使用 **iOS + BLE + Arduino** 获取温度传感器

Phodal Huang

October 24, 2017

玩点什么：https://www.wandianshenme.com

目录

步骤 1：基础知识 3

Arduino 3

低功耗蓝牙 3

步骤 2：Arduino 搭建 4

Arduino 硬件 4

Arduino 代码 5

步骤 3：iOS 项目 8

BLEManager 9

View Controller 14

步骤 4：结论 16

玩点什么：https://www.wandianshenme.com

原文链接:https://www.wandianshenme.com/play/arduino-ble-ios-read-temperature-sensor

今天我们将创造出一个很酷的玩法，温度传感器以及连接到它的 iOS 应用程序。对于这个项目，我们将使用一个连接 BLE 的 Arduino。在 iOS 端，我们将使用 CoreBluetooth。不需要担心，如果有些东西对你来说不熟悉，我会尽力给你分步说明如何创建这个。我们将使用 BLE 发送一个字符串，而不是带有温度的浮点数，这样，您可以将该电路重用于你的其他项目。在这篇文章的第一部分中，我们将讨论 Arduino，第二部分将介绍 iOS 应用。所以让我们开始吧。

步骤 1：基础知识

Arduino

对于这个项目来说，你需要一个 Arduino 开发板，它有很多个版本，但是我建议你从 Amazon 那里获得官方的入门包。在这个包中，你会有很多关于它的入门配件，以及一个很好的例子。如果您有兴趣了解有关 Arduino 的更多信息，以及如何使用它，《Arduino for Dummies》是一本很好的书，可以帮助您。

您需要的另一个硬件是，一个 Adafruit 的 BLE 扩展板。您可以使用任何其他厂商的产品，但我使用的是这一个扩展板，并且 Arduino 程序是为这个电路写的。

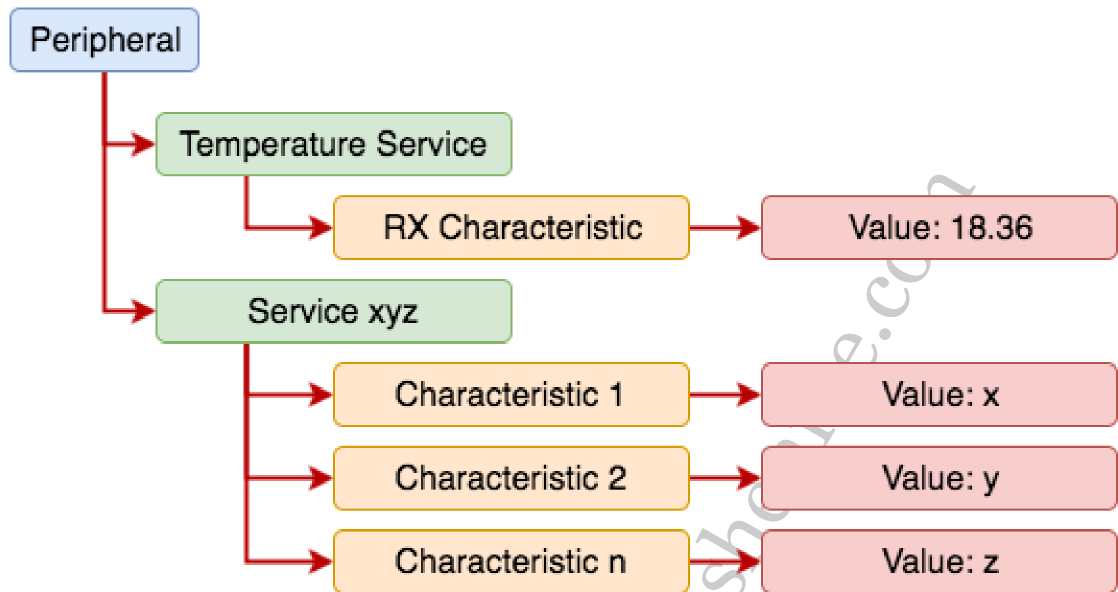
低功耗蓝牙

BLE 表示蓝牙低功耗，有时称为智能蓝牙，或蓝牙 4.1。这是一种蓝牙标准，它能使用非常低的功耗在短距离内传输小块数据。它在 IoT 项目中非常受欢迎，它允许您创建连接设备的本地网络。BLE 与标准蓝牙非常不同，如果您想了解更多信息，我会推荐一本名为《Getting Started with Bluetooth Low Energy》的书籍。

