数量	备注说明
分压电阻	金属膜电阻	10KΩ, 1%精度, 1/4W	1	精度影响测温准确性
加热元件	水泥电阻	5-10Ω, 5W-10W	1	功率计算：$P=V^2/R$，注意散热
制冷元件	DC散热风扇	5V, 0.1A-0.2A	1	可用电脑CPU风扇
驱动模块	继电器模块	5V, 10A/250VAC, 光耦隔离	2	注意触发电平（高/低电平有效）
显示设备	LCD1602+I2C	16x2字符, I2C接口	1	简化接线，只需4根线
连接材料	面包板	830孔, 带电源总线	1	用于原型搭建
连接线	杜邦线	公对公, 公对母, 母对母	各20根	不同长度备用
实验容器	玻璃瓶	1-2升, 带密封盖	1	透明便于观察
电源供应	USB电源	5V, 2A以上	1	为整个系统供电
固定材料	洞洞板	5x7cm	1	固定瓶内元件
辅助材料	热缩管, 焊锡, 电工胶带	-	若干	绝缘保护

2.2 元器件选型说明

热敏电阻选择：

- 推荐NTC-MF52-103（10KΩ, B=3950K）
- 工作温度范围：-50℃ to +125℃
- 精度：±1%（25℃时）
- 封装：环氧树脂，直径3-5mm

水泥电阻功率计算：

- 使用5V电源：$P = V^2/R = 25/10 = 2.5W$（选择5W电阻）
- 使用12V电源：$P = 144/10 = 14.4W$（选择20W电阻）

继电器规格：

- 线圈电压：5V DC
- 接触容量：10A/250VAC, 10A/30VDC
- 触发电流：15-20mA

第三步：详细硬件连接步骤

3.1 热敏电阻分压电路详细搭建

电路原理：

```
+5V ——— [R_fixed 10KΩ] ——— A0 ——— [R_ntc 10KΩ] ——— GND
                          ↑
                      电压测量点
```

具体步骤：

1. 在面包板上插入10K固定电阻
2. 将NTC热敏电阻与固定电阻串联
3. 连接点用杜邦线引至Arduino A0引脚
4. 固定电阻另一端接5V，热敏电阻另一端接GND