BLE 是非常结构化的，你有你的外设（设备），可以包含很多服务，一个服务可以包含许多特性（Characteristics）。这些都由 UUID 标识。特性（Characteristics）可以发送通知。这意味着，每当他们的值发生变化时，他们会通知观察者（在我们的例子中是 iOS 应用程序）。我们将在我们的项目中使用这一点，我们将在我们的特性（Characteristics）中编写温度数据，并由 iOS 应用程序自动读取。

如下图所示：

玩点什么：https://www.wandianshenme.com

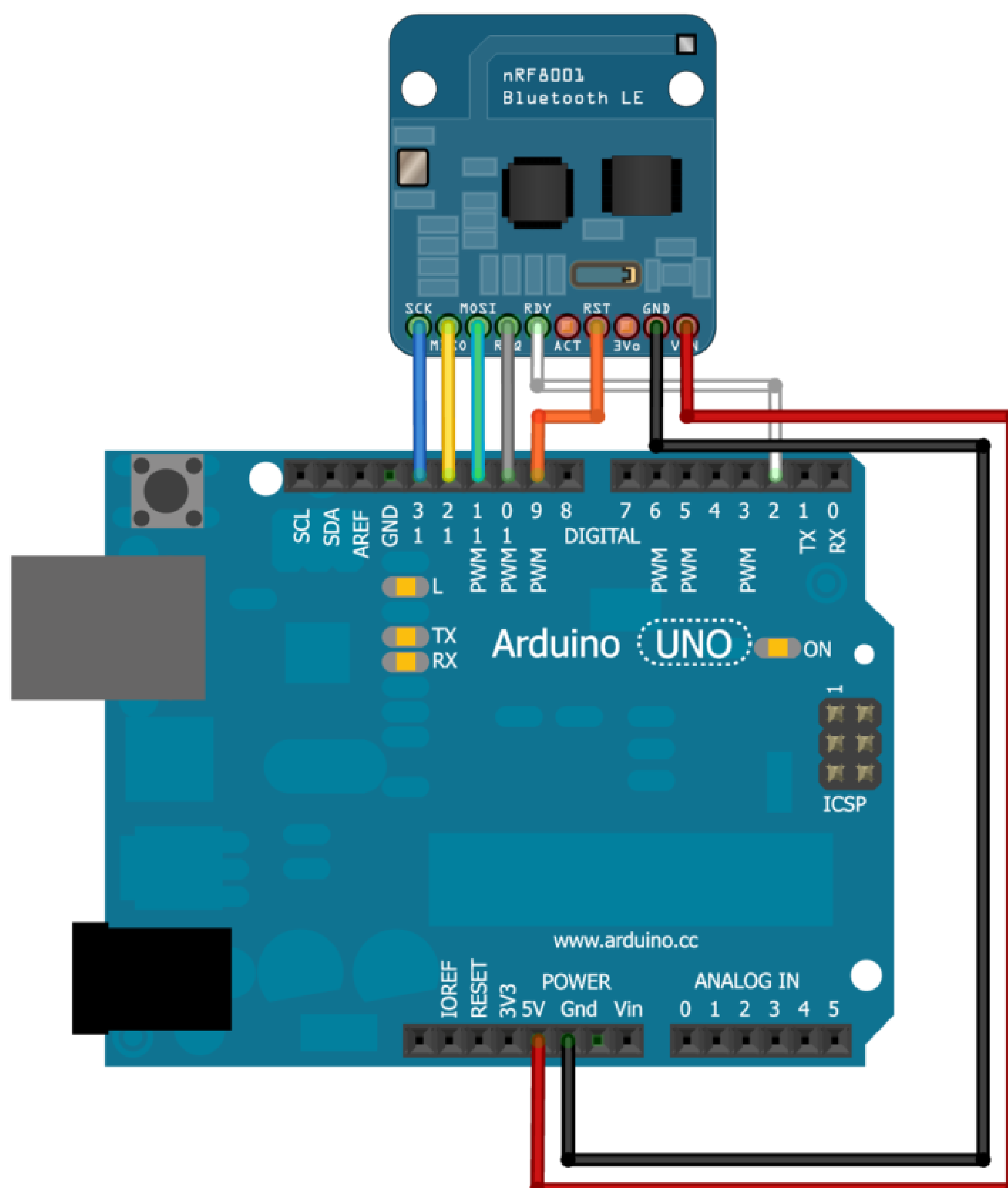


步骤 2：Arduino 搭建

Arduino 硬件

接下来，让我们来连接硬件。如果你有相同的工具包，你可以去 Adafruit 的网站，并按照说明将 BLE 连接到 Arduino。这是他们网站的连线图：

玩点什么：https://www.wandianshenme.com



您还需要一个温度传感器。如果您购买了该套件，我建议您使用套件中带的书中的示例代码找到该图。如果没有，就可以将传感器输出连接到 Arduino 上的模拟输入口 A0 。

现在我们已经有了所有的东西，让我们写一些 Arduino 代码。

Arduino 代码

要为 Arduino 编写软件，您需要安装 Arduino IDE，它可用于 Mac、Windows、Linux，因此您可以直接访问他们的网站并下载。他们的网站上有一个非常棒的指南，介绍如何设置您的 IDE：Arduino Guide MacOSX。安装 IDE 后，您需要安装 Adafruit

库。Adafruit 有一个相关的教程：Software: UART Service。

现在我们完成了硬件连接，我们引入了所需要的库。剩下的就是写一些代码，并将其上传到我们的 Arduino 板上。因为代码不是很长，我将在这里显示整个源文件：

```
1 #include <SPI.h>
2 #include "Adafruit_BLE_UART.h"
3
4 #define ADAFRUITBLE_REQ 10
5 #define ADAFRUITBLE_RDY 2
6 #define ADAFRUITBLE_RST 9
7
8 Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ,
    ADAFRUITBLE_RDY, ADAFRUITBLE_RST);
9
10 const int numReadings = 50;
11 const int initialValue = 144;
12
13 int readings[numReadings];
14 int readIndex = 0;
15 int total = initialValue * numReadings;
16
17 const int sensorPin = A0;
18
19 void setup() {
20   Serial.begin(9600);
21
22   for (int thisReading = 0; thisReading < numReadings; thisReading++) {
23     readings[thisReading] = initialValue;
24   }
25
26   BTLEserial.setDeviceName("BLETemp");
27   BTLEserial.begin();
28 }
29
30 aci_evt_opcode_t laststatus = ACI_EVT_DISCONNECTED;
31
```

```
32 void loop() {
33 BTLEserial.pollACI();
34   aci_evt_opcode_t status = BTLEserial.getState();
35   if (status != laststatus) {
36     laststatus = status;
37   }
38
39   if (status == ACI_EVT_CONNECTED) {
40     String temperatureString = averageTemperature();
41
42     uint8_t sendbuffer[20];
43     temperatureString.getBytes(sendbuffer, 20);
44     char sendbuffersize = min(20, temperatureString.length());
45
46     Serial.print(F("\n* Sending -> \"")); Serial.print((char *)sendbuffer);
         Serial.println("\"");
47
48     BTLEserial.write(sendbuffer, sendbuffersize);
49   }
50 }
51
52 String averageTemperature() {
53   int average = averageValue(sensorPin);
54   return temperature(average);
55 }
56
57 int averageValue(int inputPin) {
58   total = total - readings[readIndex];
59   readings[readIndex] = analogRead(inputPin);
60   total = total + readings[readIndex];
61   readIndex = readIndex + 1;
62
63   if (readIndex >= numReadings) {
64     readIndex = 0;
65   }
66
```

```
67   return total / numReadings;
68 }
69 String temperature(int sensorVal) {
70   float voltage = (sensorVal / 1024.0) * 5.0;
71   float temperature = (voltage - .5) * 100;
72   Serial.println(temperature);
73   return String(temperature);
74 }
```

在这段代码中，我们处理了两件事情：BLE 和温度传感器。我们先来看一下 BLE 代码。

在类的顶端，我们创建了一个带相应引脚的 BLE UART 实例。在 `setup` 方法中，我们设置了设备名称，并启动我们的 BLE 芯片。在 Arduino 上，当 Arduino 启动时，`setup` 方法将执行一次，`loop` 方法不断执行。所以我们所有的处理都是在 `loop` 方法中完成的。在这种方法中，我们得到 BLE 的当前状态，如果连接，我们发送温度数据。我们一次只能发送 20 个字节，一旦我们得到我们的温度数据（格式化为字符串），我们将其写入我们的对象。这将是我们通知的 RX 特性（characteristic），我们的 iOS 应用程序将从中读取。