热敏电阻安装注意事项：

- 使用热缩管包裹引线连接处，防止短路
- 在瓶内用热熔胶固定热敏电阻头部，确保与空气充分接触
- 引线从瓶盖钻孔引出，孔洞用硅胶密封

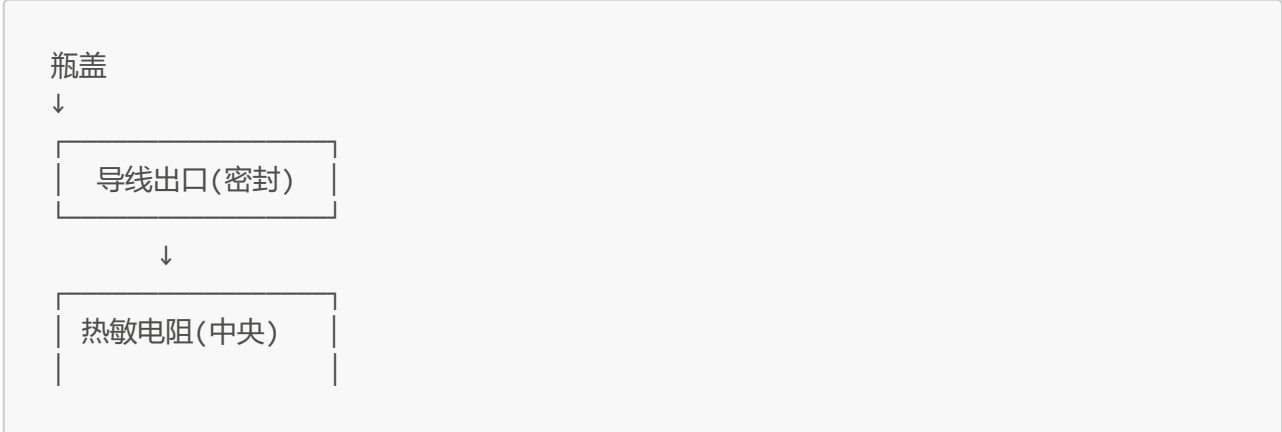
3.2 完整接线表

起点	终点	线色建议	备注
电源部分			
电源适配器 +5V	Arduino Vin	红色	总电源输入
电源适配器 GND	Arduino GND	黑色	共地
温度传感器			
分压电路中间点	Arduino A0	黄色	温度信号输入
分压电路 5V端	Arduino 5V	红色	传感器供电
分压电路 GND端	Arduino GND	黑色	传感器地线
LCD显示屏			
LCD VCC	Arduino 5V	红色	显示屏供电
LCD GND	Arduino GND	黑色	显示屏地线
LCD SDA	Arduino A4	绿色	I2C数据线
LCD SCL	Arduino A5	蓝色	I2C时钟线
继电器1（加热）			
Relay1 VCC	Arduino 5V	红色	继电器供电

起点	终点	线色建议	备注
Relay1 GND	Arduino GND	黑色	继电器地线
Relay1 IN	Arduino D6	橙色	控制信号
继电器2（制冷）			
Relay2 VCC	Arduino 5V	红色	继电器供电
Relay2 GND	Arduino GND	黑色	继电器地线
Relay2 IN	Arduino D7	紫色	控制信号
执行器电源			
独立电源 +5V	Relay1 COM	红色	加热器电源
独立电源 +5V	Relay2 COM	红色	风扇电源
独立电源 GND	水泥电阻另一端	黑色	加热器回路
独立电源 GND	风扇负极	黑色	风扇回路
执行器输出			
Relay1 NO	水泥电阻一端	棕色	加热控制
Relay2 NO	风扇正极	灰色	风扇控制

3.3 瓶内元件安装指南

1. 热敏电阻安装：
    - 用热熔胶将热敏电阻固定在洞洞板中央
    - 确保感温头暴露在空气中，不被其他元件遮挡
    - 引线留出足够长度，方便瓶盖密封
2. 加热电阻安装：
    - 水泥电阻引脚焊接延长线
    - 用支架固定，避免接触瓶壁或其他元件
    - 考虑散热，周围留出空间
3. 整体布局：



```
  │  加热电阻(侧边)  │
  └──────────────────┘
```