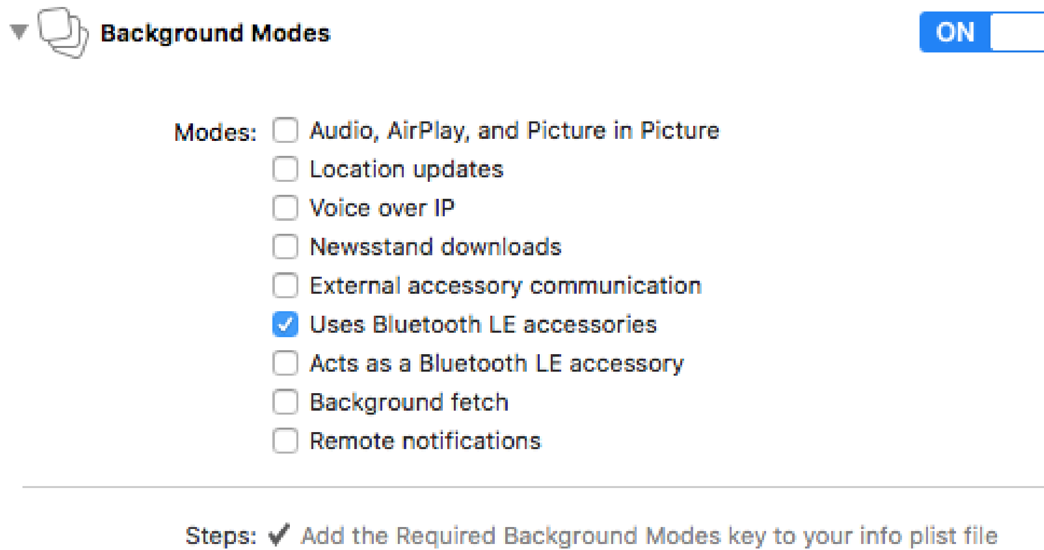
我们的温度传感器连接到模拟输入引脚 A0 ，因为我们将以非常高的频率对数据进行采样，因为需要平滑数据，以防止尖峰。为了平滑数据，我们将使用 'averageValue' 函数。此功能将进行最后 50 次测量，并给出平均值。我们将对数据进行多次采样，所以这将给我们一个平滑的样本。一旦我们得到平滑的数据，我们需要将原始传感器值转换为实际温度。我们将在 "temperature" 方法中进行。该方法将采用传感器输入值，将其转换为电压，然后转变成温度。这个字符串是我们写入我们的 BLE RX 特性（characteristic）的字符串。

在 Arduino 方面，这几乎是所有的东西，我们有一块通过 BLE 传输传感器数据的硬件。唯一要做的就是在 iOS 应用程序中读取数据。

步骤 3：iOS 项目

一旦你创建了一个项目到你的项目 `capabilities` 标签，启用 `Background Modes` 并选择 `Uses Bluetooth LE accessories` 才能工作，像这样：

玩家什么.https://www.wanjiashenme.



BLEManager

现在，我们准备开始编码了。我们只需要一个类，用它把所有东西挂起来。我们称之为 “BLEManager” 类，它大约是 200 行代码。这就是我们需要的代码。这根目录惊人的，是不□。显然，我们将使用 Core Bluetooth，所以我们需要导入到项目中。

以下是类的核心代码：

```
1  class BLEManager: NSObject, BLEManagable {
2
3      fileprivate var shouldStartScanning = false
4
5      private var centralManager: CBCentralManager?
6      private var isCentralManagerReady: Bool {
7          get {
8              guard let centralManager = centralManager else {
9                  return false
10             }
11             return centralManager.state != .poweredOff &&
                   centralManager.state != .unauthorized &&
                   centralManager.state != .unsupported
12         }
13     }
14
```

```
15     fileprivate var connectingPeripheral: CBPeripheral?
16 fileprivate var connectedPeripheral: CBPeripheral?
17     fileprivate var delegates: [Weak<AnyObject>] = []
18     fileprivate func bleDelegates() -> [BLEManagerDelegate] {
19         return delegates.flatMap { $0.object as? BLEManagerDelegate }
20     }
21 
22     override init() {
23         super.init()
24         centralManager = CBCentralManager(delegate: self, queue:
               DispatchQueue.global(qos: .background))
25         startScanning()
26     }
27 
28     func startScanning() {
29         guard let centralManager = centralManager, isCentralManagerReady ==
               true else {
30             return
31         }
32 
33         if centralManager.state != .poweredOn {
34             shouldStartScanning = true
35         } else {
36             shouldStartScanning = false
37             centralManager.scanForPeripherals(withServices:
                   [BLEConstants.TemperatureService], options:
                   [CBCentralManagerScanOptionAllowDuplicatesKey : true])
38         }
39     }
40 
41     func stopScanning() {
42         shouldStartScanning = false
43         centralManager?.stopScan()
44     }
45 
46     func addDelegate(_ delegate: BLEManagerDelegate) {
```

```swift
        delegates.append(Weak(object: delegate))
}
    func removeDelegate(_ delegate: BLEManagerDelegate) {
        if let index = delegates.index(where: { $0.object === delegate }) {
            delegates.remove(at: index)
        }
    }
}
```

这里的主要方法是 `startScanning`。我们在构造函数中调用这个方法。在我们进行一些健康检查（sanity checks）后，我们开始扫描具有特定 UUID 服务的外围设备。那将是我们的温度服务。

```swift
// MARK: CBCentralManagerDelegate
extension BLEManager: CBCentralManagerDelegate {

    func centralManagerDidUpdateState(_ central: CBCentralManager) {
        if central.state == .poweredOn {
            if self.shouldStartScanning {
                self.startScanning()
            }
        } else {
            self.connectingPeripheral = nil
            if let connectedPeripheral = self.connectedPeripheral {
                central.cancelPeripheralConnection(connectedPeripheral)
            }
            self.shouldStartScanning = true
        }
    }

    func centralManager(_ central: CBCentralManager, didDiscover
       peripheral: CBPeripheral, advertisementData: [String : Any], rssi
       RSSI: NSNumber) {
        self.connectingPeripheral = peripheral
        central.connect(peripheral, options: nil)
        self.stopScanning()
    }
```

```
23
24 func centralManager(_ central: CBCentralManager, didConnect peripheral:
       CBPeripheral) {        self.connectedPeripheral = peripheral
25         self.connectingPeripheral = nil
26
27         peripheral.discoverServices([BLEConstants.TemperatureService])
28         peripheral.delegate = self
29
30         self.informDelegatesDidConnect(manager: self)
31     }
32
33     func centralManager(_ central: CBCentralManager, didFailToConnect
           peripheral: CBPeripheral, error: Error?) {
34         self.connectingPeripheral = nil
35     }
36
37     func centralManager(_ central: CBCentralManager,
           didDisconnectPeripheral peripheral: CBPeripheral, error: Error?) {
38         self.connectedPeripheral = nil
39         self.startScanning()
40         self.informDelegatesDidDisconnect(manager: self)
41     }
42 }
```

在 `didDiscover` 回调中，我们连接到发现的外设（peripheral）。重要的是要注意，我们需要强烈的引用（reference），我们正在尝试连接的外设（connectionPeripheral 属性），因为它不被保留。在 `didConnect` 委托回调中，我们发现了我们的 TemperatureService 的可用服务，我们将在这里设置我们的外设（peripheral）委托（delegate），因此我们可以从我们的 RX 特性中读取数据。在 `centralManagerDidUpdateState` 中，如果我们在蓝牙关机时开始扫描，我们将开始扫描。此外，如果我们关闭蓝牙电源，则取消外设（peripheral）连接，并在设备重新启动时提升标志开始扫描。当您关闭设备上的蓝牙或 Arduino 时，此方法将被调用。