第四步：完整程序代码（详细注释版）

```
/
  迷你温室自动温控系统
  使用自制热敏电阻传感器
  控制加热器和风扇维持设定温度
 /

include <Wire.h>
include <LiquidCrystal_I2C.h>

// ==================== 引脚定义 ====================
define HEATER_RELAY_PIN 6     // 加热继电器控制引脚
define FAN_RELAY_PIN 7        // 风扇继电器控制引脚
define THERMISTOR_PIN A0      // 热敏电阻模拟输入引脚

// ==================== 热敏电阻参数 ====================
// 注意：这些参数需要根据实际使用的热敏电阻调整
define SERIES_RESISTOR 10000.0    // 分压固定电阻阻值（欧姆）
define NOMINAL_RESISTANCE 10000.0 // 热敏电阻在25°C时的标称阻值（欧姆）
define NOMINAL_TEMP 25.0          // 标称温度（摄氏度）
define B_COEFFICIENT 3950.0       // B值系数（根据热敏电阻规格）

// ==================== 温度控制参数 ====================
float targetTemp = 25.0;           // 目标温度（摄氏度）
float hysteresis = 0.5;            // 迟滞范围，防止频繁开关（摄氏度）
const unsigned long SAMPLE_INTERVAL = 1000; // 采样间隔（毫秒）

// ==================== LCD显示设置 ====================
// I2C地址可能是0x27或0x3F，需要根据实际模块调整
LiquidCrystal_I2C lcd(0x27, 16, 2); // 设置LCD地址和尺寸

// ==================== 温度读数滤波 ====================
const int NUM_SAMPLES = 5;          // 移动平均滤波的样本数
float temperatureSamples[NUM_SAMPLES];
int sampleIndex = 0;

// ==================== 系统状态变量 ====================
enum SystemState { HEATING, COOLING, IDLE };
SystemState currentState = IDLE;
unsigned long lastSampleTime = 0;

void setup() {
  // 初始化串口通信
  Serial.begin(115200);
  Serial.println(F("========================================="));
  Serial.println(F("    迷你温室自动温控系统启动"));
  Serial.println(F("    使用自制热敏电阻传感器"));
```

```
  Serial.println(F("========================================"));

  // 初始化LCD显示屏
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print(F("Temp:       C"));
  lcd.setCursor(0, 1);
  lcd.print(F("Target:     C"));

  // 初始化继电器控制引脚
  pinMode(HEATER_RELAY_PIN, OUTPUT);
  pinMode(FAN_RELAY_PIN, OUTPUT);

  // 初始状态：关闭所有执行器
  // 注意：根据继电器模块逻辑，可能是HIGH或LOW关闭
  digitalWrite(HEATER_RELAY_PIN, HIGH); // 假设高电平关闭
  digitalWrite(FAN_RELAY_PIN, HIGH);    // 假设高电平关闭

  // 初始化温度样本数组
  for (int i = 0; i < NUM_SAMPLES; i++) {
    temperatureSamples[i] = readRawTemperature();
  }

  Serial.println(F("系统初始化完成"));
  Serial.println(F("ADC值\t电压(V)\t电阻(Ω)\t温度(°C)"));
  Serial.println(F("----------------------------------------"));
}

void loop() {
  // 定时采样，避免过于频繁的读取
  unsigned long currentTime = millis();
  if (currentTime - lastSampleTime >= SAMPLE_INTERVAL) {
    lastSampleTime = currentTime;

    // 读取并处理温度
    float currentTemp = readFilteredTemperature();

    // 更新显示
    updateDisplay(currentTemp);

    // 执行温度控制
    controlTemperature(currentTemp);

    // 串口输出调试信息
    printDebugInfo(currentTemp);
  }
}

// ==================== 温度读取函数 ====================

/
  读取原始温度值（单次测量）
 /
```

```
float readRawTemperature() {
  // 读取模拟值 (0-1023)
  int adcValue = analogRead(THERMISTOR_PIN);

  // 转换为电压值 (0-5V)
  float voltage = adcValue  (5.0 / 1023.0);

  // 计算热敏电阻当前阻值
  // 公式: R_ntc = (固定电阻  电压) / (5V - 电压)
  float ntcResistance = (SERIES_RESISTOR  voltage) / (5.0 - voltage);

  // 使用Steinhart-Hart方程计算温度 (摄氏度)
  float temperature = calculateTemperature(ntcResistance);

  return temperature;
}

/
  使用Steinhart-Hart方程计算温度
 /
float calculateTemperature(float resistance) {
  // Steinhart-Hart方程: 1/T = 1/T0 + (1/B)  ln(R/R0)
  // 其中: T为开尔文温度, R为当前电阻, T0为标称温度(开尔文), R0为标称电阻, B为B值

  float steinhart;
  steinhart = resistance / NOMINAL_RESISTANCE;        // (R/R0)
  steinhart = log(steinhart);                         // ln(R/R0)
  steinhart /= B_COEFFICIENT;                         // (1/B)  ln(R/R0)
  steinhart += 1.0 / (NOMINAL_TEMP + 273.15);         // + (1/T0)
  steinhart = 1.0 / steinhart;                        // 求倒数得到开尔文温度
  float temperatureC = steinhart - 273.15;            // 转换为摄氏度

  return temperatureC;
}

/
  读取滤波后的温度值 (移动平均滤波)
 /
float readFilteredTemperature() {
  // 获取新样本
  temperatureSamples[sampleIndex] = readRawTemperature();
  sampleIndex = (sampleIndex + 1) % NUM_SAMPLES;

  // 计算平均值
  float sum = 0;
  for (int i = 0; i < NUM_SAMPLES; i++) {
    sum += temperatureSamples[i];
  }

  return sum / NUM_SAMPLES;
}

// ==================== 显示更新函数 ====================
```

```
/
  更新LCD显示
 /
void updateDisplay(float currentTemp) {
  // 显示当前温度
  lcd.setCursor(6, 0);
  if (currentTemp >= 0) {
    lcd.print(" "); // 正数前加空格对齐
  }
  lcd.print(currentTemp, 1);
  lcd.print(" ");

  // 显示目标温度
  lcd.setCursor(8, 1);
  lcd.print(targetTemp, 1);
  lcd.print(" ");

  // 显示系统状态
  lcd.setCursor(13, 0);
  switch (currentState) {
    case HEATING:
      lcd.print("H");
      break;
    case COOLING:
      lcd.print("C");
      break;
    case IDLE:
      lcd.print("-");
      break;
  }
}

// ==================== 温度控制函数 ====================

/
  温度控制逻辑
 /
void controlTemperature(float currentTemp) {
  if (currentTemp < targetTemp - hysteresis) {
    // 温度过低，开启加热，关闭制冷
    digitalWrite(HEATER_RELAY_PIN, LOW);  // 开启加热器
    digitalWrite(FAN_RELAY_PIN, HIGH);    // 关闭风扇
    currentState = HEATING;
  }
  else if (currentTemp > targetTemp + hysteresis) {
    // 温度过高，开启制冷，关闭加热
    digitalWrite(FAN_RELAY_PIN, LOW);     // 开启风扇
    digitalWrite(HEATER_RELAY_PIN, HIGH); // 关闭加热器
    currentState = COOLING;
  }
  else {
    // 温度在舒适区间，都关闭
    digitalWrite(HEATER_RELAY_PIN, HIGH);
    digitalWrite(FAN_RELAY_PIN, HIGH);
```

```
    currentState = IDLE;
  }
}

// ==================== 调试信息函数 ====================

/
  串口输出调试信息
 /
void printDebugInfo(float currentTemp) {
  // 为了获取详细数据，重新读取一次原始值
  int adcValue = analogRead(THERMISTOR_PIN);
  float voltage = adcValue  (5.0 / 1023.0);
  float ntcResistance = (SERIES_RESISTOR  voltage) / (5.0 - voltage);

  Serial.print(adcValue);
  Serial.print("\t");
  Serial.print(voltage, 3);
  Serial.print("\t");
  Serial.print(ntcResistance, 0);
  Serial.print("\t");
  Serial.print(currentTemp, 2);

  // 显示控制状态
  switch (currentState) {
    case HEATING:
      Serial.println("\t加热中");
      break;
    case COOLING:
      Serial.println("\t制冷中");
      break;
    case IDLE:
      Serial.println("\t维持中");
      break;
  }
}

// ==================== 辅助功能函数 ====================

/
  设置目标温度（可用于后续扩展）
 /
void setTargetTemperature(float newTemp) {
  if (newTemp >= 0 && newTemp <= 50) { // 安全范围检查
    targetTemp = newTemp;
    Serial.print("目标温度设置为: ");
    Serial.println(targetTemp);
  }
}

/
  设置迟滞范围（可用于后续扩展）
 /
void setHysteresis(float newHysteresis) {
```

```
  if (newHysteresis >= 0.1 && newHysteresis <= 5.0) {
    hysteresis = newHysteresis;
    Serial.print("迟滞范围设置为: ");
    Serial.println(hysteresis);
  }
}
```