我们使用 central manager 委托来启动服务发现，我们将使用外设（peripheral）代理回调来发现我们的特性，并从中读取数据。以下是我们将实现的回调：

```
1 // MARK: CBPeripheralDelegate
2 extension BLEManager: CBPeripheralDelegate {
```

```
3
4 func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error:
       Error?) {        if let tempService = peripheral.services?.filter({ $0.uuid.uuidStrin
       == BLEConstants.TemperatureService.uuidString.uppercased() }).first {
5              peripheral.discoverCharacteristics([BLEConstants.RXCharacteristic],
                   for: tempService)
6          }
7      }
8
9      func peripheral(_ peripheral: CBPeripheral,
           didDiscoverCharacteristicsFor service: CBService, error: Error?) {
10         if let rxCharacteristic = service.characteristics?.filter({
               $0.uuid.uuidString.uppercased() ==
               BLEConstants.RXCharacteristic.uuidString.uppercased()}).first {
11             peripheral.setNotifyValue(true, for: rxCharacteristic)
12         }
13     }
14
15     func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor
           characteristic: CBCharacteristic, error: Error?) {
16         guard let temperatureData = characteristic.value else {
17             return
18         }
19
20         if let dataString = NSString.init(data: temperatureData, encoding:
               String.Encoding.utf8.rawValue) as? String {
21             self.informDelegatesDidReceiveData(manager: self, dataString:
                   dataString)
22         }
23     }
24 }
```

当外围（peripheral）委托被发现的服务调用时，我们将过滤阵列来查找我们的温度服务，然后我们将启动对 RX 特性（characteristic）的特征发现。当发现特征时，我们将 RX 特性（characteristic）的 notify 值设置为 true。这样一来，每次 Arduino 在 RX 特性中输出某些内容时，我们会收到通知。

现在每次特性（characteristic）更新时，`didUpdateValueFor` 将会被调用。我们将从 `.value` 属性读取数据，并将其转换成字符串。最后一件事是，告知委托温度字符串，就是所有的逻辑。

View Controller

我们将让视图控制器（view controller）保持简单，只有一个带有温度值的标签。以下整个视图控制器的代码：

```
1  class ViewController: UIViewController {
2
3      var bleManager: BLEManagable = BLEManager()
4
5      @IBOutlet weak var temperatureLabel: UILabel!
6
7      override func viewWillAppear(_ animated: Bool) {
8          super.viewWillAppear(animated)
9          bleManager.addDelegate(self)
10     }
11
12     override func viewDidDisappear(_ animated: Bool) {
13         super.viewDidDisappear(animated)
14         bleManager.removeDelegate(self)
15     }
16 }
17
18 // MARK: BLEManagerDelegate
19 extension ViewController: BLEManagerDelegate {
20
21     func bleManagerDidConnect(_ manager: BLEManagable) {
22         self.temperatureLabel.textColor = UIColor.black
23     }
24     func bleManagerDidDisconnect(_ manager: BLEManagable) {
25         self.temperatureLabel.textColor = UIColor.red
26     }
27     func bleManager(_ manager: BLEManagable, receivedDataString dataString:
           String) {
```

```
28         self.temperatureLabel.text = dataString + "□"
29 }}
```

在 `viewWillAppear` 中，我们将视图控制器添加为委托，在 `viewDidDisappear` 中，我们将移除视图控制器作为委托。在委托回调中，我们更改了温度标签的颜色。如果 Arduino 未连接，则为红色，连接时为黑色。当然，在 `receivedDataString` 中我们得到了我们的温度值，所以我们只需用新值来更新标签。如下图所示：

nenme.com
玩

步骤 4：结论

在这篇文章中，我们看到了如何通过您的应用程序连接 Arduino。我们的 Arduino 板上有一个温度传感器，但我们可以轻松地连接任何其他传感器。随着物联网现在成为一个新的技术，你可以连接一堆传感器，并自动化你的家。也许在未来的一个帖子中，我们将看到如何在 Arduino 和 iOS 之间建立双向通信，这样您可以使用手机来控制物理设备。这是一个有趣的小项目，你每天都可以用来测量室内温度（我会把我的留在客厅里）。

您可以在我的 GitHub 帐户（https://github.com/dagostini/DABLETemperatureSensor）中找到所有代码（iOS 项目和 Arduino 程序）。我希望你会发现这个项目有用和有趣。

原文链接：http://agostini.tech/2017/02/20/creating-a-temperature-sensor-for-ios-using-ble-and-arduino/

原文链接：https://www.wandianshenme.com/play/arduino-ble-ios-read-temperature-sensor

玩点什么：https://www.wandianshenme.com