第五步：详细测试与优化指南

5.1 分阶段测试流程

第一阶段：基础电路测试（1-2小时） 目标： 验证各组件基本功能

1. 电源测试：
    - 只连接Arduino和电源，检查电源指示灯
    - 用万用表测量5V引脚电压，应在4.8-5.2V范围内
2. 热敏电阻测试：

```
// 简易测试代码
void setup() { Serial.begin(9600); }
void loop() {
  int val = analogRead(A0);
  Serial.println(val);
  delay(500);
}
```

    - 观察串口数值，用手握住热敏电阻，数值应有明显变化
    - 室温下ADC值应在400-600之间（对应10K电阻）
3. LCD测试：
    - 上传示例代码，检查是否显示内容
    - 如果无显示，尝试地址0x27或0x3F

第二阶段：传感器校准（1小时） 目标： 确保温度读数准确

1. 两点校准法：
    - 冰点校准： 将热敏电阻放入冰水混合物中（0°C）
    - 室温校准： 用标准温度计测量室内温度
    - 记录两个温度点的ADC读数
2. 校准代码调整：

```
// 如果读数有系统误差，可以添加修正系数
float calibratedTemperature(float rawTemp) {
```

```
  return rawTemp  calibrationFactor + calibrationOffset;
}
```

3. 校准验证：
    - 在不同环境温度下测试，与标准温度计比较
    - 误差应小于±1°C

第三阶段：执行器测试（1小时） 目标： 验证加热和制冷控制

1. 继电器测试：

```
// 测试继电器动作
void testRelay(int pin) {
  digitalWrite(pin, LOW);  // 开启
  delay(2000);
  digitalWrite(pin, HIGH); // 关闭
  delay(2000);
}
```

2. 加热电阻测试：
    - 短暂通电（2-3秒），观察是否发热
    - 安全注意： 电阻会很烫，不要徒手触摸
3. 风扇测试：
    - 检查风扇转向和风力
    - 确保风向正确（向瓶内吹风或从瓶内抽风）

第四阶段：系统集成测试（2-3小时） 目标： 验证完整系统功能

1. 静态测试：
    - 设置目标温度高于室温，检查是否开始加热
    - 设置目标温度低于室温，检查是否开始制冷
2. 动态测试：
    - 观察温度控制过程，记录超调量和稳定时间
    - 测试系统对温度变化的响应速度
3. 长时间运行测试：
    - 连续运行2-4小时，检查系统稳定性
    - 监控元件温度，确保无过热现象

5.2 故障排查指南

温度读数异常： 现象： 温度值明显错误或不稳定 排查步骤：

1. 检查热敏电阻连接是否牢固
2. 测量分压电路电压是否在合理范围（0.5-4.5V）
3. 确认热敏电阻B值和标称电阻值正确
4. 检查是否有电磁干扰（使用屏蔽线）

执行器不工作： 现象： 加热器或风扇不启动 排查步骤：

1. 检查继电器指示灯状态
2. 测量继电器控制引脚电压
3. 检查执行器独立电源是否正常
4. 验证继电器触发电平（高/低有效）

LCD不显示： 现象： 屏幕无显示或乱码 排查步骤：

1. 检查I2C地址是否正确
2. 测量LCD背光电压
3. 检查I2C线序是否正确
4. 调节LCD对比度电位器（如果有）

5.3 高级优化建议

1. PID控制算法

```
// PID控制实现
float Kp = 2.0, Ki = 0.5, Kd = 1.0;
float error, previousError = 0, integral = 0;

void pidControl(float currentTemp) {
  error = targetTemp - currentTemp;
  integral += error;
  float derivative = error - previousError;

  float output = Kp  error + Ki  integral + Kd  derivative;

  // 根据输出值控制执行器
  if (output > 1.0) {
    // 加热
    analogWrite(HEATER_PIN, constrain(output, 0, 255));
  } else if (output < -1.0) {
    // 制冷
    analogWrite(FAN_PIN, constrain(-output, 0, 255));
  }

  previousError = error;
}
```

2. 数据记录功能

```
// SD卡数据记录
include <SD.h>
```

```
File dataFile;

void logData(float temp) {
  dataFile = SD.open("datalog.txt", FILE_WRITE);
  if (dataFile) {
    dataFile.print(millis());
    dataFile.print(",");
    dataFile.println(temp);
    dataFile.close();
  }
}
```

3. 远程监控

```
// 蓝牙或WiFi监控
include <SoftwareSerial.h>
SoftwareSerial bt(10, 11); // RX, TX

void setup() {
  bt.begin(9600);
}

void loop() {
  if (bt.available()) {
    String command = bt.readString();
    // 处理远程命令
  }
}
```

5.4 安全注意事项

1. 电气安全：
    - 所有高压部分必须绝缘处理
    - 电源线规格要满足电流要求
    - 设置保险丝或断路器保护
2. 防火安全：
    - 加热电阻周围无易燃物
    - 设置温度上限保护
    - 首次长时间运行要有人看管
3. 操作安全：
    - 明显标识高温部件
    - 提供紧急停止功能
    - 定期检查线路老化